Freie Universität Berlin

Bachelorarbeit
im Fach Informatik
an der Freien Universität Berlin

Erstellung eines Echtzeit Level-Editors für die Nebula3-Engine

eingereicht bei
Prof. Dr. Lutz Prechelt
und
Prof. Dr.-Ing. Jochen Schiller
vorgelegt von Susanne Schulz
Matrikelnummer: 3599971
betreut von
Dipl. Inf. Nicolai Pogoczelski

Berlin, November 2010

Zusammenfassung

In dieser Arbeit wird die Erstellung und der Aufbau eines in C# geschriebenen Level-Editors beschrieben. Dies beinhaltet das Einbetten eines Nebula-Fensters, in dem man 3D-Objekte in Echtzeit auf einem virtuellen Gitter platzieren kann. Die Struktur des Editors soll für verschiedene Projekte und Werkzeuge erweiterbar sein.

Der Level-Editor besteht im Wesentlichen aus zwei Teilen. Der erste Teil ist das Rahmenwerk, geschrieben in C#, in dem das Nebula-Fenster eingebettet ist und welches die Daten verwaltet. Der zweite Teil ist eine in C++ geschriebene, erweiternde Nebula3 Bibliothek, mit deren Hilfe die Echtzeitpositionierung ermöglicht wird.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 10. November 2010

Susanne Schulz

Danksagung

Hiermit möchte ich mich bei allen bedanken, die mich bei meiner Bachelorarbeit unterstützt haben. Insbesondere aber bei Nicolai Pogoczelski, Michael und Andreas Prietzel, Kathrin und André Ebert, Lars Kokemohr, André „Floh“ Weißflog, Robin Henkys, Florian Schambach, Uwe Mehner, Claudia Krüger und allen anderen Testern von Level-Editor.

Inhalt

ZUSAMMENFASSUNG **2**

EIDESSTATTLICHE ERKLÄRUNG **3**

DANKSAGUNG **4**

INHALT **5**

1 EINLEITUNG **7**

1.1 DIE SPIEL-ENGINE 7

2 GRÜNDE FÜR DIE ENTWICKLUNG DES LEVEL-EDITORS **9**

2.1 MAYA ALS LEVEL-EDITOR 10

2.1.1 Datenstruktur und Darstellung *10*

2.1.2 Die Transformations-Funktionen *11*

2.1.3 Die „Nebula2 Toolkit for Maya"-Funktionen *11*

2.1.4 Gründe Maya als Level-Editor zu ersetzen *13*

2.3 DER „TORCHLIGHT-EDITOR - TORCHED" 13

2.4 ANFORDERUNGEN AN DEN LEVEL-EDITOR 15

3 NEBULA3 ENGINE **18**

3.1 DIE SCHICHTEN DER NEBULA3 ENGINE 18

3.2 GRUNDLEGENDE KONZEPTE IN DER APPLIKATIONSSCHICHT 19

3.3 DAS LEVEL-EDITOR-ADD-ON IN NEBULA3 20

3.4 ECHTZEIT PLATZIERUNG IN NEBULA3 – DAS PLACEMENTUTIL 21

3.4.1 Erweiterungen des PlacementUtil für den Level-Editor *22*

3.5 KOMMUNIKATION MIT DEM LEVEL-EDITOR – DIE REMOTE-KLASSEN VON NEBULA3 24

3.5.1 Erweiterung der Remote-Klassen für den Level-Editor *25*

3.6 KAMERA STEUERUNG 26

3.6.1 Implementierung der Level-Editor-Kamera *26*

4 DIE NEBULA3TOOL BIBLIOTHEK **28**

4.1 DIE REMOTE-KLASSEN 28

4.2 DER COMMANDMANAGER 28

4.3 DAS NEBULA3VIEWCONTROL 29

5 LEVEL-EDITOR PLUG-IN STRUKTUR IN C# **30**

5.1 PROGRAMMAUFBAU DER C# APPLIKATION 30

5.2 CONTAINER UND PLUG-INS 31

5.3 PLUG-IN SCHNITTSTELLEN 32

5.3.1 Die Datenkern Schnittstelle *33*
5.3.2 Die Schnittstelle für das Nebula-Fenster *33*
5.3.3 Die Outliner-Schnittstelle *33*
5.3.4 Die Tool-Schnittstelle *34*
5.3.5 Die Schnittstelle zum Hauptprogramm *34*
5.4 DAS LADEN DER PLUG-INS 34
5.5 STANDARD-PLUG-INS 34
5.5.1 ViewerDefault *35*
5.5.2 OutlinerDefault *35*
5.5.3 Grafik Browser und Spielobjekt Browser *37*
5.6 VERWALTUNG DER DATEN – DIE LEVELDATA-KLASSE 37
5.6.1 Attributschnittstelle - ILevelNodeAttr *38*
5.6.2 ILevelNode *38*
5.6.3 LevelData Methoden *38*
5.6.4 LevelData und CommandManager *39*
5.7 OFFENE PROBLEME 39
6 TESTS UND EVALUATION 41
7 FAZIT 44
8 AUSBLICK 45
9 LITERATURVERZEICHNIS 46
10 TABELLEN-UND ABBILDUNGSVERZEICHNIS 48
ANHANG 49
Level-Daten Format *49*
Ausschnitt aus der Bug-Datenbank *50*

1 Einleitung

Ziel dieser Arbeit ist es die Entwicklung eines Programmes zu präsentieren, das in der Lage ist, Inhalte für bestimmte Arten von Computerspielen zu generieren. Dieses Programm wird als Level-Editor[1] bezeichnet. Es wurde für die Spiel-Engine[2] Nebula3 entwickelt.

Um der Frage nach zu gehen, was die Nebula3-Engine und der Level-Editor genau sind, ist es notwendig sich mit den Grundlagen der Entwicklung von Computerspielen zu beschäftigen.

1.1 Die Spiel-Engine

Bei der Entwicklung eines neuen Computerspiels wird in der Regel nicht von Grund auf programmiert, sondern es wird auf einer existierenden Spiel-Engine aufgesetzt. Eine solche Spiel-Engine bietet ein Rahmenwerk, welches den Programmierern erlaubt sich hauptsächlich auf die Entwicklung der Spiel-Logik zu konzentrieren. Das Rahmenwerk kapselt einen Großteil der Funktionalitäten, die ein Spiel braucht. So bringt die Engine beispielsweise bereits Schnittstellen zu dem Betriebssystem, der 3D[3]-Darstellung, den Datenbanken und dem Netzwerk mit.

Zu einem Spiel gehören meistens größere Mengen an Daten. Das können Bilder, 3D-Modelle, XML[4]-Dateien oder andere Daten sein. Deswegen gehören zu einer Engine meist auch Applikationen, die es ermöglichen diese Daten zu erstellen. Das können beispielsweise Programme sein, die grafische Daten in ein spezielles Format überführen, die den Aufbau von Datenbanken ermöglichen oder die helfen das Spiel in andere Sprachen zu übersetzen.

Viele Entwickler, die an der Entstehung eines großen Spiels beteiligt sind, haben wenig oder gar keine Programmiererfahrung. Es sind Animatoren, Autoren, Designer, Grafiker, Projektmanager und Tester. Sie benutzen die bereitgestellten Werkzeug-Applikationen um die jeweiligen Inhalte des Spiels zu schaffen. Der Level-Editor gehört zu diesen Werkzeugen.

Die Entwicklung einer Engine ist ein langwieriger Prozess. Um auf dem freien Markt bestand zu haben, muss eine Engine sich ständig neuen Hardware-Anforderungen und Aufgaben anpassen. Die Open-Source-Engine Nebula3 wird von der Firma Bigpoint GmbH weiterentwickelt. Die

[1] Level, englisch für Ebene, hier verwendet als Bezeichnung für einen Abschnitt einer Spielwelt.
[2] Engine, englisch für Motor oder Antrieb
[3] 3D ist eine Abkürzung für dreidimensional und wird als Synonym für Räumlichkeit gebraucht.
[4] Extensible Markup Language, englisch für erweiterbare Auszeichnungssprache

Nebula3-Engine ist die dritte Generation der Nebula-Engine. Diese Weiterentwicklung schließt mit ein, dass sich auch die mitgelieferten Werkzeuge weiterentwickeln. Die Entwicklung des Level-Editors stellt einen weiteren Schritt in der Entwicklung der Werkzeuge für die Nebula-Engine dar.

2 Gründe für die Entwicklung des Level-Editors

Die Nebula3-Engine deckt eine große Spanne des Computerspielgenres ab. Ein besonderes Augenmerk wird auf die Entwicklung von Computer-Rollenspielen gelegt, in denen sich der Spieler in der Rolle eines Avatars[5] durch eine 3D Welt bewegt. Die damit erstellten Spiele konnten darauf ausgelegt werden, offline alleine oder online mit einer kleinen Gruppe gespielt zu werden.

Mit Hinblick auf das wachsende Genre der MMO-Spiele[6], hat sich auch der Focus bei der Weiterentwicklung der Nebula3-Engine in diese Richtung verschoben. Die zukünftigen Spielwelten sollen auf einem Server laufen, zu dem sich die Spieler mit einem Nebula3-Spiel-Client verbinden. Dort sollen sie allein oder mit anderen Spielern in einer 3D-Welt Abenteuer erleben. Die Spieler lassen eine solche Welt lebendig werden. Deshalb ist eines der Ziele eine Gemeinschaft aufzubauen, die dieses Spiel über längere Zeit spielt. Um eine solche Gemeinschaft am Leben zu erhalten, ist es aber notwendig in relativ kurzen Abständen neue Spielinhalte nach zuliefern. Daraus ergibt sich die Notwendigkeit Inhalte schnell und in gewohnter Qualität zu erstellen. Die bisherigen Werkzeuge, die Nebula3 zur Erstellung von Spielinhalten anbot, ermöglichen zwar einen hohen Qualitätsstandard, benötigen aber auch eine relativ lange Zeitspanne bis zur Fertigstellung.

Bisher wurde die sehr vielseitige 3D-Modelierungssoftware Maya [Aut09] in Verbindung mit einem Nebula-Plug-in[7] genutzt. 3D-Modelierung und Animation erfolgten in Maya und das soll so auch fortgeführt werden. Weiterhin wurde bisher aber auch die Spielwelt, in Form von Abschnitten, sogenannte Level, in Maya zusammengestellt und mit Spiellogik versehen. Um die Anfertigung der Level zu beschleunigen und zu vereinfachen, sollen sie nun in einem speziell dafür gefertigten Editor erstellt werden.

In welcher Weise Maya zum Editieren der Level benutzt wurde und welche Funktionen für den Gebrauch als Level-Editor relevant sind, soll im folgenden Abschnitt erläutert werden. Als Referenz zu anderen Editoren für Spiele, erfolgt ein Vergleich des „Torchlight“-Editors

[5] Als Avatar wird eine künstliche Person oder der Stellvertreter einer Person bezeichnet.
[6] Ein MMO (Massiv Multiplayer Online) Spiel, ist ein Computerspiel in dem über das Internet mehrere tausend Spieler gleichzeitig eine persistente virtuelle Welt bevölkern.
[7] Ein Plug-in ist ein Software-Modul, das sich in vorhandene Software einklinkt und dessen Funktionalität damit erweitert.

(TorchED) [Run09] mit Maya. Die daraus resultierenden Anforderungen für einen Nebula3 Level-Editor werden abschließend vorgestellt.

2.1 Maya als Level-Editor

Dieses Kapitel geht auf die Maya-Funktionen ein, die im Rahmen der Funktionalität als Level-Editor, benutzt wurden. Abschließend werden Probleme beschrieben, die bei der Arbeit mit Maya als Level-Editor auftreten.

2.1.1 Datenstruktur und Darstellung

Maya speichert Daten in einer hierarchischen Baumstruktur. Jeder Knoten in der Baumstruktur hat Attribute, die seine Funktion definieren.

Es gibt zwei wesentliche Benutzeroberflächenelemente, die benutzt werden um diese Struktur darzustellen und zu bearbeiten. Das erste Element ist eine dem Windows-Explorer[8]-ähnliche Baumstruktur, der sogenannte Outliner[9]. Im zweiten Bereich findet sich eine 3D-Darstellung, in der Knoten, die eine grafische Repräsentation besitzen, dargestellt werden.

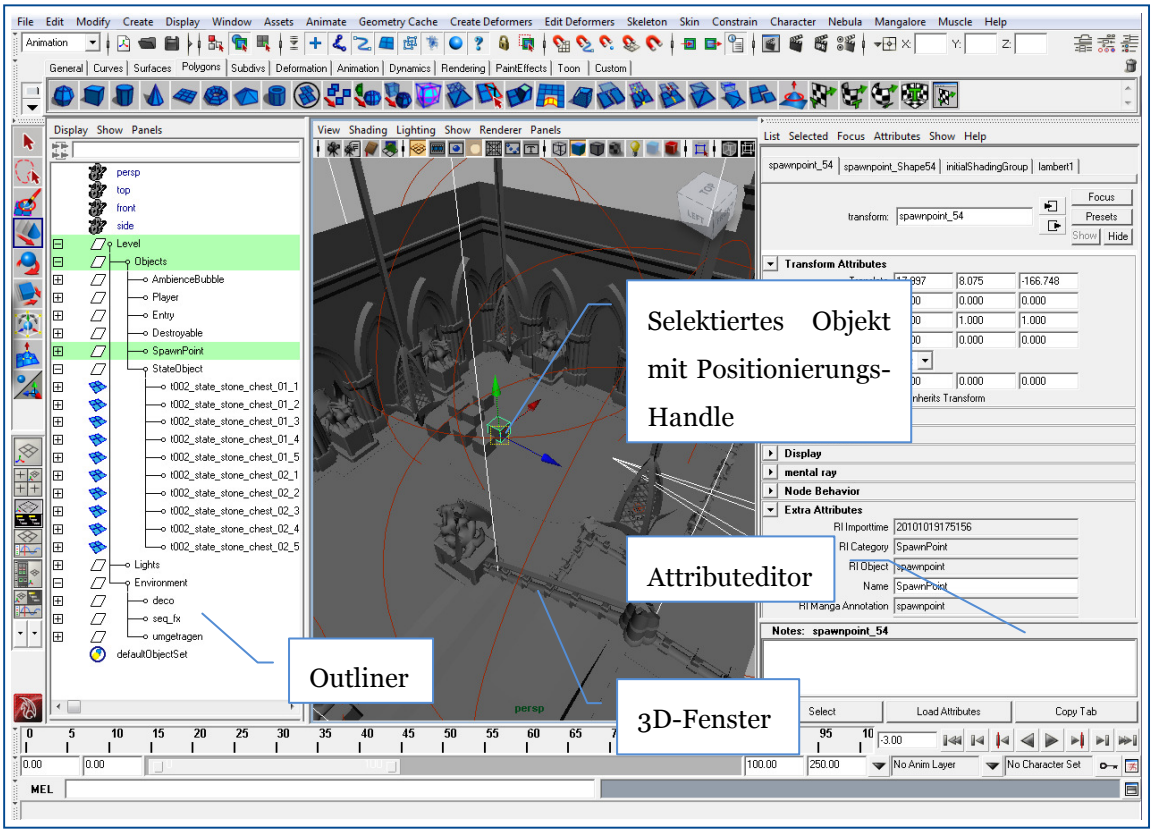


Abbildung 1: Autodesk Maya 2009 und die Zuordnung der Benutzeroberflächenelemente

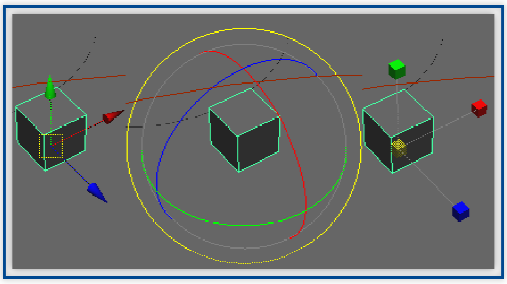
[8] Windows-Explorer ist der Standard Dateimanager des Betriebssystems Microsoft Windows
[9] Outliner, abgeleitet vom englischen „outline“ für Kontur, Entwurf aber auch Inhaltsverzeichnis.

