Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Spielbasierte Verifikation von Links
im Semantic Web

Bachelorarbeit

Leipzig, September 2011

vorgelegt von
Nguyen, Tri Quan

Betreuer:	**Dr. Jens Lehmann**
Verantwortlicher Hochschullehrer:	**Prof. Dr. Ing. habil. Klaus-Peter Fähnrich**
Institut für Informatik	

Abstract

In dieser Arbeit wurde ein Spiel implementiert, mit dessen Hilfe die Qualität von Verlinkungen im Semantic Web durch menschliche Intelligenz überprüft werden können. Das Spiel trägt den Namen Veri-Links und gehört dem Genre der Games with a Purpose (GWAP) an. Spiele dieser Art machen sich den Wunsch der Menschen nach Unterhaltung zu Nutze und bringen sie dazu, während des Spielens nützliche Metadaten - als Nebenprodukt - zu generieren. Veri-Links belohnt seine Spieler für eine abgeschlossene Verifikation mit Spielressourcen, welche für das erfolgreiche Beenden eines nebenläufigen 2D Plattform Spiels benötigt werden. Auf diese Weise soll der Spieler für eine zeitaufwendige Aufgabe motiviert werden.

Inhaltsverzeichnis

Seite

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Jedes Jahr verbringen Menschen auf der ganzen Welt Millionen Stunden mit Computerspielen. Was wäre wenn eine Person die Möglichkeit bekommt während des Computerspielens nützliche Arbeit zu verrichten, welche wohlmöglich der ganzen Welt zu Gute kommt. *Games with a Purpose* verfolgen diesen Ansatz, indem sie versuchen bestimmte Aufgabenstellungen, welche nicht vollständig von Rechnern gelöst werden können, auf Menschen auszulagern.[1] Dabei wird dem Spieler das zu lösende Problem auf eine unterhaltsame Weise präsentiert. Es soll unter anderem das Gefühl vermittelt werden, dass der Spielspaß im Vordergrund steht und die Lösung des Problems lediglich als Nebenprodukt beim Spielen generiert wird.[2] Diese Spiele haben vielversprechende Ergebnisse[1] geliefert für die erfolgreiche Bearbeitung einer Vielzahl von Problemen, welche bis heute noch nicht selbstständig von Computern gelöst werden können.

Eine Aufgabe die stark von menschlichem Input abhängig ist und dadurch noch nicht automatisiert werden kann, ist die Verifizierung von Links im Semantic Web (kurz: Link Verifikation). In dieser Arbeit wird versucht ein Spiel nach dem GWAP Paradigma zu entwickeln, welches die beschriebene Aufgabe erfüllt. Das Spiel, mit dem Namen Veri-Links, soll dem Spieler ermöglichen auf spielerische Art und Weise über die Korrektheit von Semantic Web Links zu urteilen. Dabei stehen ihm zwei Komponenten zur Verfügung: Die GameComponent und die VerifyComponent.
In der GameComponent befindet sich ein 2D Spiel, welches Elemente aus dem Tower Defense Genre besitzt. Die Aufgabe des Spielers ist es, seine Spielfiguren so in der Spielwelt zu platzieren, damit sie mehrere Anstürme von unterschiedlichen Gegnern daran hindern die Spielwelt zu durchqueren. Durch die erfolgreiche Abwehr wird der Punktestand des Spielers erhöht und er bekommt die Möglichkeit sich in die Highscore Liste des Spiels einzutragen. Um die für die Verteidigung benötigten Einheiten kaufen zu können, benötigt der Spieler Geld, welches er sich in der VerifyComponent durch eine erfolgreich getätigte Link Verifikation verdienen kann. Mit diesem Spielprinzip entsteht eine Symbiose zwischen beiden Komponenten, welche eine arbeits- und zeitintensive Aufgabe mit Spielspaß verknüpft.
Veri-Links basiert auf einer Client-Server Struktur und besitzt zusätzlich eine MySQL Datenbank zur persistenten Speicherung von Daten, wie die Highscores der Spieler, alle zu verifizierenden Links und deren Verifikation. Nur der Server hat direkten Zugriff auf diese Datenbank. Seine Aufgabe ist es dem Client die Links zu schicken, damit die Spieler diese verifizieren können. Vom Spieler getätigte Link Verifikationen werden

[1] Siehe Kapitel 5 für konkrete Beispiele

vom Client an den Server zur Auswertung geschickt. Danach aktualisiert der Server die Datenbank mit den neuen Informationen.

Die Implementierung dieses Spiels erfolgte komplett in Java durch Zuhilfenahme des Google Web Toolkits und der PlayN Game library. Zusätzlich wurden für Datenbank Operationen die JDBC Bibliothek und für die Arbeit mit Semantic Web Mechanismen das Jena Semantic Web Framework verwendet.

In Kapitel 2 wird ein allgemeiner Überblick über die Grundlagen dieser Arbeit gegeben. Es erfolgt eine Beschreibung der benötigten Bibliotheken, Technologien und Standards. Darüber hinaus wird das Projekt in dessen sich diese Arbeit befindet kurz vorgestellt.
Im 3. Kapitel wird das Entwurfskonzept vorgestellt, dabei werden zum einen die Produktanforderungen beleuchtet und zum anderen die Komponenten und Prozesse des Spiels näher beschrieben. Das Kapitel 4 erläutert die erledigte Implementierungsarbeit. Dabei werden schwer zu verstehende Passagen zum besseren Verständnis mit Grafiken und Ausschnitten aus dem Quellcode erklärt. Eine Übersicht über verwandte Arbeiten ist in Kapitel 5 aufzufinden. Im 6. Kapitel wird ein Fazit für die Bachelorarbeit gezogen, dabei werden die entwickelnden Ergebnisse noch einmal reflektiert und bewertet. Das Kapitel 7 gibt einen Überblick über die in Zukunft geplanten Arbeiten zur Verbesserung dieses Spiels. Im letzten Teil der Arbeit befindet sich der Anhang, samt ausführlicher Anleitung zur Installation und Nutzung von Veri-Links.

2. Grundlagen

2.1. Semantic Web

Das *Semantic Web* ist eine Erweiterung des World Wide Web und modelliert ein Netz aus Daten. Während das World Wide Web elektronische Dokumente miteinander vernetzt, werden beim Semantic Web Informationen auf der Ebene ihrer Bedeutung miteinander verknüpft.[3] Ziel des Semantic Web ist es, diese Bedeutung von Informationen für Computer verwertbar zu machen, damit sie diese automatisch interpretieren und weiterverarbeiten können. Zudem erlaubt das Semantic Web eine maschinelle Herleitung und Neugewinnung von Wissen aus bestehenden Daten. [4]
Das sogenannte *World Wide Web Consortium (W3C)* hat sich der Aufgabe verschrieben einheitliche und offene Standards für die Beschreibung von Informationen im Semantic Web zu vereinbaren. Speziell für das Semantic Web wurden sogenannte Ontologiesprachen (z.B. RDF(S), OWL) entwickelt. Unter dem Begriff *Ontologie* ist in dem Zusammenhang ein in RDF(S) oder OWL erstelltes Dokument zu verstehen, welches Wissen einer Anwendungsdomäne modelliert. Ein äquivalenter Begriff zur Ontologie wäre die *Wissensbasis*.[5]
Die Abbildung 2.1 ist eine Darstellung des *Semantic Web Layer Cakes*, welches die Hierarchie der Semantic Web Komponenten darstellt. Dabei werden die Fähigkeiten und die Ressourcen der unteren Schichten von den oberen ausgeschöpft.[6] Die Darstellung gibt einen Überblick über die für das Funktionieren des Semantic Web benötigten standardisierten Technologien.

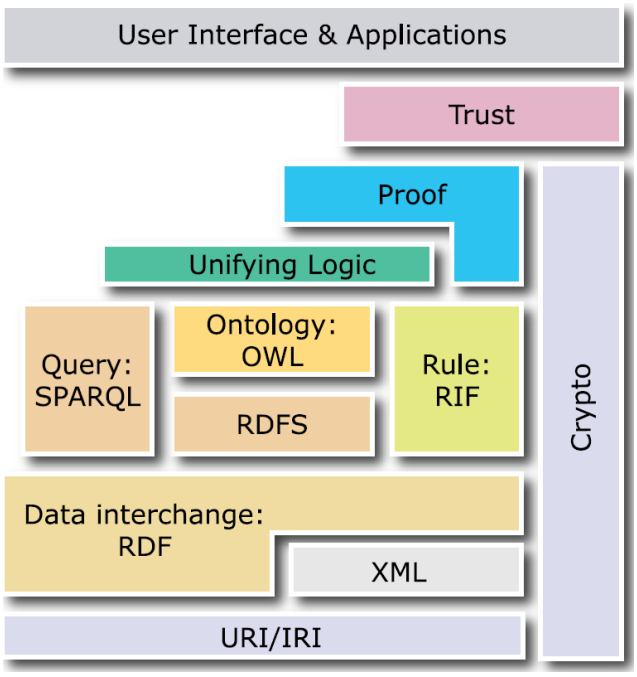

Abb. 2.1. Semantic Web Layer Cake [7]

2.1.1. Resource Description Framework (RDF)

RDF ist eine formale Sprache zur Beschreibung von Ressourcen im Web. Unter einer Ressource versteht man all das, was durch einen weltweit eindeutigen Bezeichner im *URI*-Format benannt wird.[8] Diese Sprache ist darauf ausgelegt Wissen zu repräsentieren und weniger Daten, sprich alle RDF-Ausdrücke haben eine gewisse Bedeutung. Durch die formale Repräsentation in RDF sind Informationen nicht nur von Menschen sondern auch von Maschinen automatisch lesbar und bearbeitbar.[9] Ebenso ist es möglich neue Informationen aus vorhandenem Wissen maschinell herzuleiten, mit Hilfe von Spezifikationen des modellierten Bereiches (z.B. mit Hilfe von RDF-Schema (RDFS) oder der Web Ontology Language (OWL).[10]
Datensätze aus verschiedenen Wissensbasen des Semantic Webs können durch RDF Links miteinander verknüpft werden (*Interlinking*).[2] Es gibt 2 komplementäre Wege RDF-Informationen zu betrachten. Der erste Weg ist die Betrachtung der Informationen mit Hilfe von RDF-Ausdrücken/Tripeln. Jeder Ausdruck repräsentiert dabei einen Fakt. Der Zweite ist durch einen Graphen. Ein RDF Graph drückt exakt das gleiche aus wie ein RDF-Ausdruck/Tripel, nur ist diese Form übersichtlicher und erleichtert das Erkennen von Struktur in den Daten.
Das RDF-Modell setzt sich zusammen aus Tripeln der Form (Subjekt, Prädikat, Objekt). Dieses Tripel sagt aus, dass eine gerichtete Beziehung zwischen dem Subjekt und dem Objekt besteht. Das Prädikat stellt dabei eine binare Relation dar. Subjekt und Prädikat sind immer Ressourcen, das Objekt kann entweder eine Ressource oder ein Literal sein. Literale sind Zeichenketten welche Datentypwerte repräsentieren. [7] RDF Ausdrücke bilden selbst wieder Ressourcen, auf die mit weiteren Tripeln verwiesen werden kann. Das RDF-Modell ist unabhängig von einer speziellen Darstellungsform - die verbreiteteste Notation ist die in RDF/XML. Weitere Notationen sind N3, Turtle und N-Triples. Diese unterscheiden sich lediglich in der Darstellungsform der Tripel.[11]

2.1.2. SPARQL

Die *SPARQL Query Language for RDF* ist eine graphbasierte Anfragesprache für RDF. SPARQL ermöglicht es, die durch RDF implizit gespeicherten Metadaten gezielt aus Webseiten, Wissensbasen und anderen Ressourcen herauszusuchen und darzustellen. Es ähnelt der SQL Sprache, die für Anfragen an Relationale Datenbanken genutzt wird.[9]
Der Aufbau einer typischen SPARQL-Anfrage gliedert sich in drei Bestandteile (Beispiel):

