5 Die Spielentwicklung vorbereiten

Bevor wir unseren Prototyp weiter ausbauen und ihn dann zu einem richtigen Spiel weiterentwickeln können, gehen wir also einen Schritt zur Seite und verschaffen uns einen Überblick, was »Spielentwicklung« eigentlich genau umfasst. Tatsächlich umfasst diese Tätigkeit nämlich eine Vielzahl teilweise völlig unterschiedlicher Disziplinen: Game-Design, Programmierung, Grafikdesign, 3D-Modelling und -Animation, Musikkomposition und -produktion sowie Sounddesign, um nur die offensichtlichsten zu erwähnen.

Einerseits macht das die Spielentwicklung sehr interessant. Andererseits stellt es uns als Spielentwickler aber auch vor besondere Herausforderungen – vor allem, wenn wir versuchen, ein komplettes Spiel allein zu entwickeln. Neben den Talenten in den verschiedenen Disziplinen brauchen wir auch eine Vielzahl unterschiedlicher Werkzeuge, mit denen wir diese Talente auf eine Weise zum Ausdruck bringen können, die dann auch in Spielen verwendbar ist.

Der Auswahl der Werkzeuge sollten wir in jedem Fall ausreichend Zeit widmen, da die Entscheidung für ein bestimmtes Werkzeug neben den meistens anfallenden Lizenzkosten vor allem oft einen erheblichen Einarbeitungsaufwand mit sich bringt, der je nach Wert der eigenen Zeit die Lizenzkosten häufig um ein Vielfaches übersteigt. Hinzu kommt, dass Werkzeuge in der gleichen Gattung oft nur teilweise zueinander kompatibel sind, was den Wechsel zu einem späteren Zeitpunkt weiter erschwert. Daher ist es auch empfehlenswert, schon bei der Auswahl der Werkzeuge auf die Kompatibilität mit konkurrierenden Werkzeugen zu achten.

Dieses Kapitel soll Ihnen zunächst einen oberflächlichen, aber umfassenden Eindruck von den verschiedenen Aufgaben in der Spielentwicklung vermitteln. Zu jedem Bereich werden dann die Klassen von Werkzeugen sowie konkrete Beispiele kurz vorgestellt. Damit erhalten Sie auch Einstiegspunkte für eigene tiefergehende Recherchen.

Link auf unity-buch.de

Da es zu jedem Abschnitt in diesem Buch gleich eine Vielzahl von Links auf der Website zum Buch gibt, verwende ich hier nur Fußnoten mit entsprechendem Vermerk, ohne zusätzlich am Seitenrand auf die Links aufmerksam zu machen.

5.1 Regeln: Spielmechanik und Programmierung

Zunächst hat jedes Spiel eine *Spielmechanik*, also eine Menge von Regeln, die bestimmen, welche Aktionen des Spielers welche Konsequenzen innerhalb der Spielwelt haben, die aber durchaus auch die Eigendynamik einer Spielwelt beschreiben können. Dazu gehört selbstverständlich auch, wie das Spiel beginnt und endet sowie unter welchen Umständen der Spieler das Spiel gewinnt oder verliert. Zunächst müssen diese Spielmechaniken definiert werden, was einen wesentlichen Teil des Design-Dokuments ausmacht. Zur Erstellung des Design-Dokuments gibt es verschiedenste Vorlagen und sogar einige spezielle Tools. Man kann aber auch einfach eine beliebige Textverarbeitung verwenden.

Zur konkreten Umsetzung der Spielmechaniken verwenden wir immer irgendeine Art von Programmierung, und dazu brauchen wir eine Programmiersprache. Idealerweise kann man die gleiche Sprache dann auch für alle anderen Programmieraufgaben verwenden, die bei der Spielentwicklung anfallen, egal ob man nun die Logik der Benutzerschnittstelle umsetzt, die Kommunikation mit einem Webserver implementiert, um Highscores abzuspeichern und für alle Spieler verfügbar zu machen, oder ob man gar eine Netzwerkschicht für Multiplayer-Spiele entwickelt.

Auch wenn man bei dem Stichwort »Programmierung« meistens an textuellen Programmcode denkt, gibt es durchaus visuelle Programmiersprachen – auch im Unity-Umfeld.[1] Wir beschränken uns hier aber auf die Sprachen, die Unity direkt mitliefert, also *C#, JavaScript* bzw. *UnityScript* und *Boo*.

Auswahl der Programmiersprache

Die Sprache *Boo* ist eine Programmiersprache mit Python-ähnlicher Syntax und daher sicherlich für all diejenigen besonders interessant, die bereits Erfahrung mit Python haben. Inzwischen kann Boo auch für Android- und iOS-Spiele eingesetzt werden (das war am Anfang nicht so), führt aber dennoch eher ein Schattendasein.[2]

1 Im Unity Asset Store gibt es dazu eine eigene Kategorie: *Editor Extensions/Visual Scripting*.

2 Link auf *unity-buch.de:* Falls Sie Boo lernen möchten, hilft Ihnen vielleicht das Thema *Official Boo Scripting Resource Thread* im Unity-Forum, das ich von der Website zum Buch aus verlinkt habe.

JavaScript in Unity sollte eigentlich eher *UnityScript* heißen, und tatsächlich heißt die Sprache auch so,[3] wurde aber lange Zeit von Unity Technologies offiziell als JavaScript bezeichnet. Es handelt sich auch um eine zumindest von der Syntax her an JavaScript angelehnte Sprache, die vor allem Programmierern aus dem Webbereich einen einfachen Zugang in die Programmierung in Unity ermöglichen soll. Da hat die durchaus vorhandene Ähnlichkeit mit dem bekannten JavaScript aus der Webentwicklung und ActionScript aus der Flash-Entwicklung natürlich Vorteile. Ein weiterer Vorteil von JavaScript in Unity ist eine im Vergleich zu C# kompaktere Schreibweise, da beispielsweise auf die Klassendeklaration und Angabe der Vererbung von `MonoBehaviour` verzichtet werden kann. Auch einige Namespaces stehen hier automatisch zur Verfügung und müssen nicht explizit angegeben werden. Genau diese kompaktere Schreibweise ist jedoch aus meiner Perspektive auch der größte Nachteil.