Um ein reines Grafik-Level ohne Spielobjekte zu erstellen, werden 3D-Objekte aus anderen Maya-Dateien geladen. Diese werden dann in einer 3DUmgebung platziert. Alternativ können für die eigentlichen Objekte auch Platzhalter eingesetzt werden. Diese Darstellung kann auf die Bedürfnisse der Benutzer angepasst werden. Ein Objekt kann in 3D oder in einer Gitterstruktur dargestellt werden. Zum Bearbeiten der Objekte stehen, je nach Art des Attributes, diverse Editoren zur Verfügung.

Um die Übersicht zu erhöhen und Fehler zu verringern, gibt es die Möglichkeit Knoten über ihre Attribute unsichtbar zu machen oder gegen Bearbeitung zu sperren. Es besteht außerdem die Möglichkeit Objekte sogenannten „Layern“ zuzuweisen. Das sind Bearbeitungsebenen, die Objekte zu Gruppen zusammenfassen.

2.1.2 Die Transformations-Funktionen

Die Platzierung von Objekten in Maya erfolgt durch das „Move Tool“. Die Ausrichtung wird über das „Rotate Tool“ bestimmt. Außerdem gibt es das „Scale Tool“, um die Größe eines Objektes zu ändern, das wird allerdings nur für bestimmte Objekttypen eingesetzt, wie zum Beispiel Lichtquellen. Es gibt in Maya die Möglichkeit Objekte mit Hilfe eines Rasters zu platzieren.



euge

2.1.3 Die „Nebula2 Toolkit for Maya“-Funktionen

Maya ist als Level-Editor für die Nebula-Engine nur im Zusammenhang mit dem „Nebula2 Toolkit for Maya“ (kurz Toolkit) nutzbar. Das Toolkit besteht aus einer Sammlung von Werkzeugapplikationen und enthält ein Plug-in für Maya, das beim Programmstart geladen wird. Die enthaltenden Applikationen unterstützen das Plug-in. So bietet beispielsweise der N3Viewer die Möglichkeit einer Vorschau von mit Maya erstellten Grafikobjekten.

Das Toolkit fügt zwei zusätzliche Menüs in die Menüleiste ein. Dabei handelt es ich um das Nebula-Menü und das Mangalore[10]-Menü. Das für die Funktionalität als Level-Editor relevante Menü ist, das Mangalore-Menü.

Das Mangalore Menü ermöglicht es neue Level-Dateien zu erstellen, zu laden und zu speichern. Anders als bei den Standardfunktionen für „Neu“, „Öffnen“ und „Speichern“ wird nicht ein Explorer geöffnet, sondern ein Fenster, das ausschließlich den Inhalt des Ordners mit den

[10] Mangalore ist ein Kunstwort, für ein Rahmenwerk das auf der Nebula2-Engine aufsetzte und diese um für Spiel relevante Klassen erweiterte.

jeweiligen Level-Dateien anzeigt. Nebula gibt eine fixe Ordnerstruktur für das Ablegen von Dateien vor, daher werden stets statische Pfade verwendet. Die Benutzung dieser Funktionen verhindert, dass Daten an falschen Orten abgelegt werden und vereinfacht damit die Arbeit des Benutzers.

Es existiert eine Export-Funktion, über die ein Level in ein von der Nebula-Anwendung verstandenes Format übertragen werden kann. Eine weitere Funktion ermöglicht es dem Anwender die Nebula-Anwendung zu starten, für die die Daten vorgesehen sind. Diese Vorschaufunktion wird sehr häufig genutzt.

Spielobjekte können in drei große Kategorien untergliedert werden, reine Grafikobjekte, Spiellogikobjekte mit Grafik, Spiellogikobjekte ohne Grafik und Lichtobjekte. Jedes Objekt in Maya besitzt Attribute, Objekte mit Spiellogik besitzen zusätzliche Attribute, die ihre Funktionalität im Spiel definieren. Im Mangalore-Menü finden sich auch Operationen zum Einfügen und Bearbeiten von Spielobjekten und ihren Attributen.

Um die Bearbeitung von Spielobjekten zu unterstützen existiert eine Funktion im Toolkit, die es ermöglicht eine Verbindung mit einem laufenden Nebula-Spiel aufzunehmen und Änderungen an bestimmten Spielobjekten von Maya direkt auf das Spiel zu übertragen. Die Steuerung der Verbindung wird über das sogenannte RemoteControl[11]-Fenster realisiert. Nimmt man beispielsweise ein Audioobjekt und platziert es in Mayas 3D-Umgebung, so kann man dessen Wirkung erst überprüfen, wenn man es im laufenden Spiel gehört hat. Mit Hilfe des RemoteControl-Fensters ist es möglich bestimmte Objekttypen in Maya zu bearbeiten während man gleichzeitig ein Spiel mit diesem Level geöffnet hat. Ändert man die Position oder das Audio-Ereignis, das abgespielt werden soll, so ist diese Änderung augenblicklich im Spiel wahrnehmbar.

Für die Physik-Engine, die bestimmt ob man im Spiel mit einer Wand zusammenstößt oder durch sie hindurch läuft, sind zusätzliche Grafikobjekte notwendig, die im Spiel unsichtbar sind. Diese Objekte werden als „Collide[12]“ bezeichnet. Häufig unterscheiden sie sich von der sichtbaren Grafik. In Maya ist es möglich diese Objekte zusätzlich zu laden und anzuzeigen um knifflige Stellen genauer einzustellen.

Für einige Spiele war es notwendig ein komplexes Gitter für die Wegfindung zu erzeugen, die es den Figuren ermöglicht auf einem korrekten Pfad durch den Level zu navigieren. Das Toolkit stellt Funktionen zur Verfügung, die es ermöglichen diese Gitter zu generieren und nachzubearbeiten.

---

[11] RemoteControl zusammengesetzt aus remote, engl. für fern und control, engl. für Kontrolle. Kann übersetzt werden als Fernsteuerung
[12] collide, engl. für aufeinander prallen

2.1.4 Gründe Maya als Level-Editor zu ersetzen

Hier sollen die wichtigsten Gründe aufgeführt werden, die die Notwendigkeit eines neuen Level-Editors verdeutlichen.

Einer der wichtigsten Gründe, ist der Wunsch nach einer leichteren Bedienung. Maya bietet einen übergroßen Umfang an Funktionalität, der aber selten in diesem Ausmaß genutzt wird und somit eine intuitive Bedienung erschwert. Außerdem wird das Anlernen neuer Mitarbeiter dadurch unnötig erschwert. Die hier vorgestellten Funktionen beschreiben nur einen Bruchteil der von Maya bereitgestellten Funktionspalette. Es gibt Konfigurations- und Bearbeitungsmöglichkeiten, welche die Benutzeroberfläche und Arbeitsweise individuell anpassbar machen. Das bedeutet auch, dass es bei vielen Arbeitsweisen keine einheitliche Vorgehensweise gibt, Probleme zu lösen.

Maya ist über Plug-ins und Skripte[13] erweiterbar. Benutzer können eigene Funktionen in MEL[14] oder Python[15] schreiben. Allerdings ist die Komplexität des vorhandenen Codes und der Skripte für das Toolkit-Plug-in relativ hoch. Es erfordert einen großen Programmieraufwand diese den unterschiedlichen Projekten anzupassen, zu erweitern und sie zu warten.

Die Vorschaufunktion ist, wenn man mit Maya arbeitet ein wichtiges Mittel um die korrekte Platzierung und Funktionsweise der einzelnen Objekte zu überprüfen und wird laut Aussage der Level-Designer[16] sehr häufig verwendet. Insbesondere bei größeren Projekten betrug die Startzeit aus Maya heraus 1-3 Minuten. Im Laufe eines Arbeitstages summiert sich dies zu einer beträchtlichen Wartezeit.

Eine Rolle, wenn auch untergeordnet, spielt der Preis. Maya ist als lizensiertes Produkt recht teuer und es macht wenig Sinn diese teure Software zu installieren obwohl sie nur im geringen Umfang genutzt wird.

2.3 Der „Torchlight-Editor - TorchED“

Nachdem im vorangegangenen Abschnitt Maya als Vorgänger für den Level-Editor vorgestellt wurde, wird in diesem Abschnitt der „Torchlight-Editor“ vorgestellt und mit Maya verglichen. Das Programm TorchED [Run09] wurde von der Firma Runic Games für das Spiel „Torchlight“ entwickelt. Es wurde 2009 veröffentlicht. Der Editor, wurde auch benutzt um das gleichnamige Spiel zu entwickeln. Er wurde der Fan-Gemeinde zur Verfügung gestellt. In diesem Abschnitt

[13] Skripte, in einer Skriptsprache verfasste Funktionen und Programme.
[14] MEL, Abkürzung für Maya Editor Language eine Maya-interne Skriptsprache.
[15] Python ist eine höhere Programmiersprache, die oft auch als Skriptsprache verwendet wird.
[16] Gemeint sind Level-Designer der Firma Bigpoint Berlin

werden die Besonderheiten herausgestellt, die ihn von Maya als Editor unterscheiden. Außerdem wird aufgezeigt wo dieser Editor seine Schwerpunkte setzt.

Wie bei Maya sind die Daten auch hier in einer Baumstruktur angeordnet. Jeder Knoten hat eine Reihe von Attributen. Die Knoten mit Kindern sind Gruppenknoten, Blätterknoten sind spezielle Knoten. Sie können spiel-logische Elemente oder Raum-Stücke sein, welche einem Grafikobjekt in Maya entsprechen.

Auch hier gibt es einen Outliner, der die Datenstruktur darstellt und ein Fenster für eine 3D-Ansicht auf den Level. Es gibt ein Fenster in dem die Attribute eines Knoten angezeigt werden, mit dem Unterschied das es davon nicht wie in Maya diverse Variationen gibt, sondern immer das Steuerelement, angezeigt wird. Das Steuerelement ist ein Eigenschaftenfenster indem die Attribute untereinander aufgelistet sind. Für spezielle Attribute gibt es Editoren, die über diese Übersicht erreicht werden können.

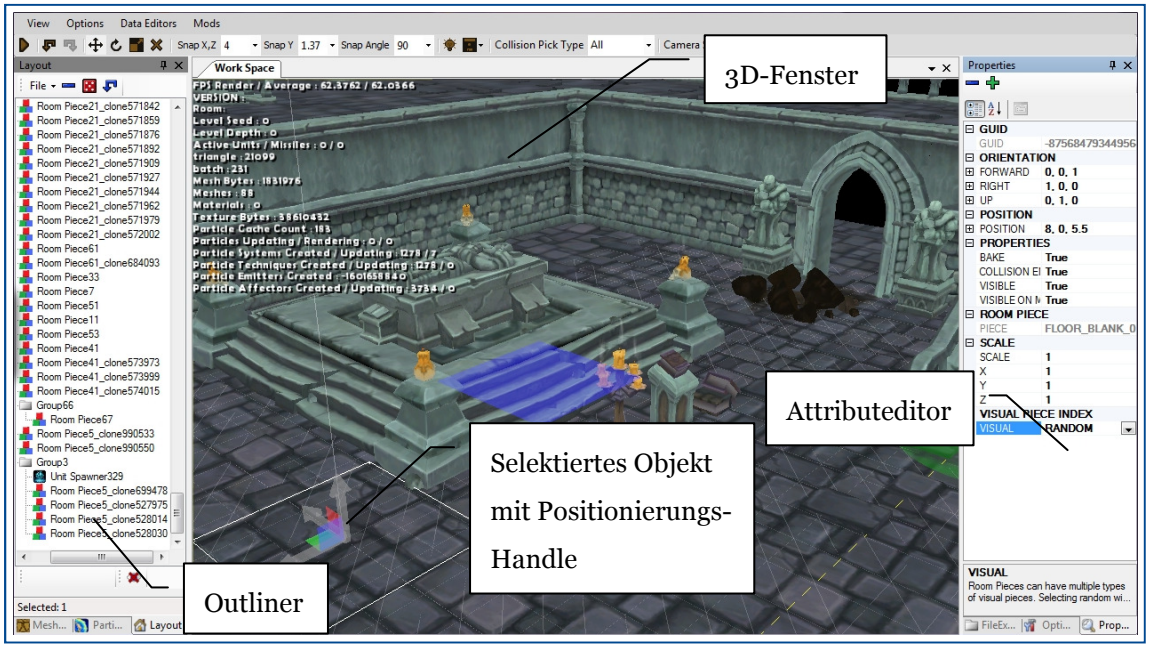


Abbildung 3: TorchED Benutzeroberfläche

Es ist möglich Objekte zu verschieben, zu rotieren und zu skalieren. Die Platzierung findet grundsätzlich auf einem Raster statt, das vom Objekt vorgegeben wird.

Um Objekte einzufügen gibt es die Palette. Das ist ein Browser, der Objekte aus einer Datenbank anzeigt. Er verfügt über Filterfunktionen und eine Vorschau auf die Objekte. Fügt man in Maya ein Objekt ein, muss es immer anhand des Namens identifiziert werden, was gerade für neue Mitarbeiter problematisch ist.

Auch der TorchED verfügt eine Vorschaufunktion. Anders als in Maya wird das Spiel im Editor gestartet und enthält auch Visualisierungen von Spiellogikobjekten, die normalerweise unsichtbar wären.

Es gibt zusätzliche Funktionen, die nichts mit dem Aufbau der Level zu tun haben. So ist es möglich die Questen, die Aufgaben die der Spieler im Spiel zu erledigen hat, in einem eigenen Fenster zu editieren. Auch gibt es noch diverse andere Werkzeuge um das Spiel zu konfigurieren, die über den Umfang eines Level-Editors hinausgehen. Sie sind in separate Fenster ausgelagert, die über Menüpunkte aufgerufen werden können.

Im Vergleich zu Maya ist das Arbeiten mit dem Torchlight-Editor einfacher, da er klar definierte Regeln festlegt. Die Benutzeroberfläche beschränkt sich auf Funktionen, die häufig verwendet werden.

Mit Maya und TorchED hat man in gewisser Weise zwei Gegensätze. Maya ist flexibel, erweiterbar und vielseitig einsetzbar, während TorchED nur für ein bestimmtes Spiel und Spielkonzept entwickelt wurde. Der im Rahmen dieser Arbeit zu erstellende Level-Editor soll einen Kompromiss aus diesen zwei Extremen darstellen. Er muss einfach zu bedienen sein und die Rolle als Level-Editor ausfüllen. Er soll nicht dazu benutzt werden Modelle zu editieren, zu animieren oder Effekte zu erstellen. Er benötigt Funktionen um Grafik- und Spiellogikobjekte zu platzieren und die Spiellogik zu konfigurieren. Er soll erweiterbar sein, so dass es möglich ist, spezielle Werkzeuge für unterschiedliche Spiele zu integrieren.

2.4 Anforderungen an den Level-Editor

Dieser Abschnitt beschäftigt sich mit den konkreten Anforderungen an den Level-Editor. Es wird geklärt, welche dieser Anforderungen im Rahmen dieser Arbeit umgesetzt werden sollen und welche Anforderungen in späteren Erweiterungen umgesetzt werden sollen. Die folgenden Anforderungen wurden in Besprechungen mit Repräsentanten der Grafik- und Level-Design-Abteilungen der Firma Bigpoint Berlin erarbeitet.

Zu den allgemeinen Zielen gehört, dass der Level-Editor benutzerfreundlich aufgebaut sein soll. Der Umgang mit ihm soll leicht zu erlernen sein. Dieses Ziel wird im ersten Anlauf nicht vollständig umgesetzt werden können, da umfangreiche Evaluationen mit den späteren Benutzern notwendig sind.

Der Editor soll erweiterbar sein, um ihn an die unterschiedlichen Projekte anpassen zu können.