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?book ?title
WHERE { ?book dc:title ?title }
```

[2] Siehe Kapitel 2.1.3. für weitere Informationen.

Mit Hilfe von PREFIX können Namespaces abgekürzt werden.
Der SELECT-Teil definiert Variablen, die nach der Anfrage angezeigt werden sollen. Variablen kennzeichnen sich stets durch ein vorangestelltes „?“ oder „$“.
In der WHERE-Klausel wird die eigentliche Anfrage durch Ausnutzung von Graph-Patterns definiert. Dabei werden für jeden Teil der Anfrage eine Ressource, ein Prädikat und eine weitere Ressource oder ein Literal benötigt. Darüber hinaus lassen sich in die WHERE-Klausel viele Anfrageoptionen integrieren (zum Beispiel FILTER, UNION). Im Anschluss an die WHERE-Klausel lassen sich noch verschiedene Optionen zur Ausgabemanipulation setzen (zum Beispiel ORDER BY, LIMIT).[8][12]

2.1.3. Linked Data

Linked Data ist eine Methode um Daten im World Wide Web zu veröffentlichen und diese mit verschiedenen Datenquellen zu verknüpfen (Interlinking).[13] Linked Data ist Teil des Semantic Web, zur Kodierung und Verlinkung der Daten wird das Resource Description Framework (RDF) und darauf aufbauende Standards wie SPARQL und die Web Ontology (OWL) verwendet. Auf Linked Data kann unter anderem mit Semantic Web Browsern zugegriffen werden, genauso wie Web Dokumente durch HTML Browser erreichbar sind. Bei Semantic Web Browsern erfolgt die Navigation zwischen verschiedenen Data Quellen, indem RDF Links verfolgt werden. Mit Hilfe von Linked Data soll ein Netz aus Daten erschaffen werden, in dem die Strukturierung der Daten, ein automatisches Auslesen durch Computer ermöglicht. [14]
Der Begriff Linked Data wurde von Sir Tim Berners-Lee geprägt, dieser definierte folgende Richtlinien für die Arbeit mit Linked Data[15]:

1. Dinge werden durch eine URI benannt und identifiziert.
2. Durch die Verwendung von http URIs können diese Dinge auch von Menschen nachgeschlagen werden.
3. Unter einem URI sollen sinnvolle Informationen bereitgestellt werden – die auch ein Mensch lesen kann.
4. Durch die Einbindung anderer Links können weitere Dinge entdeckt werden.

Linking Open Data Community Project

Das Ziel der W3C Gruppe *Linking Open Data Community Project* ist es das World Wide Web mit Daten zu erweitern, indem verschiedene freie Datensätze aus unterschiedlichen Datenquellen im Netz mittels RDF veröffentlicht und diese zusätzlich mit RDF Links verknüpft werden. Im Oktober 2007 umfasste die Anzahl der Datensätze über 2 Milliarden RDF Tripel, welche durch über 2 Millionen RDF Links miteinander verbunden wurden. Bis September 2010 stieg die Zahl schon auf 25 Milliarden RDF Tripel, welche durch 395 Millionen RDF Links miteinander verknüpft sind.[16][17]

Die Abbildung 2.2. zeigt alle Datensätze, die bis September 2011 im Linked Data Format veröffentlicht wurden.

Ein von der Europäischen Kommission unterstütztes Forschungs Projekt welches sich auf Linked Data fokusiert ist das *Linked Open Data Around the Clock (LATC) - EU project*. Als Teil des Seventh Framework Program (FP7) der Europäischen Kommission, unterstützt das Linked Open Dara Around the Clock Projekt, Menschen und Organisationen bei der Veröffentlichung und Konsumierung von Linked Open Data. [18]

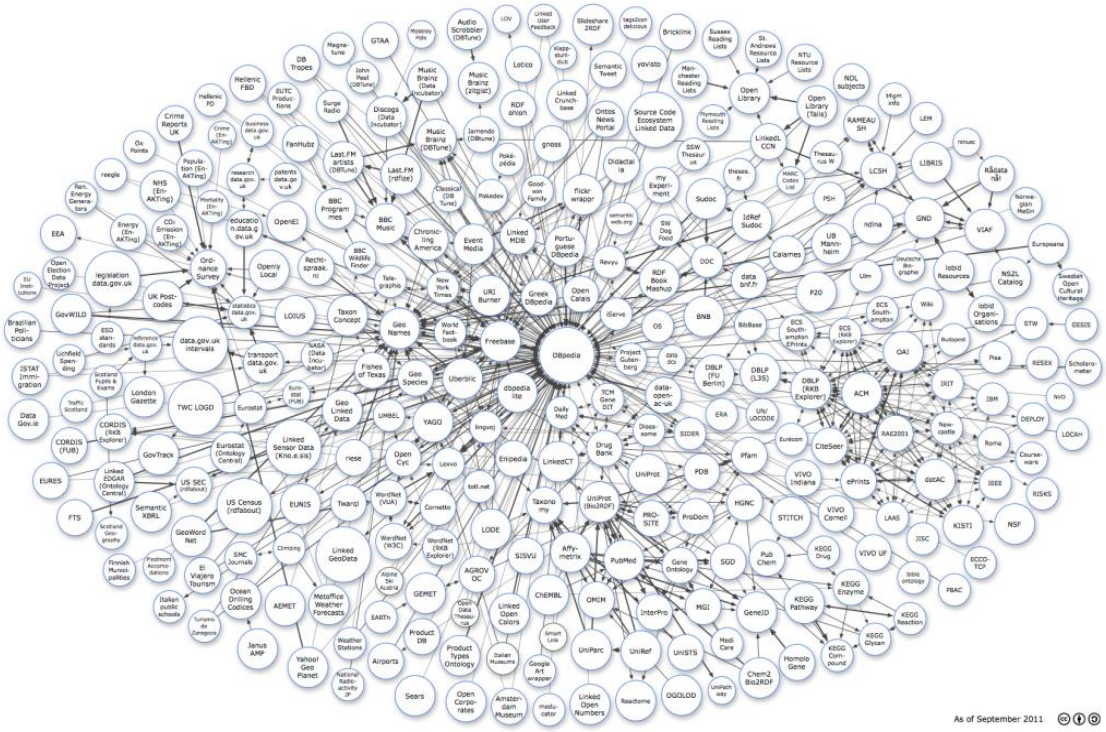

Abb. 2.2. Linking Open Data cloud diagram, by R. Cyganiak and A. Jentzsch[17]

2.2. Saim

Veri-Links findet unter dem *AKSW (Agile Knwoledge Engineering and Semantic Web)* Projekt *SAIM* des Informatik Instituts der Uni Leipzig statt. SAIM (Semi Automatic Instance Matcher) erlaubt das Interlinking von Wissensbasen im Semantic Web. Der Fokus von SAIM liegt auf das instance matching von sehr großen Wissensbasen welche als SPARQL Endpoints zur Verfügung stehen. Es benutzt Machine Learning Techniken und ist mit SILK (siehe Kapitel 2.3.5) und LIMES kompatibel.[19]

2.3. Bibliotheken

2.3.1. Google Web Toolkit (GWT)

Das *Google Web Toolkit* ist ein open-source Development Tollkit von Google. Java Entwicklern soll es die Möglichkeit bieten komplexe und performante Webanwendungen in Ihrer gewohnten Programmiersprache und Entwicklung-sumgebung (z.B. Eclipse) zu entwickeln. Die Entwickler schreiben ihre Frontends in Java unter Verwendung des Google Web Toolkits und kompilieren die Java-Klassen anschließend in JavaScript und HTML um, so dass die Anwendungen in jedem Browser richtig dargestellt werden und funktionieren.
Das GWT beinhaltet neben dem Java zu JavaScript Compiler, einen XML-Parser, ein Widget-Paket zur Gestaltung von graphischen Benutzeroberflächen, eine Schnittstelle für Remote Procedure Calls[3], JUnit Integration, Internationalisierungs-Unterstützung uvm. Wer selbstgeschriebenen JavaScript-Code benötigt, kann diesen mit dem "JavaScript Native Interface" direkt in die Java-Anwendung integrieren. [20]

Client-Server Kommunikation

Die Kommunikation mit einem Server kann auf unterschiedliche Art und Weise realisiert werden. Welches Datenformat am Ende verwendet werden soll, ist abhängig vom Typ und Aufbau des Servers:[20]

JSON oder XML Daten mit HTTP abrufen: Ist ein Server nicht in der Lage Java Servlets zu hosten oder benutzt der Server Daten im JSON oder XML Format, besteht die Möglichkeit diese Daten mit Hilfe von HTTP Requests abzurufen. GWT stellt zum einen generische HTTP Klassen zum konstruieren der Requests zur Verfügung und zum anderen JSON und XML Client Klassen, welche zur Bearbeitung der Antwort benutzt werden können

Remote Procedure Calls (GWT RPC) tätigen: Ist es möglich Java auf dem Server im backend laufen zu lassen und sollen umfangreiche Anfragen getätigt werden, so empfiehlt sich die Realisierung der Server Kommunikation mit den GWT Remote Procedure Calls.

[3] Eine Technik zur Realisierung von Interprozesskommunikation. Sie ermöglicht den Aufruf von Funktionen in anderen Adressräumen.

GWT RPC

Das *GWT RPC* ist ein auf Java Servlets basierender Mechanismus, welcher Zugriff auf serverseitige Ressourcen zulässt und Java Objekte zwischen Client und Server über Standard HTTP übertragen kann. Dabei übernimmt es die Serialisierung und Deralisierung von Java Objekten, der Benutzer muss sich also nicht um die Umwandlung von Objekten kümmern. Weiterhin werden vom AsyncCallback Interface zwei Methoden bereitgestellt, mit denen es möglich ist gezielt auf den Erfolgs- und Fehlerfall einer Anfrage reagieren zu können.[21]

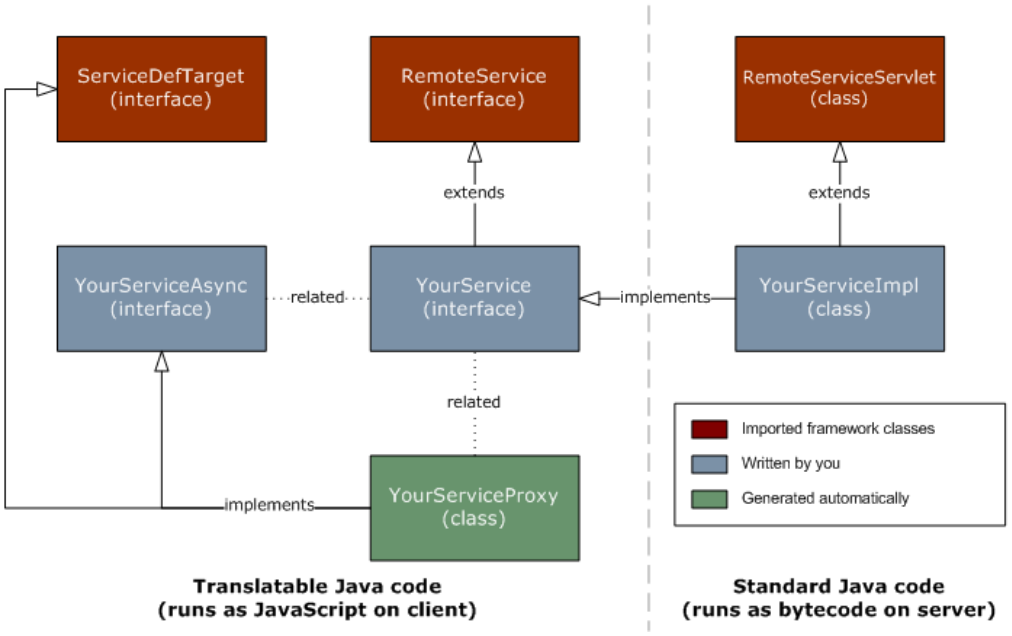

Abb. 2.3. RPC Diagramm

Die Abbildung 2.3. gibt ein Überblick über die benötigten Klassen für einen RPC. Die Implementierung eines service[4] erfolgt in der service implementation, welcher auf der servlet-Architektur beruht. Eine service implementation muss die Klasse RemoteServiceServlet erben und das service interface implementieren. Dabei setzt sich ihr Name aus dem Namen des service interfaces und dem Suffix „Impl“ zusammen. RemoteServiceServlet kümmert sich automatisch um die De- und Serialisierung der ihm übergebenen Daten und ruft die benötigte Methode in der service implementation auf. Zu jedem service interface muss ein asynchronous service interface implementiert werden, welcher nach dem service interface benannt wird und als Suffix „Async“ besitzen muss.

[4] Eine Client-Server Kommunikationsmethode.

2.3.2. PlayN

PlayN ist eine cross-platform game abstraction library, konzipiert für das Entwickeln von Spielen, welche in verschiedene Systeme kompiliert werden kann wie zum Beispiel in Desktop Java, HTML5 Browsers, Android und Flash. PlayN benutzt das Google Web Toolkit und wurde komplett in Java geschrieben. Diese open source Bibliothek steckt erst am Anfang ihrer Entwicklungsphase. Die aktuelle Version ist lediglich ein alpha Release und die API wird in regelmäßigen Abständen verändert.[22]

Die PlayN Bibliothek abstrahiert die Plattform spezifischen Details der Spiele Programmierung, indem es allgemeine APIs für das Implementieren von gebräuchlichen Spielverhalten anbietet.
Bei dieser Bibliothek erfolgt eine Isolierung des Input und Output Systems, dazu zählen Audio, 2D Graphiken, Keyboard und Pointer Inputs. Ebenfalls wird die Steuerung des Game Loop Timings abstrahiert, PlayN sucht die effektivste Timing Implementation für das System aus, auf dem es zum Laufen gebracht wird. Darüberhinaus wird das Management von Assets (Bildern, Sounds) von dieser Bibliothek unterstützt und vereinfacht.[23]

Grafik Architektur

In der PlayN Grafik Architektur existieren 3 Haupt Konzepte: Layer, Surface und Canvas.[24] Alle Grafiken werden in diesen Objekten gezeichnet.

Layers: Das Grafik System besitzt eine Hierarchie von Layern, welche vom System automatisch gerendert werden. Jedes Layer besitzt eine affine Transformation und mehrere andere Eigenschaften, welche direkt manipuliert werden können (Änderungen werden automatisch im nächsten gerenderten Frame übernommen). Neben dem ImageLayer, existieren noch GroupLayer um andere Layers in eine Gruppe zusammenzuführen, CanvasLayer und SurfaceLayer.

Surfaces: Oft ist es wünschenswert Teile einer Szene direkt zu rendern, mittels einer imperativen API. Dies ist mit dem SurfaceLayer möglich. Dabei hat es die gleichen Eigenschaften (transform, alpha, usw.) wie andere Layer und kann mit ihnen koexistieren. Surface wurde explizit entworfen um OpenGL calls Mapping zu verinfachen, deswegen ist dieses Objekt oftmals performanter als die anderen.

Images und Canvas: Im Allgemeinen dient ein Image als eine Quelle für bitmap Daten. Das Canvas Interface erlaubt ein direktes Zeichnen in ein Image. Canvas ist eine

2D rendering API, mit Möglichkeiten Text, Pfad, Kurven, Gradients uvm. zu zeichnen. Canvas sind oft langsamer als Surfaces.

GameLoop

Die meisten Spiele besitzen einen Game Loop, darin werden in einer Endlosschleife die beiden Methoden update und paint abwechselnd aufgerufen. Um in PlayN die Kontrolle über diesen Game Loop zu erlangen muss eine Klasse das Game Interface implementieren und die Methode PlayN.run(game) aufrufen. Wie schnell dieser Game Loop abläuft kann mit der update und paint Frequenz gesteuert werden. Spiele mit einer komplexen update Logik (vor allem diejenigen die Physic Engines benutzen) brauchen genügend Kontrolle über die beiden Frequenzen, da die numerischen Integrationstechniken von Physic Engines außerhalb bestimmter Frequenzen oft instabil werden können. Eine gebräuchliche Lösung für dieses Problem ist oftmals das Definieren einer fixen Update Frequenz, während paint so oft wie möglich aufgerufen wird. Jedoch bringt dieser Ansatz eine weitere Hürde mit sich. Die paint Methode wird normalerweise zu einem zufälligen Zeitpunkt zwischen updates aufgerufen, nun stellt sich die Frage welche Objekte in dieser Zeit gezeichnet werden sollen. Abhilfe für dieses Problem verschafft der alpha Parameter der paint Methode. Dieser Parameter nimmt einen Wert zwischen null und eins an, und gibt an wie weit wir zwischen den beiden letzten Updates entfernt sind. Dieser Wert kann nun benutzt werden um den Zustand der Spiele Welt zu interpolieren[5], damit die Animationen so flüssig wie möglich dargestellt werden. PlayN bietet verschiedene Möglichkeiten zum Entwickeln und Debuggen an. Es kann die Java Runtime Environment, der GWT Production Mode oder der GWT Development Mode benutzt werden. [25]

