UNIVERSITY OF APPLIED SCIENCES
HOCHSCHULE MITTWEIDA

Angewandte Computer- und Biowissenschaften

Professur Medieninformatik

Bachelorarbeit

Visualisierung eines Verfahrens zur prozeduralen Generierung von Assets in der Virtuellen Realität

Noah Stolz

Mittweida, den 3. September 2019

Erstprüfer:	Prof. Dr. rer. nat. Marc Ritter
Zweitprüfer:	Sebastian Schleier, B.Sc.
Co-Betreuerin:	Rama Hasan, M.Sc.

Stolz, Noah
Visualisierung eines Verfahrens zur prozeduralen Generierung von Assets in der Virtuellen Realität
Bachelorarbeit, Angewandte Computer- und Biowissenschaften
Hochschule Mittweida– University of Applied Sciences, September 2019

Referat

In dieser Arbeit wird ein Verfahren zur prozeduralen Generierung von 3D-Modellen von Netzwerken visualisiert. Interaktion mit dieser Visualisierung ist innerhalb der virtuellen Realität ermöglicht. Abschließend wird die Effektivität dieser Visualisierung beim Vermitteln von Lernmaterial evaluiert.

Name: Stolz, Noah
Studiengang: Medieninformatik und interaktives Entertainment
Seminargruppe: Mi16w2-B
English Title: Visualisation of a procedure for generating assets in virtual reality

Inhaltsverzeichnis

1. Einführung und Motivation

Das Erstellen von digitalen Inhalten ist in seinem Umfang in den letzten Jahren stark gewachsen, vor allem im Entertainment-Bereich, aber auch, wenn es um die verständliche Darstellung von Informationen geht. [KQM13, S.20] Diese Inhalte zu erstellen, benötigt professionelle 3D-Artists und Designer sowie lange Arbeitszyklen und finanzielle Mittel. Prozedurale Generierung stellt eine Alternative zu dem herkömmlichen Prozess zum Erstellen von Assets dar. Hier werden Assets durch Algorithmen generiert. Dies erlaubt das Erstellen von mehr und komplexeren Assets als bei einer rein intellektuellen Generierung. [TYSB11, S.172] Die auf diese Art generierten Assets können dann entweder direkt in einer Anwendung verwendet werden oder die Basis für die Arbeit eines Designers darstellen.

1.1. Problembeschreibung

Während die Ergebnisse von prozeduraler Generierung enorme Ausmaße annehmen können, wie zum Beispiel das Videospiel "No Mans Sky"[1], für welches eine Galaxie mit Millionen von Planeten generiert wurde, ist es oft nicht wirklich ersichtlich, nach welchen Regeln diese Assets erstellt wurden. Dies kann das Erlernen und Lehren der Methoden der prozeduralen Generierung erschweren, da nur durch ein langes Auseinandersetzen mit den zugrundeliegenden Prinzipien, der Weg zum jeweiligen Endergebnis verstanden werden kann. Auch für Menschen, die prozedurale Generierung bereits nutzten, ist die Nachvollziehbarkeit, gerade bei besonders komplexen Algorithmen, nicht gegeben. [TCL+13, S.67f]

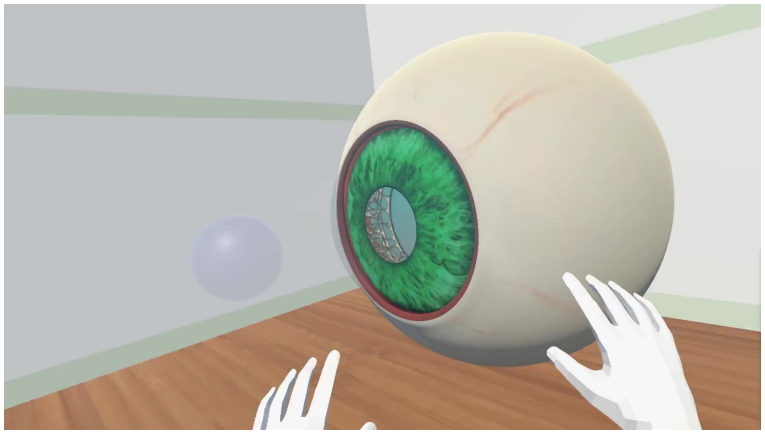


Abbildung 1.1.: Das Augenmodell der Augenblick Anwendung

1.2. Studentische Vorarbeit

Die Augenblick-Anwendung ermöglicht es dem Nutzer, das Modell eines menschlichen Auges in der virtuellen Realität zu betrachten sowie zahlreiche Optionen für Interaktion mit diesem. Die Anwendung wurde im vierten Semester des Medieninformatik Studiengangs an der Hochschule Mittweida unter Leitung von Dipl.-Inf. Holger Langner und Sebastian Schleier B. Sc. im Modul "Wissenschaft und Wirtschaft 3" von einer Gruppe von sieben Studenten erstellt. Im darauf folgenden Semester wurde sie im Modul "Wissenschaft und Wirtschaft 4" weiter ausgebaut. Beide Module wurde geleitet von Prof. Dr. rer. nat. Mark Ritter und Prof. Dr.-Ing. Wilfried Schubert. Ziel der Anwendung ist es, vor allem dem Nutzer Informationen zum Auge und zu Krankheiten, welche die Funktionsweise dessen beeinträchtigen können, zu vermitteln. So ist zum Beispiel möglich, für den Nutzer einen gewissen Dioptrien-Wert einzustellen und einerseits eine Verformung des Auges zu beobachten und gleichzeitig - wenn gewünscht - eine Simulation der Auswirkung dieses Wertes auf das Sehen zu erleben. Die Anwendung versucht dabei nicht, auf einem medizinischen Level von Beratung zu informieren, sondern will vor allem für Laien eine anschauliche und intuitive Darstellung der grundlegenden Prinzipien geben.

Die simpelste Interaktion mit dem Augenmodell der Anwendung stellt das Bewegen dessen einzelner Komponenten dar. Der Nutzer kann hierzu innerhalb der Anwendung seine Hand in die Komponente bewegen und anschließend durch Input des jeweiligen Controllers die Komponente greifen. Die Komponente folgt dann solange

[1] https://www.nomanssky.com/

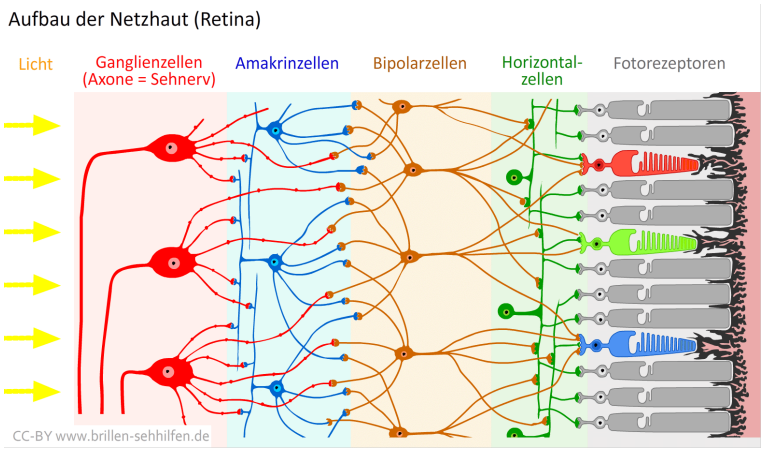


Abbildung 1.2.: Das Schema, welches als Vorlage zum Generieren des Assets der Vorarbeit diente.

der Hand, bis der Trigger wieder losgelassen wird, und bleibt dann an ihrer momentanen Position stehen. Um diese Interaktion vollständig auszunutzen, besteht das Augenmodell aus vielen Komponenten, die nicht nur die Oberfläche, sondern auch das Innere des Auges, darstellen. Außerdem kann zwischen der „Voll-Ansicht“, bei der das Auge im Gesamten zu betrachten ist, und der „Halb-Ansicht“, bei der eine Hälfte des Auges fehlt, um einen besseren Zugriff auf das Innere des Auges zu erlauben, gewechselt werden.

Im Praxismodul ”Erweiterung einer bestehenden Anwendung zur 3D-Visualisierung eines Modells des menschlichen Auges in der virtuellen Realität durch Generierung von prozeduralen Assets sowie Ausbau der Interaktionsmöglichkeiten” an der Professur Medieninformatik wurde die Anwendung unter der Leitung von Prof. Mark Ritter und Sebastian Schleier B. Sc. weiterentwickelt. In diesem Modul hat der Autor ohne das Team der vorherigen Semester über zwölf Wochen die Anwendung in verschiedenen Bereichen, wie Steuerung und neue Komponenten, verbessert. Für diese Arbeit relevant ist dabei die prozedurale Generierung eines Assets, welches ein Modell des Zellnetzwerks innerhalb der Netzhaut abbilden soll. Das Endergebnis dieses Prozesses war ein Asset, welches vier verschiedene Schichten von Zellen, die sogenannten Stäbchen und Zapfen sowie die Pigmentschicht, die diese umgibt, beinhaltet. Die Schichten unterscheiden sich dabei nicht nur in der Form, sondern sind auch farblich getrennt. Das Asset wurde ausschließlich mit prozeduraler Generierung erstellt. Diese wurde hier gewählt, da sie sich sehr gut zum Erstellen von großen, aus vielen ähnlichen Bestandteilen zusammengesetzten, Konstruktionen eignet.

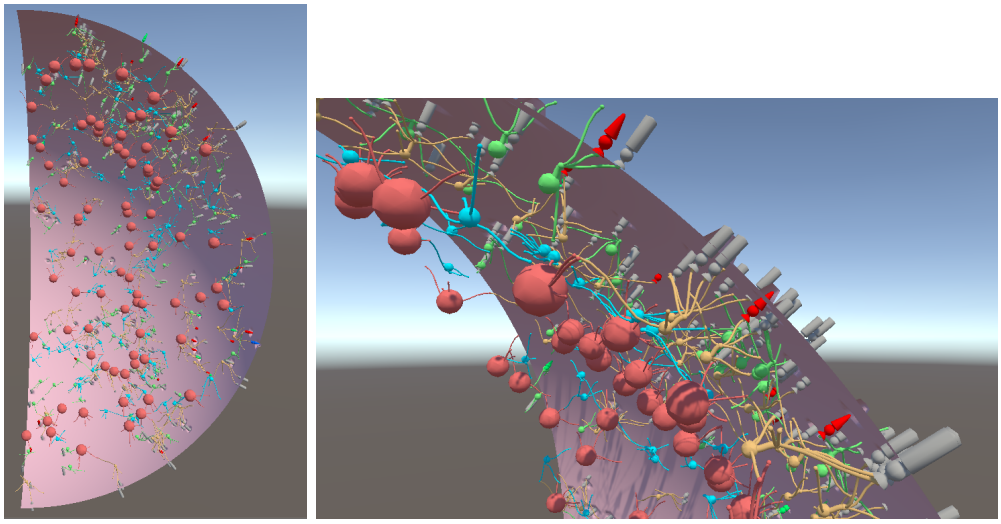


Abbildung 1.3.: Links: Ein in Rahmen der Vorarbeit mit prozeduraler Generierung erstelltes Asset, Rechts: Nähere Ansicht einer anderen Version des Assets

1.3. Zielstellung

In der Film- sowie in der Videospiel-Industrie sind prozedurale Verfahren in den letzten Jahren immer mehr zum Standard geworden. Dies erlaubt das Erstellen von größeren und komplexeren Assets mit weniger Arbeit und verringert den Anteil an sich wiederholenden Arbeitsschritten. Dabei ist die Entstehung der Assets im Nachhinein schwer nachvollziehbar. Im Rahmen dieser Bachelorarbeit soll eine Anwendung entstehen, welche ein solches Verfahren in der virtuellen Realität veranschaulicht.

In der Anwendung wird nach einem prozeduralen Verfahren ein 3D-Asset erstellt. Dieses stellt ein Netzwerk aus miteinander verknüpften Netzhautzellen dar. Die Anwendung ermöglicht das sukzessive Abhandeln sowie das Beobachten der Funktionsweise des Algorithmus. Durch die Modifizierung von beschreibenden Parametern können Ergebnis und Visualisierung des Verfahrens angepasst werden. Letztendlich soll dem Nutzer die Möglichkeit gegeben werden, die Visualisierung vorwärts und rückwärts schrittweise zu durchlaufen. Der Lerneffekt der Visualisierung wird zudem mit einer textbasierten Methode verglichen und evaluiert.

1.4. Kapitelübersicht

Kapitel 1 gibt einen Einblick in die Motivation dieser Arbeit sowie die Vorarbeit, auf welcher diese aufbaut. Außerdem beleuchtet sie die Themen, mit welchen sich dieses Werk auseinander setzt.

Im darauf folgenden **Kapitel 2** werden die Grundlagen in Form der verwendeten Technologien und Prinzipien sowie deren Relevanz für die erstellte Anwendung näher erläutert.

Nachdem das nötige Vorwissen vermittelt ist, befasst sich **Kapitel 3** mit den Spezifikationen des Projekts sowie mit den zugrundeliegenden Konzepten. Dieses Kapitel beinhaltend auch die Systemarchitektur der Anwendung.

Die in Kapitel 3 aufgestellten Konzepte werden in **Kapitel 4** umgesetzt und aus ihnen eine Anwendung erstellt, deren Funktionsweise dieses Kapitel detailliert ausführt.

Das **Kapitel 5** testet die erstellte Anwendung dann mit mehreren Probanden. Der Aufbau des Versuchs sowie die Erkenntnisse aus diesem sind in diesem Kapitel zu finden.

Die Arbeit wird in **Kapitel 6** zusammengefasst und es werden Verbesserungsvorschläge sowie ein Ausblick für die Möglichkeiten der Arbeit in der Zukunft gegeben.

2. Grundlagen

Die Entwicklung aller Skripte der Anwendung findet innerhalb der Unity Game Engine statt. Die Engine stellt alles, was für das Generieren von Meshes benötigt wird, zur Verfügung und erlaubt es, mithilfe des SteamVR Plugins[1] diese in der virtuellen Realität zu betrachten. Um die virtuelle Realität zu verwenden, wird ein Headset benötigt, in Fall dieser Arbeit die *HTC-Vive*. Die Visualisierung wird ebenfalls innerhalb der Engine erstellt und ihre Ergebnisse im JSON-Format gespeichert.

2.1. Virtuelle Realität

Die virtuelle Realität (*VR*) hebt sich von anderen Medien durch ihren hohen Grad an Immersion hervor. Im Folgenden werden zuerst einige Grundlegende Informationen über *VR* gegeben und darauf folgend wird die *HTC-Vive*, das für diese Arbeit verwendete *VR-System*, näher erläutert.

2.1.1. Übersicht

Dieses Kapitel gibt eine kurze Übersicht über die Funktionsweise moderner *VR-Systeme*, ihrer Anwendungsfelder und die Herausforderungen vor denen die Technologie noch steht.

In *VR* kann der Nutzer sich durch Bewegen seines Kopfes in einer 3D-Umgebung umsehen, welche ihm durch die zwei Bildschirme eines Headsets präsentiert wird, anstatt sie nur durch ein fixiertes Fenster zu betrachten und benutzt einen Controller in jeder Hand, um mit der Umgebung zu interagieren. Die Position des Headsets sowie der Controller wird von mindestens einer Basestation verfolgt, welche das

[1] https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647

Bewegen in der Umgebung erlaubt. Dieser Prozess nennt sich *Tracking*. In anderen Fällen werden Kameras am Headset an Stelle der Basestations verwendet, um die Position des Nutzers zu bestimmen[2]. Manche Controller erlauben es außerdem, die Position der Finger des Nutzers zu verfolgen. Dies erzeugt ein Level von Immersion, welches dem einer traditionellen 3D-Anwendung und den damit verbundenen Input-Geräten deutlich übertrifft. Das *Tracken* der Hände öffnet eine Vielzahl von Interaktionsmöglichkeiten, wie zum Beispiel Objekte zu greifen und zu werfen.

VR hat zwar durch Gaming den Mainstream erreicht, aber die Technologie findet in einer Vielzahl von Bereichen Anwendung. Gerade in der Medizin findet *VR* großen Nutzen, zum Beispiel beim Diagnostizieren von Krankheiten, die Einfluss auf die Bewegung des Genicks haben, was mit dem *Tracking* von *VR* verfolgt werden kann. Auch kann *VR* verwendet werden, um beim Trainieren von medizinischen Verfahren zu helfen, indem zum Beispiel eine Operation abgespielt und mittels der *VR* Technologie aus jedem Winkel betrachtet werden kann. Dadurch kann der Nutzer sich sehr viel eingehender mit dem Ablauf vertraut machen, als durch ein Video oder einen Text. [EGW+17, S.1] Ein weiteres Beispiel für den Nutzen von *VR* liegt in historischen Umgebungen. Ein 3D-Modell dieser Umgebungen kann dadurch nicht nur in einer Simulation betrachtet werden, sondern der Nutzer kann sich direkt in die Umgebung hineinversetzen. [KBT+17, S.403f] Dies kann gerade in Museen, welche oft nur ein passives Erlebnis bieten können, für neue Erfahrungen sorgen. [CB10, S.452ff]

Die *VR* Technologie steht noch vor einigen Herausforderungen, eine der größten davon ist *Motion Sickness*. Diese wird durch sehr schnelle Bewegung oder Bewegung, auf deren Richtung und Geschwindigkeit sich nur schwer vorbereitet werden kann, ausgelöst. *Motion Sickness* kann aber auch durch Simulationen wie *VR* ausgelöst werden und wird in solchen Fällen auch *Cyber Sickness* genannt. Symptome dieser können unter anderem Kopfschmerzen, Übelkeit und Erbrechen sein. [LCC+07, S.3872ff] Dadurch werden Entwickler in ihren Optionen eingeschränkt, auch wenn nicht jeder Nutzer an diesem Problem leidet, da die Nebenwirkungen so dramatisch sein können, dass sie ein Benutzen nicht nur erschweren, sondern nahezu unmöglich machen. Eine weitere Herausforderung ist das Einschätzen von Distanzen, was ein Problem für Aktivitäten darstellt, bei denen der Nutzer nicht still steht, und deshalb die Größe seiner Umgebung einschätzen muss. Detailliertere Darstellungen der 3D-

[2] https://www.oculus.com/rift-s/

Abbildung 2.1.: Ein HTC-Vive Headset mit Controllern und Basestations

Umgebung sowie Headsets von höherer Qualität verringern diesen Effekt. [KCS17, S.2]

2.1.2. HTC Vive

Nintendos 1995 erschienener "Virtual Boy" stellt einen der ersten Versuche dar, *VR* Technologie auf den Mainstream Markt zu bringen, allerdings mit geringem Erfolg [ZZ09, S.99]. Mehr als 20 Jahre später fand *VR* mit der *Oculus Rift*, *PlaystationVR* und der *HTC-Vive* erstmals breiteren Anklang. Diese Systeme wurden primär mit dem Ziel entwickelt, im Gaming-Bereich Verwendung zu finden aber stoßen auf Grund ihrer hohe Leistung und für *VR* geringe Kosten auch in den in Kapitel 2.1.1 erwähnten Bereichen auf Anklang. [MGMADGM17, S.470ff]

Die *Vive* sowie die meisten Anwendungen, die für diese erstellt wurden, nutzen die *SteamVR* Technologie. Diese wurde von Valve entwickelt und nach ihrem Online Gaming Marktplatz Steam benannt. *SteamVR* wird von den meisten für die *Vive* entwickelten Anwendungen genutzt und sorgt dafür, dass Entwickler sich nicht mit den technischen Details der *Vive* beschäftigen müssen. *SteamVR* wird ebenfalls

vom Valve Index *VR-System* verwendet und erlaubt es dadurch, zum Beispiel die Controller der Index mit dem Headset der *Vive* zu verwenden.

Die *Vive*-Controller haben fünf Buttons, um verschiedene Arten von Input zu geben. Die Trigger auf der Rückseite des *Vive*-Controllers werden mit dem Zeigefinger bedient und stellen den Standard-Input dar. An den Seiten des Controllers befinden sich die Grip-Buttons, welche der Nutzer durch Erhöhen des Drucks, mit dem er die Controller greift, aktiviert. Auf der Vorderseite des Controllers befinden sich der Menu-Button, welcher meistens Menüs innerhalb einer Anwendung aufruft, der System-Button, welcher verwendet wird, um das *SteamVR*-Menü zu öffnen, und das Touchpad, welches die Position des Fingers des Nutzers auf ihm verfolgt, wenn er dieses berührt. Das Touchpad besitzt darüber hinaus die Funktionalität eines Buttons.