Der Editor muss 3D-Grafikobjekte anzeigen, und zwar so wie diese im Spiel gezeigt werden. Ausnahme bilden hierbei spezielle Spiellogik-Objekte. Das Platzieren der Objekte soll auf einem Gitterraster erfolgen.

Es wurde beschlossen, dass folgende Punkte in der ersten Implementierung enthalten sein sollen:

1. Es soll zwei Kameras geben. Eine Kamera gibt die Sichtweise vor, von welcher eine 3D-Szene betrachtet wird. Mit Eigenschaften wie zum Beispiel dem Blickwinkel, Öffnungswinkel oder Position kann man die Ansicht auf die Szene verändern. Wenn von unterschiedlichen Kameras die Rede ist, dann bezieht sich dies auf unterschiedliche Möglichkeiten der Tastatur- und Maussteuerung, sowie weiteren Eigenschaften, die an der Kamera voreingestellt sind. Die erste Kamera soll der Kameraführung von Maya in der Steuerung nachempfunden sein. Die zweite Kamera soll der Kamera im Spiel entsprechen, das mit dem Editor entwickelt werden soll.
2. Es soll ein dreidimensionales Raster geben, auf dem Grafikobjekte angeordnet werden können. Dabei soll die Höhe getrennt von Tiefe und Breite des Rasters einstellbar sein. Bestimme Objekte dürfen nur auf die Höhenebenen einrasten.
3. Ein Outliner wie ihn sowohl Maya als auch TorchED besitzen, soll implementiert werden. Er muss das Gruppieren von Objekten unterstützen.
4. Es soll die Möglichkeit geben, gemachte Änderungen per Knopfdruck rückgängig zu machen.
5. Es soll einen Browser geben um Grafik- und Spiellogikobjekte dem Level hinzuzufügen.
6. Die Konfiguration von Spiellogik-Objekten muss möglich sein. Diese Objekte können zum Beispiel Fallen oder andere Auslöser sein. Welche Objekte zur Verfügung stehen wird vom Spiel vorgegeben.
7. Es wird ein Baukasten benötigt, der Objekte in verschiede Typen einteilt, wie zum Beispiel in Bodenteile, Wände, Requisiten und Vegetation. Baukasten bezeichnet hier die Möglichkeit Grafikobjekte ihrem Typ entsprechend zu sortieren und auszuwählen, ähnlich der Palette von TorchED. Dabei soll es auch hier eine Vorschau-Möglichkeit für die einzelnen Objekte geben.
8. Zusätzlich müssen „Layer“ wie in Maya implementiert werden. Sie dienen der Selektion von Objektgruppen, deren Objekte man in der 3D-Ansicht unsichtbar schalten und vor Bearbeitung schützen kann.
9. Es soll eine Vorschaufunktion geben, bei der die aktuellen Level-Daten, wie bei der Maya-Vorschau, exportiert werden und das Spiel mit diesem Level gestartet wird.

In dieser Arbeit wurde ein Rahmenwerk gebaut, dass über Plug-ins und das Ableiten von existierenden Klassen erweiterbar ist. Die Punkte 1-6 finden sich entweder in diesem Rahmenwerk oder in den Plug-ins wieder. Diese erste Implementierung ist das Grundgerüst, das heißt sie ist noch nicht final, kann aber schon verwendet werden.

Als Basis für das Rahmenwerk wurde die Programmiersprache C# gewählt, da bereits eine Bibliothek von C#-Funktionen für die Nebula3-Engine existiert, die bei der Implementierung dieser Anwendung genutzt werden kann.

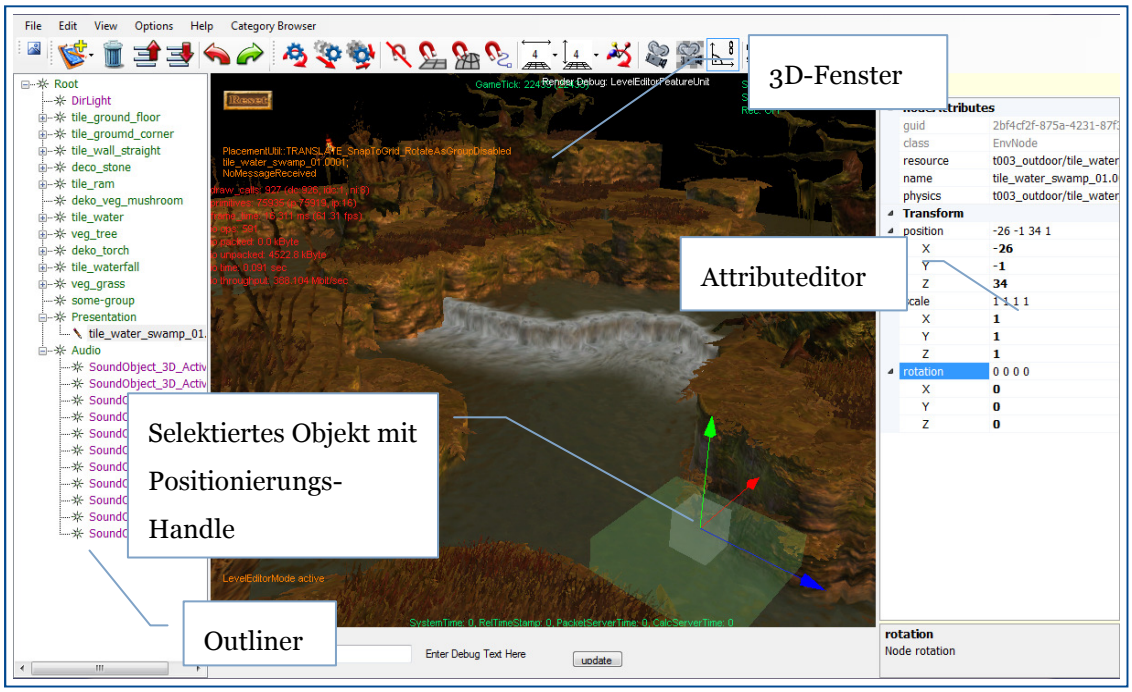


Abbildung 4: Benutzeroberfläche des Level-Editor für die Nebula3-Engine mit einem geladenen Level

Das Rahmenwerk verwaltet die Daten eines Levels. In die zur Verfügung gestellten Rahmen lassen sich Plug-ins laden. Die Nebula-Applikation, welche eingebettet wurde, ist der Spiel-Client von Drakensang-Online[17], das sich zum Zeitpunkt dieser Arbeit ebenfalls in Entwicklung befand.

Die folgenden Kapitel sollen die verwendeten Bibliotheken vorstellen und beschreiben wie sie für den Level-Editor und die aufgelisteten Anforderungen eingesetzt werden. Anschließend soll die Plug-in Schnittstellenstruktur genauer vorgestellt werden.

[17] Die dargestellten Drakensang-Online-Level sind Eigentum der Firma Bigpoint Berlin und wurden von der Grafikabteilung erstellt.

3 Nebula3 Engine

Um ein Nebula-Spiel zum Bearbeiten von Spielinhalten im Level-Editor zu verwenden, mussten in Nebula neue Komponenten erschaffen werden. Deshalb stellt dieses Kapitel kurz den Aufbau der Nebula-Engine vor. Es wird geklärt welche Komponenten verwendet, verändert oder neu hinzugefügt wurden, um aus einem Spiel einen Echtzeit-Editor für Level-Daten zu machen.

Im ersten Teil wird erklärt, wie die Level-Editor-Komponenten in der Nebula3-Engine verankert sind. Um diese Einbindung nachvollziehen zu können, ist eine kurze Übersicht über die Struktur der Nebula3 notwendig.

3.1 Die Schichten der Nebula3 Engine

Die Nebula3-Architektur [Wei107] besteht aus Schichten, die aufeinander aufbauen.

Unter der Nebula-Engine liegt das Betriebssystem. Auf diesem baut die Foundation-Schicht [Wei207] auf. Als unterste Schicht stellt sie eine einheitliche Schnittstelle zu den unterschiedlichen Plattformen bereit und kapselt die Betriebssystemfunktionen in Klassen, die von den höheren Schichten betriebssystemunabhängig benutzt werden können.

Die Foundation-Schicht implementiert dazu das Nebula3-Objektmodel, die Speicherverwaltung, das IO-System[18], das Threading[19], die Netzanbindung, die Mathematik-Bibliothek und weitere nützliche Werkzeug- und Template-Klassen.

Die Render-Schicht setzt auf der Foundation-Schicht auf und ergänzt diese unter anderem um 3D-Rendering[20], Audio-Unterstützung und Physik.

Die Applikationsschicht ist die oberste Ebene. Sie stellt ein komplettes Rahmenwerk zur Verfügung, welches es ermöglicht ein Spiel zu erstellen. Dem Programmierer soll diese Ebene ermöglichen sich primär auf die Spiel-Logik zu konzentrieren.

---

[18] IO-System – das Ein- und Ausgabesystem
[19] Thread, engl. für Strang oder Faden, beschreibt einen Ausführungsstrang bei der Abarbeitung eines Programmes.
[20] Rendering - von engl. to render, hier eine Methode um 3D-Informationen als ein für den Nutzer sichtbares 3D-Objekt darzustellen

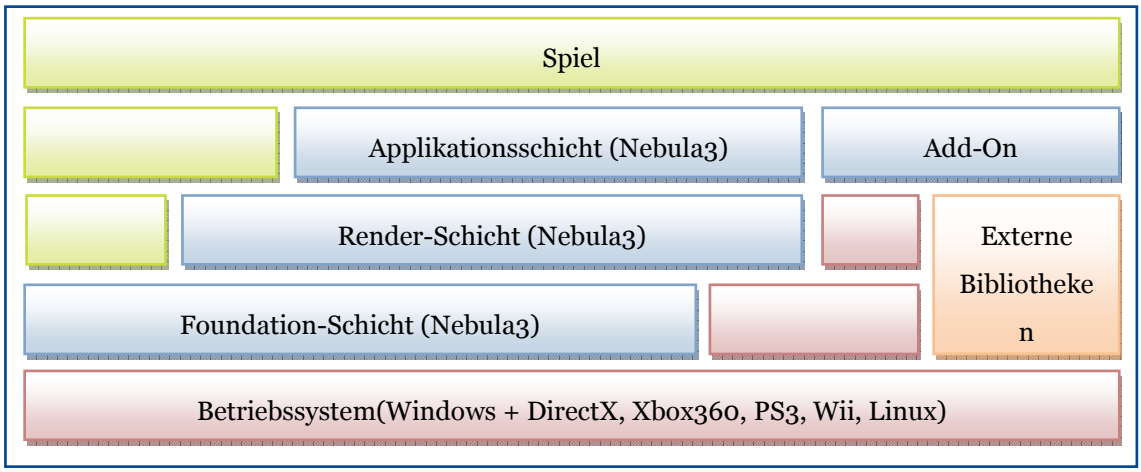


Abbildung 5: Schichten der Nebula3-Engine

Zusätzlich zu den Ebenen ist ein Teil der Nebula3 Struktur in sogenannten Add-Ons gekapselt, deren Funktionalität sich über mehrere Ebenen ausdehnen kann. Ein Add-On erweitert die Funktionalität der Engine, ohne dass diese zwingend von ihm abhängig wäre. In einem Add-On können externe Bibliotheken eingebunden werden, wie zum Beispiel Bink [RAD10] , eine Bibliothek zum Abspielen von Audio- und Videodateien. Darüber hinaus werden Klassen zusammengefasst, die thematisch verwandt sind, aber nicht in jeder Applikation benötigt werden. Diese können dann bei Bedarf eingebunden werden.

Der Level-Editor Quellcode in Nebula ist in der Applikationsschicht als Add-On organisiert. Das bedeutet, dass nicht jedes Nebula-Spiel automatisch den Level-Editor unterstützt. Das Add-On muss also erst eingebunden werden. Vorteil dieser Art der Implementierung ist, dass er aus dem Code entfernt werden kann, ohne die Spiellogik zu beeinflussen.

3.2 Grundlegende Konzepte in der Applikationsschicht

Der Level-Editor benutzt Konzepte aus der Applikationsschicht [Wei307] . Um die Funktionalität der Applikationsschicht in Nebula zu verstehen, ist es wichtig die sechs Begriffe., Entity, Property, Attribute, Manager, Message und GameFeature zu kennen:

- Als **Entity**, auch GameEntity genannt, wird eine Klasse bezeichnet, die alle Arten von Spielobjekten abbildet. Die Entity ist ein Container für Attribute und Properties. Über sie wird ein Spielobjekt definiert.
- Ein **Attribut** ist ein Klasse, die einen Schlüssel und einen Wert enthält. Das Attribut definiert den Typ des Wertes.
- Eine **Property** ist eine Klasse, die Spiellogik enthält, das heißt ausführbaren Code.
- Properties können über **Messages** mit den Properties der eigenen oder anderer Entities kommunizieren.

- **Manager** sind im allgemeinen Objekte, die als Singleton [DP394] , als einzigartige Instanz existieren und Dienste anbieten, die von überall im Thread aus erreichbar sind. So übernehmen sie zum Beispiel die Erzeugung und Verwaltung der Spielobjekte.
- **GameFeatures** bündeln Managers, Properties, Messages und Attribute, die eine funktionelle Einheit bilden. Ein GameFeature ist nicht gleichzusetzen mit einem Add-On. Ein Add-On ist eine Bibliothek ein GameFeature hingegen greift auf Bibliotheken zu und initialisiert Manager, die zu seiner Funktionalität passen.

Betrachte man als vereinfachtes Beispiel zwei klassische Spielobjekte, die man aus Rollenspielen kennt. Ein Avatar, den der Spieler steuern kann und eine Kiste, die der Spieler im Spiel benutzen soll. Kiste und Avatar werden durch die Klasse Entity repräsentiert. Sie unterscheiden sich nur in ihren Attributen und ihren Properties. Eine Kiste hat ein Attribut State, das ihren Zustand definiert. Dieser kann den Wert „geschlossen“ oder „offen“ annehmen. Außerdem hat die Kiste ein Attribut, das ihr Aussehnen definiert, in Form eines Pfades zu der grafischen Ressource, die es verwenden soll. Ein Avatar hat auch ein oder mehrere Attribute, die sein Aussehen vorgeben. Zusätzlich besitzt er noch weitere Attribute die Lebensenergie, Stärke und Fähigkeiten speichern. Die Properties, die an den Entities hängen, bestimmen wie sich diese verhalten. So hängt eine StateObjectProperty an der Entity, die eine Kiste repräsentiert. Empfängt sie eine Message, die das Attribut State von „geschlossen“ auf „offen“ ändern soll, so wird sie das Attribut neu setzen und gleichzeitig die Grafikressource umschalten. Die Property schickt weitere Messages, die dafür sorgen dass im Grafiksystem nun eine offene Kiste gerendert wird.

Die Nachricht zum Umschalten des Zustands kann zum Beispiel von einem Manager stammen, der die Maus- und Tastatureingaben übermittelt bekommt und sie an die entsprechenden Entities oder Manager weiterleitet.

3.3 Das Level-Editor-Add-On in Nebula3

Der Nebula3 Teil des Level-Editors ist ein Add-On, das durch ein GameFeature LevelEditorFeatureUnit initialisiert wird. Um gestartet zu werden benötigt es ein initialisiertes GameSystem. Das GameSystem stellt die Manager für das Erstellen von Entities und Properties zur Verfügung. Es ermöglicht Datenbank Zugriff und stellt ein Zeitmanagement zur Verfügung. Die meisten benötigten Klassen werden von der Basisklasse GameApplication bereits initialisiert, dazu gehören auch die Remote Klassen für die Unterstützung der Kommunikation zwischen dem C# Framework und dem Spiel.

Zusätzlich werden das BaseGameFeature, das GraphicsFeature und das Physicsfeature benötigt. Das GraphicsFeature stellt alles zur Verfügung was zum Rendern der Grafik benötigt wird. Das

BaseGameFeature initialisiert das Audiosystem, welches verwendet wird um Sounddaten aus dem Level-Editor heraus abzuspielen. Das PhysicsFeature ermöglicht Kollision und physikalische Repräsentation von Spielobjekten und eröffnet die Möglichkeit der Objektauswahl über die Physikobjekte.

