HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Bearbeitungsbeginn: 01.09.2014

Vorgelegt am: TBA

Thesis

zur Erlangung des Grades

Master of Science

im Studiengang Medieninformatik

an der Fakultät Digitale Medien

Dominik Steffen

Matrikelnummer: 245857

Technical game-authoring process and tool development

Erstbetreuer: *Prof. Christoph Müller*

Zweitbetreuer: *Prof. Dr. Wolfgang Taube*

Abstract

Arbeitsprozesse in heutigen Game Engines verlangen von Entwicklern meist das erlernen neuer Toolsets und das während eines Zeitlich knapp bemessenen Projekt Zeitraums. Es wäre für Entwickler einfacher sich mit den bereits bekannten Tools, wie Modeling Editoren, zu beschäftigen und mit diesen großartige Ergebnisse zu erreichen. Designer müssen sich oft in unbekannte Editoren und SDKs einarbeiten während Entwickler sich in Grafische Editoren einarbeiten sollen um ihren Code an der richtigen Stelle des Projekts einzubinden. Diese Arbeit baut eine Brücke zwischen beiden Welten. Durch die Konzeption eines Software Tools und Entwicklungsprozesses zum erstellen von Game Authoring Tools, wird gezeigt wie mit verschiedenen Frameworks bestehende Software erweitert werden kann um sie als Authoring Tool zu benutzen. Mit Hilfe eines Cinema 4D Plugins ist es möglich, dass Designer oder Entwickler jederzeit mit ihren eigenen Tools in die Entwicklung eines Projektes einsteigen. Das während der Arbeit entstandene Plugin bietet grundlegenden Funktionen um an einem Projekt mit der FUSEE Engine zu arbeiten. Ein FUSEE Projekt "managedßich durch die Nutzung des entstandenen Cinema 4D Plugins und den generierten Visual Studio Solution Dateien aus Sicht des Plugin Nutzers selbst. Das zuerst konzeptionell entworfene Tool wurde während dieser Arbeit als Prototyp umgesetzt und bietet ausreichende Funktionalität um ein Projekt als Entwickler als auch als Artist zu erstellen und zu bearbeiten. Hierzu wurden verschiedene Konzepte betrachtet und andere GameEngines auf Workflow und Anwendbarkeit untersucht. Das C# Plugin Projekt FUSEE Uniplug wurde analysiert und in seinem Funktionsumfang erweitert. Das Ergebnis dieser Arbeit ist eine grundlegende Software Bibliothek in C# die nicht nur für Cinema 4D eingesetzt werden könnte.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Masterthesis selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Alle verwendeten Quellen und Hilfsmittel die sowohl zum schreiben dieser Arbeit als auch zum Entwickeln des dazugehörigen Sourcecodes benutzt wurden, habe ich angegeben.

Dominik Steffen, Küssaberg den 15. April 2015

"Hier steht ein wichtiges Zitat zur Entstehung dieser Arbeit."

- TBD.

Dominik Steffen
Matr.-Nr.: 245857
Hochschule Furtwangen

E-Mail:
dominik.steffen@hs-furtwangen.de
dominik.steffen@gmail.com

Inhaltsverzeichnis

1 Anforderungen, Ziele und eine Fragestellung

Arbeitsprozesse in heutigen Game Engines verlangen von Entwicklern meist das erlernen neuer Toolsets und dies während eines meist sehr eingeschhränkten Projekt Zeitraums. Es wäre für Entwickler einfacher sich mit den bereits bekannten Tools zu beschäftigen und mit diesen großartige Ergebnisse zu erreichen. Authoring Tools ermöglichen also auch Teammitgliedern ohne weiteres tiefegend technisches Verständnis für die Programmierung von Spielen, an Projekten in der Entwicklung mitzuarbeiten. Dieser Ansicht ist auch F. Mehm. Sein Team veröffentlichte einen Überblick (Florian, Mehm, S. R., Christian,Reuter. (2014). Future trens in game authoring tools. S. 536-541) über Game Authoring Tools in der Gegenwart und in der Zukunft.

> It [Developing "Authoring Tools", Anmerkung des Autors] is successful due to several factors: it allows non-technical users to work on projects that would otherwise be out of their reach (due to lack of expertise, especially concerning programming languages); it can bring struc-ture into unstructured domains (such as game development) and it can speed up de-velopment by streamlining and automating common tasks. Florian, Mehm, 2014

1.1 Motivation

Diese Arbeit beschäftigt sich nun mit der Frage ob es möglich ist ein Tool zu konzipieren welches auf der Basis eines bereits bestehenden Modeling Editors (hier Cinema 4D von Maxon[1]) das Erstellen einer "fertigen"[2] Szene für die 3D Engine Fusee[3] ermöglicht. Hierbei wird nach der Konzeption versucht die Basis Funktionalität in Visual Studio mit Hilfe von C# Code und der nach C# gewrappten[4] Cinema 4D API zu implementieren. Die gewrappte Cinema 4D

[1] MAXON Computer GmbH. (2014-2015). Cinema 4D R16. http://www.maxon.net/de/products/cinema-4d-studio.html.
[2] Build fähige Version einer im Fusee Szenenformat abgespeicherten 3D Szene.
[3] FUSEE (Furtwangen Simulationd and Entertainment Engine - http://www.fusee3d.org)
[4] Eine Software welche von einem anderen Stück software umgeben wird.

API basiert auf einem ehemaligen Projekt der Hochschule Furtwangen. Dieses wird als Grundlage für die hier angedachte Implementierung genutzt und bietet einen geringen Umfang an Basisfunktionalität. So bietet es die Möglichkeit grundsätzlich Plugins für Cinema 4D in der Programmiersprache C# zu schreiben. Von Haus aus ermöglicht Maxon das Entwickeln von Plugins nur in C++, Python und Coffee (einer von Maxon selbst entwickelten Skriptsprache). Der Vollständigkeit halber sei gesagt, dass Maxon für C++ noch das Framework Melange anbietet welches es ermöglicht Cinema 4D Dateien ohne eine Cinema 4D installation zu erstellen, zu speichern und zu laden. Sollte eine Installation vorhanden sein kann das Plugin auch Szenen rendern.

Szenen in Cinema 4D werden grundsätzlich in einer Art Baumstruktur gespeichert und zur weiteren Verarbeitung im Speicher gehalten. Die hier konzipierte Software möchte diese Tatsache nutzen um Szenen aus einem Modeling Editor (Cinema 4D) in eine Szene des Fusee Szenen Formats (.fus) umzuwandeln. Eine ".fus" Datei ist ebenfalls in einer Baumartigen Struktur gespeichert. Dieses Prinzip der Szenendarstellung ist bereits aus verschiedenen Frameworks und Softwareprojekten für 2D Darstellung bekannt. Das ist zum einen der Übersichtlichkeit als auch verschiedenen Algorithmischen Operationen auf den Daten der Szene geschuldet. In Baumstrukturen organisierte Interfaces werden außerdem bei der Entwicklung des User Interfaces für das Mobile Betriebssystem Android verwendet.

1.2 Ziele der Implementierung

Die während dieser Arbeit implementierte Software hat das Ziel eine Basis für die Verwendung von Cinema 4D als Game Engine Editor aufzubauen. Es werden grundlegende Funktionen in Form einer C# Bibliothek entwickelt die es ermöglichen sollen das Projekt in Zukungt auch für andere 3D Modeling Software anzupassen. Diese Arbeit zielt nicht darauf ab ein komplettes Tool für die Entwicklung von Spielen in der Fusee Engine zu erschaffen. Es wird versucht eine art Grundstein für weitere Forschung und Entwicklung in den Bereich des Game Authoring Toolkit Developments für die Arbeit mit der Akademischen Simulations und Entertainment Software FUSEE zu legen. Das Kernziel ist das Erstellen eines Konzeptes und die Erläuterung der einzelnen Module eines solchen Systems. Verschiedene bereits bestehende Tools und Game Engines werden zu vergleichen herangezogen und wurden im Laufe dieser Arbeit untersucht und getestet.

1.3 Verwendete Software

- Microsoft Visual Studio 2010,
  verwendet als Entwicklungsumgebung für das Softwareprojekt.
- Die Erweiterung ReSharper in Version 7.1 für Visual Studio 2010 http://www.jetbrains.com/resharper
- Umlet http://www.umlet.com/
  Ein kostenloses Tool um UML Diagramme zu erstellen.
- GitHub und die GitShell www.github.com
  Verwendet als Versionskontrollsystem und als Distributionswerkzeug für den Sourcecode.
- TexWorks www.tug.org/texworks/
  Zum Schreiben dieser Arbeit.

2 Grundlegendes

2.1 Entwicklungsprozesse in Interaktiver 3D Software und Games

Um einen Entwicklungsprozess abzubilden und Tools für Entwickler, sogenannte Developer Tools, zu entwickeln bedarf es einer gewissen Organisation. Im Bereich der modernen Spieleentwicklung in kleinen bis mittleren Unternehmen (seltener bei großen AAA Produktionen [1]) wird hierfür ein agiles Modell zur Softwareentwicklung eingesetzt. Hier soll ein kurzer Überblick über aktuelle Modelle entstehen. Diese Modelle ermöglichen zum einen das schnelle Entwickeln von Tools während der knappen Entwicklungszeit eines Spiele Produkts und zum anderen unterstützen sie die Arbeit von kleinen Teams, in welchen meist Tool Developement betrieben wird, innerhalb eines großen Entwicklerteams um so gezielt plötzlich auftauchende Aufgaben ohne lange Planung und viel Bürokratie lösen zu können. Damit ist ein fortschreiten des gesamten Projektablaufs gesichert und Entwickler können ihre Zeit hauptsächlich für die Entwicklung der Tools investieren.

2.1.1 Projektmanagement Modelle

Um große Projekte wie Computergames oder Interaktive Software zu entwickeln, bedarf es meist einer detaillierten Planung und einer exakten Rollenverteilung im Entwicklerteam. Es existieren verschiedene Methoden des Projektmanagement auf welche hier kurz im Zusammenhang mit der Arbeit eingegangen werden soll. Einige der Projektmanagement Modelle wirken auf die Arbeitsweise der Teammitglieder aus. Daher wird diese Arbeit hier keinen Umfassenden Überblick über Projektmangament Methoden geben, sondern nur solche Ansprechen die sich direkt oder indirekt stark auf das Tool Development auswirken.

[1] Allgemein: Hochqualitative Spiele Software mit großem Entwicklungsbudget und einer Breiten Zielgruppe. Vgl. Cifaldi, 2005

Agile Modelle vs. klassische Modelle

Viele Entwickler (Ubisoft, siehe Schmitz, 2014) setzen heute auf moderne Modelle zum Entwickeln von Software. Die so genannten agilen Modelle (wie Beispielsweise Scrum, Extreme Programming und Feature Driven Development) ermöglichen meist das schnelle (agile) reagieren auf plötzlich auftauchende schwierige Situationen. Klassische Modelle (Wasserfallmodell, Spiralmodell) haben hier meist Probleme durch ungleich höhere Bürokratie und Komplexität und benötigen ein Zeitaufwändigeres re-iterieren im Falle von Updates und Umstrukturierungen in Folge von unvorhergesehenen Ereignissen und Problemen. Hochkomplexe Software Projekte die über längere Zeiträume entwickelt werden können meist nur durch klassiche Projektmanagement Modelle überblickt und erfasst werden. Allerdings bedeutet der zusätzliche Bürokratische Mehraufwand auch oftmals einen erhöhten Overhead im Personal-, Software- und Knowledge-Bereich. Es ist im Fall des schnell-lebigen Tool Developments also geschickter, sich mit einem ebenso schnell-lebigen und agilen Projektmanagementmodell wie Scrum zu organisieren.

Scrum

Der Scrum Prozess tauchte das erste mal in der Veröffentlichung "The New New Product Development Game" von Hirotaka Takeuchi and Ikujiro Nonaka 1986 auf - damals nicht unbedingt in der Software- sondern der allgemeinen Produktentwicklung eingesetzt. Seitdem hat sich das Modell weiter entwickelt und erfreut sich bei innovativen Softareprojekten im Games und Indie-Games Bereich (auch und meist wohl auch vor allem im Tool Development) sehr großer Beliebtheit. Die Entwickler CCP und Warhorse Studios hatten hierzu eigene Videos und Artikel veröffentlicht, siehe CCP Games, 2009, Schmitz, 2013, Martin Klekner, 2014.
Ein Scrum Entwicklerteam ist mit folgenden Rollen besetzt:

- Product Owner
- Entwicklungsteam
- Scrum Master

Bei diesen Rollen handelt es sich um das interne Scrum Team - das Entwicklungsteam des Produktes. Scrum kann innerhalb eines Projektes und Teams beliebig heruntergebrochen werden, bis die gewünschte größe eines Entwicklerteams erreicht wird. Externe Rollen wie Stakeholder etc. verlagern sich somit auf andere interne Projektleiter oder Teammitglieder. Aus diesem Grund

ist das Model gut für die Entwicklung von Development Tools und Toolkits geeignet. Mit Hilfe des Models, können benötigte Toolkits während einer Projektlaufzeit schnell und effizient entwickelt werden ohne dass ein schwerfälliger Bürokratischer Prozess die Entwicklung blockiert. Somit ergänzt sich dieser Prozess gut mit dem doch eher agilen entwickeln von Developement Tools während der Projektlaufzeit - denn in den seltensten Fällen wurde vor dem Beginn des Projekts daran gedacht alle nötigen Tools bereitzustellen. Oftmals ergeben sich auch während der Entwicklung neue Herausforderungen für das Team welche nach neuen Tools verlangen.
Hier soll nurn kurz ein Szenario aufgebaut werden, welches das Tool Developement Team eines aktiven Software Entwicklers beschreibt. Zuerst einmal sollen die Rollen verteilt werden:

- Product Owner - Meist der leitende Entwickler des Software Projektes. In diesem Fall meist ein Producer und/oder Game Developer.
- Entwicklungsteam - Das Tool Development Team selbst.
- Scrum Master - Die leitende Person des Tool Development Teams, bzw. sollte sich das Team sehr nah am Scrum Modell bewegen, dann meist ein Entwickler außerhalb des Teams aber mit guten Kontakten zum Team selbst und erhöhter Erreichbarkeit.

Die Stakeholder des Tools wären in diesem Fall die anderen Entwickler des Unternehmens die das Produkt im Produktiefbetrieb einsetzen möchten. Es kann hierbei auch von Vorteil sein, das Tool iterativ in den Arbeitsalltag des Teams zu integrieren um die Entwickler nicht durch einen Berg an neuen Features zu verunsichern und so die Einarbeitungszeit möglichst gering zu halten.

Es soll hier an einem kurzen Beispiel deutlich gemacht werden, wie ein solches Tool eingeführt werden könnte:

Szenario: Ein Team benötigt einen Textur-Editor / Tool um Texturen in das Format der Game-Engine zu transformieren.

- Der Antrag für das Tool vom Producer/Entwickler/oder anderen Personen wird gestellt.
- Das Tool wird bewilligt und das Tool Development Team wird beauftragt.
- Das Team entwickelt designed das Tool und implementiert Basisfunktionalität.

- Das Tool wird mit der Basisfunktionalität an das Produkt Team herausgegeben.
- Die fehlenden Funktionen werden implementiert.
- Das Tool wird mit der erweiterten Funktionalität herausgegeben.
- Es wird mit dem Produkt Team Rücksprache gehalten, welche Funktionen noch benötigt werden.

Dieser Prozess schließt iterrativ ab bzw. nicht ab, da während der Entwicklung einer interaktiven Anwendung / Games eventuell auch auf externe Einflüsse wie Third Party Software oder Marktentwicklungen eingegangen wird. Als Beispiel: So könnte sich durch die Veröffentlichung einer neuen GPU Generation oder den vorgezogenen PC Release die Größe der benötigten Textur-Dateien ändern.

2.1.2 Internes Tool Developing oder Tool licencing

Internes Tool Development ist ein wichtiger Aspekt im Team eines Games und Software Entwicklerteams. Erich Bethke berichtet in davon, dass Michael Abrash[2] ihm einst mitteilte, "dass 50% der Entwickler Arbeit bei idSoftware in das Tool Development fliesse." vgl. Bethke, E. (2003). *Game Development and Production*. Wordware Publishing Inc.; Auflage: Pap/Cdr (Februar 2003), S. 44.An der Relevanz des Themas hat sich trotz des zurückliegenden Zeitraums kaum etwas getan. Sony hat für den Release der Playstation 4[3] ein Development Kit[4] für die internen Entwickler Studios erstellen lassen, welches bereits während der Planung und Entwicklung der Konsole entwickelt wurde. Sony hat diese Prozedur perfektioniert und lässt die eigenen Tools sogar in einem eigens dafür gegründeten Unternehmen für die eigenen Studios erstellen [5]. Sony hat im Herbst 2014 den für Playstation 3 Spiele eigens intern entwickelten Welt Editor "Level Editor"[6] als Open Source Software veröffentlicht und für Jedermann auf GitHub verfügbar gemacht. Der Editor kommt ohne direkten Enginebezug aus und lässt sich somit für verschiedenste Projekte der Sony Studios anpassen. Das ATF Framework, auf welchem viele interne Tools von Sony basieren[7], kann von Sound Editoren über Cinematic Editoren bis hin zu State Machine Visualisierungen genutzt werden.Eine Übersicht vonn

[2]Ehemals idSoftware, ehemals Valve VR, aktuell Chief Scientist bei Oculus https://www.oculus.com/company/
[3]Playstation 4 - Erschienen im Herbst 2013
[4]Freeman, W. (2014). Kingdom come: deliverance video update 9.
[5]SNSystems http://www.snsystems.com/
[6]Wawro, A. (2014). Sony releases level editor that's open source and engine-agnostic.
[7]Hier ein Einführungsvideo: https://www.youtube.com/watch?v=aU-9vzFELxc

Tools, welche das ATF Framework erfolgreich verwendet haben findet sich unter https://github.com/SonyWWS/ATF/wiki/ATF-Gallery. Natürlich ist hier trotzdem noch ein gewisser grad an Aufwand zu betreiben, aber durch das integrierte ATF Framework werden viele Bereiche mit wiederverwendbarem Code abgedeckt. Sony Hauseigene Entwicklerstudios haben ebenfalls ihre eigenen Tool Kits und Editoren auf dem von Sony bereitgestellten ATF Framework und "Level Editor" erstellt um Spiele wie Naughty Dogs Uncharted[8], Guerilla Games' Killzone Serie[9] oder Quantic Dreams Beyond:Two Souls[10] zu erstellen. Eine Übersicht der Studios welche das ATF Framework verwenden findet sich unter dieser Adresse https://github.com/SonyWWS/ATF/wiki/ATF-Adoption. Dieser große Einfluss des Frameworks zeigt, dass selbst in großen - und kleineren - Studios immernoch Bedarf nach einfach und schnell zu erweiternden Frameworks und Editoren besteht. Das ATF Framework bzw. der "Level Editor" von Sony waren auch ein Anlass das das praktische Projekt zu dieser Arbeit an zu implementieren.

2.2 Mitglieder eines Entwicklerteams

Es soll hier ein kurzer Überblick über die gängigsten Mitglieder eines Entwicklerteams gegeben werden. Grob können Entwickler in die folgenden drei Gruppen aufgeteilt werden - Artist, Designer, Engineer. Jede Gruppe arbeitet hierbei meist interdisziplinär mit den anderen zusammen, kümmert sich aber doch um die ganz eigenen Bestandteile eines Produktes. Es ist durchaus so, dass jede Gruppe ihre eigenen Tools und Methoden verwendet. Dieser Ansatz wird in der Konzeptionierung dieser Arbeit aufgegriffen und weiter verfolgt.
Bei der Bezeichnung und Aufteilung der verschiedenen Teammitglieder in Fachbereiche orientiert sich diese Arbeit am Werk von Chandler, H. (2006). *The Game Production Handbook*. Thomson Delmar Learning, in welchem er die Produktionsprozesse eines Spiels sowohl in designtechnischer Weise als auch aus technischer Sicht beschreibt.

2.2.1 Artists

Artists sind in einem Games Projekt für jegliche Repräsentation der Spiellogik nach Außen zuständig. Sie erstellen Modelle von Spielfiguren und Umgebungen

[8] NaughtyDog. (2007, 2009, 2011). Uncharted. http://www.naughtydog.com/games/uncharted_drakes_fortune/. Sony Computer Entertainment.

[9] Guerrilla Games. (2004, 2007, 2011, 2013, 2013). Killzone 1, Killzone 2, Killzone 3, Killzone: Mercenary, Killzone: Shadow Fall. http://www.guerrilla-games.com/. Sony Computer Entertainment.

[10] Quantic Dream. (2013). Beyond: Two Souls. http://www.quanticdream.com/en/. Sony Computer Entertainment.

und kreiiren Texturen und User Interfaces. Bei den Artists handelt es sich um eine Kerngruppe für diese Arbeit da sie einen Großteil der Arbeitszeit in den Tools und Editoren des Spiels verbringt. Artists können in mehrere Untergruppen aufgeteilt werden. Dies bedeutet jedoch nicht, dass jedes Unternehmen jede Artists Rolle beschäftigt. Oftmals übernehmen einzelne Mitarbeiter mehrere Rollen je nach dem Entwicklungsstand des Projekts.

Modeling/Animation Artist

Ein Animation Artist verbringt die meiste Zeit damit Animationen und Modelle (3D, 2D), kurz: Assets [11], für die Verwendung im Spiel vorzubereiten. Programme wie Cinema 4D[12], 3DS Max[13], oder Modo[14] sind Beispiele für Kernsoftware dieser Entwickler. Der vollständigkeit halber sei hier noch das Open Source Projekt Blender[15] erwähnt.

Environment Artist

2.2.2 Designer

Designer (Gamedesigner) arbeiten eng mit Artists und Engineers zusammen. Meist Entwicklen Game Designer das Spielprinzip, den Raum des Spiels und das Regelwerk. Sie schreiben oft Skripte und kleine Implementierungen oder verbessern Grafiken oder Spielfunktionen. Sie verwenden Assets aus der Designabteilung und fügen diese mit Skripten zusammen. Spieltests werden von Ihnen überwacht um den Spielfluss und das Erlebnis des Rezipienten beim Spielen zu optimieren.

Level/World Designer

Level bzw. World Designer erstellen aus den erschaffenen Assets eine oder mehrere zusammenhängende Spielwelten - sogenannte Level. Diese Welten werden durch sie und weitere Artists mit Inhalt nach den Plänen der Game Designer gefüllt. Oft haben diese Welten einen gewissen gestalterischen Anspruch und von den Designern erwünschten Artstyle welche die Atmosphähre des Spiels repräsentiert. Meistens werden diese Welten in einem extra dafür geschaffenen Editor angefertigt und können nich in einem Modeling Tool wie Cinema 4D

[11] Assets sind Bestandteile des Produktes welche eine Grafische oder logische Repräsentation im Produkt erfahren. Dazu zählen z.B. Modelle, Texturen und Code Dateien.
[12] MAXON Computer GmbH, 2014-2015.
[13] Autodesk, Inc., 2015.
[14] The Foundry Visionmongers Ltd., 2015.
[15] Blender Foundation, 2014-2015.

entwickelt werden. Ein Beispiel für solche Level Editoren ist der GTKRadiant[16] Editor für Spiele basierend auf der idTech3 und idTech4 Engine [17], beide als Open Source auf der Platform GitHub verfügbar. Weitere Beispiele sind der Level Editor von Sony, auf welchen diese Arbeit später noch eingeht sowie der Unity3d Editor. Der Unity3d beinhaltet eine gesamte Game Engine, jedoch wird direktes Modeling und das erstellen von Texturen von Grund auf nicht unterstützt. Alle Assets, außer primitiver Geometrischer Objekte wie Würfel und Kugeln etc. müssen in externen Programmen erstellt und importiert werden.
Die Konzeptionierung dieser Arbeit wird nurn die versuchen einen Ansatz zu entwickeln der es ermöglicht zumindest einen Teil der Level und Welteditor Tools zu beseitigen. Somit könnten Level und World Designer und Environment Artists ihre Arbeit in die bereits bekannten Modeling Tools verlagern und so eine verbesserte Projektivität erreichen.

Scripter

Scripter sind meist dafür Zuständig verschiedene Ereignisse in einer für die Game Engine extra entwickelten Script Sprache zu beschreiben und so die Welt des Spiels interaktiver zu gestalten. Diese Aufgaben unterstützen die Spiellogik oder aber beschreiben die Funktionen ganzer Systeme wie z.B. die eines Aufgabensystems (Quest Systems) welches dem Spieler während des Spiels mitteilt, was er in der Spieltwelt zu tun hat. Hier sind allerdings viele Bestandteile eines Spiels anzuordnen. Meist werden

User Interface Designer

User Interface Designer kümmern sich um das Erstellen von grafischen Schnittstellen welche die Interaktion mit dem Benutzer ermöglichen. Hierfür verwenden sie oft Scriptsprachen wie Actionscript von Adobe (Zur programmierung von Adobe Flash Interfaces) oder gar fertige Middelware wie Scaleform[18] ein Cross Plattform UI Solution Tool[19] von Autodesk. Diese Gruppe der Entwickler wird durch diese Arbeit nur sehr gering beeinflusst. In der Fusee Engine werden Interfaces über Code Dateien eingebunden und daher in externen Grafikprogrammen und Visual Studio angefertigt.

[16] Open Source Projekt GTKRadiant http://icculus.org/gtkradiant/
[17] Beide Engines und weiterer Source Code von idSoftare herunterzuladen auf dem Account des Unternehmens auf GitHub unter https://github.com/id-Software - geprüft am 08.04.2015
[18] Autodesk Inc., 2014-2015.
[19] Ermöglicht das erstellen von 2D, 2.5D und 3D Ui Elementen. Wird z.B. von der Unreal Engine 4 verwendet.

2.2.3 Engineer

Engineers / Ingenieure arbeiten meist am Kern der Applikation und schrieben den Source Code für die Anwendung, Engine, Netzwerkfunktionen, KI, und Tools. Diese Entwickler arbeiten hauptsächlich in einer IDE [20] wie Visual Studio (auf welches sich das zu dieser Arbeit konzeptionierte Tool bezieht) oder XCode [21]. Der in der IDE geschriebene Code wird dann von den Engineers selbst oder von Game Designer in der Engine verwendet. Hierbei kann sich das Tätigkeitsfeld ausweiten bis hin zur Entwicklung von Gamelogic [22].

Tool Engineer

Diese Arbeit bezieht sich auf den Bereich des Tool Development. Hierbei entwickelt ein kleines Team - meist während oder vor der eigentlichen Arbeit an einem Projekt die Tools für die restlichen Entwickler des Projektes. Diese Tool Palette kann von Textureditoren bis hin zu kompletten Welteditoren fast alles vorstellbare enthalten. Verschiedene Studios haben eigene Tool Developer Teams, welche sich nur um diesen Bereich des Produktes kümmern. Diese Teams betreuen auch meist den Modding Support für ein fertiges veröffentlichtes Produkt. Beispiele für Modding Tools sind z.B. das RedKit von CDProject Red für das Spiel The Witcher 1 und 2, der LevelEditor von Sony der in einer Open Source Version vorliegt oder das Creation Kit von Bethesda Softworks welches einen Modding Support für die Spiele der The Elder Scrolls Reihe bereit stellt.

Computer-Grahics Engineer

Computer Graphics Engineers beschäftigen sich meist mit dem entwickeln der eigentlichen Engine. Oft sind Graphics Engineere aber auch an der Tool Entwicklung beteiligt. Gerade in kleineren Unternehmen könnten die eigentlichen Strukturen schnell aufbrechen um Synergien zu nutzen.

2.2.4 Weitere für diese Arbeit nicht relevante Teammitglieder

In diesem Abschnitt sollen noch die restlichen Mitglieder des Teams erwähnt werden, welche Tools nutzen aber meist nicht an der Entwicklung der selbigen beteiligt sind.

- Sound Engineer

[20] Integrated Developement Environment
[21] X-Code ist nur für MacOSX erhältlich
[22] Logik des Spiels, ermöglicht das interagieren etc. mit und in der Software

- Bla
- Bla
- Bla
- Bla

2.3 Stakeholderanalyse intern

Um herauszufinden, welche Entwicklergruppen eines Teams von neuen Development Tools im Bezug auf Fusee betroffen wären, wurde eine Analyse durchgeführt, um die Stakeholder innerhalb des Teams abzubilden. In der Praxis wäre für die Konzeption bzw. das Systemdesign eines neuen Tools der Besuch der von der Tool Entwicklung betroffenen Teammitglieder am Arbeitsplatz und das beobachten der jeweils verrichteten Aufgaben sehr aufschlussreich. Durch diese Methode könnten Probleme im Arbeitsablauf frühzeitig identifiziert, behoben und besonders wichtige Features rechtzeitig vor Beginn der Implementierung in Erfahrung gebracht und geplant werden.
Eine Bedarfsanalye und eine allgemeine Analyse der Aufgabengebiete der jeweiligen Entwickler sollte in das System Design bzw. das Requirements Engineering ebenfalls mit einfließen. Hierzu könnten interviews mit den Entwicklern geführt werden. Dies ist eine schnelle Methode welche sich gut mit einem agilen Prozess wie Scrum vereinbaren lässt.

2.4 Ein Arbeitsprozess wird entwickelt

2.4.1 Game Authoring / Game Development

Game Development beschreibt allgemein das entwickeln einer Interaktiven Software, meist eines Spiels. Hierbei spielt es keine Rolle ob die Software der puren Unterhaltung dient oder sich dem Genre des Serious Games zuordnen lässt. Auch weitere interaktive Anwendungen ohne Unterhaltungsfaktor können in diese Kategorie fallen. Die Produktionsabläufe ähneln sich stark. Im Gegensatz zum Tool Development hat das Game Development meist den Auftrag am Ende ein für den Consumer zugängliches Produkt zu schaffen. Das Tool Development beschäftigt sich in erster Linie mit dem Erstellen der Werkzeuge welche benötigt werden um das Consumerprodukt zu entwickeln.
Es ist aber in der Tat so, dass Tool Development auch auf dem Consumer Markt ankommen kann. Verschiedene Entwickler stellen in der Vergangenheit ihre Tools den Fans und Kunden zur Verfügung. Aus diesen Tools haben

sich selbst schon wieder Produkte wie Modifikationen[23], Patches oder Erweiterungen entwickelt. Einige Beispiele sind die das Creation Kit von Bethesda Softworks und die daraus entstandenen zahlreichen Fan Modifikationen wie Nehrim oder Enderal des Mod Teams SureAI[24].

2.4.2 Tool Development - der organisatorische Ablauf

Nach Wihlidals Auffasung (Wihlidal, 2006) unterscheidet sich die Planung eines Projektes zur Erstellung eines Developer Tools nicht sehr von der Planung zur allgemeinen Entwicklung von Software. Es gelten hier vier Planungsphasen die den Projektablauf kennzeichnen. Der erste Schritt wäre eine allgemeine Planung des Tools. Diese beinhaltet Funktionen und Umfang des Tools. Eine Beschreibung der Anforderungen bzw. der Ziele des Projektes wird mit den begünstigten Entwicklern abgesprochen.
Die zweite Phase beschreibt eine Bedarfsanalyse der Stakeholder. Es werden Arbeitsabläufe skizziert und mit den Beteiligten durchgesprochen. Je nach Komplexität und Relevanz des Projekts wird Software anderer Hersteller oder anderer Arbeitsbereiche ebenfalls analysiert und das Ergebnis zur Gesamtanalyse hinzugezogen. So könnte eventuell der Einsatz einer Drittanbieter Software in gewissen Situationen vorteilhafter sein als eine komplette interne Neuentwicklung. Diese Entscheidung ist aber sehr Situationsabhängig.
Daraufhin folgt die Designphase in welcher das Entwicklerteam des neuen Tools das Requirements Engineering abschliesst und mit dem Software-/Systemdesign beginnt. Hier wird das Produkt in Form von UML Diagrammen und Veranschaulichungen entwickelt. Die tatsächliche Implementierung folgt als letzte Phase des Projektablaufs. Während der Implementierung kann aufgrund der Verwendung des agilen Scrum Systems immer wieder iterativ an den Features des Tools gearbeitet werden und vor allem schnell auf Hindernisse und Wünsche der Stakeholder reagiert werden.
Im folgenden Abschnitt wird noch etwas genauer auf die jeweiligen Schritte der organisatorischen Planung eingegangen.

Planung

Es folgt die Planungsphase des Tools. Hier wird zuerst eine Requirementsanalyse [25] durchgeführt. Mit ihr sollen alle wichtigen Kerneigenschaften der Software identifiziert und niedergeschrieben werden. Ein an diese Arbeit angehäng-

[23] auch Mods genannt, Verändern das eigentliche Spiel oder ersetzen es in Form einer "Total Conversion" gleich ganz durch neue, eigens entwickelte Inhalte
[24] Freies Modifikations Team. http://www.sureai.net
[25] dt. Anforderungsmanagement

tes Designdokument führt diese Requirementsanalyse weiter aus. Die Anforderungsanalyse ist in einer solchen Situation, in welcher ein Produkt unter Zeitdruck für den Produktivbetrieb entwickelt wird, ein wichtiger Bestandteil der Projektplanung. Ein Tool, welches nicht den Anforderungen der Teammitglieder entpsricht kann nicht im Betrieb eingesetzt werden und verzögert im schlimmsten Fall die weitere Entwicklung des gesamten großen Softwareprojektes.

Requirements Analyse

- Eine Software soll es ermöglichen, dass Artists, Designer und Developer an ein und dem selben Projekt arbeiten können ohne die gewohnte Arbeitsumgebung (3D-Modellierungssoftware, IDE) zu verlassen und etwas komplett neues (Level-Editor) zu erlernen.
- Das Produkt muss auf Basis der FUSEE Engine entstehen
- Ziel ist es in Cinema 4D ein FUSEE Projekt anzulegen, zu speichern und es zu öffnen
- Assets sollen ins Spiel integriert werden können die von Artists, Designern und Entwicklern bearbeitet werden können.
- Das FUSEE C# Projekt sollte aus C4D heraus gebaut werden können.
- Eine Stakeholderanalyse schafft klarheit, welche Parteien des Teams mit dem zu erstellenden Tool arbeiten müssen.
- Es ist zu analysieren, welche Schritte für welche Art der Arbeit des Teams notwendig sind. Hierzu werden Usecases der verschiedenen Rollen und Aufgaben erstellt.

Requirements Dokumentation

Während der Requirements Dokumentation werden alle Ansprüche an die Software Dokumentiert und von den Stakeholdern geprüft. Über dieses Dokument lässt sich der spätere Funktionsumfang des Tools genaustens definieren und verfolgen. Das Dokument stellt sozusagen ein rechtlich relevantes Dokument dar. Es findet sich im Anhang dieser Arbeit.

System Design / System Modeling

Anschließend an die Analyse folgt das System Modeling in welchem die Anforderungen des Programs zu einem Softwareprodukt modeliert werden. Oft

bedient sich das Entwicklerteam hierbei Notationen wie UML [26] in Diagramm und Schrift Form. Das System Design ist ein kritischer Punkt. Hier müssen die Anforderungen des Kunden genau in die geplante Entwicklung des Systems übernommen werden. Oftmals arbeitet das System Design

Abgleich des System Designs mit den Anforderungen

Vor der Implementierung muss immer auch geprüft werden ob das geplante System aus dem Designprozess mit den tatsächlichen Anforderungen der Stakeholder zusammen passt.

Implementierung

Die Implementierung ist der praktische Schritt des ganzen Prozesses. Während der Implementierung wird das Tool entwickelt, auf Fehler geprüft eventuell auch durch einen iterativen Entwicklungsprozess erweitert.

Asset Pipeline und Feedback

SEHR KRITISCHE MARKE! Das Asset aus dem Editor in die Engine bekommen, die Darstellung etc kontrollieren. Zeitkritisch beim export etc. Carter Seite 6 und folgende.

2.5 Game Assets, Asset Pipelines und Asset Management Tools

Assets und das aufbrechen in Bestandteile eines Projektes. Level etc. bestehen aus Assets und mehr.

2.5.1 Asset Pipelines in Fusee

2.5.2 Asset Management in Fusee

2.5.3 Warum eine Trennung von Code und Content?

[26] Unified Modeling Language

3 Entwicklung eines Konzeptes

3.1 Use Cases der verschiedenen Entwickler

3.1.1 Was möchten Artists?

3.1.2 Was möchten Designer?

3.1.3 Was möchten Entwickler?

3.1.4 Prozess bezogen

Gleichzeitig an Projekt arbeiten

Model Datei importieren

Gleichzeitig an einem Objekt arbeiten

3.1.5 Projekt bezogen

Projekt anlegen

Projekt öffnen

Projekt speichern

Projekt bauen

In Projekt einsteigen

Projekt clonen etc.

3.2 Aktuelle Engines und deren Arbeitsprozesse

3.2.1 Prozesse in Game Engines und/oder Frameworks

3.2.2 Unreal Engine 4

3.2.3 Unity 3D

3.2.4 idTech X

3.2.5 Weitere

3.3 Konzeptentwurf

3.3.1 Systemdesign für ein Plugin

3.3.2 Systemdesign für einen Project-Handler

3.3.3 Entfernen von Abhängigkeiten

3.3.4 Zeitersparnis durch bekannte Tools

- Es soll möglichst wenig geparsed oder konvertiert werden
- Dateien sollen Version Control kompatibel bleiben (wenig bis keine Binarys)
- Das Projekt muss Strukturiert sein
- Eine Fusee Solution soll gehandelt werden können

3.4 Der Weg der Cinema 4D C++ API bis zur Verwendung dieser im Projekt

3.4.1 In der Implementierung verwendete Softwareprojekte / Ausgangssituation

Das Projekt "Fusee Authoring Toolkit" [1] basiert auf mehreren Software Projekten. Ein Teil dieser Projekte wurde an der Hochschule Furtwangen entwickelt. Ein anderer Teil stammt von externen Entwicklen und gehört zu proprietärer Software. Dieser Abschnitt gibt einen Überblick über die verwendeten Softwarebestandteile und ihrer Einbindung in der Implementierung zur Zeit dieser Arbeit.

Verwendet wurden:

- FUSEE Math Bibliothek, eine von Fusee unabhängige Sammlung in C# geschriebener Mathematischer Funktionen und Datentypen
- Uniplug, ein in Fusee beinhaltetes Projekt zum schreiben von Plugins in C# für Cinema 4D
- Die FUSEE Engine, eine Interaktive Simulations Engine zur Darstellung zweidimensionaler und dreidimensionaler Szenen
- Die Maxon Cinema 4D API
- Verschiedene Windows Bibliotheken aus dem .NET Universum

3.4.2 Cinema 4D Plugin API und SDK

Cinema 4D R16 (MAXON Computer GmbH, 2014-2015) ist ein kommerzielles proprietäres Produkt des Unternehmens MAXON Computer GmbH und

[1] Der Name der zu dieser Arbeit zugehörigen Softwarebibliothek

steht nicht als Open Source Projekt zur Verfügung. Maxon stellt für die Entwicklung von Plugins[2] eine API[3] bereit. Bei dieser API handelt es sich um eine Bibliothek aus C++ Code. Dieser Code kann in Form von Header Files in die eigene Applikation integriert werden. Durch diese C++ Headerfiles ergibt sich eine Schnittstelle zu interenen Methoden in der Cinema 4D Software. Die tatsächlichen Methoden können mit der API nicht eingesehen werden, es handelt sich hier lediglich um Header Files ohne den Code der tatsächlichen Implementierung. Das C++ SDK wird bei einer Installation von Cinema 4D automatisch mitinstalliert. Auf einem Windows PC findet sich das SDK und die dazugehörigen Visual Studio Solution Dateien unter folgendem Pfad ausgehend vom Installations Ordner der 64Bit[4]: "Cinema 4D/plugins/cinema4dsdk"

Maxon stellt jegliche SDKs und Beispiele auf seinem GitHub Account unter der Apache License Version 2.0, January 2004[5] zur freien Verfügung https://github.com/PluginCafe. Das C++ SDK ist unter dieser Adresse einsehbar: https://github.com/PluginCafe/cinema4d_cpp_sdk
Das folgende Schaubild erläutert die Verwendung der API. Es ist in der Tat som dass der eigene Plugin Code laut Maxons Dokumentation einige Funktionen überschreiben bzw. Klassen vererben muss. Ansonsten ist der Inhalt des Plugincodes nicht beschränkt.
Das hier dargestellte Minimalbeispiel in C++ Code zeigt ein einfaches minimalistisches Plugin welches beim Ausführen nur eine Ausgabe auf der Konsole von Cinema 4D erzeugt.

Sourcecode Beispiele 3.1: Einfaches Cinema 4D Plugin in der Programmiersprache C++

```
1 #define ID_PLUGINID 1234567 // Dies ist eine einzigartige
      Plugin ID.
2
3 #include "c4d.h"
4
5 class MinimalesPlugin : public CommandData // Das Plugin ist
      vom Plugin Typ "Command Plugin"
6 {
7 public:
```

[2]Plugins in C4D - Erweiterungen für die Software Cinema 4D und deren Funktionen
[3]Application Programming Interface, z. Dt. Programmierschnittstelle
[4]Cinema 4D liegt seit der Version R15 nur noch als 64 Bit Version vor und benötigt laut http://www.maxon.net/?id=311(Maxon C4D System Requirements) ein 64 Bit System
[5]Apache Licence 2.0 https://github.com/PluginCafe/cinema4d_cpp_sdk/blob/master/LICENSE

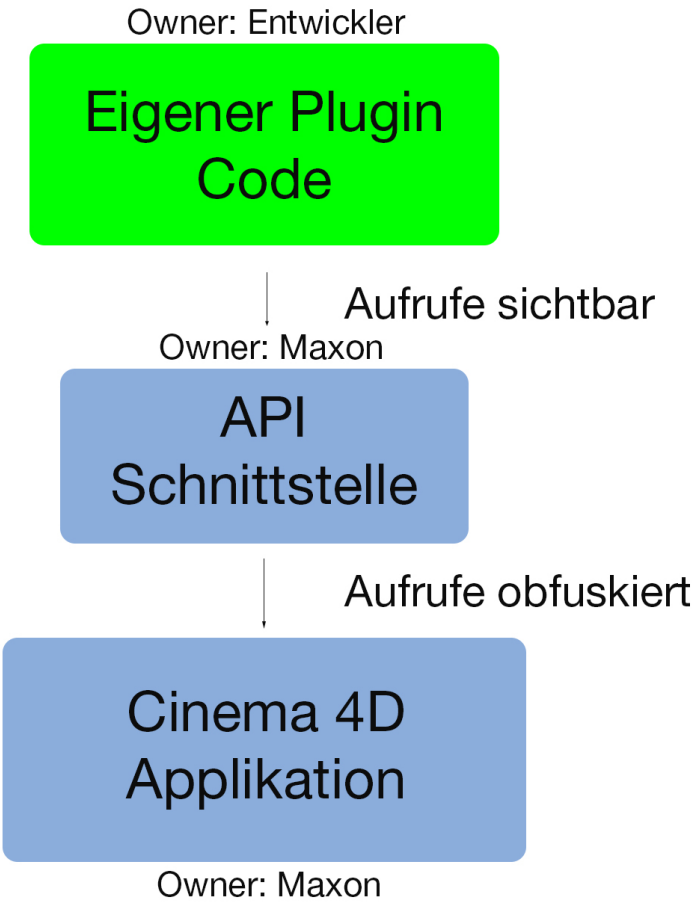

Abbildung 3.1: Schaubild des Cinema 4D API Systems

```
8   virtual Bool Execute(BaseDocument *doc)
9   {
10    GePrint("Eine Ausgabe zum Cinema 4D Konsolenfenster.");
11    return TRUE;
12  }
13 };
14
15 // Mit dieser funktion steigt das Plugin ein.
16 Bool PluginStart(void)
17 {
18   return RegisterCommandPlugin(ID_PLUGINID,"MinimalesPlugin"
       , 0,NULL,String("MinimalesPlugin"), gNew MinimalesPlugin)
       ;
19 }
20
```

```
21  // Wird beim beenden des Plugin aufgerufen.
22  void PluginEnd(void)
23  {
24  }
25
26  // Ist teil des Message Systems von C4d. Dient der
        Kommunikation.
27  Bool PluginMessage(LONG id, void *data)
28  {
29    return TRUE;
30  }
```

Das hier dargestellte Code Beispiel wird im nächsten Abschnitt relevant. Es folgt ein Überblick über das Uniplug Projekt welches das Schreiben von Cinema 4D Plugins in C# ermöglicht. Die oben gezeigte Codestruktur wird dann noch einmal als C# Version erläutert.

3.4.3 Uniplug

Bei Uniplug handelt es sich um ein Teil des Projektes FUSEE der Hochschule Furtwangen. Uniplug bietet die möglichkeit Cinema 4D Plugins in C# zu schreiben. Hierfür bedient es sich eines Interface Compilers SWIG [Beazley, David, S., Fulton, William. (1995-2015). SWIG-3.0. http://www.swig.org/]. Das gesamte Uniplug Projekt ist ein Open Source Projekt und steht auf GitHub Repository unter https://github.com/FUSEEProjectTeam/Fusee innerhalb des Fusee Projekts zum Download bereit. Das Uniplug Projekt stand bereits vor dieser Arbeit zur Verfügung. Allerdings hatte es nich den nötigen Umfang und das Projekt wurde während der Arbeit um wichtige Funktionen zur Entwicklung von Plugins erweitert.

Was ist SWIG?

Bei SWIG handelt es sich um einen SoftWare Interface Generator. Swig unterstützt Entwickler dabei, eine Codebase aus z.B. einer nativen Programmiersprache wie C++ in eine managed Programmiersprache wie z.B. C# zu "übersetzen". Hierbei handelt es sich allerdings nicht wirklich um einen übersetzungs bzw. Compilevorgang. Vielmehr unterstützt Swig den Entwickler durch das generieren spezifischer Interface Files welche die Aufrufe (calls) einer "externen" Software an den "gewarppten" Teil der Software weiterleiten. Die Ausgangssoftware wird also immernoch benötigt (im Falle von Uniplug) und auch verwendet (das Cinema 4D API Framework).

Diese kurze und knappe Beschreibung von SWIG beschreibt ddie Möglichkeiten des Tools kurz und knapp:

> SWIG is an interface compiler that connects programs written in C and C++ with scripting languages such as Perl, Python, Ruby, and Tcl. It works by taking the declarations found in C/C++ header files and using them to generate the wrapper code that scripting languages need to access the underlying C/C++ code. In addition, SWIG provides a variety of customization features that let you tailor the wrapping process to suit your application.
>
> Auszug aus: - Beazley, David, F. (2014). Swig introduction documentation.

Eine Vollständige Dokumentation des SWIG Projektes findet sich unter folgender Adresse: http://www.swig.org/Doc3.0/index.html geprüft, letzter Stand der Erreichbarkeit 15.04.2015.
Swig wird bereits erfolgreich in vielen Projekten eingesetzt. Unter anderem finden sich in der Liste das bekannte Version Control System Subversion, die 3D Engine Ogre bzw. PyOgre und die große Image Processing Bibliothek OpenCV. Die offizielle Liste der SWIG Nutzer findet sich unter http://www.swig.org/projects.html - geprüft am 15.04.2015.

Wrapping von C++ Code nach C# mit SWIG

Um aus dem C++ Code der Cinema 4D API aufrufbaren C# Code zu erzeugen sind verschiedene Schritte notwendig. Ein SWIG Projekt welches im Uniplug Projekt verfügbar ist enthält eine C4dApi.i (interface) Datei. Diese Datei kümmert sich um das Wrappen des nativen C++ Codes nach C#. Der eigentliche Wrapping Vorgang wird über ein SWIG make script aufgerufen welches während des Build Vorgangs des Projektes angestoßen wird. Dieser Vorgang muss bei der eigentlichen Pluginentwicklung nicht bei jedem Build durchgeführt werden. Es genügt, die API erneut zu wrappen wenn Veränderungen am API Code seitens Maxon oder des Benutzers vorgenommen wurden. Änderungen durch den Benutzer schliessen auch das erweitern der API Funktionalität mit ein. Das Uniplug Projekt ist zum Zeitpunkt dieser Arbeit weit von einem Vollständigen Wrapping der Cinema 4D API entfernt. Aus diesem Grund, ist noch relativ häufig die Erweiterung des API Projektes nötig. Verschiedene Eintragungen die für das Erweitern des Uniplug Projektes vorgenommen wurden, werden im nächsten Abschnitt genauer erläutert. Hierzu gehören einfache Inklusionen von Source Files, aber auch komplexe overrides von bereits bestehenden C++ Funktionen.

Das hier abgebildete Schaubild zeigt den Ablauf des Wrapping-Vorgangs. Hierbei sind folgende Schritte zu erkennen:

- Erstellen des *.i Files und inkludieren der gewünschten nativen Code Dateien.
- Erweitern von nativen Code Dateien um eigenen Nativen Code
- Überschreiben von nativen Code Dateien durch eigenen Nativen Code
- Kompilieren des SWIG Projektes.
- Verwenden des kompilierten SWIG Codes.

Die grauen Flächen im Diagramm beschreiben Aktionen in welche der Entwickler aktiv eingreifen muss. Das Interface File mit den gewünschten Dateien muss manuell geschrieben werden. Das erweitern oder überschreiben von Code muss ebenfalls manuell erfolgen. Der Aufruf des SWIG Compilers verläuft parallel zum Prozess des Erweiterns weil für eine Funktionsfähige wandlung des Codes keine Erweiterungen nötig sind. Es muss nur im Problemfall eingegriffen werden. Meist zeigt sich aber, dass in besonders komplexen Fällen, wie des wrappings der Cinema 4D API, doch recht häufig eingegriffen werden muss. Das Überschreiben (manuell) von Klassen und Methoden ist ein Optionaler Fall und wurde zum besseren Erkennen blau eingefärbt.

Erweiterung des Uniplug Codes für Plugins vom Typ TagPlugin

Um das Projekt auf den Aktuellen Stand zu bringen und das schreiben des Plugins zu ermöglichen musste zuerst das Uniplug Projekt erweitert werden. Auch hier wurde mit iterativen Methoden gearbeitet. Falls Probleme mit verschiedene gewrappten Cinema 4D API Methoden auftragen, wurde zunächst geprüft, welche “pseudo” SWIG Dateien diese Probleme verursachten und welche Cinema 4D Api Funktionen aus diesem Grund in der C# API Code nicht vorhanden sind. Diese fehlenden Dateien wurden dann inkludiert und es wurde versucht das SWIG Projekt zu bauen. Sollte es bei diesem Schritt zu Problemen kommen, mussten die API Dateien genauer analysiert werden. In manchen Fällen war es nun nötig den nativen Code der API Dateien zu erweitern oder zu überschreiben. Bei der Implementierung des Cinema 4D Plugin Typs “TagPlugin” waren besondere Änderungen im API Code nötig. Hier sollen die Problematik und die Lösung dieser aufgezeigt und erläutert werden.

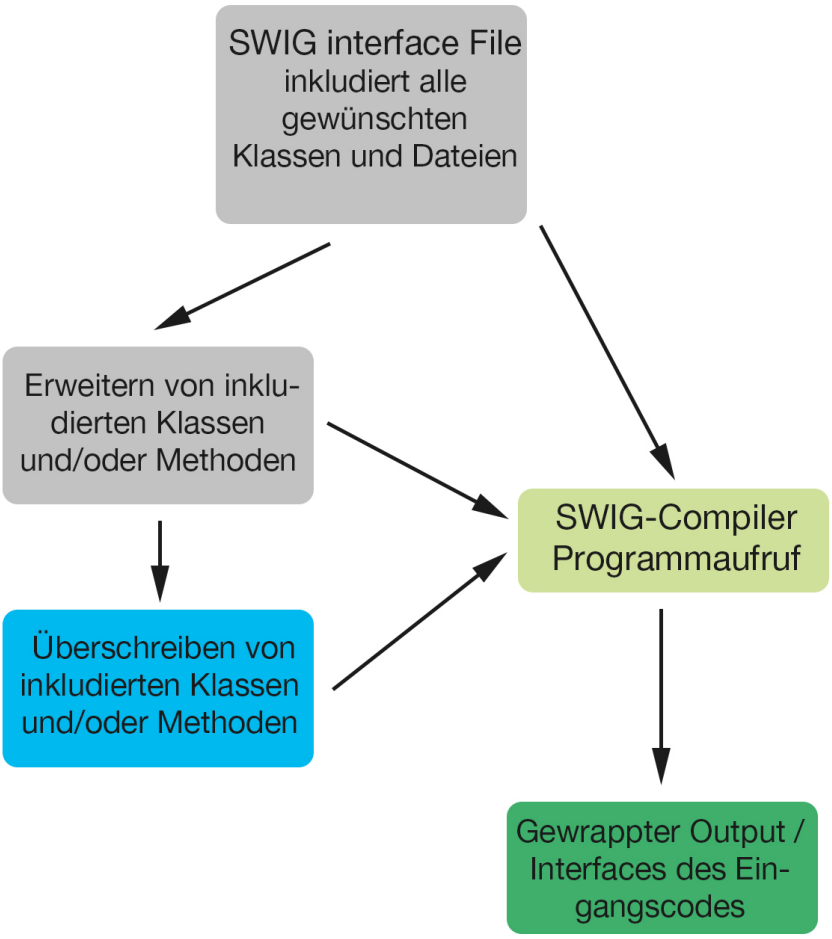

Abbildung 3.2: Wrapping Vorgang der Cinema 4D API von C++ nach C#

Sourcecode Beispiele 3.2: Einbinden des Maxon Cinema 4D C++ Datentyps TagPlugin in die C# API von Uniplug

```
1  #include "c4d_tagdata.h"
2  #include "c4d_tagplugin.h"
3
4  // "c4d_tagdata.h"
5  //%include "c4d_tagdata.h";
6  %feature("director") TagDataM;
7  %csmethodmodifiers TagDataM::TagDataM "private";
8  %typemap(cscode) TagDataM %{
9    public TagDataM(bool memOwn) : this(C4dApiPINVOKE.
       new_TagDataM(), memOwn) {
10     SwigDirectorConnect();
11   }
12 %}
```

```
13 %include "c4d_tagdata.h";
14 %include "TagDataM.h";
15
16 // "c4d_tagplugin.h"
17 //%include "c4d_tagplugin.h";
18 %feature("director") TagPlugin;
19 %csmethodmodifiers TagPlugin::TagPlugin "private";
20 %typemap(cscode) TagPlugin %{
21   public TagPlugin(bool memOwn) : this(C4dApiPINVOKE.
     new_TagPlugin(), memOwn) {
22     SwigDirectorConnect();
23   }
24 %}
25 %include "c4d_tagplugin.h";
```

Um den Datentyp TagData korrekt im C# Code zu verwenden, waren noch weitere Anpassungen im Code notwendig. Die Methode Message() des TagData ElternTyps NodeData musste Überschrieben werden um das Message System in C# nutzen zu können. Dies wurde nötig durch die Verwendung eines Void Pointers seitens Maxons in der Parameterliste der Message Funktion im C++ Code der API. SWIG kann an dieser Stelle durch den nicht auf einen Datentypen festgelegten Void Pointer des Datentyps der Parameterliste keine zufriedenstellende Wrappingfunktion bereit stellen. Das folgende Codebeispiel der Message() Funktion verdeutlicht den Fix im C++ Code des Uniplug Native Projektes.

Die Message() Funktion ist Teil eines Nachrichtensystems innerhalb von Cinema4D. Viele Elemente des Programs und selbst geschrieben Plugins kommunizieren über dieses Messagesystem miteinander. Hierbei werden Messages über einen Broadcast versandt. Dieser Broadcast sendet in einer Parameterliste einen ID Code. IdCodes von Cinema 4D werden meist als Konstante (mit define angelegte) Integer Variablen übergeben und können so in if else oder switch case Anweisungen vom Plugin Entwickler identifiziert und verarbeitet werden.

Im folgenden Code Beispiel sind einige dieser konstanten int codes dargestellt. Bei dem hier abgebildeten Code handelt es sich um die überschrieben Message Funktion, welche die Verwendung des Systems für Plugins in C# erst möglich macht. Der Voidpointer der Parameterliste wird während des Aufrufs der Funktion in andere Datentypen gecastet. Für die Wandlung der Message Funktion für die TagData Klasse war es von nöten ein Object des Typs Do-

cumentInfoData zu erhalten. Aus diesem Grund, wird das Objekt dieses Typs bei einer bestimmten empfangenen Message ID zurück gegeben.

Sourcecode Beispiele 3.3: Überschreiben der Message Funktion des TagData Datenty<ps.

```
1  Bool TagDataM::Message(GeListNode *node, Int32 type, void *
       data)
2  {
3    switch (type)
4    {
5    case MSG_EDIT:
6
7      break;
8    case MSG_GETCUSTOMICON:
9      break;
10   case COLORSYSTEM_HSVTAB:
11     break;
12   case MSG_DOCUMENTINFO:
13     {
14       DocumentInfoData* did = (DocumentInfoData*)data;
15       return MessageDocumentInfo(node, did);
16     }
17     break;
18   case MSG_DESCRIPTION_GETINLINEOBJECT:
19     break;
20   case DRAW_PARAMETER_OGL_PRIMITIVERESTARTINDEX:
21     break;
22   }
23
24   return true;
25 }
```

Durch die eingeschränkte Möglichkeit nur bis zum call des Debugging API Codes debuggen zu können (Hierzu mehr im Kapitel Probleme und Herausforderungen), war die Lösung des Problems nicht wie meistens in mit SWIG gewrappten Codebasen durch einen einfachen %typemap[6] sondern nur durch die Nachimplementierung der oben dargestellten switch case Anweisung zu bewerkstelligen.

[6]Das mappen eines Datentyps der Eingangssprache auf einen anderen Datentyp der Ausgangssprache

3.4.4 Fusee

Der Fusee Szenengraph

Das Fusee Level, Welt wie auch immer es hier bezeichnet werden sollte. Eine Basis wird gebraucht. Hierzu eine Zentrale anlaufstelle, ein Spiele “Kernel”? Irgend etwas dass im Zentrum steht. Alles andere muss auf Level etc aufgeteilt werden und bis zum einzelnen Asset heruntergebrochen werden.

3.5 Das eigentliche Plugin

3.5.1 Visualisierung der Systemarchitektur

Schaubilder und Grafiken sind hier nützlich. Abhängigkeiten und Aufteilung sinnvoll gestalten.

Welche Programme und Systeme sind beteiligt?

Kette aufzeigen von Cinema4D nach Fusee Authoring Toolkit.

3.5.2 Generieren eines Fusee Projektes

Ablauf erklären.

3.5.3 Code Generation und die Vermeidung von Roundtrips (nicht so ganz roundtrips, generierung um generierung etc.)

3.5.4 Partial Classes in .NET

Sind bis jetzt noch nicht verwendet worden.

3.5.5 XPresso Schaltungen - Visual Programming / Programmieren ohne Programmieren

Ein Ausblick auf die Zukunft. Eventuell hier den Artikel anführen welcher über Visuelle Programmierung berichtet hat.

Probleme und Herausforderungen während der Entwicklung

4 Ergebnisse und Erkentnisse

4.1 Game Authoring Entwicklungsprozesse jetzt und in Zukunft

4.2 Wie weit ist die Implementierung fortgeschritten?

4.3 Welcher Mehrwert wurde erreicht?

4.4 Integration des Systems in den weiteren Projektverlauf von FUSEE

Anhang

Verzeichnis der Sourcecode Beispiele

Tabellenverzeichnis

Abbildungsverzeichnis

UML Diagramme

Literaturverzeichnis

Abrash, M. (1997). *Graphics Programming Black Book*. The Coriolis Group Inc.

Autodesk Inc. (2014-2015). Autodesk Scaleform. http://gameware.autodesk.com/scaleform.

Autodesk, Inc. (2015). 3DS Max. http://www.autodesk.de/products/3ds-max/overview.

Beazley, David, F. (2014). Swig introduction documentation.

Beazley, David, S., Fulton, William. (1995-2015). SWIG-3.0. http://www.swig.org/.

Bethke, E. (2003). *Game Development and Production*. Wordware Publishing Inc.; Auflage: Pap/Cdr (Februar 2003).

Blender Foundation. (2014-2015). Blender 2.74. http://www.blender.org/.

Carter, B. (2004). *The Game Asset Pipeline*. Delmar Cengage Learning.

CCP Games. (2009). Eve online fanfest 2009 - scrum and agile development processes.

Chandler, H. (2006). *The Game Production Handbook*. Thomson Delmar Learning.

Cifaldi, F. (2005). E3 report: the path to creating aaa games.

Florian, Mehm, S. R., Christian,Reuter. (2014). Future trens in game authoring tools. S. 536-541.

Freeman, W. (2014). Kingdom come: deliverance video update 9.

Gamma, E., Johnson, R., Helm, R. & Vlissides, J. (2014). *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Deutschland.

Gergory, J. (2009). *Game Engine Architecture*. Taylor & Francis Ltd.

Guerrilla Games. (2004, 2007, 2011, 2013, 2013). Killzone 1, Killzone 2, Killzone 3, Killzone: Mercenary, Killzone: Shadow Fall. http://www.guerrilla-games.com/. Sony Computer Entertainment.

Lengyel, E. (2012). *Mathematics for 3D Game Programming and Computer Graphics*. Course Technology, Cengage Learning.

Martin Klekner, D. V., Votja Nedved. (2014). Kingdom come: deliverance video update 9.

MAXON Computer GmbH. (2014-2015). Cinema 4D R16. http://www.maxon.net/de/products/cinema-4d-studio.html.

NaughtyDog. (2007, 2009, 2011). Uncharted. http://www.naughtydog.com/games/uncharted_drakes_fortune/. Sony Computer Entertainment.

Nilsson, T. (2014). Sony releases free, open source game development toolset.

Quantic Dream. (2013). Beyond: Two Souls. http://www.quanticdream.com/en/. Sony Computer Entertainment.

Schmitz, C. (2013). The art of waiting.

Schmitz, C. (2014). Projektmanagement mit scrum. Available online - last checked 07.04.2015 http://www.makinggames.biz/features/projektmanagement-mit-scrum,2534.html.

The Foundry Visionmongers Ltd. (2015). Modo. https://www.thefoundry.co.uk/products/modo/.

Wawro, A. (2014). Sony releases level editor that's open source and engine-agnostic.

Wihlidal, G. (2006). *Game Engine Toolset Development*. Thomson Course Technologies.

SOFTWARE ENTWURF FUSEE AUTHORING TOOLS FÜR CINEMA 4D

Name: Fusee Authoring Tools

Autor: Master-Thesis an der Hochschule Furtwangen von Dominik Steffen WS14/15

Version: 0.2

EINLEITUNG

Die Fusee Authoring Tools for Cinema 4D entstehen während einer Masterarbeit an der Hochschule Furtwangen. Das Projekt wird von Dominik Steffen (Medieninformatik) umgesetzt. Der Bearbeitungszeitraum erstreckt sich auf das Wintersemester 14/15.

Dieses Dokument beschreibt die Anforderungen an das Projekt Fusee Authoring Tools for Cinema 4D. Unter der Berücksichtigung einer Stakeholder Analyse unter Game Developern werden verschiedene Funktionen des Tools herausgearbeitet und beschrieben.

UMFANG DER ARBEIT

Das Softwareprojekt wird als Plugin für Cinema 4D und als Bibliothek und Ergänzung des bereits bestehenden Uniplug Projektes im Rahmen des Fusee Game Engine Projektes entwickelt. Uniplug bietet bereits ein gewisses Maß an basis Funktionalität ium Plugins für Cinema 4D in .Net bzw. C# zu erstellen. Diese Funktionalität wird erweitert um dem Funktionsumfang der Anforderungen zu entsprechen.

ERLÄUTERUNGEN ZU BEGRIFFEN UND / ODER ABKÜRZUNGEN

FUSEE / Fusee = Furtwangen University Simulation and Entertainment Engine

C4D = Cinema 4D, Software von Maxon

Fusee Authoring Tools for Cinema 4D = Plugin welches im Rahmen dieses Projektes entsteht

Fusee Authoring Tools = Bibliothek welche im Rahmen dieses Projektes entsteht

Uniplug = Bereits bestehendes Projekt welches das Erstellen Plugins mit Hilfe von C# für Cinema 4D erlaubt.

VERWEISE UND QUELLEN

Maxon Developer Support: https://developers.maxon.net

Maxon Developer Forum: http://www.plugincafe.com/forum/default.asp

Fusee GitHub Projekt: https://github.com/FUSEEProjectTeam/Fusee

ALLGEMEINES

Das Fusee Authoring Tools Projekt ist eine Bibliothek die es ermöglicht Plugins (und Ähnliches) für 3D Modelling Software zu schreiben welche diese Software um Authoring Tool Funktionen erweitert. Dieses Dokument beschäftigt sich mit den Anforderungen der Software Bibliothek selbst und der Anbindung an Cinema 4D.

Die Software soll es ermöglichen, dass Artists, Designer und Engineers an ein und dem selben Projekt arbeiten können ohne die gewohnte eigene Arbeitsumgebung (3D-Modellierungssoftware, Developer IDEs etc.) zu verlassen und etwas komplett neues zu erlernen.

Das Projekt wird als Teil des Fusee Engine Projekts entwickelt. Das Projekt ist ein Teil des Fusee Engine Uniplug Projektes.

Jeglicher Code ist Open Source.

Hier könnte ein Vergleich zu anderen Game Engine Tools stehen.

PRODUKTFUNKTIONEN

- Projekt anlegen
- Projekt bearbeiten (Durch hinzufügen von Assets oder Funktionalität)
- Projekt öffnen
- Projekt speichern
- Projekt bauen

BENUTZERANALYSE

Bei den Stakeholdern des Projektes handelt es sich in erster Linie um Mitglieder des Teams welche durch die Entwicklung des Tools direkt betroffen sind.

Hierzu gehören in erster Mitarbeiter in Folgenden Positionen:

- Artist (Arbeit in 3D Modellierungssoftware)
    - Animator
    - World Builders
- Engineer (Software Entwickler, Programmierer)
    - Tools
    - Graphics
    - Network
    - AI
    - Sound
- Game Designer (Entwickeln und umsetzen von Spielideen)
    - Level
    - Scripter (Game Logic Scripts)
    - UI
- QA Tester (Testen des Spiels als Builds und in der Entwicklungsumgebung)

RESTRIKTIONEN

- Das gesamte Projekt nutzt die Basis des Uniplug Projektes.
- Das Projekt wird fast komplett in C# und maximal geringe Teile in C++ entwickelt.
- Die Entwicklungsdauer ist begrenzt auf Teile des Wintersemesters 14/15.
- Das Projekt muss auf Windows 8 lauffähig sein.
- Der Code muss auf GitHub zur Verfügung gestellt werden.

PROJEKTABHÄNGIGKEITEN

- Das Projekt ist auf Grund verschiedener Abhängigkeiten im Fusee Uniplug Projekt aktuell auf Windows beschränkt.
- Das Projekt ist auf C# beschränkt.
- Das Projekt nutzt eine gewrappter Version des Maxon C++ SDK.

ANFORDERUNGEN

FUNKTIONALE ANFORDERUNGEN UND USE CASES

Hierbei handelt es sich um spezifische Funktionalität welche die Fusee Authoring Tools Bibliothek anbieten soll:

- Projekt anlegen
    - Visual Studio Solution Datei öffnen
        - Inhalt einlesen

      Visual Studio Solution Datei bearbeiten
        - Korrekte Stellen des einzufügenden Codes finden
        - Projekt anlegen und nötigen XML Code hinterlegen
    - Visual Studio Solution Datei speichern
- Projekt bearbeiten (Durch hinzufügen von Assets oder Funktionalität)
    - Neuen Code hinzufügen
        - Klassen anlegen
        - Klassen als Partial Classes anlegen
    - Neue Assets hinzufügen
        - Bilder
        - Modelle
        - Sound
- Projekt öffnen
    - Ein bestehendes Projekt in Cinema 4D öffnen
- Projekt speichern
    - Ein Projekt von Cinema 4D heraus speichern
    - Unnötige Dateien (Temporärer Natur) nach dem Speichern entfernen
- Projekt bauen
    - Ein Projekt von Cinema 4D heraus bauen

NICHT-FUNKTIONALE ANFORDERUNGEN

Das Projekt muss während des Projektzeitraums des Wintersemesters 2014/2015 mit einer ausreichenden Basisfunktionalität erstellt werden.

Das Projekt muss mit Sandcastle oder ähnlichem Dokumentiert werden und die Dokumentation muss dem Projekt beigelegt werden.

SCHNITTSTELLEN DER BIBLIOTHEK

Die Fusee Game Authoring Tool Bibliothek bietet eine Schnittstelle um Plugins für unterschiedliche Modellingsoftware zu erstellen. Die Bibliothek ist nicht von Cinema 4D Funktionen abhängig sondern unterstützt Windows spezifische C# Funktionalität.

DESIGN CONSTRAINTS

Das Interface wird nicht großartig frei gestaltet. Es orientiert sich an den gegebenen Cinema 4D Interface Optionen und nutzt die Cinema 4D SDK API um die Interface Elemente darzustellen.

QUALITÄTSANFORDERUNGEN

Das Projekt wird im Alpha Stadium beendet und stellt daher nur einen gewissen Pool an Basisfunktionalitäten zur Verfügung. Das Projekt kann Problemlos um weitere Funktionalität erweitert werden und es wird das Fusee und das Uniplug Projekt weiterhin begleiten.