2.2. Unity

Die 2005 veröffentlichte Unity Gameengine erlaubt es Nutzern 2D-Anwendungen und 3D-Anwendungen zu Erstellen. Gameengines stellen Entwicklern grundlegende Tools zum Erstellen von Videospielen zur Verfügung und stellen die Plattform dar, in welcher die verschiedenen Aspekte einer Anwendung wie 3D-Modelle, Animationen und Skripte zusammenkommen. Als Gameengine sind Videospiele das Hauptziel der Engine, aber sie findet auch Gebrauch für die meisten anderen Arten von 3D-Anwendungen. [Haa14, S.1] Dieses Kapitel gibt erst eine allgemeine Übersicht über die Engine und anschließend über die für diese Arbeit primär relevante Mesh Generierung.

2.2.1. Übersicht

Im Folgenden wird eine Übersicht über die Benutzeroberfläche sowie die für diese Arbeit relevanten Elemente der Anwendung gegeben. Die Informationen entstammen der offiziellen Unity Dokumentation [Uni19a].

Beim ersten Öffnen der Gameengine sieht der Nutzer die Benutzeroberfläche in ihrem Standardlayout, siehe Abbildung 2.2. Die Abbildung entspricht der Version 2018.2.6.1f. In der Mitte der Oberfläche ist die Scene-View zu sehen. Diese zeigt eine

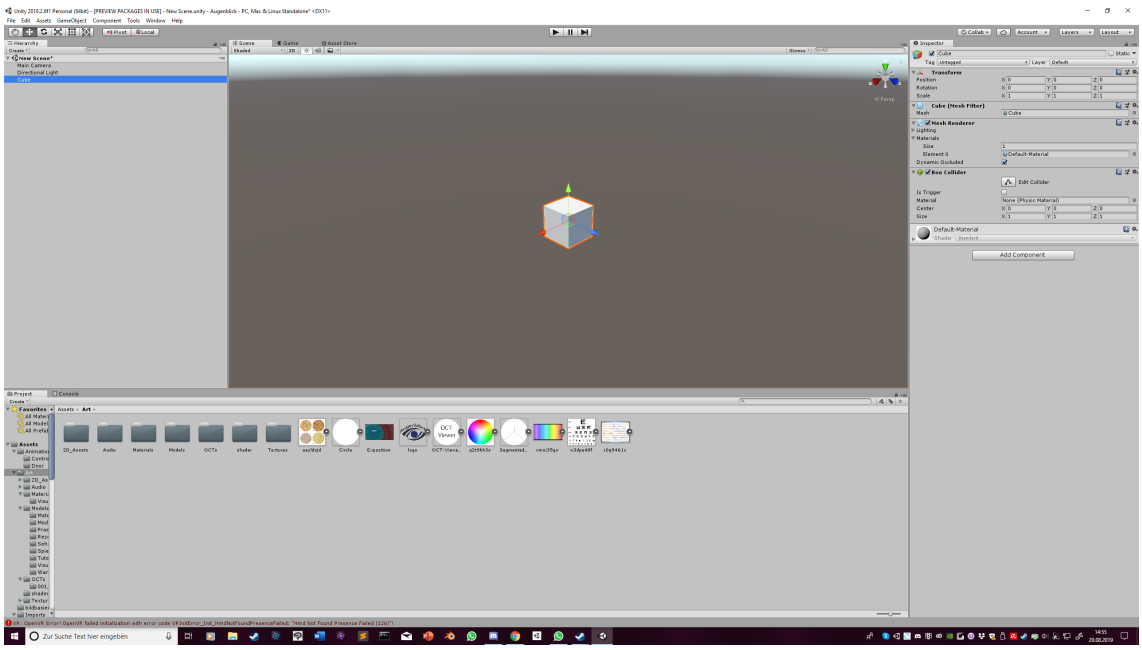


Abbildung 2.2.: Benutzeroberfläche der Unity Game Engine Version 2018.2.6.1f

3D-Umgebung sowie alle Objekte, die vom Nutzer in dieser platziert wurden. Auf der oberen Leiste sind Dropdown Menüs für viele Funktionen wie das Laden, Speichern und Erstellen von Projekten zu finden, wie es für viele Anwendungen typisch ist. Auf der rechten Seite ist die Hierarchie zu sehen, welche alle Gameobjects in der aktuell ausgewählten Scene zeigt. Auf der linken Seite ist der Inspektor zu finden. Dieser enthält Informationen über das selektierte Objekt oder Asset und erlaubt es, diese zu verändern. Der untere Bereich der Oberfläche zeigt den Projekt-Tab und in diesem die Assets, also die Bestandteile wie 3D-Modelle, Audiodateien und Texturen der Anwendung und die Ordner, in denen sich diese befinden. Die Struktur ist hierbei dieselbe wie die des Projekts auf der Festplatte. Sämtliche Bereiche können je nach Bedarf angepasst werden, indem bestehende Fenster verschoben oder auf andere Weise ersetzt werden.

Jede Anwendung verwendet eine oder mehrere Scenes. Jedes Objekt, das während der Laufzeit existieren soll, muss sich in einer Scene befinden. Scenes können meistens als die Abschnitte angesehen werden, die von der Anwendung geladen werden. Ein Dorf könnte beispielsweise eine Scene darstellen und das Innere jedes Hauses eine weitere Scene, welche beim Betreten geladen werden muss. In einem Spiel könnte zum Beispiel das Hauptmenü eine Scene sein und jedes Level eine weitere. Sie dienen dazu,

die verschiedenen Aspekte der Anwendung zu organisieren und diese aufzuteilen, sodass die Rechen- und Speicherkapazität des PCs für eine hinreichend schnelle Verarbeitung ausreicht. Es ist aber auch möglich mehrere Scenes gleichzeitig zu laden oder eine Scene zu laden, bevor die aktuelle Scene verlassen wird.

Objekte der Anwendung, *Gameobjects* genannt, können Assets beinhalten. Dabei sind viele Arten von Assets möglich, beispielsweise 3D-Modelle und selbst geschriebene "Skripte". *Gameobjects* können andere *Gameobjects* als Elternobjekte erhalten, um eine hierarchische Organisation zu erzielen. Darüber hinaus behalten Kindobjekte ihre Position relativ zu ihrem Elternobjekt bei, was es zum Beispiel erlaubt, die Distanzen zwischen einer Vielzahl von Kindobjekten zueinander beizubehalten und sie trotzdem alle um denselben Vector zu bewegen. Welche Objekte Kinder von anderen Objekten darstellen, ist in der Hierarchie der Benutzeroberfläche durch ein Einrücken der Namen der Kinder ablesbar. *Gameobjects* können außerdem zu *Prefabs* konvertiert werden. Diese erlauben es, ein *Gameobject* mit seinem Namen, Kindern und Skripten auf einfache Art zu vervielfältigen. Der große Vorteil eines *Prefabs* ist, dass alle Dublikate dieses in allen Scenes verändert werden können, indem das Original angepasst wird. Um ein *Prefab* zu erstellen, kann der Nutzer es aus der Hierarchie in einen Ordner im Projekt-Tab per Drag and Drop bewegen.

Components geben den *Gameobjects* ihre Funktionalität. Dabei handelt es sich um Scripts, die entweder bereits in der Engine vorhanden sind oder von einem Nutzer geschrieben wurden. Für diese Arbeit relevant sind vor allem die im folgenden beschriebenen *Components*. Die einzige *Component* welche jedes *Gameobject* benötigt, ist eine *Transform*. In diesem ist die Translation, Rotation und Skalierung des jeweiligen *Gameobjects* gespeichert. Außerdem kann sie verwendet werden, um das *Gameobject* sowie dessen Kind- und Elternobjekte zu identifizieren. Die Informationen über das Mesh, also die Geometrie eines 3D-Assets, werden in einer *Meshfilter Component* gespeichert. Das Mesh wird dann durch einen *Meshrenderer* sichtbar gemacht, der festlegt, wie Licht mit dem Objekt interagiert.

2.2.2. Mesh Generierung

Meshes beschreiben die Form eines Objektes. Bei einem Würfel würde das Mesh die acht Ecken sowie die Flächen zwischen diesen beinhalten [Uni19a]. Jede Ecke stellt hierbei ein sogenanntes *Vertex* dar. Die Flächen werden in Dreiecke unterteilt,

da Dreiecke beliebig in einem dreidimensionalen Raum gelegt werden können und immer eine zweidimensionale Fläche bilden.

Um ein Mesh in Unity durch ein Script zu generieren, wird zuerst ein *Gameobject* erstellt. Dieses benötigt einerseits eine Meshfilter-Komponente, welche die Informationen des Meshs beinhaltet und andererseits eine MeshRenderer-Komponente, die für das Zuweisen der Materials zuständig ist, ohne welche das Mesh nicht sichtbar wäre. Im Skript werden dann zwei Listen erstellt, wobei die erste die *Vertices* des Mesh in Form von dreidimensionalen Vektoren beinhaltet und die zweite die Dreiecke des Mesh in Form von Integern darstellt. Die Vektoren der ersten Liste stellen jeweils ein *Vertex* dar und haben ihren Ursprung bei der Position des jeweiligen *Gameobjects*. Die Integer der zweiten Liste stellen jeweils einen Index der *Vertex* Liste dar, wodurch jeweils drei Integer ein Dreieck beschreiben. Die Indices der Dreiecke stellen auch eine der häufigsten Fehlerquellen dar, wenn zum Beispiel beim Erstellen eines Kreises nicht darauf geachtet wird, dass das letzte Dreieck den Kreis schließen muss und deshalb das erste *Vertex* des Kreises verwenden muss. Falls in der Liste der Dreiecke ein Index angesprochen wird, der größer oder gleich der Länge der *Vertex*-Liste ist wird ein Fehler ausgelöst. Der Fokus des Generierens liegt vor allem auf dem Festlegen der Positionen der *Vertices*, sodass sie die gewünschte Form ergeben, und dem korrekten Bilden der Dreiecke, welche die *Vertices* verbinden. Im letzten Schritt werden die Listen dann dem Mesh des Meshfilters zugewiesen.

2.3. Prozedurale Generierung

Die prozedurale Generierung stellt eine Vorgehensweise der Content Generierung dar, bei welcher ein Algorithmus das Erstellen des Contents zumindest teilweise übernimmt. Dabei wird erst ein Verfahren mit Regeln und Parametern definiert, um die gewünschte Art von Content zu generieren. Aufgabe eines Designers ist es dann, das Generieren des Contents durch Anpassen der Parameter zu kontrollieren, anstatt den Content selbst zu erstellen [CLL11, S.395f]. Dabei können vollwertige Inhalte oder aber die Basis für die Arbeit des Designers erstellt werden. Im Folgenden werden die Vor- und Nachteile dieses Ansatzes genauer erläutert.

Eine der frühsten und heute noch viel verwendeten Methoden der prozeduralen Generierung ist Perlin Noise. Es handelt sich dabei um einen effizienten Weg, eine

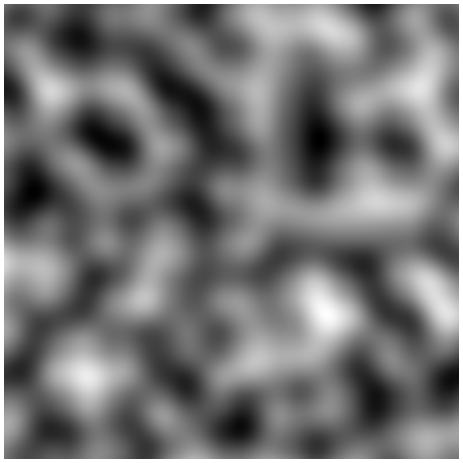


Abbildung 2.3.: Eine mit Perlin Noise erstellt Textur

natürlich aussehende, zufällige Verteilung zu erreichen. Einer der häufigsten Anwendungsfälle sind Texturen, bei denen die Helligkeit jedes Pixels durch die Perlin Noise Funktion beschrieben wird, wodurch eine Textur einem Rauschen gleicht. [Per02, S.681f] Eine solche Textur könnte zum Beispiel verwendet werden, um realistisch aussehende Flecken auf einem Objekt zu simulieren, welche sehr künstlich aussehen würden, wenn ein Designer versuchen würde, sie von Hand zu erstellen. Ein andere Anwendung des Perlin Noise wäre die Höhe der *Vertices* einer Fläche durch die Werte der Perlin Noise Funktion zu bestimmen, was eine "Berglandschaft" erzeugen könnte.

Perlin Noise stellt ein gutes Beispiel dar, um die Vorteile der prozeduralen Generierung zu erläutern. Der erste Vorteil ist die Vielfalt von Anwendungsfällen, die mit einer prozeduralen Methode behandelt werden können. Dieselbe Funktion findet Anwendung in einer Vielzahl von verschiedenen Fällen, die eine natürliche, zufällige Verteilung benötigen. Der nächste Vorteil ist die geringe benötigte Zeit, um den Content zu erstellen. Eine realistisch aussehende Textur von Hand zu gestalten, erfordert eine große Menge an Zeit für Recherche und Malen sowie ein großes Maß an Erfahrung von Seiten des Designers. Gerade Aufgaben, die sich häufig mit geringen Unterschieden wiederholen, können sich gut für einen prozeduralen Workflow eignen. Ein weiterer Vorteil ist die Möglichkeit, die prozedurale Generierung durch das Nutzen anzupassen. Während herkömmliche Content Generierung ein Produkt fertigstellt und es dann dem Nutzer zur Verfügung stellt, kann prozedurale Generierung abhängig von den Anfragen oder Aktionen des Nutzers neuen Content erstellen, der

den gewünschten Kriterien entspricht, oder bestehenden Content anpassen. Wenn zum Beispiel in einem Videospiel ein Nutzer eine Vorliebe für sehr kurze Level mit einer hohen Dichte von Gegner hat, könnten entweder die bestehen Level mit mehr Gegnern gefüllt werden oder mehr Level mit einer kurzen Länge generiert werden. [VDLLB13, S.78ff]

Die prozedurale Generierung bringt aber auch ihre eigenen Herausforderungen mit sich. Eine der größten davon ist die Kontrolle über die Generierung. Den Einfluss, den ein Nutzer der Generierung auf den Prozess haben kann, ist meist auf die Parameter, welche bei der Konzeption dieses Prozesses festgelegt wurden, beschränkt [HMVDVI13, S.1ff]. Sollten nur sehr wenige Parameter verfügbar sein, kann dies zu einer geringen Anzahl an potenziellen Ergebnissen führen, da eine geringe Anzahl von Parametern eine reduzierte Anzahl von Kombinationen zwischen diesen bedeutet. Außerdem erschwert eine geringe Anzahl von Parametern, es den Einfluss eines einzigen Parameters auf das resultierende Ergebnis nachzuvollziehen, weil ein Parameter womöglich mehr als eine Variable des Vorgangs beeinflusst. Wenn aber zu viele Parameter verwendet werden, kann dies die Arbeit mit dem System anspruchsvoller und zeitaufwändiger gestalten. Grund hierfür ist, dass der Nutzer für jeden neuen Inhalt die Parameter anpassen und verstehen muss. Dies führt auch zu einer weiteren Herausforderung, namentlich, dass das System möglichst intuitiv sein muss [STdKB10, S.2f]. Die Person, welche den Algorithmus des Prozesses schreibt, ist selten dieselbe Person, welche das System letztendlich benutzt. Dies ist vor allem der Fall bei Software, die mit grundlegenden Prozessen hilft und deshalb bei einer Vielzahl von Projekten verwendet werden kann, wie zum Beispiel "Substance Painter"[3], eine Software zum Texturieren von 3D-Assets die bei Animationsfilmen und Videospielen zum Einsatz kommt. In einem solchen Fall ist es selten möglich, sich den eigentlichen Algorithmus anzusehen, und selbst wenn dies möglich ist, kann die Person, die von dem Verfahren gebraucht macht, ein Laie im Bereich der Programmierung sein. Deshalb sollten Parameter und ihr Einfluss auf eine einfach zu verstehende Weise dargestellt werden.

[3] https://www.substance3d.com/

2.4. Visualisierung

Es gibt viele verschiedene Methoden, um Wissen zu vermitteln, zum Beispiel durch ein Buch oder eine Präsentation. Da bei einer großen Anzahl von Lernenden die individuellen Lerntempos nicht in hinreichender Weise berücksichtigt werden können, müssen Methoden gefunden werden, Wissen auf neue, effektivere Wege zu vermitteln, auch wenn der Lernende keinen unmittelbarer Kontakt zum Lehrenden hat. Die *Visualisierung* stellt einen Weg dar, dem Lernenden direkt dabei zu helfen, sein eigenes mentales Modell eines Konzepts zu erstellen. [BK00, S.285f] Dieser Abschnitt befasst sich mit den verschiedenen Anwendungsfällen von *Visualisierung*.

Die für diese Arbeit vor allem relevante Form der *Visualisierung* ist das Verdeutlichen von Prozessen, deren zugrundeliegenden Prinzipien durch simples Beobachten schwer nachzuvollziehen sind. Grund hierfür kann zum Beispiel die Geschwindigkeit eines Prozesses sein. Wenn der Prozess sehr schnell abläuft, beispielsweise der Schuss eines Gewehrs, oder äußerst langsam, zum Beispiel im Fall von geologischen Prozessen. Im Falle des Gewehrs kann eine 3D-*Visualisierung* den Prozess, welcher im Innern der Waffe abläuft, nachdem diese betätigt wurde, veranschaulichen. Bei geologischen Prozessen, wie der Bewegung von tektonischen Platten, kann ein 3D-Modell die verantwortlichen Kräfte darstellen und in Zeitraffer zeigen, wie sich diese auf das System auswirken. Ein weiterer Aspekt der *Visualisierung*, der in beiden Fällen relevant ist, ist die Perspektive. Wenn ein Prozess zu groß oder zu klein ist, um beobachtet zu werden oder wenn er sich, wie im Beispiel des Gewehrs, in einem schwer zu betrachtenden Ort abspielt.

Ein weiterer wichtiger Aspekt von *Visualisierung*, ist das Darstellen von Daten in einer einfach zu verstehenden Form. Da die Menge von erstellten und zu interpretierenden Daten zunimmt [MBD+12, S.60ff], suchen zum Beispiel Analytiker oder Wissenschaftler nach Methoden, diese Daten in eine Form zu bringen in der Zusammenhänge zwischen verschiedenen Variablen verdeutlicht werden können.

Für viele Bereiche der *Visualisierung* stellt sich die Frage, ob die *Visualisierung* von Hand erstellt werden soll oder durch ein automatisiertes System. [And95, S.97f] Der große Vorteil eines automatisierten Systems ist das Einsparen von Arbeitszeit. Allerdings ist ein solches System selten flexibel, falls der Vorgang, auf welchem die *Visualisierung* basiert, sich verändert. In diesem Fall muss die *Visualisierung* bei jeder Änderung angepasst werden oder das System für ihre Generierung muss bei

seiner Konzeption schon für die wahrscheinlichsten Änderungen vorbereitet sein. Beide Fälle führen zu mehr Arbeit, die trotzdem geringer sein sollte, als die *Visualisierung* für jeden Anwendungsfall manuell zu erstellen.

Ein konkretes Beispiel für den Nutzen von *Visualisierungen* ist *Visual Programming*. *Visual Programming* beschreibt eine Methode der Programmierung, bei der anstelle von Textzeilen Diagramme und Bilder verwendet werden, um ein Programm zu erstellen. [Mye86, S.59f] Ziel dieser Art von *Visualisierung* ist es vor allem, ein intuitives Verständnis der gegebenen Funktionen zu vermitteln, während ein neues Programm zusammengesetzt wird, und um später den Inhalt eines Programms schneller verstehen zu können, als wenn eine große Anzahl an Zeilen gelesen werden müsste. Ein Beispiel für diese Methodik sind *Blueprints*, in der *Unreal Game Engine*, die es ermöglichen, eine Anwendung zu erstellen, ohne eine einzige Zeile Code zu schreiben. [Gam19] Dies macht *Visual Programming* gerade für Neulinge im Bereich der Programmierung sehr attraktiv.