Wenn man nämlich in Unity einsteigt und noch nicht mit Unity vertraut ist, gibt JavaScript-Code dem Einsteiger meistens relativ wenige Anhaltspunkte, um zu verstehen, welche Klassen genau verwendet werden. Mit dieser Information könnte man deutlich leichter in der *Scripting API* nachschlagen, um seine Kenntnisse zu vertiefen.

Ein weiteres Problem für den Programmieranfänger ist, dass JavaScript in Unity eben nicht wirklich JavaScript ist. Daraus resultiert, dass die meisten JavaScript-Tutorials eher Verwirrung stiften, weil sie sich auf die üblicherweise auf Webseiten verwendete Version von JavaScript beziehen, die zwar ähnlich ist, aber doch verschieden. Ist man aber bereits mit JavaScript aus dem Webbereich (oder ActionScript) und mit Unity vertraut, ist JavaScript eine gute Wahl.[4] Früher war ein großer Teil der Standard Assets in JavaScript implementiert, und auch viele Beispiele in der Dokumentation gab es nur in JavaScript. Das hat sich in den letzten Jahren aber zugunsten von C# verändert.[5]

Bei *C#* steht in Unity im Gegensatz zu JavaScript/UnityScript nicht nur C# drauf, sondern es ist auch wirklich C# drin, und zwar genau so, wie man es aus der.NET- und Mono-Welt kennt. Somit kann man sich auch für Unity auf die offizielle Sprachspezifikation beziehen und findet auch ein entsprechendes Programmierhandbuch.[6] Außerdem gibt es eine Vielzahl

3 Der Name »UnityScript« wird für die Projekte auf GitHub und *code.google.com* verwendet, und man findet auch innerhalb des mit Unity installierten Mono-Frameworks die Dateien *UnityScript.dll* und *UnityScript.Lang.dll*.

4 Link auf *unity-buch.de:* Im *Unify Community Wiki* gibt es einen Artikel, der die beiden Dialekte UnityScript (also JavaScript in Unity) und JavaScript (also den Web-Dialekt) direkt gegenüberstellt und vergleicht. Für den Einstieg dürfte dieser Artikel sehr hilfreich sein. Er ist unter dem Titel *UnityScript versus JavaScript* verlinkt.

5 Link auf *unity-buch.de:* Dazu gab es am 03.09.2014 auch ein offizielles Statement von Unity Technologies: *Documentation, Unity scripting languages and you*

6 Link auf *unity-buch.de:* Siehe auch den Link *C#-Sprachspezifikation*. Von dort aus sind das Programmierhandbuch und die Referenz verlinkt.

von Tutorials und anderen Möglichkeiten, die Sprache zu lernen; teilweise auch im Unity-Umfeld.[7]

Obwohl man früher teilweise den Eindruck gewinnen konnte, dass JavaScript die »Unity-Sprache« ist, hat schon 2012 eine informelle Umfrage im Unity-Forum ergeben, dass C# bei den Forumsteilnehmern tatsächlich etwas populärer ist als JavaScript. Von den 482 Teilnehmern an der Umfrage[8] haben 34,4 % angegeben, dass sie ausschließlich C# verwenden, 29,8 %, dass sie ausschließlich JavaScript verwenden, und 28,4 %, dass sie sowohl JavaScript als auch C# verwenden. Es ist zwar möglich, beide Sprachen in einem Projekt zu verwenden, dabei muss man aber aufgrund der Kompilierung in mehreren Schritten aufpassen, welche Klassen von C# aus auf JavaScript-Klassen zugreifen können und anders herum.[9]

Die Popularität der Sprache ist insofern relevant, als man bei einer populäreren Sprache Fragen leichter beantwortet bekommt und man auch leichter Team-Kollegen findet, wenn man die Programmierung nicht komplett allein erledigen möchte. Falls Sie die Programmiersprache Java kennen, ist der Umstieg auf C# auch sehr einfach, da die Sprachen sich sehr ähnlich sind.[10]

Diese Vorteile und das bei JavaScript höhere Risiko, Code zu produzieren, der zwar problemlos kompiliert, aber zur Laufzeit Probleme bereitet, haben mich dazu bewogen, selbst ausschließlich C# zu verwenden und auch als Implementierungssprache in diesem Buch durchgängig C# zu nutzen.

Auswahl der Programmierumgebung

Theoretisch könnte man auch heute noch mit einem einfachen Texteditor den Programmcode schreiben. Puristen mögen mir verzeihen, aber das halte ich für keine gute Idee: Moderne Entwicklungsumgebungen vereinfachen und beschleunigen den Prozess unter anderem durch Hilfsmittel, wie die automatische Vervollständigung des Codes[11], die sofortige Markierung von Syntax-Fehlern und die elegante Einbindung der API-Dokumentation. Eine solche Entwicklungsumgebung, die bei Unity mitgeliefert wird, haben wir bereits kennengelernt. Sie ist jedoch eine eigenständige und von Unity unabhängige Anwendung: *MonoDevelop*.[12]

---

7 Link auf *unity-buch.de:* Zum Beispiel das *CSharp Unity Tutorial* im Unify Community Wiki.

8 Link auf *unity-buch.de:* Siehe auch *Boo, C# and JavaScript in Unity – Experiences and Opinions*. Abgerufen am 21.01.2012.

9 Link auf *unity-buch.de:* Siehe *Special Folders and Script Compilation Order*.

10 Interessanterweise sind C# und Java sich viel ähnlicher als JavaScript und Java.

11 »Code completion« wäre hier auch in einem deutschen Umfeld der gängigere Begriff; IntelliSense ist Microsofts Variante davon.

