Nutzung der Game Engine *Unity* als Charakter-Animationstool für Filmproduktion

Eva-Maria Hobl

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Digital Arts

in Hagenberg

im Januar 2020

Betreuung:

Designer FH Alexander Wilhelm

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 22. Januar 2020

Eva-Maria Hobl

Gender Erklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Arbeit die Sprachform des generischen Maskulinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

Inhaltsverzeichnis

Kurzfassung

Diese Arbeit beschäftigt sich mit den Möglichkeiten der Charakter-Animation in einer Game Engine. Im Speziellen wird auf die bereits vorhandenen Funktionen der Game Engine *Unity* eingegangen und welche weiteren Funktionen für eine ideale Umgebung zur Charakter-Animation nötig wären. Zu diesem Zweck werden das *Unity*-Plugin *UMotion* – das der Game Engine einen Animationsbereich hinzufügt – und die Machinima-Software *Source Filmmaker* genauer betrachtet und die für Animatoren sinnvollsten Funktionen hervorgehoben. Des Weiteren werden die Animationen mehrere Videospiele analysiert und besondere Vorgehensweisen bei der Erstellung derselben beschrieben. Die Erkenntnisse des analytischen Teils, wurden im Kurzfilm-Projekt *The Sapling* genutzt, um eine Pipeline zur Charakter-Animation in der derzeitigen Version von *Unity* zu entwickeln.

Abstract

This work deals with the possibilities of character-animation in a game engine. In particular, the already existing functions of the game engine *Unity* and which further functions would be necessary for an ideal environment for character animation are discussed. For this purpose, the *Unity*-plugin *UMotion* — which adds a separate animation area to the game engine — and the Machinima-Software *Source Filmmaker* are examined in detail and their most useful functions for animators are highlighted. Furthermore, the animations of several video games will be analyzed and special procedures for creating the animations will be described. The findings from the analysis section were used in the short film *The Sapling* to develop a pipeline for character-animation in the current version of *Unity*.

Kapitel 1

Einleitung

Der technische Fortschritt in der Unterhaltunsindustrie im Bereich der Videospiele ermöglicht mittlerweile aufwendige Produktionen, die in Echtzeit auf verschiedenen Konsolen und Computern funktionieren. Neben möglichst realitätsnahem Rendering und immer besserer Optimierung der Performanz der Spiele, ist auch Animation – speziell die der Protagonisten des Spiels – ein wichtiger Aspekt, der für mehr Immersion sorgt. Spieleentwickler greifen neben Motion Capture Aufnahmen auf verschiedene Methoden zur automatisierten Generierung von organischen Reaktionen der Charaktere auf verschiedene Einflüsse wie Umgebung und Spielereingabe zurück. Außerdem stehen für die Verwendung in Videospielen eine Vielzahl an fertigen Animationen (zumeist Motion Capture Aufnahmen), die in Schleifen abgespielt werden können, zur Verfügung, die vor allem für Studios, denen die Mittel für eigens produzierte Aufnahmen fehlen, von Vorteil sind. Diesen Aufnahmen fehlt zwar der nötige Eigencharakter einer Filmanimation, könnten aber eine Basis für effizienteres Arbeiten bieten und in Kombination mit Animation von Hand interessante stilistische Eigenschaften ergeben.

Auch Filmschaffende haben in den letzten Jahren begonnen mit Game Engines [33] zu arbeiten, die ein Grundgerüst für die Produktion von Spielen bieten, inklusive einer zumeist sehr starken Grafiksimulation. Die frei nutzbare Game Engine *Unity*[1] unterstützt verstärkt Filmschaffende mit Erweiterungen wie dem Kamerasystem „Cinemachine“ und der Möglichkeit der Echtzeit Post Produktion. *Unity* selbst produziert Tech Demos in Form von Kurzfilmen, die immer möglichst realistisch gerendert sind und somit das Potenzial der aktuellen Version der Game Engine ausreizen. Des Weiteren entstand über Zusammenarbeit mit Disney [30] die Kurzfilmreihe *Baymax Dreams* [14] die im typischen Stil der Disney 3D-Animationsfilme gerendert ist. Unabhängig von dem Unternehmen *Unity* selbst, nutzte ein Team von Filmschaffenden die Game Engine um den Kurzfilm *Sonder* [11] zu produzieren.

Diese Beispiele zeigen, dass Filmproduktion in *Unity* schon möglich ist. Eine standardmäßige Pipeline zur Integration von Charakter Animation in einem Echtzeitfilm muss sich noch herauskristallisieren und es wurden bestimmt noch nicht alle Möglichkeiten der Game Engine ausgenutzt. In den genannten Filmen wurden Charakter-Animationen beispielsweise über *Autodesk Maya*[2] und Motion Capture Verfahren reali-

[1] https://unity.com/

[2] https://www.autodesk.de/products/maya/overview

siert. Dabei wurde alles in einer externen Software finalisiert und dann in *Unity* eingefügt, obwohl die Game Engine sowohl Rigs als auch die Keys der einzelnen Animationen erkennt. Allerdings ist die Anpassung dieser nicht sehr benutzerfreundlich und dementsprechend ist *Unity* in diesem Bereich noch ausbaufähig.

1.1 Fragestellung

Es stellt sich nun die Frage, welche Funktionen *Unity* aus Sicht eines Animators aufweisen sollte und ob es dann genutzt werden kann, um in der Game Engine zu animieren bzw. Charakter-Animationen aus einer externen Software zu finalisieren? Können des Weiteren Konzepte zur automatisierten Animation aus der Spieleentwicklung für Filmanimation von Nutzen sein?

1.2 Methodik

Zu Beginn werden Charakter-Animationsmethoden und -konzepte sowohl aus der Filmproduktion als auch aus der Spieleentwicklung recherchiert und zusammengefasst. Für Konzepte aus der Spieleentwicklung soll auch nach bestehenden Werkzeugen sowohl von *Unity* selbst, als auch von Dritten gesucht und diese anschließend in Testumgebungen in der Game Engine getestet werden. Im erweiterten Sinne sind hier auch externe Werkzeuge die Rigging und Animation vereinfachen von Interesse. Dazu gehören beispielsweise *UMotion*[3] – ein Plugin für *Unity* das Animieren in der Game Engine ermöglicht – und *Akeytsu*[4] – hiermit soll eigenständiges Riggen und Animieren vereinfacht werden. Im Speziellen werden auch Cinematics aus Videospielen betrachtet, da hierfür oft Charaktere aus dem Spiel verwendet werden. Potenziell interessante Spiele sind hierfür: *The Witcher 3: Wild Hunt* [24], *World of Warcraft* [22] und *Overwatch* [21].

Nach dieser Recherche- und Testphase werden die entsprechend anwendbaren Methoden in einer Szene im Projekt *The Sapling* angewandt. Anhand der Arbeit an dieser Szene sollen die angewandten Techniken in ihrer Praktikabilität bewertet und die sich ergebende Ästhetik analysiert werden.

[3] https://www.soxware.com/umotion/
[4] https://www.nukeygara.com

Kapitel 2

Grundlagen von Rigging und Animation

Der Begriff Animation bzw. Animationsfilm ist sehr breit gefächert. Animation bedeutet grundsätzlich eine schnelle Abfolge von Einzelbildern, die so für die menschliche Wahrnehmung eine Bewegung ergeben. Wie diese Einzelbilder erstellt werden kann sich stark unterscheiden: Im 2D-Trickfilm werden die Einzelbilder – auch Frames genannt – von Hand gezeichnet, bekannte Beispiele hierfür sind *Disney*-Klassiker wie *Schneewittchen* [13] oder *König der Löwen* [6]. Eine weitere bekannte Methode der Animation ist das Stop-Motion-Verfahren, bei dem verschiedenste Gegenstände arrangiert und dann fotografiert werden. Zwischen jedem Foto werden die Gegenstände minimal verändert, sodass schlussendlich beim schnellen Abspielen der Fotos die Illusion von Bewegung entsteht. Beispiele für Stop-Motion-Filme sind *Nightmare Before Christmas* [10] von *Tim Burton* und die Serie *Wallace und Gromit* [12] von *Aardman Animations* [44].

Diese Arbeit konzentriert sich auf 3D-Animation also solche, die mittels einer Software am Computer erstellt wurden. Computergenerierte Animation findet sich nicht nur in Filmen (z. B. *Findet Nemo*), sondern auch in Videospielen (z. B. *The Witcher 3: Wild Hunt* [24]), wobei sich die Herangehensweisen und Produktions-Pipelines größtenteils ähneln, schlussendlich jedoch unterschiedlichen Ansprüchen entsprechen müssen. So wird ein 3D-Animationsfilm normalerweise mithilfe einer Render-Engine aus der Animations-Software gerendert, die 3D Szene also in ein 2D-Bild umgewandelt [1]. Dies geschieht für jeden einzelnen Frame des Films und kann bei aufwendigen Simulationen auch mehrere Minuten pro Bild dauern. Großen Animationsstudios stehen hierfür ganze Render-Farmen zur Verfügung, also eine große Anzahl an leistungsfähigen Computern, die ausschließlich zum Rendern genutzt werden. Videospiele wiederum müssen alles in Echtzeit rendern, weshalb normalerweise die Qualität der 3D-Inhalte reduziert werden muss, um dennoch die entsprechende Leistung zu bringen. Trotz der augenscheinlichen Unterschiede gelten jedoch sowohl für Animationen im Film als auch in Videospielen die sogenannten Animationsprinzipien und auch die verschiedenen Herangehensweisen an das Animieren der Charaktere können in beiden Medien angewandt werden.

2.1 Rigging

Um ein 3D Objekt zu animieren muss ein sogenanntes „Rig“, das die Geometrie des Objekts beeinflusst, erstellt werden. Dieses Rig besteht aus einer Reihe hierarchisch an-

geordneter Joints, die wie die Knochen eines Skeletts angeordnet sind. Wie bei einem Skelett auch, bewegen sich die Joints nur an ihrem Startpunkt, der das ungefähre Äquivalent zu einem Gelenk darstellt. Die Geometrie wird, wenn das Rig fertig aufgebaut ist, daran gebunden („Skinning“). Über „Weight Painting“ wird festgelegt, wie stark jeder Vertex von den einzelnen Joints beeinflusst wird.

Neben Joints können auch Blend Shapes oder sogenannte „Morphs“ erstellt werden. Die Deformation der Geometrie geschieht über „Target Shapes“, die dieselbe Anzahl Vertices wie das ursprüngliche Mesh besitzen (zumeist wird das ursprüngliche Mesh dupliziert und anschließend verändert). Die Target Shapes können mit Sculpting und Modelling Tools angepasst werden, sodass sie z. B. im Falle eines Gesichts lächeln oder die Brauen zusammenziehen. Blend Shapes werden vorrangig für Gesichtsanimation genutzt, können aber auch als Korrekturhilfen dienen, wenn bei Animationen Fehler in der Geometrie entstehen, die allein über Weight Painting nur schwer zu beheben sind.

Um das Animieren des geriggten Charakters zu vereinfachen, wird das Rig aus Joints mithilfe von Constraints gesteuert. Constraints sind weitere 3D Objekte deren eigene Position, Rotation und manchmal auch Skalierung, die des an sie gebundenen Joints steuern. In den meisten 3D Programmen ist es auch möglich Attribute, die nicht derselben Kategorie angehören zu verbinden. So kann z. B. ein Blend Shape Attribut, das die Brauen des Charakters hebt und senkt, über die Y-Position (vertikale Achse) eines Constraint-Objekts gesteuert werden. Dies ermöglicht zumeist ein intuitiveres Arbeiten für den Animator.

Eine besondere Form von Contraints sind IK-Contraints. IK steht für Inverse Kinematik und bedeutet, dass eine Hierarchie von Objekten vom letzten Objekt aus gesteuert wird. In den meisten Fällen handelt es sich um IK-Contraints für Arme und Beine, die es ermöglichen die Extremitäten von den Handgelenken bzw. Knöcheln aus zu steuern. Knie und Knöchel beugen und strecken sich automatisch, je nach Position von Endgelenk zu Schulter bzw. Hüfte. Die Richtung, in die sie sich abbiegen, wird durch einen Pole Vector Constraint festgelegt. Da der IK-Constraint nicht von anderen Objekten beeinflusst wird, kann er den entsprechenden Joint an einer bestimmten Stelle „festhalten“. Dies ist vor allem für die Animation von Schritten nützlich, um ein Gleiten der Füße zu verhindern, oder wenn ein Charakter sich an etwas festhält. Ein IK-Constraint kann auch auf eine größere Hierarchie von Joints angewandt werden, z. B. auf die Jointkette die den Rücken bzw. die Wirbelsäule ergibt. Das Beugen und Strecken der Joints wird dann vom Joint am Kopfansatz gesteuert.

2.2 Keyframe Animation

Animiert ein Animator die Animation von Hand über das Setzen von Keyframes in einer Animationssoftware wird dies als Keyframe Animation bezeichnet. Hierbei gibt es Ähnlichkeiten zur analogen Trickfilm-Animation, da der Animator selbst die Extremposen und Inbetweens der Animation animiert. Im Gegensatz zur analogen Animation stehen dem 3D-Animator jedoch verschiedene Hilfen der Animationssoftware zur Verfügung. So gibt es Funktionen zur Interpolation zwischen einzelnen Keyframes, wodurch das Setzen jedes einzelnen Frames obsolet wird. Außerdem ist eine nachträgliche Anpassung der einzelnen Posen um ein Vielfaches leichter, weshalb die einzelnen Posen nicht gleich zu Beginn bis in die kleinsten Details festgelegt werden müssen. Daraus ergeben sich

Abbildung 2.1: Motion Capture Aufnahmen für den Kurzfilm *ADAM* [15] von *Unity*. Mithilfe von Requisiten werden die Gegebenheiten im Film nachgebaut. Bildquelle [43].

leicht veränderte Herangehensweisen die sich vom Straight-Ahead und Pose-to-Pose[1] Prinzip ableiten [1].

Das Hybrid-Konzept [1, S. 208] beschreibt eine gemeinsame Nutzung beider Prinzipien, die in der 3DAnimation zumeist genutzt wird. Dabei werden zunächst die Extremposen und zumeist auch einige Inbetweens nach dem Pose-to-pose Prinzip erstellt und durch die Animationssoftware interpoliert. So haben der Animator und der Regisseur ein erstes Layout der Animation und können grundsätzliches Timing, Fluss und Bewegungsmuster der Animation anpassen, ohne alles überarbeiten zu müssen. Anschließend arbeitet der Animator Straight-ahead über die gesamte Animation, um sie zu finalisieren und Overlapping Action zu animieren [1].

Aus der Möglichkeit der nachträglichen Überarbeitung heraus, hat sich außerdem das Hierarchie-Prinzip entwickelt. Hierbei werden die einzelnen Teile des Körpers hierarchisch nacheinander animiert. Bei einem Walk-Cycle (Animation eines zwei Schritte gehenden Charakters, die in Schleifen abgespielt wird, wodurch der Charakter beliebig viele Schritte gehen kann) werden z. B. zuerst die Hüfte und die Beine animiert. Ist die Animation derselben zufriedenstellend, folgen Torso und Kopf. Schlussendlich nacheinander Schulter, Ellbogen, Handgelenk und die einzelnen Finger. Diese Art und Weise zu animieren eignet sich vor allem für Cycle Animationen, also solche die in Schleifen abgespielt werden, da so der Fluss und der nahtlose Übergang vom letzten zum ersten Frame Körperteil für Körperteil sichergestellt wird [1].

[1]Das vierte der 12 Prinzipien der Animation von *Disney* beschreibt die Straight-Ahead- und die Pose-to-Pose-Methode. Bei Straight-Ahead zeichnet der Animator einen Frame nach dem Anderen, ohne genauer im Voraus zu planen. Dies ergibt fließende und manchmal auch kasperhafte Bewegungen und erlaubt dem Animator spontane Ideen umzusetzen. Bei Pose-to-Pose werden die Aktionen und Bewegungen über die Extrem-Posen geplant bzw. festgelegt und im Anschluss die dazwischenliegenden Frames („Inbetweens“) animiert [2, S. 56].

2.3 Motion Capture

Motion Capture ist die Aufnahme der Bewegung eines Schauspielers und die Übertragung der gewonnenen Daten auf ein Rig in einer entsprechenden Software. Die Erfassung der Bewegung geschieht meistens mit einem Marker-System, bei dem der Schauspieler einen Anzug mit den namensgebenden Markern trägt. Mehrere Kameras zeichnen dies auf, wodurch eine dreidimensionale Abbildung der Aufnahme erstellt werden kann (siehe Abb. 2.1). Es gibt auch markerlose Systeme, bei denen der Anzug selbst die Bewegung erfasst und überträgt. Dies ist vor allem dann sinnvoll, wenn der Bewegungsradius durch die bei einem Marker-System benötigten Kameras zu stark eingeschränkt wäre oder der Aufbau von Kameras nicht möglichst ist. Unabhängig davon mit welchem System die Bewegung aufgenommen wurde, müssen am Computer zumeist noch etwaige bei der Aufnahme entstandenen Fehler korrigiert werden. Des Weiteren sollte auch hier das Prinzip der Exaggeration[2] angewandt werden, die Animation also noch verstärkt bzw. übertrieben werden.

Neben der Aufnahme der Bewegung des Körpers können auch Bewegungen im Gesicht über Detailaufnahmen immer genauer erfasst und übertragen werden. Dies kann sowohl getrennt voneinander als auch gemeinsam geschehen. Dazu wird eine kleine Kamera, die am Kopfteil des Motion Capture Anzugs befestigt ist, vor dem Gesicht des Schauspielers angebracht. Auch hier wird zumeist mit Markern gearbeitet – an den benötigten Stellen aufgemalte schwarze Punkte – es gibt aber auch hier schon Möglichkeiten zur markerlosen Aufnahme.

Bis vor kurzem war Motion Capture in Echtzeit noch schwierig zu bewerkstelligen. Inzwischen gibt es Systeme, die dies ermöglichen und auch direkt in Game Engines übertragen. Unter anderem ist dies mit der „Live Face“ [37] Funktion des iPhone X möglich, wenn auch nicht in der Qualität einer durch Marker generierten Aufnahme.

Sind die Ressourcen für eine eigene Motion Capture Aufnahme speziell für Videospiele nicht vorhanden, so kann auch auf eine der im Internet zur Verfügung stehenden Motion Capture-Sammlungen zurückgegriffen werden. Eine der größten ist Mixamo [40] das zusätzlich ein automatisches Rigging-Werkzeug beinhaltet. Aufnahmen aus einer solchen Sammlung können auch eine Basis für anschließend dem Charakter angepasste Animationen bieten.

[2] Exaggeration ist das zehnte der zwölf Animationsprinzipien von *Disney* und bedeutet Übertreibung der Animationen. In Animationsfilmen wirken ganz realistische Bewegungen (z. B. rohe Motion Capture Aufnahmen) schwung- und energielos, weshalb kaum auffällige Übertreibung auch bei realistischen Animationen benötigt wird. Extreme Übertreibung kann zudem bewusst eingesetzt werden, wenn ein Cartoon-hafter Stil angestrebt wird. Auch die Einstellungsgröße muss beachtet werden. Wegen der Entfernung des Charakters zur Kamera, muss bei einer Totalen die Exaggeration deutlicher eingesetzt werden als bei einem näheren Bildausschnitt [2, S. 65].