2.5. JSON

JavaScript Object Notation (JSON) ist ein Dateiformat für den Austausch von Daten und wird verwendet, um Daten von Programmen zu speichern und später wieder zu lesen. Wie der Name vermuten lässt basiert JSON auf der JavaScript Programmiersprache. Eine JSON Datei setzt sich aus Objekten und Listen von auf C basierenden Programmiersprachen zusammen und erlaubt es damit, diese zu speichern [ECM00]. Formate wie JSON und XML werden verwendet, um Daten schnell zu lesen und schreiben, Informationen über Objekte und Struktur der Daten beim Austausch zu behalten und diese Daten dabei in einer von Menschen lesbaren Form zu belassen. Der Hauptvorteil des JSON Formats gegenüber anderen Formaten, wie zum Beispiel XML, ist vor allem seine Geschwindigkeit, mit dem Nachteil, dass manche Funktionen, die in anderen Formaten vorhanden sind, wie zum Beispiel die Fähigkeit Namespaces zu behalten, bei JSON fehlen. Da solche Funktionen für diese Arbeit jedoch nicht relevant sind und da Unity bereits Methoden für das Erstellen und Auslesen von JSON Dateien bereitstellt, wird es für diese Arbeit gewählt, um Daten persistent abzuspeichern. [NPRI09]

3. Anforderungen und Spezifikation

Die Anwendung kann als Einführung für die Möglichkeiten von prozeduraler Generierung angesehen werden und ist damit vor allem für Neulinge des Themas von Nutzen. Die Anwendung eignet sich damit vor allem für den Bereich der Lehre.

3.1. Konzeption

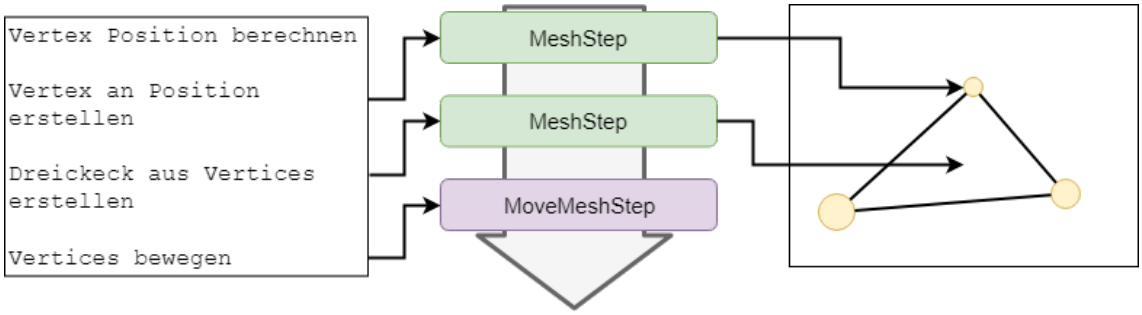


Abbildung 3.1.: Das Grundlegende Konzept des *Step-Systems*. Das System wandelt Codezeilen (links) in *Steps* um (Mitte), welche anschließend ihren Inhalt nacheinander einem Mesh hinzufügen (rechts)

Um das Vorgehen des Verfahrens zu visualisieren, wird ein *Step-System* erstellt. Dieses System besteht aus *Steps*, die an wichtigen Orten innerhalb des Algorithmus eingefügt werden können, um einen Schritt in der Generierung festzuhalten. Der Algorithmus zum Erstellen des Assets fügt dann die *Steps* einer Liste hinzu, während er fortschreitet. Diese Liste ist das, was anschließend die eigentliche Visualisierung beinhaltet. Durch Input des Nutzers wird der Inhalt des nächsten oder des vorhergehenden *Steps* dem momentanen Stand des Assets hinzugefügt beziehungsweise von diesem entfernt. Wichtig zu bemerken ist hierbei, dass jeder *Step* nur die Veränderung zur nächsten Position enthält, nicht den momentanen Zustand des Assets. Jeder *Step* kann außerdem Text beinhalten, welcher innerhalb

der Anwendung angezeigt wird, sobald der Nutzer diesen *Step* erreicht. Dabei wird kein vollständiges Asset persistent gespeichert, was eine leichtgewichtige Erweiterung der Anwendung ermöglicht. Zu diesem Zweck werden verschiedene Typen von *Steps* erstellt, die im Folgenden näher erläutert werden.

Der *Step*-Typ, welcher dafür zuständig ist, das eigentliche Mesh zu generieren, ist der ***MeshStep***. Dieser *Step* speichert eine Liste von Vertices und eine Liste von Dreiecken, welche in diesem *Step* hinzugefügt werden.

MoveMeshSteps sind dafür zuständig, bereits generierte Meshes zu bewegen. Um den gewünschten Teil des Meshes zu bewegen, müssen dessen Vertices bewegt werden. Deshalb benötigt dieser *Step* den Index des ersten Vertex, welches bewegt werden soll, zusammen mit der Anzahl der insgesamt zu bewegenden Vertices. Der Vektor, um den die Vertices bewegt werden, wird ebenfalls im *Step* gespeichert.

Um bereits erstellte Vertices zu entfernen, werden ***RemoveMeshSteps verwendet***. Diese benötigen ebenfalls den Index des ersten zu löschenden Vertex oder Dreiecks aber anstelle der Anzahl erhält der *Step* eine Liste der zu löschenden Elemente. Diese wird benötigt, um das gelöschte wieder herzustellen, wenn der *Step* rückgängig gemacht werden soll.

CircleSteps sind für die Visualisierung von Kreisen zuständig. Sie fügen die Vertices von Kreisen hinzu und visualisieren die Berechnung dieser mit Vektoren. In ihnen müssen eine Vielzahl von Informationen, wie das Vertex des Radius des Kreises, seine Position und Rotation usw. festgehalten werden.

TransformSteps werden verwendet, um die Position und Rotation von Transforms zu visualisieren. Sie fügen nichts zum Mesh hinzu und verändern auch nicht die existierenden Vertices und Dreiecke, sondern dienen nur dazu, mehr Informationen über das Generieren zu geben.

Für den spezifischen Anwendungsfall ein Gitter aus Position zu generieren werden ***GridSteps*** verwendet. Dies ist zum Zeitpunkt der Fertigstellung der Arbeit nur an einer einzigen Stelle innerhalb des Prozesses notwendig und wie die *TransformSteps* dient dieser *Step* ausschließlich der Visualisierung und verändert das generierte Mesh auf keine Weise.

Den letzten *Step*-Typen stellen die ***Segment-Steps*** dar. Sie werden verwendet, um zu den verschiedenen Abschnitten des Generierens springen zu können.

3.2. Anwendungssteuerung

Den größten Aspekt der Anwendung stellt die Visualisierung des Verfahrens dar, weshalb diese alle Eingabefunktionalitäten, mit Ausnahme der Fortbewegung und des Anwendungsmenüs, ausmacht. Die Fortbewegung innerhalb der Anwendung findet durch Teleportation statt, indem der Nutzer den in Tabelle 3.1 angegebenen Knopf drückt und hält und den Controller in die Richtung zeigt, in welche er sich teleportieren möchte. Es erscheint ein Bogen vom Controller des Nutzers zur Position, zu welcher er teleportiert wird, sobald er den Knopf loslässt. Diese Art der Fortbewegung ist bereits in SteamVR vorhanden und muss lediglich in der Anwendung implementiert werden.

Input	**Aktion links**	**Aktion rechts**
Trigger	Einen *Step* zurück	Einen *Step* weiter
Touchpad berühren		Rechte Hälfte: Visualisierung fortlaufen lassen Linke Hälfte: Visualisierung zurück laufen lassen
Touchpad drücken	Teleportation	
Grip	Zum vorherigen SegmentStep springen	Zum nächsten SegmentStep springen
Menübutton	Anwendungs-Menü aufrufen	Anwendungs-Menü aufrufen

Tabelle 3.1.: Die möglichen Inputs der Anwendung und die damit verknüpften Aktionen

3.3. Systemarchitektur

Abbildung 3.2 zeigt die Systemarchitektur, welche der Anwendung zu Grunde liegt. Die ProceduralGeneration Klasse beinhaltet den Algorithmus, welcher das Mesh generiert. Dieser ist eine simplifizierte Version des Algorithmus, welcher verwendet wurde, um in der vorangehenden Vorarbeit dieser Arbeit ein Mesh eines Zellnetzwerks zu generieren.

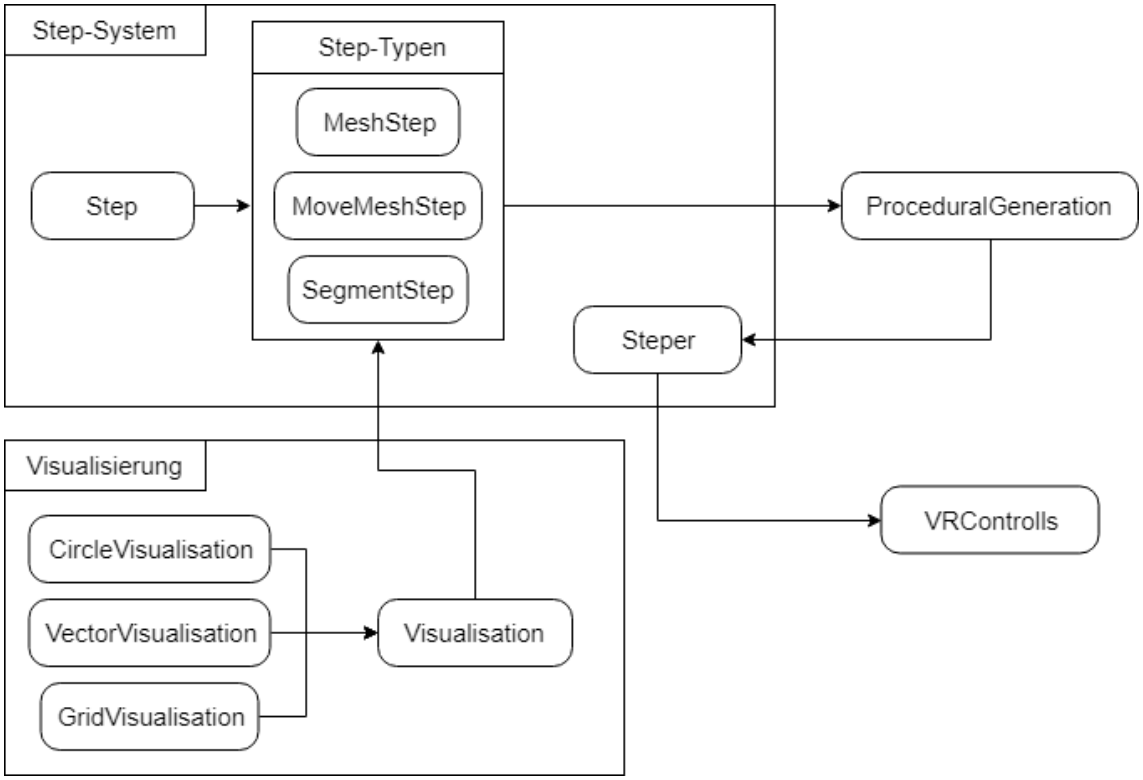


Abbildung 3.2.: Die Systemarchitektur der Anwendung. Die Rechtecke stellen die beiden großen Aspekte der Anwendung dar. Die Rechtecke mit runden Ecken stellen Klassen dar, die Pfeile verdeutlichen die Beziehungen zwischen diesen

In diese Klasse werden die verschiedenen *Steps* des *Step-Systems* eingefügt. Ein *MeshStep* würde zum Beispiel nach dem Generieren eines Vertex eingefügt, ein *MoveMeshStep* nach dem Bewegen dieses. Jeder Typ von *Step* erbt von der abstrakten *Step* Klasse und implementiert die Methoden, welche den *Step* nach vorne abspielen und welche den Inhalt des *Steps* rückgängig machen, sollte der Nutzer den Schritt zurück springen wollen. Die verschiedenen Typen von *Steps* werden dann an passenden Stellen in der ProceduralGeneration Klasse erstellt. Welche Zeilen innerhalb des Codes der ProceduralGeneration Klasse als passend angesehen werden, kommt auf den Typ des jeweiligen *Steps* an. *MeshSteps* bieten sich an den Stellen an, bei welchen der Algorithmus neue Vertices oder Dreiecke dem Mesh hinzufügt, *RemoveMeshSteps* dort, wo er diese entfernt, usw. Die auf diese Weise generierten *Steps* werden dann einer Liste innerhalb der Steper Klasse hinzugefügt. Diese Klasse stellt die Methoden zum Durchlaufen der *Steps* zur Verfügung sowie die Methoden zum Speichern und

Laden der Liste. Da alle *Step*-Typen von der Step Klasse erben, muss die Steper Klasse nicht auf die verschiedenen Typen Rücksicht nehmen, sondern es können nur die Methoden der Step Klasse, welche die *Step*-Typen implementieren, verwendet werden. Dies sorgt dafür, dass neue *Step*-Typen hinzugefügt werden können, ohne dass das System verändert werden muss, um diese abspielen zu können, und sorgt damit für ein hohes Maß an Modularität.

Um die verschiedenen *Steps* zu visualisieren, stellt die Visualisation Klasse alle Methoden für das Erstellen von verschieden Visualisierungs-Elementen zur Verfügung. Sie fasst dabei mehrere Klassen zusammen, wie die VectorVisualisation Klasse, welche für das Erstellen von Vektoren verantwortlich ist. Die verschiedenen Typen von *Steps* erstellen dann aus diesen Elementen die Visualisierung für den jeweiligen *Step*, welche daraufhin von der Steper Klasse aufgerufen wird. Die Visualisation Klasse enthält zu diesem Zweck auch Referenzen zu Objekten, wie Kugeln und Dreiecke, von welchen Kopien erstellt werden können, um eine bestimmte Visualisierung zu erstellen.

Die VRControlls Klasse dient dazu, mit den Inputs der HTC-Vive Controller die Methoden der Steper Klasse zum Bewegen innerhalb der *Step*-Liste aufzurufen. Zu diesem Zweck lösen die verschiedenen Inputs der Controller sogenannte Unity Events aus. Diese Events werden dann verwendet, um eine oder mehrere Methoden aufzurufen, wenn sie ausgelöst werden. Wenn der Nutzer also den Trigger der Controller drückt, löst dies das TriggerEvent aus und die daran gebundene Methode der Steper Klasse wird aufgerufen. Die verschiedenen Inputs auf diese Weise zusammenzufassen, erleichtert auch das spätere Hinzufügen von weiteren Features.

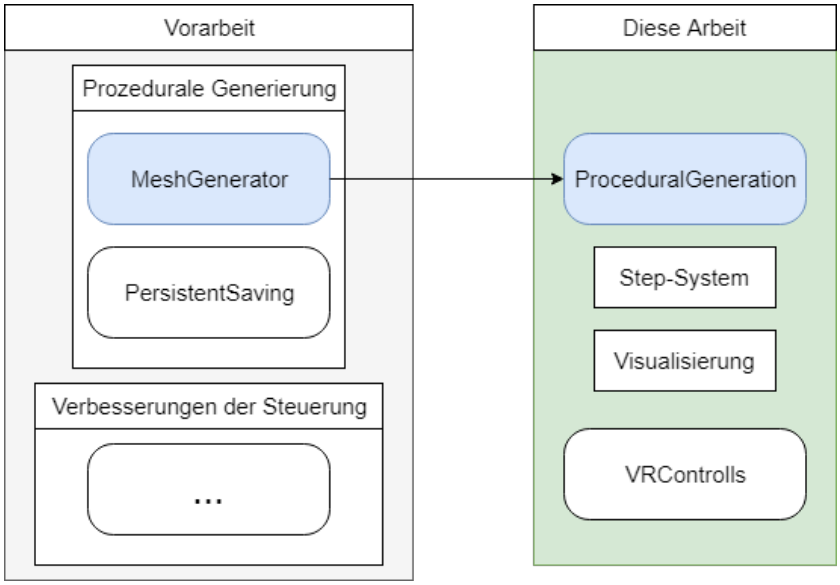


Abbildung 3.3.: Nur die MeshGenerator Klasse der Vorarbeit wird in dieser Arbeit verwendet

4. Implementierung

Wie in Kapitel 1 erwähnt, wurde im dieser Arbeit vorangehenden Praxismodul das Verfahren erstellt, welches visualisiert werden soll. Da der Algorithmus darauf ausgelegt ist ein fertiges Asset zu erstellen, welches dann persistent abgespeichert wird, muss der Algorithmus erst so angepasst werden, dass er nach Beginn nicht nur ein fertiges Asset hervorbringt, sondern einzelne Schritte, die dann vom Nutzer durchlaufen werden können. Anschließend werden die einzelnen *Steps* visualisiert, um den Nutzen hinter diesen zu erklären.

4.1. Step System Implementieren

Um das oben beschriebene *Step System* zu implementieren, muss an den meisten Stellen im Code, welche als ein Schritt angesehen werden soll, ein *Step* instanziiert werden und anschließend der Inhalt des *Steps* zugewiesen werden. Die grundlegendsten *Steps* sind die *Meshsteps*, welche nur aus Vertices oder Dreiecken, also aus den Eckpunkten und den Flächen zwischen diesen, bestehen. Da die Vertices nicht sichtbar sind, bevor eine Fläche zwischen ihnen gebildet wird, werden ihre Position mit Objekten markiert, deren Form in der Engine wählbar ist, wobei für die vorliegende Arbeit gelbe Kugeln benutzt werden. Diese Objekte werden beim Starten der Anwendung generiert und im weiteren Verlauf einer Sitzung nur bewegt, aktiviert und deaktiviert, aber nicht neu instanziiert, um Ressourcen zu sparen. Der eigentliche Algorithmus zum Erstellen des Assets läuft dabei genau wie ohne das *Step System* ab, mit Ausnahme der Methode, welche die generierten Vertices und Dreiecke dem Mesh des Objektes zuweist. Außerdem wird in diesem Kapitel das Verfahren zum Generieren des Assets erläutert, um die spätere Visualisierung zu erklären. Die Abschnitte sind in aufsteigender Reihenfolge der Komplexität des generierten Objekts nach geordnet.

4.1.1. Kreis

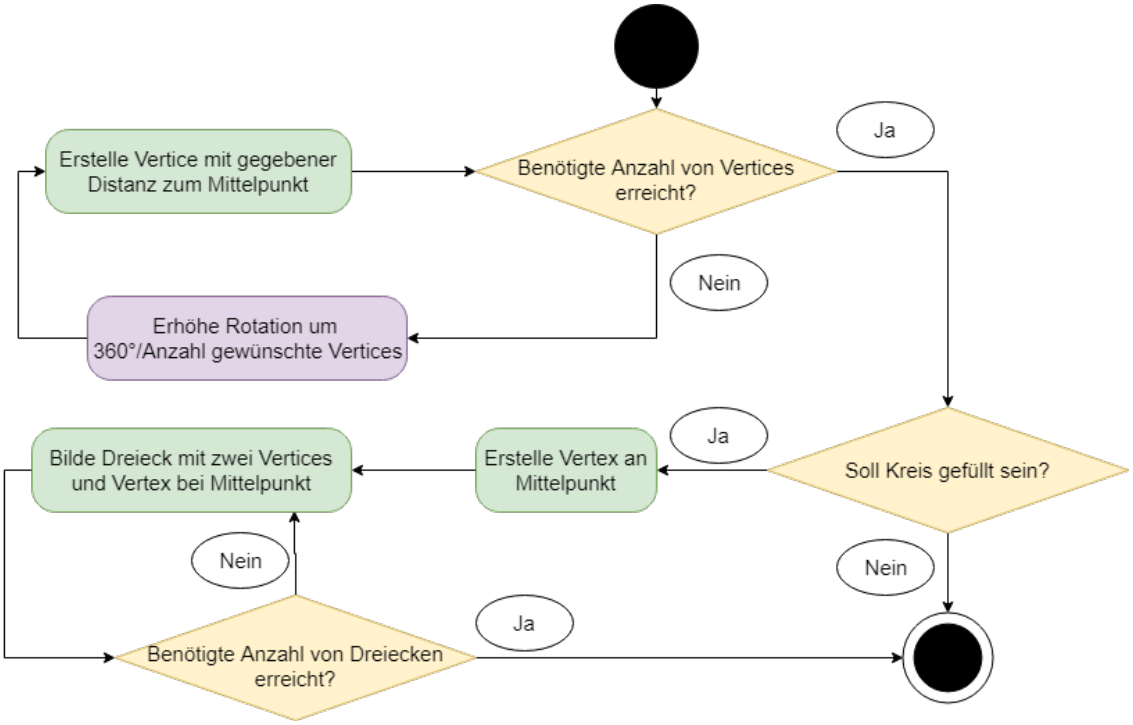


Abbildung 4.1.: Algorithmus zum Erstellen von Kreisen. Grün: Schritte, die Neues hinzufügen, Lila: Schritte, die Bestehendes bewegen, Gelb: Abfragen

Die einfachste geometrische Form innerhalb des Verfahrens stellt ein Kreis dar, welcher hier in *Steps* eingeteilt wird. Um einen Kreis zu generieren, werden als erstes die Vertices, welche den Rand des Kreises ausmachen, in gleichen Abständen um den Mittelpunkt gesetzt. Um die Position zu bestimmen, teilt der Algorithmus zuerst 360 Grad, also einen vollen Kreis, durch die Anzahl der gewünschten Vertices. Der daraus resultierende Wert wird innerhalb einer Schleife mit jedem Durchlauf hinzugefügt. Vom momentanen Stand der Summe wird dann der Sinus berechnet für die X-Koordinate des Vertice sowie der Kosinus für die Z-Koordinate und beide werden mit dem gewählten Radius des Kreises multipliziert. Jedes Vertice wird einem eigenen Schritt zugewiesen, weil jeweils eines dieser Vertices mit dem Durchlaufen der verantwortlichen Schleife hinzugefügt wird und somit die *Steps* der Funktionsweise des Algorithmus am nächsten kommen. Außerdem erleichtert dies später die Visualisierung, da diese für jeden einzelnen Schritt gemacht werden kann und die Unterschiede zwischen den einzelnen *Steps* die Vorgehensweise verdeutlichen.