12 Allerdings ist die mit Unity ausgelieferte Version von MonoDevelop speziell für Unity angepasst.

MonoDevelop hat einige Vorteile. Es unterstützt alle von Unity unterstützten Programmiersprachen und Plattformen. Man kann mit MonoDevelop also Scripts in C#, JavaScript und Boo bearbeiten, und zwar sowohl unter Mac OS X als auch unter Windows.[13]

Ich persönlich bevorzuge – obwohl ich selbst an einem Mac arbeite[14] – *Microsoft Visual Studio*. Die wesentlichen Nachteile sind hier, dass die Verwendung auf *Mac OS* etwas umständlich ist und Visual Studio zwar JavaScript unterstützt, aber nicht den in Unity verwendeten Dialekt; von Boo ganz zu schweigen. Das spielt aber natürlich nur dann eine Rolle, wenn man auch mit diesem Betriebssystem oder diesen Sprachen arbeiten möchte. Mit Visual Studio Express gibt es zwar eine kostenfreie Lizenz, diese ist aber im Vergleich zu den kostenpflichtigen Versionen deutlich eingeschränkt.

Der wesentliche Vorteil von Visual Studio besteht darin, dass dieses Werkzeug schon seit vielen Jahren entwickelt und konsequent verbessert wird und auch durch Plug-ins von Drittherstellern erweiterbar ist. Bei MonoDevelop hat man demgegenüber oft das Gefühl, noch mit Kinderkrankheiten konfrontiert zu werden – was aber natürlich mit der Zeit ein immer kleineres Problem wird. Praktischerweise erzeugt Unity für MonoDevelop auch mit Visual Studio kompatible *Solutions*.[15] Das heißt, zumindest technisch spricht nichts dagegen, beide Entwicklungsumgebungen parallel zu verwenden.

Falls Sie doch lieber einen reinen Texteditor verwenden, wäre *Sublime Text* vielleicht eine interessante Alternative. Da gibt es noch viele Editoren mehr, und Sie können natürlich auch Ihren Lieblingseditor weiterverwenden. Letztlich müssen wir nur einen Code-Editor bzw. eine Entwicklungsumgebung installieren und verwenden.

So ausgerüstet, können wir dann schon die meisten Herausforderungen bei der Programmierung angehen.[16] Würden Computerspiele allerdings nur aus der programmierbaren Logik ihrer Spielmechaniken bestehen, wären sie wohl kaum so populär, wie sie es tatsächlich sind. Tatsächlich sprechen

13 Außerdem läuft MonoDevelop unter Linux – tatsächlich lief es dort sogar zuerst und wird dort am besten unterstützt. Da aber Unity nicht unter Linux läuft, ist das in diesem Kontext nur eine Fußnote wert.

14 Ich verwende VMware Fusion, um Visual Studio als Windows-Anwendung neben meinen Mac-Anwendungen laufen zu lassen. Wie man das einrichtet, ist im Unify Community Wiki beschrieben: *Setting up Visual Studio for Unity on Mac,* verlinkt von *http://unity-buch.de*.

15 Eine Solution in Visual Studio oder auch in MonoDevelop sind mehrere Projekte, die in einer übergreifenden Datei zusammengefasst sind. Sie wird von Unity erzeugt, wenn Sie auf eine Scriptdatei doppelklicken oder auch auf eine Konsolenmeldung, die sich auf ein Script bezieht – dabei wird dann normalerweise auch MonoDevelop gestartet, außer Sie haben eine andere Entwicklungsumgebung in den Preferences eingestellt. Sie können aber auch über das Menü *Assets/Synchronize MonoDevelop Project* die Solution erzeugen. Sie wird dann auch bei jeder Änderung automatisch synchronisiert.

16 Für Softwaredesign bzw. Softwarearchitektur gibt es außerdem noch UML-Tools, und bei größeren Spielprojekten würde ich so etwas auch verwenden.

Computerspiele auf direkte Art und Weise die Sinne Hören und Sehen an, auf indirekte Weise über die Erfahrung auch das Fühlen (üblicherweise natürlich nicht im Sinne von haptischer Wahrnehmung, sondern im Sinne von Gefühlen und Emotionen wie Freude oder Trauer).

5.2 Spiel an Augen: Bildschirmdarstellung

Was wir am Bildschirm sehen, lässt sich zunächst in Spielwelt und virtuelle Benutzerschnittstelle unterteilen.[17]

5.2.1 Virtuelle Benutzerschnittstelle: Pixel- und Vektorgrafik

Die virtuelle Benutzerschnittstelle umfasst Bedienelemente (wie beispielsweise Buttons, die der Spieler anklicken kann) sowie die Darstellung von Aspekten des Spielzustands (wie z. B. die Punktezahl oder Lebensenergie des Spielers). Üblicherweise bieten Spiele auch ein *Hauptmenü,* aus dem heraus das Spiel gestartet werden kann und über das ggf. auch Einstellungen vorgenommen werden können. Um die virtuelle Benutzerschnittstelle umzusetzen, benötigen wir Grafiken und meistens auch Texte, die über die Verwendung von Schriftarten letztlich auch in Grafiken umgewandelt werden. Generell sind grafische Darstellungen, z. B. Icons und Fortschrittsbalken, nach Möglichkeit gegenüber Texten zu bevorzugen, weil sie weniger Aufmerksamkeit zum Verständnis erfordern und somit weniger vom eigentlichen Spielgeschehen ablenken. Außerdem sind Symbole meistens international verständlich und müssen nicht internationalisiert werden. (Mehr zu diesem Thema folgt in Abschnitt *8.1.2, Über Internationalisierung und Lokalisierung.*)

Bei der Gestaltung von Benutzerschnittstellen ist es oft vorteilhaft, zunächst die Anordnung der verschiedenen Bedienelemente abstrakt vorzunehmen. Diesen Prozess bezeichnet man als *Screendesign*. Zu diesem Zweck gibt es verschiedene Werkzeuge. Ich habe sehr gute Erfahrungen mit *Balsamiq Mockups* gemacht.