Kapitel 3

Machinima

Machinima (Kombination aus den Wörtern „Machine“ und „Cinema“) sind Filme, die in einer Echtzeit 3D Umgebung erstellt wurden. Dies bezieht sich zumeist auf Videospiele bzw. Game Engines, da diese für die Ersteller von Machinima – die diese meist als Hobby produzieren – leicht zugänglich sind. Es gibt inzwischen aber auch eigene Software, in der im weitesten Sinne ebenfalls Machinima erstellt werden können. Eine der bekanntesten Machinima-Serien ist die in *Halo* [19] stattfindende Serie *Red vs. Blue* [52] von *Rooster Teeth* [47], die mithilfe von zwei XBoxen, um mehrere Charaktere gleichzeitig zu bewegen, aufgenommen wurde. Besonders viele Freiheiten bietet die Online Multiplayer Plattform *Second Life* [48], die den Spielern auch erlaubt eigene Assets zu erstellen und in das Spiel einzupflegen. Auch die Lebenssimulation *Sims* [25] ist durch die Möglichkeit Charaktere und Häuser selbst zu erstellen und eine im Spiel integrierte Screen-Capture Funktion, ein beliebtes Spiel bei Machinima-Produzenten.

Als der erste Machinima-Film gilt *Diary of a Camper*, der im Spiel *Doom* erstellt wurde. Genutzt wurde hierbei die im Spiel enthaltene Funktion zur Aufnahme von Gameplay Sequenzen in Form von Engine-eigenen Dateien, die ausschließlich mit dem Spiel selbst angesehen werden konnten. Die Dateien waren um ein Vielfaches kleiner als Videodateien, was den Austausch mit anderen Spielern erleichterte. Auch das nachfolgende Spiel *Quake* nutzte ein ähnliches Engine-eigenes Dateiformat. Zusätzlich war es in dem Spiel möglich, eigene 3D-Models und Kameras zu positionieren, wodurch noch viel mehr der damals „Quake Movies“ genannten Filme entstanden. Anfang 2000 gründete Hugh Hancock dann die Webseite *machinima.com* auf der Tutorials und Artikel, sowie neue Machinima veröffentlicht wurden. Diese Seite prägte dann auch den Begriff Machinima, um zu zeigen, dass nicht nur *Quake* als Spiel für diese Art von Filmen infrage kam. Die Videos wurden nun auch in allgemein gebräuchlichen Videodateiformaten veröffentlicht, sodass es nicht mehr nötig war, das entsprechende Spiel zu besitzen. Mit der steigenden Popularität entstanden auch Machinima, die im Fernsehen ausgestrahlt wurden. *MTV* produzierte beispielsweise *Video Mods*, eine Reihe von Videos, die unterschiedliche Spiele nutzte um bekannte Musikvideos „neu“ zu verfilmen u. a. mit dem Spiel *Die Sims*. Auch in der Emmy-preisgekrönten Folge *Make Love, Not Warcraft* der Serie *South Park* wurde Machinima aus dem Spiel *World of Warcraft* eingebaut.

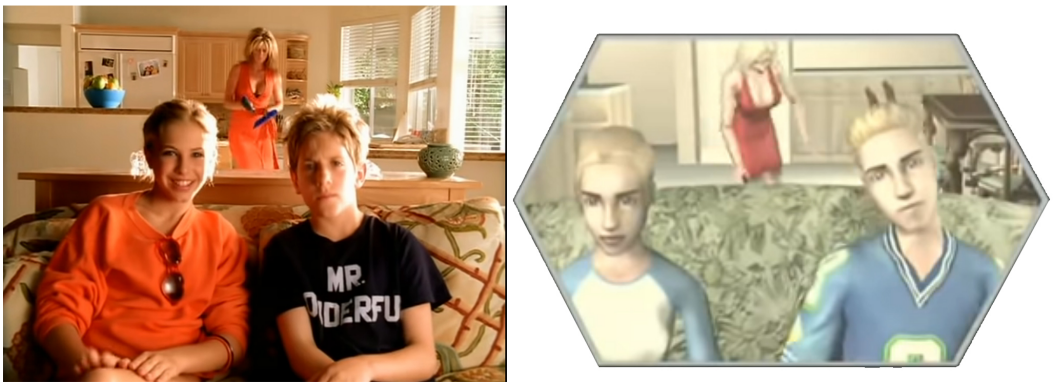


Abbildung 3.1: Vergleich des originalen Musikvideos zum Lied *Stacy's Mom* von *Fountains of Wayne* (links) neben der in *Die Sims* erstellten Version (rechts). Bildquelle [5].

3.1 Videospiel Machinima

Dieser Abschnitt bezieht sich auf Machinima, die in einem Videospiel oder einer Game Engine aufgenommen wurden. Es gibt mehrere Möglichkeiten, wie die Charaktere für den Film gesteuert werden können, die meist von Spiel zu Spiel variieren bzw. auch von den Fähigkeiten des Machinima-Produzenten abhängig sind. Die erste und für viele auch einfachste Möglichkeit besteht darin, die AI (Artificial Intelligence) des Spiels die Charaktere steuern zu lassen. Somit hat man aber nur wenig bis gar keine Kontrolle über die einzelnen Charaktere. *Die Sims* ist ein Beispiel für ein Spiel, das stark auf AI zurückgreift, wobei die Charaktere – Sim genannt – jedoch auch minimal über Interaktionen miteinander kontrolliert werden können. Dennoch benötigt es oft mehrere Versuche, bis der Sim auch genau die gewünscht Animation an der korrekten Stelle ausführt. Die zweite Möglichkeit ist das sogenannte „Digital Puppeteering“ (Digitales Puppenspiel), bei dem andere Spieler in die Rollen der Charaktere schlüpfen. Dafür werden also ein Spiel mit Multiplayer-Optionen (wie z. B. *Second Life* [48] oder *World of Warcraft* [22]) sowie weitere Personen, die als digitale Schauspieler fungieren, benötigt, was den Koordinations-Aufwand im Gegensatz zur ersten Methode meist deutlich erhöht. Außerdem kann es insbesondere bei Online Multiplayer-Spielen immer dazu kommen, dass andere nicht-beteiligte Spieler die Aufnahme stören. Die dritte Methode wird als „Recamming“ bezeichnet und wird nur von einigen wenigen Game Engines unterstützt. Hierbei wird die gesamte Handlung als solche aufgenommen und erst im Nachhinein werden Kameras, Lichter, etc. gesetzt und das Endergebnis dann als Video gerendert. Methode vier ist zu vergleichen mit Machinima, die in einer eigenen Software erstellt wurden (siehe Abschn. 3.2). Hierbei verlässt sich der Machinima-Produzent auf Skripte die Charaktere, Kameras, etc. steuern. Auch das ist nur in wenigen Game Engines für normale Spieler möglich. Inspiriert wurde diese Vorgehensweise von Spieleherstellern, die die Cutscenes ihrer Spiele mithilfe von Skripten erstellten.

3.2 Software Machinima

Durch die Popularität von Machinima entstanden auch Software-Lösungen, die explizit für die Produktion von Machinima-artigen Filmen gemacht wurden. Sie bieten eine Echtzeit 3D Umgebung, eine große Bibliothek an vorgefertigten 3D-Modellen und Animationen, sowie zumeist eine Timeline wie sie aus Videoschnitt-Programmen bekannt ist. Die Software *iClone*[1] ist über die Jahre immer weiterentwickelt worden und inzwischen nicht mehr nur ein Werkzeug für Filmemacher, sondern auch Spielehersteller, die auf die umfangreiche Animationsbibliothek und den Charakter-Ersteller zurückgreifen. Außerdem unterstützt die Software Echtzeit-Motion Capture u. a. über das iPhone X – bietet also weiterhin Werkzeuge an, die auch für Personen, die nicht über aufwendige technische Möglichkeiten verfügen leicht erhältlich und zu bedienen sind. Software wie *iClone* kann als Hybrid zwischen Animationssoftware wie *Autodesk Maya* und Game Engines bezeichnet werden, da es die Animationsmöglichkeiten des Ersten und die Echtzeitumgebung des Zweiten besitzt.

3.3 *Source Filmmaker*

Der *Source Filmmaker*[2] ist eine frei verfügbare Software von *Valve* [64] und basiert auf *Source Engine*, der Game Engine von *Valve*. Da der *Source Filmmaker* für die Erstellung von Teasern und Kurzfilmen zum Spiel *Team Fortress 2* [17] konzipiert wurde, werden mit dem Programm selbst auch zahlreiche Assets aus diesem Spiel mitgeliefert. Vereint werden die wichtigsten Funktionen zur Erstellung eines 3D-Animationsfilms: Animationswerkzeuge, Videoschnitt Editor, Audio Editor und Renderer. Nur die Erstellung der benötigten Assets ist nicht in der Software möglich. Um eigene Assets zu importieren, müssen diese im Format DMX gespeichert werden. Für die 3D-Software *Autodesk Maya* gibt es zu diesem Zweck ein von *Valve* bereitgestellten Export-Plugin. Eine weitere Möglichkeit zum Import von weiteren Assets, bietet der *Steam Workshop* [51] in den Spieler von *Team Fortress 2* eigenhändig erstellte Items, Animationen und Effekte hochladen und anderen zur Verfügung stellen [50].

Zum Erstellen von animierten Sequenzen bietet der *Source Filmmaker* mehrere Möglichkeiten: Vorerst kann wie in anderen 3D-Animations Programmen über Translations- und Rotationswerkzeuge im Viewport animiert und dies dann in der Timeline und im Graph Editor angepasst werden. Des Weiteren gibt es den Motion Editor, der in Kurven den Motion Path der ausgewählten Elemente anzeigt (siehe Abb. 3.2). Dieser Motion Path ist auch im Viewport zu sehen. Im Motion Editor wird hauptsächlich mit den vorhandenen Presets [53] gearbeitet, er eignet sich daher für allgemeine Korrekturen an einem Animation-Clip, während im Graph Editor die Animationskurven eines jeden Keyframes genau angepasst werden können [38]. Um dies zu vereinfachen, kann ein „Control Rig“ an den zu animierenden Charakter gebunden werden. Dies ist ein mitgeliefertes Skript, das auf alle neun Standard-Charaktere des *Source Filmmakers* angewandt werden kann und dem Standard-Rig u. a. IK-Kontrollelemente an Armen und Beinen hinzufügt. Das Skript kann dupliziert und für eigene Charaktere entspre-

[1] https://www.reallusion.com/de/iclone/
[2] https://store.steampowered.com/app/1840/Source_Filmmaker/

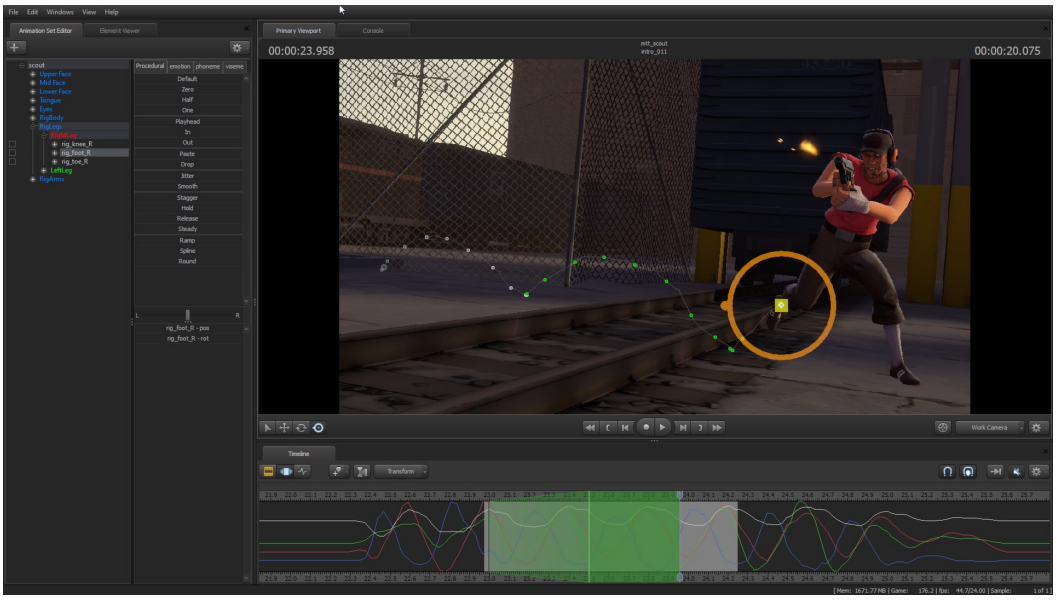


Abbildung 3.2: Das Interface des *Source Filmmakers*. Der Motion Path des ausgewählten Joints wird im Motion Editor und im Viewport angezeigt. Bildquelle [49].

chend angepasst werden. Besitzt ein Charakter, an den ein Control Rig gebunden wird, bereits Animationsdaten, so werden diese automatisch auf das Control Rig gebaked, sodass an der vorhandenen Animation nichts geändert wird. So wird ermöglicht, bereits vorhandene Animations Clips auch mit dem Control Rig zu bearbeiten [66].

Da der *Source Filmmaker* auf einer der Source Game Engine basiert, können auch einige Funktionen derselben zum Aufnehmen von Animationsdaten genutzt werden. Dazu wird in den Spielmodus gewechselt, in dem dann, wie in *Team Fortress 2*, ein beliebiger Charakter gesteuert. Über die Record-Funktion kann beinahe alles in der Spielwelt aufgenommen werden, beispielsweise neben den Charakteren auch die Kamera und Effekte wie Explosionen und Projektile. Die aufgenommenen Daten werden vorerst im „Floating Modification Layer“ zwischengespeichert. So können die aufgenommenen Takes gesichtet werden, bevor sie in den Shot gebaked werden. Für jeden Shot können mehrere Takes aufgenommen werden, die als solche erhalten bleiben und auch nach dem Baken einzeln aus dem Shot entfernt werden können [46].

Die dritte Möglichkeit Animation im *Source Filmmaker* zu erstellen ist „Puppeteering“. Hierbei wird ein Objekt bewegt, während die entsprechende Sequenz abgespielt wird. Die daraus entstehende Bewegung ist zumeist verwackelt und kann nicht in dieser Form verwendet werden, bietet jedoch eine gute Basis zur weiteren Verfeinerung der Animation [45]. In der Software werden einige Presets angeboten, die die Bearbeitung der Animationskurven vereinfachen bzw. automatisieren. Auf eine Puppeteering Aufnahme könnte z.B. das Smooth-Preset angewandt werden, dass die Animationskurven ausgleicht. Auch die Presets werden zuerst im Floating Modification Layer angewandt, sodass sie ohne Weiteres verworfen werden können, sollten sie nicht der Vorstellung des Animators entsprechen [53].

Kapitel 4

Animation in Videospielen

Animationen für Videospiele sind zumeist anders aufgebaut als Animationen für Filme. Da in einem Spiel nicht vorherzusehen ist, welche Animation wann – in einigen Fällen auch nicht wie lange – abgespielt werden muss, müssen die Animationen anders aufgebaut werden als die Shot für Shot geplanten Animationen eines Films. Die Animationen werden als einzelne Clips in das Spiel eingebaut und dann immer, wenn erforderlich abgespielt. Bei jedem Clip sollten Anfangs- und Endpose gleich sein, sodass diese nahtlos nacheinander oder in Schleifen abgespielt werden können. Welche Animation abgespielt wird, hängt von den Eingaben des Spielers oder im Spiel ausgelösten Ereignissen ab, die über Skripte oder in der Game Engine vorhandene State Machine (siehe Abschn. 5.2.1) Systeme mit der jeweiligen Animation verknüpft werden.

Um auf die unterschiedlichen Gegebenheiten, in denen die Animationen funktionieren müssen, reagieren zu können, werden manche Elemente in Videospielen simuliert. Ein Beispiel hierfür sind Haare die durch Shader z. B. auf Wind reagieren oder mittels in der Game Engine programmierter und berechneter Komponenten auf Bewegungen, Richtungswechsel und Drehungen des Charakters reagieren. Auch automatische Overlapping Action ist über eigene Skripte oder in der Game Engine vorhandene Tools (z. B. Animation Rigging in Unity) möglich.

Facial Animation, also die Animation von Gesichtern, wird entweder über ein aufwendiges Joint-basiertes Gesichts-Rig oder über Blend Shapes gesteuert. Da Blend Shapes allerdings dem Computer oder der Konsole hohe Rechenleistung abverlangen, werden diese nur selten oder in Kombination mit einem Joint-Rig angewandt. Auch Motion Capture eines Gesichts muss auf ein auf Joints oder Blend Shapes basierendes Rig übertragen werden, um in der Game Engine entsprechend dargestellt zu werden. Auch bei Facial Animation gilt das Prinzip der Exaggeration, Bewegungen sollten dementsprechend übertrieben bzw. von einem Animator überarbeitet werden. Vor allem bei fast realistischen Charakteren kann ansonsten ein Uncanny Valley Effekt[1] auftreten. Ein Beispiel für eine gute Überarbeitung von Motion Capture ist in Naughty Dog's [42] *Uncharted 4: A Thief's End* [23] zu sehen. Motion Capture Aufnahmen wurden als Basis genutzt und dann verstärkt, um die Emotionen der Charaktere noch deutlicher dar-

[1] Das Uncanny Valley bezeichnet einen paradox erscheinenden Effekt bei der Akzeptanz des Zusehers bezüglich künstlicher Figuren – wie z. B. 3D-Charaktere. Der Zuseher akzeptiert einen abstrahierten Charakter eher, als einen fast, aber nicht ganz realistischen [4, S. 33–35].

zustellen. In der Gegenüberstellung von rohen Motion Capture Daten und finalisierten Animationen – wie im Animation Breakdown Reel des Senior Animators Richard Pince [56] – ist der Unterschied klar zu erkennen.

4.1 Dialog Szenen in *The Witcher 3: Wild Hunt*

Cinematische Cutscenes sind in Videospielen ein beliebtes Mittel zum Erzählen der Geschichte. Vor allem bei größeren Spielen werden dafür allerdings mehrere Stunden an Videomaterial benötigt. Insgesamt waren es 37,5 Stunden im Spiel *The Witcher 3: Wild Hunt* [24] des Spieleentwicklers *CD Projekt Red* [29] [27, T=00:03:35]. Diese Szenen alle eigens in der angestrebten Qualität zu animieren, wäre ein Aufwand, für den die wenigsten Studios die nötigen Ressourcen aufbringen könnten. Daher wurden in *The Witcher 3* nur ca. 2,5 der 37,5 Stunden mithilfe von Motion Capture Verfahren neu animiert, für die restlichen 35 Stunden wurde der in die Studio-eigene Game Engine integrierte Dialog-Editor genutzt.