Anschließend wird, abhängig davon, ob die Fläche des Kreises gefüllt sein soll, das Vertex für den Mittelpunkt gesetzt und Dreiecke mit jeweils zwei Vertices des Rands und dem des Mittelpunkts gebildet, die wieder jeweils einen Schritt repräsentieren. Andernfalls wäre es schwierig, zu erkennen, dass die resultierende Fläche aus Dreiecken besteht, da die Fläche keine Unebenheiten aufweist und es keine Schatten oder Linien zwischen den Vertices gibt, welche den Nutzer die Dreiecke erkennen lassen würden.

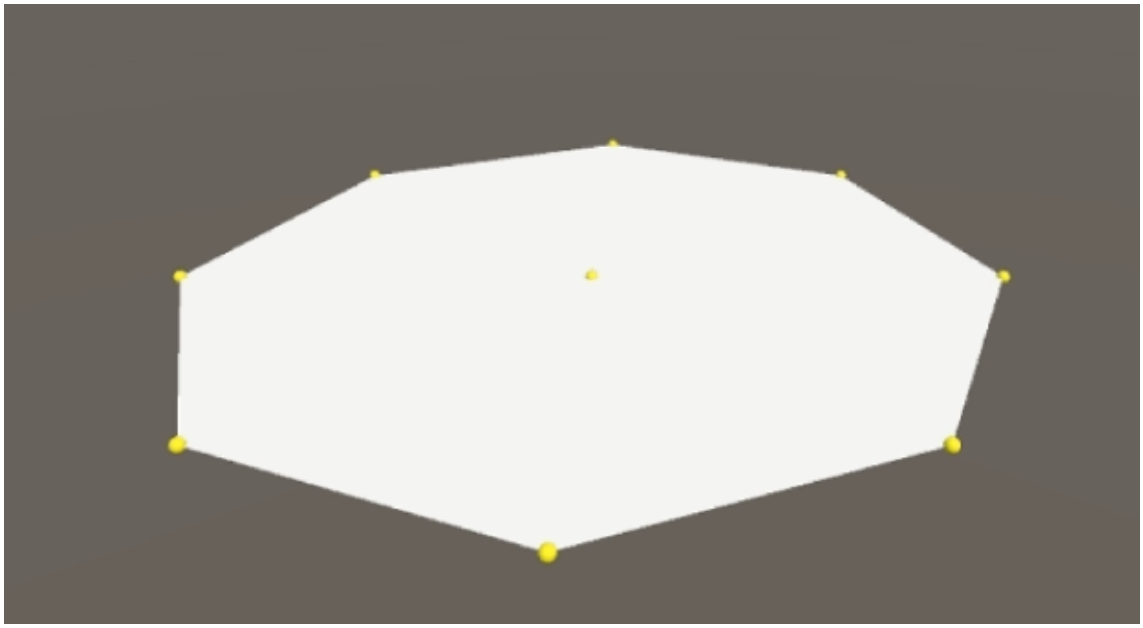


Abbildung 4.2.: Ein generierter Kreis

4.1.2. Zylinder und Kurven

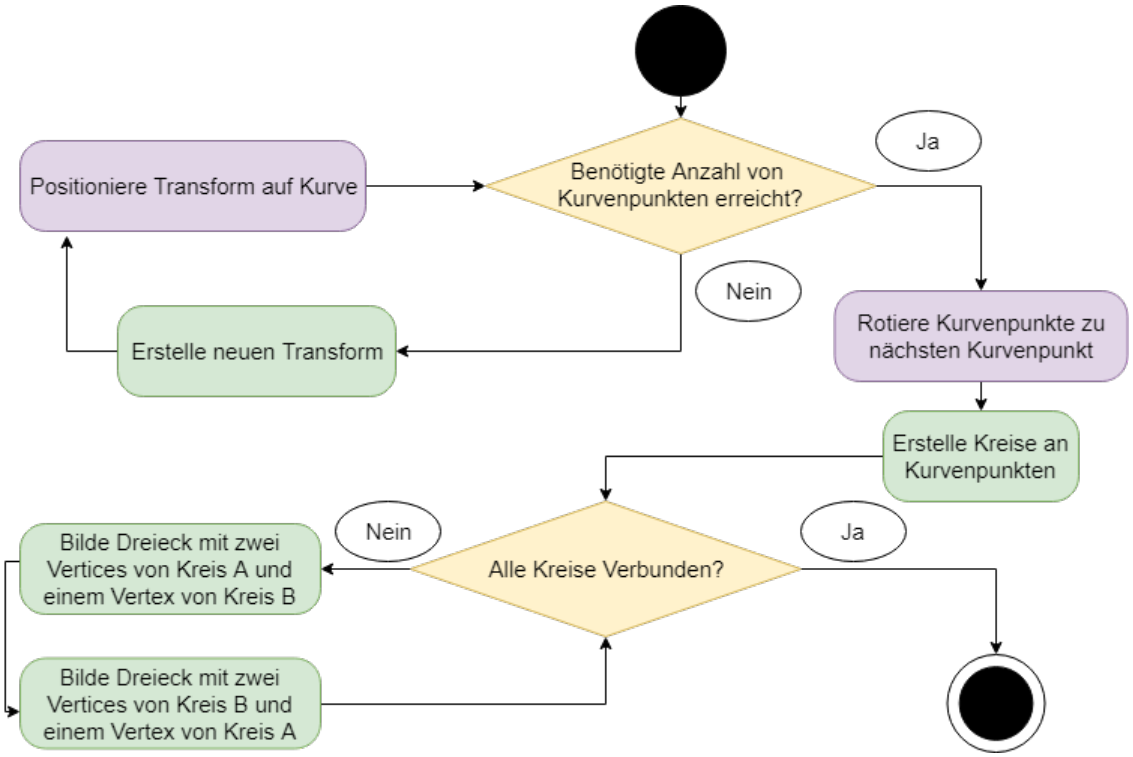


Abbildung 4.3.: Algorithmus zum Erstellen von Kurven. Grün: Schritte, die Neues hinzufügen, Lila: Schritte, die Bestehendes bewegen, Gelb: Abfragen

Der Zylinder baut auf den Kreisen auf, indem er zwei nicht gefüllte Kreise mit Dreiecken verbindet. Die Kreise werden genau wie oben beschrieben erstellt und die resultierenden Vertices in *Steps* festgehalten. Jeweils zwei Dreiecke, die ein Rechteck der Seite des Zylinders ergeben, werden hier in einem *Step* hinzufügt, um das Vorgehen des Algorithmus zu demonstrieren, der mit jedem Schritt der Schleife die zwei Dreiecke hinzufügt. Auf Basis der Zylinder können die Kurven, die als Verbindung zwischen den *Zellen* verwendet werden, ebenfalls schrittweise erstellt werden. Diese bestehen aus mehreren Kreisen von Vertices, wobei jeder Kreis zum Mittelpunkt des nächsten Kreises rotiert ist, sodass die Y-Achse, welche die mittlere Achse jedes Kreises darstellt, auf den jeweiligen Kreis zeigt. Die *Steps* sind hier die gleichen wie beim Zylinder. Auch hier werden erst Vertices nacheinander angelegt und im Anschluss die Dreiecke, welche diese verbinden, erstellt.

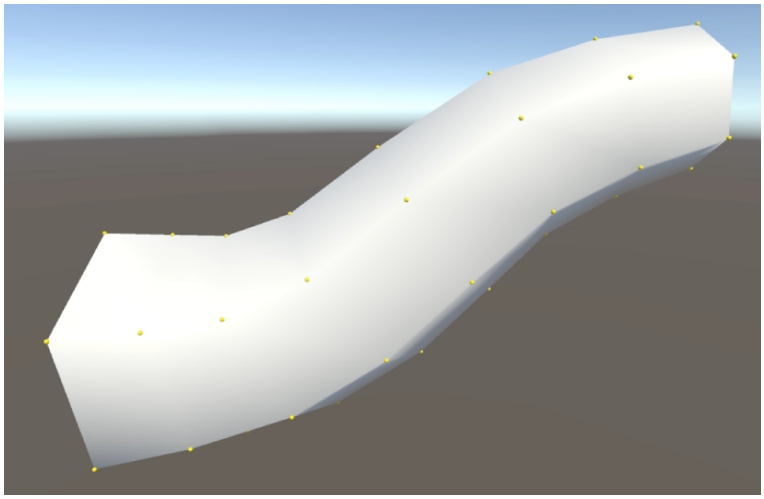


Abbildung 4.4.: Eine generierte Kurve

4.1.3. Icosphere

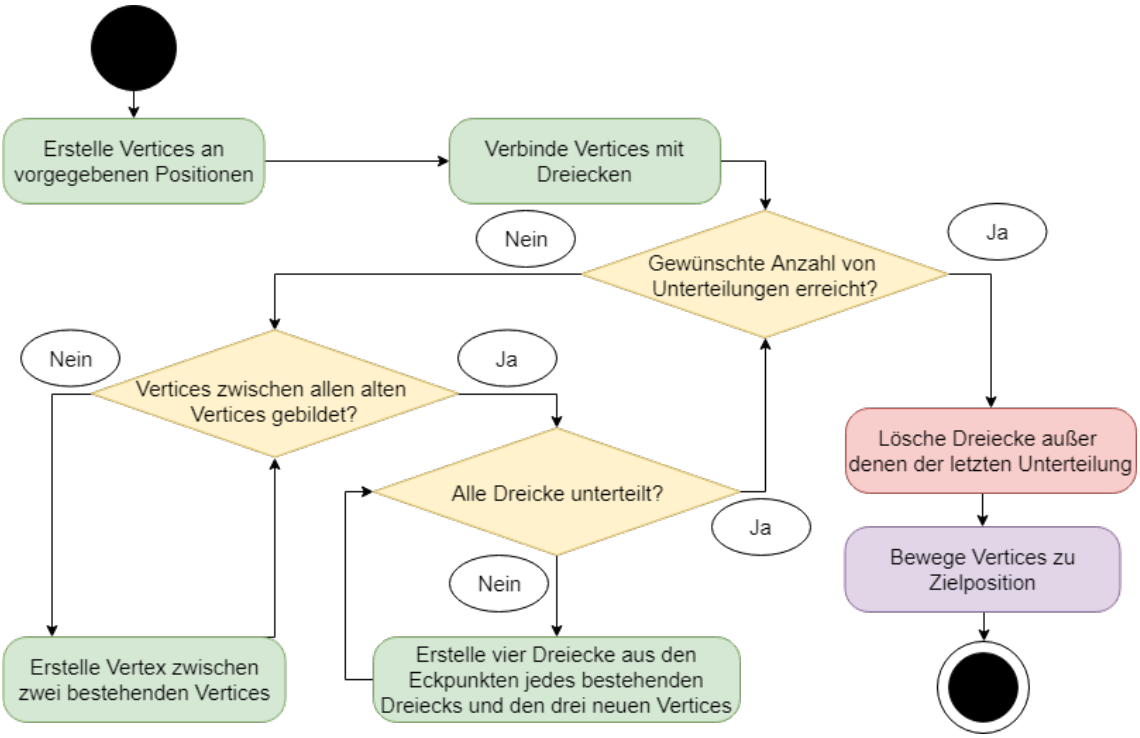
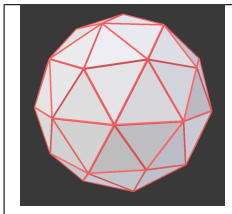
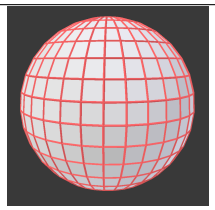
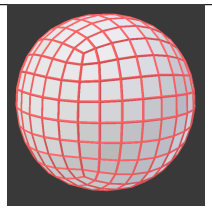


Abbildung 4.5.: Algorithmus zum Erstellen von Icospheres. Grün: Schritte, die Neues hinzufügen, Lila: Schritte, die Bestehendes bewegen, Rot: Schritte, die Bestehendes entfernen

	Icosphere	**UV-Sphere**	**Flächenbasierte Kugel**
			
Beschr.	Kugel auf Basis eines 20-seitigen Würfels dessen Dreiecke unterteilt werden	Kugel aus zur Mitte größer werdenden Kreisen	Kugel basierend auf Flächen deren Vertices den selbe Abstand zum Mittelpunkt erhalten
Vorteile	-Gute Verteilung von Vertices -Geringe Anzahl von Vertices	-Große Kontrolle über Anzahl Vertices -Leicht zu Teilen	-mittelmäßige Verteilung -mittelmäßig zu Teilen
Nachteile	-Schwer zu Teilen	-Vertices sammeln sich an Polen	-Vertices sammeln sich in Ecken

Die letzte grundlegende Form innerhalb des Verfahrens ist die Icosphere. Dabei handelt es sich um eine Kugel, die in ihrer simpelsten Version die Form eines 20-seitigen Würfel annimmt. Vorteil einer solchen ist die gleichmäßige Verteilung der Vertices. Dies reduziert die benötigte Menge zum Erreichen einer runden Form. Da der Nutzer zu diesem Zeitpunkt das simple Hinzufügen von Vertices sowie Dreiecken bereits ausreichend durch die Kreise und Zylinder kennengelernt hat, fügen die einzelnen *Steps* für das schrittweise Erstellen der Icosphere mehrere Vertices und Dreiecke hinzu. Es werden zuerst zwölf Vertices innerhalb von drei *Steps* erstellt. Diese stellen die Ecken des 20-seitigen Würfels, welcher die Grundlage der Icosphere bildet, dar. Die Dreiecke der Oberfläche werden in einem einzigen *Step* hinzugefügt, da die Reihenfolge innerhalb des Algorithmus nicht durch eine Berechnung, sondern nur für einfache Lesbarkeit im Code gewählt wird. Es würden deshalb keine relevanten Informationen durch das einzelne Hinzufügen vermittelt werden. Falls die Dreiecke der Icosphere unterteilt werden sollen, um die Form mehr der einer Kugel gleichen zu lassen, müssen als nächstes die Kanten jedes Dreiecks geteilt werden. Dies wird durch ein neues Vertice in der Mitte der Kante erreicht, welches dann noch vom Zentrum des Kreises wegbewegt wird, um auf der gewünschten Kugel zu liegen.

Hier erhält jedes Vertice wieder einen eigenen *Step*, um den beschriebenen Vorgang einzeln zu visualisieren. Aus den so entstandenen drei neuen Vertices zusammen mit den drei Vertices des originalen Dreiecks werden vier neue Dreiecke erstellt, die innerhalb eines *Steps* hinzugefügt werden. Dieser Prozess wird mit jedem Dreieck wiederholt. Falls mehr als ein Level von Unterteilungen gewünscht ist, wird der Prozess noch weitere Male für jedes neu erstellte Dreieck wiederholt. Um das Generieren der Icosphere abzuschließen, müssen letztendlich die originalen Dreiecke sowie die Dreiecke jeder Unterteilung außer der letzten gelöscht werden. Dies spart später Ressourcen für das Rendern von Dreiecken, die weder erwünscht noch sichtbar sind. Um die Kugel zu einer gewünschten Position zu bewegen, wird ein Vektor, der zum jeweiligen Ziel zeigt, auf jedes Vertice, welches die Kugel definiert, addiert.

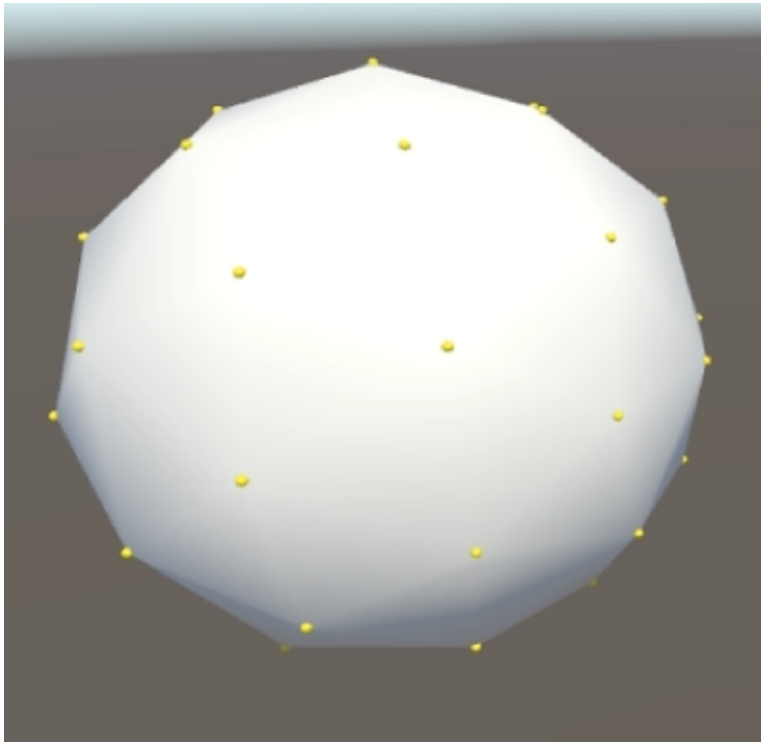


Abbildung 4.6.: Eine generierte Icosphere

4.1.4. Netzwerk

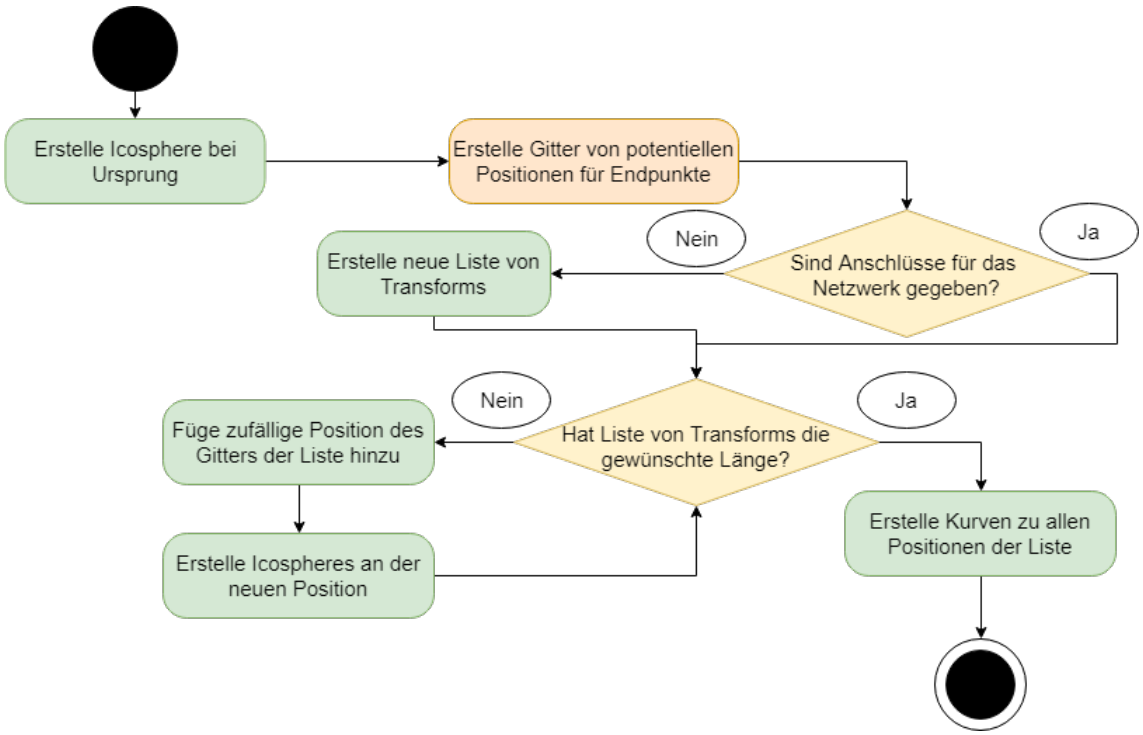


Abbildung 4.7.: Algorithmus zum Erstellen von Netzwerken. Grün: Schritte, die Neues hinzufügen,Gelb: Abfragen, Orange: spezielle Schritte