2.3.3. JBox2D

PlayN simuliert die physikalischen Prozesse mit der Physics Engine *JBox2D*, welche eine open source Java-Schnittstelle von dem in C++ geschriebenen Physik Simulator Box2D ist. Programmierer können JBox2D in ihren Spielen verwenden um Objekte auf eine realistische Weise zu bewegen und um die Spielwelt interaktiver zu gestalten. JBox2D besitzt Algorithmen zur Kollisionserkennung und –auswertung, welche eine effektive Simulation von schnellen Körpern ermöglicht ohne dabei Instabilitäten hervorzurufen oder Kollisionen zu versäumen. Diese Physics Engine ist simpel gehalten und besitzt daher keinen graphischen Output, es aktualisiert leidglich die internen Modelle.

[5] Interpolationen sind Algorithmen, die beim Skalieren einer Pixelgraphik die Pixel so auf die angezeigten Bildschirmpunkte umrechnen, dass ein möglichst stimmiges Gesamtbild entsteht.

Für eine möglichst effektive und realitätsnahe Simulation der Physik arbeitet JBox2D mit verschiedenen Fundamentalen Objekten. Einige Beispiele wären:

Shapes: Ein 2 dimensionales geometrisches Objekt, z.B. ein Kreis oder ein Polygon.

Bodies: Instanzen dieser Klasse haben eine Position und eine Geschwindigkeit. Auf ihnen können Kräfte, Impulse und Drehmomente wirken. Sie können statisch, kinematisch oder dynamisch sein und werden durch Joints zusammengefügt.

Fixtures: Bindet ein Shape an einen Body und fügt Material Eigenschaften (z.B. Dichte, Reibung) an das resultierende Objekt an.

World: In JBox2D modelliert die Klasse World die Welt in der sich die physikalischen Prozesse abspielen sollen. Es ist das Hauptobjekt der gesamten Simulation und in ihm werden alle anderen JBox2D Objekte gespeichert und verwaltet. Diese Klasse besitzt Methoden um Bodies zu erschaffen und zu zerstören.

Am Anfang jeder Physik Simulation muss ein World Objekt für JBox2D erstellt werden. Im nächsten Schritt sind die physischen Körper der Objekte der Spielewelt, sprich Bodies und Shapes, zu konstruieren, welche dann durch eine Fixture miteinander verbunden werden können. Die Simulation kann gestartet werden, wenn alle benötigten Bodies in das World Objekt eingefügt wurden. Durch den periodischen Aufruf der sogenannten step Funktion auf das World Objekt werden in jedem time step die physikalischen Prozesse (wie z.B. Translation, Kollision, neue Positionen der Objekte) der Welt berechnet. Dabei ist unter einem time step die diskrete Zeitdauer zu verstehen in der die Simulation durchzuführen ist. [26]

2.3.4. Jena Semantic Web Framework

Jena ist ein open source Java Framework zur Erstellung von Semantic Web Applikationen.[27] Es bietet eine Vielzahl von Klassen und Interfaces an, welche zur Bearbeitung von RDF Repositories verwendet werden können. Zusätzlich besitzt Jena eine API mit dessen Hilfe das Laden und Speichern von Daten in RDF Graphen ermöglicht wird. Die Graphen repräsentieren ein abstraktes Model, welches Daten aus einer Datei oder URLs beinhalten kann. Anfragen an das Model oder an SPARQL Endpoints können mit SPARQL gestellt werden.
Jena unterstützt die Serialisierung von RDF Graphen nach:

- Relationalen Datenbanken
- RDF/XML
- Terse RDF Triple Language (Turtle)
- Notation 3

2.3.5. Silk Framework

Das *Silk Framework* ist ein Tool konzipiert für das Entdecken und Generieren von Beziehungen in Form von RDF Links zwischen Data Items[6] innerhalb einer Linked Data Source. Mit der Verwendung der deklarativen Silk - Links Specification Language (Silk -LSL) können Entwickler spezifizieren welche Typen von RDF Links, zwischen Data Sources entdeckt werden sollen und welche Konditionen für Data Items erfüllt werden müssen, damit sie miteinander verlinkt (interlinked) werden. Diese Link Konditionen können aus verschiedenen Ähnlichkeits Metriken oder aus Informationen eines zu einem Data Item zugehörigen Graphen bestehen. Bei der Link Generierung wird für jedes Data Item Paar die Link Kondition evaluiert, dabei berechnet Silk einen Confidence Wert zwischen 0.0 und 1.0, welche den Grad der Ähnlichkeit zwischen den beiden Data Items repräsentiert. Die Ergebnisse der Link Generierung werden in einer Link Datei in XML/Alignment oder N-TRIPLE Format gespeichert. Mit Hilfe des SARQL Protokolls greift Silk auf die Data Sources zu, welche interlinked werden sollen. Dadurch ist es sowohl mit lokalen als auch mit fernen SPARQL Endpoints kompatibel.

Das Framework wird in drei verschiedenen Varianten angeboten, welche unterschiedliche Anwendungsfälle behandeln: Silk Single Machine, Silk MapReduce, Silk Server. Alle Varianten basieren auf dem Silk Link Discovery Engine.
Für diese Arbeit ist leidglich die Silk Single Maschine von Bedeutung. Diese Variante wird benutzt um RDF links auf einer einzelnen Maschine zu generieren. Die Datensätze, welche interlinked werden sollen können sich entweder auf der gleichen Maschine oder auf einer entfernten befinden und der Zugriff erfolgt durch das SPARQL Protokoll. Darüberhinaus ermöglicht die Silk Single Machine Multithreading und Caching.[28][29]

[6] Daten Element aus der Linked Data source

3. Entwurf

3.1. Produktanforderungen

Das Ziel von Veri-Links ist es, einen Spieler dazu zu bringen Links im Semantic Web auf Korrektheit zu überprüfen. Doch die Motivation eines Spielers für solch eine Verifikation, ist auf langer Sicht gesehen, stark begrenzt. Aus diesem Grund sollte das Spiel um eine 2D Spiel Komponente erweitert werde, welche den Spielern die Möglichkeit bietet durch getätigte Verifikationen, im 2D Spiel voranzukommen. Dadurch soll der Spielspaß und die Motivation für das Spielen von Veri-Links erhöht werden. Die Abbildung 3.1. stellt das Use-Case Diagramm von Veri-Links dar.

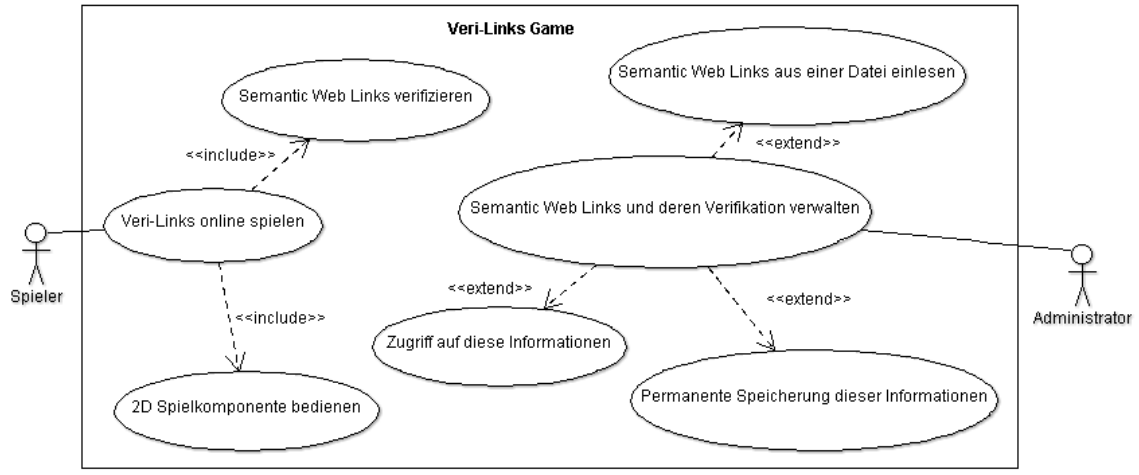

Abb. 3.1. Veri-Links Use-Case Diagramm

Für die Realisierung der Verifikation von Semantic Web Links werden folgende Funktionalitäten benötigt:

- Veri-Links soll das Einlesen und Speichern von Semantic Web Links ermöglichen.
- Es soll eine GUI, für die Darstellung und Verifizierung der Semantic Web Links, und die dazugehörige Logik implementiert werden.
- Realisierung einer Möglichkeit zur permanenten Speicherung, der vom Spieler getätigten Verifikationen, für eine spätere Evaluierung.

Das 2D Spiel muss folgende Kriterien erfüllen:

- Veri-Links soll Online gespielt werden.
- Das Spielprinzip und die Steuerung müssen simpel gehalten werden, damit der Spieler sich auch „nebenbei“ auf die Link Verifikation konzentrieren kann.
- Die getätigten Verifikationen eines Spielers müssen ins Spielgeschehen eingebunden werden.

3.2. Produktbeschreibung

3.2.1. Komponenten

Veri-Links basiert auf einer Client-Server Struktur und verwaltet eine MySQL Datenbank zur permanenten Speicherung aller relevanten Daten. Die folgende Abbildung 3.2. gibt einen Überblick über die wichtigsten Komponenten des Spiels und wie sie miteinander interagieren. Im Kapitel 3.2.2. wird der Datenaustausch zwischen den Komponenten genauer beschrieben.

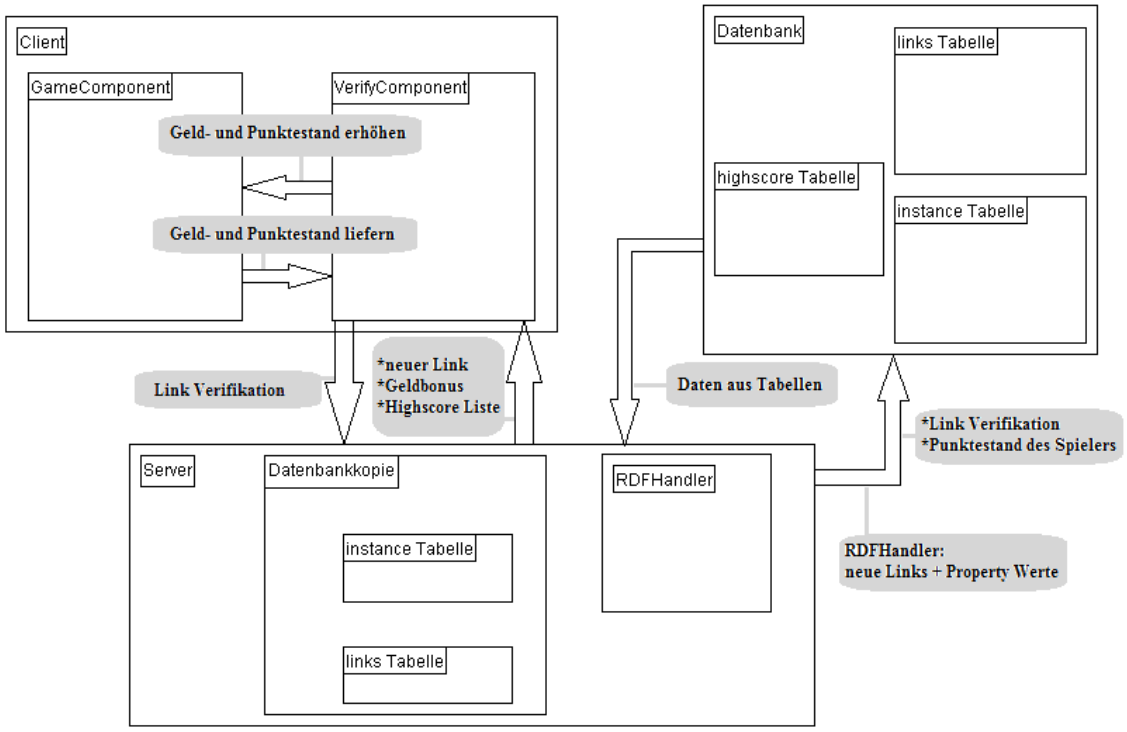

Abb. 3.2. Übersicht Veri-Links Komponenten

Datenbank

Die Datenbank besteht aus der *links*, *instances* und *highscore* Tabelle.
Wie der Name schon erahnen lässt, modelliert die *links* Tabelle einen Semantic Web Link. Solch ein Link besteht aus einem RDF Tripel, wobei das Subjekt und das Objekt jeweils Instanzen repräsentieren und das Prädikat ihre Beziehung darstellt. Jedem Link wird automatisch eine fortlaufende *ID* zugeordnet, das Attribut *ID* ist der Primärschlüssel der Tabelle. Darüberhinaus speichert die *links* Tabelle die URIs, die zusammen den Link bilden (*Subject*, *Predicate*, *Object*) und die zum Link dazugehörigen Verifikationsinformationen. Die Verifikationsinformationen bestehen

aus dem *Confidence*[7] Wert, der Anzahl der positiven und negativen Auswertungen (*Positive*, *Negative*), und dem *Counter*. Ein Link bekommt den *Confidence* Wert -1 zugeteilt, wenn seine Property Werte nicht über SPARQL Anfragen abgerufen werden können. Dieser Link ist somit nicht für eine Verifikation geeignet. Wie oft ein Link von einem Spieler verifiziert wurde, ist durch den *Counter* Wert gegeben. Die Verifikationsinformationen geben zum einen Aufschluss über die Korrektheit eines Links und zum anderen werden sie für die Punkte- und Geldbonusverteilung im Spiel genutzt (siehe Kapitel 3.2.2., Abschnitt Spielablauf). Mit Hilfe von *linkedOntologies* kann herausgefunden werden aus welcher Link Datei bzw. aus welchen verlinkten Ontologien die Instanzen des Links stammen. Die Abbildung 3.3. zeigt einen Ausschnitt aus der *links* Tabelle.

ID	Subject	Predicate	Object	linkedOntologies	Confidence	Counter	Positive	Negative
434	http://dbpedia.org/resource/Annang_language	http://dbpedia.org/ontology/spokenIn	http://www4.wiwiss.fu-berlin.de/factbook/resource/Niger	Dbpedia-Factbook	-1	0	0	0
435	http://dbpedia.org/resource/Libyan_Arabic	http://dbpedia.org/ontology/spokenIn	http://www4.wiwiss.fu-berlin.de/factbook/resource/Nigeria	Dbpedia-Factbook	0	3	2	1
436	http://dbpedia.org/resource/Deori_language	http://dbpedia.org/ontology/spokenIn	http://www4.wiwiss.fu-berlin.de/factbook/resource/Tunisia	Dbpedia-Factbook	0	3	2	1
437	http://dbpedia.org/resource/Pennsylvania_German_language	http://dbpedia.org/ontology/spokenIn	http://www4.wiwiss.fu-berlin.de/factbook/resource/India	Dbpedia-Factbook	0	3	3	0
438	http://dbpedia.org/resource/Susu_language	http://dbpedia.org/ontology/spokenIn	http://www4.wiwiss.fu-berlin.de/factbook/resource	Dbpedia-Factbook	0	3	3	0

Abb. 3.3. links Tabelle

In der Abbildung 3.4. ist ein Teil der *instances* Tabelle dargestellt. In der *instances* Tabelle werden zu einer Instanz eines Semantic Web Links die dazugehörigen Property Werte *Label* (rdfs:label), *Type* (rdf:label), *Image*, *Optional* und die *Ontology*, in der sich dieser Link befindet, gespeichert. Die Spalte *Resource* ist der Primärschlüssel dieser Tabelle und speichert die URI der Instanzen. Die Ontologie-Templates (siehe Kapitel 2.4.2.) definieren für jede Wissensbasis eine ontologie-spezifische Property, diese sollen dem Spieler nähere Informationen über die Instanzen liefern. Dabei werden diejenigen Properties ausgewählt, die für den Menschen verständliche Informationen beinhalten. Zum Beispiel wurde für die Ontology DBpedia die Property dbpedia-owl:abstract ausgewählt oder für Factbook die Property factbook:background. Die Properties rdfs:label (*Label*) und rdf:type (*Type*) hingegen sind im Allgemeinen nicht ontologie-spezifisch und können für die meisten Ontologien übernommen werden. Besitzt eine Instanz ein Bild wird die Bild URI in *Image* vermerkt.