In den *The Witcher*-Spielen [24] sind die Entscheidungen des Spielers – vor allem im Dialog mit NPCs (Non-Playable Characters) – ausschlaggebend für den Ausgang der Geschichte. Da ein maßgeblicher Teil der Charakter-Entwicklung über Cutscenes geschieht, war es für die Entwickler wichtig eine effiziente Möglichkeit zur Produktion derselben mithilfe des Dialog-Editors zu finden. Dabei sollten die mit Spiel-Animationen erstellten Sequenzen auf den ersten Blick nicht von eigens animierten zu unterscheiden sein. Für *The Witcher 3* stand schon eine erste Version des Dialog-Editors aus dem Vorgängerspiel zur Verfügung, der allerdings nur einfache Schnitte ohne Kamerafahrten und die jeweiligen Idle- bzw. Sprachanimationen erlaubte. Dadurch entstand ein starker Qualitätsunterschied zu den aufwendigeren „normalen“ Cutscenes. Im überarbeiteten Dialog-Editor waren Kamerafahrten, unterschiedliche Charakter-Animationen sowie situationsbedingte Anpassungen derselben und Überblendung zwischen unterschiedlichen Animations-Clips. Die wichtigste Anpassung war die des Generators, der automatisch anhand des eingespeisten Dialogtextes und der Audiospur, sowie grundsätzlicher Informationen über die sprechenden Charaktere eine erste Version der Cutscene erstellt. Es wurden einige grundsätzliche Regeln wie die 180°-Regel[2] und das Konzept von Establishing Shots einprogrammiert. Diese erste Version ist noch sehr einfach gehalten und wirkt zumeist etwas plump, jedoch ist es für Game Designer die mehrere Stunden an Dialog-Szenen erstellen müssen einfacher vorhandene Kameraschnitte und Charakter-Animationen anzupassen, als jede einzelne Szene von Grund auf aufzubauen [27, T=00:13:37].

In einer Beispiel-Szene aus der Erweiterung *Heart of Stone* kommen die verschiedenen Möglichkeiten zur Anwendung. Verweise auf Zeitangaben im Folgenden, beziehen sich auf [16]. Eine der wichtigsten Ressourcen für die Erstellung der Dialog-Szenen bei *CD Projekt Red*, ist die bereits vorhandene Menge an Animationen die theoretisch auf alle Charaktere mit passendem Skelett bzw. Rig angewandt werden können und die Möglichkeit diese miteinander zu kombinieren. Beispielsweise ist es so möglich auf Hüfte und Beine eines Charakters eine Lauf-Animation anzuwenden, während auf Torso und Arme

[2] Die 180°-Regel besagt, dass in jedem Bild bzw. in jeder Szene eine virtuelle Handlungsachse – z. B. zwischen zwei Menschen – besteht, die in einem Schnitt nicht übersprungen werden darf. Geschieht doch ein Achsensprung, verliert der Zuschauer die Orientierung in der Szene [26].

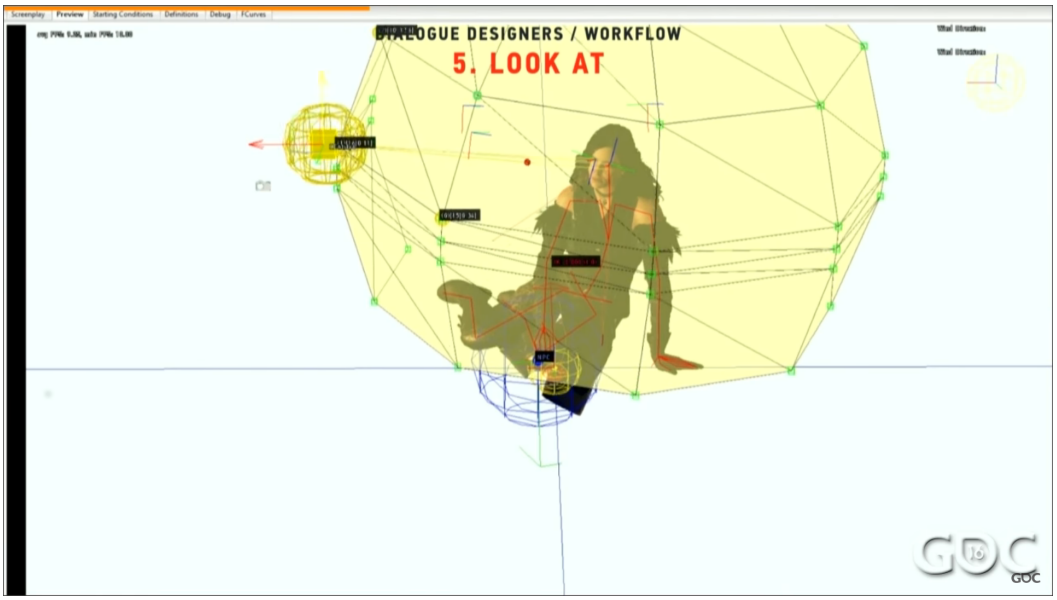


Abbildung 4.1: Die Vertices eines kugelförmigen Meshes rund um den Charakter, bestimmen die ausgeführte Look-At Pose. Befindet sich das Look-At Ziel zwischen zwei oder mehreren Vertices, wird zwischen den Posen interpoliert (im Bild zwischen drei verschiedenen Posen). Bildquelle [27].

eine winkende Animation ausführen. Diese Methode Animationen kombinieren zu können, ist bereits auch in anderen Game Engines eingebaut. In *Unity* beispielsweise, wird diese Funktion „Avatar Masking“ genannt. In der Beispielszene wurde dies möglicherweise in Shot drei und vier auf den Charakter Gaunter o'Dimm angewandt, sodass er in seiner eigenen Idle-Animation verweilen kann und gleichzeitig mit einem Arm ausladend gestikuliert. Des Weiteren ist es möglich die Länge der Animationen in der Timeline des Dialog-Editors zu skalieren, sodass diese verlangsamt bzw. schneller abgespielt werden. Dadurch können selbst durch eine generische Geste mit der Hand unterschiedliche Stimmungen des gesprochenen Dialogs unterstrichen werden. Auch das Schneiden der Clips, um nur einen Teil der Animation abzuspielen, ist möglich. Durch die automatisierte Interpolation zwischen aufeinanderfolgenden Clips bzw. zwischen Idle-Animation und Clip, werden abrupte Sprünge in der Bewegung der Charaktere vermieden. Beispielsweise wechselt Shani vor dem Tanz von einer Idle-Animation in einen Knicks (zu sehen bei [16, T=00:01:29]) [27, T=00:10:35].

Die Animationen können noch weiter über eine „Look-At“ Steuerung angepasst werden, diese bestimmt in welche Richtung der Charakter schauen soll. Normalerweise wird dies über eine Aim-Steuerung (ein Ziel wird gesetzt und die entsprechend daran gebundenen Teile des Rigs richten sich danach aus) gelöst, durch die Augen, Kopf und Torso automatisch je nach Position des Aims ausgerichtet werden. Da diese Option auch zu unnatürlichen Verdrehungen führen kann, vor allem wenn ein Großteil davon automatisiert geschieht, wurde in der Game Engine von *CD Projekt Red* eine andere über Posen gesteuerte Methode eingebaut. Für jede Idle-Animation werden mehrere Look-At Posen angelegt, in denen der Charakter in alle aus dieser Position möglichen Richtung sieht.

Die Posen werden außerdem nur als Kopfrotation, sowie als „Ganzkörper“-Rotationen angelegt. In der Game Engine wird um den Charakter ein trianguliertes, kugelförmiges Mesh angelegt, dessen Vertices für je eine Look-At Pose stehen zwischen denen je nach Position des Look-At Ziels interpoliert werden kann (siehe Abb. 4.1). Zu erkennen ist dies sehr gut bei [16, T=00:01:18], in dem Geralts Look-At Ziel von Gaunter o'Dimm zu Shani wechselt. Auch während der Tanzsequenzen sind die Look-At Ziele der beiden Protagonisten auf den jeweils anderen gerichtet [27, T=00:18:35].

Für noch mehr Kontrolle über bereits in die Game Engine importierte Animationen, ließ sich *CD Projekt Red* von *Valves* [64] *Source Filmmaker* (siehe Abschn. 3.3) inspirieren, in der ein Control Rig die nachträgliche Anpassung von Animationen ermöglicht. Dies wurde nur für kleine, situationsbedingte Änderungen an den bereits vorhandenen Animationen genutzt. Beispielsweise um die Animationen einem anderen Körperbau als dem des ursprünglich animierten Charakters anzupassen. Feine Details wie die Ausrichtung der Finger und zum Dialog passende Gesichtsausdrücke konnten Situationsspezifisch von Quest- und Dialogdesignern, in der Game Engine angepasst werden, sodass keine weiteren angepassten Animationsclips zu den über 2000 bereits existierenden Dialog-Animationen in das Spiel eingefügt werden mussten [27, T=00:22:09].

4.2 Cinematics in *World of Warcraft*

Im MMORPG (Massively Multiplayer Online Role-Playing Game) *World of Warcraft* [22] werden seit der zweiten Erweiterung *Wrath of the Lich King* [20] Cinematics im Spiel selbst eingesetzt. Bereits zuvor gab es Cinematics, um neue Erweiterungen anzukündigen, diese wurden allerdings mit hochauflösenden, realitätsnahen Models mithilfe klassischer Animations- und Render-Software erstellt, nutzten also in keiner Weise die Game Engine. Die erste filmische Cutscene in *Wrath of the Lich King* ist *The Wrathgate* [55] und wurde inspiriert von den bereits damals im Internet kursierenden *World of Warcraft* Machinima. Produzent dieser Cutscene war Terran Gregory, der durch sein eigenes Machinima *Return* eine Anstellung als Cinematic Producer bei *Blizzard* erhielt um Machinima-artige Cutscenes auch für das Spiel selbst zu produzieren.

Inzwischen hat sich die Qualität der Cutscenes gesteigert, einige weniger wichtige werden aber noch genau wie Machinima über das Abspielen der Spiel-Animationen erstellt. Dies geschieht in Echtzeit, da zumeist der Charakter des Spielers selbst in die Cutscene integriert ist, wie z. B. im Intro der Quest *Stormwind Extraction* [54, T=00:12:00] am Beginn der aktuellen Erweiterung *Battle for Azeroth* [18]. Hier wurde die Kamera manuell animiert und die Charaktere Sylvanas und Nathanos Blightcaller, sowie der des Spielers in einer separaten Instanz des Spiels (ohne Instanziierung würden NPCs und Charaktere anderer Spieler zu sehen sein) positioniert und mithilfe der Animationen aus dem Spiel animiert. Cutscenes dieser Art werden in *World of Warcraft* häufig genutzt, da sie schnell erstellt werden können und den Fokus des Spielers auf essenzielle Momente im Spiel lenkt, die ansonsten in den vielen Quest-Anweisungen und Dialogen, die nebenbei abgespielt werden, untergehen.

Auch hochauflösende, realistische Cinematics werden vermehrt genutzt, nicht nur um Erweiterungen anzukündigen, sondern auch um besonders „epische“ Momente darzustellen. Diese Momente sind für die Geschichte so wichtig, dass sie bereits Monate vor Erscheinen der Erweiterung bzw. des jeweiligen Cinematics feststehen und somit be-

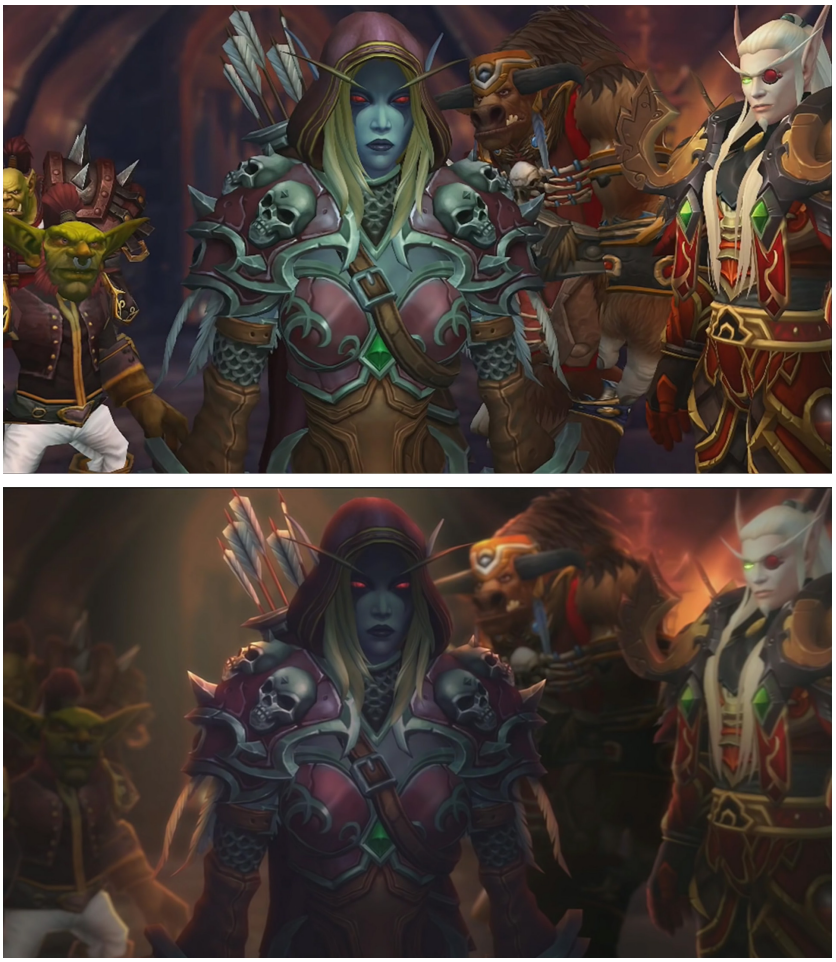


Abbildung 4.2: Sylvanas tritt an Warchief Vol'Jin heran. Oben: Assets des Spiels bereits mit neuen Animationen. Unten: Lichtsetzung und Post Processing angepasst. Bildquelle [28].

sonders früh mit der Arbeit an diesen aufwendigen Kurzfilmen begonnen werden kann. Diese Filme zeigen des Weiteren oft Szenen, die der Spieler als Charakter in dieser Welt nicht miterleben würde, ein weiterer Grund warum sie sich besonders gut für eine Veröffentlichung außerhalb des Spiels – sie werden zumeist auf Plattformen wie *Youtube* veröffentlicht – eignen.

Als Zwischenstufe der Machinimas und realistischen Cinematics wurden Cutscenes etabliert, die zwar die Models des Spiels nutzen, nicht jedoch die rohen Spiel-Animationen. Für Cutscenes dieser Art wird eine eigene Version der Game Engine genutzt, die Werkzeuge zur Erstellung von Cinematics besitzt und auf den aktuellen Stand des Spiels zurückgreifen kann [28]. Mit der Planung der Cinematics kann somit auch schon in der Anfangsphase der Erweiterungs-Entwicklung begonnen werden. Oft wirken sich die durch Ausarbeitung des Cinematics entstandenen Ideen auch auf das Umgebungsdesign im Spiel selbst aus (z. B. ist nach dem Tod von Ysera der Handlungsort mit Pflanzen überwachsen, was für den Spieler auch nach Ende des Cinematics zu

sehen ist [28, T=00:12:00]). Auch höher aufgelöste und aufwendigere Charakter Models landen nach einer Überarbeitung durch das Cinematics-Team als diese überarbeiteten Versionen im Spiel. Ein Beispiel hierfür ist der Charakter „Genn Greymane“ der bereits in seiner menschlichen Form im Spiel vorhanden war, jedoch nicht in seiner Worgen-Form (Worgen sind Kreaturen in *World of Warcraft* die sich bewusst in eine Art Werwolf verwandeln können). Da er als solcher jedoch für Szenen inmitten eines Kampfes benötigt wurde, entstanden dann insgesamt vier unterschiedliche Versionen von ihm, die auch im Spiel eingesetzt werden konnten [28, T=00:30:30]. Dadurch dass diese Cinematics als pre-gerenderte Sequenzen im finalen Spiel landen, haben auch Lighting- und VFX-Artists mehr Möglichkeiten, ohne auf Limitationen der Endgeräte der Spieler achten zu müssen. Vor allem durch geplante Lichtsetzung und Post Processing erhalten diese Cinematics ein viel hochwertigeres Aussehen als das Spiel selbst, obwohl Models und Texturen dieselben sind (siehe Abb. 4.2). Durch die Nutzung einer Echtzeit-Render-Umgebung und der Spiel-Assets ist das Cinematics-Team auch flexibel genug, um schnell auf Änderungen in der Handlung aufgrund von angepasstem Questdesign oder Setting zu reagieren und die Cinematics dementsprechend anzupassen [28].

4.3 *Meet the Team*: Teaser für *Team Fortress 2*

Die Kurzfilm-Reihe *Meet the Team* [39] wurde mithilfe des *Source Filmmakers* (siehe Abschn. 3.3) produziert und stellt die neun Klassen des Spiels *Team Fortress 2* vor. Dabei liegt der Fokus auf dem Charakter der jeweiligen Person, die diese Klasse verkörpert, statt nur auf die Fähigkeiten derselben im Spiel. Die ersten sechs Kurzfilme erschienen bereits in den Jahren 2007 und 2008 kurz vor und nach dem Erscheinen des Spiels. Diese ersten Kurzfilme sind Interviews mit den jeweiligen Charakteren, sowie humorvolle Ausschnitte aus ihrem Leben als Söldner. Die Idee dazu entstand aus den Casting-Skripts für die Sprecher der Charaktere, die auch teilweise übernommen wurden. Für die finalen drei Kurzfilme zu Spy, Medic und Pyro die erst später erschienen sind, wurde dieser Ansatz jedoch verworfen und stattdessen eine jeweils zum Charakter passende Geschichte erzählt. In *Meet the Medic* [8] beispielsweise, kann der Zuseher verfolgen wie der Medic seine Spezial-Fähigkeit „Übercharge“ entwickelt. Durch die Art und Weise wie er seinen Patienten Heavy behandelt, wird außerdem gezeigt, dass er keineswegs ein normaler, vertrauenswürdiger Arzt oder Heiler ist [41]. Im Folgenden wird diese Episode genauer betrachtet und auf Basis der offiziellen *Source Filmmaker*-Tutorials [65] analysiert, wie die Charaktere in diesem Kurzfilm vermutlich animiert wurden. Zur genaueren Analyse wurde ein Sequenzprotokoll erstellt (siehe Anhang A).

Die Charaktere des roten Teams sind in *Meet the Medic* zu einem großen Teil Keyframe animiert. Um im *Source Filmmaker* effektiv animieren zu können, werden die Charaktere mit einem Control Rig versehen. Dieses belegt Arme und Beine mit Inverser Kinematik und reduziert die im Editorfenster sichtbaren Kontrollelemente, darunter u. a. die oft benötigten Joints für Kleidungsstücke und andere zugehörige Gegenstände. Im Animation Set Editor können auch alle versteckten Elemente angezeigt werden und bei Bedarf auf sichtbar gestellt werden. Um die Charaktere zu positionieren und animieren, können das Editorfenster, der Animation Set Editor und der Graph Editor genutzt werden. Um im Editorfenster die Kontrollelemente zu sehen, muss der Graph Editor aktiv sein und ein Hotkey (in der Standardeinstellung auf „Strg“) gedrückt werden. Das

ist für den Animator sehr angenehm, da der Charakter nicht von zahlreichen Elementen überdeckt wird. Für die Gesichtsanimation sind keine Elemente im Editorfenster vorhanden. Diese kann über den Animation Set Editor erstellt und im Graph Editor bearbeitet werden.