Grafiken können prinzipiell pixelbasiert oder vektorbasiert sein. Vektorgrafik hat den Vorteil beliebiger Skalierbarkeit und ist dabei meistens auch speichereffizienter – wird aber von Unity derzeit zumindest direkt noch nicht unterstützt.[18] Zur Erstellung von Pixelgrafiken verwendet man Bildbearbeitungsprogramme.[19] Kostenfrei und auf allen Plattformen verfügbar ist

17 Vgl. Schell, Jesse: The Art of Game Design, 2008. S. 223 ff.

18 Link auf *unity-buch.de:* Eine erwähnenswerte Erweiterung, die über den Unity Asset Store verfügbar ist, ist *RageSpline* von Juhakiili Oy. Eine komplett vektorbasierte GUI-Lösung wäre *NoesisGUI*.

19 Link auf *unity-buch.de:* Eine Auswahl von Vektorgrafik-Programmen finden Sie auf der Website zum Buch.

Gimp. *Pixelmator* – eine echte Mac-Anwendung – ist zwar nicht kostenfrei, aber auch recht erschwinglich. Der große (und teure) Platzhirsch ist in diesem Bereich nach wie vor *Adobe Photoshop,* und bei der Verwendung mit Unity hat Photoshop vor allem den Vorteil, dass das Dateiformat PSD direkt in Unity unterstützt wird.[20] Photoshop gibt es für Windows und Mac OS X.

5.2.2 Spielwelt: 2D, 3D, Modelling, Texturing und Animation

Während die virtuelle Benutzerschnittstelle zumindest im Prinzip bei allen Spielen ähnlich ist, gibt es völlig unterschiedliche Spielwelten: Vergleichen Sie nur beispielsweise 2D-Spiele mit einer eher abstrakten Welt, die auf einen Bildschirm passt (z. B. Pac Man, Tetris oder auch Schach), mit 2D-Spielen mit größeren Spielwelten, die Scrolling benötigen (z. B. Super Mario Bros), und diese wiederum mit 3D-Spielen, bei denen die Spielwelten zumindest vom räumlichen Empfinden her unserer physischen Umgebung entsprechen. Je nach Art der Spielwelt kommen auch völlig unterschiedliche Technologien zum Einsatz, um die jeweilige Spielwelt zu erschaffen.

2D-Spielwelten – Bitmaps oder Vektorgrafiken

Wie wir bereits gesehen haben, unterstützt Unity auch die Entwicklung von 2D-Spielen. Dazu brauchen wir Bitmap-Grafiken, die wir in Unity als Sprites verwenden können. Im Hinblick auf die Wiederverwendbarkeit dieser Assets kann es sinnvoll sein, vektorbasierte Tools zu verwenden und die entsprechenden Grafiken dann lediglich für Unity zu rastern (also aus der Vektorgrafik eine Bitmap mit einer spezifischen Auflösung zu erstellen).

Neben den bereits behandelten Bildbearbeitungsprogrammen gibt es auch spezielle Werkzeuge, mit denen Sprites und Sprite-Animationen komfortabel erstellt werden können.[21]

3D-Spielwelten – Modelling, Texturing und Animation

Ursprünglich war Unity ja eine 3D-Game-Engine, und von daher werden wir mit Unity auch oft Spiele entwickeln, die über 3D-Spielwelten verfügen. Eine sehr einfache und kleine 3D-Spielwelt haben wir ja sogar schon direkt in Unity erschaffen – und sind dabei aber recht schnell an die Grenzen dessen gestoßen, was mit Unity möglich ist. Hierzu ist es wichtig zu verstehen, dass Unity in keiner Weise als Modelling-Tool gedacht ist. Die *Scene View* und *Hierarchy View* sind eher als Level-Editor zu verstehen, mit dem

20 Prinzipiell unterstützen auch andere Bildbearbeitungsprogramme (wie Gimp oder Pixelmator) das PSD-Format. Da es sich aber um ein proprietäres Format von Adobe handelt, ist die Unterstützung hier nie optimal.

21 Link auf *unity-buch.de:* Einen kurzen Einstieg in diese Thematik sowie Beispiele für Tools finden Sie in dem Artikel *2D Game Sprite and Animation Software*.

man in anderen Tools erstellte Assets platzieren kann. Eine nennenswerte Ausnahme ist die in Unity integrierte Terrain Engine, mit der man Landschaften modellieren kann. Sie wird in Kapitel *15, Terrain Engine: Eine Landschaft bauen,* behandelt. Um herauszufinden, welche Werkzeuge wir für die Erstellung von 3D-Spielwelten brauchen, sollten wir uns erst einmal klar machen, was eine Spielwelt in einem 3D-Spiel eigentlich ausmacht.

Begriffe aus der Welt der 3D-Spiele

Grundsätzlich besteht eine 3D-Welt aus dreidimensionalen Formen, die wir als *3D-Modelle* bzw. verkürzt einfach nur als *Modelle* bezeichnen. Auch die Bezeichnung *Mesh* haben Sie dafür schon kennengelernt. Diese Modelle haben Oberflächen, deren Eigenschaften über *Materialien* definiert sind, die in den allermeisten Fällen über farbige Oberflächenstrukturen verfügen, die wir als *Texturen* bezeichnen.

Wie diese Texturen – bei denen es sich letztlich um einfache Pixelgrafiken handelt – auf die Oberfläche der Modelle gelegt werden, bestimmt die sogenannte *UV-Map,* die einfach gesagt die 2D-Koordinaten in der Textur-Grafik auf die 3D-Koordinaten des Modells abbildet. Neben den Farben für die Oberfläche gibt es auch verschiedene andere Eigenschaften der Oberflächen, die über Texturen definiert werden können. Beispielsweise gibt es *Specular Maps,* bei denen jeder Pixel in der Textur bestimmt, wie stark das Licht an dieser Stelle auf dem Modell reflektiert wird, oder *Bump Maps,* mit denen über die Textur feine Erhöhungen und Vertiefungen auf das eigentliche Modell gelegt werden können, um beispielsweise eine Riffelung darzustellen, ohne die einzelnen Riffeln modellieren zu müssen. Im Kontext von *physikbasierten Shadern* gibt es auch Texturen, die bestimmen, wie sehr die Oberfläche an bestimmten Stellen eher einem Metall oder einem Nichtleiter ähnelt oder wie glatt oder rau die Oberfläche ist – oder auch, wie sehr die Oberfläche selbst Licht abgibt.