Resource	Label	Type	Optional	Ontology	Image
http://dbpedia.org/resource/Afrikaans	Afrikaans || Afrikaans || Afrikaans	owl:Thing || <http://dbpedia.org/ontology/Languag...	Afrikaans is a West Germanic language, spoken nati...	Dbpedia	<http://upload.wikimedia.org/wikipedia/commons/thu...
http://dbpedia.org/resource/Amazonian_Manatee	Amazonas-Manati || Amazonian Manatee || Trichech...	owl:Thing || <http://dbpedia.org/ontology/Animal>...	The Amazonian Manatee (Trichechus inunguis) is a s...	Dbpedia	<http://upload.wikimedia.org/wikipedia/commons/thu...
http://dbpedia.org/resource/American_Lion	Amerikanischer Löwe || Panthera leo atrox || Ame	owl:Thing || <http://dbpedia.org/ontology/Animal>...	The American lion (Panthera atrox) also known as t...	Dbpedia	<http://upload.wikimedia.org/wikipedia/commons/thu...

Abb. 3.4. instance Tabelle

[7] Siehe Kapitel 2.3.5.

Die Highscores der Spieler werden in der *highscore* Tabelle aufgelistet. Diese Tabelle speichert den Namen und den Punktestand eines Spielers.

Client

Der Client besitzt 2 Kern Komponenten, zum einen die *VerifyComponent* konzipiert für die Steuerung der Verifikation im Client und zum anderen die *GameComponent* zur Steuerung von allen Prozessen die mit dem 2D Spiel im Zusammenhang stehen. Diese beiden Komponenten verwalten die Benutzereingaben und bilden zusammen die graphische Benutzeroberfläche des Veri-Links Clients. Die Möglichkeit Semantic Web Links darzustellen und dem Spieler die Verifikation dieser Links zu ermöglichen bietet die *VerifyComponent* an. Die gesamte Logik und Grafik des 2D Spiels wurden in der *GameComponent* implementiert. Der Client hat keinen direkten Zugriff auf die MySQL Datenbank. Datenbank Operationen oder Anfragen, welche vom Client ausgehen, müssen zunächst an den Server geschickt werden, diese werden vom Server bearbeitet und dann ausgeführt. Die Abbildung 3.5. ist zeigt einen Ausschnitt aus dem Veri-Links Client, die *GameComponent* wurde rot markiert und die *VerifyComponent* grün.

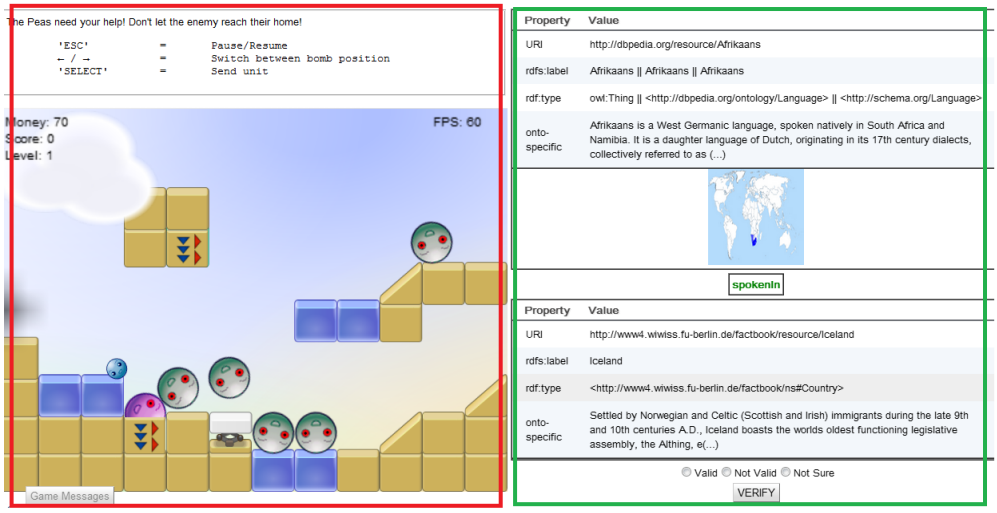

Abb. 3.5. Client Screenshot

Server

Der Server verwaltet eine Teil-Kopie der *instances* und *links* Tabelle der Datenbank im Hauptspeicher. Benötigt der Client neue zu verifizierende Semantic Web Links, so stellt er eine Anfrage an den Server, welcher den benötigten Link aus seiner Datenbankkopie auswählt und diesen an den anfragenden Client schickt. Es wird immer nur ein Link pro Anfrage verschickt.

Stellt der Client eine Anfrage nach der Highscore Liste von Veri-Links, wird diese Liste vom Server aus der Datenbank abgerufen und dem Client zugeschickt.
Informationen wie die vom Spieler getätigten Link Verifikationen oder der Punktestand des Spielers, werden vom Client zur Bearbeitung an den Server übertragen. Dieser aktualisiert daraufhin die MySQL Datenbank mit den neuen Informationen.
Das Einfügen von neuen Daten in die *instance*- und *link* Tabelle wird durch den *RDFHandler* realisiert.

3.2.2. Prozesse

Datenbank Initialisierung

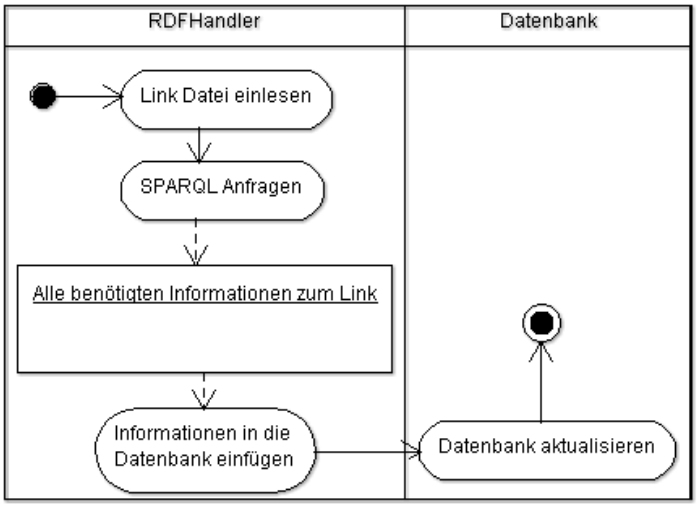

Abb. 3.6. Initialisierung der Datenbank Aktivitätsdiagramm

In der Abbildung 3.6. wird dargestellt wie mit Hilfe der RDFHandler Klasse, Semantic Web Links aus einer Link Datei in die MySQL Datenbank eingefügt werden. Nachdem die RDFHandler Klasse die RDF-Tripel aus einer N-TRIPLE oder Alignment Datei ausliest, werden für jede Instanz eines Links SPARQL Anfragen an die Endpoints der Ontologien gestellt, um die Property Werte der Instanzen zu erhalten. Die gesamten entstandenen Daten werden am Ende in die Datenbank eingefügt.

Server Daten initialisieren

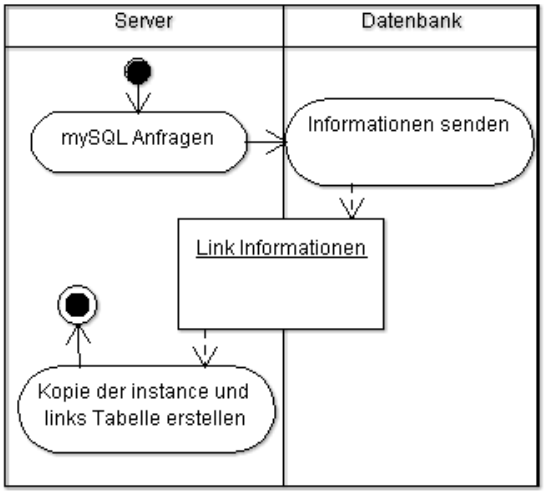

Abb. 3.7. Serverdaten Initialisierung

Der Client hat keinen direkten Zugriff auf die MySQL Datenbank. Benötigt der Client Daten aus der Datenbank, werden ihm diese durch den Server zugesendet, welcher eine Kopie der Datenbanktabellen *instance* und *links* in seinem Zwischenspeicher aufbewahrt. Einen groben Überblick über den Prozess der Initialisierung der Serverdaten ist in der Abbildung 3.7. zu finden. Mittels SQL Anfragen an die Datenbank werden die benötigten Links und die Property Werte der Instanzen abgerufen.

Spielablauf

Die Abbildung 3.8. zeigt das Aktivitätsdiagramm für einen Veri-Links Spielablauf. Beim Start des Spiels wird der Spieler gebeten Veri-Links seinen Namen mitzuteilen und die verlinkten Ontologien auszuwählen. Der Name wird benötigt falls der Spieler sich in die Highscore Liste eintragen möchte. Die Informationen bezüglich der verlinkten Ontologien begrenzen die Anzahl der möglichen Links die dem Spieler zur Verifikation freigegeben werden. Nach dieser Auswahl werden dem Spieler nur noch diejenigen Links präsentiert, die eine Verbindung zwischen Instanzen aus den beiden ausgewählten Ontologien definieren. Wurden die benötigten Informationen vom Spieler eingegeben, stellt der Client eine Link Anfrage an den Server. Der Server wählt einen Link aus seiner gepufferten Datenbankkopie aus und schickt diesen an den Client. Bei diesem Datentransfer wird kein Geldbonus mit übertragen, da der Spieler noch keine Verifikation getätigt hat. Nach dem Erhalt des Links, stellt der Client diesen für die Verifikation dar. Der Spieler hat nun die Möglichkeit das 2D Spiel zu spielen oder eine Link Verifikation durchzuführen.

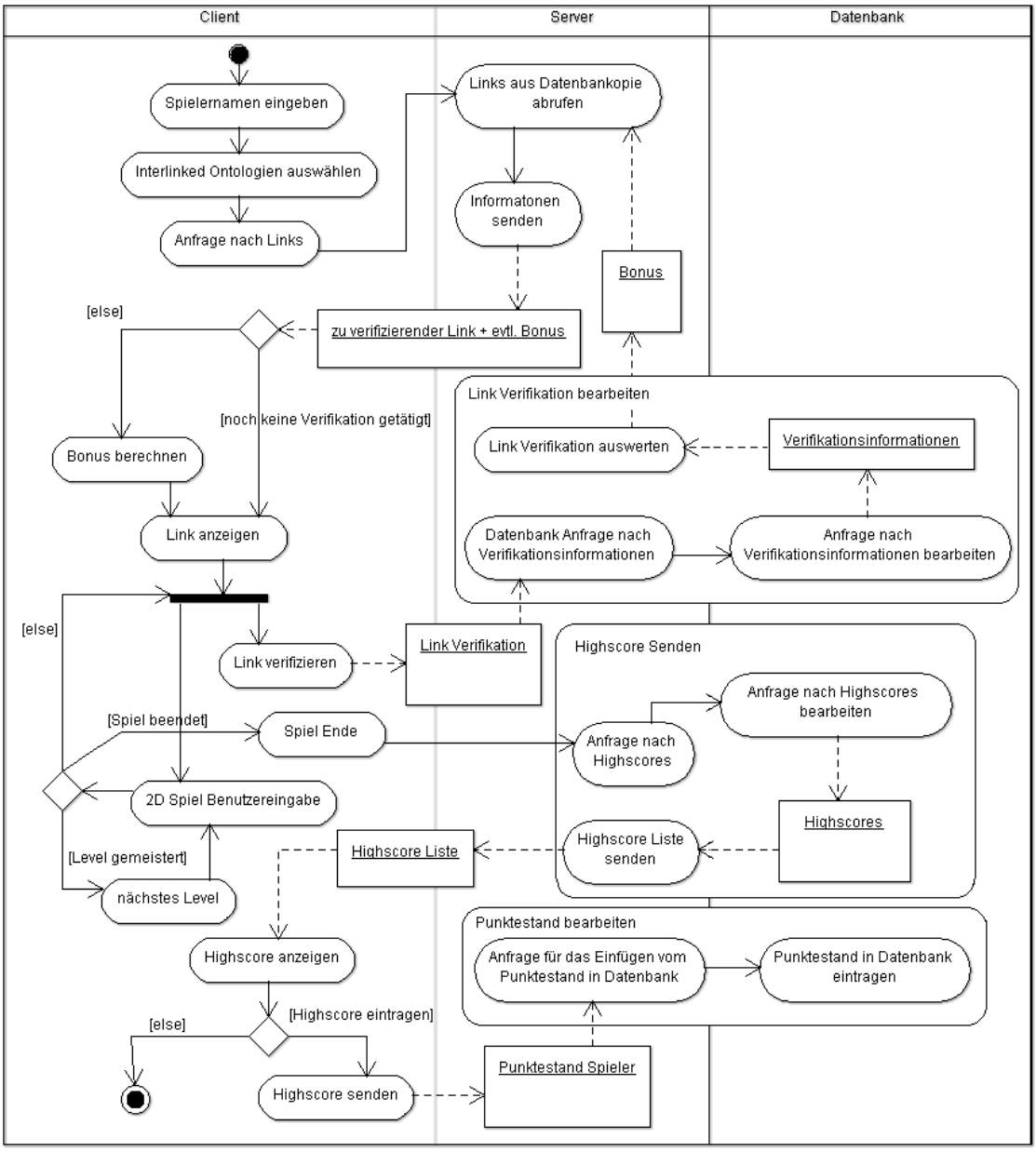

Abb. 3.8. Spielablauf Aktivitätsdiagramm

Bei der Ausführung einer Link Verifikation werden die Entscheidungen des Spielers an den Server übertragen. Der Spieler bekommt für jede getätigte Verifikation einen bestimmten Spielgeld Betrag gutgeschrieben, dieses Spielgeld wird benötigt um das 2D Spiel erfolgreich beenden zu können. Um die Ausnutzung[8] dieses Belohnungsmechanismus einzudämmen, kann ein Spieler erst nach einer bestimmten Wartezeit eine weitere Verifikation durchführen. Der Server wertet die ihm zugesendete Verifikation aus, indem er sie mit anderen Verifikationen aus der Datenbank vergleicht. Stimmt die Verifikation des Spielers mit den Verifikationen der Mehrheit überein, so wird der

[8] Sollte der Spieler ununterbrochen Verifikationen durchführen nur des Geldes wegen, ohne die Links davor gründlich zu analysieren.

Spieler mit einem Geldbonus belohnt. Dieser Mechanismus soll den Spieler motivieren, eine zuverlässige Verifikation durchzuführen. Der Geldbonus wird daraufhin zusammen mit einen neuen zu verifizierenden Link an den Client geschickt. Der Spieler bekommt den Geldbonus gutgeschrieben und der neue Link wird zur Verifikation freigegeben.
In der 2D Spiel Komponente kann der Spieler sich durch verschiedene Level durchspielen. Das 2D Spiel ist beendet, wenn er entweder das letzte Level gemeistert hat oder wenn er vom Gegner besiegt wurde. Trifft einer der genannten Fällen ein, so wird dem Spieler die Highscore Liste angezeigt, welche der Server von der Datenbank abfragt und dem Client daraufhin zusendet. Der Spieler hat zusätzlich die Möglichkeit seinen Punktestand in die Highscore Liste einzutragen. Entscheidet er sich dagegen so wird das Spiel beendet. Entscheidet er sich für einen Highscore Eintrag, so werden der Benutzername und der Punktestand des Spielers an den Server geschickt. Diese Informationen werden daraufhin vom Server in die Datenbank eingefügt.

4. Implementierungsarbeit

4.1. Paketstruktur

Die Abbildung 4.1. gibt einen Überblick über die Paketstruktur von Veri-Links.
In dem Paket *net.saim.game.client* befindet sich der clientseitige Quellcode, welcher am Ende vom GWT Compiler in JavaScript umgewandelt und im Web Browser ausgeführt wird. Die Implementierung der *VerifyComponent* wird in dem Paket *net.saim.game.client.verify* realisiert und die der *GameComponent* in dem *net.saim.game.client.core* Paket. Die Serverklassen müssen im Vergleich zum Client-Code nicht in JavaScript übersetzt werden, sie laufen als JavaServlets und sind dem Paket *net.saim.game.server* untergeordnet. Das *net.saim.game.shared* Paket beinhaltet alle Datentypen, welche sowohl vom Client als auch vom Server benutzt werden, dadurch wird eine redundante Speicherung von Klassen verhindert und ihre Konsistenz bewahrt. Diese Klassen definieren die Nachrichten, die zwischen Client und Server ausgetauscht werden. Sie besitzen serialiserbare Arrays in denen sich die zu übertragenen Daten befinden, in GWT können diese Arrays ohne große Anstrengungen im Netzwerk verschickt werden. Das Paket *net.saim.game.templates* umfasst die *Template* Klassen. Templates definieren für alle Instanzen einer Wissensbasis den SPARQL-Endpoint und diejenigen Properties, welche in der Veri-Links Datenbank gespeichert und dem Spieler in der *VerifyComponent* angezeigt werden.

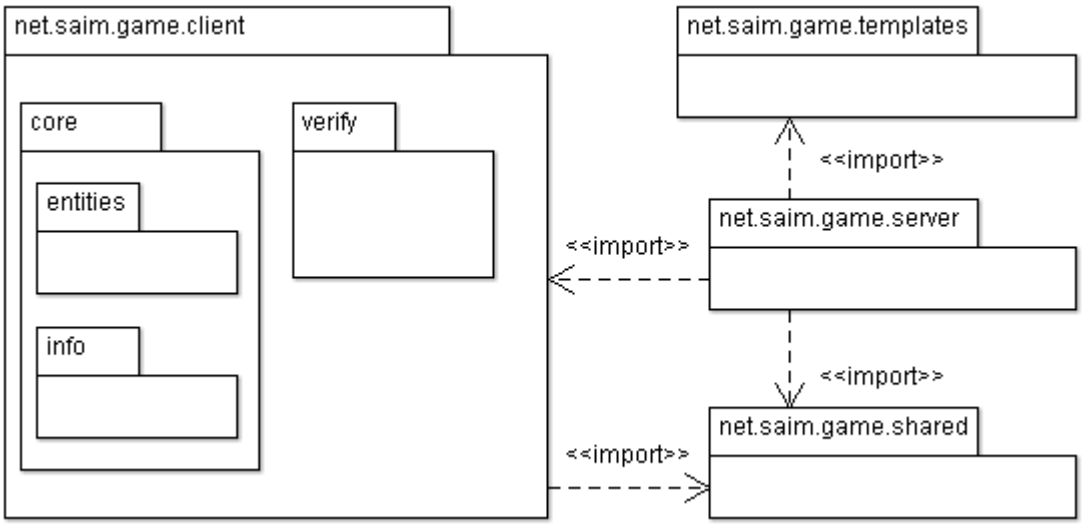

Abb. 4.1. Paketdiagramm

4.2. Implementierungsbeschreibung

Dieses Kapitel behandelt die Veri-Links Pakte der Reihe nach, dabei werden die Implementierungen der einzelnen Klassen erläutert. Für ein besseres Verständnis werden einige Erläuterungen mit Grafiken und Ausschnitten aus dem Quellcode ergänzt.

4.2.1. net.saim.game.client

Application

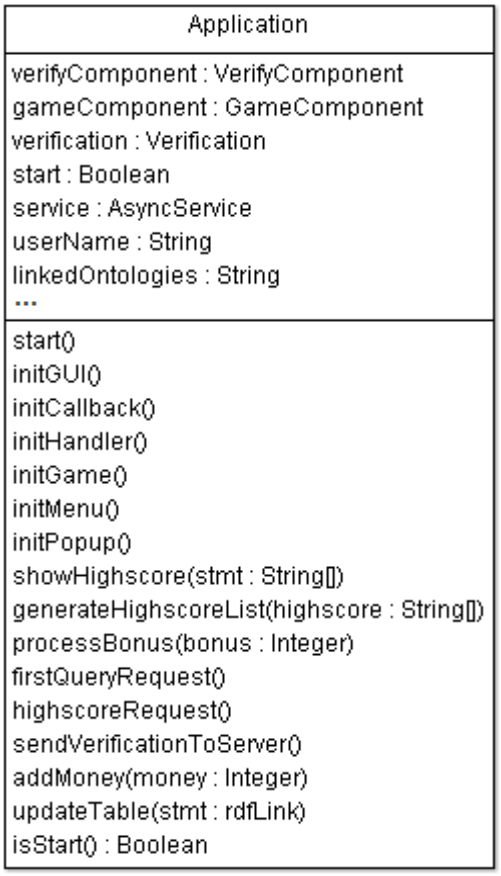

Abb. 4.2. Application UML Klassendiagramm

Die Klasse *Application* ist eine Unterklasse von *playn.html.HtmlGame* und dem Interface *com.google.gwt.core.client.EntryPoint*. Sie bildet den Entry Point und somit den Startpunkt für das gesamte GWT Modul. *Application* dient dazu alle benötigten graphischen und logischen Komponenten des Clients zu initialisieren. Die Methode *start()* leitet den Vorgang ein, zuerst wird das Startmenü durch *initMenu()* initialisiert, welches dem Spieler die Möglichkeit bietet, zum einen Veri-Links seinen Benutzernamen mitzuteilen und zum anderen zwei zu verlinkende Ontologien auszuwählen. Nach dieser Auswahl werden dem Spieler nur noch diejenigen Links

präsentiert, die Instanzen aus den beiden ausgewählten Ontologien miteinander verbinden. Diese Information wird in dem Attribut *linkedOntologies* gespeichert und bei einer Serveranfrage nach neuen zu verifizierenden Links genutzt, um den Server mitzuteilen welche Art von Links der Client benötigt. Der Benutzername des Spielers wird in dem Attribut *userName* aufbewahrt, dieser wird für einen eventuellen Highscore Eintrag in der Datenbank benötigt. Danach werden die restlichen init Methoden aufgerufen, welche die *VerifyComponent*, die für die Kommunikation mit dem Server benötigten AsyncCallbacks[9], den EventHandler[10] und die *GameComponent* erzeugen. Dabei wird das Attribut *start* auf true gesetzt, welches den Komponenten signalisiert das, dass Spiel geladen wurde und nun anfängt.