Sollen Animationen, die in einem anderen Programm erstellt wurden, importiert werden, so müssen diese im Format DMX abgespeichert werden und zum jeweiligen Rig passen. Animationen aus dem Spiel *Team Fortress 2* sind hingegen bereits im *Source Filmmaker* vorhanden und werden als „Sequenzen“ bezeichnet. Diese Sequenzen sind auf die neun Charaktere aus dem Spiel angepasst und können bereits im Import-Fenster geringfügig angepasst (z. B. Schrittgeschwindigkeit bei Walk Cycle) und miteinander kombiniert werden. Sequenzen können nicht für einen Charakter mit aktivem Control Rig importiert werden, weshalb dieses zuerst im Animation Set Editor entfernt werden muss. Bereits erstellte Animationen werden dabei auf das FK-Rig (Forward Kinematik – der Parent-Joint bewegt alle Child-Joints im jeweiligen räumlichen Verhältnis mit) gebaked, sodass diese nicht verloren gehen. Die Bearbeitung von Sequenzen geschieht zudem im Motion- statt im Graph Editor, wo u. a. die Dauer und Übergänge zwischen Sequenzen justiert werden können. Diese Methode wurde vermutlich bei den blauen Gegnern angewandt. Die feindliche Horde besteht aus zahlreichen Soldaten mit unterschiedlichen Kopfbedeckungen, jedoch gleicher Walk Cycle Animation, wie bei [8, T=00:02:56] gut erkennbar. Nach Aktivierung des Übercharges ab [8, T=00:03:26], fallen die Soldaten nacheinander zu Boden. Dabei wurden zwei unterschiedliche Sequenzen genutzt, um eine geringfügige Varianz einzubringen. In älteren *Meet the Team*-Episoden wurden Animationen aus dem Spiel öfter genutzt, vor allem für die Nebencharaktere im Hintergrund. In der letzten Szene der Episode *Meet the Heavy* [7] sind Charaktere des roten und blauen Teams zu sehen, die eindeutig die Animationen des Spiels ausführen. Vergleicht man die Episode *Meet the Scout* [9] mit der offiziellen *Source Filmmaker* Tutorial-Reihe, kann angenommen werden, dass die im Tutorial vorgestellte Methode Inhalte aus der Spielansicht aufzunehmen, am Anfang des Videos angewandt wurde. Hierbei läuft der Scout durch gegnerisches Gebiet und überspringt große Hindernisse mit einem Doppelsprung. Die aufgenommenen Animationen wurden anschließend noch bearbeitet, wie man anhand der Schulterblicke, die er auf die Gegner wirft, erkennen kann.

Die Animation der Gesichter kann wie erwähnt nur über Animation Set Editor und Graph Editor bearbeitet werden. Für die Synchronisation von Dialogszenen kann zusätzlich die Funktion „Extract Phonemes“ (engl.: Phoneme extrahieren) genutzt werden, die mit allen Rigs aus *Team Fortress 2* funktioniert. Hierbei wird auf Basis der Waveform der aktiven Audiospur Lippenanimation generiert. Anschließend kann der Animator diese generierte Animation bearbeiten und das restliche Gesicht entsprechend animieren.

Für *Meet the Medic* wurden des Weiteren zusätzliche Models und Rigs angefertigt, um den speziellen Anforderungen gerecht zu werden. Die einfachste Änderung wurde an den blauen Soldaten vorgenommen, die unterschiedliche Kopfbedeckungen tragen. Der rote Scout und Demoman mussten für den Anfang der Episode verletzt sein. Da ab [8, T=00:02:40] beide geheilt werden, wurde hierbei vermutlich auf Blend Shapes für Scouts Gesicht und separate Objekte für Demomans Verbände zurückgegriffen. Für den Protagonisten Medic werden zumindest zwei verschiedene Charakter-Models genutzt, wobei das ab [8, T=00:02:16] genutzte Model mit Arztkittel möglicherweise eine

Abbildung 4.3: Die unterschiedlichen Varianten von Medics Kleidung im Kurzfilm *Meet the Medic*. In Bild 1 und 2 (v.l.n.r.) wurde vermutlich dasselbe Model mit veränderter Textur genutzt. Für das Model in Bild 3 wurde möglicherweise das Model aus dem Spiel *Team Fortress 2* genutzt. Bildquelle [8].

höher aufgelöste Version des Spiel-Models ist. Im ersten Teil des Films trägt Medic keinen Kittel und ist an Händen und Kleidung zu unterschiedlichen Graden mit Blut beschmiert (siehe Abb. 4.3). Am stärksten angepasst wurde das im Operationssaal genutzte Model bzw. Rig von Heavy, der, wie bei [8, T=00:00:25] sehr gut zu sehen ist, mit offenem Abdomen auf dem Behandlungstisch liegt. Auch sind mehrere Eintrittswunden von Schüssen sichtbar, die das normale Spielmodel nicht besitzt. Da sich die Organe auch unabhängig vom restlichen Körper bewegen (z. B. „atmen“ sich die Lungenflügel), sind diese allem Anschein nach zusätzlich mit Joints und Blend Shapes geriggt worden. Auch das rapide Verheilen von Heavys Körper bei [8, T=00:01:53] wurde vermutlich über aufwendige Blend Shapes animiert.

Kapitel 5

Charakter-Animation in *Unity*

Animationen für Charaktere sind in *Unity* vorrangig auf die Nutzung in einer interaktiven Umgebung wie der eines Computerspiels ausgerichtet. Es sind zumeist relativ kurze, sehr spezifische Bewegungsabläufe die auf Eingabe des Spielers hin oder als Reaktion auf ein Event im Spiel selbst abgespielt werden. So können zwar kurze aber dennoch sehr viele Animationen für einen einzelnen Charakter benötigt werden, um auf die verschiedenen Begebenheiten der interaktiven Umgebung zu reagieren. Diese Animationen werden wiederum unterschiedlich eingesetzt: Manche werden in Schleifen abgespielt (z.B. Idle-Animationen, die abgespielt werden, wenn der Charakter keine Eingaben vom Spieler erhält), andere werden nur einmal abgespielt (z.B. das Werfen eines Gegenstandes). Animationen können auch nur bestimmte Teile des Rigs betreffen (z.B. Winken mit einem Arm) wodurch verschiedene Animationen miteinander kombiniert werden können (z.B. Winken beim Gehen) [3].

5.1 Rigs

Die Konfiguration des Rigs bietet die Basis für alle Animations-Lösungen in *Unity*, sollte also vor allem bei stilisierten oder unproportionalen Charakteren entsprechend konfiguriert werden. Zumeist sind eingebaute Funktionen und Plugins von Drittanbietern auf humanoide Rigs ausgelegt und können für diese mehr Funktionen (z. B. Foot-Roll und IK) bieten.

Rigs (siehe Abschn. 2.1) werden in *Unity* als eine hierarchisch angeordnete Gruppe von bis auf Transformationswerten leeren Objekten dargestellt. Ein Rig kann nicht in der Game Engine erstellt werden, sondern muss zuerst in einer 3D Software aufgebaut werden. Ein geriggter Charakter kann einfach in das *Unity* Projekt importiert werden. In den Import Einstellungen im Tab Rig kann nun ein Animations-Typ festgelegt werden [59]. Im Falle eines menschlichen Charakters (Kopf, zwei Arme, zwei Beine) ist es ein „Humanoides“ Rig, das automatisch als solches erkannt wird. Für humanoide Rigs gibt es die Möglichkeit dieses zu konfigurieren. Im Tab „Avatar Mapping“ werden die einzelnen Joints zugewiesen. Wurden die Joints bereits klar benannt, erfolgt die Zuweisung automatisch. Hier ist auch zu sehen welche Joints für ein humanoides Rig vorhanden sein müssen, alle zusätzlichen Joints können den entsprechenden Bereichen zugewiesen werden. Im Tab „Muscles & Settings“ kann das Rig auf korrekte Rotationen überprüft und

die maximalen Rotationswerte der einzelnen Muscle-Gruppen festgelegt werden. Weitere Einstellungsmöglichkeiten nehmen Einfluss auf Twist- und Stretch-Faktoren des Rigs. Entspricht das Rig nicht den Vorgaben eines humanoiden Charakters, wird der Animation Typ „Generic“ festgelegt (z.B. vierbeinige Tiere). Rigs dieses Typs können nicht wie humanoide Rigs genauer konfiguriert werden.

Sind in einer importierten FBX-Datei auch Blend Shapes enthalten, so werden diese in *Unity* bei Auswahl des Objekts in der Komponente „Skinned Mesh Renderer“ angezeigt. Wie stark eine Blend Shape das Mesh beeinflusst kann über einen Regler gesteuert werden, der auch im Animation Window (siehe Abschn. 5.2.3) animiert werden kann.

5.2 Animationen

Mehrere Animationen für einen Charakter bzw. ein spezifisches Rig können gemeinsam in einer einzigen FBX-Datei importiert werden. Sie können auch auf mehrere Dateien verteilt sein und müssen mindestens ein Rig und eine Animation (minimal ein gesetzter Key) enthalten. Ein auf das Rig geskinntes Mesh muss in der Animations-Datei nicht vorhanden sein. So können Animationen auf mehrere verschiedene Meshes bzw. Charaktere angewandt werden, einzige Voraussetzung ist, dass das Rig der Charaktere und der Animationen denselben Aufbau und dieselbe Benennung hat.

In der Vorschau der Datei können Zeitspannen innerhalb der gesamten Animation in sogenannte „Animation Clips“ aufgeteilt werden. In einer 3D Software wie Autodesk Maya können auch beim Export bereits die einzelnen Clips festgelegt werden. Jeder Animation Clip enthält eine Animation die einzeln als solche verwendet werden kann. Dies kann beispielsweise eine Idle-Animation sein, die genutzt wird, wenn der Charakter nichts tut, oder ein Walk-Cycle, die Animation eines gehenden oder laufenden Charakters. Diese Animationen sind zumeist so erstellt worden, dass die Anfangsposition der Endposition gleicht, sodass diese in Schleifen abgespielt werden können. Die Animationskurven der Clips sind in *Unity* über das „Animation Window“ einsichtbar, jedoch nicht bearbeitbar. Für die Filmproduktion würde es sich anbieten jeweils einen Shot als Animation Clip zu speichern.

5.2.1 Mecanim

In einem Computerspiel werden die Animationen über „State Machines“ [60] gesteuert. In einer State Machine wird festgelegt welche Animation ein Charakter zu welchem Zeitpunkt ausführen soll, sodass immer eine Animation abgespielt wird. Des Weiteren wird festgelegt welche Animation auf welche folgen kann (z.B. kann in der in Abb. 5.1 gezeigten State Machine ein Charakter nur springen, wenn er vorwärts geht, aber nicht, wenn er rückwärtsgeht). Damit die verschiedenen Clips beim Wechsel nicht abrupt enden und anfangen, werden sie miteinander verblendet. Bei Animationen deren Bewegungsmuster sich stark unterscheidet, kann eine „Transition“ direkt in der State Machine eingestellt werden. Diese fällt zumeist relativ kurz aus und dient rein dazu sprunghafte Änderungen zu vermeiden. Wenn die Bewegungsmuster sich ähnlich sind und somit eine möglichst gleichmäßige Verblendung passieren soll (z.B. der Übergang von gehen zu sprinten), werden „Blend Trees“ [58] in die State Machine eingebaut, die eine genauere Steuerung ermöglichen.

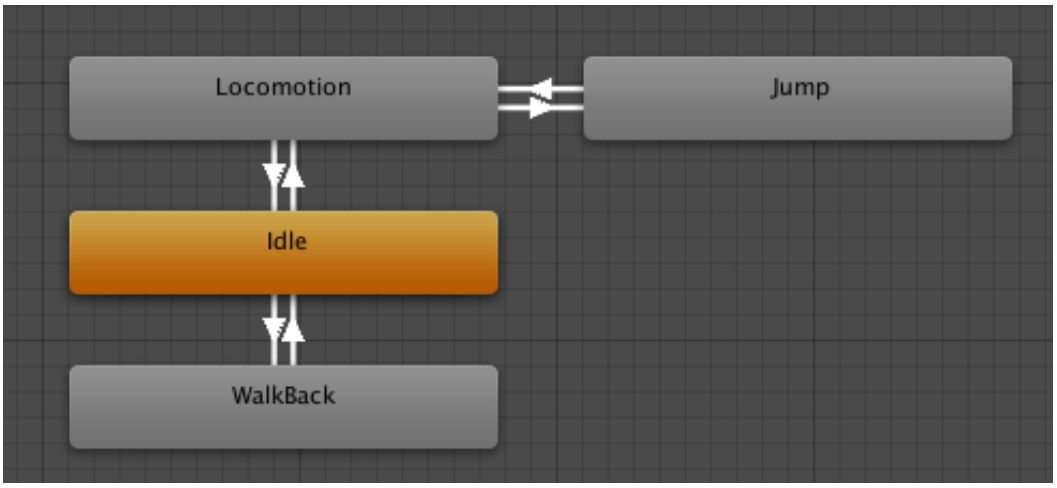


Abbildung 5.1: Eine einfache State Machine in *Unity*. Der Charakter verweilt in der Idle-Animation (orange) solange er keine anderen Befehle erhält. Über eine Eingabe des Spielers bewegt er sich vorwärts (Locomotion) oder rückwärts (WalkBack). Solange er sich im Vorwärts-State befindet, kann er auch einen Sprung ausführen (Jump). Bildquelle [60].

Mecanim ist die in *Unity* eingebaute State Machine. Um Mecanim für einen Charakter zu nutzen, muss eine Animator Controller Komponente erstellt werden. Im Fenster Animator ist dann die neu erstellte State Machine zu sehen, die bereits zwei States enthält: Entry und Any State. Entry bezeichnet den Start des Spiels und Any State ist ein spezieller State der genutzt wird, wenn ein anderer State ausgeführt werden soll, egal welcher zurzeit eigentlich aktiv ist. Es können nun weitere States erstellt werden, denen Animation Clips zugewiesen werden. Normalerweise wird z. B. ein Idle-State wird erstellt und direkt mit dem Entry-State verbunden, sodass der Charakter sich am Anfang des Spiels in einer Idle-Animation befindet. Um festzulegen, wann zu einem anderen State gewechselt wird (z. B., wenn der Charakter losgeht), müssen noch Transitions und Parameter festgelegt werden. Transitions sind einfache Verbindungen zwischen den States, die einfach im Animator Fenster gezogen werden. Parameter legen fest unter welchen Bedingungen gewechselt werden soll. Sie können Integer, Float und Boolean Werte sein oder auf Trigger reagieren. Mittels eines Scripts wird festgelegt welchen Wert der Parameter besitzt, z. B. ob und wie schnell sich ein Charakter vorwärtsbewegt und somit den Gehen- oder Laufen-State aktiviert.

5.2.2 Timeline

Während die State Machine (siehe Abschn. 5.2.1) auf interaktive Eingaben und Events reagiert, ist die „Timeline“ [61] in *Unity* das Werkzeug zur Erstellung linearer Sequenzen und Animationen. In der Entwicklung von Computerspielen kann sie für Cinematics inmitten des Spielverlaufs, die zumeist die Erzählung vorantreiben genutzt werden, aber auch für kleine Sequenzen im Spiel, wie beispielsweise einer Kamerafahrt, die dem Spieler einen Überblick über ein Level verschafft. Auch für komplexere Partikel-Effekte oder Audio Sequenzen kann die Timeline genutzt werden. Analog zu Compositing-Programmen,

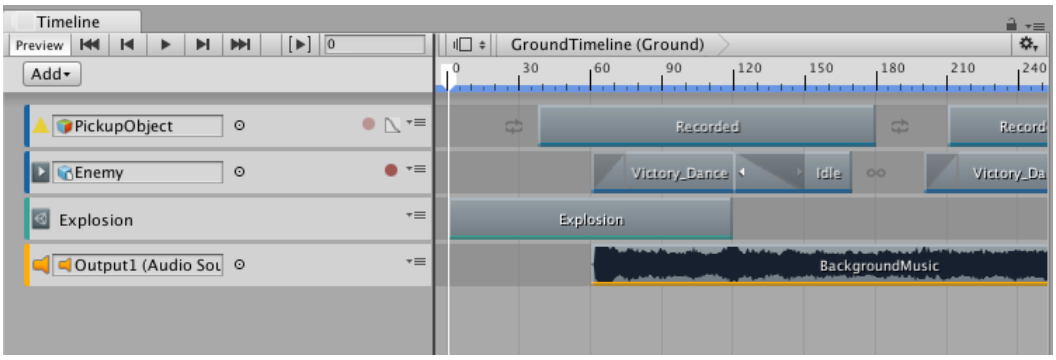


Abbildung 5.2: In dieser Timeline werden ein Partikel-Effekt (Explosion), Animation Clips (VictoryDance, Idle und Recorded) und Hintergrundmusik zusammengesetzt. Bildquelle [61].

wie z.B. Adobe After Effects[1] sind die verschiedenen für die Sequenz relevanten Objekte auf Ebenen – sogenannten „Tracks“ – verteilt. Es gibt fünf verschiedene Arten von Tracks, für Kurzfilme werden Animations-, Control- und Audio-Tracks am häufigsten in Verwendung sein [62].

Eine Timeline wird als eine eigene Komponente an ein Objekt in der *Unity*-Szene gebunden, somit können auch mehrere Timelines in einer Szene existieren. Da eine Timeline in Form einer Datei auch im Projekt-Ordner abgelegt wird, kann dieselbe Timeline auf mehrere Objekte in der Szene angewandt werden, so kann beispielsweise eine zu einem bestimmten Zeitpunkt ausgelöste Charakter-Animation, die zusätzlich von einem Partikel-Effekt und Sound-Effekten begleitet wird, auf mehrere Charaktere in der Szene gelegt werden. Ausgelöst wird eine Timeline durch gescriptete Events oder beim Start des Spiels bzw. des Play-Modus im *Unity*-Editor. Für Filme ist letzteres von Belang, da beim Starten des Play-Modus der in der Timeline zusammengestellte Film abgespielt werden soll [61].

Für Filme und Cinematics bietet es sich an, je eine Timeline pro Shot anzulegen und diese dann in eine weitere Timeline mittels „Control Tracks“ – einzubetten. Anpassungen an einzelnen Shots können so gezielter vorgenommen werden. Nachdem die Timelines nicht von anderen Objekten in der Szene abhängig sind, sollten sie leeren Objekten zugewiesen werden. Diese können dann dem Shot entsprechend benannt werden, was wiederum die Übersichtlichkeit in der Hierarchie der *Unity*-Szene verbessert.