Wie die verschiedenen Texturen eingesetzt werden können, bestimmen die schon oben erwähnten *Materialien*[22], die Sie ja auch schon als Eigenschaft der `MeshRenderer`-Komponente in Unity kennengelernt haben und die ihrerseits wiederum durch Shader ihre Eigenschaften erhalten. Über Materialien bzw. Shader wird auch bestimmt, wie sich Licht auf die Modelle auswirkt: Die Modelle können damit transparent gemacht werden, und sogar beliebige Deformationen sind möglich. *Bump Maps* sind hier ein einfaches Beispiel. Häufiger kommen *Normal-Maps* zum Einsatz, bei denen die Farbwerte R, G, B als Richtungsvektoren für die *Normale*[23] an dieser Stelle der Oberfläche interpretiert werden.

22 Siehe Abschnitt *6.1, Eigene Materialien erstellen und verwenden*, ab Seite 129.

23 Die Normale bzw. der Normalenvektor ist der Vektor, der von einer Fläche (wenn wir im dreidimensionalen Raum sind) orthogonal »nach außen« zeigt.

Eine spezielle Art von Texturen sind sogenannte *Skyboxes*. In Unity handelt es sich dabei üblicherweise um sechs Texturen für die verschiedenen Richtungen (vorne, hinten, rechts, links, oben und unten), die praktisch von innen auf einen immer maximal weit entfernten Würfel projiziert werden. Seit Unity 5.0 können Skyboxes auch in verschiedenen für Skyboxes üblichen Formaten (Cubic, Cylindrical, Spheremap) direkt in Unity importiert werden. Damit kann ressourcensparend der Himmel oder auch ein Gebirge am Horizont dargestellt werden, ohne dass man diese tatsächlich als 3D-Modelle in die Spielwelt einfügen müsste. Seit Unity 5 können Skyboxes auch im Kontext von *Global Illumination* dazu verwendet werden, die Szene realistisch auszuleuchten.

Mit den soeben im Schnelldurchlauf zusammengefassten Konzepten könnten wir eine statische Spielwelt erstellen, in der sich nichts bewegt. Das wäre für ein Spiel aber ziemlich langweilig. Bewegung kommt durch *Animation* ins Spiel – und das ist ein weiteres weites Feld, das hier nur kurz angerissen werden kann: Bereits gesehen haben Sie, dass sich Objekte als solche in der Spielwelt bewegen können – wie der Tracer in unserem Prototyp. Diese Art von Animation kann also durch ein wenig Scripting und die Verwendung der Physik-Engine bereits leicht in Unity umgesetzt werden. Sie haben in Abschnitt *2.3.4, Einfache 2D-Animationen mit Mecanim und Dope Sheet erstellen,* auch schon den in Unity eingebauten Animationseditor kennengelernt, mit dem einfache Animationen direkt in Unity umgesetzt werden können. Sie können damit sogar Animationen aus externen Tools bearbeiten. Später, in Abschnitt *11.2, Das Startmenü aufhübschen,* werden Sie dieses Unity-Tool noch besser kennenlernen.

Möchten wir aber menschen- oder tierähnliche Charaktere in unserer Spielwelt auftreten lassen, so brauchen wir *Character Animation,* also die Möglichkeit, diese Gestalten mit Persönlichkeit auf realistische Art und Weise zu bewegen. Ein wesentlicher Arbeitsschritt hierzu ist das *Rigging,* bei dem ein bestehendes Modell, beispielsweise von einem Menschen, um ein Skelett erweitert wird, das bestimmt, wie die verschiedenen Gliedmaßen sich zueinander bewegen können. Damit können zu dem Modell *Animationen* erzeugt werden, also Bewegungsabläufe wie Gehen, Springen oder Tanzen. Häufig werden solche Bewegungsabläufe über *Keyframe Animation* (zu Deutsch: Schlüsselbildanimation) erfasst, d. h., verschiedene Schlüsselstellungen werden nacheinander gespeichert und die Bewegung dazwischen wird dann interpoliert. Eine andere Möglichkeit, die Bewegungsabläufe direkt aus der realen Welt zu erfassen, ist *Motion Capture*. Dabei werden je nach Verfahren die Bewegungen entweder mit mehreren Kameras erfasst und durch die Verwendung von Markern die Berechnung der Bewegung im Raum ermöglicht, aus der dann wiederum eine Animation erstellt werden kann; oder die Bewegungen werden durch Verwendung spezieller Anzüge direkt erfasst. Mit *Mecanim* können dann wiederum in Unity verschiedene

solche Animationen in komplexen Zustandsgraphen organisiert und anhand beliebiger Parameter ineinander überblendet werden.

Character Animation geht schließlich bis hin zur *Gesichtsanimation* (Facial Animation), die die Darstellung von Mimik und lippensynchronisierter Sprache ermöglicht. Zur Gesichtsanimation kann einerseits ebenfalls eine Art Skelett verwendet werden, ein sogenanntes *Facial Rig,* häufig werden aber auch sogenannte *Morph Targets* oder *Blend Shapes* verwendet. Dabei werden direkt im Modell verschiedene Versionen gespeichert, beispielsweise das Gesicht mit einer hochgezogenen Augenbraue oder der Mund in der Stellung bei dem Phonem »O«. Diese werden seit Unity 4.3 sogar nativ unterstützt, also ohne dass dazu zusätzliche Add-ons notwendig wären.[24]