Neben der Initialisierung steuert *Application* die clientseitige Kommunikation mit dem Server durch Zuhilfenahme von GWT Remote Procedure Calls (GWT RPCs).[11] Es werden dabei drei AsyncCallbacks definiert, welche auf den Erfolgs- oder Fehlerfall einer Anfrage reagieren. Das Attribute *callbackGetLink* wird bei Anfragen nach neuen Semantic Web Links genutzt, *callBackSendScore* und *callBackGetScore* hingegen sind für Highscore spezifische Anfragen konzipiert.
Die Methode *sendVerificationToServer* überträgt die Verifikation des Spielers an den Server, indem es die *userVerification* Methode des ServiceAsync Interfaces aufruft, mit der getätigten Verifikation und *callbackGetLink* als Parameter. Der Server antwortet darauf mit einem String Array, welches die zu verifizierenden Links und den Geld Bonus beinhaltet. Daraufhin ersetzt die Methode *updateTable* die alten Links der *VerifyComponente* durch die neuen Links. Der Bonus wird mit *processBonus* bearbeitet und dem Spieler durch *addMoney* auf sein Punktekonto gutgeschrieben.
Die *firstQueryRequest* Methode dient zur erstmaligen Anfrage nach zu verifizierenden Links. Die Serveranfrage wird mit der *highscoreRequest* Methode des ServiceAsync Interfaces gestellt, dabei wird dieser Methode nur das *callbackGetLink* Attribut übergeben. Daraufhin bekommt sie vom Server die benötigten Informationen zugeschickt und bearbeitet diese auf die gleiche Weise wie *sendVerificationToServer*.
Um die Highscore Liste von der Datenbank zu bekommen wird *highscoreRequest()* ausgeführt, welche die gleichnamige Methode des ServiceAsync Interfaces mit dem Parameter *callbackgetScore* aufruft. Diese liefert bei erfolgreicher Durchführung ein String Array indem sich die Highscore Informationen befinden. Das Array wird mit *generateHighscoreList* bearbeitet und mit *showHighscore* in einem Popup Fenster angezeigt.

[9] com.google.gwt.user.client.rpc.AsyncCallback
[10] com.google.gwt.event.dom.client.ClickHandler
[11] Siehe Kapitel 2.3.1., Abschnitt Remote Procedure Calls für Informationen bzgl. der Realisierung der Client-Server Kommunikation.

Der *MyHandler* wird von *initHandler()* initialisiert und ist für die Ereignisbehandlung des *verify Buttons* der *VerifyComponent* zuständig, es erbt das Interface ClickHandler und kann dadurch mit der Methode *onClick* auf ClickEvents reagieren. Der Methodenrumpf von *onClick* wird nur ausgeführt wenn das Spiel nicht pausiert ist. Bei Betätigung des *verify Buttons* überprüft *onClick* ob der Spieler schon ein Radiobutton selektiert hat, wenn nicht wird er aufgefordert dies nachzuholen, wenn ja ruft *onCLick* die *sendVerificationToServer()* Methode auf. Daraufhin wird der *verify Button* zunächst deaktiviert und nach Ablauf eines Timers wieder aktiviert, dieser Mechanismus verhindert eine ununterbrochene Betätigung des *verify Buttons*.

VerifyComponent

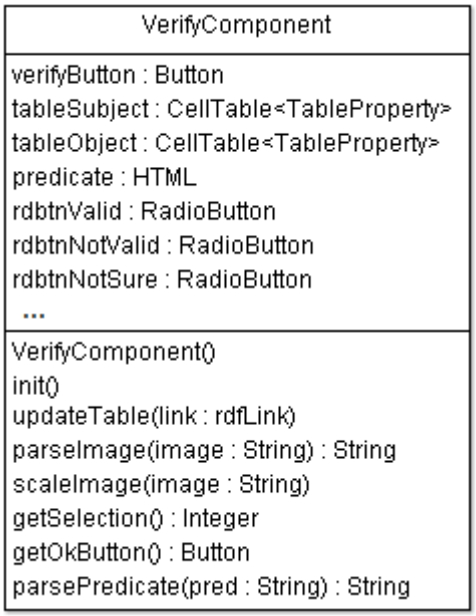

Abb. 4.3. VerifyComponent UML Klassendiagramm

VerifyComponent ist eine Unterklasse von VerticalPanel[12], es bildet die GUI für die Darstellung und Verifikation der Semantic Web Links. Diese Links werden in Veri-Links mit dem Datentyp *rdfLink*[13] modelliert. Die beiden Instanzen eines Links werden durch zwei Tabellen[14] repräsentiert, dabei stellen die Tabellenzeilen die Property Werte der Instanzen dar. Die URI des Prädikats wird durch die Methode *parsePredicate* gekürzt und in Form eines HTMLs[15] dargestellt. Empfängt der Client einen neuen Link wird die *updateTable* Methode aufgerufen und es erfolgt eine Ersetzung der alten Property Werte mit den Neuen in den Tabellenspalten. Besitzt eine Instanz ein Bild,

[12] com.google.gwt.user.client.ui.VerticalPanel
[13] Siehe Kapitel 4.2.3. Abschnitt rdfLink.
[14] com.google.gwt.user.cellview.client.CellTable
[15] com.google.gwt.user.client.ui.HTML

wird dieses mit Hilfe der *scaleImage* Methode in eine angemessene Größe skaliert und angezeigt. Die URI der Bilder im String Format werden durch Spitze Klammern[16] eingeschlossen, deswegen müssen diese Klammern mit der *parseImage* Methode entfernt werden. Die *init* Methode initialisiert die graphischen Elemente und die benötigten Objekte der *VerifyComponent*.

Während des Spiels soll der Spieler die angezeigten Instanzen analysieren und dann entscheiden ob das Prädikat eine korrekte Beziehung zwischen den beiden Instanzen darstellt. Es stehen drei Optionen zur Auswahl, die mit Radio Button[17] selektiert werden können: Dieser Link ist valid, nicht valide, oder der Spieler ist sich nicht sicher. Der ausgewählte RadioButton kann mit *getSelection()* ermittelt werden. Wurde eine Entscheidung getroffen, kann diese samt der *ID* des Links durch das Betätigen des *verify Buttons* an den Server gesendet werden.

Table Property

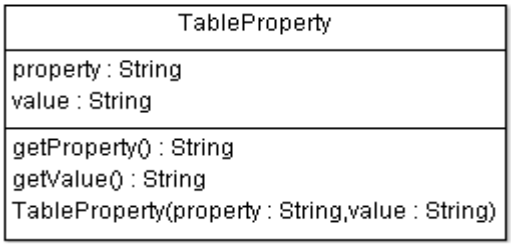

Abb. 4.4. TableProperty UML Klassendiagramm

TableProperty ist ein einfacher Datentyp, welcher eine Property und ihren Wert repräsentiert. Mit zwei Getter Methoden können auf die Attribute der *TableProperty* zugegriffen werden. *TableProperty* wird als Datentyp in der *VerifyComponent* ind den Reihen der Instanztabellen *tableSubject* und *tableObject* verwendet.

Service

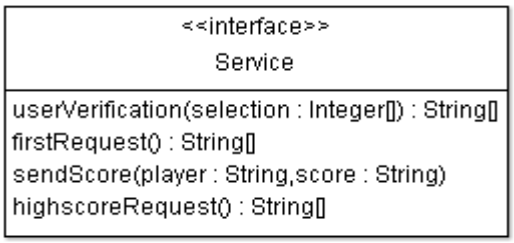

Abb. 4.5. Service UML Diagramm

[16] Siehe Kapitel 3.2.1. Abschnitt Datenbank
[17] com.google.gwt.user.client.ui.RadioButton

Veri-Links benutzt GWT RPCs[18] zur Realisierung der Client-Server Kommunikation. Das Service Interface ist eine Unterklasse von RemoteService, es listet alle Client-Server Kommunikationsmethoden von Veri-Links auf. Um die RPC Funktion von GWT nutzen zu können muss parallel zu diesem Interface ein ServiceAsync Interface im Clientpaket und eine *ServerImpl* Klasse im Serverpaket implementiert werden.

Wenn ein Spieler eine Verifikation durchgeführt hat, wird diese als Integer Array mit der Methode *userVerification* an den Server gesendet. Der Server verarbeitet die Nachricht und antwortet mit einem String Array, in dem sich zum einen der neue zu verifizierende Link befindet und zum anderen der vom Server ausgerechnete Bonus für die Verifikation. Am Anfang des Spiels hat der Spieler noch keine zu verifizierenden Daten und es wird lediglich eine Anfrage nach einem Link gestellt, dies erledigt die *firstRequest()* Methode. Die Methode sendScore schickt den Punktestand des Spielers an den Server. Mit *highscoreRequest* wird nach der Hichscore Liste der Datenbank gefragt, welche daraufhin vom Server in Form eines String Arrays verschickt wird.

4.2.2. net.saim.game.server

ServiceImpl

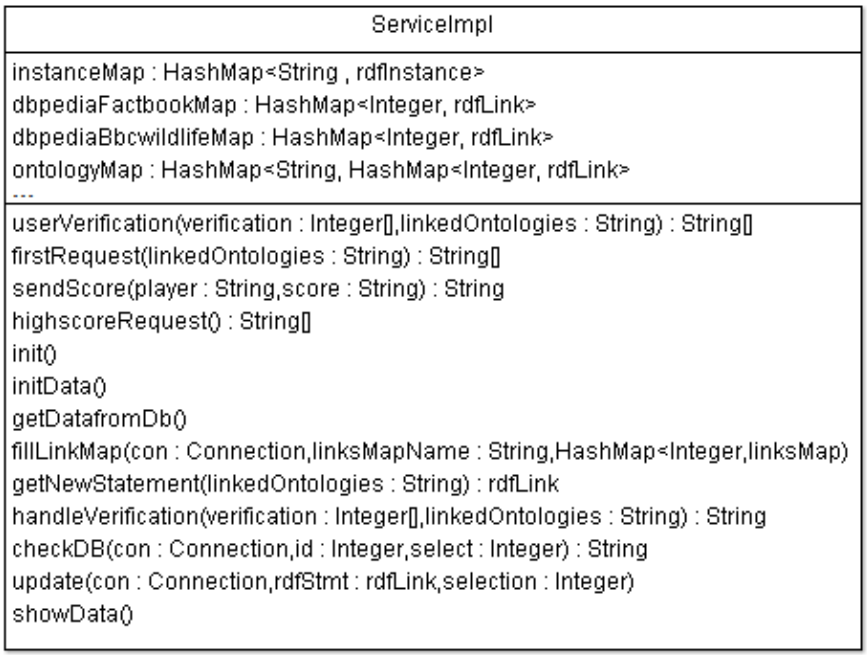

Abb. 4.6. ServiceImpl UML Klassendiagramm

[18] GWT RPCs benutzen asynchrone Kommunikation. Hat man nun ein RemoteService Interface mit dem Namen "Service" definiert, muss eine asynchrone Version dieses Interfaces erstellt werden. Es ist notwendig, dass der Name dieses asynchronen Interfaces „ServiceAsync" lautet (Interfacename als Präfix und „Async" als Suffix). Siehe Kapitel 2.3.1 für weitere Informationen.

Anfragen vom Client werden in der Klasse *ServiceImpl* bearbeitet. Nach dem GWT RPC Standard muss diese Server Klasse von der RemoteServiceServlet[19] Klasse abgeleitet werden und das Service Interface, welches sich im Client Paket befindet, implementieren.[20] Interaktionen mit der MySQL Datenbank werden mit Hilfe der Methoden des *DBTools* realisiert.

Beim Start dieser Klasse wird die *init()* Methode ausgeführt und eine Kopie der Datenbank auf dem Server erstellt. Dabei werden die beiden Datenbanktabellen *instances* und *links* auf die HashMaps[21] *instanceMap* und *ontologyMap* des Servers abgebildet. Die *ontologyMap* verwaltet wiederum einzelne HashMaps, in denen sich die URIs der Links befinden. Für je zwei verlinkte Ontologien erfolgt eine manuelle HashMap Deklarierung, diese Aufteilung der links Tabelle in verschiedene HashMaps ermöglicht ein effektives Verwalten und Bearbeiten von Linkanfragen. Die Abbildung 4.7. veranschaulicht den Aufbau der Server HashMaps.

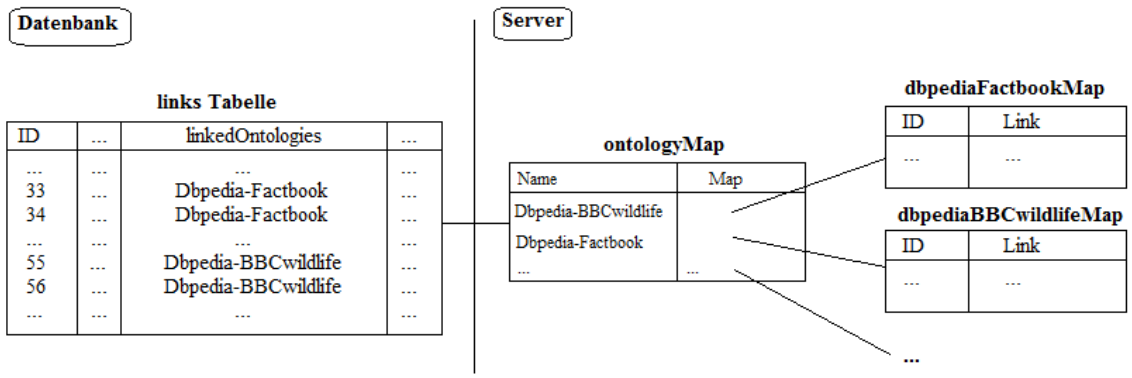

Abb. 4.7. HashMaps der ServiceImpl Klasse

Aus Performance Gründen holt sich der Server zunächst nur einige wenige Daten in den Zwischenspeicher, im weiteren Verlauf des Spiels können die verifizierten Daten mit neuen Daten aus der Datenbank ersetzt werden. Die Methode *getDatafromDB* füllt die *ontologyMap* und die *fillLinkMap* Methode füllt die HashMaps, welche der *ontologyMap* untergeordneten sind, mit Daten auf.
Ruft der Client die Kommunikationsmethode *firstRequest* auf, so schickt der Server ihm einen zu verifizierenden Link in Form eines *rdfLinks*[22] aus der Datenbankkopie. Dabei liefert die Methode *getNewLink* den Link, mit den kleinsten *counter* Wert.
Durch die Methode *userVerification* schickt der Client dem Server die vom Spieler getätigte Link Verifikation. Diese Daten werden vom Server mit der Methode

[19] com.google.gwt.user.server.rpc.RemoteServiceServlet
[20] Das asynchrone Service Interface wird jedoch nicht geerbt
[21] java.util.HashMap
[22] Es wird genauer gesagt das serialisierbare array des rdfLinks gesendet.

handleVerification bearbeitet und danach für die Aktualisierung der Daten in der MySQL Datenbank genutzt. Die Berechnung des Geld Bonus für eine getätigte Verifikation erfolgt durch die *checkDB* Methode. Der Datenbankaktualisierungs-prozess wird von der Methode *updateDB* gesteuert. Nachdem diese Prozesse beendet wurden, sendet der Server dem Client auf die gleiche Art und Weise wie bei *firstRequest* einen Link zur Verifikation.
Ein Spieler überträgt seinen Punktestand mit Hilfe der *sendScore* Methode an den Server. Dieser fügt die Information daraufhin in die Datenbank ein.
Beim Aufruf der *highscoreRequest* Methode sendet der Server dem Client die Highscore Liste der Datenbank in Form eines String Arrays.

RDFHandler

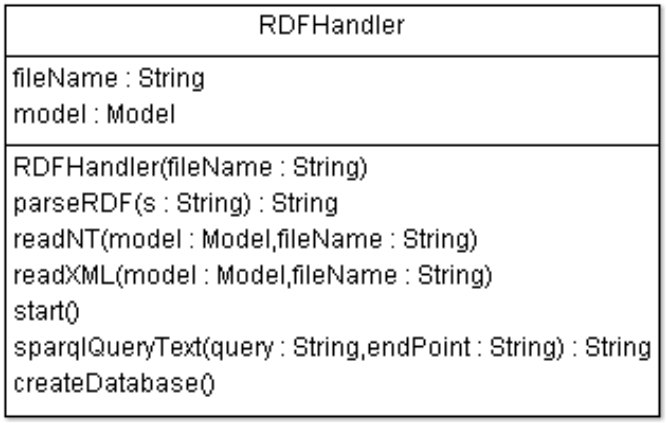

Abb. 4.8. RDFHandler Klassendiagramm

Alle Einträge der Veri-Links Datenbank werden von der *RDFHandler* Klasse generiert. Diese Klasse bedient sich der Methoden des Jena Frameworks um RDF-Ausdrücke einzulesen, SPARQL Queries auszuführen und deren Result Set zu bearbeiten.

Die MySQL Datenbank kann mit der Methode *createDatabase* automatisch erstellt werden, einzig die Verbindungsdaten zur Datenbank bedürfen einer manuellen Eingabe. Ausgehend von einer Link Datei in XML/Alignment oder N-Triple Format, werden die benötigten Informationen mit *readXML* oder *readNT* in ein RDF Model[23] eingelesen. Für jede Ontologie einer Instanz, welche in die Datenbank eingefügt werden soll, besitzt Veri-Links ein Template. Diese Templates definieren die abzufragenden Properties der Instanz und besitzen alle benötigten Informationen für eine RDF-Anfrage. Sie müssen in der *start()* Methode manuell gesetzt werden, der weitere Ablauf ist jedoch automatisiert.