Aus den Kurven und Icospheres kann der Algorithmus Netzwerke, welche einzelne Zellen mit ihren Anschlüssen darstellen, generieren. Diese dienten ursprünglich dazu, Zellen der Netzhaut des menschlichen Auges und deren Synapsen darzustellen. Zu Beginn des Generierens wird an der Position, die als Ursprung des Netzwerks gewählt wird, eine Icosphere erstellt, von der aus mehrere Kurven zu verschiedenen Zielpunkten führen. Diese Zielpunkte haben zwei verschiedene Ursprünge. Den ersten stellen leere Anschlüsse dar. Diese werden erstellt, indem zuerst eine Position als Richtung, in welche die Anschlüsse des Netzwerks zeigen sollen, gegeben wird. An dieser Position wird dann ein zum Ursprung des Netzwerks senkrecht stehendes Gitter aus potenziellen Positionen für die Anschlüsse erstellt. Die Positionen des Gitters werden einerseits durch eine Angabe der Breite und Höhe des Gitters sowie durch die festgelegte Größe der Enden bestimmt. Die endgültige Position eines Anschlusses besteht aus der lokalen Position des richtungsgebenden Transforms, auf den ein Vektor addiert wird, der zu einer zufällig gewählten Position innerhalb des

Gitters zeigt. Jede Position hat dabei dieselbe Chance, gewählt zu werden. Um für einen größeren Zufall bei der endgültigen Position zu sorgen, kann der Wert, der zum Ursprung zeigenden Z-Achse, ebenfalls mit einem Grad von Zufall versehen werden. Dieser Wert geht aus dem Bereich zwischen einer angegeben Zahl und derselben Zahl im Negativen hervor, die der Algorithmus auf die Z-Koordinate addiert. Nachdem die Kurve erstellt ist, wird eine Icosphere, mit der für das Ende des Anschlusses angegebenen Größe, als Durchmesser erstellt. Den anderen Ursprung der Zielpunkte stellen die freien Anschlüsse eines anderen Netzwerks dar. Wenn diese dem Netzwerk beim Erstellen bekannt sind, bildet dieses eine Verbindung zwischen seinem Ursprung und dem jeweiligen Anschluss. In diesem Fall werden keine neuen Icospheres erstellt. Jeder Anschluss kann nur mit einem einzigen anderen Anschluss verbunden werden. Es ist auch möglich, dem Netzwerk eine gewisse Zahl von freien Anschlüssen anderer Netzwerke zu geben und gleichzeitig eine höhere Zahl von Anschlüssen insgesamt zu definieren. In diesem Fall verbindet sich das Netzwerk erst mit den gegebenen Anschlüssen und erstellt dann Anschlüsse, welche noch keine Verbindung zu anderen Netzwerken darstellen, bis die gewünschte Zahl erreicht ist.

Das Netzwerk stellt den Übergang zwischen den Schritten des Generierens, die allein durch das Hinzufügen, Bewegen und Löschen visualisiert werden können, und denen, die nur durch erklärende Hilfestellung visualisiert werden können, dar. Die Kurven und Icospheres fallen in die erste Gruppe, da sie wie bereits beschrieben mit Vertices und Dreiecken iterativ erstellt werden können. Die anderen Aspekte wie das Gitter oder der Transform der Richtung des Netzwerks sind den Vertices nicht anzusehen. Deshalb wird in der nächsten Sektion eine Visualisierung erstellt, welche die fehlenden Informationen vermittelt.

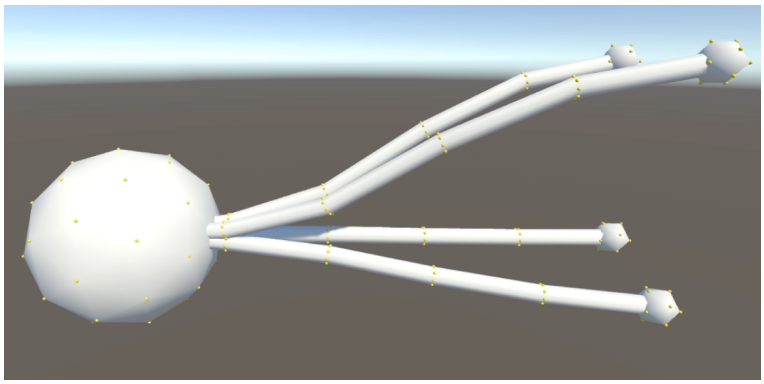


Abbildung 4.8.: Ein generiertes Netzwerk

4.1.5. Zellnetzwerk

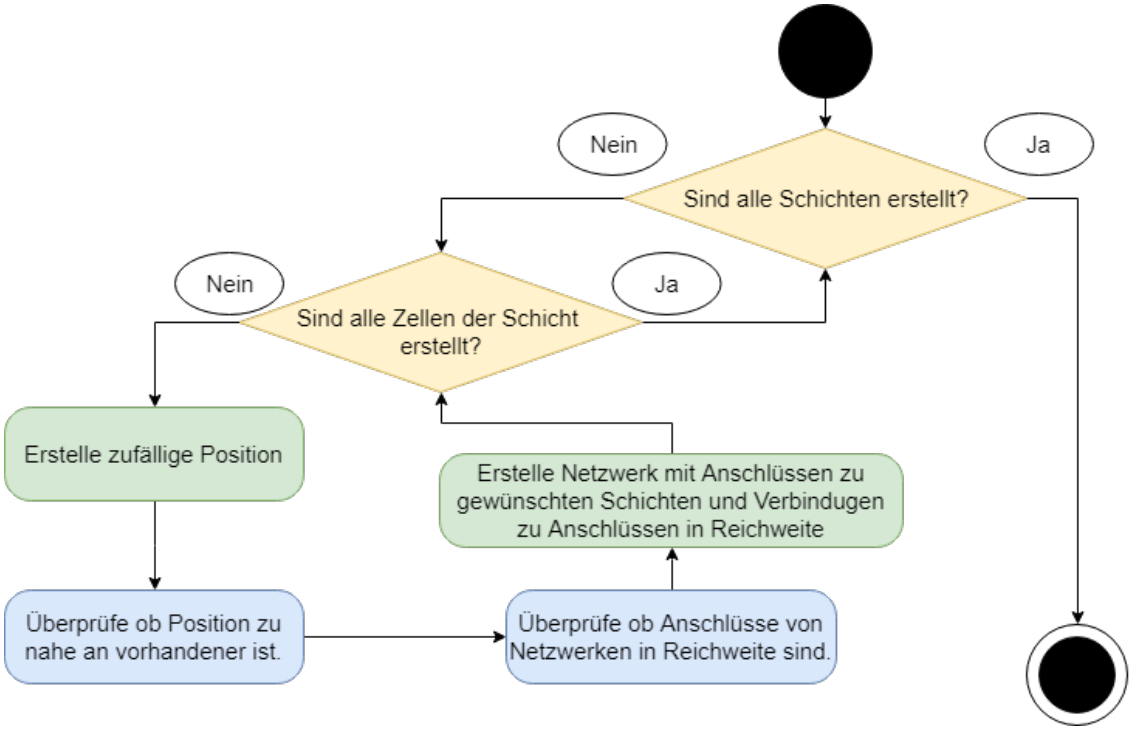


Abbildung 4.9.: Algorithmus zum Erstellen von Zellnetzwerken. Grün: Schritte, die Neues hinzufügen, Gelb: Abfragen, Blau: Schritte, die Reichweiten überprüfen

Der Algorithmus generiert das Zellnetzwerk aus mehreren Schichten von Netzwerken, welche er im Bereich eines Viertels einer Kugel um den Ursprung des Gameobjects anordnet. Die Schichten repräsentieren dabei verschiedene Zell-Typen innerhalb der Netzhaut. Die Zellen verteilen sich dabei annähernd auf eine Halb-Kugel, welche in eine Viertel-Kugel geteilt wird, um das Generieren von inhaltlich gleichen Hälfte zu sparen und das Betrachten des Querschnitts zu ermöglichen. Das Zellnetzwerk stellt das Endergebnis des Verfahrens dar. Zu Beginn des Generierens gibt der Nutzer eine Vielzahl von Parametern an, welche die Generierung beeinflussen. Einer dieser Parameter ist die Distanz, welche die Netzwerke zum Ursprung haben und welche auch die Größe des Ergebnisses bestimmt, zusammen mit dem Parameter für die Breite der einzelnen Schichten. Hauptsächlich maßgeblich für die Zeit, welche der Computer benötigt, um das Zellnetzwerk zu erstellen, sind vor allem die folgenden Parameter. An erster Stelle steht die Anzahl der Zellen, welche bestimmt, wie viele

Netzwerke der Algorithmus pro Schicht generiert. Außerdem versucht der Algorithmus, für den Ursprung eines Netzwerks eine Position zu finden, welche nicht einen minimalen Abstand zu den anderen Netzwerken der jeweiligen Schicht unterschreitet. Die Anzahl der Versuche für diesen Schritt kann die benötigte Zeit drastisch beeinflussen. Der Algorithmus generiert erst alle Netzwerke einer Schicht, bevor die nächste Schicht erstellt wird. Die nicht verbundenen Anschlüsse jeder Schicht werden bei ihrem Erstellen in einer Liste gespeichert und beim Generieren der nächsten Schicht wird überprüft, welche Anschlüsse in Reichweite für die neu generierten Netzwerke sind.

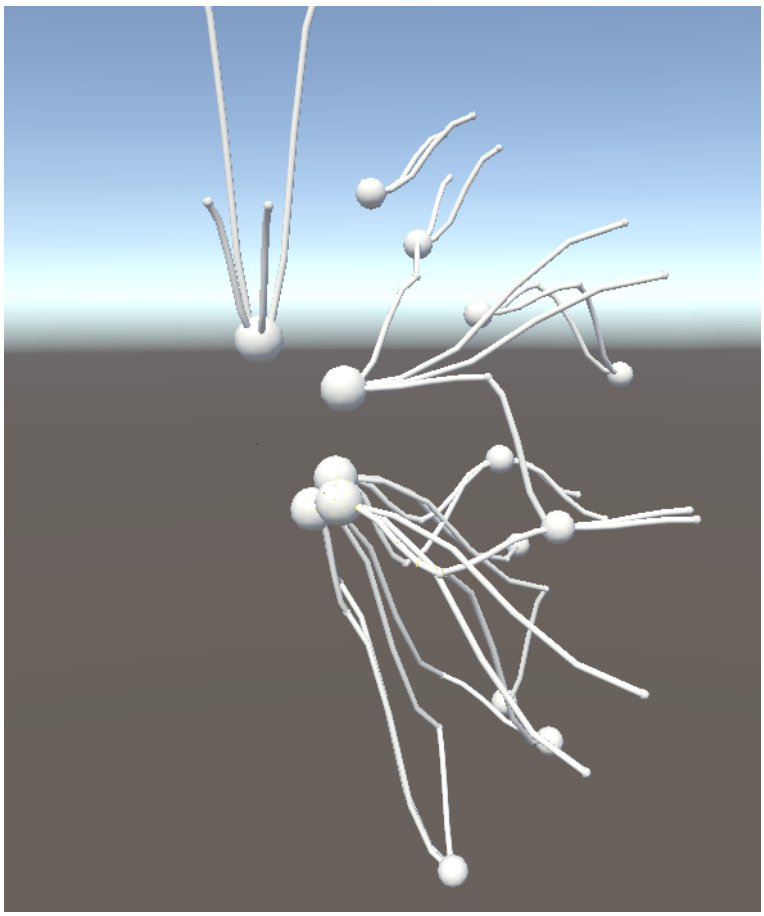


Abbildung 4.10.: Ein generiertes Zellnetzwerk

4.2. Visualisierung

Aufbauend auf dem *Step-System* entsteht die Visualisierung des Generierens des Meshs. Diese Visualisierung setzt sich aus den Visualisierungen der individuellen *Steps* zusammen. Diese setzen sich wiederum aus kleineren Visualisierungselementen zusammen.

4.2.1. Visualisierungs Elemente

Für die Visualisierung der einzelnen *Steps* wurden Elemente erstellt, welche bei mehreren *Steps* zum Einsatz kommen. Diese Elemente und ihre Funktionsweise werden in den nachfolgenden Absätzen näher erläutert.

Um die Vektoren zu visualisieren, wurden zuerst ein Zylinder und ein Kegel in Blender, einer 3D-Modelling Software, erstellt. Der Kegel stellt hierbei die Spitze und der Zylinder den Körper eines Pfeils dar, welcher den Vektor repräsentieren soll. Um einen Vektor zu visualisieren, wird der Ursprung des Vektors, das Ziel des Vektors und die gewünschte Dicke des Pfeils der Visualisierung benötigt. Der gesamte Pfeil wird zuerst zum Ursprung gesetzt. Danach wird der Zylinder skaliert, bis er die Distanz von Ursprung zu Ziel erreicht hat, wobei die Distanz um die Länge der Spitze reduziert wird. Die Spitze wird dann am Ende des Zylinders positioniert. Falls die Dicke größer oder kleiner als ihr Standard-Wert ist, wird der Durchmesser des Körpers sowie die Skalierung der Spitze in jede Richtung verändert und die Länge des Körpers und die Position der Spitzte angepasst, um dies auszugleichen. Zuletzt wird der gesamte Pfeil rotiert, sodass er auf das Ziel zeigt.

Jedes Dreieck wird durch drei Vektoren, die zu den Vertices zeigen, veranschaulicht. Jedes Vertices stellt hierbei eine Ecke des Dreiecks dar und sie werden, für die Visualisierung, mit farbigen Kugeln markiert. Die Fläche des Dreiecks wird markiert, indem die Normale des Dreiecks gebildet wird, also der Vektor, der senkrecht auf der Fläche des Dreiecks steht. Um den Normalen Vektor zu bilden, wird eines der Vertices des Dreiecks gewählt und zwei Vektoren, die von diesem Vertice zu den anderen beiden Vertices zeigen, gebildet. Das Kreuzprodukt dieser beiden Vektoren resultiert im Normalen Vektor. Es werden dann zwei neue Dreiecke erstellt mit gleichen Vertices wie die des Originals mit dem einzigen Unterschied, dass auf die Vektoren der Vertices des einen Dreiecks der Normalen Vektor addiert wird und von

den Vektoren des anderen Dreiecks subtrahiert wird. Die neuen Dreiecke werden dann verbunden, um eine dreidimensionale Darstellung des ursprünglichen Dreiecks zu erstellen.

Wie oben beschrieben werden Kreise erstellt indem Vertices mit einer gewissen Distanz zum Mittelpunkt in Abständen angeordnet werden, welche von der gesamten Anzahl an Vertices abhängen. Für die Visualisierung wird in jedem Schritt ein Vektor zu dem in diesem Schritt erstellten Vertice und ein weiterer zu dem im nächsten Schritt erstellten Vertice visualisiert. Der Kreisausschnitt zwischen diesen Vektoren wird durch eine Kurve in runder Form gekennzeichnet. Diese Kurve wird mit einer abgewandelten Version des Algorithmus zum Bilden der Kurven der Netzwerke generiert. Der Gradwert des Winkels dieses Kreisausschnitts wird anschließend in dem Bereich zwischen Kurve, Vektoren und Mittelpunkt angezeigt.

4.2.2. Ablauf

Die Visualisierung ist in den einzelnen *Step*-Typen definiert. Diese bedeutet, dass zum Beispiel *MeshSteps* immer auf eine Art visualisiert werden und *MoveMeshSteps* immer auf eine andere. Mit jedem Step wird die Visualisierung für den nächsten *Step* ausgeführt und die Visualisierung des momentanen *Steps* wird beendet. Außerdem wird ein Text angezeigt, welcher zu jedem *Step* zusätzliche Informationen gibt. Diese können einerseits Informationen geben, welche nicht aus der Visualisierung selbst zu erkennen sind, oder das Ziel mehrerer folgender *Steps* beschreiben. Im Folgenden wird der Ablauf der Anwendung beschrieben, wenn der Nutzer sie vom ersten bis zum letzten *Step* abspielen lässt.

Die Anwendung findet in einer Umgebung statt, welche aus einer quadratischen Fläche als Boden besteht und einem blauen Fenster, welches den Informationstext der jeweiligen Schritte anzeigt. Die Fläche ist transparent, damit die Visualisierung sichtbar bleibt, auch wenn sie durch den Boden reichen sollte. Der Nutzer kann sich auf jeden Punkt der Fläche teleportieren. Das Fenster, welches den Text anzeigt, rotiert dabei immer in Richtung des Kopfs des Nutzers, sodass der Text gut zu lesen bleibt. Es werden dem Nutzer außerdem 3D-Modelle seiner Controller angezeigt, was vor allem Neulingen helfen sollte, wenn sie Hinweise zur Steuerung erhalten, während sie sich in der Anwendung befinden.

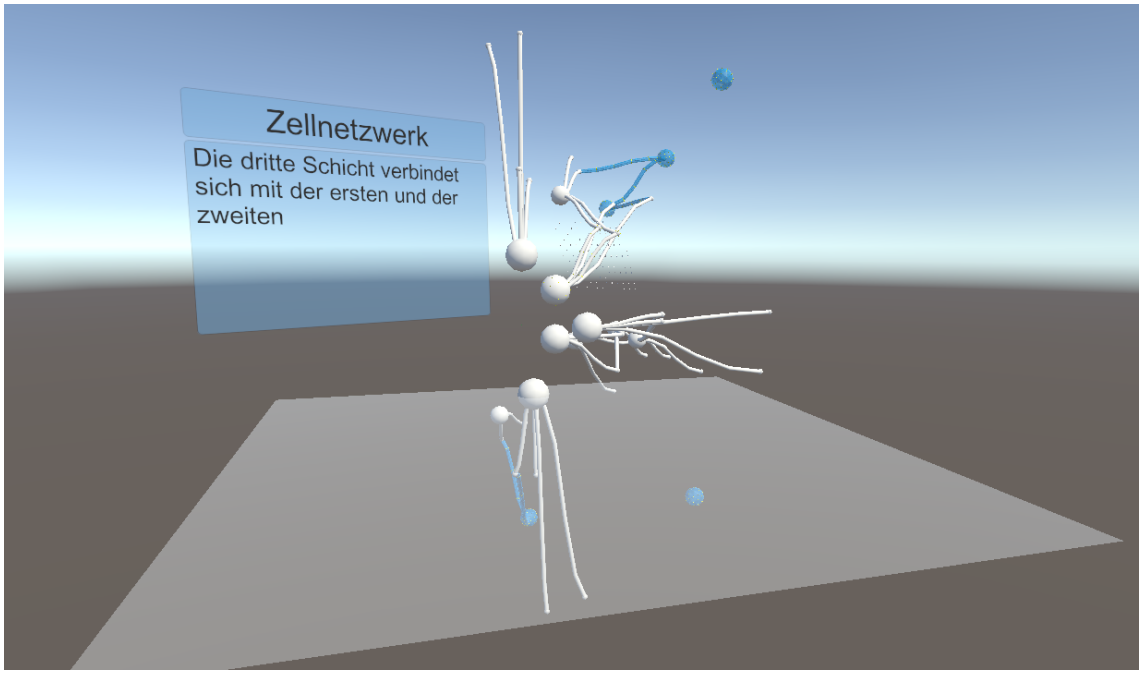


Abbildung 4.11.: Die Umgebung der Anwendung. Das blaue Fenster zeigt die Beschreibung des Textes, das transparente Quadrat ist der Boden, auf welchem sich der Nutzer teleportieren kann, und die Kugeln und Kurven stellen das generierte Mesh dar

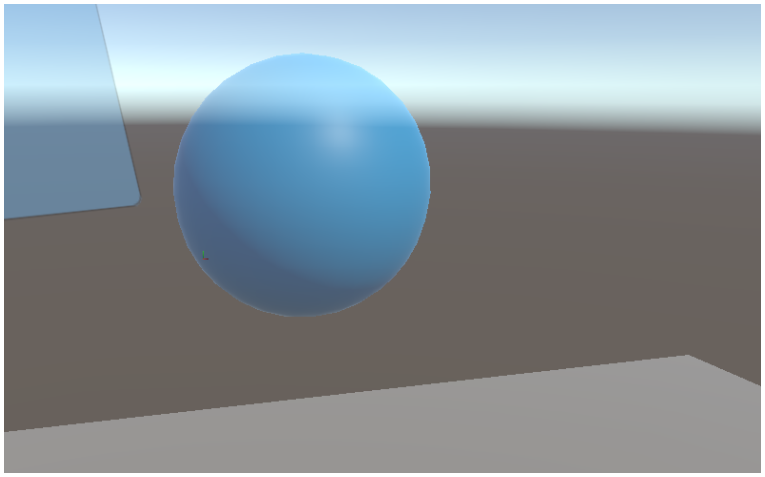


Abbildung 4.12.: Die blaue Kugel stellt den Mindestabstand dar, den eine Position zu anderen Netzwerken einhalten muss.

Der Algorithmus beginnt mit dem Generieren des Zellnetzwerks. Dieses versucht als erstes, eine Position für das Generieren des ersten Netzwerks der ersten Schicht von Zellen zu finden. Es wird überprüft, ob diese Position zu nahe an einer bereits vorhandenen ist. Dies wird durch eine blaue, transparente Kugel visualisiert, deren Ursprung sich an der zu überprüfenden Position befindet und deren Radius dem Mindestabstand zur nächsten Position entspricht. Dadurch ist es für den Nutzer leicht, zu sehen, welche Punkte innerhalb der Kugel liegen und damit den Mindestabstand unterschreiten. Da hier die erste Position generiert wird, gibt es keine Position, an der diese zu nahe sein könnte. Danach beginnt das Generieren des ersten Netzwerks.

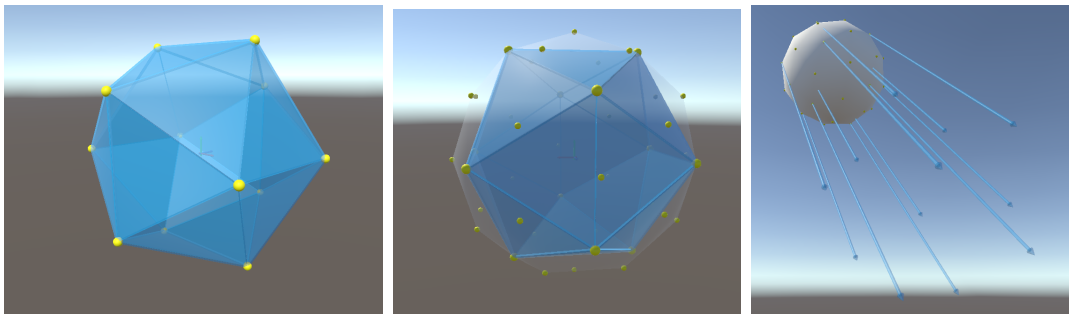


Abbildung 4.13.: Visualisierung von: Die Vertices einer Icosphere werden mit Dreiecken verbunden (links), die Dreiecke, die vor der Unterteilung der Icosphere entstanden sind, werden gelöscht (Mitte), die Vertices der Icosphere werden um einen Vektor bewegt (rechts).

Der erste Schritt ist hierbei das Generieren der Icosphere, welche den Ursprung des Netzwerks markiert. Der erste Schritt, der dabei eine eigene Visualisierung erhält, ist das Verbinden der Vertices mit Dreiecken, wofür die in Abschnitt 4.2.1 erwähnte Visualisierung von Dreiecken zum Einsatz kommt. Dies geschieht auch bei jedem folgenden *Step*, der Dreiecke hinzufügt. Der nächste Schritt, der visualisiert wird, ist das Löschen der Dreiecke, die anfänglich für die Icosphere erstellt wurden. Hierbei wird das Objekt transparent gemacht, damit die Dreiecke im Innern zu sehen sind, und diese werden wieder ebenfalls mit der Visualisierung für Dreiecke markiert. Zu guter Letzt wird die Visualisierung der Vektoren verwendet, um das Bewegen der Vertices zur Zielposition zu markieren. Dabei wird an jedem Vertex, welches vor dem Unterteilen der Icosphere entstanden ist, der Vektor zur Zielposition markiert. Es wäre hierbei zutreffender, wenn jedes Vertex der Icosphere einen Vektor erhalten

hätte, aber dies wäre besonders bei mehreren Unterteilungen äußerst unübersichtlich und für die Ressourcen des PCs belastend gewesen.

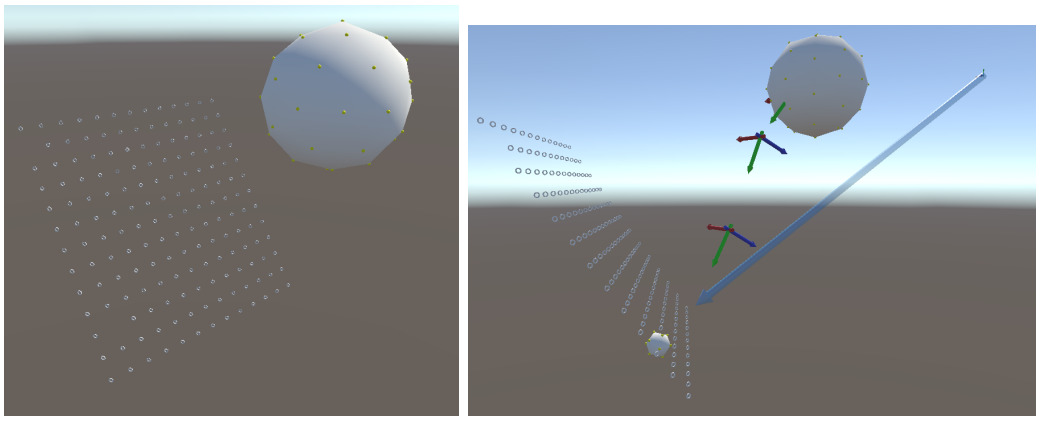


Abbildung 4.14.: Ein Gitter aus Ringen markiert die potenzielle Position für Anschlüsse des Netzwerks (links), die Transforms, welche einen Kurvenpunkt definieren, werden mit Pfeilen visualisiert. Ein blauer Vektor zeigt zur Position des nächsten Transforms

Anschließend geht es mit dem Generieren des Netzwerks weiter, indem das Gitter der potenziellen Position visualisiert wird. Die potenziellen Positionen werden dabei mit Ringen markiert, welche sich zum Kopf des Nutzers rotieren. Nachdem eine Icosphere an einem dieser Punkte gesetzt wurde, beginnt die Visualisierung der Kurve vom Ursprung zu der neuen Icosphere. Die Visualisierung veranschaulicht das Generieren der Kurven, indem sie die Transforms der Kurvenpunkte mit jeweils drei Vektoren markiert, welche die Rotation des Transforms anzeigen. Ein Vektor zeigt vom Ursprung des Zellnetzwerks zur Position des nächsten Kurvenpunkts, um zu verdeutlichen, dass die Transforms sich immer zum nächsten Kurvenpunkt rotieren.

Am ersten Kurvenpunkt, welcher sich innerhalb der Icosphere des Ursprungs des Netzwerks befindet, findet die Visualisierung des Kreises statt. Hierfür wird das Objekt erneut transparent, um dem Nutzer einen besseren Einblick zu geben. Hierbei zeigt ein Vektor zum in diesem Schritt erstellten Vertex und ein weiterer zum Vertex des nächsten Schritts. Zwischen den beiden Vektoren wird einer der Kreisabschnitte aus Abschnitt 4.2.1 generiert sowie ein Text, welcher den Gradwert des Winkels zwischen den Vektoren anzeigt. Anschließend werden weitere Kreise an jedem Kurvenpunkt erstellt und das Verbinden dieser mit Dreiecken visualisiert, was für das

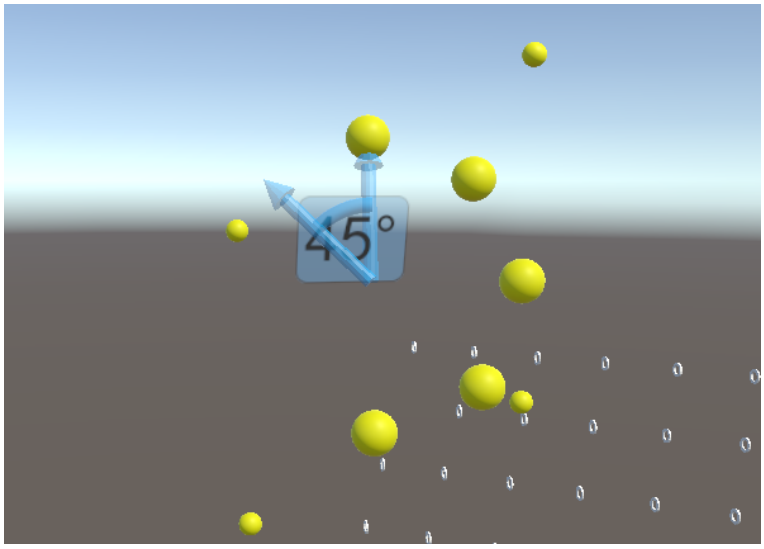


Abbildung 4.15.: Visualisierung des Generierens eines Kreises. Es wird ein Vektor zum zuletzt entstandenen Vertex gezogen und ein weiterer zur Position des nächsten. Diese werden mit einen Kreisausschnitt verbunden und der Gradwert zwischen ihnen angezeigt.

Generieren der restlichen Kurven und Icospheres ebenfalls geschieht. Damit ist das Netzwerk abgeschlossen.

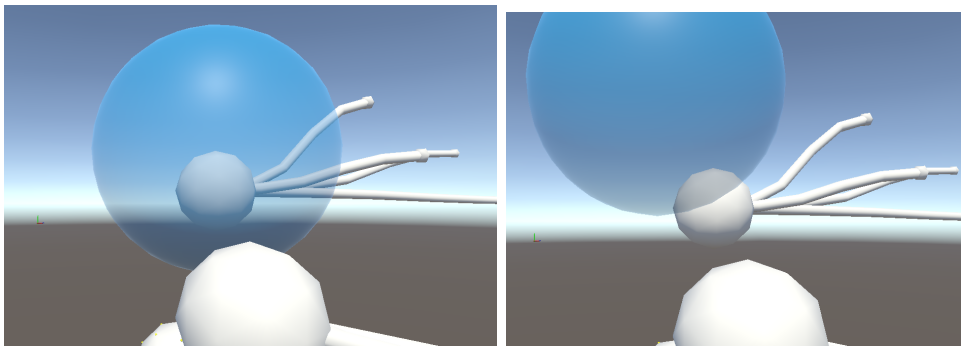


Abbildung 4.16.: Es wird versucht, die Position für ein weiteres Netzwerk innerhalb des Zellnetzwerks zu finden. Der erste Versuch ist zu nahe am Ursprungspunkt eines anderen Netzwerks (links). Der zweite Versuch hat den nötigen Abstand und erlaubt das Generieren eines Netzwerks am Mittelpunkt der blauen Kugel (rechts).

Nachdem die Visualisierung des ersten Netzwerks abgeschlossen ist, wird das Generieren der restlichen Netzwerke des Zellnetzwerks visualisiert. Bei den Netzwerken

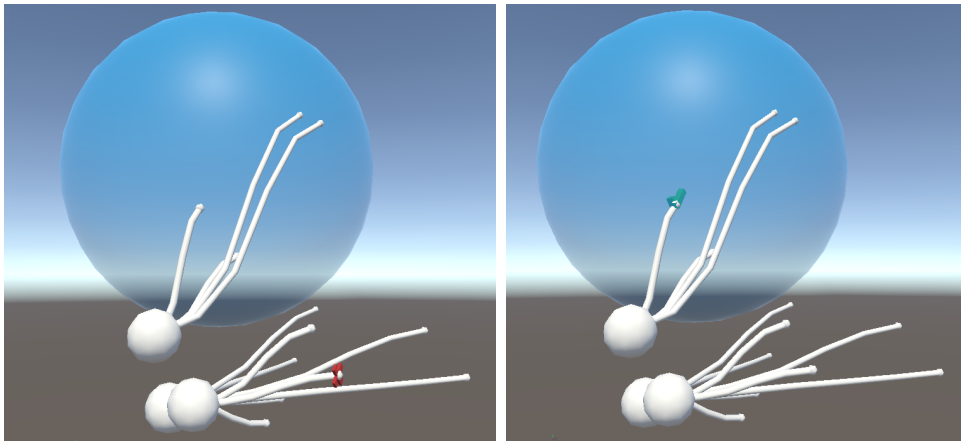


Abbildung 4.17.: Es wird überprüft, welche Anschlüsse in Reichweite für ein neues Netzwerk, welches am Mittelpunkt der blauen Kugel generiert wird, sind. Der erste Anschluss der überprüft wird, befindet sich nicht innerhalb der Kugel und wird deshalb mit einem roten Kreuz markiert (links). Der zweite ist in Reichweite und wird deshalb mit einem grünen Häkchen markiert (rechts).

der ersten Schicht versucht das Verfahren erneut Positionen zu finden, welche nicht den Mindestabstand zu den bereits vorhandenen Positionen unterschreiten. Anders als beim ersten Netzwerks kann es hier zu Kollisionen kommen und das Testen mehrerer Positionen kann notwendig sein, um eine passende Position zu finden. Die Anzahl der verbleibenden Versuche, bevor das Verfahren abbricht, um Zeit zu sparen, kann dem Informationstext entnommen werden. Die erste Schicht generiert Anschlüsse zur zweiten sowie zur dritten Schicht. Das Hinzufügen dieser Anschlüsse wird in jeweils einem Schritt durch die Dreiecksvisualisierung dargestellt. Bei den Netwerken der zweiten Schicht, wird überprüft welche Anschlüsse der ersten Schicht noch keine Verbindung gefunden haben. Dies zeigt sich erneut durch eine Kugel um die Position des zu generierenden Netzwerks. Es wird dann mit jedem Schritt einer der potenziellen Anschlüsse überprüft. Wenn der Punkt innerhalb des Radius liegt, wird ein grünes Häkchen an seiner Position angezeigt, andernfalls ein rotes Kreuz. Die Anzahl der verbleibenden potenziellen Anschlüsse sowie die Anzahl, mit welchen sich das Netzwerk maximal noch verbinden kann, zeigt der Informationstext an. Ein Netzwerke der zweiten Schicht wird, wenn das Finden von Anschlüssen für dieses abgeschlossen ist, zusammen mit den Anschlüssen zur dritten Schicht mit

der Dreiecksvisualisierung gezeigt. Die dritte Schicht wird dann in einem Schritt ebenfalls mit der Dreiecksvisualisierung dargestellt.

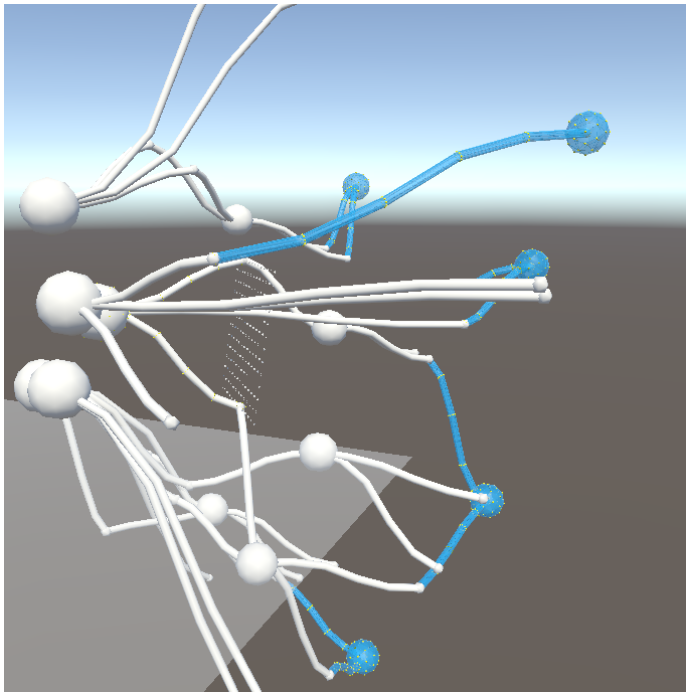


Abbildung 4.18.: Die dritte Schicht des Netzwerks wird blau visualisiert bevor sie hinzugefügt wird.

4.3. Persistentes Speichern

Ein wichtiger Aspekt der Anwendung stellt das Speichern der Visualisierung von generierten Assets dar. Hierfür werden die durch das oben erwähnte *Step System* generierten *Steps*, in einer JSON-Datei gespeichert und bei Bedarf wieder geladen. Für das Erstellen der Json-Datei können die Funktionen der Unity Game-Engine benutzt werden, die zwar in ihren Möglichkeiten nicht über die simpelsten Optionen hinausgehen, aber dafür eine hohe Geschwindigkeit beim Erstellen und Lesen der Dateien vorweisen. Eine Herausforderung stellt hier vor allem das Herauslesen der *Steps* aus einer erstellten JSON-Datei dar, weil es bei selbst erstellten Klassen ausschließlich möglich ist, ein bestehendes Objekt zu überschreiben, nicht aber ein neues direkt aus der JSON-Datei zu generieren. Dies bedeutet, dass der Typ eines *Steps* bereits bekannt sein muss, bevor dieser aus der Datei ausgelesen wird. [Uni19b]

Aus diesem Grund erhält jeder Typ von *Step* eine Nummer zur Identifizierung, die ebenfalls mit dem jeweiligen *Step* in der JSON-Datei gespeichert wird. Diese wird ermittelt, bevor das Objekt aus der Datei gelesen wird, und eine Instanz des benötigten Objekts wird erstellt, deren Parameter dann vom Inhalt der JSON-Datei überschrieben werden.

5. Evaluation

Um die Effektivität der Visualisierung im Hinblick auf das Vermitteln von Lehrinhalten zu testen, wird ein Versuch mit freiwilligen Probanden ausgeführt. Dieses Kapitel beschreibt den Ablauf, die resultierenden Daten und abschließend die Schlüsse, die aus diesen gezogen werden können.

5.1. Aufbau

Die Probanden werden in zwei Gruppen mit jeweils fünf Personen geteilt. Die Versuchsgruppe benutzt die Visualisierung, um sich mit dem Verfahren vertraut zu machen, die Kontrollgruppe benutzt stattdessen einen Text mit Bildern (siehe Anhang A.2).

Die Probanden der Versuchsgruppe erhalten zu Anfang eine kurze Erklärung über die Visualisierung, mit welcher sie sich anschließend in VR über das Verfahren informieren. Sie werden dabei kurz über den originalen Anwendungsfall des Zellnetzwerks als Sehzellen eines Modell des menschlichen Auges aufgeklärt. Darauf folgend werden sie darüber informiert, dass sie sich erst mit der Steuerung der Anwendung vertraut machen werden. Wenn die Probanden dann das Gefühl haben, sich ausreichend mit der Steuerung auszukennen, erhalten sie fünf Minuten Zeit, um sich die Visualisierung anzusehen. Nach fünf Minuten wird das Ansehen in VR beendet oder auch schon früher, falls der Proband bereits der Ansicht ist, alles Nötige über das Verfahren zu wissen. Falls die Probanden darauf bestehen, kann der Versuch auch über das Zeitlimit hinaus andauern, wobei die benötigte Zeit notiert wird. Abschließend füllt der Proband einen Fragebogen A.1.1 aus, welcher aus zwei Teilen besteht, einerseits allgemeine Fragen zur Person des Probanden stellt und andererseits das Wissen des Probanden über das Verfahren abfragt. Anschließend findet ein Gespräch mit dem Probanden statt, bei dem dieser Feedback gibt.

Die Kontrollgruppe hat ihren Text sowie den Fragebogen in einer Mail erhalten, in welcher dieselben grundlegenden Informationen über den Hintergrund des Projekts vermittelt werden. Außerdem informiert die Mail sie darüber, dass sie ein Zeitlimit von fünf Minuten haben, um sich den Text durchzulesen. Das Zeitlimit einzuhalten, wird dabei den Probanden selbst überlassen. Der Text, mit welchem die Probanden sich informieren, besteht aus Screenshots der Anwendung sowie Text, welcher diese erklärt. Der Text ist hierbei derselbe, wie der Text, der innerhalb der Anwendung zu den verschiedenen *Steps* angezeigt wird. Dies stellt sicher, dass beide Gruppen Zugang zu denselben Informationen haben und der einzige Unterschied in dem Medium liegt, durch welches sie diese Informationen erhalten. Nachdem sie den Text gelesen haben, füllen sie den selben Fragebogen A.1.2 wie die Versuchsgruppe aus, mit dem Unterschied, dass der einleitende Text am Anfang des Bogens leicht abgeändert ist, um das veränderte Medium zu berücksichtigen. Außerdem fehlt in diesem die Frage zur Erfahrung mit VR. Anschließend schicken sie den ausgefüllten Fragebogen per Mail, optional mit zusätzlichem Feedback, zurück.