Werkzeuge zur Erstellung von 3D-Spielwelten

Es gibt eine Vielzahl von Modelling-Tools, die zur Erstellung von Modellen und Animationen für Unity verwendet werden können. Ein kostenfreies und dennoch sehr mächtiges Werkzeug ist *Blender,* zu dem es auch ein empfehlenswertes Buch im dpunkt.verlag gibt.[25] Am anderen Ende der Preisskala befinden sich Tools wie *Maya* und *3D Studio Max,* die auch in vielen großen Spielproduktionen zum Einsatz kommen. Für Mac-User, die in die Welt des Modelling einsteigen, ist *Cheetah 3D* zu empfehlen, da es gleichzeitig einen günstigen Preis hat und relativ einfach zu erlernen ist. Ein Werkzeug, das speziell auf das Erstellen von Character-Animationen zugeschnitten ist, wäre *MotionBuilder*. Eine relativ erschwingliche Möglichkeit, unter Verwendung von zwei Microsoft-Kinects Animationen mittels Motion Capturing zu erstellen, ist *iPi Desktop Motion Capture*. Gesichtsanimation mit Lippensynchronisation zu Audiodateien ist beispielsweise mit *FaceFX* möglich.

Einige Modelling-Tools (z. B. auch *modo*) ermöglichen es, Texturen direkt auf das 3D-Modell zu »malen«. Texturen können aber natürlich auch in Bildbearbeitungsprogrammen erstellt werden, und häufig werden Bildbearbeitungsprogramme verwendet, um Texturen aus verschiedenen Quellen zu optimieren oder Fehler darin zu korrigieren. Als Grundlage für Texturen kann man – je nach visuellem Stil der 3D-Spielwelt – auch Fotos verwenden. Wenn sich die Textur wiederholen soll, z. B. für Kacheln oder eine Wand aus Ziegeln, ist es wichtig, dass die Ränder sich nahtlos aneinanderfügen (Stichwort: *Seamless Textures*). Dazu gibt es unter anderem eine prozedurale Textur (*Substance*) von Allegorithmic: *Bitmap2material*. Unity unterstützt Substances, und wenn Sie diese selbst erstellen wollen, können Sie dazu *Substance Designer* verwenden. So ist es prinzipiell möglich, auch ohne Bildbearbeitungsprogramm an hochwertige Texturen zu kommen, die perfekt an die jeweilige Spielwelt angepasst werden können und zusätzlich

24 Link auf *unity-buch.de: Animation Blend Shapes* im Unity Manual.
25 Wartmann, Carsten: Das Blender-Buch, 5. Auflage, 2014.

auch noch deutlich weniger Speicherplatz verbrauchen. Ein relativ neues Tool ist der *Substance Painter,* mit dem Substances und andere Materialien direkt auf ein 3D-Modell aufgetragen werden können. Hier können mit einer Partikel-Engine auch Abnutzungseffekte sehr einfach simuliert werden. Außerdem unterstützt Substance Painter die verschiedenen Workflows für physikbasiertes Shading sehr gut.

Skyboxes kann man theoretisch auch mit einem Fotoapparat erstellen. Einfacher ist aber auch hier die Verwendung von Software. Zwei Werkzeuge, die sehr gut zum Berechnen von Skyboxes für typische Himmel geeignet sind, wären *Terragen 2* und *Vue*. Wer Skyboxes für Weltraumspiele selbst erzeugen möchte, findet in *Spacescape* einen verlässlichen Verbündeten.

5.3 Spiel an Ohren: Musik und Soundeffekte

Die Atmosphäre in einem Spiel wird vor allem durch musikalische Untermalung und Soundeffekte erzeugt. Daher ist der Audiobereich keinesfalls weniger wichtig als der visuelle Bereich oder die Programmierung. Jesse Schell, ein bekannter Game-Designer und Buchautor, berichtet sogar von einer Studie, in der Spieler zwei Versionen des gleichen Spiels bezüglich der visuellen Qualität beurteilen sollten. Der einzige Unterschied zwischen den beiden Versionen war, dass die eine über bessere Audioqualität verfügte als die andere. Interessanterweise bewerteten die Spieler die visuelle Qualität entsprechend der Audioqualität. Das heißt, das Spiel mit dem schlechteren Ton »sah« für die Spieler schlechter aus. Daher empfiehlt Jesse Schell sogar, die Musik für das Spiel so früh wie möglich im Entwicklungsprozess auszuwählen, idealerweise sogar ganz am Anfang.[26] Dieser Empfehlung schließe ich mich voll und ganz an.

Im Prinzip ist Audio in Computerspielen vergleichbar mit Audio in Film und Fernsehen, und Sie können den Effekt von Audio leicht selbst erfahren, indem Sie sich verschiedene herausragende Filmszenen einmal ohne Ton ansehen und danach mit Ton. Am besten geeignet für solch ein Experiment sind natürlich Action-Szenen sowie Szenen, in denen Spannung aufgebaut wird – aber auch eine romantische Szene wird ohne Ton in den allermeisten Fällen deutlich flacher wirken als mit Ton. Über den Hörsinn erreichen wir mit Spielen also auch das Fühlen – zumindest bezogen auf Gefühle und Emotionen. Den Tastsinn erreicht man nur, wenn der Spieler Musik und Effekte extrem laut aufdreht. ;-)

Ein wesentlicher Unterschied zwischen Audio in Film und Fernsehen gegenüber Audio in Computerspielen ist, dass Spiele interaktiv sind und daher die Ereignisse nicht oder nur in gewissem Rahmen vorhersehbar sind. Wenn beispielsweise in einer Spielszene durch musikalische Untermalung

26 Vgl. Schell, Jesse: The Art of Game Design, 2008. S. 351 f.

Spannung aufgebaut werden soll, hängt es meistens von Spieler-Aktionen ab, zu welchem Zeitpunkt diese Szene losgeht und wann sie endet.[27] Daher ist es bei Spielen wünschenswert, dass die Musik sich flexibel an die Spielsituation anpassen kann, was Komponisten und Musikproduzenten natürlich vor interessante Herausforderungen stellt.