Charakter-Animationen werden über Animation-Tracks abgespielt. Ein Rechtsklick auf die Zeitleiste des jeweiligen Tracks öffnet ein Optionsfenster zum Hinzufügen von Animation Clips, die zum Rig des zu animierenden Charakters passen. Soll der Clip über eine längere Zeitspanne als seiner eigentlichen Länge abgespielt werden, kann er wie z.B. auch in *Adobe After Effects* länger gezogen werden. Dies wird vor allem bei Clips genutzt, die in Schleifen abgespielt werden können. Aufeinanderfolgende Clips können außerdem miteinander verblendet werden, wodurch ähnlich wie bei einer Transition in der State Machine die beiden Animationen ineinander übergehen, um abrupte Wechsel der Körperhaltung zu vermeiden. Charaktere bzw. Objekte in der Szene kön-

[1] https://www.adobe.com/at/products/aftereffects.html

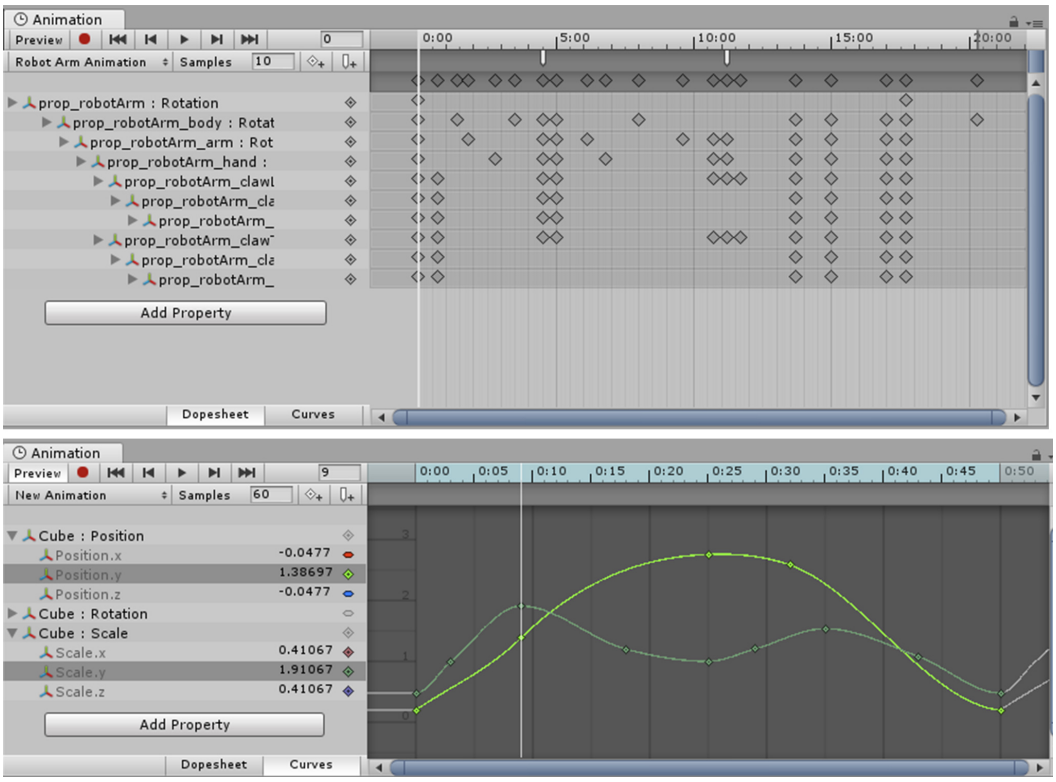


Abbildung 5.3: Oben: In der Dopesheet Ansicht werden die einzelnen Keys der verschiedenen Parameter angezeigt. Wird der Parameter ausgeklappt, ist der aktuelle Wert je nach Zeitpunkt in der Zeitleiste angezeigt [57]. Unten: In der Curves Ansicht wird über Kurven angezeigt wie sich die Werte von einem Key zum Nächsten verändern. Bildquelle [63].

nen auch direkt in der Timeline animiert werden. Dies ist aber nicht dazu konzipiert, komplexe Charakter-Animationen zu erstellen. Vielmehr können beinahe alle Parameter in den verschiedenen Komponenten eines Objekts animiert werden. Dazu gehören die Translations-, Rotations- und Skalierungswerte, aber auch Objektspezifische Eigenschaften wie die Farbe oder Intensität einer Lichtquelle. Für Charaktere sind zumeist ausschließlich die Translations- und Rotationswerte von Belang, da die einzelnen Joints eines Rigs nicht in der Timeline animiert werden sollen. Ist der Charakter bereits über einen Animations Clip animiert, können über einen Override Track – eine Unterebene des eigentlichen Animations Tracks – weitere Werte animiert werden. Dies kann zum Ausgleich von kleinen Translationsabweichungen zwischen verschiedenen Clips dienen oder zur eigentlichen Translation, wenn die Animationen der Clips am Stand animiert sind, eine Praktik bei Computerspiel-Animationen die es erlaubt die tatsächliche Geschwindigkeit des Charakters gezielt zu steuern [61].

5.2.3 Animation Window

Die Animationsdaten von aus anderer 3D-Software importierten Animations Clips, können im Animation Window eingesehen werden. Es kann immer nur die Animation eines

Objekts und dessen Child-Objekte angezeigt werden, weshalb sich das Animation Window lediglich zur Einsicht und Bearbeitung einzelner Objekte eignet. Zur Kombination mehrerer animierter Objekte sollte die Timeline (siehe Abschn. 5.2.2) genutzt werden. Es gibt zwei verschiedene Ansichten: „Dopesheet“ und „Curves“. Dopesheet zeigt jedes animierte Child-Objekt und seine Keys hierarchisch geordnet an (siehe Abb.5.3 oben). Wird das Child-Objekt ausgeklappt, sind die einzelnen animierten Parameter und deren jeweilige Keys sichtbar (z.B. die Translationswerte für x,y und z). Das Dopesheet bietet somit eine Übersicht aller Objekte und erlaubt über Eingabefelder neben den Parametern genaue Werte anzugeben [57]. Um den Übergang zwischen einzelnen Keys zu steuern, nutzt man die Curves-Ansicht (siehe Abb. 5.3 unten). Die Kurven zeigen an wie sich der animierte Wert verändert. Nähert sich der erste Wert gleichmäßig dem Wert des nächsten Keys, so wäre die Kurve eine gerade Linie, die beide Keys verbindet. Zumeist sind die Kurven an den Keys jedoch abgeflacht, um einen natürlicheren Übergang zu erzielen. Die Kurven können über die Tangenten an jedem Key gesteuert werden [63].

Wie in Abschn. 5.2 bereits erwähnt sind importierte Animationen im Animation Window nicht editierbar. So bleiben die Daten einer importierten Datei immer erhalten. Sollte man eine importierte Animation in *Unity* bearbeiten wollen, muss im Projekt-Fenster ein neuer, leerer Animation Clip erstellt werden. Anschließend kopiert man die Keys aus der importierten Animation in den neuen Animation Clip, in dem die Keys nun bearbeitet werden können. Will man den editierten Animation Clip nutzen, kann er wie andere importierte Clips auch, in der Timeline oder in der State Machine für einen Charakter mit passendem Rig eingesetzt werden.

Wird ein neuer, leerer Animation Clip erstellt, können Animationen auch in *Unity* neu erstellt werden. Dies wird allerdings nur für kleinere Animationen empfohlen, wie beispielsweise das Öffnen und Schließen einer Tür oder das Schwingen eines Pendels. Für aufwendige Charakter-Animation eignet sich das Animation Window nicht, nicht zuletzt, weil es einige Elemente eines zuvor erstellten Rigs (wie z.B. IK Handles) nicht korrekt animieren kann. Auf Animation ausgerichtete 3D Software bietet eine um einiges effizientere Darstellung und Handhabung. Mit „Animation Rigging“ baut *Unity* nun allerdings die Möglichkeiten der effizienten Nutzung von Character Rigs in der Engine aus (siehe Abschn. 5.2.4). Dieses Plugin befindet sich derzeit allerdings noch in Entwicklung.

5.2.4 Inverse Kinematic

Wird in der 3D Software Inverse Kinematik (IK) in das Rig eingebaut und beim Animieren genutzt, wird *Unity* diese beim Importieren des Rigs nicht als solche erkennen. Auch wenn das IK-Kontroll-Objekt in der Hierarchie als weiteres Transform-Objekt angezeigt wird, verliert es beim Import seine Verbindung zum Rig und kann die Joints und somit das Mesh nicht beeinflussen. Wird in der 3D Software mit IK animiert, kann die Engine diese Animation dennoch interpretieren, da die Animation beim Export gebaked wird.

5.3 Animation Rigging

Das von *Unity* bereitgestellte Package Animation Rigging ermöglicht die Darstellung eines Joint-Rigs in der Game Engine sowie das Erstellen von Kontroll-Objekten, die ähnlich wie Constraints in 3D-Programmen, die Beeinflussung der Animation ermöglichen. In *Unity* ist dies derzeit allerdings nur zur Laufzeit möglich, um also die Auswirkungen der Kontroll-Objekte zu sehen, muss zumindest in den Play-Modus des Editors gewechselt werden. Des Weiteren ist das Animation Rigging eher auf die vereinfachte Animation und Interaktion von Gegenständen in Spielen gedacht. Sollen sich die Kontroll-Objekte auch auf in der Timeline (siehe Abschn. 5.2.2) statt im Animator (siehe Abschn. 5.2.1) instanziierte Animationen auswirken, muss derzeit ein programmierter Workaround eingesetzt werden. Dazu werden in einem Skript zwei Funktionen angelegt, die das Animation Rig aktivieren bzw. deaktivieren. Vor der ersten Animation mit Kontroll-Objekten werden per Timeline-Signale zuerst die Funktion zur Deaktivierung und dann zur Aktivierung des Animation Rigs aufgerufen. Dieses Problem sollte zur endgültigen Veröffentlichung des Packages – derzeit befindet es sich noch in der Beta – behoben werden.

Die bereits im Standard-Package vorhandenen Kontroll-Objekte lassen sich dennoch schon vielseitig einsetzen. Unter anderem ist es möglich Two-Bone-IK einzusetzen, was z. B. die genauere Positionierung der Füße auf dem Untergrund ermöglicht. Dies könnte genutzt werden um Walk-Cycles, die nicht auf unebenes Terrain angepasst sind, erst in der Engine dementsprechend anzupassen. Auch simple Overlapping Action könnte mithilfe des Damp-Effektors automatisch generiert werden. Zu den weiteren vorhandenen Kontroll-Möglichkeiten zählen u. a. Twist-Korrektur z. B. für Extremitäten, Spline-IK und Aim-Effektoren, die über ein Ziel-Objekt die Rotation des daran gebundenen Objekts kontrollieren. Mit den entsprechenden Kenntnissen in C# ist es auch möglich eigene Kontroll-Objekte zu scripten.

Allerdings ist das Animation Rigging Plugin nicht für das Erstellen von Keyframe-Animationen geeignet. Die Kontroll-Objekte können zwar im Animation Window animiert werden, die dabei entstehende Animation ist allerdings erst zur Laufzeit sichtbar. Werden wiederum Änderungen während der Laufzeit – also im sogenannten Play-Modus – vorgenommen, werden diese zurückgesetzt, sobald der Play-Modus verlassen wird. Dieses Plugin sollte somit nur als ergänzendes Werkzeug innerhalb von *Unity* gesehen werden. Im Kurzfilm-Projekt *The Sapling* wurden beispielsweise mehrere Damp-Kontroll-Objekte für die Äste am Kopf der Hauptcharaktere genutzt (siehe Abschn. 7.2).

Kapitel 6

Animation mit *UMotion*

UMotion ist eine Erweiterung für *Unity*, die es ermöglicht in der Game Engine zu animieren. Es können Charaktere mit beiden Rig-Typen (Humanoid und Generic) animiert werden, wobei einige Funktionen bei humanoiden Rigs automatisiert bzw. nur dafür vorhanden sind. Die kostenpflichtige Pro-Version von *UMotion* enthält die Möglichkeit dem Rig Inverse Kinematik Constraints hinzuzufügen, sowie weitere Funktionen wie Animation Layers, Import und Bearbeitung von Animationen aus FBX-Dateien und Child-of Contraints. Da *UMotion* nicht direkt in *Unity* integriert ist, werden die Konfigurations- und Animationsdaten in Asset-Dateien gespeichert. Diese können nicht direkt in der State Machine Mecanim (siehe Abschn. 5.2.1) oder in der Timeline (siehe Abschn. 5.2.2) genutzt werden. Stattdessen werden die Animationen zuerst als Anim-Datei exportiert und dann wie andere Animation Clips genutzt.

Bei der Installation von *UMotion* werden *Unity* zwei neue Fenster – Clip Editor und Pose Editor – hinzugefügt (siehe Abb. 6.1). Der Clip Editor beinhaltet die Timeline die, wie *Unitys* Animation Window (siehe Abschn. 5.2.3), die Animationsdaten in Form eines Dopesheets oder als Animations Kurven anzeigen kann. Des Weiteren werden im Clip Editor die einzelnen Animation Clips und das aktive Projekt verwaltet.

Im Pose Editor wird das zu animierende Rig festgelegt bzw. aktiviert und konfiguriert. Wenn ein Rig für *UMotion* aktiv ist, ist es für andere Funktionen von *Unity* gesperrt, um Konflikte zu vermeiden, es kann jedoch jederzeit deaktiviert, und so für *Unity*-Funktionen freigegeben werden. Der Pose Editor wird in den „Config“ und in den „Pose“ Bereich geteilt. In Config – kurz für Configuration (engl.: Konfigurierung) – wird das Rig für die Nutzung in *UMotion* vorbereitet. In diesem Bereich findet sich u. a. der „IK Wizard“ der das einfache Erstellen von üblicherweise benötigten IK-Constraints für Arme und Beine ermöglicht, sowie Custom IK-Constraints die genauer definiert werden können. Auch die „Reference Pose“ – die angewandt wird, wenn keine Animation auf das Rig wirkt –, sowie „Custom Constraints“ – beispielsweise für Blend Shapes – können unter Config bearbeitet bzw. erstellt werden. Im Bereich Pose findet sich alles was der Animator während des Animierens selbst benötigt. Darunter eine Funktion zum Angleichen der FK- und IK-Rigs, sowie eine Übersicht über alle „Channels“ bzw. Werte des derzeit ausgewählten Objekts.

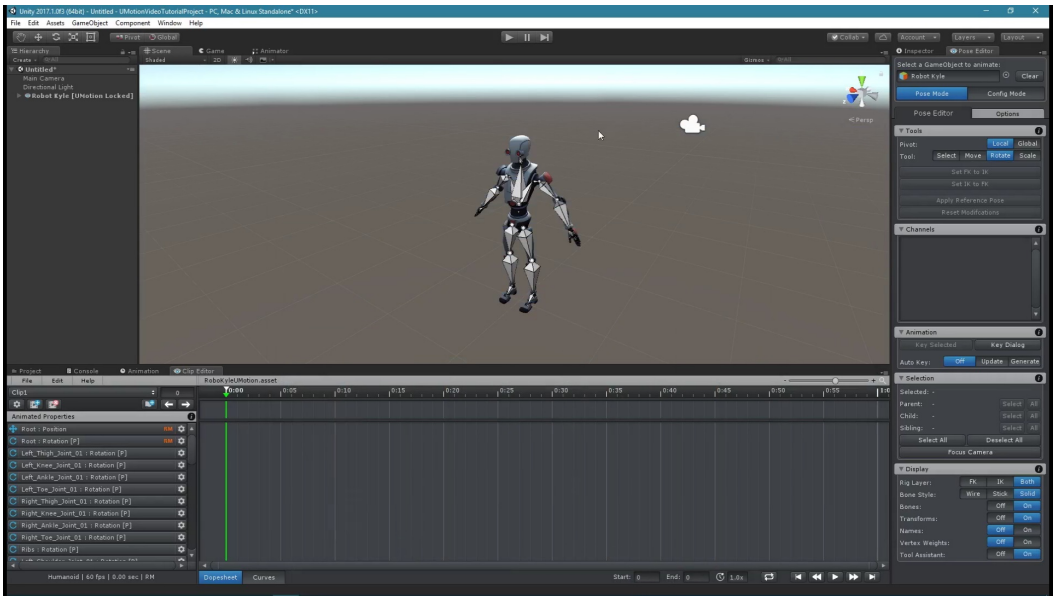


Abbildung 6.1: Ein in *Unity* geöffnetes *UMotion*-Projekt. Im unteren Bereich befindet sich der Clip Editor mit Animated Properties Liste und Timeline. Rechts ist der Pose Editor angedockt. Bildquelle [35].

6.1 Animieren in *UMotion*

In *UMotion* können Animationen von Grund auf erstellt oder bereits Vorhandene bearbeitet werden. Mit den im Plugin integrierten IK-Constraints, können z. B. gleitende Füße in einem bereits vorhandenen Walk Cycle ausgebessert werden. Die Animationen werden, wie in *Unity* üblich, in Animation Clips unterteilt. Bei Spielen werden so zumeist die einzelnen, für die State Machine nötigen, Animationen voneinander getrennt. Für Filme bietet es sich an, für jeden Shot einen Clip anzulegen. Um einen Abgleich mit weiteren Animationen und Kamerabewegungen zu ermöglichen, kann der Clip Editor mit der *Unity*-Timeline synchronisiert werden. Da der zu animierende Clip in den meisten Fällen nicht bei null anfängt, kann die Synchronisation auch mit einem beliebigen Versatz aktiviert werden. Dazu wird der Playhead der Timeline an den Start-Zeitpunkt des zu animierenden Clips gebracht und anschließend im Clip Editor über die Schaltfläche „Sync“ die Synchronisation mit der Timeline aktiviert. Ist der zu animierende Clip bereits in der Timeline platziert und soll weiter bearbeitet werden, kann man diesen auch einfach anwählen und erhält dann beim Aktivieren der Synchronisation die Möglichkeit direkt den ausgewählten Clip zu bearbeiten.

Die einzelnen Attribute die animiert werden können (Position, Rotation und Skalierung der Joints, Blend Shapes, Custom Properties) werden im Clip Editor unter „Animated Properties“ aufgelistet (siehe Abb. 6.1). Die Liste ist, vor allem bei komplexeren Rigs, sehr unübersichtlich, da die einzelnen Attribute zwar hierarchisch geordnet, jedoch nicht entsprechend (z. B. mithilfe von Einrückungen) dargestellt werden. Auch eine Suchfunktion ist nicht vorhanden. Bei Objekten, die zumeist in der Scene Ansicht animiert werden, wie z. B. die Joints des Rigs, ist dieser Faktor nicht sonderlich störend.

Sollen jedoch nicht direkt anwählbare Attribute – wie z. B. Blend Shapes ohne Custom Constraints – animiert werden, kann diese Unübersichtlichkeit ein großer Störfaktor sein. Dementsprechend sollte bereits beim Riggen darauf geachtet werden, die Geometrie, die die Blend Shapes beinhaltet, in der Objekt Hierarchie möglichst weit oben zu platzieren, sodass diese in der Attribut Auflistung relativ am Anfang der Liste stehen. Custom Constraints werden, entsprechend der hierarchischen Anordnung, unter den jeweiligen Parent-Objekten angezeigt. Dies gilt auch für die von *UMotion* generierten IK-Constraints, bestehend aus dem Constraint selbst, sowie einem Constraint-Objekt für den Pole Vector, der bestimmt in welche Richtung die IK gesteuerte Joint-Kette gebeugt wird. Neben den üblichen Positions- und Rotationswerten, wird auch die „Animation“ von IK-Blend (legt fest, ob das IK- oder FK-Rig aktiv ist) und IK-Pinning („pinnt“ einen IK-Constraint an seine derzeitige Position im World Space) angezeigt. Die Steuerungselemente für die einzelnen Werte von IK- oder sonstigen Custom Constraints, finden sich im Pose Editor.

Das Setzen von Keyframes kann entweder manuell oder automatisch erfolgen, ähnlich wie in üblichen Animationsprogrammen. Die Auswahl hierfür findet sich im Pose Editor im Bereich „Animation“. Standardmäßig ist der „Auto Key“ Modus ausgeschaltet, Keyframes müssen also manuell gesetzt werden. Dies geschieht mithilfe der zwei Schaltflächen „Key Selected“ und „Key Dialog“. Key Selected erstellt einen Keyframe für alle ausgewählten Objekte, während sich bei Key Dialog eine Liste aller Attribute öffnet. Die modifizierten Attribute, die aber noch nicht gekeyed wurden, werden rot angezeigt und können nun ausgewählt und mit einem Keyframe versehen werden. Diese Option ist praktisch, wenn nur bestimmte Attribute animiert werden sollen oder ein Fehler unterlaufen ist. Zumeist wird ein Animator jedoch den Auto Key Modus nutzen, der wiederum zwei Optionen hat: „Update“ und „Generate“. Wird Update ausgewählt, so werden nur Attribute die zum jeweiligen Modifikations-Zeitpunkt bereits einen Keyframe besitzen automatisch angepasst. Ist kein Keyframe vorhanden, so muss dieser wie beschrieben manuell gesetzt werden. Wird der Modus Generate gewählt, so werden Keyframes immer gesetzt bzw., wenn bereits ein Key vorhanden ist, modifiziert. Der Manuelle-, sowie der Update-Modus eignen sich vor Allem zum Erstellen der Extrem-Posen einer Animation, während der Generate-Modus zum anschließenden Ausarbeiten der Animation genutzt werden sollte.