Die wichtigste Klasse der LevelEditorFeatureUnit ist der LevelEditorManager. Er ist die zentrale Verwaltungsklasse für das Add-On. Durch ihn werden die Liste der angezeigten Objekte sowie eine Liste der selektierten Objekte verwaltet. Er koordiniert den Input und leitet diesen an das PlacementUtil und die LevelEditorCameraProperty weiter. Deren Funktionalität wird in den folgenden Abschnitten erläutert wird. Über den LevelEditorManager werden Nachrichten erstellt, die an das C# Level-Editor Framework gesendet werden. Außerdem stellt er Methoden zur Verfügung die es erlauben die angezeigten Objekte und die Selektion zu verändern.

3.4 Echtzeit Platzierung in Nebula3 – Das PlacementUtil

In diesem Abschnitt soll das PlacementUtil vorgestellt werden. Es ist ein Teil des InGameEditing-Nebula3-Add-Ons und stellt Werkzeuge zur Verfügung, die es erlauben ein Objekt im Dreidimensionalen Raum in Echtzeit zu positionieren, zu rotieren und zu skalieren. Es soll zunächst die Funktionsweise erklärt werden und dann auf die Abwandlungen eingegangen werden, die für den Level-Editor gemacht wurden.

Um Position, Rotation und Skalierung von Objekt festzulegen, wird die Transform-Matrix verwendet. Eine Transform-Matrix in Nebula3 ist eine 4x4 Matrix:

$$m = \begin{pmatrix} x1, & x2, & x3, & x4 \\ y1, & y2, & y3, & y4 \\ z1, & z2, & z3, & z4 \\ p1, & p2, & p3, & p4 \end{pmatrix}$$

Der Vektor $\vec{p} = (p1, \quad p2, \quad p3)$ gibt die Position im Dreidimensionalen Raum an. Die Vektoren $\vec{x} = (x1, \quad x2, \quad x3)$, $\vec{y} = (y1, \quad y2, \quad y3)$ und $\vec{z} = (z1, \quad z2, \quad z3)$ stellen zueinander orthogonale Achsen dar. Die Rotation ergibt sich aus ihrer Relation zur Einheitsmatrix. Aus der Länge der Vektoren $|\vec{x}|, |\vec{y}|, |\vec{z}|$ ergibt sich die Skalierung für die jeweiligen Achsen.

Das PlacementUtil kapselt drei verschiedene Modi. Den Translations-, den Rotations- und den Skalierungsmodus. Jeder Modus stellt sogenannte Handles[21] zur Verfügung, die es ermöglichen eine Matrix auf dem Bildschirm mit der Maus zu verschieben, zu rotieren oder zu skalieren. Die unterschiedlichen Achsen sind farblich gekennzeichnet, Rot steht für die x-Achse, Grün für die y-Achse und Blau für die z-Achse.

[21] Handle, abgeleitet aus dem Englischen, bedeutet übersetzt Griff.

Im Translations-Modus verschiebt man die Matrix auf den Achsen in dem man das jeweilige Handle anwählt oder auf den Würfel in der Mitte klickt um die Matrix „frei“ zu platzieren. Im Rotations-Modus rotiert man um die jeweiligen Achsen oder um die Sichtachse, die als äußerer Kreis dargestellt wird. Skaliert werden kann über die jeweiligen Achsen oder über den Würfel im Zentrum, der dann alle Achsen skaliert.

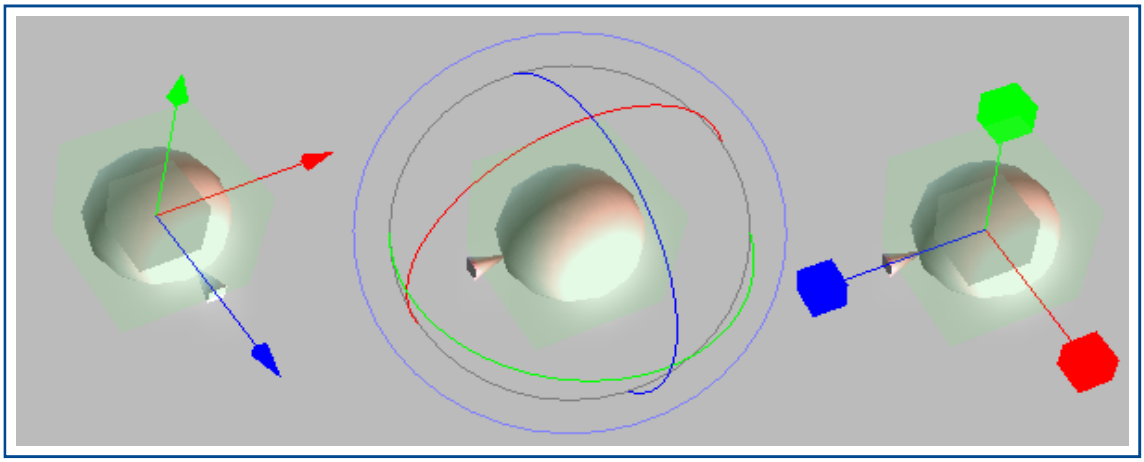


Abbildung 6: Transformation-Werkzeuge des Level-Editor – Handle für Platzierung, Handle für Rotation und Handle für Skalierung

Das Transformieren der Matrix wird in drei Phasen aufgeteilt, eine Initialisierungsphase in der die Startmatrix übergeben wird, eine Platzierungsphase in der die Matrix sich verändert bis die Maus losgelassen wird, dann folgt die Endphase. In der Endphase wird die Position entsprechend des gewählten Modus gesetzt. Während des Ziehens wird die neue Transformation auf das Objekt übertragen, wodurch das Objekt in Echtzeit verschoben wird.

3.4.1 Erweiterungen des PlacementUtil für den Level-Editor

Die Modi werden in separaten Klassen abgehandelt, was es möglich macht sie zu erweitern in dem man von den Basisklassen erbt. Das PlacementUtil wurde dahingehend erweitert, dass es nun möglich ist die Klassen für die Translation und die Rotation durch abgeleitete Klassen zu ersetzen. Dies war nötig um die Platzierung auf einem Gitterraster zu ermöglichen.

Das Gitterraster für die x-z-Ebene besteht aus Quadraten, die Höhe ist ein separater Wert. Die Größen für das Gitterraster können angepasst werden. Für die Platzierung auf dem Gitter wurden drei umschaltbare Modi eingebaut, zusätzlich zu dem freien Platzieren das von vornherein enthalten war. Die Modi heißen SnapToGrid, SnapToGround und SnapToEdge.

Der Modus SnapToGrid platziert die Matrix in der x-z Ebene in die Mitte eines Gitterstückes. In der y-Koordinate wird sie auf den Boden des Quaders, den das Raster bildet, platziert. Die Matrix wird dem Raster zugeordnet, zu dem sie den geringsten Abstand besitzt. Visualisiert wird

dieser Modus bei dem Ziehen der Matrix über einen Würfel, der die Ausmaße eines Gitterfeldes hat. Dieser Modus soll vorwiegend dazu verwendet werden Bodenplattenobjekte zu platzieren.

Im SnapToGround Modus fällt das Korrigieren in der x-z Ebene weg. Es wird nur die Höhe angepasst nach dem man das Handle losgelassen hat. Ein sehr flacher Quader, der beim Ziehen zur nächsten Höhenebene springt wird als Visualisierung für diesen Modus verwendet. Damit können Objekte auf einem ebenen Boden platziert werden, der dem Gitterraster entspricht.

Die letzte Variante ist SnapToEdge. Sie dient dazu Objekte auf den Kanten des Gitternetzes zu platzieren. Dieser Modus soll dabei helfen Wände und andere Abschlusselemente einer Ebene anzuordnen. Die Matrix wird auf einen zentrierten Punkt auf dem Rand ausgerichtet. Die Platzierungshöhe wird wie bei den anderen beiden berechnet Modi berechnet.

Für die Rotation wurde ebenfalls ein Mechanismus zum Einrasten entwickelt, genannt RotationSnap. Für die Rotation von Wandstücken ist es notwendig, dass sich diese genau im 90° Winkel drehen lassen. Ist diese Funktion aktiv, so dreht sich das Objekt sprunghaft im 90° Winkel relativ zu den Achsen der Matrix.

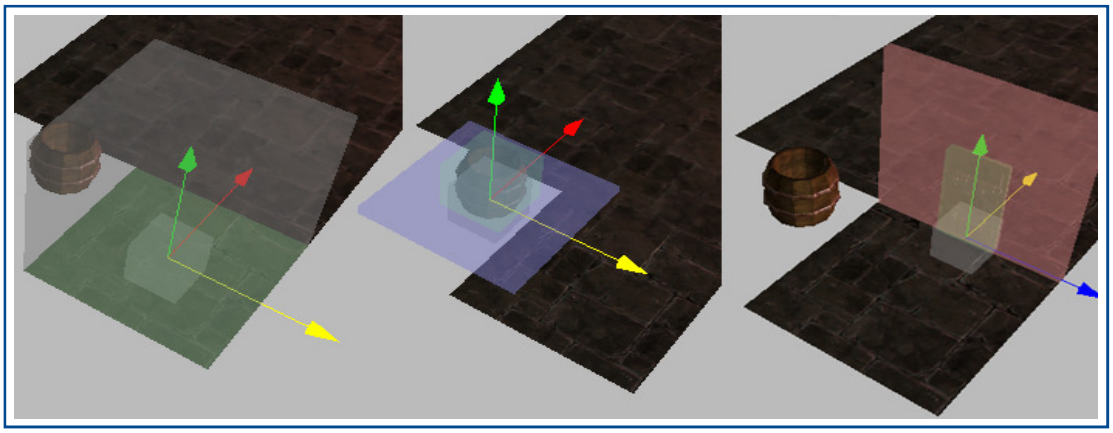


Abbildung 7: Platzierungsvarianten des Level-Editor - SnapToGrid, SnapToGround und SnapToEdge

Im Verlauf der Evaluation durch die Level-Design-Abteilung haben sich zusätzliche Anforderungen ergeben. So wurde für die Rotation von mehreren Objekten eingebaut, so dass entweder die Möglichkeit besteht mehrere Objekte auf einmal um ihr eigene Achse zu drehen, oder um einen gemeinsamen Pivot. Zusätzlich wird bei aktivem SnapToGrid dieser Pivot auf einen Punkt auf dem Gitterraster festgelegt, um zu vermeiden, dass nach einer Rotation die Positionen der Objekte nicht mehr auf dem Raster liegen.

3.5 Kommunikation mit dem Level-Editor – Die Remote-Klassen von Nebula3

In diesem Abschnitt wird die Schnittstelle vorgestellt, über die eine Nebula3-Applikation und der Level-Editor kommunizieren. Es wird erklärt, welche Klassen speziell für den Level-Editor implementiert werden mussten und welche Funktionen sie erfüllen.

Die Schnittstelle über welche die Kommunikation stattfindet, besteht aus Remote-Klassen. Sie wurden ursprünglich entworfen um mit externen Programmen, wie zum Beispiel mit einem Internetbrowser, über eine TCP/IP[22] Verbindung Daten aus dem Spiel abzurufen. Durch die Weiterentwicklung dieser Klassen wurde es möglich, Daten im XML-Format an das Spiel zu senden. Dem Spiel ist es möglich durch einen Broadcast[23] auf seinen bestehenden Verbindungen Nachrichten zurück zu senden.

Der RemoteServer läuft in einem eigenen Thread. Dieser Thread wird vom RemoteInterface erstellt. Er enthält einen RemoteMessageHandler, der in der Lage ist TCP/IP Verbindungen anzunehmen und Nachrichten zu empfangen. Das RemoteInterface stellt die Kommunikationsverbindung vom RemoteServer-Thread zum Haupt-Thread der Applikation bereit. Die Threads kommunizieren über Messages miteinander. Auf der Seite des Haupt-Threads übernimmt die RemoteControlProxy Klasse die Kommunikation.

Um spezielle Nachrichten zu empfangen und weiter zu verarbeiten müssen Steuerungsklassen, sogenannte Handler, implementiert und dem RemoteControlProxy übergeben werden. Dieser benutzt die Handler zum Abarbeiten der Messages.

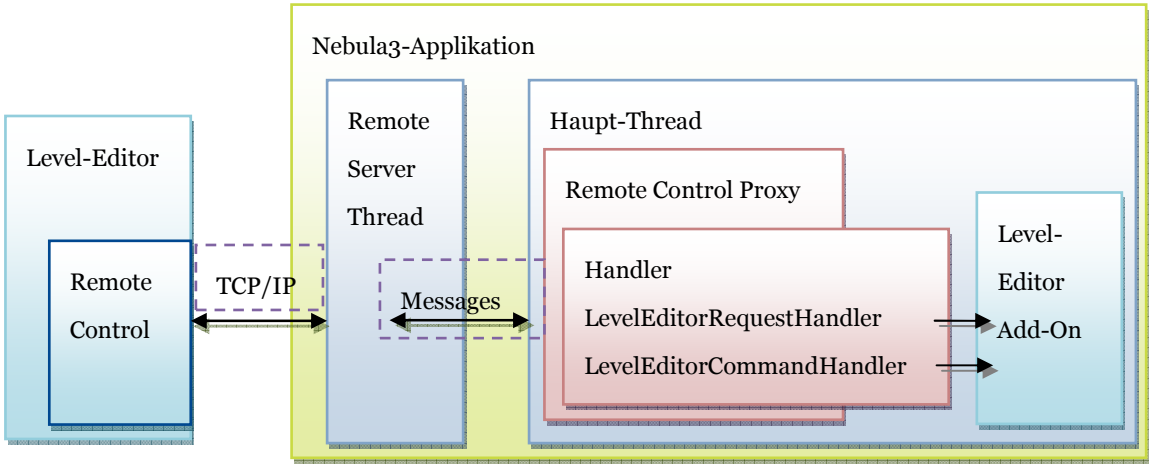


Abbildung 8: Kommunikation des Level-Editors mit dem Level-Editor Add-On in Nebula

[22] TCP/IP –Transmission Control Protocol/Internet Protocol
[23] Broadcast -engl. für Rundsendung, ein Senden von Daten an alle Empfänger

3.5.1 Erweiterung der Remote-Klassen für den Level-Editor

Es gibt zwei verschiedene Arten von Handlern, die für die Kommunikation mit dem Level-Editor implementiert wurden, den RemoteRequestHandler und den RemoteCommandHandler.

Ein RemoteRequestHandler dient dem Empfang und der Verarbeitung von XML-Dokumenten, während ein RemoteCommandHandler für die Verarbeitung von einem einzigen Befehl und einer Argumentauflistung gedacht ist. Im Level-Editor-Add-On heißen diese beiden Klassen LevelEditorRequestHandler und LevelEditorCommandHandler.

Der LevelEditorCommandHandler empfängt kurze Befehle und gibt sie an den LevelEditorManager weiter. Solche Befehle sind momentan nur Anfragen aus dem Benutzerinterface, wie zum Beispiel der Befehl SetSnapMode um die Art der Platzierung im Raster umzuschalten.

Die Aufgabe des LevelEditorRequestHandler ist es die empfangenden XML-Dokumente zu parsen. Es gibt verschiedene Arten von Inhalten, die über diesen Handler empfangen werden können. Zum einen gibt es die Möglichkeit, die gesamten Level-Daten, oder Fragmente daraus zu empfangen. Eine weitere Form ist eine generische Anfrage, identifiziert über das Schlüsselwort GenericRequest.

Werden Level-Daten empfangen, so werden diese genutzt um die Darstellung der Objekte zu erneuern. Im einfachsten Fall werden alle Objekte gelöscht und mittels der empfangenden Daten neu erzeugt. Das ist zum Beispiel der Fall, wenn ein Level geladen wird.