Soundeffekte gibt es in Spielen überall dort, wo sie auch in Film und Fernsehen vorkommen: Ob es Schritte sind (die sich je nach Untergrund, Schuhwerk und Gangart anders anhören können), Explosionen und Schüsse, die Umgebungsgeräusche in einem Wald oder in einer Stadt – alles, was man am Bildschirm sieht und wovon der Spieler erwarten würde, dass es ein Geräusch macht, sollte auch ein Geräusch machen.

In Computerspielen gibt es aber noch weitere Ereignisse, die durch Soundeffekte wirkungsvoller dargestellt werden können: Ob das das Klicken auf einen Button ist, das Tippen eines Textes auf der Tastatur oder das Scrollen eines User-Interface-Elements – alles wirkt mit Ton dramatischer und wird den Spieler eher in die Spielwelt einbinden, wenn es auch entsprechend vertont ist.

Technisch kann man Audioeffekte unterteilen in andauernde Geräusche, wie z. B. ein Meeresrauschen, und solche, die einmal abgespielt werden, z. B. ein Schuss. Für andauernde Geräusche ist wichtig, dass diese ohne Knackser oder sonstige Unregelmäßigkeiten wiederholt werden können (Stichworte: *Loop* bzw. *Looping*).

Werkzeuge zur Musikproduktion für Spiele

Zur Produktion von Musik für Spiele können typische *DAWs* (*Digital Audio Workstations*) verwendet werden. Das sind Programme, die den vollständigen Produktionsprozess von Komposition bis Mastering unterstützen und somit auch als virtuelles Musikstudio bezeichnet werden könnten. Zwei Beispiele hierfür sind *Cubase* und *Logic Pro*. Beide unterstützen auch sogenannte *VST*-(*Virtual Studio Technology*-)Instrumente und -Effekte. Also vereinfacht gesagt Synthesizer, Sampler und Effektgeräte, die als Plug-ins auf dem Rechner laufen. Natürlich kann man auch Hardware-Synthesizer, -Sampler und -Effektgeräte verwenden. Dann benötigt man aber zusätzlich ein Mischpult, um die Audiosignale der verschiedenen Geräte wieder zusammenzumischen, und ein entsprechendes Audio-Interface, um die Musik wieder zurück in den Computer zu bekommen.

Unity bietet seit der Version 3.0 auch die Unterstützung von sogenannten *Tracker-Formaten*. Dabei handelt es sich um eine Kombination aus Audiosamples, die als Instrumente dienen, und aus der Sequenz, also der Abfolge, in der diese Samples in verschiedenen Tonhöhen abgespielt werden. Auf

27 Ein kleiner Trick, der in manchen Spielen funktioniert und dieses Problem elegant löst, besteht darin, die spannende Situation zeitlich zu beschränken: Wenn die Zeit abläuft, ist die Situation zu Ende. Dann kennen wir den Zeitpunkt im Voraus.

diese Art und Weise lassen sich ebenfalls Musikstücke in hoher Qualität produzieren, die aber nicht nur einen deutlich geringeren Speicherplatzverbrauch haben, was z. B. für Webspiele und auf mobilen Geräten eine Rolle spielt, sondern die prinzipiell auch zur Laufzeit, also während der Spieler spielt, geändert werden können. Somit können sich Musikstücke in Tracker-Formaten dynamisch der Spielsituation anpassen. Unity bietet dazu leider noch keine entsprechende API an. Musikstücke in Tracker-Formaten können Sie beispielsweise mit *MilkyTracker* (Mac OS X und Windows) oder *OpenMPT* (nur Windows) erstellen.

Seit Unity 5 gibt es in Unity einen mächtigen Audiomixer, der auch Effekte unterstützt. Den lernen Sie in Abschitt 8.4.6, *Audio-Engineering: Musik und Effekte aufeinander abstimmen* kennen.

Werkzeuge zum Erstellen von Audioeffekten

Eine naheliegende Möglichkeit, Audioeffekte zu erzeugen, besteht darin, die Geräusche mit einem Mikrofon aufzunehmen und über ein entsprechendes Audio-Interface in den Computer zu übertragen. Die so erzeugten Audiodateien können Sie dann mit einem beliebigen Audioeditor bearbeiten, z. B. mit *Audacity* (kostenfrei, Mac OS X und Windows) oder *WaveLab* (kommerziell, ebenfalls Mac OS X und Windows).

Alternativ können Sie auch die verschiedensten elektronischen Klangerzeuger (wie Synthesizer und VST-Instrumente) verwenden, um die Klänge ganz nach Ihren Vorstellungen zu designen. Zwei nennenswerte Hersteller von VST-Instrumenten und Effekten sind *Steinberg* und *Native Instruments*. Falls Sie Retro-Spiele mit dem typischen »8-Bit-Sound« entwickeln wollen, ist hierzu auch das bereits erwähnte *SFXR* verwendbar.

Falls Ihnen die Komposition von Musik oder das Sounddesign nicht liegt oder Sie nicht die entsprechenden Fähigkeiten mitbringen, können Sie natürlich auch bestehende Musik- oder Effektbibliotheken verwenden. Und diese Möglichkeit gibt es natürlich auch für 2D- und 3D-Assets und sogar – zumindest in einem gewissen Rahmen – für Scripts. Auf eine sehr einfache Möglichkeit, sich der Talente anderer zu bedienen, oder auch die eigenen Talente anderen anzubieten gehe ich im nächsten Abschnitt ein.

5.4 Teamwork für Einzelspieler: Unity Asset Store

Es liegt nahe, dass ein einzelner Entwickler kaum in allen Disziplinen der Spielentwicklung die entsprechenden Talente mitbringt. Und selbst in einem kleinen Team kann möglicherweise nicht jeder Bereich abgedeckt werden. Diese Problematik hat Unity Technologies erkannt und daher den *Unity Asset Store* eingeführt, den ich bereits mehrfach erwähnt habe. Die Idee ist dabei einerseits, jedem Unity-Spielentwickler eine Vielzahl von Assets

aus allen Bereichen zur Verfügung zu stellen, sodass man nicht jedes einzelne Asset selbst erstellen muss, das man zur Umsetzung seiner Spielidee braucht. Andererseits bietet der Asset Store natürlich denjenigen, die ihre Assets anderen anbieten möchten, einen lukrativen Marktplatz.