Die Animation von Rotationswerten setzt *UMotion* als Quaternion-, statt Euler-Rotationen um. Dies geschieht, um einen Gimbal Lock [1] zu vermeiden. Da Quaternion zwar robuster als Euler ist, die Werte jedoch für den Animator nur schwer verständlich sind, erlaubt *UMotion* keine Bearbeitung der Animationskurven von Quaternion Rotationen. Stattdessen wird „Progressive Quaternion“ zur Verfügung gestellt. Die Animationskurve zeigt in diesem Modus einen stetigen Anstieg und die Veränderung der Werte selbst hat keinen praktischen Nutzen. Jedoch können die Animationskurven und somit die Geschwindigkeit der Rotation beeinflusst werden. Muss der Animator die Rotations-

[1] Der Gimbal Lock ist ein Problem der Euler Rotationsdarstellung. Rotiert die hierarchisch gesehen Mittlere der drei Achsen um 90°, so rotiert sie unweigerlich auf eine der anderen Koordinatenachsen und ist mit ihr identisch. Dadurch kann sie anschließend nicht mehr unabhängig rotiert werden [32]. In einer 3D-Animationssoftware kann die nötige Rotation noch immer über die lokalen Rotationsachsen gesteuert werden, jedoch wird die Rotation nicht über diese berechnet und es kommt zu unerwünschten Drehungen des Objekts, wie in diesem Video [31, T=00:04:41] zu sehen ist.

werte der einzelnen Achsen dennoch genauer über Animationskurven steuern, so kann in der Animated Properties Liste der Rotationsmodus jederzeit umgestellt werden.

Kapitel 7

Rigging und Animation in *The Sapling*

The Sapling ist ein mit *Unity* erstellter Kurzfilm, der von einem alten Baumwesen und dessen Tochter bzw. Schössling (engl.: Sapling) handelt. Sie wandern gemeinsam durch einen Wald, bis das alte Baumwesen sich verändert und der Schössling seinen Weg alleine finden muss. In diesem Kapitel wird beschrieben, wie diese beiden Charaktere für *Unity* vorbereitet, geriggt und anschließend in der Game Engine animiert wurden. Dabei wird auch auf Fehlversuche bzw. Methoden eingegangen, die im finalen Projekt nicht genutzt wurden.

Die Charaktere wurden in *ZBrush*[1] modelliert und die Retopologie in *Maya* erstellt. Durch die besonderen Formen der „Rinde“, ist neben einem für Animation geeigneten Edge Flow, auch auf möglichst genaue Herausarbeitung der Formen geachtet worden. In *Maya* sind außerdem die Models für das Blattwerk erstellt worden.

7.1 Rigging

Die Retopologie des Charakter-Models, ist in *Akeytsu* importiert worden. In diesem Programm wird ein Basis-Rig bereitgestellt, dass den Vorgaben eines humanoiden Rigs in *Unity* entspricht. Für die Baumwesen ist dieses Rig angepasst worden, u. a. sind Joints für die Äste an Kopf und Armen hinzugefügt worden. Auch dem Kopf wurden mehrere Joints hinzugefügt, um grundlegende Gesichtsanimation zu ermöglichen. Entfernt wurden die Joints des kleinen Fingers und der Zehen. Schlussendlich sind beide Charakter-Rigs in *Unity* als Generic-Rigs deklariert worden.

Das zwei-stufige Weight Painting System von *Akeytsu* hat sich bei der Arbeit mit demselben, als sehr intuitiv und übersichtlich herausgestellt. Im ersten Schritt werden die Vertices zu 100 % je einem Joint zugewiesen. Vor allem bei eng aneinander liegender Geometrie, z. B. bei den Ästen am Kopf der Charaktere, ist es von Vorteil gewesen die Vertices zuerst grob einzuteilen. Wenn zum zweiten Schritt übergegangen wird, werden die Weight Painting Übergänge zwischen den einzelnen Joints automatisch geglättet. Zuvor kann der Glättungsgrad für jeden Joint eingestellt werden. Die Einstellung ist dreistufig möglich, wobei der höchste und niedrigste Wert manuell eingestellt werden können. Alternativ kann die Glättung pro Joint auch deaktiviert werden, was vor allem bei Joints, die eine separate Geometrie bewegen (z. B. Augen) von Nutzen ist. Das

[1] https://pixologic.com

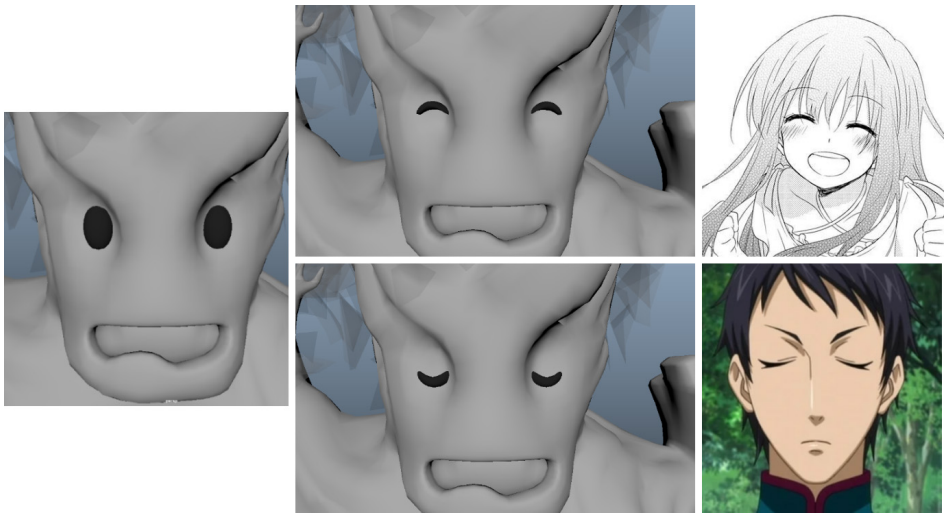


Abbildung 7.1: Links: Der Baumwesen-Charakter mit geöffneten Augen. Mitte oben: „Glückliche“ Augen, inspiriert von glücklichen Gesichtsausdrücken in Mangas (Rechts oben. Bildquelle [34]). Mitte unten: Geschlossene Augen bzw. Blinzeln ebenfalls inspiriert von Augen in Mangas (Rechts unten. Bildquelle [36]).

manuelle Anpassen des so generierten geglätteten Weight Paintings ist ebenfalls vergleichsweise einfach und nutzerfreundlich.

Ursprünglich sollten auch die Animationen für *The Sapling* in *Akeytsu* erstellt werden, jedoch unterstützt das Programm keine Blend Shapes, die vor allem für die Animation der Mimik benötigt worden sind (Anmerkung: Genutzt wurde Version 1.2.1, allerdings werden seit Version 2019-4-1, die im Dezember 2019 erschien, auch Blend Shapes unterstützt). Daher sind die geriggten Models erneut exportiert und in *Maya* die nötigen Blend Shapes erstellt worden. Vorrangig waren dies Blend Shapes zur Animation der Gesichter. Da die Charaktere comicartige Knöpfchen Augen besitzen, sind auch für diese Blend Shapes in mehreren Formen angelegt worden, die stark von gezeichneten Augen aus Anime und Manga inspiriert sind (siehe Abb. 7.1). Auch korrektive Blend Shapes sind genutzt worden. Beispielsweise ist erst beim Riggen der Charaktere aufgefallen, dass der Kiefer des alten Baumwesens beim Schließen nicht wie geplant über der Oberlippe liegt, sondern im oberen Teil des Kopfes verschwindet. Daher ist eine korrektive Blend Shape angelegt worden, die dafür sorgt, dass der Unterkiefer bei geschlossenem Mund über der Oberlippe liegt.

7.2 Animation

Da das Programm *Akeytsu* auf Animationen für Videospiele bzw. Game Engines ausgerichtet ist und bereits für das Rigging der Charaktere genutzt wurde, sollten zu Beginn auch die Animationen in dieser Software erstellt werden. Die Aufteilung der einzelnen Animationen erfolgt, wie auch in *Unity*, in einzelne Animations Clips, die unabhängig voneinander erstellt und bearbeitet werden. Als erste Animation ist ein Walk Cycle für das alte Baumwesen erstellt worden, anhand dessen ein Workflow für die weiteren

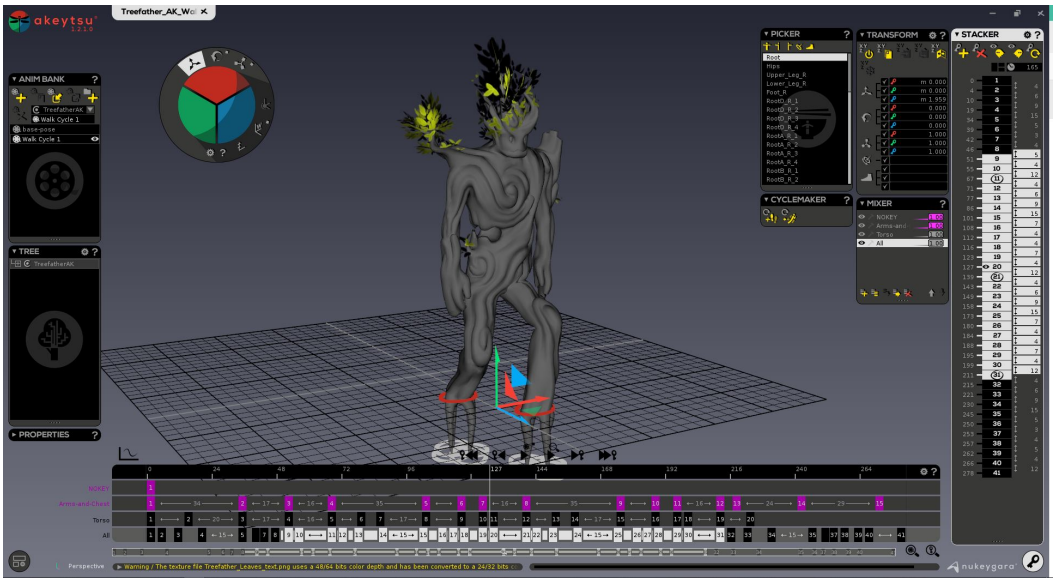


Abbildung 7.2: Bildschirmaufnahme aus Akeytsu mit der geöffneten Walk Cycle Animation für das Baumwesen. Links befindet sich die „Animation Bank“ mit den einzelnen Animations Clips. Unten ist die Timebar. Zu sehen sind die Stacker-Ebenen mit jeweils unterschiedlich gesetzten Keyframes. Rechts befinden sich u. a. Stacker und Mixer, die die Stacker und Keyframes des derzeit ausgewählten Stackers anzeigen.

Animationen getestet werden sollte.

Die Erstellung des Walk Cycles stellte sich, wie auch das Riggen, als sehr intuitiv heraus. Besonders hervorzuheben ist eine Funktion von *Akeytsu*, die für Animatoren, die bereits mit anderer Animationssoftware gearbeitet haben, möglicherweise gewöhnungsbedürftig ist: Keyframes werden in *Akeytsu* nicht für die einzelnen ausgewählten Joints angelegt, sondern immer für ein ganzes Set, genannt „Stacker“. Standardmäßig ist nur ein Stacker, der alle Joints beinhaltet, vorhanden. Der Animator kann im „Mixer“-Fenster selbst weitere Stacker und deren Inhalt bestimmen. Es ist z. B. sinnvoll für die Joints von Gesicht und Armen jeweils neue Stacker-Sets zu erstellen. Da die Stacker wie Animation Layer arbeiten, überschreiben höher liegende Stacker die darunterliegenden, wenn derselbe Joint auch in diesen vorhanden ist. In der Timeline werden die Keys der einzelnen Stacker separat voneinander in einzelnen Ebenen angezeigt, wobei immer alle Stacker sichtbar sind (siehe Abb. 7.2). Ergänzend dazu gibt es noch eine weitere Auflistung der Keyframes im Stacker-Fenster, das immer die Keyframes des im Moment ausgewählten Stackers anzeigt. Dieses gibt eine sehr gute Übersicht und Anpassungsmöglichkeit des Timings zwischen den einzelnen Keyframes.

Anhand weiterer Funktionen von *Akeytsu* ist der Fokus auf Videospiel-Animation noch deutlicher erkennbar. Beispielsweise ist es möglich, im Fenster „Cyclemaker“, die Animation zu duplizieren und am Ende der bereits vorhandenen Keyframes anzuhängen. Außerdem kann ein ausgewählter Bereich (z. B. die soeben duplizierte Animation) gespiegelt werden. Dies erleichtert vor allem Animationen wie Walk Cycles, da dem Animator die Animation des zweiten Schritts erspart wird. Für den ersten Walk Cycle für *The Sapling* ist die Duplizierungs-Funktion auch für die Lösung eines anderen Problems

zum Einsatz gekommen: Da *Akeytsu* mit Quaternion-Berechnung für die Rotations-Kurven arbeitet, ist es nicht möglich, die Tangenten der einzelnen Kurven-Punkte zu bearbeiten. Dadurch entstehen speziell am Anfang und Ende der Animations-Clips, oft unpassend steile Kurven (z. B. wenn die Animation mit dem Hoch- oder Tiefpunkt einer Animationskurve startet und endet), die in zuckenden oder anderweitig unpassenden Bewegungen beim Abspielen in Schleifen resultiert. Da im Falle des Walk Cycles die automatischen Tangenten jedoch korrekt berechnet worden sind, wenn jeweils davor und danach weitere Keyframes gesetzt wurden, ist die gesamte Animation einmal dupliziert und anschließend nur der „mittlere“ Part exportiert worden.

Bald hat sich jedoch gezeigt, dass der Verzicht auf Blend Shapes für dieses Projekt nicht möglich gewesen ist. Es ist überlegt worden, die Blend Shapes im Anschluss direkt in *Unity* zu animieren. Wegen der eingeschränkten Animations-Möglichkeiten in der Game Engine, ist diese Möglichkeit jedoch als Notlösung verbucht worden. Also wurde vorerst erneut auf *Maya* zurückgegriffen. Da bereits ein Rig mit Blend Shapes erstellt worden war, das genau denselben Joint-Aufbau aufweisen konnte (da es aus *Akeytsu* importiert wurde), ist angenommen worden, dass ein einfaches Retargeting der Animationen auf dieses Rig in *Unity* möglich sein würde. Mehrere Hürden wie die Tatsache, dass *Unity* eine exakt gleiche Benennung der Joints verlangte (in *Maya* waren durch den Import Namespaces hinzugefügt worden) und die Frage wie die Joint-Animation aus *Akeytsu* mit der Blend Shape-Animation aus *Maya* kombiniert werden sollte, verzögerten die Findung einer praktikablen Lösung. Schließlich ist *Maya* wieder als alleiniges Animationsprogramm in Betracht gezogen worden, was weiterer Rigging-Arbeit bedurft hätte, da bis zu diesem Zeitpunkt kein Control-Rig benötigt worden war.

Während dieser Überlegungen ist das Team durch Zufall auf das *Unity*-Plugin *UMotion* (siehe Kapitel 6) gestoßen, das das Animieren in der Game Engine ermöglicht. *UMotion* ist dann zum ersten Testen in der „Blumenszene“ in Akt 1 des Kurzfilms zum Einsatz gekommen. Da das Baummädchen in diesem Teil des Films noch von ihrem Vater – dem Baumwesen – getragen wird und dieser in der Blumenszene die meisten Aktionen vollführt, ist er als Erstes animiert worden. Die Aktionen bzw. Animationen des Mädchens orientieren sich dann an denen des Vaters.

Genutzt worden ist die in *Maya* mit Blend Shapes vervollständigte Version des Rigs (siehe Abb. 7.3). In der Konfiguration sind Inverse Kinematik Constraints auf Arme und Beine angewandt worden. Des Weiteren ist die X-Skalierung des Hüft-Joints auf -1, eingestellt worden. Dies hat den Grund gehabt, dass durch die ursprüngliche Version des Baumwesen-Models, das Mädchen immer auf der von der Kamera abgewandten Seite sitzen würde. Da dies erst nach dem Rigging aufgefallen ist, ist auf diese einfache Variante zurückgegriffen worden, um den Charakter zu spiegeln. Um ein unbeabsichtigtes Animieren dieses Wertes zu verhindern, sind die Skalierungs-Werte für den Hüft-Joint nach der Anpassung gesperrt worden.

Sobald der Charakter konfiguriert worden ist, ist zunächst als Test eine bereits vorhandene Animation – der Walk Cycle der bereits in *Akeytsu* animiert worden war – importiert worden. Dies hat ohne Probleme funktioniert und auch die Anpassung der IK-Constraints an die Animation ist möglich gewesen. Eine Funktion, die vor allem für das Erstellen von Kurzfilmen oder Spiel-Cinematics nützlich wäre, wäre das automatischen Anhängen von einem Animations Clip an einen Anderen. Dies ist derzeit nur über das Kopieren und Einfügen der Keyframes von einem Clip in den Nächsten mög-

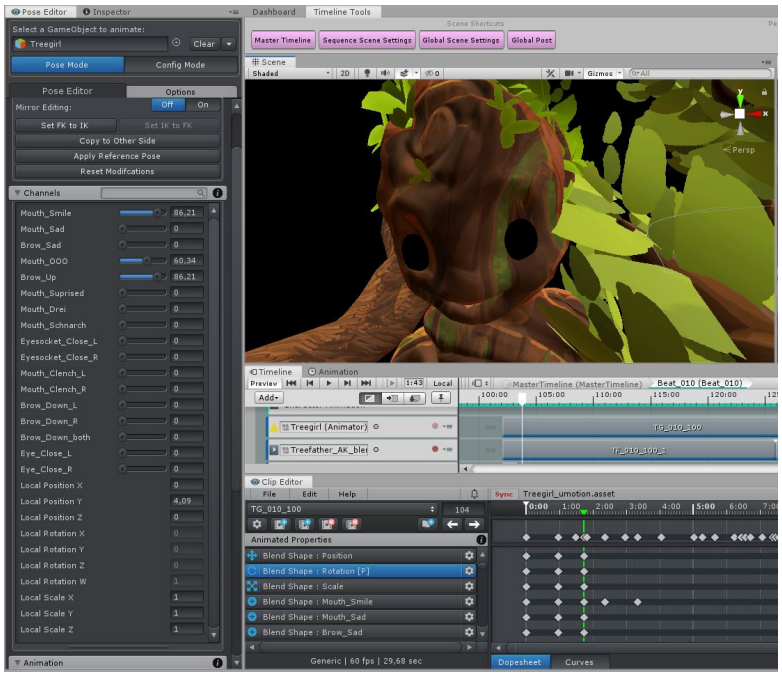


Abbildung 7.3: Das im geöffnete *UMotion*-Projekt des Baummädchens. Links im Pose Editor ist die Auflistung der Blend Shapes zu sehen. Die einzelnen Blend Shapes sind auch unten in der Timeline aufgelistet und sind teilweise bereits animiert.

lich. Besonders nützlich wäre eine optionale automatische Anpassung der Root Motion des angehängten Animation Clips an die des Vorherigen. Die *Unity*-Timeline stellt eine solche Funktion für aufeinander folgende Animations-Clips zur Verfügung.