Eine generische Anfrage funktioniert ähnlich wie eine Befehlszeile im RemoteCommandHandler. Sie enthält ein Attribut mit dem Namen des Befehls und untergeordnete Knoten als Parameter. Mit Hilfe der Klasse Core::Factory wird zur Laufzeit aus dem Befehlsnamen eine Klasse generiert, die von LevelEditorRemoteRequest abgeleitet sein muss. Dieser Klasse können die geparsten Parameter übergeben werden. Sie besitzt eine Execute Methode, die nach dem Parsen aller Daten ausgeführt wird. Auf diese Weise werden unter anderem das Aktualisieren und das Selektieren von Objekten realisiert.

Das generische Erzeugen von Klassen, die von LevelEditorRemoteRequest abgeleitet sind, dient dazu die Nachrichtenschnittstelle leicht erweiterbar zu halten. Man muss um eine neue Nachricht zu empfangen lediglich eine neue Klasse schreiben, die Parameter aufnimmt und dann eine bestimmte Funktion ausführt.

3.6 Kamera Steuerung

Dieser Abschnitt beschäftigt sich mit der Steuerung der Kamera. Es soll zunächst erklärt werden wie die Nebula-Kamerasteuerung aufgebaut ist. Anschließend wird auf die Implementierung der Kamera für das Level-Editor-Add-On und auf die unterschiedlichen Einstellungsmodi eingegangen.

Ein Kameraobjekt in Nebula wird benutzt um die Sichtweise auf die 3D-Szene einzustellen. An diesem Objekt lassen sich Position, Ausrichtung, Bildausschnitt, Öffnungswinkel und noch einige weitere Eigenschaften einstellen.

Die Steuerung eines Kameraobjekts erfolgt über eine GameEntity, mit einer CameraProperty. Das eigentliche Kameraobjekt befindet sich im Render-Thread und wird über Nachrichten von der CameraProperty gesteuert. Es kann mehrere solcher GameEntities geben. Über einen Manager wird bestimmt welche benutzt wird.

3.6.1 Implementierung der Level-Editor-Kamera

Für den Level-Editor wurde die LevelEditorCameraProperty von der CameraProperty abgeleitet. Die Erzeugung und Verwaltung der GameEntity übernimmt das LevelEditorCameraUtil. Dies ist eine Hilfsklasse, die auch dazu dient die Benutzereingaben für die Kamera zu kapseln und an die LevelEditorCameraProperty weiter zu geben.

Die Standardkamera mit der man sich im Level-Editor durch den 3D Raum bewegt, ist der Maya-Kamera nachempfunden. Diese Kamera war in Nebula bereits vorhanden und wurde für diverse andere Applikationen verwendet.

Um Benutzereingaben umzuwandeln und zu übertragen, und so das Verhalten der Maya-Kamera nachzuahmen, wird die Klasse MayaCameraUtil verwendet. Über diese Klasse kann mit der Kamera näher heran oder weiter weg gefahren werden. Die Kamera kann geschwenkt und um einen Fokuspunkt rotiert werden. Tastenkombination und Mausbewegungen sind im LevelEditorCameraUtil so gewählt, dass sie denen in Maya entsprechen.

Zusätzlich zum Maya-Modus für die Kamera, ist ein Modus angedacht, der die Kamera aus Sicht des Spielers widerspiegelt. Dieser Spielmodus sieht in der ersten Implementierung eine Kamera vor, die eine isometrische Ansicht auf den Level bietet und nicht frei drehbar ist. Diese Implementierung wird durch den aktuell entwickelten Spieltitel Drakensang-Online vorgegeben, der mit dieser Perspektive veröffentlich werden soll.

Die Kamera ist für andere Nebula-Spiele erweiterbar. Um neue Kameraperspektiven oder eine geänderte Steuerung einzubauen, muss lediglich von der LevelEditorCameraProperty geerbt werden und als CameraProperty eingetragen werden.

Abbildung 9: Kameraansichten Maya-Kamera(frei drehbar) vs. Spielperspektive (feste isometrische Ansicht)

4 Die Nebula3Tool Bibliothek

Dieses Kapitel stellt die Nebula3Tool-Bibliothek vor. Sie dient als Grundlage für die C#-Applikationen im „Nebula2 Toolkit for Maya“, indem sie Grundfunktionen zur Verfügung stellt, die diese Applikationen gemeinsam haben. In ihr werden Nebula-spezifische Eigenheiten gekapselt. Es sollen in diesem Kapitel die Remote-Klassen, der CommandManager und das Nebula3ViewControl vorgestellt und ihr Zusammenspiel im Level-Editor erläutert werden.

4.1 Die Remote-Klassen

Die Remote-Klassen bilden das Gegenstück zu den Remote-Klassen in den C++ Bibliotheken der Nebula3-Engine. Über diese Klassen wird eine TCP-Verbindung zu einer Nebula-Applikation aufgebaut, über die Daten versandt und empfangen werden können.

Die Verwaltung der Verbindung sowie den Versand der Daten übernimmt der Nebula2Controler. Zum Aufbau einer Verbindung müssen der Name der Applikation sowie der Empfangsport der Nebula-Applikation bekannt sein. Nachrichten werden grundsätzlich im XML-Format versandt. Zum Empfang der Nachrichten wird der Name des Befehls aus der Nachricht extrahiert und die entsprechende Klasse vom Typ CommandHandler generiert. Diese XML-Nachricht wird vom CommandHandler ausgewertet und ausgeführt. Hierbei ist zu beachten, dass CommandHandler threadsicher implementiert sein müssen, da die Nachrichten von einem anderen Prozess gesendet werden.

Der Level-Editor implementiert eine Reihe CommandHandler. Sie dienen dazu Transformationsdaten aus der Nebula-Applikation zurückzusenden, die Selektion der einzelnen Objekte zu verändern oder spielspezifische Daten, wie zum Beispiel eine Liste der verfügbaren Audio-Ereignisse an den Editor zu übermitteln.

4.2 Der CommandManager

Der CommandManager ist eine Klasse, über die die Ausführung von Befehlen verwaltet wird. Es wird mit Hilfe der Schnittstelle ICommand das Entwurfsmuster für Befehle [DP194] und das Memento-Muster [DP294] implementiert. Auf diese Weise wird es möglich, dass jede Änderung am Programm durch einen Befehl ausgeführt wird. Die Schnittstelle ICommand gibt vor, dass es für jeden Befehl eine Methode zum Ausführen (Execute), Rückgängig machen (Undo) und

erneutem Ausführen (Redo) geben muss. Zusätzlich speichert ein Befehl alle Daten die notwendig sind um den ursprünglichen Zustand wiederherzustellen.

Die Undo-Funktionalität wird benötigt um zu gewährleisten, dass der Anwender relativ sorglos mit dem Level-Editor arbeiten kann und nicht befürchten muss mit einem falschen Klick die Resultate seiner Arbeit zu vernichten.

Die Befehle, die für den Level-Editor implementiert wurden, beziehen sich ausschließlich auf das Bearbeiten der Daten. Das heißt Änderungen die sich ausschließlich auf die Benutzeroberfläche auswirken, werden nicht als Befehle gekapselt.

Der CommandManager bietet zwei Möglichkeiten auf die Ausführung von Befehlen zu reagieren. Die erste Möglichkeit ist die Implementierung einer Methode, die nach dem Ausführen eines Befehls aufgerufen wird.

Diese wird im Level-Editor implementiert und bei der Initialisierung des CommandManagers übergeben. Der Level-Editor benutzt diese Methode um die Aktualisierung der Daten über die Remote-Verbindung zu realisieren.

Die zweite Möglichkeit ist ein Notify-Ereignis das nach Ausführung eines Befehls gesendet wird. Diese Variante wird vom Level-Editor genutzt um Plug-ins zu informieren das sich Daten geändert haben. Es ist dem Plug-in-Programmierer überlassen in welcher Art darauf reagiert wird.

4.3 Das Nebula3ViewControl

Das Nebula3ViewControl ist ein C#-Benutzersteuerelement. Es ist in der Lage eine Nebula-Fensteranwendung zu starten. Diese wird über den Handle auf das Fenster in die Benutzeroberfläche integriert und kann über Betriebssystemfunktionen gesteuert werden.

Standardmäßig wird durch das Nebula3ViewControl versucht eine Remoteverbindung zum Erstellten Nebula-Fenster aufzubauen. Die Verbindung wird durch das Fenster automatisch konfiguriert.

Der Level-Editor benutzt dieses Element um sich als Nebula-Applikation drasaonline_client einzubetten. Über die Startargumente wird diese Applikation so konfiguriert, dass sie in einem Modus startet, der es ihr erlaubt als Level-Editor zu agieren.

5 Level-Editor Plug-in Struktur in C#

Eine Anforderung an den Level-Editor ist, dass er erweiterbar sein soll. Im Nebula3 Add-On soll dies durch Vererbung, im C# Rahmenwerk über eine Plug-in Struktur erreicht werden. Dieses Kapitel beschreibt diese Plug-in-Struktur. Diese ist fest mit der generellen Programmstruktur verwoben, deshalb wird dies zuerst beschrieben. Danach werden die Plug-in-Schnittstellen einzeln betrachtet und erläutert. Darauf folgt ein Überblick über die implementierten Plug-ins.

5.1 Programmaufbau der C# Applikation

Dieser Abschnitt beschreibt die generellen Programmteile, die für das C#-Rahmenwerk essentiell sind.

Bei einer groben Untergliederung benötigt man ein System, das die Level-Daten verwaltet, eine grafische Benutzeroberfläche und Werkzeuge, die man über diese Benutzeroberfläche bedienen kann um die Daten zu bearbeiten.

Es gibt einen Datenkern, der die Level-Daten in einer definierten Struktur verwaltet und Operationen zur Verfügung stellt, um diese Daten zu verändern. Der Datenkern kann Daten speichern und laden. Außerdem ist er der einzige Ort, an dem die Daten gehalten werden, die dann als Level-Daten abgespeichert werden. Abbildung 10 zeigt den Informationsfluss von

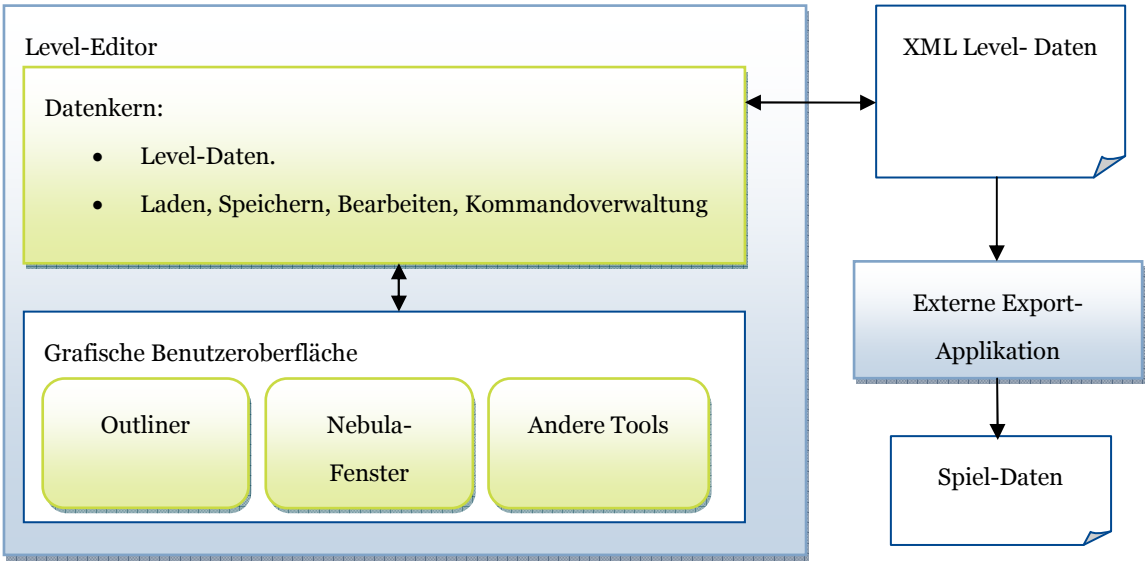


Abbildung 10: Datenfluss von Benutzeroberflächen über Datenkern bis hin zu fertigen Spieldaten

Level-Daten, wie er sich zwischen dem Kern, der Benutzeroberfläche und externen Applikationen darstellt.

Die Benutzeroberfläche beinhaltet das Fenster in dem die Nebula3-Applikation läuft. Zu diesem Fenster gehören Menüs und Funktionen um das Fenster zu steuern. Ein weiterer Teil der Benutzeroberfläche ist der sogenannte Outliner. Er dient im Gegensatz zur Nebula-Applikation nicht dazu die Level-Daten als 3D-Visualisierung anzuzeigen, sondern stellt eine Übersicht der gesamten Daten dar. Hierfür wurde eine Baumstruktur gewählt. Der Editor beinhaltet noch andere Werkzeuge zum Bearbeiten von Daten, die sich auch in die Benutzeroberfläche eingliedern. Diese werden als Tools bezeichnet.

Die Daten des Level-Editors werden als XML-Dokument (siehe Anhang 1) gespeichert. Diese Dokumente enthalten sowohl Daten, die nur für den Level-Editor relevant sind, als auch die Daten, die für die Nebula3-Spiele benötigt werden. Eine externe Export-Applikation ist in der Lage die Daten in die Formate zu bringen, die zu den jeweiligen Spielen passen.

5.2 Container und Plug-ins

In diesem Abschnitt wird die Gliederung in Hauptapplikation und Plug-ins beschrieben.

Es gibt Programmteile, die als Plug-ins implementiert werden, aber zwingend vorhanden sein müssen um eine funktionstüchtige Level-Editor Applikation zu erhalten. Das sind der Datenkern, das Nebula-Fenster und der Outliner.

Die Hauptapplikation beinhaltet eine Containerklasse für den Datenkern, eine für das Nebula-Fenster, eine für den Outliner und eine für diverse Tools. Mit Ausnahme des Tool-Containers kapselt jede dieser Klassen jeweils ein Plug-in. Es gibt in der Hauptapplikation also jeweils eine

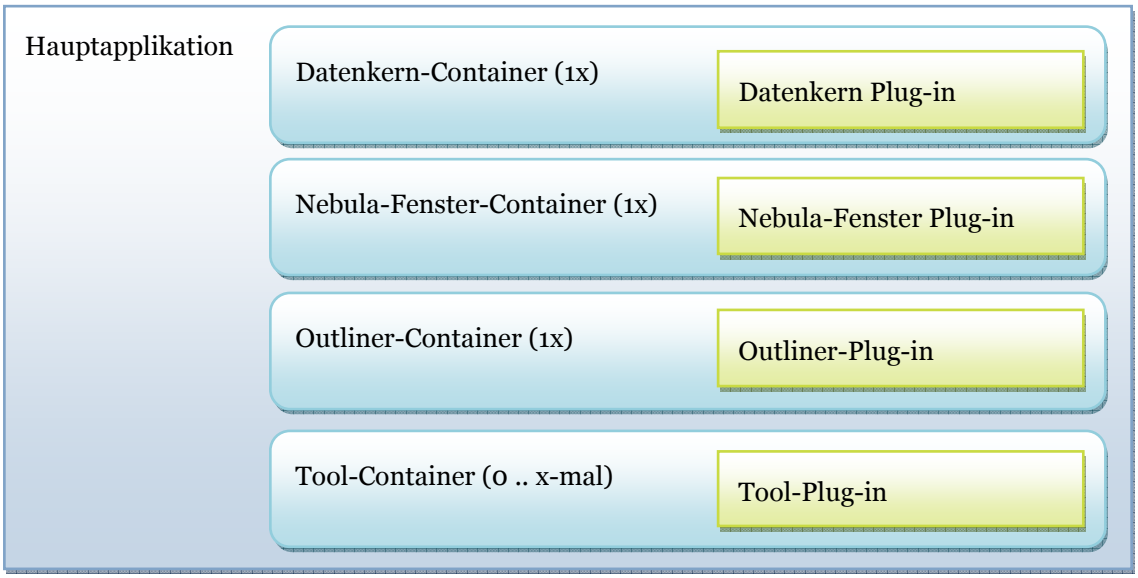


Abbildung 11: Einbettung der Plug-ins in die jeweiligen Containerklassen

Instanz der Containerklasse für den Datenkern, den Outliner und das Nebula-Fenster und eine beliebige Anzahl an Containern für Tools. Folgende Klassen in der Level-Editor Applikation setzen diese Struktur um.