[23] com.hp.hpl.jena.rdf.model.Model

Um nun an die Werte der Properties zu gelangen, welche dem Spieler am Ende in der *VerifyComponent* angezeigt werden, ist es notwendig für je eine Instanz alle ihre Properties anzufragen. Aus dem Grund wird das Model durchlaufen und der Algorithmus generiert für jedes Subjekt und Objekt eines RDF Tripels eine SPARQL Query für die jeweiligen Properties. Der Methode *sparqlQueryText* werden die SPARQL Query und der dazugehörige SPARQL Endpoint (beide in String Format) übergeben. In dieser Methode erfolgt die Ausführung der SPARQL Querys, welche ein Result Set mit den Werten der Properties zurückliefert. Diese müssen bevor sie in die Datenbank eingefügt werden können, durch die *parseRDF* geparsed und eventuell gekürzt werden, wenn sie zu lang sind.

DBTool

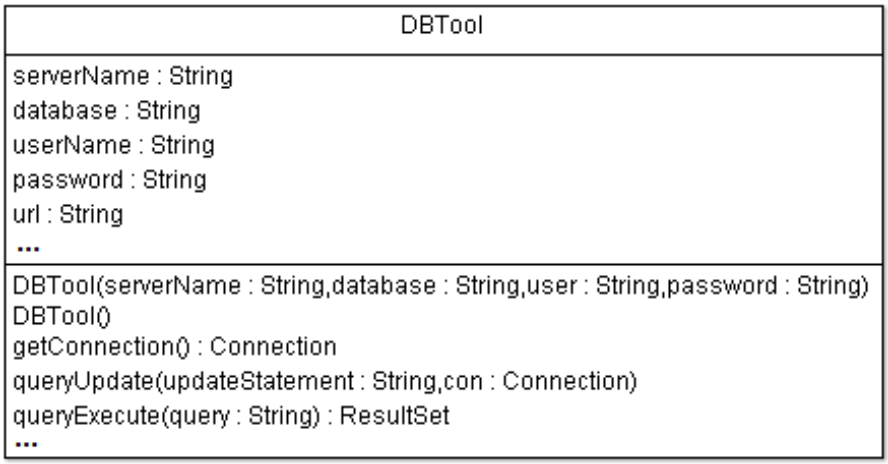

Abb. 4.9. DBTool Klassendiagramm

In diesem Paket sind Methoden für die Arbeit mit der MySQL Datenbank implementiert. Für das Spiel Veri-Links sind nur einige Methoden relevant.
Mit den Konstruktoren werden die Verbindungsdaten initialisiert. Der Konstruktor ohne Parameter ist auf die Veri-Links Datenbank geeicht, das heißt die Verbindungsdaten wurden schon im Vorfeld definiert. Die Methode *getConnection()* liefert die Verbindung zur Datenbank, wobei der JDBC Driver com.mysql.jdbc.Driver verwendet wird. Nachdem eine erfolgreiche Verbindung hergestellt wurde kann entweder ein SQL-EXECUTE Befehl, welcher ein Resultset als Rückgabewert liefert, mit *queryExecute* ausgeführt werden oder ein einfacher SQL-UPDATE Befehl mit *queryUpdate*.

4.2.3. net.saim.game.shared

rdfLink

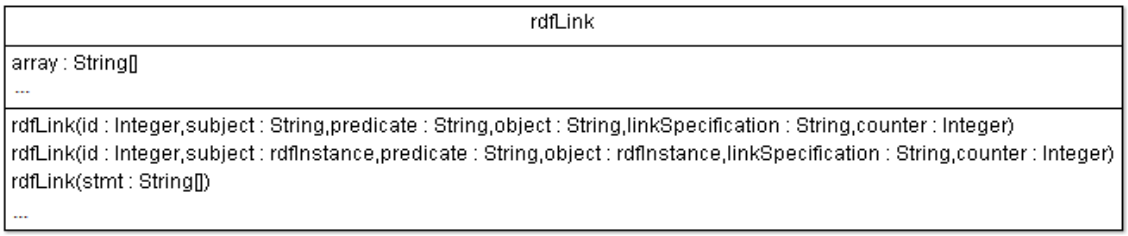

Abb. 4.10. Ausschnitt rdfLink Klassendiagramm

Ein rdfLink modelliert einen Semantic Web Link aus der VeriLink Datenbank mit den Informationen aus beiden Datenbank Tabellen (*links* und *instances* Tabelle). Insgesamt werden 14 Informationen über einen Link gespeichert:

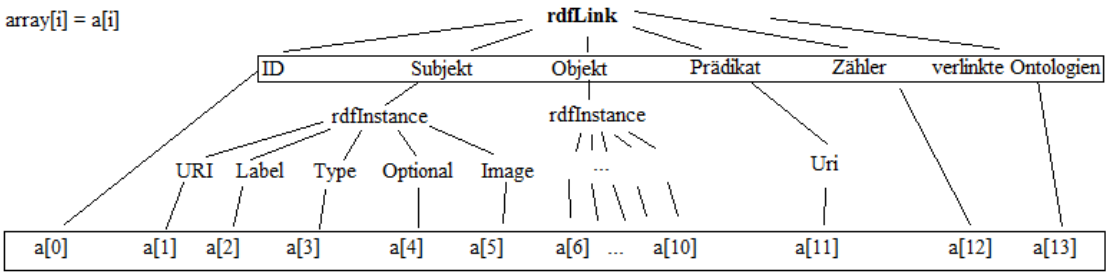

Abb. 4.11. rdfLink array

Die *VerifyComponent* zeigt die Felder eins bis elf an, die restlichen Daten werden für die spätere Auswertung der Verifikationen genutzt und sollen vom Spieler versteckt werden. Diese Klasse kann mit drei verschiedenen Konstruktoren aufgerufen werden.

rdfInstance

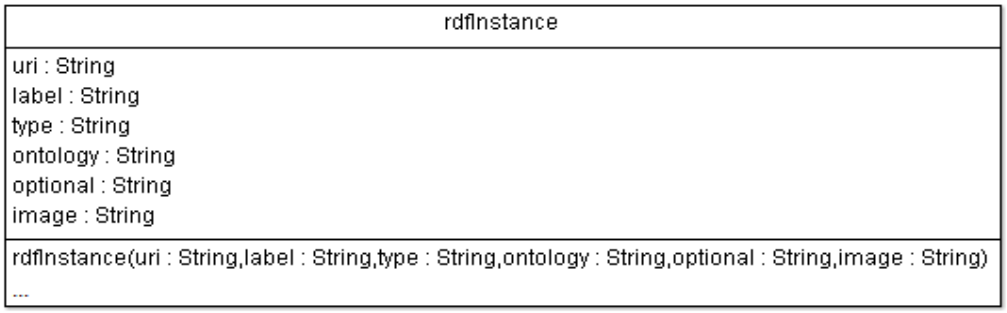

Abb. 4.12. rdfInstance UML Klassendiagramm

Ein *rdfInstance* stellt ein Objekt aus der Veri-Links Datenbanktabelle *instance* dar und besitzt dementsprechend dieselben Informationen. Dieser Datentyp wurde hauptsächlich für eine übersichtlichere Darstellung eingeführt.

Verification

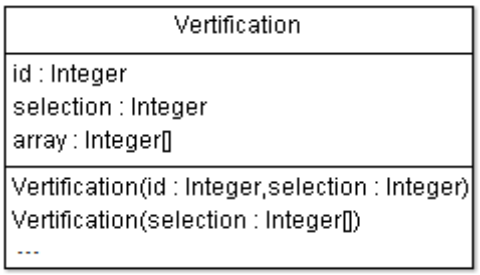

Abb. 4.13. Verification UML Klassendiagramm

Wenn der Spieler eine Verifikation durchführt, wird die *ID* des verifizierten Links und die getroffene Entscheidung in dem Datentyp *Verification* gespeichert und an dem Server zur Verarbeitung übertragen. Eine getroffene Entscheidung kann entweder aus dem Integer Wert 1, 0 oder -1 bestehen, wobei diese Werte die Aussagen „Der Link ist korrekt“, „Der Link ist nicht korrekt“ oder „Der Spieler ist sich nicht sicher“ repräsentieren.

4.2.4. net.saim.game.templates

TemplateOntology

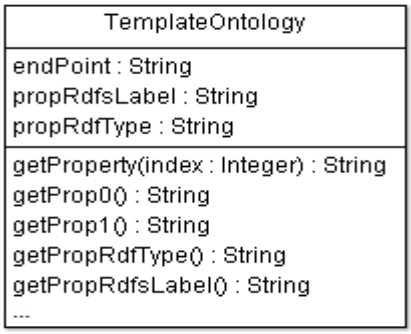

Abb. 4.14. TemplateOntology UML Klassendiagramm

TemplateOntology ist die Superklasse aller Templates. Für jede Wissensbasis lässt sich ein Template definieren, welches die für die Anzeige in der *VerifyComponent* benötigten Properties definiert. Da diese Properties ontologie-spezifisch sind ist ein automatischer Zugriff nicht ohne weiteres möglich. Die abstrakte Methode *getProperty* löst dieses Problem, sie ordnet allen Klassenattributen dieser Klasse einen Index zu, dadurch wird ein automatischer Zugriff auf diese Attribute ermöglicht.

4.2.5. net.saim.game.client.core

Dieses Paket beinhaltet die gesamten graphischen und Logischen Elemente der 2D Spiel-Komponente.

GameComponent

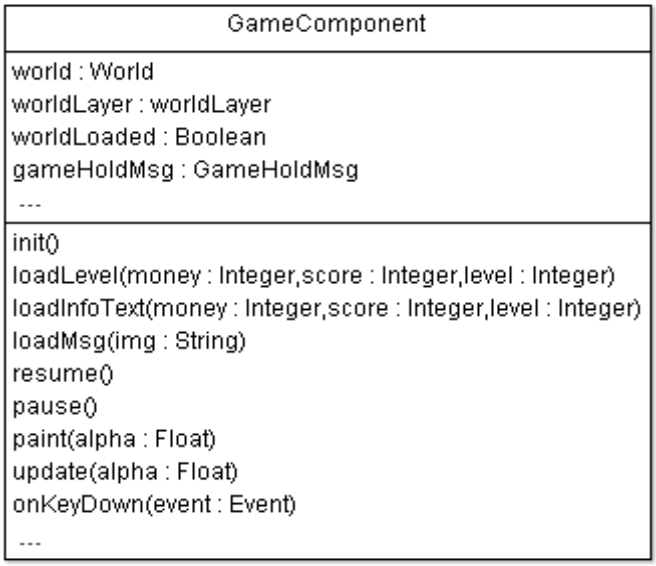

Abb. 4.15. GameComponent UML Klassendiagramm

Das gesamte Spielgeschehen des 2D Spiels findet in der *GameComponent* statt. In dieser Klasse werden die Benutzereingaben verarbeitet und der Spielablauf definiert.
Die *GameComponent* implementiert zum einen den GameLoop, bestehend aus den Methoden *init, update, updateRate* und *paint* des Game Interfaces, und zum anderen den Keyboard Listener, welcher auf Benutzereingaben mit der Tastatur reagiert.

Beim Aufruf der *init* Methode wird das Hintergrundbild des Spiels vom PlayN AssetManager geladen und der *worldLayer* erstellt. Der *wordlLayer* ist ein playn.core.GroupLayer, in ihm werden die gesamten *Entities*[24] des Spiels hinzugefügt und gerendert.
Weiterhin wird in *init()* die *GameHoldMessage*[25] initialisiert, das Level mit der *loadLevel* Methode geladen und der Keyboard Listener gesetzt.
Nachdem in *loadLevel* die statische Methode *CreateWorld* der *Loader*[26] Klasse erfolgreich beendet wurde, werden die restlichen Variablen - unter anderem die

[24] Siehe Kapitel 4.2.7.
[25] Siehe Kapitel 4.2.6., Abschnitt GameHoldMessage
[26] Siehe Kapitel 4.2.5., Abschnitt Loader.

GameWorld[27] - initialisiert, *loadInfoText* aufgerufen und *worldLoaded* auf den Wert true gesetzt. Die globale Variable *worldLoaded* gibt an, ob die Welt erfolgreich geladen wurde.

Die Methode *loadInfoText* konstruiert aus ihren Parametern ein *InfoText* Objekt und ein *FpsCounter* Objekt, diese beiden Objekte werden in den *wordLayer* hinzugefügt.
Mit Hilfe von *loadMsg* wird das Bild der *gameHoldMsg* geändert, das heißt jenes Bild wird gewechselt, welches bei der Pausierung des Spiels angezeigt wird. Nähere Informationen dazu sind im Kapitel 4.2.6. unter dem Abschnitt *GameHoldMsg* zu finden.
Das Spiel kann mit *pause()* pausiert und mit der Methode *resume()* wieder fortgesetzt werden. Die Methode *loadMsg* wird beim Eintritt in eine Pause mit dem Parameter "pause" aufgerufen. Während dieser Pause können die Methoden update und *paint* nicht ausgeführt werden.

Benutzereingaben werden in der Methode *onKeyDown* verarbeitet. Dabei lösen die folgenden Keyboard-Tasten Events aus:

- Die Rechte oder Linke Taste, bewegen den *Marker* mit der Methode *movePointer* nach rechts bzw. links. Für nähere Informationen siehe Kapitel 4.2.6. Abschnitt *Marker*.
- Wenn der Spieler genug Geld hat, erstellt die Leertaste ein neues *Pea*[28] in der *GameWorld* an der Position des *Markers*. Der Geldbetrag des Spielers wird daraufhin aktualisiert.
- Die Escape Taste hat mehrere Funktionen, zum Beispiel versetzt es das Spiel in den Pause Zustand oder beendet diesen wieder, oder es ruft die Methode *firstQueryRequest* der Klasse *Application* am Anfang eines Spiels auf, oder es startet ein neues Level mit der *newLevel* Methode von *GameWorld.*

Loader

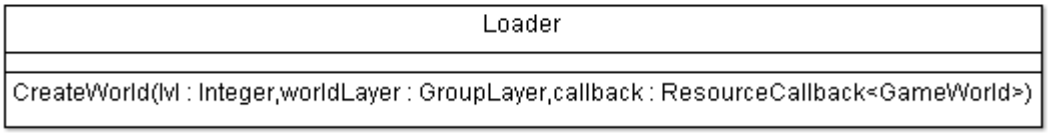

Abb. 4.16. Loader UML Klassendiagramm

Die Aufgabe der *Loader* Klasse ist es, die *Entities* der *GameWorld* zu initialisieren. In einer JSON-Datei werden alle Informationen eines Levels aufbewahrt, diese beinhalten:

[27] Siehe Kapitel 4.2.5., Abschnitt GameWorld
[28] Siehe Kapitel 4.2.7., Abschnitt Sprite

- Typen und Positionen der im Level vorkommenden *Entities*
- Positionen der *Marker*
- Level ID
- Benötigter Score für das Erreichen des nächsten Levels

Die *Loader* Klasse besteht aus einer static Methode, liest die Datei ein, verarbeitet ihre Informationen und fügt diese in die *GameWorld* ein.

GameWorld

Abb. 4.17. GameWorld UML Klassendiagramm

Die Spielewelt mit ihren „Bewohnern“, graphischen Elementen und ihrer Physik wird durch die Klasse GameWorld modelliert. Dabei stellt die Physics Engine JBox2D Methoden zur Verfügung für die Simulation und Verarbeitung der auftretenden physikalischen Prozesse in der Spielewelt. Um die JBox2D Algorithmen für die Kollisionserkennung (zum Beispiel *beginContact*) zwischen den Entities verwenden zu können, muss GameWorld das Interface ContactListener[29] implementieren.

[29] org.jbox2d.callbacks.ContactListener

GameWorld besitzt drei GroupLayers, in denen die graphischen Elemente gerendert werden:

- *staticLayerBack*, beinhaltet die ImageLayer aller Unterklassen von *StaticPhysicsEntity*
- *staticLayerFront*, im Moment wird nur der ImageLayer von *Portal* hier eingefügt
- *dynamicLayer*, umfasst alle ImageLayer der Unterklassen von *DynamicPhysicsEntity*

Beim Konstruktoraufruf werden die drei GroupLayers initialisiert und in die *worldLayer* der *GameComponent* eingefügt. Weiterhin werden im Konstruktor die physische Welt, ihre Grenzen (sprich Boden, Linke und Rechte Abgrenzung) initialisiert, und das Spiel zunächst pausiert indem die Variable paused auf true gesetzt wird.

Die Methode *newLevel* dient dazu, die gesamten *Entities* der Spielewelt zu entfernen und ein neues Level zu laden. Dabei wird unterschieden ob ein Spieler das nächste Level erreicht hat oder ob dieser verloren hat, im ersten Fall fließen die Informationen von *InfoText* (sprich *money, score ,level*) in das nächste Level ein, im zweitem Fall wird die Methode *resetInfo* aufgerufen und diese Informationen werden auf den Anfangswert zurückgesetzt.

Die update Methode besitzt eine Vielzahl von wichtigen Aufgaben. Sie reagiert auf alle Veränderungen in der GameWorld und sorgt dafür, dass *Entities* hinzugefügt, gelöscht und aktualisiert werden. Ist das Spiel jedoch pausiert, wird der Methodenrumpf nicht ausgeführt.

In jedem *update* Zyklus wird das Attribute *timer* inkrementiert, damit ist es möglich zu überprüfen ob neue *Enemies* in die *GameWorld* hinzugefügt werden sollen. Erreicht *timer* einen durch 500 teilbaren Wert, so werden durch die *sendNewWave* Methode neue *Enemies* generiert. Mit dieser Technik ist es möglich, ohne Berücksichtigung von Pausen, *Enemies* in bestimmten Zeitintervallen in die GameWorld einzufügen.
Eine weitere Aufgabe der *update* Methode ist es zu Erkennen wann ein Spiel pausiert, gewonnen, verloren oder erfolgreich beendet wurde. Dafür stehen vier Attribute zur Verfügung, welche die einzelnen Spiel-Zustände repräsentieren:

- *paused* - nimmt den Wert true an wenn das Spiel pausiert ist.
- *win* - werden die Voraussetzungen für das Erreichen des nächsten Levels erfüllt, wird dieses Attribut auf true gesetzt
- *lose* – erreicht eine Instanz von Enemy den linken Rand der Spielewelt, hat der Spieler das Spiel verloren. Das Attribut lose nimmt daraufhin den Wert true an

- *end* - der Wert true bei diesem Attribute, signalisiert das erfolgreiche Beenden des Spiels

Diese Attribute werden im Nachhinein von der *GameComponent* ausgewertet.
Zusätzlich ruft update noch die Methode *processContact* auf, welche die Kollisionen zwischen den *Entities* bearbeitet.