5.2. Ergebnisse

Tabelle 5.1 zeigt die Fragen des Wissenstest zusammen mit den Antworten und der Anzahl von richtigen Antworten von beiden Gruppen. Die Erfolgsrate der Versuchsgruppe liegt bei 52,0 und die Erfolgsrate der Kontrollgruppe bei 49,3 Prozent. In beiden Gruppen gab es Fälle, bei denen das Zeitlimit überschritten wurde. Die genauen Ergebnisse sind im Anhangskapitel B zu finden.

Nr.	Frageninhalt	Antwort	richtige Antworten VR	richtige Antworten Text
1a	Das Objekt wird durch einen Algorithmus generiert.	wahr	4	0
1b	Es werden zuerst Dreiecke und anschließend die Vertices gesetzt.	falsch	3	1
2a	Die Grad Zahl zwischen den Vektoren wird vom Nutzer festgelegt.	falsch	5	3
2b	Kreise sind immer gefüllt.	falsch	1	1
3a	Kurven bestehen aus Kreisen, die mit Dreiecken verbunden werden.	wahr	1	3
3b	Alle Kurvenpunkte sind zum ersten Kurvenpunkt rotiert.	falsch	2	5
4a	Wenn die Icosphere Unterteilt wird, wird jedes ihrer Dreiecke in drei weitere Dreiecke unterteilt.	falsch	1	3
4b	Die Dreiecke die vor der Unterteilung hinzugefügt wurden, werden gelöscht.	wahr	3	2
4c	Nachdem alle Vertices und Dreiecke gesetzt wurden, wird die Icosphere bewegt.	wahr	1	2
5a	Netzwerke bestehen aus Kurven, Rechtecken und Icospheres.	falsch	1	3
5b	Als erstes werden die Kurven der Netzwerke erstellt.	falsch	5	3
5c	Die Endpunkte des Netzwerkes werden Zufällig um den Ursprung gesetzt.	falsch	3	4

Nr.	Frageninhalt	Antwort	richtige Antworten VR/Text	
6a	Das Zellnetzwerk besteht aus mehreren Netzwerken.	wahr	1	4
6b	Das Verfahren versucht, einen Mindestabstand zwischen den Positionen der Netzwerke zu garantieren.	wahr	4	1
6c	Alle Anschlüsse der Netzwerke verbinden sich mit anderen Netzwerken.	falsch	4	2
			39/75 52,0%	37/75 49,3%

Tabelle 5.1.: Ergebnisse des Wissenstest für die Versuchs- und Kontrollgruppe, jeweils bestehend aus fünf Personen

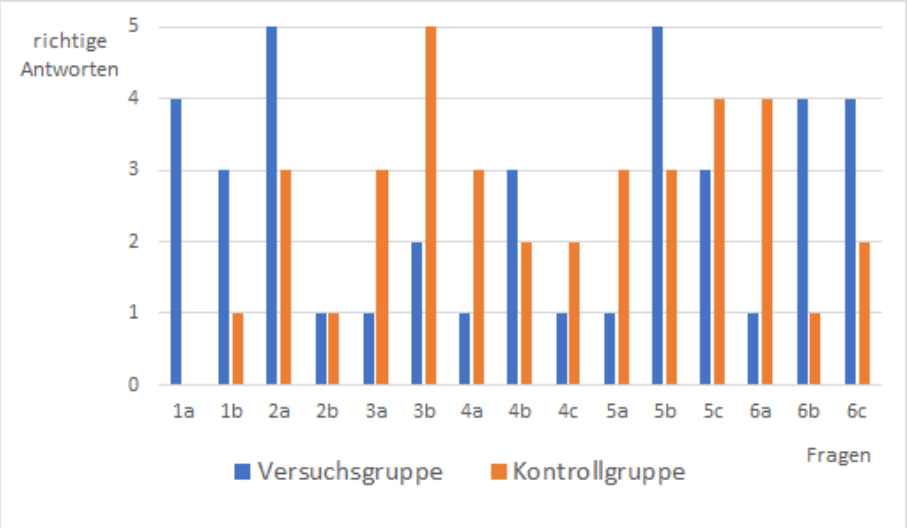


Abbildung 5.1.: Ergebnisse der Gruppen als Balkendiagramm

5.3. Auswertung

Die Ergebnisse der beiden Gruppen zeigen, dass die Gruppen insgesamt sehr nahe beieinander lagen, aber sich sehr stark darin unterscheiden, welche Fragen sie richtig

beantworten können. Da beide Erfolgsraten sehr nahe an 50 Prozent liegen und die Fragen nur als richtig oder falsch gewertet werden können, ist es nicht auszuschließen, dass die Probanden die Fragen vor allem zufällig beantwortet haben und nur ein sehr geringer Wissenstransfer stattfand. Der Mittelwert der jeweiligen Gruppen liegt bei 2,60 beziehungsweise 2,47 und die daraus resultierende Standardabweichung ergibt 1,50 für die Versuchsgruppe und 1,31 für die Kontrollgruppe. In beiden Fällen ist die Standardabweichung relativ hoch, da die doppelte Standardabweichung in positive und negative Richtungen ausgehend vom Mittelwert alle Ergebnisse einschließt. Dies könnte bedeuten, dass die Schwierigkeit der Fragen stark verteilt war, gibt aber zusammen mit der Erfolgsrate von 50 Prozent eher den Eindruck des zufälligen Beantwortens. Dass die Fragen, die richtig beantwortet wurden, sich stark bei beiden Gruppen unterscheiden, verstärkt diesen Eindruck weiter, wobei Grund hierfür sein könnte, dass manche Punkte leichter zu erkennen sind, wenn ein Proband das Modell in VR vor sich sieht, und andere leichter zu erkennen sind, wenn ein Proband ein einziges Bild vor sich hat, was auf den relevanten Punkt fokussiert ist.

Einige der Fragen bieten sich auch für ein näheres Betrachten an. Zum Beispiel Frage 1a, welche die vermutlich grundlegendste Frage des Wissenstest stellt. Kein einziger Proband der Kontrollgruppe konnte diese Frage richtig beantworten. Grund hierfür könnte sein, dass der Versuchsleiter dies während der Einführung für die Probanden der Versuchsgruppe erwähnt hat, aber nicht in den Einführungstext der Kontrollgruppe geschrieben hat. Ebenfalls von Interesse ist die Frage 2b, welche in beiden Gruppen kaum richtig beantwortet wurde. Gemeint war mit der Frage, ob die aus Vertices bestehenden Kreise Dreiecke besitzen, welche den Mittelpunkt mit dem Rand verbinden. Es ist allerdings möglich, dass die Probanden die Frage so verstanden haben, als ob es um Dreiecke zwischen mehreren Kreisen gehen soll, weshalb die Formulierung verdeutlicht werden sollte. Die letzten beiden Fragen könnten in ihren unterschiedlichen Ergebnissen mit dem Zeitlimit zusammen hängen. Bei der Versuchsgruppe hat der Versuchsleiter den Probanden die Option gelassen, das Zeitlimit zu überziehen, wenn diese sich noch weiter mit der Anwendung vertraut machen wollten. Die Kontrollgruppe hatte jedoch keine Kommunikation mit dem Versuchsleiter während des Versuchs und sah sich deshalb womöglich dazu gezwungen, das Lesen des Textes vorzeitig abzubrechen. Dadurch würde diesen Probanden das nötige Wissen fehlen, um die letzten Fragen zu beantworten. Das folgende Feedback beider Gruppen gibt einen Einblick in die Schwierigkeiten der beiden Ansätze sowie Verbesserungsvorschläge (siehe auch Anhangskapitel B).

Die Text-Gruppe hat insgesamt sehr viel weniger Feedback geliefert im Gegensatz zur VR-Gruppe. Hauptgrund hierfür ist vermutlich, dass das Feedback dieser Gruppe ausschließlich in Textform gegeben wurde, während das Feedback der VR-Gruppe in Interviews gegeben wurde. Ein weiterer Grund könnte auch sein, dass ein Text mit Bildern ein sehr simples und typisches Medium darstellt, für welches es schwierig ist, noch weitere Verbesserungen zu finden. Eine Kritik, welche fast alle Probanden lieferten, war, dass das Zeitlimit zu gering gesetzt wurde für die Menge an Inhalt. Dies hat die Qualität der Antworten vermutlich stark verringert. Es haben jedoch mehrere Probanden berichtet, das Zeitlimit überschritten zu haben, wobei zu bedenken ist, dass die Probanden die Zeit selbst überprüft haben, weshalb die Zahl der Probanden, welche die Zeit nicht eingehalten haben, noch höher sein könnte. Es wurden außerdem Schwierigkeiten beim Verstehen des Textes berichtet, wobei die Ursache nicht aus dem Nachfragen deutlich wurde. Mögliche Ursachen könnten aber sein: Die Formulierung des Textes, die Komplexität des Themas beziehungsweise dass dieses nicht gut für Laien aufbereitet wurde oder dass manche Schritte nicht ausreichend ausgeführt wurden. Außerdem gab es bei einem Probanden Verwirrung darüber, ob beim Wissenstest mehrere Antworten per Abschnitt als richtig angekreuzt werden konnten.

Das Feedback der VR-Gruppe ist in manchen Fällen sehr ausführlich geworden und insgesamt auch um einiges vielfältiger. Das häufigste Feedback war die Schwierigkeit, den Text und das generierte Mesh gleichzeitig im Blick zu behalten, beziehungsweise war es oft unklar, welches von beiden zu jedem Zeitpunkt eine höhere Relevanz besitzt. Es wurde auch eine Reihe von Verbesserungsvorschlägen für diese Herausforderung gegeben. Zum Beispiel wurde vorgeschlagen den Text größer und damit einfacher und schneller lesbar zu machen. Weitere Vorschläge betrafen die Position, für welche vorgeschlagen wurde, den Text direkt hinter das Objekt zu platzieren oder den Text fest an die Rotation des Kopfs des Nutzers zu binden, sodass er es immer im Blickfeld hat. Ein letzter Vorschlag hierfür war es, den Text als Audiodatei abzuspielen, um das Problem der Positionierung zu umgehen. Diese Schwierigkeit hat die Ergebnisse des Wissenstest vermutlich stark beeinträchtigt, da die Probanden sich *Steps* entweder angesehen haben, ohne die Theorie hinter diesen zu verstehen, oder Informationen gelesen haben, welche sie später nicht mit einem bestimmten Schritt assoziieren konnten.

Der Punkt, welcher fast so sehr wie die Positionierung des Textes bemängelt wurde, war die mangelnde Interaktivität der Anwendung. Nutzer können sich im Raum

der Anwendung bewegen und die Anwendung vor- und zurückspielen aber nicht mit dem generierten Mesh interagieren. Verbesserungsvorschläge reichen hier davon, die Parameter der Generierung anpassen zu können, bis zur Möglichkeit, eigene Netzwerke zusammenstellen zu können. Viele dieser Vorschläge gehen weit darüber hinaus, was diese Arbeit zu erreichen versucht, aber Interaktion ist eines der besten Mittel, um Interesse zu schaffen und Wissen zu festigen und würde dabei helfen die Visualisierung vom Text abzuheben.

Ein weiterer Punkt, der oft negativ aufgefallen ist, ist die Steuerung der Anwendung. Die größten Schwierigkeiten entstanden dabei beim automatischen Abspielen der *Steps*, was durch Berühren des Touchpads des rechten Controllers ausgelöst wird. Dieses Feature wurde meistens unabsichtlich ausgelöst oder nicht benutzt, auch wenn die Probanden über dessen Existenz informiert wurden. Ein häufig geäußerter Wunsch ist, das Abspielen erst durch Drücken des Touchpads zu aktivieren oder es nur solange abspielen zu lassen, wie das Touchpad berührt wird, und es nur auch nach Loslassen des Touchpads weiterspielen zu lassen, wenn es davor gedrückt wurde. Außerdem wurde angemerkt, dass die Menge an verschiedenen Optionen, um durch die Visualisierung zu navigieren, nicht proportional zum geringen Inhalt und der Zeit, sich mit diesem vertraut zu machen, ist. Eine schlechte Steuerung kann das Arbeiten mit einer Anwendung stark beeinträchtigen. Es wurde versucht, dieses Schwierigkeiten mit der Zeit, welche zu Anfang des Versuchs zum Ausprobieren der Steuerung gegeben wurde, zu mindern.

6. Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde eine Visualisierung eines Verfahrens zur prozeduralen Generierung von Assets in der virtuellen Realität konzeptioniert, implementiert und mit Probanden getestet. Zuerst entstand ein Konzept für ein System, welches Algorithmen zur prozeduralen Generierung in Steps unterteilt, welche es später nacheinander abspielen kann. Anschließend entstanden mehrere Typen von Steps für dieses System.

Das System wurde daraufhin implementiert und auf ein aus der Vorarbeit zu dieser Arbeit stammendes Verfahren angewendet. Die Steps des Systems fanden anschließend in der Visualisierung weitere Verwendung. Außerdem erfolgte eine Erweiterung des Systems um die Option, Steps persistent speichern zu können.

Die Visualisierung wurde dann evaluiert mithilfe von zwei Gruppen von Probanden. Eine Gruppe hat die VR-Anwendung erhalten und die andere ein Dokument mit Bildern und Text aus der Anwendung. Nachdem beide Gruppen sich mit ihrem Medium vertraut gemacht hatten, füllten sie einen Wissenstest aus. Das Ergebnis des Versuchs ist eine Erfolgsrate von 52 Prozent für die VR-Gruppe und 49,3 Prozent für die Text-Gruppen. Dass beide Gruppen eine Erfolgsrate von nahezu 50 Prozent aufwiesen, stellt in mancher Hinsicht den schlechtesten Fall dar, da dieses Ergebnis es besonders schwierig macht, einerseits die Effektivität der beiden Ansätze abzuschätzen und andererseits ihre Schwächen zu finden und nachzuvollziehen.

Die Anwendung kann von dem in dieser Arbeit erreichten Stand auf verschiedene Arten weiterentwickelt werden. Eine Option ist das Umsetzten des Feedbacks der Evaluation. Die Steuerung stellt einen der simpleren Punkte dar, welcher von weiterer Verbesserung profitieren könnte. Dasselbe gilt für die Positionierung von Text und Mesh. Ein größeres Unterfangen ist das Hinzufügen von mehr Interaktivität zur Anwendung. Beispielsweise dafür zu sorgen, dass ein Nutzer die Parameter des Algorithmus verändern kann, würde eine Überarbeitung des Algorithmus zum Generieren des Meshs voraussetzen, da er bei Abschluss dieser Arbeit noch darauf

ausgelegt ist, ein Mesh zu erstellen, welches der Entwickler anschließend persistent speichert.

Vom Feedback der Probanden abgesehen gibt es noch eine Vielzahl andere Wege, die Anwendung in der Zukunft zu verbessern. Es könnte zum Beispiel eine 3D-Umgebung erstellt werden, welche aus mehr als nur einem Quadrat als Boden besteht. Außerdem könnten ergänzende Verfahren zum Generieren von neuen Objekten hinzugefügt werden sowie neue Typen von Steps, um diese zu visualisieren. Schlussendlich könnten Polishing, neue Inhalte und Features die Anwendung in einen Zustand bringen, in welchem diese Verwendung in Vorlesungen zu diesem Thema findet.

Literaturverzeichnis

[And95] Keith Andrews: *Case study. visualising cyberspace: information visualisation in the harmony internet browser*, in *Proceedings of Visualization 1995 Conference*, S. 97–104, IEEE, 1995.

[BK00] Lynne P Baldwin und Jasna Kuljis: *Visualisation techniques for learning and teaching programming*, in *ITI 2000. Proceedings of the 22nd International Conference on Information Technology Interfaces (Cat. No. 00EX411)*, S. 285–290, IEEE, 2000.

[CB10] Marcello Carrozzino und Massimo Bergamasco: *Beyond virtual museums: Experiencing immersive virtual reality in real museums*, *Journal of Cultural Heritage*, Bd. 11(4):S. 452–458, 2010.

[CLL11] Luigi Cardamone, Daniele Loiacono und Pier Luca Lanzi: *Interactive evolution for the procedural generation of tracks in a high-end racing game*, in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, S. 395–402, ACM, 2011.

[ECM00] ECMA: *Introducing JSON*, 2000.
URL https://www.json.org/

[EGW+17] Jan Egger, Markus Gall, Jürgen Wallner, Pedro Boechat, Alexander Hann, Xing Li, Xiaojun Chen und Dieter Schmalstieg: *HTC Vive MeVisLab integration via OpenVR for medical applications*, *PloS one*, Bd. 12(3), 2017.

[Gam19] Epic Games: *Unreal Engine 4 Documentation > Blueprints Visual Scripting*, 2019.
URL https://docs.unrealengine.com/en-US/Engine/Blueprints/index.html

[Haa14] John K Haas: *A history of the Unity game engine*, 2014.

[HMVDVI13] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden und Alexandru Iosup: *Procedural content generation for games: A survey*, *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, Bd. 9(1):S. 1, 2013.

[KBT+17] TP Kersten, G Büyüksalih, F Tschirschwitz, T Kan, S Deggim, Y Kaya und AP Baskaraca: *THE SELIMIYE MOSQUE OF EDIRNE, TURKEY-AN IMMERSIVE AND INTERACTIVE VIRTUAL REALITY EXPERIENCE USING HTC VIVE.*, *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, Bd. 42, 2017.

[KCS17] Jonathan W Kelly, Lucia A Cherep und Zachary D Siegel: *Perceived space in the HTC vive*, *ACM Transactions on Applied Perception (TAP)*, Bd. 15(1):S. 2, 2017.

[KQM13] Daniel Keim, Huamin Qu und Kwan-Liu Ma: *Big-data visualization*, *IEEE Computer Graphics and Applications*, Bd. 33(4):S. 20–21, 2013.

[LCC+07] Chin-Teng Lin, Shang-Wen Chuang, Yu-Chieh Chen, Li-Wei Ko, Sheng-Fu Liang und Tzyy-Ping Jung: *EEG effects of motion sickness induced in a dynamic virtual reality environment*, in *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, S. 3872–3875, IEEE, 2007.

[MBD+12] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil und Dominic Barton: *Big data: the management revolution*, *Harvard business review*, Bd. 90(10):S. 60–68, 2012.

[MGMADGM17] Jorge Martín-Gutiérrez, Carlos Efrén Mora, Beatriz Añorbe-Díaz und Antonio González-Marrero: *Virtual technologies trends in education*, *EURASIA Journal of Mathematics Science and Technology Education*, Bd. 13(2):S. 469–486, 2017.

[Mye86] Brad A Myers: *Visual programming, programming by example, and program visualization: a taxonomy*, in *ACM sigchi bulletin*, Bd. 17, S. 59–66, ACM, 1986.

[NPRI09] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds und Clemente Izurieta: *Comparison of JSON and XML data interchange formats: a case study.*, *Caine*, Bd. 9:S. 157–162, 2009.

[Per02] Ken Perlin: *Improving noise*, in *ACM transactions on graphics (TOG)*, Bd. 21, S. 681–682, ACM, 2002.

[STdKB10] Ruben Smelik, Tim Tutenel, Klaas Jan de Kraker und Rafael Bidarra: *Integrating procedural generation and manual editing of virtual worlds*, in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, S. 2, ACM, 2010.

[TCL+13] Julian Togelius, Alex J Champandard, Pier Luca Lanzi, Michael Mateas, Ana Paiva, Mike Preuss und Kenneth O Stanley: *Procedural content generation: Goals, challenges and actionable steps*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[TYSB11] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley und Cameron Browne: *Search-based procedural content generation: A taxonomy and survey*, *IEEE Transactions on Computational Intelligence and AI in Games*, Bd. 3(3):S. 172–186, 2011.

[Uni19a] Unity: *Unity Manual*, 2019.
URL https://docs.unity3d.com/Manual/index.html

[Uni19b] Unity: *Unity Manual: Json Serialization*, 2019.
URL https://docs.unity3d.com/Manual/JSONSerialization.html

[VDLLB13] Roland Van Der Linden, Ricardo Lopes und Rafael Bidarra: *Procedural generation of dungeons*, *IEEE Transactions on Computational Intelligence and AI in Games*, Bd. 6(1):S. 78–89, 2013.

[ZZ09] Matt Zachara und José P Zagal: *Challenges for success in stereo gaming: a Virtual Boy case study*, in *Proceedings of the international conference on Advances in Computer Enterntainment Technology*, S. 99–106, Acm, 2009.

Anhang

A. Versuchsunterlagen

Für die Versuche wurden mehrere Unterlagen benötigt. Die beiden Gruppen haben dabei unterschiedliche Fragebögen erhalten. Außerdem hat die Kontrollgruppe einen Text mit Abbildungen der Anwendung erhalten um sich über das Thema zu informieren.

A.1. Fragebögen

Fragebogen zu: Visualisierung eines Verfahrens zur Prozeduralen Generierung von 3D Assets

Dieser Fragebogen hat zum Ziel die Effektivität der Visualisierung die Sie benutzt haben zu ermitteln. Zuerst werden ihnen ein paar allgemeine Fragen zu ihrer Person gestellt, anschließend folgt ein Wissenstest zu dem was Sie von der Visualisierung gelernt haben.

Geschlecht?

Männlich ☐ Weiblich ☐ Andere ☐

Alter?

< 18 ☐ 18 – 22 ☐ 23 – 27 ☐ 28 – 36 ☐ 37 – 50 ☐ 50 < ☐

Haben Sie Erfahrung mit VR?

Keine ☐ Weniger als 1x pro Monat ☐ 1x pro Monat ☐ 1x pro Woche ☐
Mehr als 1x pro Woche ☐

Haben Sie Erfahrung mit Prozeduraler Generierung?

Keine ☐ In einem Projekt benutzt ☐ In mehreren Projekten benutzt ☐

Kreuzen Sie die richtigen Aussagen an!

1. Allgemein
   a. Das Objekt wird durch einen Algorithmus generiert. ☐
   b. Es werden zuerst Dreiecke und anschließend die Vertices gesetzt. ☐
2. Kreise
   a. Die Grad Zahl zwischen den Vektoren wird vom Nutzer festgelegt. ☐
   b. Kreise sind immer gefüllt. ☐
3. Kurven
   a. Kurven bestehen aus Kreisen die mit Dreiecken verbunden werden. ☐
   b. Alle Kurvenpunkte sind zum ersten Kurvenpunkt rotiert. ☐
4. Icospheres
   a. Wenn die Icosphere Unterteilt wird, wird jedes ihrer Dreiecke in drei weitere Dreiecke unterteilt. ☐
   b. Die Dreiecke die vor der Unterteilung hinzugefügt wurden werden gelöscht. ☐
   c. Nachdem alle Vertices und Dreiecke gesetzt wurden, wird die Icosphere bewegt. ☐
5. Netzwerke
   a. Netzwerke bestehen aus Kurven, Rechtecken und Icospheres. ☐
   b. Als erstes werden die Kurven der Netzwerke erstellt. ☐
   c. Die Endpunkte des Netzwerkes werden Zufällig um den Ursprung gesetzt. ☐
6. Zellnetzwerk
   a. Das Zellnetzwerk besteht aus mehreren Netzwerken. ☐
   b. Das Verfahren versucht einen Mindestabstand zwischen den Positionen der Netzwerke zu garantieren. ☐
   c. Alle Anschlüsse der Netzwerke verbinden sich mit anderen Netzwerken. ☐

(Optional) Haben Sie Verbesserungsvorschläge?

Abbildung A.1.1.: Fragebogen der Versuchsgruppe, welche eine VR Visualisierung verwendet hat.

Fragebogen zu: Visualisierung eines Verfahrens zur Prozeduralen Generierung von 3D Assets

Dieser Fragebogen hat zum Ziel die Effektivität des Textes den Sie benutzt haben zu ermitteln. Zuerst werden ihnen ein paar allgemeine Fragen zu ihrer Person gestellt, anschließend folgt ein Wissenstest zu dem was Sie von dem Text gelernt haben.

Geschlecht?

Männlich ☐ Weiblich ☐ Andere ☐

Alter?

< 18 ☐ 18 – 22 ☐ 23 – 27 ☐ 28 – 36 ☐ 37 – 50 ☐ 50 < ☐

Haben Sie Erfahrung mit Prozeduraler Generierung?

Keine ☐ In einem Projekt benutzt ☐ In mehreren Projekten benutzt ☐

Kreuzen Sie die richtigen Aussagen an!

1. Allgemein
   a. Das Objekt wird durch einen Algorithmus generiert. ☐
   b. Es werden zuerst Dreiecke und anschließend die Vertices gesetzt. ☐
2. Kreise
   a. Die Grad Zahl zwischen den Vektoren wird vom Nutzer festgelegt. ☐
   b. Kreise sind immer gefüllt. ☐
3. Kurven
   a. Kurven bestehen aus Kreisen die mit Dreiecken verbunden werden. ☐
   b. Alle Kurvenpunkte sind zum ersten Kurvenpunkt rotiert. ☐
4. Icospheres
   a. Wenn die Icosphere Unterteilt wird, wird jedes ihrer Dreiecke in drei weitere Dreiecke unterteilt. ☐
   b. Die Dreiecke die vor der Unterteilung hinzugefügt wurden werden gelöscht. ☐
   c. Nachdem alle Vertices und Dreiecke gesetzt wurden, wird die Icosphere bewegt. ☐
5. Netzwerke
   a. Netzwerke bestehen aus Kurven, Rechtecken und Icospheres. ☐
   b. Als erstes werden die Kurven der Netzwerke erstellt. ☐
   c. Die Endpunkte des Netzwerkes werden Zufällig um den Ursprung gesetzt. ☐
6. Zellnetzwerk
   a. Das Zellnetzwerk besteht aus mehreren Netzwerken. ☐
   b. Das Verfahren versucht einen Mindestabstand zwischen den Positionen der Netzwerke zu garantieren. ☐
   c. Alle Anschlüsse der Netzwerke verbinden sich mit anderen Netzwerken. ☐

(Optional) Haben Sie Verbesserungsvorschläge?

Abbildung A.1.2.: Fragebogen der Kontrollgruppe, welche einen Text mit Abbildungen verwendet hat.

A.2. Informationstext

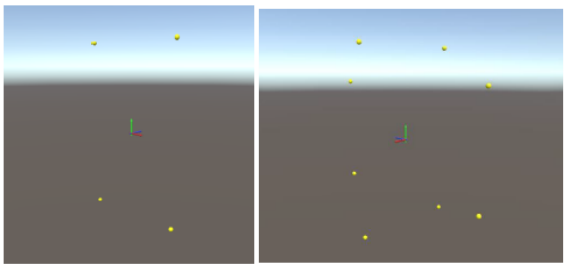
Die Bilder des Informationstexts wurden an dieser Stelle in einer geringeren Größe dargestellt, um Platzt zu sparen.

Informationen

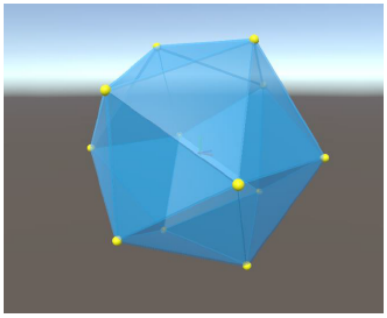
Im Folgenden wird ein Prozess beschrieben der ein 3D Modell eines Zellnetzwerk durch prozedurale Generierung erstellt. Versuchen Sie sich in der ihnen gegebenen Zeit mit dem Prozess vertraut zu machen um anschließend Fragen zu diesem beantworten.

Icosphere

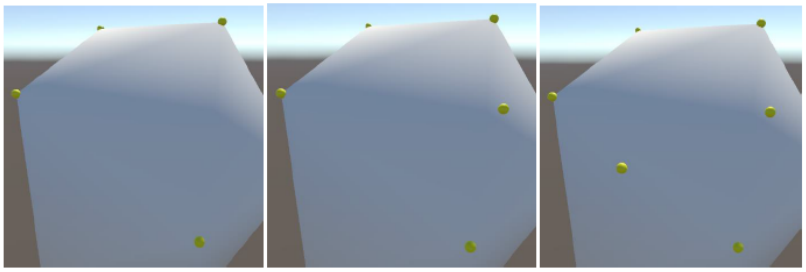
Es wird eine Icosphere, eine Kugel auf Basis eines 20-Seitigen Würfels, erstellt.



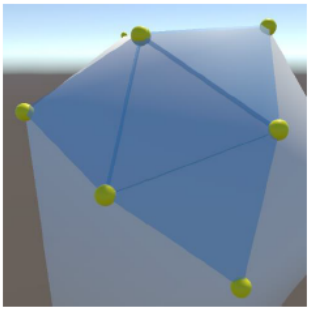
Die Eckpunkte (Vertices) werden an vorgegebenen Positionen platziert



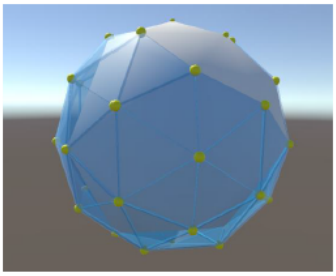
Die Vertices werden mit Dreiecken verbunden



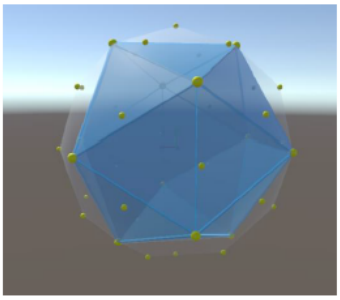
Es wird ein Vertex zwischen zwei bestehenden Vertices gebildet



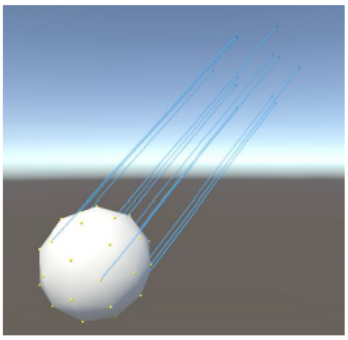
Mit den neuen Vertices werden aus jedem Dreieck vier kleinere Dreiecke gemacht



Der Prozess wird für alle Dreiecke der Icosphere wiederholt



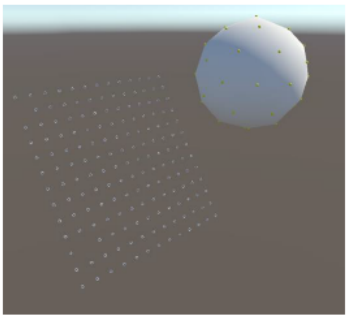
Die nicht unterteilten Dreiecke werden entfernt



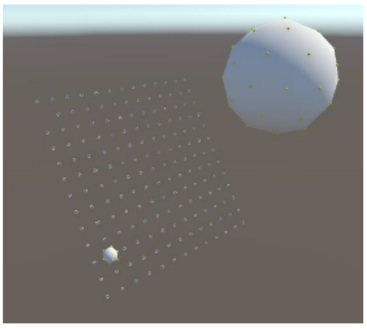
Die Vertices der Kugel werden dann zur gewünschten Position bewegt

Netzwerk

Ein Netzwerk besteht aus einer Icosphere im Ursprung, die sich mit Kurven mit weiteren Icospheres verbindet



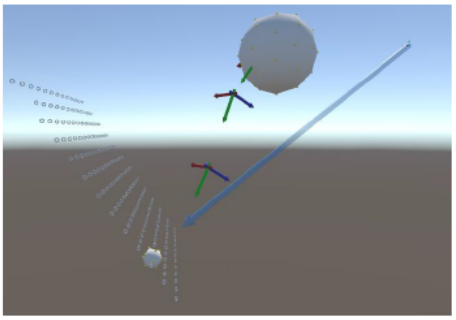
Ein Gitter aus potentiellen Positionen wird erstellt, und für jeden Endpunkt wird eine der Positionen Ausgewählt



Die Endpunkte werden mit Icospheres markiert

Kurve

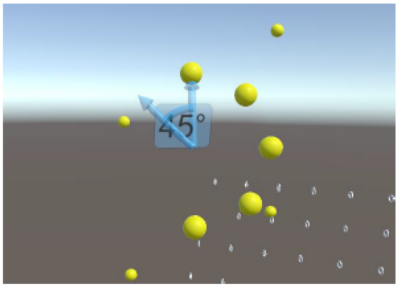
Die Kurven stellen die Verbindungen der Netzwerke dar



Punkte werden auf eine Kurve angeordnet

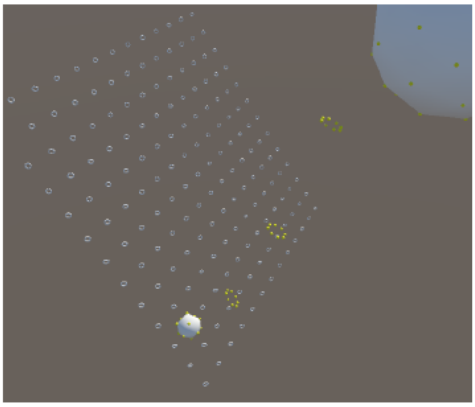
Kreis

Der Kreis ist die einfachste geometrische Form des Verfahrens

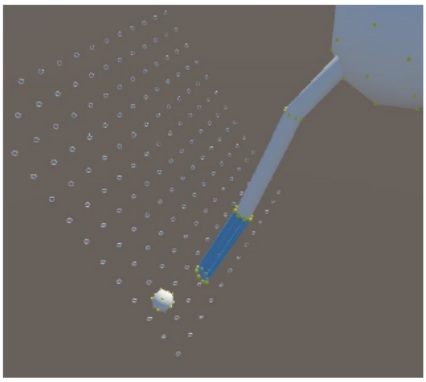


Jedes Vertice wird mit im selben Abstand vom Mittelpunkt plaziert. Der Vektor wird jedes mal um 360°/(Anzahl von Vertices) rotiert"

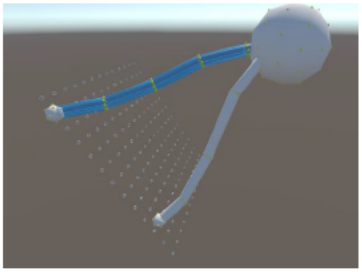
Kurve Fortsetzung



Es werden Kreise an jedem weiteren Kurvenpunkt erstellt



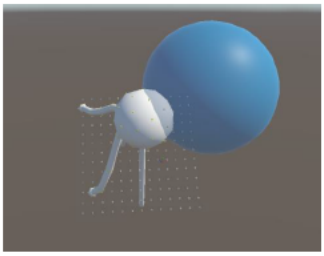
Die Kreise werden mit Dreiecken verbunden



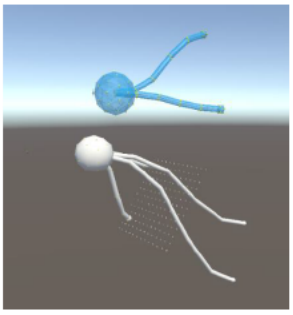
Die Endpunkte werden mit Kurven mit dem Anfang verbunden

Zellnetzwerk

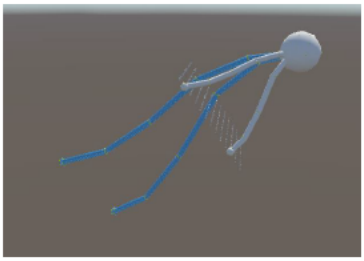
Das Zellnetzwerk wird aus den Netzwerken erstellt



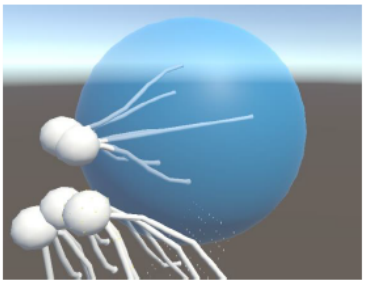
Es wird überprüft ob die gewählte Position zu nahe an einer vorhandenen ist



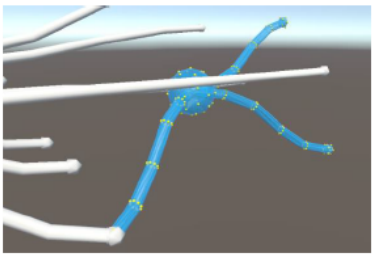
Die ersten Anschlüsse gehen zur zweiten Schicht



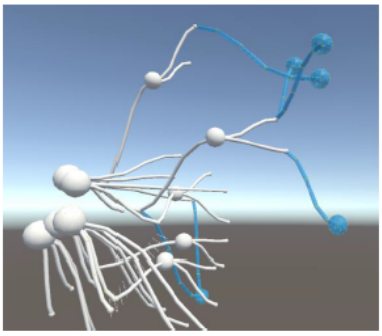
Die zweiten Anschlüsse gehen zur dritten Schicht



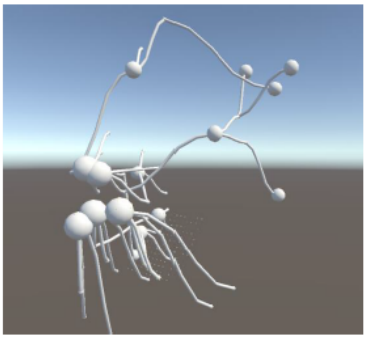
Es wird überprüft ob einer der Zielpunkte in Reichweite ist



Die zweite Schicht verbindet sich mit der ersten und dritten



Die dritte Schicht verbindet sich mit der ersten und der zweiten



Das Generieren ist abgeschlossen

B. Versuchsergebnisse und Rückmeldung

Die ausgefüllten Fragebögen der Gruppe sowie das Feedback dieser wurde für beide Gruppen festgehalten. Es wird hier nur angezeigt welche Fragen richtig beantwortet wurden und welche falsch. Der Inhalt der Fragen ist in Kapitel A zu finden.

B.1. Versuchsgruppe

Proband	1	2	3	4	5
Geschlecht	M	M	M	W	M
Alter	18-22	28-36	37-50	18-22	18-22
Erfahrung VR	Weniger als 1x pro Monat	1x pro Woche	Weniger als 1x pro Monat	1x pro Monat	1x pro Monat
Erfahrung Proz. Gen.	Keine	in einem Projekt	in mehreren Projekten	in einem Projekt	in einem Projekt
1a	X	X		X	X
1b	X		X		
2a					
2b	X	X	X	X	
3a		X			
3b	X		X	X	
4a	X	X	X	X	X
4b		X		X	X
4c		X			
5a	X		X	X	X
5b					
5c	X	X			
6a				X	
6b	X	X	X		X
6c				X	
Zeit	7min	11min	3min	5min	5min

Tabelle B.1.: Ergebnisse der fünf Probanden der Versuchsgruppe

Feedback	**Anzahl**
Betrachten von Text uns Visualisierung gleichzeitg schwierig	5
Zu wenig Zeit	4
Fortlaufen lassen durch Touchpad schwierig	4
Mehr Interaktion gewünscht	3
Parameter des Netzwerks verstellbar gewünscht	2
Zusätzlicher SegmenStep bei Zellnetzwerk gewünscht	1
Bedeutung von Zellnetzwerk unklar	1
Anwendung erweckt Eindruck von Tutorial anstatt vollwertiger Anwendung	1
Animation zwischen Steps	1
Fortbewegung in vertikale Richtung gewünscht	1
Mesh sollte bewegbar sein	1
Weitere Verwendung des Generierten gewünscht	1
Welche Zellen miteinander verbunden werden schwer erkennbar	1

Tabelle B.2.: Feedback der Versuchsgruppe nach Häufigkeit geordnet

B.2. Kontrollgruppe

Proband	1	2	3	4	5
Geschlecht	M	M	W	M	M
Alter	23-27	50 <	50 <	18-22	18-22
Erfahrung Proz. Gen.	Keine	Keine	Keine	Keine	Keine
1a					X
1b	X		X	X	X
2a	X			X	
2b		X	X	X	X
3a	X		X	X	X
3b					
4a				X	X
4b			X	X	
4c	X	X			
5a	X			X	
5b			X		X
5c					X
6a	X	X	X	X	
6b			X		
6c		X		X	
Zeit	5min	15min	5min	7min	5min

Tabelle B.3.: Ergebnisse der fünf Probanden der Kontrollgruppe

Feedback	**Anzahl**
Zu wenig Zeit	4
Text unklar	2
Unklar ob mehrere Antworten ankreuzbar	1

Tabelle B.4.: Feedback der Kontrollgruppe nach Häufigkeit geordnet

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Mittweida, den 3. September 2019

Noah Stolz