Für den Export der Clips aus *UMotion* wird in den Einstellungen ein Zielordner im Assets-Ordner des *Unity*-Projekts festgelegt. In *The Sapling* ist pro Charakter ein eigener Ordner angelegt und die einzelnen Clips sind entsprechend der Shot-Nummern benannt worden. Da sowohl für den Export eines einzelnen Clips als auch für den aller Clips Hotkeys festgelegt werden können, gestaltet sich die Übertragung der Animationen in die *Unity*-Timeline des eigentlichen Films, als relativ unkompliziert. Für die Blumenszene sind pro Charakter zwei Clips angelegt worden, die im Film durch einen Shot ohne Blick auf die Charaktere getrennt sind – der zweite Clip überdauert somit mehrere Kameraschnitte. Dies erleichtert das Erstellen einer in sich schlüssigen, durchgehenden Animation. Für die Animationen der einzelnen Clips, ist zuerst ein grobes Layout erstellt worden, das ein ungefähres Timing der einzelnen Shots erlaubt hat. In dieser groben Version sind die Clips bereits erstmalig exportiert und in die Timeline des Films eingefügt worden. Anschließend ist die *Unity*-Timeline mit dem Clip Editor von *UMotion* synchronisiert worden, sodass im Game View immer der entsprechende Bildausschnitt zu sehen war. So ist es möglich gewesen, die Animationen noch besser auf die Sicht des Zusehers anzupassen. Unter anderem konnte so die Hand des Baumwesens schon beim Animieren passend unter den Lichtstrahl platziert werden, ohne mehrmals zwischen *Unity*-Timeline und *UMotion*-Clip Editor wechseln zu müssen.

Da das Baummädchen im ersten Teil von *The Sapling* auf der Schulter des alten

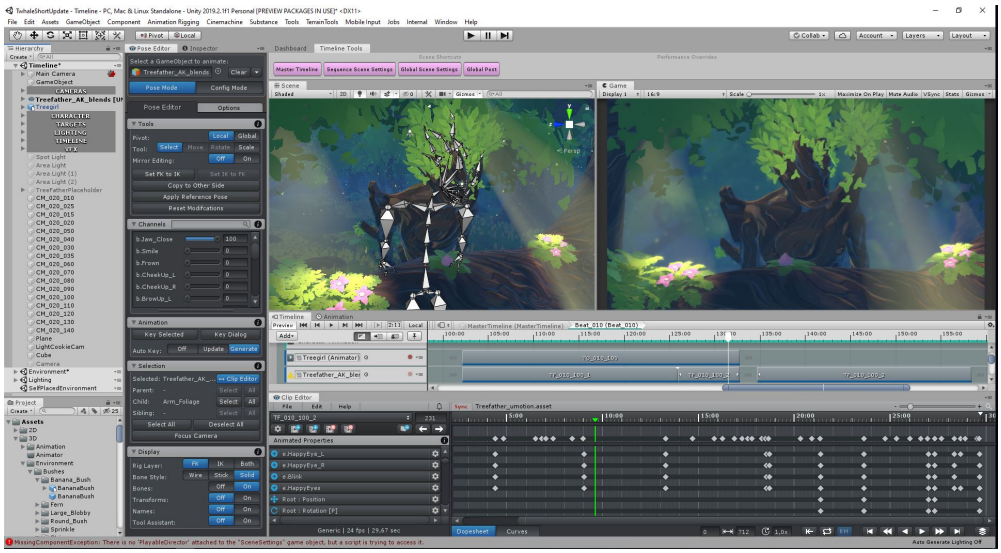


Abbildung 7.4: Das *UMotion*-Projekt des Baumwesens. Die Timeline von *Unity* ist mit der Timeline im Clip Editor synchronisiert. Da das Rig des Baumwesens während der Nutzung in *UMotion* in der Game Engine „deaktiviert“ ist, greift der Parent-Constraint des Baummädchens nicht und es befindet sich nicht auf seiner Schulter.

Baumwesens sitzt, ist es naheliegend gewesen, die Beiden durch einen Parent- bzw. Child-Constraint aneinander zu binden. *UMotions* Child-Of-Constraint akzeptiert allerdings nur Objekte die sich im selben *UMotion*-Projekt befinden. Somit wäre es nötig gewesen beide Charaktere in einem Projekt zu vereinen, was u. a. das unabhängige Animieren und Exportieren der Clips erschwert und auch für noch weniger Übersichtlichkeit in der Liste der Animated Properties gesorgt hätte. Der Child-Of-Constraint ist allerdings in den darauffolgenden Szenen des Films genutzt worden, um die Blume an den Kopf des Baummädchens zu binden. Für das Verbinden von Baumwesen und Baummädchen ist jedoch stattdessen der Parent-Constraint von *Unity* genutzt worden. Dieser funktioniert in der normalen Film-Timeline ohne Weiteres, die *UMotion*-Timeline kann diesen jedoch nicht einwandfrei verarbeiten. Bei der normalen Wiedergabe der Animation bleibt das gebundene Objekt (in diesem Fall das Mädchen) an seinem ursprünglichen Platz und springt erst nach dem Pausieren der Wiedergabe an das Parent-Objekt (die Schulter des Baumwesens). Da im Fall von *The Sapling* die Haupt-Animation vom Baumwesen ausgeht, ist es relativ einfach gewesen das Mädchen an den Extrem-Posen entsprechend zu animieren. Für Detail-Animation – bei der ein sich durch die Welt bewegender Charakter ohnehin unpraktisch wäre – ist der Parent-Constraint kurzzeitig deaktiviert worden, sodass das Mädchen an seiner Ursprungsposition verblieben ist. Anschließend ist die Detail-Animation erstellt und der Parent-Constraint wieder aktiviert worden. Wie in Abb. 7.4 zu sehen, ist der Parent-Constraint außerdem nicht mehr funktionstüchtig, wenn das Parent-Objekt „deaktiviert“ bzw. nicht vorhanden ist. Da *UMotion* den für das Plugin jeweils aktiven Charakter für alle anderen Funktionen der Game Engine deaktiviert, kann auch der Parent-Constraint nicht auf den benötigten Schulter-Joint zugreifen. Da das Baumwesen jedoch relativ unabhängig agiert,

konnte beim Animieren auf den Constraint größtenteils verzichtet werden. Im zweiten Clip der Szene interagieren die beiden jedoch, weshalb dieser nur über mithilfe eines Work-Arounds und direkter Clip-Synchronisierung der Timlines bearbeitet wurde. Der Playhead der *Unity*-Timeline ist zunächst an einen Zeitpunkt gebracht worden, an dem sich das Baummädchen bereits in der Position des zweiten Clips befindet. Anschließend ist die Clip-Synchronisation aktiviert und erst danach der Baumwesen-Charakter für *UMotion* freigegeben worden.

Neben *UMotion* ist in *Unity* auch das Animation Rigging Package (siehe Abschn. 5.3) zum Einsatz gekommen. Alle Joints in den Ästen an Kopf und Schulter des Baumwesens haben „Damped Transform“ Komponenten erhalten. Diese sorgen dafür, dass die Rotation und teilweise auch Position des angegeben Quell-Objekts (der in der Rig-Hierarchie direkt darüber liegende Joint) übernommen und zeitversetzt abgespielt wird. Da das Animation Rigging Package für den Gebrauch in Videospielen, mit State Machine gesteuerten Charakteren, konzipiert ist, musste für die Nutzung mit Timeline-animierten Charakteren ein eigenes Skript erstellt und eingefügt werden, das das Animation Rig vor dem ersten Animations-Shot einmal deaktiviert und wieder aktiviert. Da *The Sapling* mit einem langen Establishing-Shot über den Wald beginnt, kann das Skript in dieser Zeitspanne ausgeführt werden.

Kapitel 8

Fazit

Die Game Engine *Unity* wird immer wieder mit neuen Funktionen und Möglichkeiten verbessert, nicht zuletzt mit Werkzeugen zur Erstellung von Animationsfilmen. Jedoch bleibt die Handhabung von Rigging und Animation seit Längerem auf demselben Stand. Es wird normalerweise auf externe Software zurückgegriffen, in der das Animieren der Charaktere abgekoppelt von der restlichen Echtzeit-Umgebung geschieht. Oft ist auch der Arbeitsablauf in diesen Programmen unnötig kompliziert. Wurde ein Charakter geriggt und animiert, so können die Rigging- und Animationsdaten von Constraints nicht über die standardmäßig genutzten Dateiformate wie FBX übertragen werden. Selbst kleine Korrekturen einer Animation können in *Unity* nur schwer umgesetzt werden, weshalb die Anpassung von Animationen ebenfalls in der externen Software geschehen.

Für Animatoren die an Filmen, aber auch Videospielen, in *Unity* arbeiten, wäre daher ein System zur Erstellung von praktikablen Constraint-Rigs und dem anschließenden Animieren der Charaktere sinnvoll. So können die Vorteile der Echtzeit-Umgebung auch in diesem Bereich genutzt werden, da die Charaktere immer direkt mit der Umgebung interagieren können. Neben diesen Vorteilen für den Animator kann eine Game Engine noch weitere Funktionen bieten. Der *Source Filmmaker* ist ein sehr gutes Beispiel dafür, wie Entwickler ihre Game Engine ausgebaut haben, um die Funktionen derselben auch für Animationsfilme zu nutzen. Das Plugin *UMotion* bietet eine gute Basis zum Animieren in *Unity*. Es besitzt ein Constraint-System, dass neben Benutzerdefinierten, auch die gängigste Constraint-Variante Inverse Kinematik bietet. Das User Interface von *UMotion* ist nicht ideal und bedarf Überarbeitung jedoch bleibt der größte Nachteil die indirekte Integration in die Game Engine, wodurch Konflikte mit anderen nativen Komponenten entstehen können – ein Mangel der bei direkter Integration behoben werden könnte. Zusätzlich dazu sollte das bereits vorhandene *Unity*-eigene Plugin Animation Rigging dahingehend verbessert werden, dass es auch für Animationen in der Timeline funktioniert.

Anhang A

Sequenzprotokoll: *Team Fortress 2 – Meet the Medic*

Timecode	Kamera	Bildinhalt (Fokus auf Animation)	Dialog/ Text (engl.)
00:00:23	Totale. Kamera schwenkt von links-oben nach rechts-unten	Scout rutscht von der Glasscheibe.	**Heavy**: lacht **Medic**: „Wait, wait, wait – it gets better."
00:00:25	Totale. Weiterführung des Kamera-schwenks	Heavy liegt mit geöffnetem Abdomen auf einem Behandlungstisch. Medic hält sein Herz in der Hand und erzählt eine Geschichte. Überall verteilt sitzen Tauben. Angepasste Versionen der 3D-Models: Medic ohne Arztkittel, Arme voller Blut. Heavy mit offenem Abdomen und zerschossener Brust. Die Innereien werden ebenfalls animiert (z.B. sich aufblähender Lungenflügel).	**Medic**: „When the patient woke up, his skeleton was missing and the doctor was never heard from again!" lacht
00:00:33	Halbnah	Beide lachen herzhaft über Medics Geschichte. Medic seufzt amüsiert und stützt sich ohne Rücksicht auf seinen Patienten mit dem Ellbogen auf Heavys Brustkorb ab.	**Heavy**: lacht laut **Medic**: „Ahh, anyway.."
00:00:38	Nah.	Weiterhin amüsiert beendet Medic die soeben erzählte Geschichte. Heavy blickt Medic daraufhin beunruhigt an. In diesem Moment streckt eine Taube ihren Kopf aus den Innereien von Heavy. Beide sehen die Taube entsetzt an, aber aus unterschiedlichen Gründen.	**Medic**: „That's how I lost my medical license." – „Archimedes, no!"
00:00:43	Halbnah	Medic verscheucht die Taube verärgert und wischt dann seine Hand an seinem Jacket ab. Heavy sieht ihn wegen seiner Aussage nun verärgert an. Medic zuckt mit den Schultern, scheinbar ist er sich nicht bewusst über die Problematik einer Taube im Abdomen.	**Medic** angeekelt: „It's filthy in there!" – entschuldigend: „Birds, haha."
00:00:48	Detail	Medic greift nach dem Über-Modul.	**Medic**: „Now…"
00:00:50	Halbnah	Er steckt Heavys Herz auf das Über-Modul, dieser ist verärgert über die nachlässige Handhabung seines Herzens und zieht die Augenbrauen zusammen. Medic ist weiterhin ungerührt und hält das Herz unter den roten Heilungsstrahl. Das Herz explodiert sofort und Medics Gesicht zeigt zum ersten Mal wahres Entsetzten wegen dieses Fehlers.	**Medic**: „Most hearts couldn't withstand this voltage, but I'm fairly certain your heart…"
00:00:55	Nah	Ein Stück des explodierten Herzens trifft Archimedes und schleudert ihn weg. Die anderen beiden Tauben blicken ihm hinterher.	
00:01:02	Halbnah	Heavy hat nicht gesehen was passiert ist und blickt sich um, sein Gesicht zeigt, dass er nicht besonders klug und aufmerksam ist. Medic versteckt das Über-Modul vor Heavys Blick.	**Heavy**: „What was noise?" **Medic**: „The sound of progress, my friend."
00:01:03	Nah	Medic öffnet den Kühlschrank.	
00:01:04	Detail	Er greift zu dem großen Herz in einer Schale mit der Aufschrift „Mega Baboon". Dahinter liegt der abgetrennte Kopf eines Spys, der um seinen Tod bittet. Ton und Gesichtsausdruck verdeutlichen, dass er bereits öfter seinen Tod verlangt hat.	**Medic**: „Ah, perfect." **Spy Head**: „Kill me." **Medic**: „Later."
00:01:06	Halbnah	Medic steckt das Über-Modul in das Mega Baboon Herz.	**Medic**: „Where was I? Ah, there we go."
00:01:10	Detail Die Kamera dreht sich langsam um das Herz	Er hält das Herz unter den roten Heilungsstrahl.	**Medic**: „Come on, come on."
00:01:14	Totale Kamera führt die Drehung des vorherigen Shots fort	Medics Gesicht verzieht sich zu einer verrückten, lachenden Grimasse. Heavy rückt vorsichtig etwas weg von ihm, sein Gesicht verrät Unbehagen.	**Medic**: lacht verrückt

00:01:18	Close-Up	Heavy lacht verunsichert mit, scheint aber nicht zu wissen warum gelacht wird.	
00:01:20	Detail	Die Über-Anzeige gerät in den roten Bereich.	
00:01:21	Close-Up	Medic dreht nun das Gesicht zur Sicherheit doch etwas von dem Herz weg, er erwartet, dass das Herz gleich wieder zerplatzt.	
00:01:22	Nah	Die beiden äußeren Tauben treten ein paar Schritte von Archimedes weg.	
00:01:23	Close-Up	Medics Augen sind inzwischen geschlossen. Als das Herz zur Ruhe kommt öffnet er diese zögerlich und blickt hinab.	
00:01:25	Nah	Freudig überrascht betrachtet er das nun metallisch-rot schimmernde Herz…	**Medic**: „Oh, that looks good.“
00:01:28	Nah	… und lässt es in Heavys Abdomen fallen.	**Medic**: „Very nice there.“
00:01:30	Close-Up	Heavy betrachtet das Herz skeptisch. Auch ihm dämmert langsam, dass dies kein normaler Eingriff ist.	**Heavy**: „Should I be awake for this?“
00:01:32	Nah	Medic lacht amüsiert und rückt dann seine Brille zurecht. Er beginnt das viel zu große Herz mit großer Anstrengung (seine Augenbrauen senken sich und die Zähne sind zusammengepresst) in Heavys Brustkorb zu schieben.	**Medic**: „Ah haha. Well, no. But as long as you are, could you hold your rib cage open a bit? I can't… seem…“
00:01:39	Close-Up	Heavys Gesicht ist schmerzerfüllt verzerrt.	**Heavy**: schreit schmerzerfüllt auf
00:01:40	Halbnah	Heavy hält eine seiner Rippen in der Hand und sieht diese entsetzt an. Medic blickt kurz entsetzt auf die Rippe, fängt sich aber schnell und tätschelt Heavy mit einem übertriebenen Lächeln die Wange. Er nimmt die Rippe und wirft sie über die Schulter. Als er sich wegdreht verändert sich sein Gesichtsausdruck schlagartig zu besorgniserregt.	**Medic**: „Oh, don't be such a baby, ribs grow back!“
00:01:46	Nah	Als er Archimedes zuflüstert ist sein Blick wieder besorgt.	**Medic** flüsternd: „No they don't.“
00:01:47	Close-Up	Archimedes gurrt und legt seinen Kopf schief.	
00:01:49	Halbnah Zoom out	Medic nimmt das Strahlengerät und richtet es auf Heavys Brustkorb. Er setzt wieder seinen böse-grinsendem Blick auf.	
00:01:51	Close-Up	Heavy sieht noch immer beunruhigt zu.	
00:01:52	Halbnah		
00:01:53	Close-Up Zoom in auf den Abdomen, danach Schwenk zu Heavys Gesicht	Heavys Abdomen verheilt in Sekundenschnelle unter dem Heilungsstrahl. Als er bemerkt, dass er geheilt ist grinst er.	
00:02:00	Halbnah	Heavy greift sich an die Brust, die nun rot leuchtet und atmet tief ein und aus. Als 3D-Model wird wieder sein Standard-Model eingesetzt.	**Heavy**: „What happens now?“
00:02:05	Halbnah	Medic hilft Heavy sich aufzurichten.	**Medic**: „Now?“ kichert
00:02:09	Close-Up	Medics Gesichtsausdruck verheißt Unheilvolles – seine Brauen sind gesenkt und er lächelt.	**Medic**: „Let's go practice medicine.“
00:02:12	Detail	Die roten Gummihandschuhe werden strammgezogen.	
00:02:14	Detail	Der Mantel flattert.	
00:02:14	Detail	Er schnallt den Rucksack auf seinen Rücken.	