```
public void update(float delta) {
	if(!paused) {
		// increment Timer
		timer++;
		if (timer % 500 == 0)
			newWave = true;
		// Update Entities
		for (Entity e : entities) {
			if(e instanceof Enemy && e.getBody().getPosition().x <0)
				lose = true;
			e.update(delta);
		}
		// Add Entities
		while (!entitiesToAdd.isEmpty())
			doAdd(entitiesToAdd.pop());
		// Remove Entities
		while(!entititesToRemove.isEmpty())
			doRemove(entitiesToRemove.pop());
		// Do time step
		world.step(0.33f, 10, 10);
		processContacts();
		…
		// Check game status
		…
		//Send new wave
		if (newWave) {
			sendNewWave();
			newWave = false;
		}
	}
}
```

Quellcodeausschnitt GameWorld update Methode

Nachdem die GameWorld aktualisiert wurde, zeichnet die *paint* Methode die gesamten *Entities* und den *InfoTex*t samt *FpsCounter* unter der Voraussetzung, dass das Spiel nicht pausiert wurde.

In JBox2D kann ein Body (Siehe Kapitel 2.3.3.) nicht einfach innerhalb eines time steps entfernt werden[30], das heißt, dass es nicht möglich ist ein *Entity* unmittelbar nach seiner Kollision zu löschen. Um dieses Problem zu umgehen, wird die zu löschende

Entity bei einer Kollision zunächst mit der Methode *remove* in den *entityToRemove Stack* hinzugefügt. Nachdem der time step beendet wurde können im nächsten update Zyklus nun alle *Entities*, welche sich in diesem Stack befinden, samt ihrer Bodies und ImageLayers endgültig gelöscht werden, indem die Methode *doRemove* auf die jeweilige *Entity* aufgerufen wird.
Die gleiche Idee steckt hinter den Methoden *add* und *doAdd*, sie ermöglichen ein verzögertes Hinzufügen von *Entities* in die GameWorld, nach dem obigen Prinzip.

In der Klasse *GameWorld* wird einzig die Methode *beginContact* vom JBox2D ContactListener überschrieben. Wenn 2 Fixtures miteinander in Berührung treten generiert JBox2D ein Contact Objekt und ruft die Methode *beginContact* auf, mit dem Contact als Parameter. Die übergebene Variable wird daraufhin in den *contacts Stack* hinzugefügt. Die abstrakte Klasse Contact verwaltet Kollisionen zwischen zwei Shapes. Alle auftretenden Contacts werden in den *contacts Stack* aufbewahrt und zu einem späteren Zeitpunkt in der Methode *processContacts* bearbeitet.
Beim Aufruf der *processContacts* Methode werden zum einen Kollisionen zwischen einem *Pea* und einem *Enemy* aufgelöst. Dabei wird als erstes überprüft ob die beiden *Sprites* ihre Farbe mit der Methode *hit* ändern müssen, danach wird ihnen durch die *damage* Methode *hp* abgezogen. Zum anderen kontrolliert *processContacts* ob eine Instanz der Klasse *Portal* bei diesem Contact beteiligt ist, wenn ja wird ihre *contact* Methode aufgerufen.

4.2.6. net.saim.game.client.core.infos

Informationsklassen (InfoText, FPSCounter und GameHoldMessage)

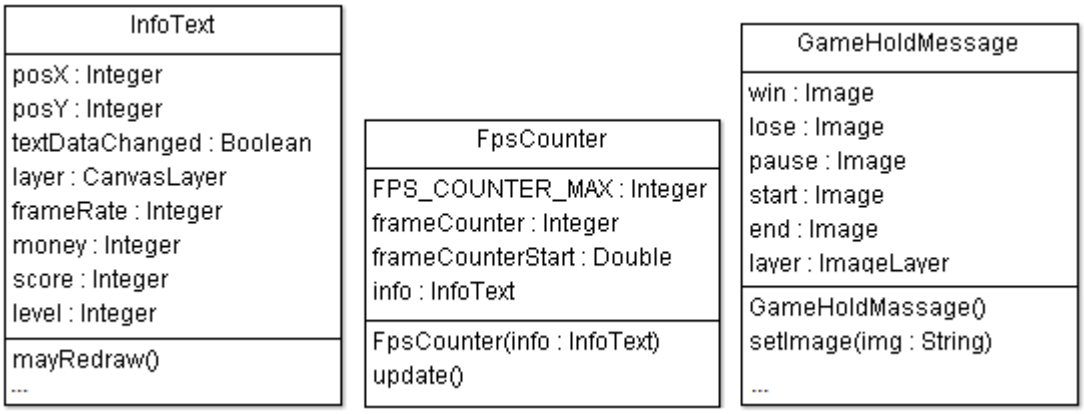

Abb. 4.18. Klassendiagramme: InfoText, FPSCounter und GameHoldMessage

InfoText: Diese Klasse zeichnet mit der Methode *mayRedraw* die gespeicherten Informationen über den Spielstand (*money, score, level*) und die *frameRate* in die *GameComponent*. Wenn das Attribut *textDataChanged* den Wert true annimmt, wird in

der Methode *mayRedraw* der Informations-Text mit playn.core.Canvas.drawText in ein playn.core.CanvasLayer gezeichnet. Es wird also nur dann gezeichnet, wenn sich die Werte der Klassenattribute geändert haben.

FPSCounter: Die frames per seconds werden in dieser Klasse berechnet und daraufhin *InfoText* übergeben. Der Berechnung der fps erfolgt in der update Methode.

GameHoldMessage: Die *GameHoldMessage* wird angezeigt wenn ein Spiel pausiert wird. Sie besitzt einen ImageLayer, welcher in den *worldLaye*r hinzugefügt wird. In diesen ImageLayer kann mit *setImage* eins von fünf verschieden Bildern (*win, lose, pause, start, end*) geladen werden.

Marker

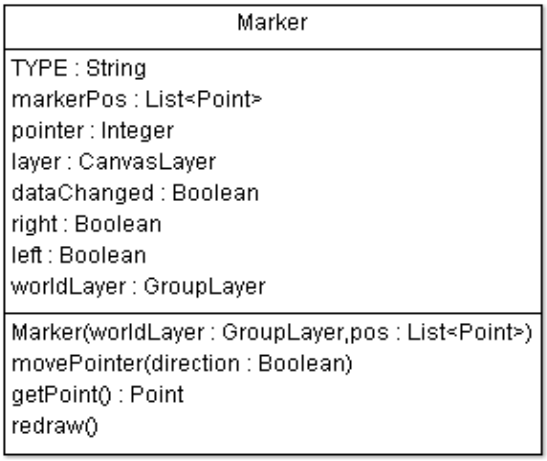

Abb. 4.19. Marker UML Klassendiagramm

In dieser Klasse werden die Positionen der Marker aufbewahrt. Ein *Marker* ist eine graphische Komponente, welche die Abwurfstelle eines *Peas* in der *GameWorld* festlegt. Innerhalb der *markerPos* Liste werden die Koordinaten als Points[30] gespeichert und mit der *movePointer* Methode kann durch diese Liste navigiert werden. Das Attribut *pointer* symbolisiert einen Zeiger, welcher die Position des zu zeichnenden Markers in der *markerPos* Liste besitzt. Durch *redraw* wird der Marker gezeichnet, dabei liefert die *getPoint* Methode mit Hilfe von *pointer* den benötigten Marker.

[30] com.google.gwt.touch.client.Point

4.2.7. net.saim.game.client.core.entities

Die Objekte der Spielwelt befinden sich in diesem Paket, sie werden auch *Entities* genannt. Jede *Entity* besitzt einen Body und mindestens ein Bild in PNG Format.

Entity

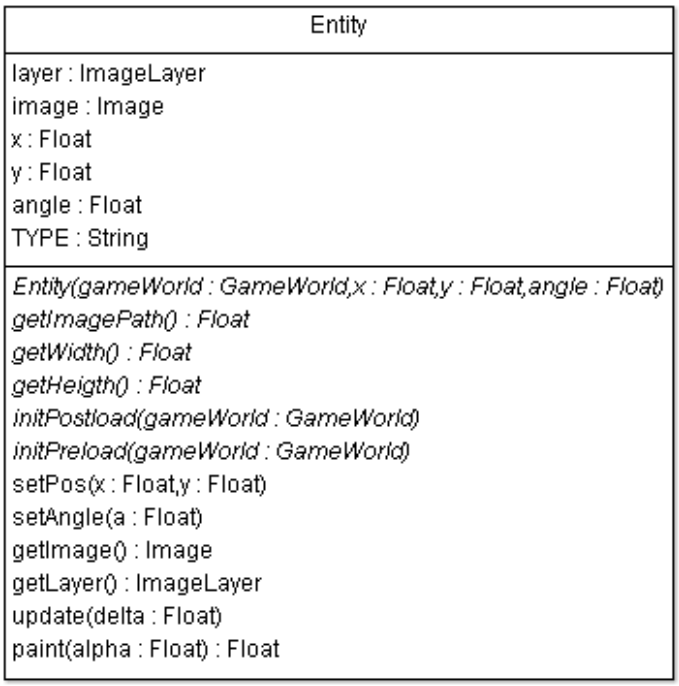

Abb. 4.20. Entity UML Klassendiagramm

Alle Klassen in net.saim.game.client.core.entities sind Unterklassen von *Entity*. Jeder Instanz dieser Klasse wird ein PNG-Bild zugeordnet, welches im ImageLayer eingefügt und gerendert wird. Zusätzlich besitzt es noch eine Koordinate für die Angabe seiner Position und Anordnung in der *GameWorld*. Der Konstruktor leitet die Objekt Initialisierung ein, dabei werden zum einen Position, Angle und *GameWorld* für das *Entity* gesetzt und zum anderen die graphischen Elemente mit dem PlayN AssetManager geladen. Die abstrakten Methoden *initPreLoad* und *initPostLoad* definieren dabei den Zeitpunkt für das Rendern der Bildressource. Jedem *Entity* steht eine *paint* und *update* Methode zur Verfügung. Zur Identifikation besitzen alle *Entities* eine zu ihrer Klasse gehörende *TYPE*, der *Loader* kann die *Entities* an diesem Attribut unterscheiden.

PhysicsEntity

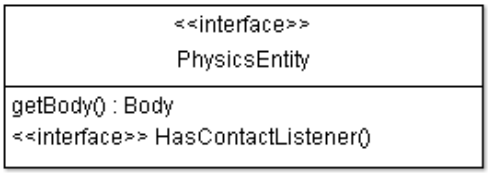

Abb. 4.21. PhysicsEntity UML Klassendiagramm

Dieses Interface definiert *getBody()* und das *HasContactListener* Interface mit der Methode *contact*, welche für die Kollisionsverarbeitung von *Portal*, *BlockSpring* oder *Enemy* mit anderen *Entities* genutzt wird. *PhysicsEntity* wird auf die beiden Klassen *StaticPhysicsEntity* und *DynamicPhysicsEntity* abgebildet.

DynamicPhysicsEntity

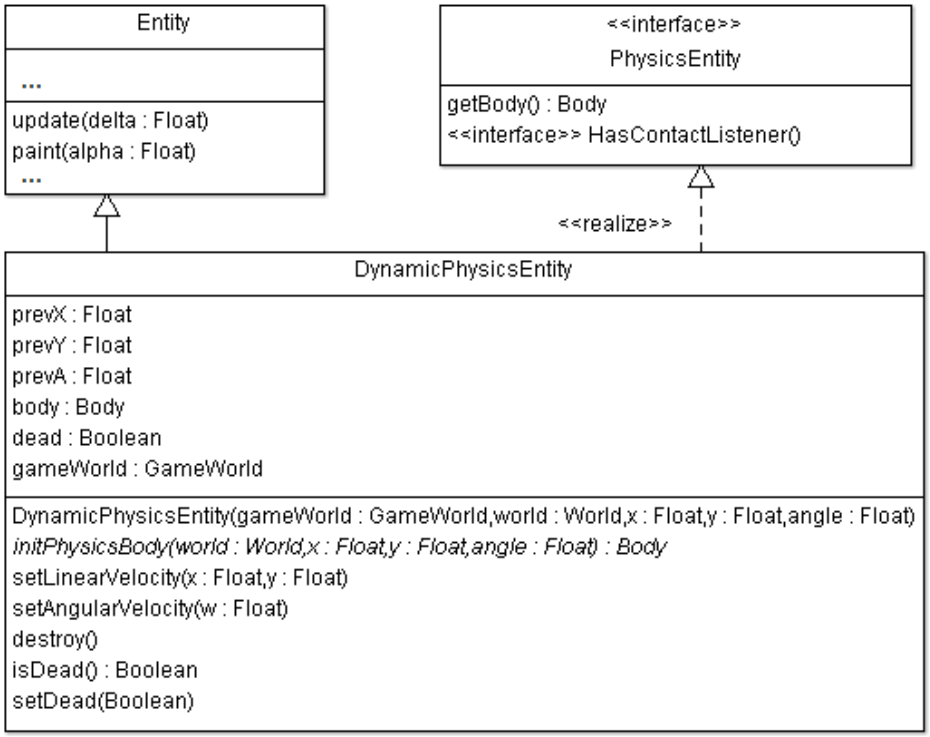

Abb. 4.22. DynamicPhysicsEntity UML Diagramm

Alle Beweglichen Körper der *GameWorld* sind Abbildungen von *DynamicPhysicsEntity* und werden in der *dynamicLaye*r der GameWorld aufbewahrt. Zu ihnen zählen die *Clouds (Cloud1* und *Cloud3)* und die *Sprites*. Die *Sprites* bilden die Hauptfiguren der *GameComponent* und besitzen im Spielgeschehen eine große Rolle. Sie werden durch die Methode *setAngluarVelocity* in Rotation versetzt und können sich so in der

GameWorld fortbewegen. Kollidieren beide Protagonisten miteinander, verarbeitet die Klasse *GameWorld* diesen Kontakt.

DynamicPhysicsEntity verwendet Interpolation[31] um die Animationen der *Entitites* so geschmeidig wie möglich darzustellen, dafür ist es notwendig die *paint* und die *update* Methode von der Oberklasse *Entity* zu überschrieben.
Beim Aufruf von *update* wird der Zustand der *Entity* für die Interpolation gespeichert und *setAngluarVelocity* auf die *Entity* angewendet. Zusätzlich überprüft die *isDead()* Methode ob diese *Entity* von einer anderen besiegt wurde, ist dies der Fall muss diese *Entity* mit *destroy()* zerstört und aus der *GameWorld* entfernt werden.

```
public void update(float delta) {
        if (this.isDead() == true)
                destroy();
        if(this istanceof Pea)
                this.setAngluarVelocity(2);
        if(this istanceof Enemy)
                this.setAngluarVelocity(-4);
        // Store state for interpolation
        prevX = body.getPosition().x;
        prevY = body.gtePosition().y;
        prevA = body.getAngle();
}
```

Quellcodeausschnitt DynamicPhysicsEntity update Methode

Die Interpolation wird am Ende in der *paint* Methode berechnet, basierend auf diesen Werten kann die Translation und Rotation auf den ImageLayer der *Entity* ausgeführt werden. Diese Methodik ermöglicht eine stimmige Darstellung der Bewegungen der Sprites.

```
public void paint(float alpha) {
        //  Interpolate based on previous state
        float x = (body.getPosition().x * alpha) + (prevX  * (1f – alpha));
        float x = (body.getPosition().y * alpha) + (prevY  * (1f – alpha));
        float x = (body.Angle() * alpha) + (prevA  * (1f – alpha));
        getLayer().setTranslation(x,y);
        getLayer().setRotation(a);
}
```

Quellcodeausschnitt DynamicPhysicsEntity paint Methode

[31] Siehe Kapitel 2.3.2. Abschnitt GameLoop

StaticPhysiscEntity

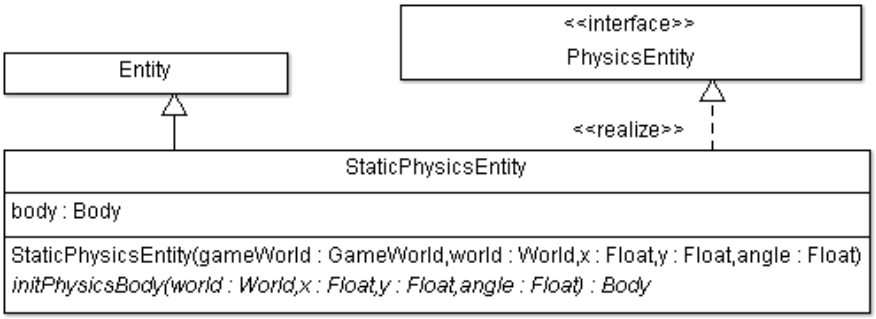

Abb. 4.23. StaticPhysicsEntity UML Klassendiagramm

Mit der *StaticPhysicsEntity* Klasse werden statische Körper in der *GameWorld* modelliert. Der Konstruktor ruft zum einen die *super* Methode auf, welche der Oberklasse die benötigten Parameter übergibt und zum anderen die abstrakte Methode *initPhysicsBody*, welche den physischen Körper/Body der *Entity* konstruiert. Diese haben eine fixe Position und werden in den Layer *StaticLayerBack* der *GameWorld* eingefügt. Unterklassen von *StaticPhysicsEntity* sind *Block, BlockGel, BlockLeftRamp, BlockRightRamp* und *Portal*.

Sprite

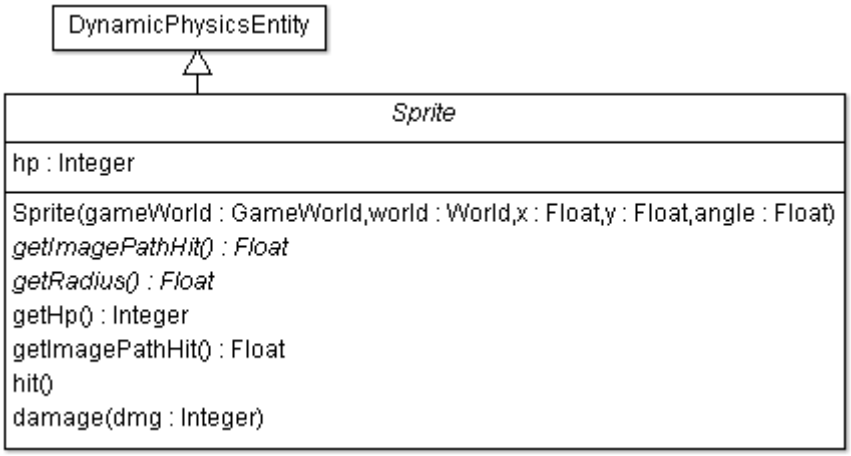

Abb. 4.24. Sprite UML Klassendiagramm

Die *Sprites* sind die „Hauptakteure“ des 2D Spiels, mit ihnen wird das Spielprinzip und das Spielgeschehen bestimmt. Zu den *Sprites* gehören die *Enemies* und *Peas* Klassen. *Peas* sind die einzigen *Entitites* die vom Spieler Weise gesteuert werden können.
Jedes *Sprite* besitzt eine gewisse Anzahl von *hp* (health points) und eine *hit* Methode, welche definiert wie sich ihre Attribute und Eigenschaften bei einer Kollision verändern. Im Moment wird einem Sprite bei dem Aufruf der *hit* Methode ein anderes Image zugeordnet.

Im Laufe des Spiels müssen sich *Peas* und *Enemies* „bekämpfen“, kollidieren beide Parteien miteinander werden ihnen mit der *damage* Methode *hp* abgezogen. Unterschreitet die *hp* eines Sprites den Wert null, so wird die *setDead* Methode der Oberklasse *DynamicPhysicsEntity* aufgerufen, welche das *Sprite* daraufhin aus der *GameWorld* entfernt.

Clouds (Cloud1 und Cloud3)

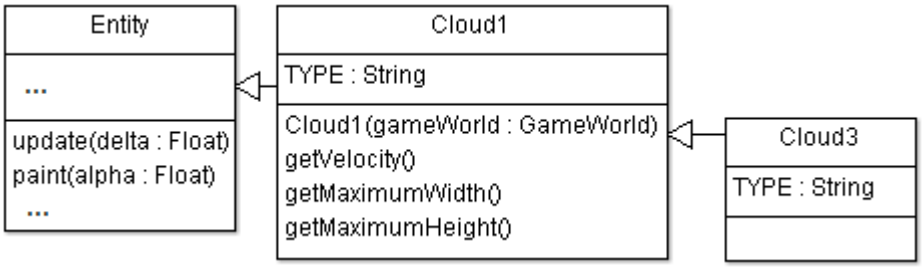

Abb. 4.25. Cloud1 und Cloud3 UML Klassendiagramm

Clouds modellieren die Wolken in der *GameWorld*, diese bewegen sich mit einer konstanten Geschwindigkeit von einem Spielfeldrand zum anderen, wird der rechte Rand der *GameWorld* von einer Wolke überquert veranlasst die *update* Methode eine Neugenerierung der x und y Koordinaten der *Clouds*, so dass sie wieder in den linken Rand der *GameWorld* eintreten.

Block

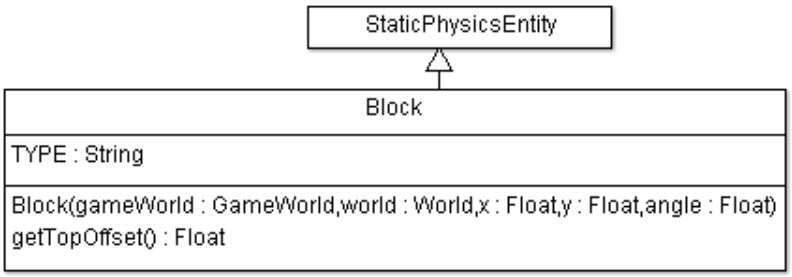

Abb. 4.26. Block UML Diagramm

Blocks sind statische Objekte. Sie bilden zusammen die Plattform der Spielewelt, auf der sich die Spielfiguren bewegen. Es gibt vier verschiedene Unterklassen (*BlockGel*, *BlockLeftramp*, *BlockRightRamp* und *FakeBlock*) von Blocks die sich lediglich in ihrem Aussehen und physischen Eigenschaften (Reibung, Elastizität) unterscheiden.

BlockSpring

BlockSpring modelliert ein Sprungbrett und gehört ebenfalls wie die Blocks zur Landschaft der Spielewelt, doch ist dieses Objekt nicht statisch. Um das Aussehen und

die Wirkung eines Sprungbretts zu simulieren wird der Body von *BlockSpring*, welcher eine hohe Elastizität besitzt, durch org.jbox2d.dynamics.joints.MousJoint mit einem zweiten Body verbunden. Der zweite Body ist in diesem Fall der statische „ground“ Body (sozusagen die Plattform/Landschaft). Bei einer Kollision zwischen einem *Entity* und *BlockSpring* werden die Kräfte und die resultierenden Positionen des *BlockSprings* berechnet, dabei bindet der MouseJoint den *BlockSpring* wie eine „elastische Fessel“ an den „ground“ Body.

Portal

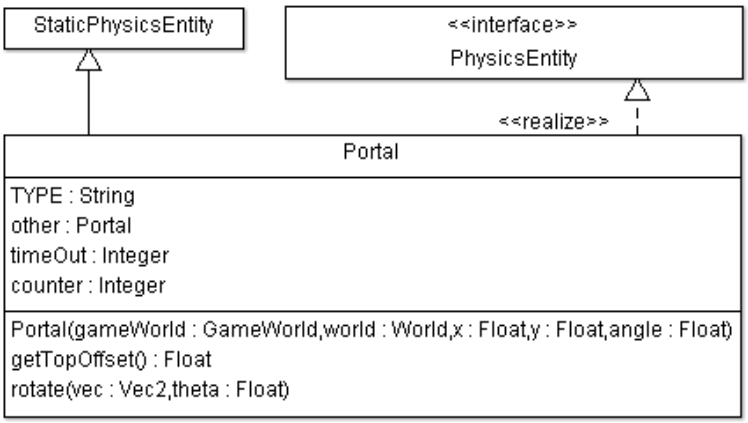

Abb. 4.27. Portal UML Klassendiagramm

Die Klasse *Portal* implementiert das PhysicsEntity.HasContactListener Interface und erbt von der Klasse *StaticPhysicsEntity*. Ein *Portal* ermöglicht das „Teleportieren“ von *Entities.* In der *GameWorld* wird jedem *Portal* genau ein anderes *Portal* als Partner zugeteilt. Kollidiert ein *Entity* mit einem *Portal* werden seine Positions-Informationen in der *contact* Methode verändert. Dabei führt die Methode *rotate* eine Translation auf das *Entity* aus, wodurch dieses zum Partner *Portal* teleportiert wird. Mit den Attributen *timeOut* und *counter* wird ein Timer Mechanismus in der *update* und *contact* Methode umgesetzt, welcher Teleportationen erst nach Ablauf eines Time Outs ermöglicht.

5. Verwandte Arbeiten

In diesem Kapitel werden einige Spiele vorgestellt, welche ebenso wie Veri-Links nach dem GWAP Paradigma entwickelt wurden.

Image Annotation

ESP Game:
Eins der ersten GWAPs ist das Spiel ESP Game, welches zum Taggen von Bildern genutzt wird. In diesem Spiel sollen 2 Spieler versuchen einem Bild das gleiche Tag zuzuordnen. Das Spiel speichert die übereinstimmenden Ergebnisse als Image Labels, welche im Nachhinein für eine verbesserte Bildersuche im Netz verwendet werden können. Dieses Spiel ist temporeich, unterhaltsam und kompetitiv; seit Juli 2008 haben über 200.000 Spieler, mehr als 50 Millionen Labels beigesteuert. [2]

Video Alignment

PopVideo:
PopVideo ist ein Beispiel für ein zwei Personen Onlinespiel, welches das Ziel hat Videos zu annotieren. Beiden Spieler wird zur gleichen Zeit das gleiche Video abgespielt, dabei müssen sie eingeben was sie sehen oder hören. Für übereinstimmende Antworten werden sie mit Punkten belohnt. [31]

Ontology Alignment

SpotTheLink:
SpotTheLink ist ein 2 Spieler Quizgame konzipiert für Ontology Alignments. Das Ziel dieses Spiels ist es Concepts aus 2 Ontologien miteinander zu verbinden. Im ersten Schritt müssen sich beide Spieler auf ein übereinstimmendes Concept für beide Ontologien einigen und im zweiten Schritt sollen beide Spieler eine gemeinsame Beziehung für die Konzepte vereinbaren. Das Spiel endet wenn Spielzeit abgelaufen ist oder einer der beiden Spieler das Spiel verlässt.[32][33]

6. Fazit

Ziel dieser Arbeit war die Entwicklung eines Spiels zur Verifikation von Semantic Web Links. Zu diesem Zwecke wurde das Browser Spiel Veri-Links entwickelt, welches die geforderte Funktionalität erfüllt. Es ermöglicht einem Spieler auf eine unterhaltsame Weise Links im Semantic Web zu überprüfen und die Ergebnisse dieser Überprüfung in einer permanenten Datenbank für eine spätere Evaluation zu speichern.
Bei der Entwicklung dieses Spiels stellte sich jedoch heraus, dass nicht alle Semantic Web Links geeignet für diese Art der Verifikation sind. Oftmals ist es nicht möglich einen Link verständlich für einen Menschen darzustellen, da viele Links von Natur aus nur unanschauliche Informationen modellieren.
Ein weiteres Problem ist die Differenzierung zwischen zuverlässigen und unzuverlässigen Verifikation eines Spielers. Es wurden zwar Mechanismen implementiert, die einige mutwillig oder nur ausversehen falsch getätigte Verifikationen entgegenwirken und abfangen können, doch lassen sich Falschaussagen bzw. nicht korrekte Verifikationen niemals ganz vermeiden.
Diese unzuverlässigen Verifikationen würden sich vor allem im Anfangsstadium (sprich: einige Zeit nach dem Release dieses Spiels) von Veri-Links negativ auf die Qualität der Verifikationen auswirken, da Veri-Links zu Beginn nur relativ wenig Input zur Verfügung steht. Dieser Faktor sollte jedoch mit zunehmender Zeit und steigender Spieleranzahl weniger Einfluss auf die Qualität der Verifikationen haben.
Insgesamt kann jedoch gesagt werden, dass Veri-Links auf lange Sicht hochwertige Aussagen über die Korrektheit von verschiedenen Links liefern kann - wenn das Spielprinzip von Veri-Links und die gute Absicht dieses Spiels - die Verbesserung des Semantic Webs - in der Lage sind die Menschen an sich zu binden und zum Spielen zu motivieren.

7. Zukünftige Arbeiten

Das Spiel Veri-Links funktioniert und erfüllt die gestellten Produktanforderungen. In naher Zukunft ist eins der Ziele das Testen des Unterhaltungswertes dieses Spieles. Es muss herausgefunden werden, ob dieses Spielprinzip in der Praxis greift und Menschen gewillt sind ihre Zeit in das Spiel zu investieren. Dabei wäre das Sammeln von Meinungen und Bewertungen der Spieler zum Spiel hilfreich. Mittelfristig sollten die Verifikationen der Spieler ausgewertet und auf Zuverlässigkeit geprüft werden. Basierend auf den gewonnenen Ergebnissen werden eventuelle Veränderungen und Verbesserungen an dem Spiel vorgenommen.
Einige Funktionalitäten die in Zukunft zusätzlich implementiert werden könnten wären zum Beispiel die Realisierung eines zwei Spieler Modus zur Erhöhung des Spielspaßes. Das 2D Spiel könnte abwechslungsreicher gestaltet werden, indem die Anzahl der spielbaren Levels und Gegner erhöht oder die GUI verbessert wird. Es besteht die Möglichkeit einer Erweiterung der VerifyComponent zur Darstellung und Verifizierung von Instanzen aus Ontologien, deren Property Werte von Menschen nur schwer zu verstehen sind. Ein Beispiel wären Instanzen aus der LinkedGeoData Ontologie, diese Instanzen könnten anhand ihrer geographischen Koordinaten in einer Landkarte dargestellt werden.

Literaturverzeichnis

[1] Von Anh L. Games with a Purpose. [Internet]. 2006 [cited 2011 Sep 01]. Available from: http://www.cs.cmu.edu/~biglou/ieee-gwap.pdf.

[2] Van Anh L, Dabbish L. Designing Games With A Purpose. [Internet]. 2008 [cited 2011 Sep 01]. Available from: http://www.cs.cmu.edu/~biglou/GWAP_CACM.pdf.

[3] Unternehmen Region - Bundesministerium für Bildung und Forschung. [Internet]. 2010 [cited 2011 Sep 05]. Available from: http://www.unternehmen-region.de/de/4721.php.

[4] Berners-Lee T, Hendler J, Lassila O. Scientific America : The Semantic Web. [Internet]. 2001 [cited 2011 Sep 04]. Available from: http://www.ds3web.it/miscellanea/the_semantic_web.pdf.

[5] Pascal H, Krötzsch M, Rudolph S, Sure Y. In: Semantic Web. 1st ed. Springer; 2008. p. 12.

[6] Horrocks I, Parsia B, Patel-Schneider P, Hendler J. Semantic Web Architecture: Stack or Two Towers? [Internet]. 2005 [cited 2011 Sep 09]. Available from: http://www.cs.ox.ac.uk/people/ian.horrocks/Publications/download/2005/HPPH05.pdf.

[7] Luczak-Rösch M. AG Netzbasierte Informationssysteme. [Internet]. 2011 [cited 2011 Sep 10]. Available from: http://blog.ag-nbi.de/wp-content/uploads/2011/04/09_SemWeb2011.pdf.

[8] Auer S, Tramp S, Martin M. AKSW. [Internet]. 2010 [cited 2011 Aug 9]. Available from: http://aksw.org/Events/2010/LeipzigerSemanticWebDay/files?get=tutorial.pdf.

[9] Lehmann J. In: Learning OWL Class Expressions. AKA-Verlag; 2010. p. 11.

[10] Ternes N. Wissensmanagement und die Chancen von Web 3.0. Norderstedt: GRIN Verlag; 2010.

[11] Sikos L. The Resource Description Framework. [Internet]. 2011 [cited 2011 Sep 11]. Available from: http://www.lesliesikos.com/tutorials/rdf/.

[12] McCarthy P. Search RDF data with SPARQL. [Internet]. 2005 [cited 2011 Sep 10]. Available from: http://www.ibm.com/developerworks/xml/library/j-sparql/.

[13] DBpedia - Interlinking DBpedia with other Data Sets. [Internet]. 2011 [cited 2011 Sep 01]. Available from: http://wiki.dbpedia.org/Interlinking?v=13gd.

[14] Bizer C, Heath T, Berners-Lee T. Linked Data - The Story So Far. [Internet]. 2009 [cited 2011 Sep 11]. Available from: http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf.

[15] Geisler M. In: Semantic Web. entwickler.press; 2009. p. 83.

[16] Linked Open Data. [Internet]. 2011 [cited 2011 Sep 11]. Available from: http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData.

[17] The Linking Open Data cloud diagram. [Internet]. 2010 [cited 2011 Sep 11]. Available from: http://richard.cyganiak.de/2007/10/lod/.

[18] LATC. [Internet]. [cited 2011 Sep 11]. Available from: http://latc-project.eu/.

[19] Lehmann J. SAIM (Semi-Automatic Instance Matcher). [Internet]. 2011 [cited 2011 Sep 12]. Available from: http://aksw.org/Projects/SAIM.

[20] Google Web Toolkit. [Internet]. 2011 [cited 2011 Aug 20]. Available from: http://code.google.com/webtoolkit/overview.html.

[21] GWT - Communicate with a Server. [Internet]. 2011 [cited 2011 Aug 20]. Available from: http://code.google.com/webtoolkit/doc/latest/DevGuideServerCommunication.html#DevGuideGettingUsedToAsyncCalls.

[22] PlayN. [Internet]. 2011 [cited 2011 Aug 22]. Available from: http://code.google.com/p/playn/.

[23] Cromwell R, Philip R. Kick-ass Game Programming with Google Web Toolkit. [Internet]. 2011 [cited 2011 Aug 25]. Available from: http://static.googleusercontent.com/external_content/untrusted_dlcp/www.google.com/en//events/io/2011/static/notesfiles/Kick-assGameProgrammingwithGoogleWebToolkit.pdf.

[24] PlayN - GraphicsArchitecture. [Internet]. 2011 [cited 2011 Sep 03]. Available

from: http://code.google.com/p/playn/wiki/GraphicsArchitecture.

[25] PlayN - GameLoop. [Internet]. 2011 [cited 2011 Sep 03]. Available from: http://code.google.com/p/playn/wiki/GameLoop.

[26] Catto E. Box2D v2.2.0 User Manual. [Internet]. 2011 [cited 2011 Sep 04]. Available from: http://www.box2d.org/manual.html.

[27] Jena – A Semantic Web Framework for Java. [Internet]. [cited 2011 Sep 01]. Available from: http://jena.sourceforge.net/.

[28] Isele R, Jentzsch A, Bizer C, Volz J. Silk - A Link Discovery Framework for the Web of Data. [Internet]. 2011 [cited 2011 Sep 06]. Available from: http://www4.wiwiss.fu-berlin.de/bizer/silk/.

[29] Isele R, Jentzsch A, Bizer C, Volz J. Silk Link Discovery Framework. [Internet]. 2011 [cited 2011 Sep 06]. Available from: http://www.assembla.com/spaces/silk/wiki/dg7jfup58r4jZseJe5cbLA.

[30] Box2D C++ tutorials - Removing bodies safely. [Internet]. 2011 [cited 2011 Sep 06]. Available from: http://www.iforce2d.net/b2dtut/removing-bodies.

[31] GWAP - Popvideo. [Internet]. 2011 [cited 2011 Sep 13]. Available from: http://www.gwap.com/gwap/gamesPreview/popvideo/.

[32] insemitives - Games For Semantic Content Creation. [Internet]. 2011 [cited 2011 Sep 14]. Available from: http://www.insemtives.eu/games.php.

[33] Thaler S. SpotTheLink - Userguide. [Internet]. 2010 [cited 2011 Sep 13]. Available from: http://www.insemtives.eu/tools/SpotTheLink-userguide.pdf.

[34] Sowa H, Radinger W, Marinschek M. Google Web Toolkit - Ajax Anwendungen einfach und schnell entwickeln. dpunkt.verlag; 2009.

[35] Passin T. Explorer's Guide To The Semantic Web. Manning; 2004.

[36] Hanson R, Tacy A. In: GWT In Action. Manning; 2007.

Anlagenverzeichnis

Anlagen

Anlage 1: Installationshinweise

Kompatibilität (Getestete Browser):

Veri-Links ist mit folgenden Browsern kompatibel:

- Mozilla Firefox 5.x, 6.x
- Google Chrome

Veri-Links ist mit folgenden Browsern inkompatibel:

- Opera 11.51
- Microsoft Internet Explorer 9

Veri-Links Installation:

Bei der Kompilierung von Veri-Links entsteht ein WAR Datei. Diese kann auf einen Web-Server deployed werden.
Bei einem Apache Tomcat Server kann der Deploy auf folgende Weise realisiert werden:

1. Web-Browser öffnen und den Tomcat Web Application Manager ansteuern.
2. Zu der Sektion "WAR file to deploy" scrollen.
3. Auf den "Browse..." Button klicken und die WAR Datei auswählen.
4. "Deploy" klicken. Daraufhin wird die WAR Datei auf den Server hochgeladen und von Tomcat deployed.

Veri-Links benötigt eine mySQL Datenbank. Im Folgenden wird beschrieben wie diese Datenbank erstellt wird und wie Links aus einer Link Datei in diese Datenbank eingefügt werden. Zur Demonstration befinden sich im „LinkFiles“ Ordner zwei Link Dateien im NTriple Format, die benutzt werden können. Diese wurden aus den Link Spezifikations Dateien (welche sich ebenfalls in dem Ordner befinden), mit dem Silk Framework erstellt. Im Quellcode befinden sich Passagen die manuell an die individuellen Bedürfnisse und Systemvoraussetzungen angepasst werden müssen, diese wurden mit dem Hinweis „// <--- Change this“ kommentiert. Nun zur Anleitung:

1. Zuerst müssen einige Einträge in der **RDFHandler.java** Datei angepasst werden.

2. In der **main** Methode den Wert der Variable **path** umändern, in den Pfad der Link Datei. Dadurch wird dem RDFHandler mitgeteilt wo er nach der Link Datei suchen soll.

3. In der **createDatabase()** Methode die **Parameter** (serverName, databaseName, userName, password) des **Konstruktoraufrufes von DBTool** umändern, in die benötigten Verbindungsdaten für die Datenbank. Die Datenbank mit dem Namen „veri_links“ wird durch diese Methode erstellt.

4. In der **start** Methode die benötigten **Templates setzen**. Wurde zum Beispiel die "factbooks_links_verify.nt“ Datei als Link Datei in der „main“ Methode definiert, müssen die Templates „TemplateDbpedia“ als Subjekt und „TemplateFactbook“ als Objekt gesetzt werden. Welches Template dabei Subjekt oder Objekt ist, richtet sich nach der Link Datei und der Link Spezifikations Datei.

5. In der **start** Methode richtige **read** Methode definieren. Für NTriple Dateien readNT und für XML/Alignment Dateien readXML.

6. In der **start** Methode die benötigten **Verbindungsdaten** für den **Konstruktoraufruf von DBTool** definieren.

7. MySQL Server starten.

8. RDFHandler.java ausführen.

Anlage 2: Spielablauf

1. Web-Browser öffnen und Veri-Links ansteuern.
2. Namen eingeben und verlinkte Ontologien auswählen. Danach werden nur noch Links aus diesen Ontologien zur Verifikation freigegeben.
3. Spiel mit der Escape Taste starten.
4. Spieler hat die Möglichkeit a. eine Link Verifikation durchzuführen oder b. das 2D Spiel zu bedienen
    a. Die oberste und unterste Tabelle stellen Instanzen aus den ausgewählten Ontologien dar. Der Spieler soll entscheiden ob die Beziehung, welche zwischen den Tabellen in grüner Schrift dargestellt wird, eine korrekte Verbindung zwischen den beiden Instanzen definiert. Indem er eine der drei Optionen (Valid, Not Valid, Not Sure) auswählt und den „send verification“ Button betätigt. Danach wird der Geldbetrag des Spielers aktualisiert.
    b. Der Spieler versucht durch geschicktes „Platzieren/Abwerfen“ seiner Spielfiguren den gegnerischen Angriff abzuwehren. Mit der rechten und linken Pfeiltaste wählt der Spieler die Abwurfstelle seiner Spielfiguren aus. Die Leertaste fügt die Spielfigur an die entsprechende Position in die Spielwelt ein. Das Spiel kann mit der Escape Taste pausiert und wieder fortgesetzt werden.
5. Das Spiel ist beendet wenn einer der gegnerischen Figuren den linken Rand der Spielewelt erreicht hat oder der Spieler die gesamten Level erfolgreich gemeistert hat.
6. Beim Spielende bekommt der Spieler die Möglichkeit sich in die Veri-Links Highscore Liste einzutragen.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, den..... .....
(Unterschrift)