Tabelle 1: Plug-in-Container und Klassen

Container	Klasse
Datenkern-Container	CoreDataContainer
Outliner-Container	OutlinerContainer
Nebula-Fenster-Container	ViewerContainer
Tool-Container	ToolContainer

5.3 Plug-in Schnittstellen

Im folgenden Abschnitt sollen die Plug-in Schnittstellen, sowie deren Anbindung an das Hauptprogramm, beschrieben werden. Die Plug-in Schnittstellen sind in C# als Interface definiert. An einer Schnittstelle gibt es drei Arten von Methoden.

Die Erste Art wird vom Hauptprogramm aufgerufen. Über diese Methoden können bestimmte Eigenschaften von Plug-ins abgefragt werden. Zum Beispiel kann so eine Liste von Menü-Objekten angefordert werden, die dann zur Benutzeroberfläche des Hauptfensters hinzugefügt werden.

Die zweite Art sind die EventHandler. Sie werden immer dann aufgerufen, wenn ein bestimmtes Ereignis eintritt, zum Beispiel das Drücken einer Taste. In einem EventHandler kann auf diese Ereignisse reagiert werden.

Die dritte Art von Methode übergibt die Referenz auf ein Objekt. Dieses Objekt ist vom Typ ICoreCommunication und ermöglicht es dem Plug-in auf die Level-Daten und Funktionen im Hauptprogramm zu zugreifen. ICoreCommunication ist eine Schnittstelle und wird vom CoreDataContainer implementiert.

Jede Schnittstelle erbt von anderen Schnittstellen. Alle Plug-ins müssen die Schnittstellen IDisposable, IUpdateListener und IKeyReceiver implementieren. Die Implementierung der C# Schnittstelle IDisposable soll sicher stellen, dass nicht verwaltete Ressourcen des Plug-ins freigegeben werden, wenn dieses entladen wird. Die Schnittstelle IUpdateListener, gehört zu der Nebula3Tools-Bibliothek. Die Änderung der Level-Daten erzeugt ein Ereignis auf das, durch

Implementierung des EventHandlers dieser Schnittstelle, reagiert werden kann. Die IKeyReceiver Schnittstelle beinhaltet den EventHandler für Tastatureingaben.

Alle Methoden, über die man auf Benutzeroberflächenelemente zugreifen kann, wurden in die Schnittstelle IUIElements ausgelagert. Bis auf das Datenkern-Plug-in, muss diese Schnittstelle von allen implementiert werden.

Gemeinsam haben alle Schnittstellen die Initialize Methode, welche genutzt wird um das Plug-in zu initialisieren, nachdem das ICoreCommunication-Objekt verfügbar ist.

Tabelle 2: Implementierte Plug-in-Interface-Namen

Plug-in	Interface
Datenkern	ICoreData
Nebula-Fenster	IViewer
Outliner	IOutliner
Tool	ITools

5.3.1 Die Datenkern Schnittstelle

Diese Schnittstelle wird dazu verwendet, das Level-Daten-Format zu erweitern. So ist es zum Beispiel möglich über die Methode CreateNode zusätzliche Attribute an die Daten-Knoten anzuhängen und damit die Daten speziell an ein Projekt anzupassen.

5.3.2 Die Schnittstelle für das Nebula-Fenster

Das Plug-in, das diese Schnittstelle implementiert, hat die Aufgabe ein Nebula-Fenster einzubetten. Über diese Schnittstelle ist es dem Hauptprogramm möglich die Daten für die Remote-Verbindung zu erhalten, die das Hauptprogramm dann herstellt. Außerdem kann auf diese Weise auf die Komponente der Benutzeroberfläche, die das Nebula-Fenster enthält, zu gegriffen werden. Dieses C#-Steuerelement[24] wird in das Hauptfenster integriert.

5.3.3 Die Outliner-Schnittstelle

Diese Schnittstelle erfordert die Implementierung eines Outliners, der die Level-Daten darstellt und einer Übersicht, die die Eigenschaften einzelner Knoten darstellt. Diese beiden Steuerelemente werden im Hauptfenster eingebettet.

[24] „Steuerelemente (oder Designelemente der Benutzeroberfläche) sind die sichtbaren Komponenten einer Anwendung. Steuerelemente (wie Schaltflächen oder Listen mit auswählbaren Elementen) ermöglichen den Benutzern die Interaktion mit der Anwendung…“ [Mic10]

5.3.4 Die Tool-Schnittstelle

Über diese Schnittstelle können verschiedene Arten von Funktionen und Fenster eingefügt werden. Ein Tool-Plug-in muss nicht auf den Level-Daten operieren. Die Methode GetForms ist so zu implementieren, dass sie alle Fenster, die direkt nach dem Laden des Tools angezeigt werden sollen, zurückliefert.

5.3.5 Die Schnittstelle zum Hauptprogramm

ICoreCommunication ist wie bereits beschrieben die Schnittstelle zum Hauptprogramm, die sich alle Plug-ins teilen.

Über die Eigenschaft LevelData steht dem Plug-in ein direkter Zugriff auf die Level-Daten-Struktur und all ihrer öffentlichen Methoden zur Verfügung. Auch das Selektieren und Erstellen von Daten-Knoten erfolgt über diese Schnittstelle.

Über Methoden wie Redo, Undo und GetCommandInfos kann indirekt auf den CommandManager zugegriffen werden. Es gibt Methoden wie zum Beispiel SendCommand, die benutzt werden um dem verbundenen Nebula-Fenster Befehle zu senden.

5.4 Das Laden der Plug-ins

Es wurden Standard-Plug-ins programmiert, die alle in einer Bibliothek liegen. Dieser Abschnitt beschreibt wie sie geladen werden.

Sofern keine andere Bibliothek angegeben wurde, wird die DefaultEditorPlugin.dll bei Programmstart geladen. C#-Reflektion[25] ermöglicht es sie zur Laufzeit zu laden und die Plug-in-Objekte zu generieren.

Die einzelnen Plug-ins in der Bibliothek werden über C#-Klassen-Attribute [Mic210] identifiziert. Attribute verknüpfen Deklarationsinformationen mit C#-Code (Typen, Methoden, Eigenschaften usw.). Die Plug-in Attribute sind als Klassen definiert. Für jeden Plug-in-Typ gibt es eine Attributklasse. Um als ein bestimmtes Plug-in identifiziert zu werden, muss eine Klasse eines dieser Attribute besitzen.

5.5 Standard-Plug-ins

In diesem Abschnitt werden einzelne Plug-in-Implementierungen vorgestellt.

[25] „Bei der Reflektion werden Objekte (vom Typ Type) bereitgestellt, die Assemblys, Module und Typen kapseln. Mithilfe von Reflektion können Instanzen von Typen dynamisch erzeugt, Typen an ein vorhandenes Objekt gebunden und Typinformationen von vorhandenen Objekten abgefragt werden. Ebenso können die Methoden vorhandener Objekte aufgerufen werden, und es kann auf ihre Felder und Eigenschaften zugegriffen werden.“ [Mic110]

5.5.1 ViewerDefault

Dieses Plug-in enthält ein Nebula3ViewControl, welches das Nebula-Fenster einbettet und zusätzliche Funktionen zur Verfügung stellt. Hier soll die Benutzerschnittstelle vorgestellt und kurz erläutert werden welche Funktionalität sich dahinter versteckt.

Abbildung 12: Benutzeroberfläche des ViewerDefault mit Werkzeugleiste und eingebettetem Nebula-Fenster

Im Nebula3ViewControl wird der Drakensang-Online-Client als Prozess gestartet. Über Kommandozeilenargumente wird der Port angegeben, über den per Remote-Controls kommuniziert wird. Die eigentliche Verbindung wird erst nach dem Laden des Plug-ins im Hauptprogramm initiiert.

Die in Abbildung 12 abgebildete obere Symbolreihe, ist die Werkzeugleiste. Die Symbole dieser Leiste wurden von der Grafikabteilung eigens für den Level-Editor erstellt. Über die ersten drei Knöpfe kann zwischen den Transformationsformen Rotieren, Positionieren und Skalieren umgeschaltet werden. Die vier folgenden Knöpfe schalten die Art der Platzierung auf dem Gitter um (siehe Kapitel Die wichtigste Klasse der LevelEditorFeatureUnit ist der LevelEditorManager. Er ist die zentrale Verwaltungsklasse für das Add-On. Durch ihn werden die Liste der angezeigten Objekte sowie eine Liste der selektierten Objekte verwaltet. Er koordiniert den Input und leitet diesen an das PlacementUtil und die LevelEditorCameraProperty weiter. Deren Funktionalität wird in den folgenden Abschnitten erläutert wird. Über den LevelEditorManager werden Nachrichten erstellt, die an das C# Level-Editor Framework gesendet werden. Außerdem stellt er Methoden zur Verfügung die es erlauben die angezeigten Objekte und die Selektion zu verändern.

3.4 Echtzeit Platzierung in Nebula3 – Das PlacementUtil). Es wird umgeschaltet zwischen freiem Platzieren, auf dem Gitter platzieren (SnapToGrid), platzieren auf den Höhenebenen (SnapToGround) und platzieren auf den Kanten des Gitters(SnapToEdge).

Über die folgenden Schaltflächen kann die Maschenweite und die Höhe für das Gitter festgelegt werden. Es folgen weitere Knöpfe um zwischen Gruppenrotation und das Rotieren im 90° Winkel umzuschalten, sowie die Knöpfe um den Kameramodus von Maya-Kamera oder Spiel-

Kamera zu wählen. Alle diese Funktionen werden als Befehl an die eingebettete Applikation versandt und von ihr ausgewertet.

5.5.2 OutlinerDefault

Dieses Plug-in besteht aus zwei Teilen, dem Outliner und dem Eigenschaften-Fenster. Sie sind angelehnt an die äquivalenten Benutzersteuerungselemente, die in TorchED verwendet wurden.

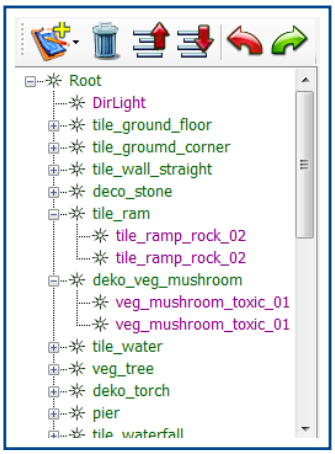


Abbildung 13: Outliner-Fenster des Default-Outliner

Der Outliner besteht aus einem abgewandelten TreeView[26]-Steuerelement, das in der Lage ist mehre Knoten gleichzeitig auszuwählen. Der Baum bildet die Datenstruktur ab, so wie sie vom LevelData-Objekt über die Plug-in Schnittstelle definiert wird.

Die Operationen, die es ermöglichen mit dem Outliner zu arbeiten, sind entweder über Tastenkombinationen, die Symbolleiste oder ein Kontextmenü verfügbar. Kopieren und Einfügen ist zum Beispiel durch Drücken der Taste Steuerung und ziehen der selektierten Elemente auf einen anderen Knoten möglich. Es kann aber auch per Rechtsklick über das Kontextmenü erreicht werden. Die Knöpfe in der Symbolleiste, die in der Abbildung 13 zu sehen sind, führen Standard Funktionen aus, wie Rückgängig, Wiederholen, Knoten Entfernen, Knoten Hinzufügen und Knoten Verschieben.

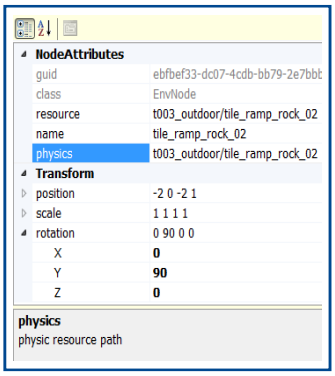


Abbildung 14: Eigenschaftenfenster des Default-Outliner

Die Attribute eines selektierten Knotens können im Eigenschaftenfenster betrachtet und wenn sie nicht schreibgeschützt sind, bearbeitet werden. Das verwendete PropertyGrid-Steuerelement[27] hat den Vorteil, dass man für Attribute, die manuell nur umständlich zu bearbeiten wären, spezielle Klassen anfügen kann, um die Bearbeitung zu erleichtern. Im Prinzip kann jeder Knoten über dieses Attributfenster vollständig bearbeitet und platziert werden.

[26] Ein TreeView-Steuerelement zeigt eine hierarchische Auflistung seiner Elemente an, die in einer Baumstruktur angeordnet sind. [Mic310]
[27] „Das PropertyGrid-Steuerelement zeigt Eigenschaften für beliebige Objekte oder Typen an und ruft die Eigenschaften von Elementen vorwiegend unter Verwendung von Reflektion ab.“ [Mic02]

5.5.3 Grafik Browser und Spielobjekt Browser

Einer der ersten Schritte um ein Objekt in einem Level zu platzieren, ist es dieses auszuwählen. Der CategoryBrowser und der GameObjectBrowser implementieren die dafür notwendigen Funktionen.

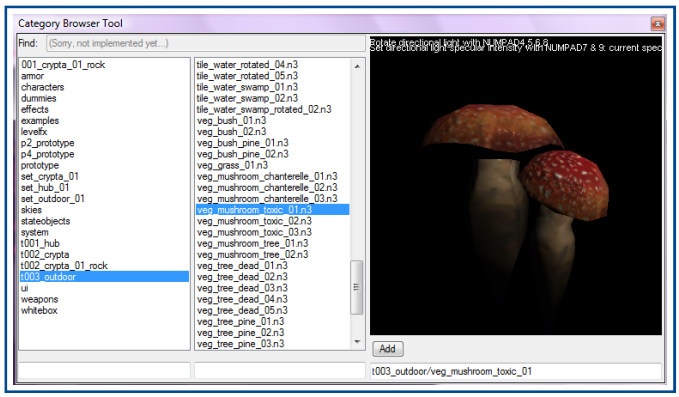


Abbildung 15: Fenster des CategoryBrowser mit Dateiselektion und Vorschaufenster

Der CategoryBrowser befindet sich in einem eigenen Fenster, das über das Menü aufgerufen wird. Seine Benutzeroberfläche besteht aus einem Datei-Browser und einem Vorschaufenster. Um die Suche nach Modellen zu erleichtern, ist der Datei-Browser auf den Ordner festgelegt, in dem in Nebula-Projekten die Grafikressourcen liegen. Das Vorschaufenster ist in der Lage den Inhalt der Grafikdateien anzuzeigen Es ist ein eingebettetes Nebula-Fenster in der die n3Viewer-Anwendung läuft. Wie schon in Kapitel 2.1.3 „Nebula Toolkit for Maya“ erwähnt wird, dient es dazu eine Vorschau auf die exportierten Nebula-Grafikdaten zu geben. Die Kommunikation zwischen dem C#-Fenster und dem n3Viewer funktioniert über eine TCP-Verbindung, die das RemoteInterface des n3Viewers anspricht. Sie ist jedoch wesentlich einfacher und beschränkt sich auf das Senden kurzer Befehle.

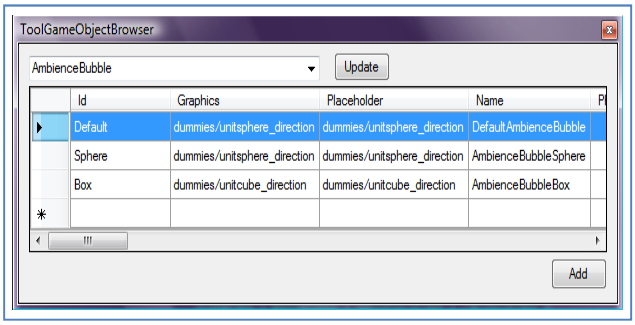


Abbildung 16: Der GameObjektBrowser

Über den Category Browser können nur Grafikobjekte eingefügt werden.

Der GameObjectBrowser dient dazu andere Spielobjekte einzufügen.

Zukünftig sollen beide Objekttypen jedoch über eine einheitliche Schnittstelle verwaltet werden.

5.6 Verwaltung der Daten – Die LevelData-Klasse

Dieser Abschnitt beschäftigt sich mit der Klasse, in der die Level-Daten gehalten werden. Als erstes wird die Struktur erläutert und anschließend werden die Operationen zusammengefasst, die auf dieser Datenstruktur möglich sind. Zusätzlich wird erläutert, wie diese Klasse mit dem CommandManager zusammenspielt.

Die Klasse LevelData enthält eine Datenstruktur von Typ ILevelNode. ILevelNode ist der Name einer Schnittstelle, die von der internen Klasse LevelNode in LevelData implementiert wird.

LevelNode ist ein Knoten in einer Baumstruktur, der eine beliebige Anzahl an Kind-Knoten haben kann. LevelData enthält eine Referenz auf die Wurzel des Baumes. Jeder ILevelNode besitzt eine beliebige Anzahl von Attributen vom Typ ILevelNodeAttr. Die generische Klasse [28]LevelNodeAttr<T> ist die Implementierung für ILevelNodeAttr.

Zusätzlich zur Baumstruktur enthält LevelData noch eine Sammlung von Referenzen auf Knoten des Baumes. Dies sind die selektierten Knoten, die über den Outliner oder über Anklicken im Nebula-Fenster ausgewählt werden können.

5.6.1 Attributschnittstelle - ILevelNodeAttr

Die Schnittstelle ILevelNodeAttr enthält zwei Eigenschaften. Die Erste ist *Name*, der Name des Attributes. Die Zweite ist *Value*, der ein Objekt beliebigen Typs. Durch eine Instanziierung der generischen Klasse wird der Typ festgelegt. Die Methode GetTypValue() liefert den Typ.

5.6.2 ILevelNode

Die Schnittstelle ILevelNode erbt von der generischen C#-Schnittstelle IList. Das heißt die Klasse LevelNode muss zusätzlich zu den in ILevelNode aufgeführten Methoden, Listen Methoden implementieren, die Zugriff auf ihre Kind-Knoten erlauben (Hinzufügen, Entfernen, Einfügen, Contains, Count usw.).

Jeder ILevelNode enthält eine Referenz auf seinen Eltern-Knoten, den Parent-Knoten. Dies hilft bei der schnellen Navigation durch die Baumstruktur.

5.6.3 LevelData Methoden

LevelData besitzt eine statische Methode CreateLevelNode mit der ein ILevelNode erschaffen wird, der vom Typ LevelNode ist. Außerdem gibt es Suchfunktionen über die man nach Knoten mit bestimmten Attributen suchen kann. Einige Methoden, die Level-Daten verändern, sind über die Schnittstelle ICoreCommunication erreichbar.

---

[28] „Generika führen das Konzept der Typparameter in .NET Framework ein. Dadurch wird es möglich, Klassen und Methoden zu entwerfen, in denen die Angabe eines oder mehrerer Typen verzögert wird, bis die Klasse bzw. Methode durch Clientcode deklariert und instanziiert wird.“ [Mic410]

5.6.4 LevelData und CommandManager

Alle datenverändernden Methoden haben gemeinsam, dass sie eng mit dem CommandManager und über diesen mit dem RemoteControl verwoben sind. Jede Methode oder Eigenschaft, die eine Änderung an der Baumstruktur, den Attributen der Knoten oder der Selektion der Knoten hervorruft, erstellt einen Command. Dieser Befehl wird der dem CommandManager übergeben. Der CommandManager speichert ihn und führt ihn aus. Jeder Befehl kann über den CommandManager rückgängig gemacht werden, um den Zustand vor dem Anwenden des Befehls wiederherzustellen.

Das Ausführen eines Commands im CommandManager ruft eine Methode im Remote System auf. Je nachdem welche Art von Befehl ausgeführt wurde, schickt diese Methode Aktualisierungsdaten an die Applikation die im Nebula-Fenster läuft.

5.7 Offene Probleme

Folgende Probleme haben sich bei der Arbeit mit den existierenden Nebula3Tools-Klassen ergeben, für die eine Lösung noch aussteht.

Es wurden Funktionen aus den Nebula3Tools verwendet, wie zum Beispiel der CommandManager oder die RemoteControl Klassen. Diese Klassen wurden nicht für eine Kapselung ausgelegt. Sie sind öffentlich zugängliche Singleton-Instanzen, die von überall aus dem Code erreichbar sind. Das heißt die Plug-ins könnten auf sie zugreifen. Da die Plug-ins nur über Schnittstellen kommunizieren sollten, ist das kein wünschenswerter Zustand.

Aus den Singleton-Instanzen resultiert ein weiteres Problem. Kein Plug-in darf diese Instanzen benutzen, obwohl sie Muster implementieren, die man wiederverwenden sollte. Als Beispiel sei der CommandManager genannt, der die Funktion zum Rückgängigmachen der letzten Operationen anbietet. Eine Operation, die man durchaus auch in Tools benutzen möchte, welche die Level-Daten nicht verändern. Das würde aber zum Konflikt mit den Operationen auf den Daten führen.

Für diese Problematik gibt es zwei Lösungen, die allerdings nicht im Rahmen in dieser Arbeit angegangen wurden. Die erste Lösung wäre eigene Klassen zu schreiben, die keine Singletons sind. Der Nachteil wäre, dass man in diesem Fall die gleiche Funktionalität an mehreren Stellen implementiert und den Code dann auch an unterschiedlichen Stellen warten muss. Außerdem wäre es verboten diese Nebula3Tools Klassen für den Level-Editor und zukünftige Plug-ins zu verwenden.

Die zweite Lösung wäre die Nebula3Tools Klassen zu überarbeiten. Der Nachteil hierbei wäre, dass man auch alle anderen „Nebula2 Toolkit for Maya“ C#-Applikationen, die auf dieser

Bibliothek aufbauen, anpassen müsste. Was einen entsprechenden aber einmaligen Programmieraufwand mit sich bringen würde.

Weiterhin hat sich herausgestellt, dass es bei manchen Funktionen, die in die Plug-ins eingebaut wurden, es wünschenswert wäre, wenn man Daten direkt von der eingebetteten Nebula-Applikation abrufen könnte. So hat eine Nebula-Applikation zum Beispiel Audio-Ressourcen geladen. Um eine Liste der Musikstücke zu bekommen, müsste eine Nachricht an die Applikation gesendet werden, die daraufhin die Liste der Musikstücke als Antwort zurückgesendet.

Als Lösungsansatz wäre eine Erweiterung der Manager-Klasse vorstellbar, die die Anfragen inklusive einer Methode entgegennimmt, welche dann ausgeführt wird, wenn eine Antwort von der Nebula-Applikation eintrifft.

6 Tests und Evaluation

Dieses Kapitel befasst sich mit der Qualitätssicherung und der Evaluation des Level-Editors. Zuerst soll exemplarisch anhand des Level-Editor-Projektes erklärt werden wie einzelne Aufgaben erstellt und verwaltet wurden. Es soll beschrieben werden wie getestet wurde und wie Entwicklungszyklen aus planen, implementieren, testen und evaluieren entstanden sind.

Einem Projekt geht eine Planungs- und Evaluationsphase voraus. Wenn diese abgeschlossen ist, kann das Projekt in größere Aufgabenblöcke für die Implementierung unterteilt werden. Das Level-Editor- Projekt wurde in folgende Blöcke unterteil:

Tabelle 3: Aufgabenblöcke für die Basisimplementierung des Level-Editors

Aufgabenblock	Beschreibung
C# Plug-in Framework	Beinhaltet die Implementierung des C# Rahmenwerks um Plug-ins zu laden, sowie die Datenbasis auf die von den Plug-ins zugriffen wird.
C++ Level-Editor Add-On	Beinhaltet das Level-Editor Add-On für Nebula3.
Remote Anbindung im C# Framework	Kommunikationsschnittstelle zum eingebetteten Nebula-Fenster.
Remote Anbindung im Nebula3 Add-On	Kommunikationsschnittstelle vom Level-Editor Add-On zur Außenwelt.
Objekt Positionierung	Objektpositionierung auf einem virtuellen Gitter.
Undo Funktionalität	Management-Befehle die Daten verändern und zwar auf eine Weise, das man diese Änderungen wieder rückgängig machen kann.
Laden/Speichern	Laden und Speichern der Daten in Dateien.
Outliner Plug-in	Ein Plug-in, das es ermöglicht alle Daten in einer übersichtlichen Struktur zu sichten und zu verändern.
Viewer Plug-in	Einbindung des Nebula-Fensters.

Dies sind die großen Blöcke, die es zu implementieren galt, bevor das Programm von anderen Abteilungen getestet werden konnte. Zusätzlich mussten noch unzählige geringer priorisierte Funktionen, Wünsche und Anregungen berücksichtig werden. Dies war notwendig um zukünftige Erweiterungen nicht zu erschweren bzw. unmöglich zu machen.

Alle geplanten Funktionen wurden als User-Stories umschrieben und in die Aufgaben-Datenbank übertragen, die auch als Bug-Datenbank bezeichnet wird. Es gibt drei verschiedene Kategorien von Einträgen in dieser Datenbank, die User-Story, die Aufgabe und den Bug.

Eine User-Story, ist eine Aufgabenstellung, die Funktionalität in ein bis zwei umgangssprachlich formulierten Sätzen umschreibt. In Besprechungen, die im Abstand von ca. 14 Tagen stattfinden, werden die User-Stories in Bezug auf Zeitaufwand und Nutzen neu bewertet. Sie werden mit einer Periodisierung versehen. Es werden stets die User-Stories mit dem größten Nutzen ausgewählt und dann in den kommenden zwei Wochen bearbeitet.

Ist eine User-Story komplexer, sollte sie in Aufgaben aufgeteilt werden. Aufgaben sind technisch genaue Beschreibungen der Problemstellung und können Anweisungen enthalten, die beschreiben auf welche Art das Problem zu lösen ist.

Die letzte Kategorie, die noch nicht beschrieben wurde, ist der Bug. Als Bug werden alle Arten von Fehlern bezeichnet, vom Programmabsturz, über Fehlverhalten von Funktionen, bis hin zu kleinen Unschönheiten. Bugs können und sollen von jedem geschrieben werden dem sie auffallen. Sie müssen parallel zu den User-Stories bearbeitet werden. Man sollte ein bis zwei Tage im zweiwöchigen Bearbeitungszyklus für das Korrigieren von Fehlern einberechnen.

Nachdem die Grundlagen für das Testen beschrieben wurden, soll auf den Testzyklus genauer eingegangen werden.

Das Testen neuer Funktionen erfolgt in mehreren Schritten. Zuerst durch den Programmierer, der sicherstellen muss, dass jede neue Funktion in der Applikation wie angedacht eingebaut ist und funktioniert. Dieses Testen kann durch selbstgeschriebene Testklassen erfolgen. Bei komplexeren Szenarien lassen sich Fehler häufig nur durch Log-Ausgaben oder das schrittweise Durcharbeiten der Instruktionen im Quellcode ausfindig machen.

In der zweiten Stufe wird das für den Programmierer augenscheinlich funktionierende Programm der Testabteilung übergeben. Die Testabteilung führt vorwiegend Tests durch, die Benutzereingaben erfordern. Wenn es sich um eine neue Funktion handelt, die eingebaut wurde, so überprüfen sie ob die gelieferte Funktion der User-Story oder der Aufgabe entspricht. Werden die Erwartungen nicht erfüllt oder sind im Zusammenhang mit der neuen Funktionalität Fehler auftreten, so wird die Aufgabe an den Programmierer zurückgegeben.

Die Evaluation der Funktionen erfolgt im Idealfall nachdem die Testabteilung das Programm auf Fehler überprüft hat. Im Fall des Level-Editors wurde die Evaluation von der Level-Design-Abteilung durchgeführt. Die Änderungs- und Erweiterungswünsche wurden in Form von User-Stories in die Bug-Datenbank eingetragen. Diese werden in diesem Zyklus weiter bearbeitet. So wird der Level-Editor Schritt für Schritt weiter entwickelt, auch über den Umfang dieser Arbeit hinaus.

Am Beispiel der User Story mit der Anforderung die Objektpositionierung zu ermöglichen, soll der Bearbeitungszyklus beschrieben werden.

Die Implementierung wird in Kapitel Die wichtigste Klasse der LevelEditorFeatureUnit ist der LevelEditorManager. Er ist die zentrale Verwaltungsklasse für das Add-On. Durch ihn werden die Liste der angezeigten Objekte sowie eine Liste der selektierten Objekte verwaltet. Er koordiniert den Input und leitet diesen an das PlacementUtil und die LevelEditorCameraProperty weiter. Deren Funktionalität wird in den folgenden Abschnitten erläutert wird. Über den LevelEditorManager werden Nachrichten erstellt, die an das C# Level-Editor Framework gesendet werden. Außerdem stellt er Methoden zur Verfügung die es erlauben die angezeigten Objekte und die Selektion zu verändern.

3.4 Echtzeit Platzierung in Nebula3 – Das PlacementUtil beschrieben.

In der ersten Implementierung wurden Objekte flüssig im Raum bewegt, erst beim Loslassen der Maus wurde das Objekt auf die neue Position auf dem Gitterraster platziert. Das heißt der Benutzer konnte das Objekt wie im Freien Modus ziehen, die Position, die es auf dem Gitter annehmen soll, wurde durch einen Würfel simuliert, der von Gitterknoten zu Gitterknoten sprang. Das Objekt selbst wurde fließend bewegt. Diese Art der Platzierung funktionierte und wurde durch die Testabteilung abgenommen.

Bei der Evaluation dieser Funktion wurde es als hinderlich empfunden, dass das Objekt nicht sofort auf die Position gesetzt wird. Diese Rückmeldung wurde als neue User-Story in die Bug-Datenbank aufgenommen und in einem weiteren Bearbeitungsschritt geändert.

In Tabelle 4 im Anhang sind exemplarisch für die gesamte Evaluation des Level-Editors alle Wüsche, Anregungen und Fehler in Bezug auf Objektpositionierung aufgelistet, die während der ersten Evaluation gesammelt wurden. Die Übersicht stammt direkt aus der Bug-Datenbank und es werden nur die relevanten Spalten angezeigt.

Einige dieser User-Stories wurden bereits umgesetzt, dazu gehört die Voreinstellung des Snap-Modus und dass die Werte im Editor in Grad eingegeben werden. Die Rotation im festen Winkel wurde ebenfalls fertiggestellt. Andere Aufgaben bedürfen noch Vorarbeit, zum Beispiel müssen

um die Handles einstellen zu können, noch Änderungen an den Basis-Klassen des PlacementUtil vorgenommen werden.

7 Fazit

Das Thema dieser Arbeit lautete „Erstellung eines Echtzeit Level-Editors für die Nebula3-Engine.“ Es wurde ein Editor gebaut mit dem sich Level-Dateien erstellen und bearbeiten lassen. Der Editor bettet das Spiel (eine Nebula-Applikation) ein. Dementsprechend ist die dargestellte Version nicht nur eine Vorschau wie es im Spiel aussehen könnte. Anders als im Torchlight-Editor oder in Maya sind Animationen und Effekte an Grafikobjekten sichtbar.

Die Zeitplanung um einen Level-Editor für die Nebula-Engine zu bauen, war von Anfang an für eine größere Zeitspanne ausgelegt, als sie im Rahmen dieser Arbeit zur Verfügung stand. Das Ziel war es ein solides Rahmenwerk zu schaffen.

Dies ist in vollen Umfang gelungen. Durch Zusammenarbeit mit Level-Design- und Grafikabteilung konnte der Funktionsumfang schon in diesem ersten Schritt besser an die Bedürfnisse der späteren Benutzer angepasst werden.

8 Ausblick

Die Ergebnisse der Evaluation haben gezeigt, dass es noch viele kleine und große Wünsche gibt, die in der bisherigen Planung nicht berücksichtigt wurden. Der Level-Editor wird in seinen Standard-Plug-ins weiter entwickelt werden. Außerdem werden kontinuierlich weitere Werkzeuge hinzugefügt, die speziell für bestimmte Arten von Spielen benötigt werden.

Ein weiterer spannender Aspekt wird die Verwendung des Level-Editors in anderen Nebula-Projekten. Die Integration erfolgte bisher nur in der Drakensang-Online-Client Applikation. Die Nebula-Klassen für den Level-Editor wurden zwar auf Erweiterbarkeit konzipiert, aber ein Test am „lebenden“ Projekt wirft meist neue Herausforderung auf.

9 Literaturverzeichnis

[Aut09] Autodesk. (2009). *Maya 2009*.
http://www.autodesk.de (Abgerufen am 2. November 2010).

[DP194] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994).
Command. In *Design Pattern – Elements of Reusable Object-Oriented Software* (S. 233-242).
Addison Wesley.

[DP294] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994).
Memento. In *Design Pattern – Elements of Reusable Object-Oriented Software* (S. 283-291).
Addison Wesley.

[DP394] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994).
Singleton. In *Design Pattern – Elements of Reusable Object-Oriented Software* (S. 127-134).
Addison-Wesley.

[Mic210] Microsoft Corporation. (2010).
MSDN Library, Attribute (C#-Programmierhandbuch).
http://msdn.microsoft.com/de-de/library/default.aspx (Abgerufen am 12. Oktober 2010).

[Mic410] Microsoft Corporation. (2010).
MSDN Library, Generika (C#-Programmierhandbuch).
http://msdn.microsoft.com/de-de/library/default.aspx (Abgerufen am 12. Oktober 2010).

[Mic110] Microsoft Corporation. (2010).
MSDN Library, Reflektion (C#-Programmierhandbuch).
http://msdn.microsoft.com/de-de/library/default.aspx (Abgerufen am 12. Oktober 2010).

[Mic10] Microsoft Corporation. (2010).
MSDN Library, Steuerelemente (Übersicht).
http://msdn.microsoft.com/de-de/library/default.aspx (Abgerufen am 12. Oktober 2010).

[Mic310] Microsoft Corporation. (2010).
MSDN Library, TreeView-Klasse.
http://msdn.microsoft.com/de-de/library/default.aspx (Abgerufen am 12. Oktober 2010).

[Mic02] Microsoft Corporation, Rideout, Mark. (2002).
MSDN Library, Optimale Nutzung des PropertyGrid-Steuerelements in .NET Framework.
http://msdn.microsoft.com/de-de/library/default.aspx (Abgerufen am 10. September 2010).

[RAD10] RAD Game Tools. (2010). *Bink Video*.
http://www.radgametools.com/bnkmain.htm (Abgerufen am 7.Oktober 2010).

[Run09] Runic Games, Inc. (2009). *TorchED*.
http://www.torchlightgame.com (Abgerufen am 22. Oktober 2010).

[Wei107] Weißflog, A. (2007). *Nebula3 Architechture Overview*.
http://flohofwoe.blogspot.com/2007/01/nebula3-architecture-overview.html
(Abgerufen am 16. September 2010).

[Wei207] Weißflog, A. (2007). *Nebula3 Foundation Layer Overview*.
http://flohofwoe.blogspot.com/2007/01/nebula3-foundation-layer-overview.html
(Abgerufen am 16. September 2010).

[Wei307] Weißflog, A. (2007). *The Nebula3 Application Layer*.
http://flohofwoe.blogspot.com/2007/11/nebula3s-application-layer-provides.html
(Abgerufen am 17. September 2010).

10 Tabellen-und Abbildungsverzeichnis

Abbildung 1: Autodesk Maya 2009 und die Zuordnung der Benutzeroberflächenelemente 10
Abbildung 2: Maya Transformationswerkzeuge 11
Abbildung 3: TorchED Benutzeroberfläche 14
Abbildung 4: Benutzeroberfläche des Level-Editor für die Nebula3-Engine 17
Abbildung 5: Schichten der Nebula3-Engine 19
Abbildung 6: Transformation-Werkzeuge des Level-Editor – Handle für Platzierung, Handle für Rotation und Handle für Skalierung 22
Abbildung 7: Platzierungsvarianten des Level-Editor - SnapToGrid, SnapToGround und SnapToEdge 23
Abbildung 8: Kommunikation des Level-Editors mit dem Level-Editor Add-On in Nebula 24
Abbildung 9: Kameraansichten Maya-Kamera vs. Spielperspektive 27
Abbildung 10: Datenfluss von Benutzeroberflächen über Datenkern bis hin zu fertigen Spieldaten 30
Abbildung 11: Einbettung der Plug-ins in die jeweiligen Containerklassen 31
Abbildung 12: Benutzeroberfläche des ViewerDefault-Plug-ins mit Werkzeugleiste und eingebettetem Nebula-Fenster 35
Abbildung 13: Outliner-Fenster des Default-Outliner-Plug-ins 36
Abbildung 14: Eigenschaftenfenster des Default-Outliner-Plug-ins 36
Abbildung 15: Fenster des ToolEnvBrowser-Plug-ins mit Dateiselektion und Vorschaufenster 37

Tabelle 1: Plug-in-Container und Klassen 32
Tabelle 2: Implementierte Plug-in-Interface-Namen 33
Tabelle 3: Aufgabenblöcke für die Basisimplementierung des Level-Editors 41
Tabelle 4: Exemplarische Auflistung der Aufgaben und Fehler zur Objektplatzierung 50

Anhang

Level-Daten Format


```xml
<?xml version="1.0" encoding="UTF-8"?>
<LevelEditor>
 <LevelNode>
  <LevelNodeAttr name="guid" type="System.Guid" value="3b8d26f4-a061-4750-93ca-183c9128118d" />
  <LevelNodeAttr name="class" type="System.String" value="RootNode" />
  <LevelNodeAttr name="name" type="System.String" value="Root" />
  <LevelNodeAttr name="DefaultMusic" type="System.String" value="" />
  <LevelNodeAttr name="ReverbPreset" type="System.String" value="" />
  <LevelNodeAttr name="DefaultAmbient" type="System.String" value="" />
  <LevelNode>
   <LevelNodeAttr name="guid" type="System.Guid" value="ab253a04-00ed-4f02-942e-c63af9240d5e" />
   <LevelNodeAttr name="class" type="System.String" value="LightNode" />
   <LevelNodeAttr name="name" type="System.String" value="DefaultDirLight" />
   <LevelNodeAttr name="resource" type="System.String" value="" />
   <LevelNodeAttr name="position" type="LevelEditorContainer.Math.Vector4" value="0 0 0 1" />
   <LevelNodeAttr name="scale" type="LevelEditorContainer.Math.Vector4" value="50 50 50 1" />
   <LevelNodeAttr name="rotation" type="LevelEditorContainer.Math.Vector4" value="0 0 0 0" />
   <LevelNodeAttr name="LightType" type="System.Int32" value="0" />
   <LevelNodeAttr name="LightIntensity" type="System.Single" value="1" />
   <LevelNodeAttr name="LightColor" type="LevelEditorContainer.Math.Vector4" value="1 1 1 1" />
   <LevelNodeAttr name="LightOppositeColor" type="LevelEditorContainer.Math.Vector4" value="0.45 0.52 0.6 1" />
   <LevelNodeAttr name="LightShadowIntensity" type="System.Single" value="1" />
   <LevelNodeAttr name="LightConeAngle" type="System.Single" value="45" />
   <LevelNodeAttr name="LightCastShadows" type="System.Boolean" value="False" />
   <LevelNodeAttr name="LightAmbient" type="LevelEditorContainer.Math.Vector4" value="0.4 0.4 0.4 1" />
  </LevelNode>
  <LevelNode>
   <LevelNodeAttr name="guid" type="System.Guid" value="904fb6c9-5497-4c55-8a53-6540067222d9" />
   <LevelNodeAttr name="class" type="System.String" value="GroupNode" />
   <LevelNodeAttr name="name" type="System.String" value="group" />
   <LevelNodeAttr name="visible" type="System.Boolean" value="True" />
   <LevelNodeAttr name="visibilityGroup" type="System.Boolean" value="False" />
   <LevelNodeAttr name="variationGroup" type="System.Boolean" value="False" />
   <LevelNode>
    <LevelNodeAttr name="guid" type="System.Guid" value="1aa24241-86ee-482f-8de2-beace49437e8" />
    <LevelNodeAttr name="class" type="System.String" value="EnvNode" />
    <LevelNodeAttr name="resource" type="System.String" value="t001_hub/arch_windmill_tower_01" />
    <LevelNodeAttr name="name" type="System.String" value="arch_windmill_tower_01" />
    <LevelNodeAttr name="physics" type="System.String" value="t001_hub/arch_windmill_tower_01" />
    <LevelNodeAttr name="position" type="LevelEditorContainer.Math.Vector4" value="0 0 0 1" />
    <LevelNodeAttr name="scale" type="LevelEditorContainer.Math.Vector4" value="1 1 1 1" />
    <LevelNodeAttr name="rotation" type="LevelEditorContainer.Math.Vector4" value="0 0 0 0" />
   </LevelNode>
  </LevelNode>
  <LevelNode>
   <LevelNodeAttr name="guid" type="System.Guid" value="1c391877-4117-4603-8fe1-4818362dd401" />
   <LevelNodeAttr name="class" type="System.String" value="GameNode" />
   <LevelNodeAttr name="name" type="System.String" value="t003_destroy_mushroom_toxic_01" />
   <LevelNodeAttr name="category" type="System.String" value="Destroyable" />
   <LevelNodeAttr name="objectId" type="System.String" value="t003_destroy_mushroom_toxic_01" />
   <LevelNodeAttr name="position" type="LevelEditorContainer.Math.Vector4" value="-6 0 -2 1" />
   <LevelNodeAttr name="scale" type="LevelEditorContainer.Math.Vector4" value="1 1 1 1" />
   <LevelNodeAttr name="rotation" type="LevelEditorContainer.Math.Vector4" value="0 0 0 0" />
   <LevelNodeAttr name="Id" type="System.String" value="t003_destroy_mushroom_toxic_01" />
   <LevelNodeAttr name="Graphics" type="System.String" value="t003_outdoor/destroy_mushroom_toxic_01" />
   <LevelNodeAttr name="Physics" type="System.String" value="t003_outdoor/destroy_mushroom_toxic_01" />
   <LevelNodeAttr name="Placeholder" type="System.String" value="" />
   <LevelNodeAttr name="Name" type="System.String" value="FIXME" />
   <LevelNodeAttr  name="StateFilename"  type="System.String"  value="t003_outdoor/destroy_mushroom_toxic_01"
/>
   <LevelNodeAttr name="State" type="System.String" value="active" />
  </LevelNode>
 </LevelNode>
 <Version version="" />
</LevelEditor>
```

Ausschnitt aus der Bug-Datenbank

Tabelle 4: Exemplarische Auflistung der Aufgaben und Fehler zur Objektplatzierung

Summary	Issue Type	Priority	Created	Description
Leveleditor: Pivot-Handling verbessern (Pivot-Pfeile und -Würfel)	Story	Minor	30.09.2010 10:35	- Die drei Vektorpfeile sind zu lang, aber vor allem ist das Selektieren zu fistelig. Die Pfeile brauchen ein größeres Picking-Collide und Längenverstellbarkeit wäre auch nicht schlecht. - Der Würfel im Zentrum des Pivots, der ein freies Verschieben des Objekts ermöglicht, ergibt so wie er jetzt funktioniert, keinen Sinn. Denn freies Verschieben parallel zur Sichtebene ist niemals erwünscht. Zweckdienlich hingegen ist es, wenn man das Objekt parallel zu der angeklickten Oberfläche des Pivot-Würfels bewegen kann.
Leveleditor: Snap to Grid als Standard einstellen	Story	Minor	30.09.2010 10:00	Von den Snap-Möglichkeiten sollte das Snap to Grid standardmäßig aktiviert sein.
Leveleditor: Klickbereich von Translate-Tool-Achsen verschmälern	Story	Minor	30.09.2010 08:54	Ich möchte mit Shift und Klick in alle Richtungen Objekte der Selektion hinzufügen können. Derzeit sind jedoch die Klickbereiche der Achsen des Translate-Tools so breit, dass ich ein Groundtile, über das eine der Achsen liegt, gar nicht oder nur in einem ganz kleinen Bereich am Rand anwählen kann. Deswegen muss man derzeit bei angewähltem Translate-Tool eine Selektion immer mit dem Groundtile unten rechts beginnen.
Leveleditor: Objekte einer Gruppe einheitlich um einen Gruppen-Pivot rotieren lassen können	Task	Minor	29.09.2010 15:27	Noch besser wäre es sogar, generell beim Rotieren zwischen zwei Modi wechseln zu können: 1) alle Objekte rotieren um ihren eigenen Pivot 2) alle Objekte rotieren um den Pivot des zuletzt selektierten Objekts

Leveleditor: Rotation in Grad angeben	Story	Minor	29.09.2010 14:37	Die Rotation von Objekten in Grad anzeigen. In Maya hat jedes Objekt die Rotation um die drei Achsen in Grad angegeben. Setzt man ein frisches Objekt und dreht es um 90°, dann wird aus den Rotationsangaben 0 0 0 dadurch 0 90 0. Im Leveleditor wird derzeit aus 0 0 0 0 durch das Drehen um 90° die Angabe 3,141593 -1,550405 3,141593 0.
Leveleditor: Verschieben von StateObjects verschiebt Grafik mit Verzögerung	Bug	Minor	29.09.2010 12:48	StateObj erscheinen im Leveleditor mit ihrer Grafik plus einem grauen Dummywürfel. Verschiebt man das StateObj, so wird visuell zunächst nur der Dummywürfel verschoben und die Grafik bleibt an der alten Position stehen. Erst wenn man eine andere Aktion vornimmt - z.B. ein weiteres Objekt einfügt - wird auch die Position des grafischen Objekt aktualisiert.
Leveleditor: Rotation mit festem Winkel	Story	Minor	29.09.2010 10:43	Ähnlich wie das Snap to Grid beim Verschieben sollte es auch für das Drehen einen Snap auf bestimmte Winkel geben. Wände, Treppen etc. möchte man meist um genau 90° drehen.
Leveleditor: Grid einblenden	Story	Major	29.09.2010 10:10	Ich möchte das Grid einblenden können, um mich daran zu orientieren.
Leveleditor: Snap-To-Grid auch auf selbsteinstellbarem Abstand	Story	Minor	29.09.2010 09:41	Ich möchte den Snap-Abstand für das Grid selbst einstellen können bzw. das Snap für die Höhe anders einstellen können als für die horizontale Bewegung. Es gibt nämlich von den Grafikern Treppen und Rampen, die nur 2m oder 1m statt 4m an Höhe überwinden. Dann hat man derzeit eine Zwischenhöhe, für die das Snap to Grid nicht funktioniert.
Leveleditor: Eingestellte Snap-Option an Buttons anzeigen	Story	Minor	29.09.2010 09:38	Ich möchte an den Snap-Buttons sehen, welche Snapoption gerade aktiv ist.

Leveleditor: Im Snapmodus Objekte nicht frei gleiten, sondern gleich an nächste valide Stelle erscheinen lassen	Task	Minor	29.09.2010 15:21	Ein zügiges und fehlerfreies Arbeiten ist im Snap-Modus nur möglich, wenn man stets das Endresultat des Verschiebens/Rotierens augenblicklich sieht. Auf gerate wohl da ein wenig zu schieben/drehen, in der Hoffnung, dass irgendwann möglicherweise ein Schwellenwert überschritten ist, dann loszulassen, um zu erkennen, dass dem doch nicht so ist, raubt enorm Zeit und frustriert.
Generated at Sun Oct 03 14:03:05 GMT+01:00 2010 by Susanne Schulz using JIRA 4.1.2#531.				