Auch bei über 20.000 Paketen im Januar 2015 ist es zwar nicht unbedingt so, dass man für jeden Bedarf etwas findet, aber zumindest findet man für praktisch alle Bereiche zumindest eine Arbeitsgrundlage: Ob das nun Texturen sind, Modelle, um einen Raum zu füllen, animierte Charaktere, verschiedenste Musikstücke zur stimmungsvollen Untermalung oder Soundeffekte. Außerdem gibt es eine Vielzahl von Erweiterungen des Unity Editors, die nicht nur die tägliche Arbeit mit Unity erleichtern, sondern teilweise Dinge ermöglichen, die man sonst erst mit hohem Aufwand selbst entwickeln müsste.

Natürlich ist der Asset Store nicht die einzige Quelle für Game Assets. Aber da die Pakete erst nach einer Prüfung durch Mitarbeiter von Unity Technologies veröffentlicht werden, entsprechen die Assets einem gewissen Standard. Und da es der Unity Asset Store ist, sind sie speziell für die Unity Engine optimiert und funktionieren dort auch reibungslos.

Gleiches gilt natürlich auch für die Assets, die wir im Folgenden zur Entwicklung unseres Buch-Spiels verwenden. Es würde den Rahmen sprengen, hier die konkrete Bedienung der vielfältigen Werkzeuge zu beschreiben oder gar die Künste lehren zu wollen, denen diese Werkzeuge dienen. Auf dieser Reise kann ich Sie leider nicht begleiten.

Daher biete ich im Folgenden die jeweils benötigten Assets zum Download an und beschränke mich auf deren Anwendung in Unity. Dies soll den geneigten und entsprechend talentierten und mit den richtigen Werkzeugen ausgestatteten Leser natürlich nicht davon abhalten, seine ganz eigene Version des Spiels mit komplett eigenen Assets umzusetzen.

5.5 Klassisches Teamwork oder Backup: Versionsverwaltung

Sind Sie in der glücklichen Situation, in einem kleinen Team arbeiten zu können, in dem die verschiedenen zur Spielentwicklung notwendigen Talente vereint sind? Dann stellt sich natürlich die Frage, wie mehrere Leute möglichst reibungslos und am besten gleichzeitig an einem Projekt arbeiten können. Die Standardantwort darauf kommt aus der Softwareentwicklung und heißt *Versionsverwaltung*. Und das ist gleichzeitig durchaus auch für Einzelentwickler eine sehr nützliche Sache, wenn auch aus anderen Gründen:

Die von einer Versionsverwaltung gehaltene Versionshistorie ermöglicht es nämlich, jede wesentliche Veränderung auch im Nachhinein zu überprüfen und rückgängig zu machen, wenn das mal notwendig wird – beispiels-

weise, wenn man feststellt, dass eine Funktionalität, die früher einwandfrei gearbeitet hat, nach verschiedenen Änderungen plötzlich nicht mehr das tut, was man von ihr erwartet.

Vereinfacht gesagt bestehen Systeme zur Versionsverwaltung aus einem zentralen Repository, in dem der aktuelle Projektstand sowie die gesamte Versionshistorie abgelegt sind, so wie einer Software, die den Zugriff auf dieses Repository ermöglicht. Dabei arbeitet man mit einer lokalen *Arbeitskopie* des Projekts, die sich vom aktuellen Stand des Repository unterscheidet, aber mit verschiedenen Aktionen synchronisiert werden kann: Jederzeit können Änderungen von anderen Entwicklern im Repository auf die aktuelle Arbeitskopie übertragen werden. Dieser Vorgang wird meistens mit *Checkout, Aktualisieren* oder *Get Latest Version* bezeichnet. Und Änderungen in der lokalen Arbeitskopie können – wenn sie fertig sind – jederzeit in das Repository eingespielt werden, was meist mit *Checkin* bezeichnet wird.

Beispiele für konkrete Lösungen zur Versionsverwaltung mit direkter Integration in Unity wären der *Unity Asset Server, Perforce* und *Plastic SCM*. Weitere Lösungen wären z. B. *Microsoft Team Foundation Server, Subversion, Bazaar* oder *Git*.[28]

Eine wichtige Besonderheit für die Versionsverwaltung im Zusammenhang mit Unity ist, dass wir es hier häufig mit vielen, teilweise sehr großen Binärdateien zu tun haben, was im Softwareentwicklungsalltag, für den die meisten Versionsverwaltungen ursprünglich entwickelt wurden, eher untypisch ist.[29] Mit den direkt in Unity integrierten oder spezifisch für die Spielentwicklung entworfenen Lösungen sollte es damit aber keine Schwierigkeiten geben.

Prinzipiell würde ich die Verwendung einer geeigneten Versionsverwaltung auf jeden Fall empfehlen, und falls Sie im Team arbeiten und ihr Spiel dennoch ohne Versionsverwaltung entwickeln wollen, müssen Sie sich sehr eng koordinieren und die Arbeitsabläufe entsprechend koordinieren. Leider würden alle dazugehörigen Details aber hier den Rahmen sprengen. Daher belasse ich es bei dieser Empfehlung. Denn: Ein unvollständiger Prototyp wartet darauf, zu einem vollwertigen Spiel zu werden – gehen wir es an!

28 Link auf *unity-buch.de:* Details zum Einrichten der verschiedenen Versionsverwaltungen würden hier den Rahmen sprengen, sind aber von *http://unity-buch.de* aus verlinkt: *Version Control Integration* und *Using External Version Control Systems with Unity*.

29 Link auf *unity-buch.de:* Zu diesem Thema gibt es auch Fragen mit entsprechenden Antworten im Internet, z. B. *Is there a distributed VCS that can manage large files?*