00:02:15	Detail	Detail-Shot des Heilungsstrahl Geräts.	
00:02:16	Halbtotale	Das Tor zum medizinischen Labor öffnet sich. Heraus tritt Medic begleitet von seinen zahlreichen Tauben. Als 3D-Model wird nun wieder das Standard-Model von Medic eingesetzt.	
00:02:24	Halbnah	Heavy läuft an Medic vorbei nach draußen.	
00:02:26	Totale Kamera schwenkt von Medic nach rechts	Kamera schwenkt über das Schlachtfeld. Heavy wurde vermutlich mit seinem Standard-Walk-Cycle animiert. Mehrere Hintergrund-Charaktere verharren in Idle-Positionen. Die Raketen wurden möglicherweise im Gameplay-Record Modus abgeschossen und aufgenommen.	
00:02:29	Nah Kamera folgt Demoman	Demoman schiebt seinen Rollstuhl so schnell er kann, wird jedoch von Raketen getroffen.	**Demoman**: „Medic!“
00:02:31	Close-Up	Medic zuckt zusammen.	
00:02:32	Halbtotal	Demoman landet mit dem Gesicht voran am Boden.	
00:02:34	Close-Up Kamera folgt Medics Hand	Medic lächelt selbstsicher und schiebt seine Brille zurecht. Dann aktiviert er den Heilungsstrahl – dabei verzieht sich sein Gesicht wieder zu einem bösen Grinsen.	
00:02:37	Nah zu Amerikanisch Zoom out	Er zielt den Strahl auf Demoman.	
00:02:40	Totale zu Nah Kamerafahrt zu Demoman	Demoman wird geheilt, reißt sich den Gips vom Arm – ein zusätzliches 3D-Model, das an sein Standard-Model geparented wurde – und greift nach seiner Waffe.	
00:02:43	Amerikanisch	Medic dreht sich zur Seite und zielt auf Scout.	
00:02:46	Close-Up	Scout blickt hoch. Sein blaues Auge verheilt und der fehlende Zahn wächst nach. Hierfür wurde ein separates 3D-Model mit Blend Shapes genutzt, ähnlich wie bei Heavy auf dem Operationstisch.	
00:02:49	Halbtotale	Er springt voller Energie auf…	**Scout**: „Yeah!“
00:02:50	Halbnah	… und fängt seinen Baseballschläger aus der Luft.	
00:02:51	Amerikanisch	Heavy sucht Deckung hinter einem Lastwagen. Scout läuft an ihm vorbei.	**Scout**: „Woo hoo hoo!“
00:02:52	Halbtotale	Der gegnerische Soldat springt aus der Deckung, Scout schlägt ihn im Vorbeilaufen mit seinem Baseballschläger.	**Scout**: „Oh yeah!“
00:02:54	Close-Up	Heavy blickt aus seiner Deckung zu den heranstürmenden Gegnern.	
00:02:55	Close-Up	Close-Up zu den Beinen der gegnerischen Soldaten. Die 3D-Models der Soldaten wurden durch ausgetauschte Kopfbedeckungen variiert. Ihre Animationen sind versetzte Walk-Cycles mit unterschiedlichen Waffen. Aufgenommen wurden diese vermutlich im Gameplay-Record Modus.	
00:02:56	Halbnah	Torso Shot der heranstürmenden Soldaten.	**Heavy**: „Doctor!“
00:02:58	Close-Up	Heavy ruft Medic zu und sieht zu ihm.	**Heavy**: „Are you sure this will work?“
00:03:00	Nah	Medics Gesichtsausdruck ist trotz seiner Aussage unbesorgt.	**Medic**: lacht „I have no idea!“

Anhang B

Inhalt der CD-ROM/DVD

B.1 PDF-Dateien

Pfad: /

_Unity-als-Charakter-Animationstool.pdf Masterarbeit
SequenzprotokollMeettheMedic.pdf Sequenzprotokoll von *Meet the Medic*

B.2 Bilder

Pfad: /images

Akeytsu_WalkCycle.jpg
AnimationWindow.jpg .
Charakter-Squash-and-Stretch.png
cloth.jpg . . . . . .
MecanimStateMachine.png
Medic-Models.jpg . . .
MoCap_Adam.png . . .
Overlapping-Action.jpg
Source-Filmmaker.jpg .
Source-Filmmaker_UI.jpg
StacysMom.jpg . . . . .
Sylvanas_Comparison.jpg
The-Sapling-Umotion-1.jpg
The-Sapling-Umotion-Blend-Shapes.jpg
timeline_example.png .
Treefather_Augen.jpg .
UMotion_example.jpg .
Yennefer_LookAt.png .

B.3 Online-Quellen

Pfad: /online

- ADAM-Episode-1.pdf
- Adobe-AfterEffects.pdf
- Akeytsu.pdf
- Baymax-Dreams.pdf
- BlizzCon2016-In-Game-Cinematics-Panel-Transcript.pdf
- BTS-of-the-Cinematic-Dialogues-in-The-Witcher-3.pdf
- CDProjectRed.pdf
- Disney.pdf
- Euler-Gimballock-Explained.pdf
- Eulerwinkel.pdf
- Game-Engines-how-do-they-work.pdf
- HYper-Anime-Eyes-happy.pdf
- Installation-First-steps-UMotion-Official-Tutorials.pdf
- Lau-Black-Butler-Wiki.pdf
- LIVE-Face-Reallusion.pdf
- Maya.pdf
- Meet-the-Heavy.pdf
- Meet-the-Medic.pdf
- Meet-the-Scout.pdf
- Meet-the-Team.pdf
- Mixamo.pdf
- Movies-Official-TF2-Wiki.pdf
- NaughtyDog.pdf
- Pixologic-ZBrush.pdf
- Reallusion-iClone.pdf
- Rooster-Teeth.pdf
- Rooster-Teeth-Red-vs-Blue.pdf
- Second-Life.pdf
- SONDER-Vimeo.pdf
- Source-Filmmaker.pdf
- Source-Filmmaker-FAQ.pdf
- Stacys-Mom-Sims-Version.pdf
- SteamWorkshop.pdf
- Team-Fortress-2.pdf
- The-Stormwind-Extraction.pdf
- The-Wrathgate.pdf

UMotion-Soxware.pdf .
UncannyValley.pdf . . .
Uncharted-4-Animation-Reel-Breakdown.pdf
Unity3d.pdf . . . . . .
Unity-Manual-Animation-Window.pdf
Unity-Manual-Blend-Trees.pdf
Unity-Manual-Rig-Tab.pdf
Unity-Manual-State-Machine-Basics.pdf
Unity-Manual-Timeline.pdf
Unity-Manual-Track-list.pdf
Unity-Manual-Using-Animation-Curves.pdf
ValveCorporation.pdf .
Valve-Developer-Manipulating-objects-overview.pdf
Valve-Developer-Puppeteering.pdf
Valve-Developer-Recording-Gameplay.pdf
Valve-Developer-The-presets-panel.pdf
Valve-Developer-Working-with-IK-Rigging.pdf
Valve-Tutorials-Youtube.pdf
Witcher-WeddingDance.pdf
Youtube.pdf . . . . . .

Quellenverzeichnis

Literatur

[1] Andy Beane. *3D Animation Essentials*. Indianapolis: John Wiley & Sons, 2012 (siehe S. 3, 5).

[2] Ollie Johnston Frank Thomas. *Disney Animation: The Illusion of Life*. New York: Abbeville Press, 1981 (siehe S. 5, 6).

[3] Jason Gregory. *Game Engine Architecture*. Wellesley: A K Peters/CRC Press, 2009 (siehe S. 19).

[4] M Mori. „Bukimi no tani [the Uncanny Valley]“. *Energy* 7 (Jan. 1970), S. 33–35 (siehe S. 11).

Audiovisuelle Medien

[5] *Fountains of Wayne — Stacy's mom (Sims Version)*. Aug. 2007. URL: https://www.youtube.com/watch?v=r4YjSs2zBNA (besucht am 04. 09. 2019) (siehe S. 8).

[6] *König der Löwen*. Film. Regie: Roger Allers, Rob Minkoff. Drehbuch: Irene Mecchi, Jonathan Roberts, Linda Woolverton. Prod.: Don Hahn. 1994 (siehe S. 3).

[7] *Meet the Heavy*. Prod.: Valve Corporation. Jan. 2009. URL: https://youtu.be/jHgZh4GV9G0?list=PLHy7G7ndrUmpWqBkNKjJRT5urGiPW63Iq (besucht am 15. 12. 2019) (siehe S. 17).

[8] *Meet the Medic*. Prod.: Valve Corporation. Juni 2011. URL: https://youtu.be/36lSzUMBJnc?list=PLHy7G7ndrUmpWqBkNKjJRT5urGiPW63Iq (besucht am 15. 12. 2019) (siehe S. 16–18).

[9] *Meet the Scout*. Prod.: Valve Corporation. Jan. 2009. URL: https://youtu.be/geNMz0J9TEQ?list=PLHy7G7ndrUmpWqBkNKjJRT5urGiPW63Iq (besucht am 15. 12. 2019) (siehe S. 17).

[10] *Nightmare Before Christmas*. Film. Regie: Henry Selick. Drehbuch: Caroline Thompson, Michael McDowell (nach einer Geschichte von Tim Burton). Prod.: Tim Burton, Denise Di Novi. 1993 (siehe S. 3).

[11] Neth Nom. *SONDER*. Nov. 2018. URL: https://vimeo.com/253571683 (besucht am 31. 12. 2019) (siehe S. 1).

[12] Nick Park. *Wallace and Gromit – Welt der Erfindungen*. DVD. Regie: Merlin Crossingham. Studio: Concorde Video. Sep. 2011 (siehe S. 3).

[13] *Schneewittchen und die sieben Zwerge*. Film. Regie: David D. Hand. Drehbuch: Ted Sears, Richard Creedon. Prod.: Walt Disney. 1937 (siehe S. 3).

[14] Simon J. Smith. *Baymax Dreams*. Jan. 2019. URL: https://www.youtube.com/watch?v=Q2XEyCFAMuk (besucht am 15. 01. 2020) (siehe S. 1).

[15] Unity Technologies. *ADAM: Episode 1*. Regie: Veselin Efremov. Juni 2016. URL: https://www.youtube.com/watch?v=GXI0l3yqBrA (besucht am 31. 12. 2019) (siehe S. 5).

[16] *Witcher 3: HEARTS OF STONE — Geralt Dances like a Pro with Shani #11*. Screencapture aus The Witcher3: Hearts of Stone von CD Projekt RED. Okt. 2015. URL: https://www.youtube.com/watch?v=MlZlVGpfHPQ (besucht am 10. 12. 2019) (siehe S. 12–14).

Software

[17] John Cook und Robin Walker. *Team Fortress 2*. Windows, Mac OS X, Linux, Xbox 360, PlayStation 3. 2007 (siehe S. 9).

[18] Ion Hazzikostas, Alex Afrasiabi und Chris Robinson. *World of Warcraft: Battle for Azeroth*. Windows, Mac OS X. 2018 (siehe S. 14).

[19] Jason Jones. *Halo: Combat Evolved*. Windows, Mac OS X, Xbox. 2001 (siehe S. 7).

[20] Jeff Kaplan und Tom Chilton. *World of Warcraft: Wrath of the Lich King*. Windows, Mac OS X. 2008 (siehe S. 14).

[21] Jeff Kaplan, Chris Methen und Aaron Keller. *Overwatch*. Windows, PlayStation4, Xbox One, Nintendo Switch. 2016 (siehe S. 2).

[22] Jeff Kaplan, Rob Pardo und Tom Chilton. *World of Warcraft*. Windows, Mac OS X. 2004 (siehe S. 2, 8, 14).

[23] Bruce Straley und Neil Druckmann. *Uncharted 4: A Thief's End*. PlayStation 4. 2016 (siehe S. 11).

[24] Konrad Tomaszkiewicz, Mateusz Kanik und Sebastian Stępień. *The Witcher 3: Wild Hunt*. Windows, PlayStation 4, Xbox One, Nintendo Switch. 2015 (siehe S. 2, 3, 12).

[25] Will Wright. *The Sims*. Windows, Mac OS X, Linux, PlayStation 2, GameCube, Xbox. 2000 (siehe S. 7).

Online-Quellen

[26] *180 Grad Regel – Filmproduktion – Film-Connexion*. Datei auf CD-ROM (Datei 180Grad-Regel.pdf). URL: https://www.film-connexion.de/filmlexikon/180-grad-regel/ (besucht am 19. 12. 2019) (siehe S. 12).

[27] *Behind the Scenes of the Cinematic Dialogues in The Witcher 3: Wild Hunt*. 2017. URL: https://www.youtube.com/watch?v=chf3REzAjgI (siehe S. 12–14).

[28] *BlizzCon 2016 In-Game Cinematics*. Transkript des Panels. 2016. URL: http://warcraft.blizzplanet.com/blog/comments/blizzcon-2016-game-cinematics-cutscenes-legion-panel-transcript (siehe S. 15, 16).

[29] *CD Projekt Red*. URL: https://en.cdprojektred.com (besucht am 05. 09. 2019) (siehe S. 12).

[30] *Disney*. URL: https://www.disney.com/ (besucht am 29. 01. 2019) (siehe S. 1).

[31] *Euler (gimbal lock) Explained — YouTube*. Jan. 2009. URL: https://youtu.be/zc8b2Jo7mno?t=281 (besucht am 30. 12. 2019) (siehe S. 28).

[32] *Eulerwinkel – Comupteranimation*. URL: http://www.informatikseite.de/animation/node16.php (besucht am 30. 12. 2019) (siehe S. 28).

[33] *Game Engines –– How do they work?* URL: https://unity3d.com/what-is-a-game-engine (besucht am 29. 01. 2019) (siehe S. 1).

[34] *HYpER AS fUCK RN SOMEONE HALP ...* URL: https://picsart.com/i/image-hyper-as-fuck-rn-someone-halp-manga-mangagirl-185164024001202 (besucht am 03. 01. 2020) (siehe S. 31).

[35] *Installation & First Steps — UMotion Official Tutorials - YouTube*. Sep. 2017. URL: https://www.youtube.com/watch?v=0QZBYNFyqLQ (besucht am 30. 12. 2019) (siehe S. 27).

[36] *Lau — Black Butler Wiki*. URL: https://black-butler.fandom.com/de/wiki/Lau (besucht am 03. 01. 2020) (siehe S. 31).

[37] *LIVE FACE*. URL: https://apps.apple.com/us/app/live-face/id1357551209 (besucht am 01. 09. 2019) (siehe S. 6).

[38] *Manipulating objects overview — Valve Developer Community*. URL: https://developer.valvesoftware.com/wiki/SFM/Manipulating_objects_overview (besucht am 10. 12. 2019) (siehe S. 9).

[39] *Meet the Team*. URL: https://www.youtube.com/playlist?list=PLHy7G7ndrUmpWqBkNKjJRT5urGiPW63Iq (besucht am 15. 12. 2019) (siehe S. 16).

[40] *Mixamo*. URL: https://www.mixamo.com/ (besucht am 29. 01. 2019) (siehe S. 6).

[41] *Movies — Official TF2 Wiki*. URL: https://wiki.teamfortress.com/wiki/Movies (besucht am 13. 12. 2019) (siehe S. 16).

[42] *Naughty Dog*. URL: https://www.naughtydog.com (besucht am 02. 09. 2019) (siehe S. 11).

[43] Krasimir Nechevski. *Adam — Animation for the real-time short film*. Aug. 2016. URL: https://blogs.unity3d.com/2016/08/31/adam-animation-for-the-real-time-short-film/ (siehe S. 5).

[44] *Official site of Aardman Animations*. URL: https://www.aardman.com (besucht am 01. 09. 2019) (siehe S. 3).

[45] *Puppeteering — Valve Developer Community*. URL: https://developer.valvesoftware.com/wiki/SFM/Puppeteering (besucht am 10. 12. 2019) (siehe S. 10).

[46] *Recording gameplay — Valve Developer Community*. URL: https://developer.valvesoftware.com/wiki/SFM/Recording_gameplay (besucht am 10. 12. 2019) (siehe S. 10).

[47] *Rooster Teeth*. URL: https://roosterteeth.com (besucht am 04. 09. 2019) (siehe S. 7).

[48] *Second Life*. URL: https://www.secondlife.com (besucht am 04. 09. 2019) (siehe S. 7, 8).

[49] *Source Filmmaker bei Steam*. URL: https://store.steampowered.com/app/1840/Source_Filmmaker/ (besucht am 07. 12. 2019) (siehe S. 10).

[50] *Source Filmmaker FAQ*. URL: http://www.sourcefilmmaker.com/faq/ (besucht am 10. 12. 2019) (siehe S. 9).

[51] *Steam Community – Steam Workshop*. URL: https://steamcommunity.com/workshop/ (besucht am 07. 12. 2019) (siehe S. 9).

[52] Rooster Teeth. *Red vs. Blue*. Apr. 2003. URL: https://roosterteeth.com/series/red-vs-blue (besucht am 04. 09. 2019) (siehe S. 7).

[53] *The Presets panel and the Controls panel — Valve Developer Community*. URL: https://developer.valvesoftware.com/wiki/SFM/The_Presets_panel_and_the_Controls_panel (besucht am 10. 12. 2019) (siehe S. 9, 10).

[54] *The Stormwind Extraction*. Aug. 2018. URL: https://www.youtube.com/watch?v=8nbnmRwd9x0 (besucht am 09. 09. 2019) (siehe S. 14).

[55] *The Wrathgate*. Mai 2011. URL: https://www.youtube.com/watch?v=Ch4rc5W4dKY (besucht am 09. 09. 2019) (siehe S. 14).

[56] *Uncharted 4 — AnimationReel Breakdown — Richard Pince*. Juli 2016. URL: https://vimeo.com/173688684 (besucht am 05. 09. 2019) (siehe S. 12).

[57] *Unity — Manual: Animation Window*. URL: https://docs.unity3d.com/Manual/AnimationEditorGuide.html (besucht am 07. 06. 2019) (siehe S. 23, 24).

[58] *Unity — Manual: Blend Trees*. URL: https://docs.unity3d.com/Manual/class-BlendTree.html (besucht am 06. 06. 2019) (siehe S. 20).

[59] *Unity — Manual: Rig Tab*. URL: https://docs.unity3d.com/Manual/FBXImporter-Rig.html (besucht am 07. 06. 2019) (siehe S. 19).

[60] *Unity — Manual: State Machine Basics*. URL: https://docs.unity3d.com/Manual/StateMachineBasics.html (besucht am 06. 06. 2019) (siehe S. 20, 21).

[61] *Unity — Manual: Timeline*. URL: https://docs.unity3d.com/Manual/TimelineSection.html (besucht am 06. 06. 2019) (siehe S. 21–23).

[62] *Unity — Manual: Track List*. URL: https://docs.unity3d.com/Manual/TimelineTrackList.html (besucht am 06. 06. 2019) (siehe S. 22).

[63] *Unity — Manual: Using Animation Curves*. URL: https://docs.unity3d.com/Manual/animeditor-AnimationCurves.html (besucht am 07. 06. 2019) (siehe S. 23, 24).

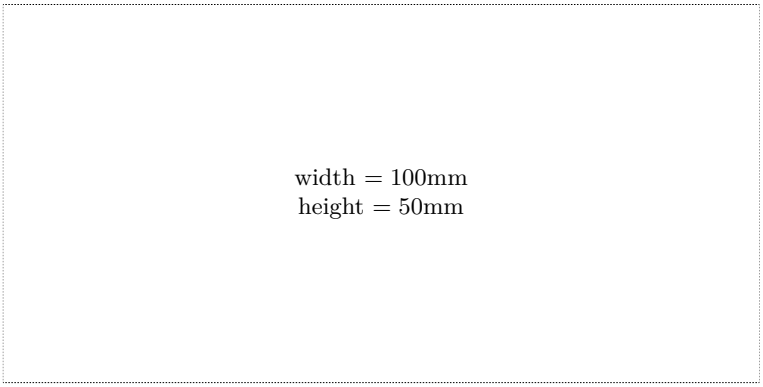
[64] *Valve Corporation*. URL: https://www.valvesoftware.com/de/ (besucht am 06. 09. 2019) (siehe S. 9, 14).

[65] *Valve Tutorials – Youtube*. URL: https://www.youtube.com/playlist?list=PL2B46DEB4157E67C4 (besucht am 13. 12. 2019) (siehe S. 16).

[66] *Working with IK rigging — Valve Developer Community*. URL: https://developer.valvesoftware.com/wiki/SFM/Working_with_IK_rigging (besucht am 10. 12. 2019) (siehe S. 10).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —