Hochschule Neu-Ulm
University of Applied Sciences

Bachelorarbeit

im Bachelorstudiengang

Game Produktion und Management

an der Hochschule für angewandte Wissenschaften Neu-Ulm

Thema

Workflow-Optimierung in der Spieleentwicklung: Entwicklung und Evaluation von Editor-Utilities zur Automatisierung von Arbeitsschritten in Unreal Engine 5

Erstkorrektor: Prof. Dr. Harald Gerlach

Zweitkorrektor: Prof. Dr. Stefan Faußer

Verfasser: Jannis Mack (303194)

Thema erhalten: 02.12.2025

Arbeit abgegeben: 03.02.2026

Hochschule Neu-Ulm
University of Applied Sciences

Gender Hinweis

Zur besseren Lesbarkeit wird in dieser Bachelorarbeit das generische Maskulin verwendet. Alle personenbezogenen Bezeichnungen gelten gleichermaßen für alle Geschlechter. Die gewählte Schreibweise dient der sprachlichen Vereinfachung und beinhaltet keinerlei Wertung.

Hochschule Neu-Ulm
University of Applied Sciences

Abstract

Die vorliegende Bachelorarbeit untersucht die Editor-Utilities in der Unreal Engine 5 und wie diese den Workflow der Spieleentwicklung verbessern und die Fehleranfälligkeit in diesem reduzieren können. Dazu wurden zwei Utilities entwickelt. Das erste automatisiert einen simplen Prozess des Kopierens von Skelett- und Knochenhierarchien in Data Assets. Das zweite dient zur Unterstützung des Level-Design-Prozesses, indem es eine Massenplatzierung von Static Meshes sowohl in der Luft als auch auf bestimmten Untergründen ermöglicht. Mit sechs Teilnehmenden wurde die Wirksamkeit der entwickelten Utilities empirisch untersucht. Als Messgrößen dienten Bearbeitungsdauer, dokumentierte Fehlerfälle, Entwicklungsdauer und subjektive Usability-Einschätzungen.

Die Ergebnisse zeigen, dass Editor-Utilities den Workflow in der Spieleentwicklung verbessern können, dass dies jedoch nicht in jedem Anwendungsfall zutrifft. Das erste entwickelte Utility verkürzte die Bearbeitungsdauer beim Export von Knochenhierarchien deutlich und reduzierte die Fehleranzahl vollständig auf null. Auch das Massenplatzieren von Assets in der Luft erwies sich mithilfe des Tools als effizienter als die manuelle Vorgehensweise. Demgegenüber führte der Einsatz des Tools bei Aufgaben, die eine präzise Platzierung von Assets auf bestimmten Oberflächen erforderten, zu einem erhöhten Zeitaufwand und teilweise zu einer höheren Fehlerquote.

Diese Ergebnisse verdeutlichen, dass der praktische Nutzen von Editor-Utilities maßgeblich davon abhängt, wie genau sie auf den jeweiligen Anwendungsbereich zugeschnitten sind. Tools mit zu umfangreicher oder komplexer Funktionalität können den Arbeitsprozess eher behindern als unterstützen. Komplexe Automatisierungen und zusätzliche Hilfestellungen sind daher insbesondere dann sinnvoll, wenn ein hoher Nutzungsumfang zu erwarten ist. Andernfalls kann der Entwicklungsaufwand den tatsächlichen Mehrwert übersteigen und die Effizienz des Gesamtprozesses beeinträchtigen.

Key Words: Unreal Engine 5, Editor-Utilities, Game Development, Workflow

Hochschule Neu-Ulm
University of Applied Sciences

Inhaltsverzeichnis

Hochschule Neu-Ulm
University of Applied Sciences

Abbildungsverzeichnis

Hochschule Neu-Ulm
University of Applied Sciences

Abkürzungsverzeichnis

AcAU	Actor Action Utility
AsAU	Asset Action Utility
BEP	Break-Even-Point
EUTME	Editor Utility Tool Menu Entry
EUW	Editor Utility Widget
IQR	Interquartilsabstand
PCG	procedural content generation
PIE	play in editor
SUS	System Usability Scale
UE	Unreal Engine
UE5	Unreal Engine 5
UI	User Interface
z. B.	zum Beispiel

Hochschule Neu-Ulm
University of Applied Sciences

Liste der Fachbegriffe

Asset	Einzelnes Element wie ein 3D Modell, mp4-Datei, Textur das im Content Drawer liegt
Blueprint System	Programmierumgebung der Unreal Engine, welche nicht per klassischem Code, sondern nur über Nodes benutzt wird
Data Asset	Speichert strukturierte Daten
FAB Marketplace	Offizieller Asset-Marktplatz von Epic Games
Foliage-Tool	Vegetationstool der Unreal Engine zur performanten und effizienten Platzierung von Assets
Kontextmenü	Rechtsklick-Menü im Editor; zeigt passende Aktionen zum Objekt an; Kontextabhängig
Level	Virtuelle Spielwelt / Szene die alle Actor und Objekte enthält
Material	Bestimmt das Aussehen von Oberflächen
Nanite	Virtuelle Geometrie-Technologie; erlaubt detailreiche Meshes und ist performant
Outliner	Liste aller Actor im Level; zum Finden und Auswählen von Objekten
Static Mesh	3D-Objekt; wird in Level platziert

Hochschule Neu-Ulm
University of Applied Sciences

1. Einleitung

Die Anzahl der Spieler, die gleichzeitig auf der weltweit größten PC-Spieleplattform *Steam* aktiv sind, nimmt kontinuierlich zu. Zuletzt wurden rund 13 Millionen gleichzeitige Spieler verzeichnet (*Steam Steam Charts (App 753)*, o. J.). Parallel dazu steigt auch die Anzahl der jährlich auf Steam veröffentlichten Spiele. Im Jahr 2024 wurde mit weltweit rund 15000 neu erschienenen Titeln ein Rekordwert erreicht, während 2020 lediglich 9204 erschienen sind (*Steam Annual Game Releases 2024*, 2025).

Die Anzahl der mit Unreal Engine (UE) entwickelten Spiele ist seit 2015 allmählich gestiegen. Der Anteil der veröffentlichten Spiele, die mit UE entwickelt wurden, erhöhte sich von 17 % im Jahr 2012 auf den bislang höchsten Wert 28 % im Jahr 2024. Besonders deutlich zeigt sich dieser Trend im Verkaufsanteil. Der Anteil der weltweit verkauften mit UE kreierten Spiele, lag im Jahr 2020 bei 15 % und stieg bis zum Jahr 2024 auf 31 % an, was einer Verdoppelung in nur vier Jahren entspricht. (Byshonkov, 2025).

Große Studios wie *CD Projekt*, die Entwickler hinter *The Witcher 3* und *Cyberpunk 2077*, haben angekündigt, ihre eigene interne Game Engine zugunsten der Unreal Engine aufzugeben. Als zentrale Gründe werden die steigenden Kosten für die Wartung und Aktualisierung der firmeneigenen Engine genannt. Mit Hilfe der UE soll die Entwicklungszeit verkürzt und die Effizienz gesteigert werden, da dieselbe Technologie künftig für mehrere Projekte gleichzeitig genutzt werden kann (*CD Projekt Will Swap REDengine for Unreal Engine 5 to Create the next Witcher Saga*, 2022).

Eine Möglichkeit, die Effizienz in der Entwicklung zu steigern, ist das Automatisieren des Unreal Editors selbst. Durch Editor-Utilities ist es möglich, dem Editor Funktionen zu geben, die genau auf die Anforderungen des Projektes zugeschnitten sind (*Accelerating Your In-Editor Workflows with Editor-Utilities | GDC 2024 | Talks and demos*, 2024). Sowohl UI als auch neue Funktionen können einfach erstellt und genutzt werden. So können wiederkehrende und monotone Aufgaben automatisiert und neue und optimierte Workflows entwickelt werden (*Scripting and Automating the Unreal Editor | Unreal Engine 5.6 Documentation | Epic Developer Community*, o. J.).

2. Stand der Forschung

Im Folgenden wird untersucht, welche Tools und Funktionen Spieleentwicklern in bestimmten Bereichen der Unreal Engine zur Verfügung stehen und welche Einschränkungen diese aufweisen. Anschließend wird erläutert, wie sich durch den Einsatz von Editor-Utilities neue Tools und Funktionen entwickeln lassen, um die bestehenden Grenzen in diesen Bereichen zu erweitern.

2.1. Level-Design

Zunächst wird der Aspekt des Level-Designs untersucht. Dabei werden Prozesse, die Entwickler beim Bau von Levels durchlaufen, analysiert. Zudem wird direkt ein möglicher Lösungsansatz für die Probleme aufgeführt.

2.1.1. Asset-Platzierung

Es existieren verschiedene Ansätze, um Assets aus dem *Content Drawer* in ein Level zu platzieren. Diese reichen von einfachen Drag-and-Drop-Methoden bis hin zu vollständig prozedural generierten Welten.

Das manuelle Platzieren von Assets ermöglicht ein hohes Maß an Präzision und bietet dem Entwickler vollständige gestalterische Kontrolle. Dadurch können Objekte gezielt positioniert werden, um durch subtile visuelle und auditive Hinweise die Spielerführung im Level zu unterstützen. Hierbei muss jedes Asset per Hand rotiert, skaliert und an die gewünschte Position gezogen werden.

Diese Methode stößt jedoch an ihre Grenzen, wenn großflächige Open-World-Umgebungen erstellt werden sollen. Das manuelle Platzieren einer großen Anzahl identischer Objekte, wie beispielsweise mehrerer tausend Bäume, ist nicht nur äußerst zeitaufwendig und damit kostenintensiv, sondern beeinträchtigt die Effizienz des Entwicklungsprozesses und die Performance des Spiels erheblich. In solchen Fällen ist der Einsatz automatisierter oder prozeduraler Platzierungsmethoden wesentlich zweckmäßiger, da eine schnellere und ressourcenschonendere Gestaltung umfangreicher Szenarien möglich ist.

Diese Methoden sind in der Unreal Engine 5 das *Foliage-Tool* und das *PCG-Tool (Procedural Content Generation)*. Beide Werkzeuge ermöglichen das prozedurale Platzieren von Assets

anhand definierter Kriterien. Sie sind besonders effektiv, wenn große Welten mit begrenzt und gezielt ausgewählten Assets zu füllen sind, wie zum Beispiel bei der Erstellung eines Waldes.

Vor allem PCGs erweisen sich als äußerst nützlich und effizient, wenn mehrere Level und Bereiche nach ähnlichen Kriterien gestaltet werden sollen. Sie können wiederverwendet und mit geringfügigen Anpassungen auf unterschiedliche Szenarien zugeschnitten werden, wodurch bereits kleine Änderungen große Auswirkungen erzielen können.

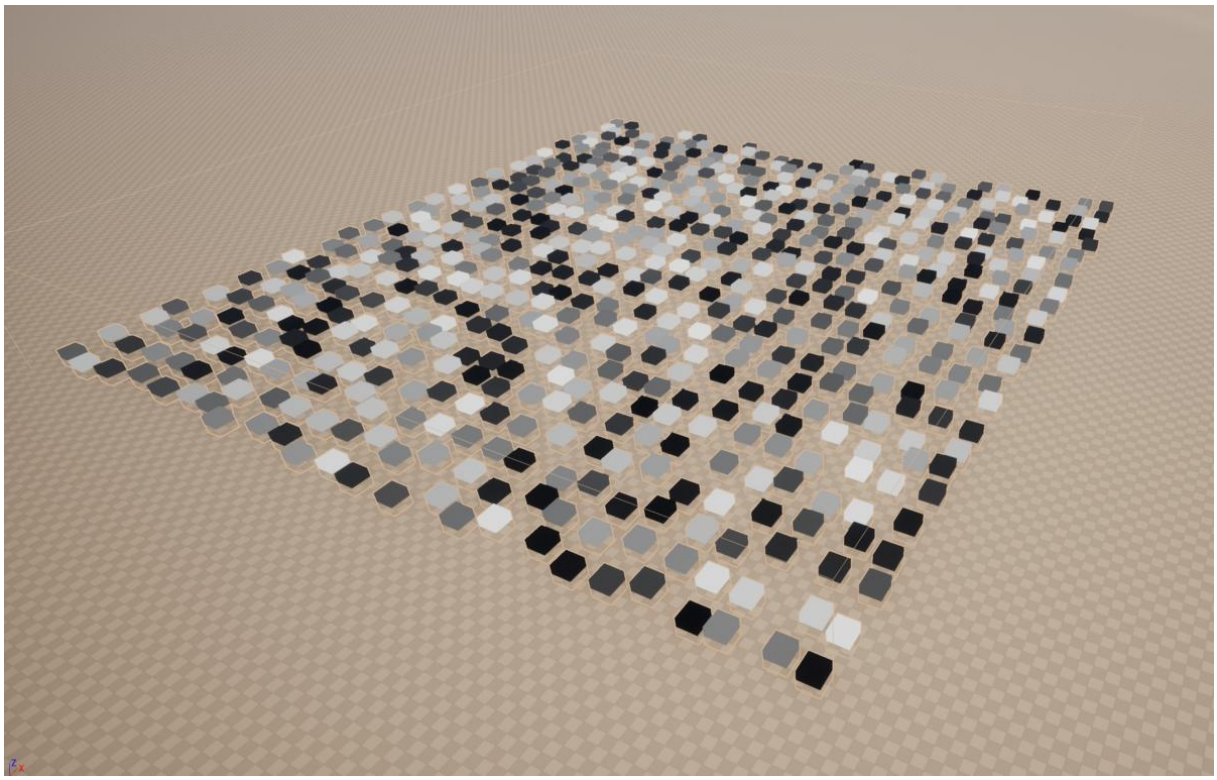


Abbildung 1: PCG-Beispiel
Quelle: https://unrealengine.hatenablog.com/entry/2023/05/27/221601

Die von diesem Tool platzierten Assets sind zudem sehr performant, da UE im Hintergrund zahlreiche Optimierungen vornimmt, um Entwicklern möglichst viel Freiheit bei der Menge der platzierbaren Assets zu gewähren. Ein weiterer Vorteil von PCGs besteht darin, dass sie sich dynamisch an Veränderungen im Level anpassen können. So müssen keine Assets per Hand neu platziert und bewegt werden.

Das *Procedural Foliage Tool* lässt den Entwickler eine Auswahl an Assets nach bestimmten Kriterien auf die Landschaft „malen“ (Abbildung 2: Procedural Foliage Tool Quelle: https://www.reddit.com/r/unrealengine/comments/u612k2/can_manually_place_foliage_but_cannot_paint/).

Abbildung 2: Procedural Foliage Tool
Quelle: https://www.reddit.com/r/unrealengine/comments/u612k2/can_manually_place_foliage_but_cannot_paint/

Der Entwickler kann wie auf einer Leinwand mit einem Pinsel Objekte in das Level malen. Die platzierten Assets sind auch hier performant. Allerdings passen sich die platzierten Objekte nicht an neue Veränderungen im Level an. Die platzierten Assets können nur mithilfe des Foliage-Tools angepasst werden. Somit sind präzise Anpassungen nur erschwert möglich. Jedes Areal muss mit Hand ausgemalt werden, was bei vielen Leveln oder Arealen zu einem hohen Zeitaufwand führt, der bei jeder neuen Veränderung im Level erneut bearbeitet werden muss.

Trotz dieser Vorteile bieten beide Tools nur begrenzte Kontrolle über die präzise Platzierung einzelner Assets. Das Platzieren von Objekten in der Luft gestaltet sich besonders komplex und zeitintensiv, da die Assets nicht mehr nur auf einem festen Untergrund platziert werden.

Um die begrenzte Kontrolle über einzelne Objekte und die Platzierung in der Luft zu verbessern, habe ich ein Tool entwickelt, das es Entwicklern ermöglicht, schnell und gezielt zahlreiche Objekte an frei wählbaren Positionen im dreidimensionalen Raum anhand von

ähnlichen Kriterien wie im Foliage-Tool zu platzieren. Zudem kann genau ausgewählt werden, auf welchem Untergrund diese Assets generiert werden sollen. Das erhöht die Kontrolle über die Orte, an denen Assets generiert werden können und macht die nachträgliche Bearbeitung der generierten Objekte einfacher, da jedes Objekt separat anklickbar ist. Das entwickelte Tool stellt somit eine flexible und zeitsparende Ergänzung zu den bestehenden Werkzeugen dar und kann in bestimmten Situationen effektiv genutzt werden.

2.1.2. Interaktion mit platzierten Objekten im Level

Um während des Level-Design-Prozesses mit Objekten zu interagieren, müssen diese entweder über den *Outliner* (Abbildung 4) oder durch direktes Anklicken im Level selektiert werden. Diese Methode ermöglicht eine präzise und kontrollierte Bearbeitung einzelner selektierter Assets. Wenn jedoch eine große Anzahl an Objekten bearbeitet werden soll, müssen diese manuell gesucht und jeweils einzeln ausgewählt werden.

Beispielsweise kann nicht unmittelbar ermittelt werden, wie viele Actor (platzierte Objekte in einem Level) eines bestimmten Typs im Level platziert worden sind. Diese müssen im Outliner oder direkt im Level gesucht werden, was zeitaufwendig ist und das Risiko birgt, manche Objekte zu übersehen.

Durch den Einsatz von *Editor Utility Widgets* kann dieser Prozess erheblich vereinfacht und beschleunigt werden. Mithilfe eines von mir entwickelten Tools lassen sich z. B. mit einem Mausklick alle Actors vom Typ Static Mesh eines bestimmten Assets (Abbildung 3; 1, 2) im Level finden und selektieren. Darüber hinaus kann eine Übersicht generiert werden, in der alle im Level vorhandenen Objekttypen aufgelistet sind (Abbildung 3; 3). Auf diese Weise ist unmittelbar erkennbar, welche Actors platziert worden sind. Diese können bei Bedarf gesammelt selektiert werden, um eine Massenbearbeitung zu ermöglichen.

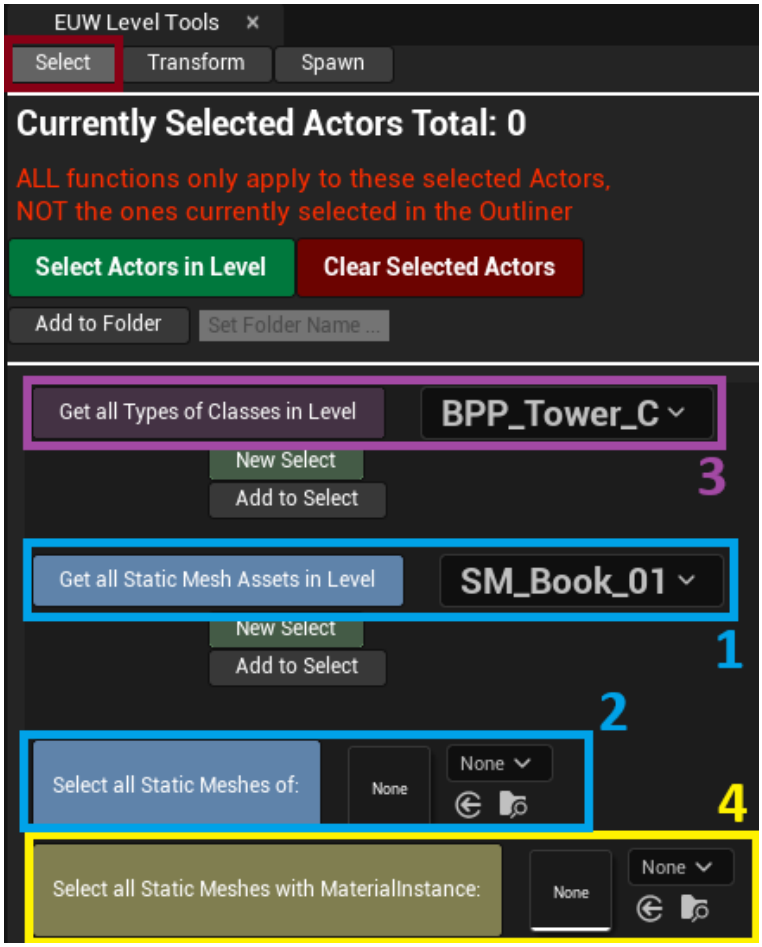


Abbildung 3: Select-Tool nummeriert

Ein indirekter Weg, Actors in einem Level zu finden, ist die Suche über die M*aterial Instance (*Abbildung 3; 4). Alle Static Meshes besitzen verschiedene Materials, die ihre Farbe und Beschaffenheit beschreiben. Mit dem neuen Tool kann direkt nach bestimmten Material Instances gesucht werden, wodurch im Anschluss alle platzierten Objekte, die dieses Material besitzen, auswählt werden. Das kann sehr nützlich sein, wenn man beispielsweise alle Objekte haben möchte, die das „Holz_1"-Material verwenden, um es an manchen Stellen auszutauschen.

Die Gefahr, manche Actors zu übersehen, läuft mit diesem Tool gegen null. Zudem kann gezielt nach beliebigen Objekttypen gesucht werden, was das Tool vielseitig einsetzbar macht.

Dadurch werden sowohl der Zeitaufwand als auch die Fehleranfälligkeit im Level-Design-Prozess reduziert.

2.1.3. Outliner Management

Ein weiterer Anwendungsbereich der automatischen Selektion besteht in der Kombination mit dem Outliner. Im Outliner sind alle im Level platzierten Objekte aufgelistet, wodurch dieser bei zunehmender Anzahl schnell unübersichtlich werden kann. Das gezielte Auffinden bestimmter Objekte gestaltet sich dadurch schwierig und geht mit einem erheblichen Zeitaufwand einher. Eine manuelle Organisation des Outliners ist zudem aufwendig und muss regelmäßig wiederholt werden, wenn neue Assets dem Level hinzugefügt werden.

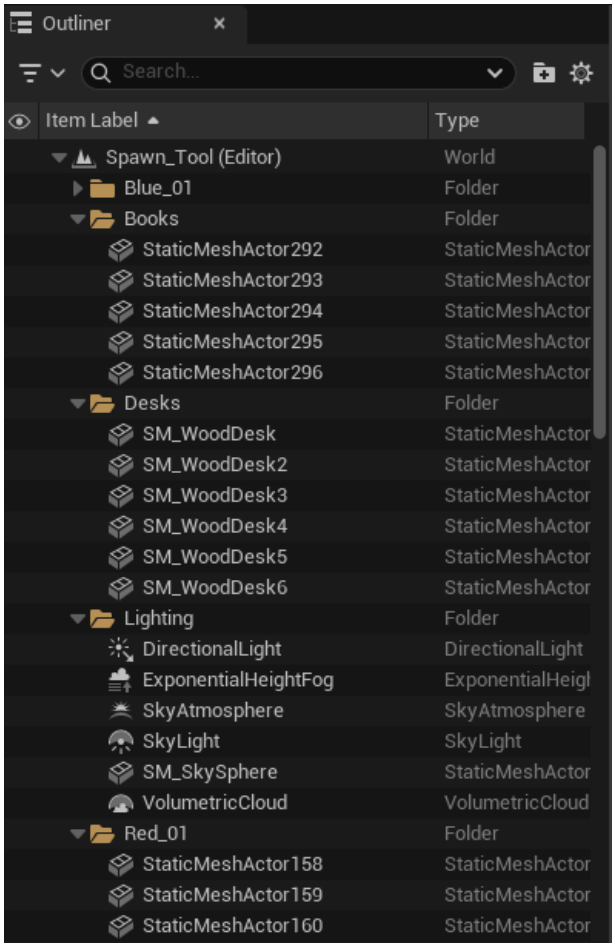


Abbildung 4: Outliner-Beispiel

Dieser Prozess kann ebenfalls durch Editor-Utilities automatisiert werden. So können beispielsweise alle Lichtobjekte über die Typensuche identifiziert und per Knopfdruck in einen gemeinsamen Ordner verschoben werden (Abbildung 5).

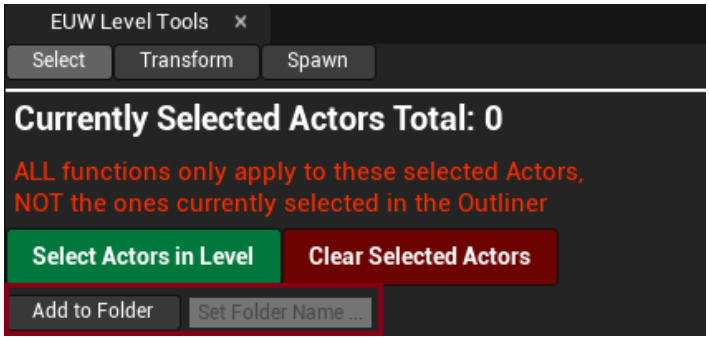


Abbildung 5: „Add to Folder“-Funktion

Darüber hinaus kann Code kreiert werden, der den gesamten Outliner anhand spezifischer Kriterien automatisch strukturiert und die Objekte in bestimmte Ordner einsortiert. Auf diese Weise lässt sich eine konsistente und effiziente Organisationsstruktur sicherstellen, die den Arbeitsaufwand reduziert und die Navigation im Level-Editor erheblich vereinfacht.

2.2. Asset Masseninteraktion im Content Drawer

Im Verlauf eines Projekts füllt sich der *Content Drawer* zunehmend. Dieser Bereich dient als zentrale Bibliothek des Projekts, in der alle Assets in ihrer Grundform gespeichert sind. Jedes Asset befindet sich an einem bestimmten Speicherort.

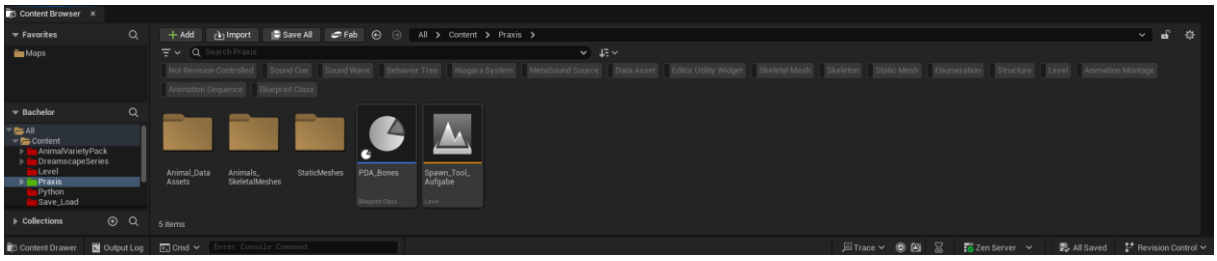


Abbildung 6: Content Drawer

Wenn mit einer großen Anzahl an Assets interagiert werden soll, beispielsweise um diese umzubenennen, müssen diese einzeln über ihren Namen gesucht oder nach ihrem Typ gefiltert

werden. Um die Assets an die Naming-Conventions anzupassen, war es zur UE-Version 5.4 erforderlich, jedes Asset manuell auszuwählen und einzeln umzubenennen. Dieser Prozess war zeitaufwendig und musste insbesondere nach dem Import neuer Assets regelmäßig wiederholt werden.

Durch die Entwicklung eines Editor Utility Widgets konnte dieser Prozess erheblich verbessert werden. Das Widget ermöglicht es, mehrere ausgewählte Assets gleichzeitig umzubenennen. So konnte ihnen ein bestimmtes Präfix oder Suffix hinzugefügt werden, ohne jedes Asset manuell umzubenennen. Dadurch wurde der Aufwand für die Einhaltung der standardisierten Namenskonventionen deutlich reduziert und die Konsistenz innerhalb des Projekts verbessert.

Seit der UE 5.4 ist ein vergleichbares Massen-Umbenennungstool (Abbildung 7) fester Bestandteil der Engine. Über das Kontextmenü können nun mehrere selektierte Assets gleichzeitig umbenannt werden, wodurch die zuvor manuell erstellte Lösung funktional in den Standardumfang der Engine integriert wurde (*Unreal Engine 5.4 Release Notes | Unreal Engine 5.4 Documentation | Epic Developer Community*, o. J.).

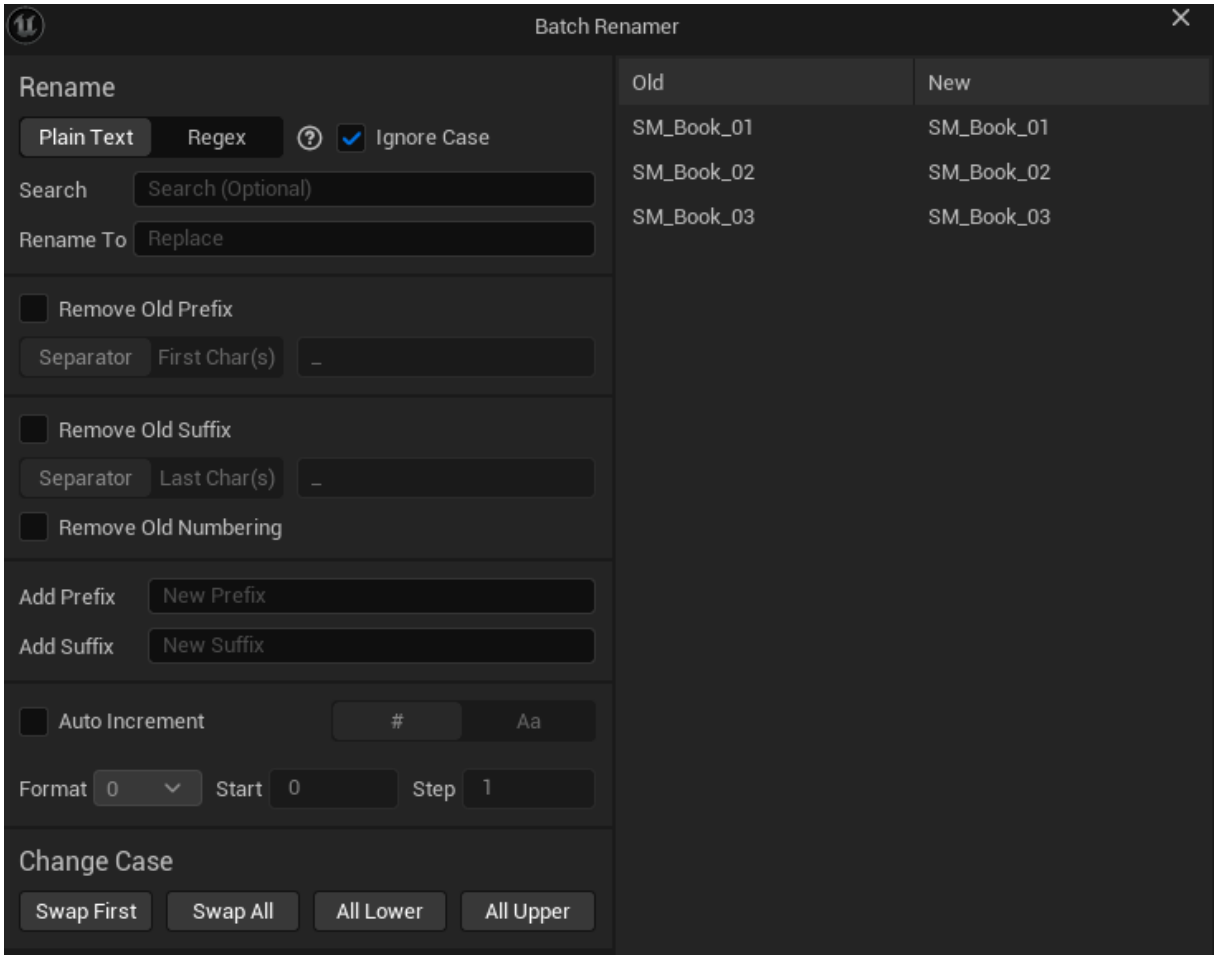


Abbildung 7: Massen-Umbenennungstool

Ein großer Aspekt der Spieleentwicklung ist das Befüllen von Daten zu spezifischen Datentypen. Im Falle der Unreal Engine heißen diese Datensätze *Data Assets*. In diesen werden zum Beispiel Eigenschaften von Items festgelegt, wie der Kaufpreis oder ähnliches. Das Befüllen dieser Data Assets erfolgt in der Regel manuell. Abhängig von Inhalt und Umfang der Daten kann dieser Vorgang einen erheblichen Zeitaufwand bedeuten. Durch den monotonen Prozess der Befüllung von Daten ist die Fehleranfälligkeit durch händisches Befüllen gegeben.

Durch den Einsatz von Editor Utility Widgets lässt sich dieser Prozess, je nach Anwendungsfall, automatisieren. Der Automatisierungsgrad ist abhängig von der Komplexität und Ähnlichkeit der Daten. Handelt es sich beispielsweise um einen Datentyp wie Schusswaffen, können typenspezifische Werte automatisch befüllt werden, während individuell variierende Eigenschaften wie Schaden oder Magazingröße pro Data Asset (also Waffe) manuell angepasst

werden müssen. Dadurch reduzieren sich Bearbeitungszeit und Fehleranfälligkeit, gleichzeitig wird die nachträgliche Anpassung der Daten erleichtert, da durch bereits kleine Anpassungen des Tools viele Datensätze gleichzeitig schnell überarbeitet werden können.

Für die Massenbearbeitung von Assets im Content Drawer eignen sich *Editor Utility Blueprints*, die mithilfe eigens entwickelter Funktionen, den als *Scripted Asset Actions* bezeichneten Operationen, eine automatisierte Datenverarbeitung ermöglichen.

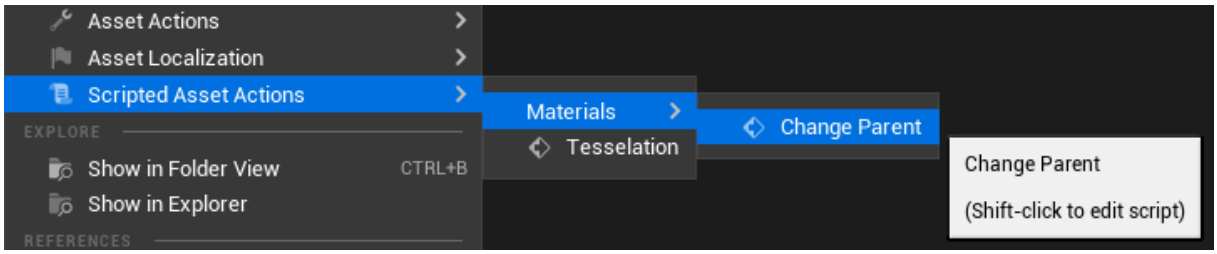


Abbildung 8: Scripted-Asset-Action-Beispiel

Zusätzlich besteht die Möglichkeit über C++-Skripte, Einstellungen einzelner Assets für Blueprints freizulegen, die normalerweise nicht direkt zugänglich sind. Diese Einstellungen können daraufhin innerhalb des Blueprint-Systems bearbeitet werden. Funktionen lassen sich so spezialisieren, dass sie nur für einen bestimmten Asset-Typ sichtbar und anwendbar sind.

Dadurch wird die Gefahr von Fehlanwendungen reduziert und die Benutzerfreundlichkeit verbessert. Scripted *Asset* Actions lassen sich auf Assets im Content Drawer anwenden, Scripted *Actor* Actions auf im Level platzierte Objekte. Beide werden über das Kontextmenü per Rechtsklick aufgerufen.

Dieser Ansatz umfasst auch den Asset-Typ *Actor*. Ohne Editor-Utilities ist eine gleichzeitige Bearbeitung mehrerer Actors nicht möglich. Häufig müssen jedem Actor manuell Eigenschaften oder bestimmte Komponenten zugewiesen werden. Das Vererbungssystem, das es erlaubt, bestimmte Eigenschaften von einem Blueprint Actor auf ein *Child* Actor automatisch zu übernehmen, mildert das Problem nur teilweise. Gemeinsame Eigenschaften können so zwar zentral gepflegt werden, jedoch erben nicht alle Actors voneinander, sodass weiterhin viele manuell angepasst werden müssen.

Mit Editor Utility Widgets lassen sich Funktionen entwickeln, die mehrere Blueprint Actor gleichzeitig modifizieren. So können z. B. automatisch Komponenten hinzugefügt werden oder Eigenschaften in bestimmten Actors verändert werden, ohne die Vererbungshierarchie zu beachten.

3. Methodik

In diesem Kapitel wird das methodische Vorgehen zur Überprüfung der Forschungsfrage beschrieben. Es beschreibt, wie das Experiment aufgebaut wurde, welche Teilnehmenden befragt wurden und welche Daten im Rahmen der Evaluation erhoben und ausgewertet wurden.

3.1. Forschungsfrage und Hypothese

Ziel dieser Arbeit ist es zu untersuchen, ob speziell entwickelte Editor-Utilities den Workflow in der Spieleentwicklung verbessern können. Editor-Utilities ermöglichen das Automatisieren wiederkehrender und zeitintensiver Aufgaben in der UE5 und sollen dadurch Entwicklungsprozesse effizienter machen. Mit dieser Automatisierung sollen sowohl der Zeitaufwand als auch die Fehleranfälligkeit reduziert und gleichzeitig die Benutzerfreundlichkeit gewährleistet werden. Im Rahmen dieser Arbeit werden zwei selbst entwickelte Utilities empirisch untersucht.

Das erste ermöglicht den Export von Knochenhierarchien beliebiger Skelette in Data Assets zur Verwendung in einem Kampfsystem.

Das zweite dient der automatischen Platzierung von Static Meshes innerhalb eines Levels, sowohl auf Oberflächen als auch im dreidimensionalen Raum.

Zusätzlich zur Nutzungsanalyse wird der erforderliche Entwicklungsaufwand berücksichtigt, um eine ökonomische Bewertung der erstellten Utilities vorzunehmen. Die Entwicklungszeit umfasst sämtliche Phasen der Tool-Erstellung, einschließlich Konzeption, Implementierung und Gestaltung der Benutzeroberfläche. Da die Versuchspersonen die Tools nicht selbst entwickeln, wird die Entwicklungszeit ausschließlich auf Grundlage meiner eigenen Entwicklungsaufzeichnungen ermittelt. Zur Bewertung des wirtschaftlichen Nutzens wird ein Break-Even-Point berechnet, der angibt, ab welcher Anzahl von Einsätzen die eingesparte Arbeitszeit die für die Entwicklung aufgewendete Zeit kompensiert.

Die zentrale Forschungsfrage lautet:

Verbessern die entwickelten Editor-Utilities in der Unreal Engine 5 die Effizienz, Fehleranfälligkeit und Benutzerfreundlichkeit bei typischen Prozessen in der Spieleentwicklung im Vergleich zur manuellen Durchführung?

Zur Beantwortung dieser Forschungsfrage werden im Rahmen der experimentellen Untersuchung folgende Hypothesen aufgestellt:

H1: Die Nutzung der Editor-Utilities reduziert die Bearbeitungszeit im Vergleich zur manuellen Durchführung.

H2: Die Verwendung der Utilities führt zu einer geringeren Anzahl an Fehlern bzw. Nachkorrekturen.

H3: Die Editor-Utilities werden von den Nutzern als benutzerfreundlich und intuitiv wahrgenommen.

Die Überprüfung dieser Hypothesen erfolgt anhand einer empirischen Evaluation mit sechs Testpersonen, die von mir gestellte Aufgaben sowohl manuell als auch mithilfe der entwickelten Tools bearbeiten. Die Aufgaben stellen typische Prozesse in der Spieleentwicklung dar. Die erhobenen Daten zu Zeit, Fehlern und subjektiver Usability-Bewertung bilden die Grundlage für die Beantwortung der Forschungsfrage.

3.2.Forschungsdesign

Das Forschungsdesign dieser Arbeit folgt einem quantitativen, experimentellen Ansatz mit ergänzenden qualitativen Elementen zur Bewertung der Benutzerfreundlichkeit der Utilities. Ziel ist es, den Nutzen der entwickelten Editor-Utilities in der UE5 empirisch zu untersuchen und deren Einfluss auf Arbeitszeit, Fehleranfälligkeit und Benutzerfreundlichkeit zu messen.

3.2.1. Stichprobe

Für die Tests werden sechs Probanden mit grundlegender Erfahrung im Umgang mit der Unreal Engine 5 ausgewählt. Die Teilnehmenden sind Studierende im Bereich Game Development. Eine Stichprobe von N = 6 ist ausreichend, um erste explorative Aussagen über Effizienz- und Usability-Verbesserungen zu treffen. Ziel ist nicht die statistische Generalisierbarkeit, sondern der direkte Vergleich zwischen manuellem Vorgehen und Nutzung der entwickelten Tools.

3.2.2. Untersuchungsaufbau

Jede Testperson erhält das vorbereitete Unreal-Engine-Projekt, das beide entwickelten Editor-Utilities enthält. Anschließend bearbeiten alle Personen drei typische Prozesse in der Spieleentwicklung in jeweils zwei Varianten: einmal mit einem konventionellen, manuellen Vorgehen und einmal unter Verwendung der entwickelten Utilities. Die Funktionsweise beider Tools wird nicht erläutert, damit überprüft werden kann, wie intuitiv und benutzerfreundlich sie wahrgenommen werden.

3.2.3. Messgrößen

Kategorie	Messgröße	Beschreibung / Messmethode
Effizienz	Bearbeitungszeit (in Minuten)	Zeit von Aufgabenstart bis Abschluss, gemessen mit Stoppuhr
Fehleranfälligkeit	Anzahl der Fehler	z.B. falsche Platzierungen, unvollständige Knochen-Export-Liste
Usability	Subjektive Bewertung	Nach jeder Aufgabe füllen die Teilnehmenden einen kurzen Fragebogen aus zu Verständlichkeit, Kontrolle, Zufriedenheit, Verbesserungsvorschläge
Entwicklungszeit	Entwicklungsdauer des Tools (in Stunden)	Zeit von Beginn der Konzeption bis zur fertigen Integrierung in die Editor Umgebung

3.2.4. Datenerhebung

Die Tests werden jeweils einzeln durchgeführt und per Screen-Recording aufgezeichnet. Die Bearbeitungszeiten werden manuell protokolliert. Fehler werden durch Beobachtung des Arbeitsprozesses sowie durch eine anschließende Analyse der Ergebnisse erfasst. Der Usability-Fragebogen wird digital ausgefüllt. Die Entwicklungszeit wird von mir per Zeitaufzeichnung ermittelt.

3.2.5. Datenanalyse

Die erhobenen Daten werden tabellarisch aufbereitet und ausgewertet. Für die Zeitmessungen werden Median, Interquartilsabstand (IQR) sowie die absolute und prozentuale Zeitersparnis zwischen manuellem und automatisiertem Vorgehen berechnet. Die Fehleranalyse erfolgt anhand eines Vergleichs absoluter und relativer Fehlermengen. Die Usability-Bewertungen werden über alle Teilnehmenden gemittelt und anschließend grafisch visualisiert. Es wird die durchschnittliche Zeitersparnis pro Nutzung ermittelt sowie die Anzahl der Nutzungen, die erforderlich sind, um den Break-Even-Point zu erreichen.

4. Ergebnisse zum Forschungsstand

Im folgenden Abschnitt werden Editor-Utilities und deren Funktionsweise beschrieben. Dabei werden die verschiedenen Unterkategorien erläutert und ihr jeweiliger Anwendungsnutzen dargestellt. Zudem wird aufgezeigt, in welchen Situationen welche Art von Editor-Utilities sinnvoll angewandt werden kann und welche Entwicklungsprozesse durch deren Verwendung vereinfacht oder beschleunigt werden.

4.1. Editor-Utilities

Mit Editor-Utilities lassen sich fehlende oder projektspezifische Funktionen der Unreal Engine selbst erstellen und erweitern. Dank des integrierten Blueprint-Systems ist es auch Personen ohne umfassende Programmierkenntnisse möglich, solche Utilities zu entwickeln und produktiv einzusetzen.

Grundsätzlich lassen sich Editor-Utilities in zwei Kategorien einteilen: *Editor Utility Blueprints* und *Editor Utility Widgets*. Beide Kategorien haben unterschiedliche Anwendungszwecke und ermöglichen jeweils spezifische Formen der Anpassung und Automatisierung innerhalb des Editors.

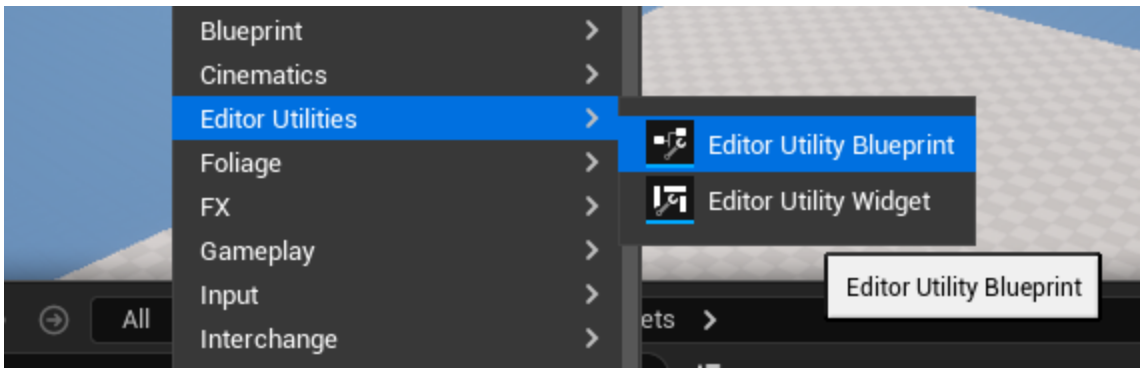


Abbildung 9: Editor Utility Erstellung

4.1.1. Editor Utility Widgets

Editor Utility Widgets sind vom Entwickler erstellte Benutzeroberflächen, die während der Entwicklungsphase eingesetzt werden. Sie erscheinen als eigenständige Fenster, die frei innerhalb der Editoroberfläche positioniert werden können und somit eine flexible Arbeitsumgebung ermöglichen.

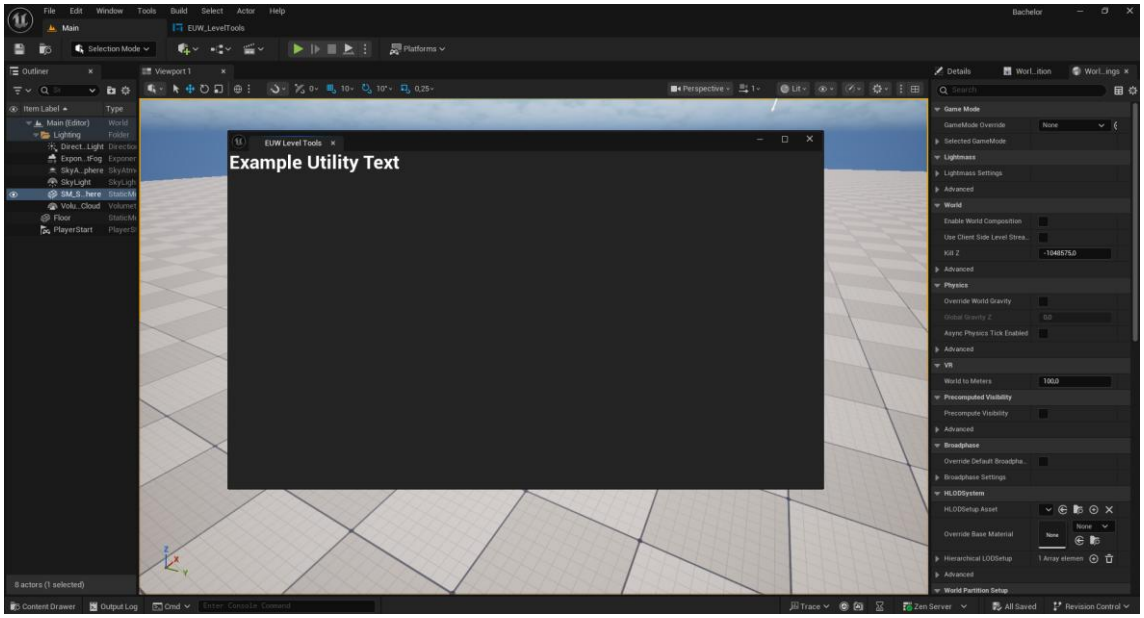


Abbildung 10: Leeres Editor Utility Widget

Diese Widgets können Logik enthalten, die sowohl während der Nutzung des Editors als auch im aktiven Spielmodus, den *Play in Editor,* (PIE) ausgeführt wird. Ihr primäres Ziel besteht darin, den Arbeitsprozess besonders im Level-Design zu optimieren. Im Gegensatz zu regulären User Widgets, die ausschließlich im PIE-Modus funktionsfähig sind, ermöglichen Editor Utility Widgets die Ausführung ihrer Funktionen direkt im Editor. Dies führt zu einer effizienteren Arbeitsweise, da weder Ladezeiten noch die zur Laufzeit aktive Logik der Assets berücksichtigt werden müssen.

4.1.2. Scripted Action Utilities

Scripted Actions sind Editor Utility Blueprints. Sie unterstützen darin, repetitive Aufgaben auf eine große Anzahl von Assets im Content Drawer und in Leveln anzuwenden. Darüber hinaus gibt es Asset Action Utilities und Actor Action Utilities.

Asset Action Utilities (AsAU) sind Editor Utility Blueprints, die direkt im Content Drawer ausgeführt werden. Durch einen Rechtsklick auf ein Asset können unter dem Menüpunkt *Scripted Asset Actions* definierte Funktionen aufgerufen werden. Da viele dieser Funktionen nur für bestimmte Typen an Assets relevant sind, kann in den Asset Action Utilities definiert werden, auf welche Asset-Kategorien sie angewendet werden dürfen.

Wird eine Funktion beispielsweise ausschließlich für Material Instances definiert, erscheint sie bei einem Rechtsklick auf einen Actor nicht. Dadurch wird die Benutzeroberfläche übersichtlich gehalten und potenziellen Fehlanwendungen vorgebeugt.

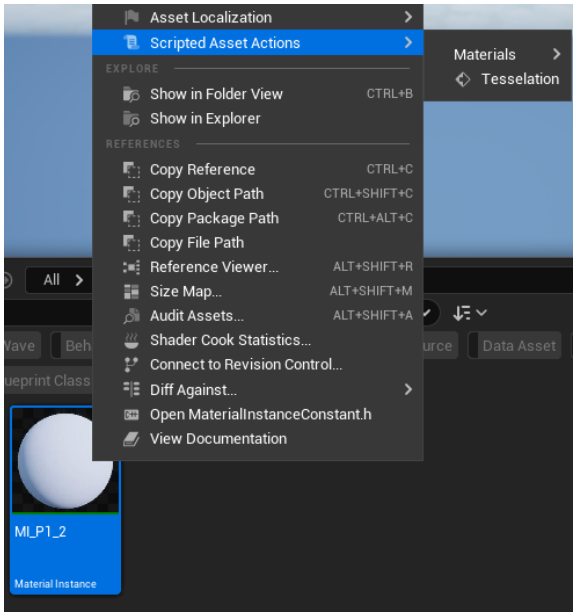


Abbildung 11: Asset mit einer Scripted Asset Action

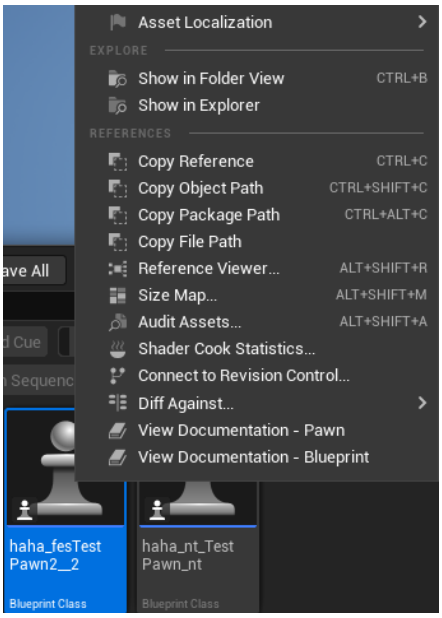


Abbildung 12: Asset ohne einer Scripted Asset Action

Die Logik dieser Tools hängt stark vom jeweiligen Anwe
Operationen wie das Aktivieren von Nanite umfassen oder den Zugriff auf Einstellungen ermöglichen, die üblicherweise nur über C++ verändert werden können. Die Logik ist meist sehr präzise auf einen bestimmten Asset-Typ ausgerichtet, um vor allem komplexere Aufgaben zu automatisieren. Auf diese Weise lassen sich spezifische Arbeitsprozesse direkt im Editor abbilden und erweitern.

Actor Action Utilities (AcAU) können ausschließlich auf im Level platzierte Objekte angewendet werden. Sie werden ebenfalls über einen Rechtsklick aufgerufen und unter dem Menüpunkt *Scripted Actor Actions* ausgeführt. Insbesondere für den Bereich des Level-Designs bieten AcAUs erhebliche Vorteile, da sie die Möglichkeit eröffnen, zusätzliche Funktionen zu implementieren, die wiederkehrende oder komplexe Arbeitsschritte automatisieren. Da sie auf sämtliche Objekte im Level angewandt werden können, entsteht ein großer Gestaltungsspielraum, der sowohl einfache als auch anspruchsvolle Arbeitsprozesse effizient

unterstützen kann. Dadurch werden der Entwicklungsprozess beschleunigt und die Arbeitsabläufe im Level-Design nachhaltig optimiert.

4.2. Anwendungsfälle

Im folgenden Abschnitt werden verschiedene Vorschläge und Ideen zur Nutzung dieser Utilities beschrieben. Diese dienen als Inspiration und Beispiel für die vielen Möglichkeiten Entwicklungsprozesse durch Editor-Utilities zu verbessern.

4.2.1. Automatisierung von repetitiven Aufgaben

Viele Spiele verfügen über ein Kampfsystem, in dem ein Charakter an unterschiedlichen Körperstellen getroffen werden kann. Der verursachte Schaden variiert dabei in Abhängigkeit von der getroffenen Stelle. Ein Treffer am Kopf verursacht beispielsweise einen höheren Schaden als ein Treffer am Fuß, während ein Treffer am Fuß zusätzliche Auswirkungen wie eine reduzierte Bewegungsgeschwindigkeit haben kann. Die Trefferzonen entsprechen in der Regel der Skelettstruktur des Charakters.

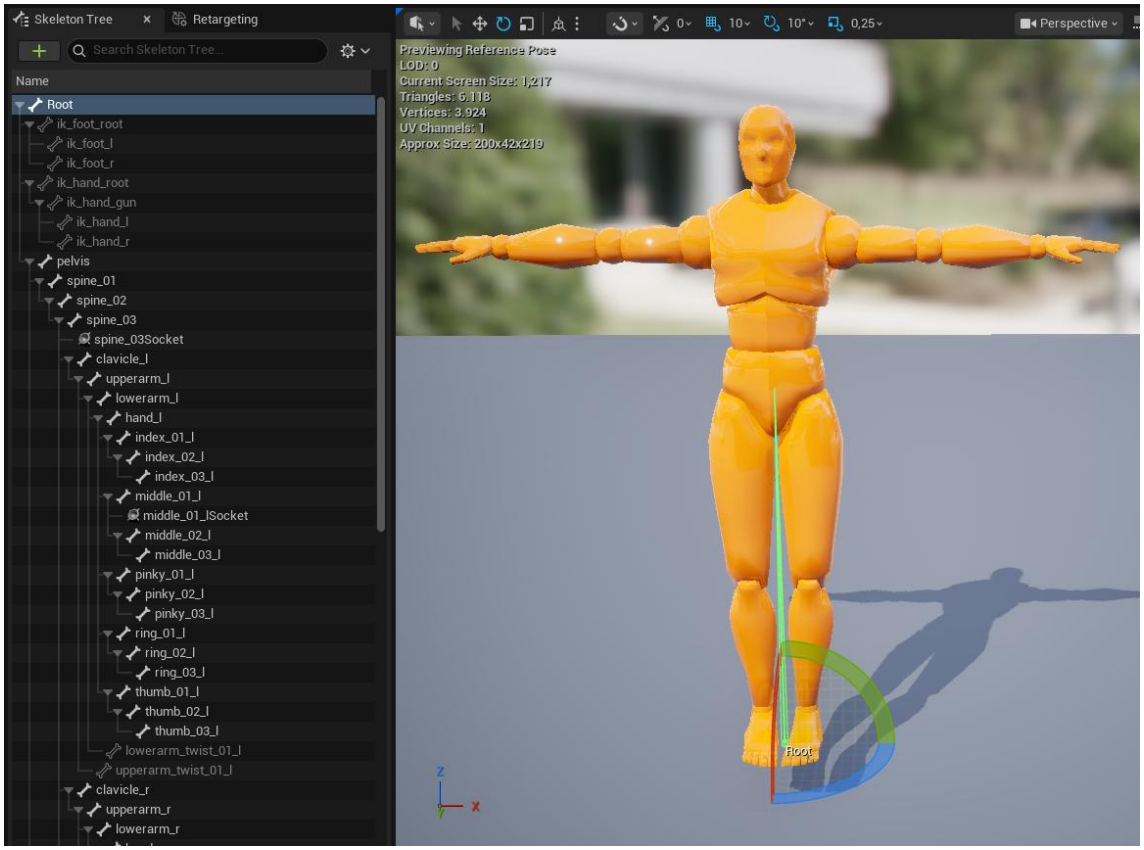


Abbildung 13: Knochenhierarchie-Beispiel

Wird eine Figur getroffen, wird überprüft, welcher Knochen des Skeletts betroffen ist. Anschließend wird die für diesen Knochen definierte Schadensverarbeitung ausgeführt. Mithilfe eines *Data Assets* können alle Knochenbezeichnungen zusammen mit den jeweiligen Schadensmultiplikatoren in einer Datei gespeichert werden. Dadurch lässt sich der Schaden für jeden einzelnen Knochen gezielt anpassen und zentral verwalten.

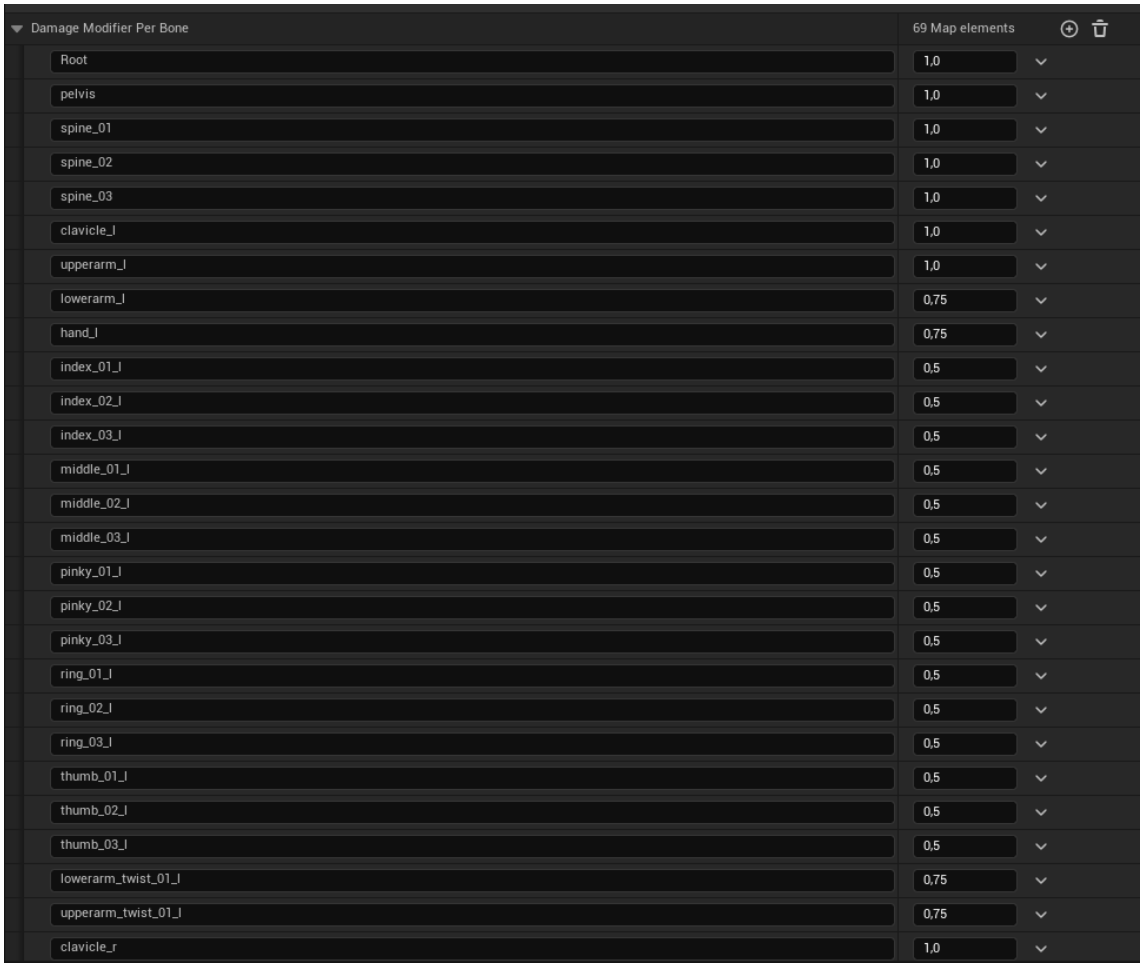


Abbildung 14: Data Asset mit übertragener Knochenhierarchie

In diesem Fall müsste für jeden der in diesem Beispiel (Abbildung 14) gezeigten 69 Knochen der jeweilige Name manuell kopiert und eingefügt sowie der zugehörige Schadensmultiplikator in das Data Asset eingetragen werden. Bei der Verwendung unterschiedlicher Skelette, etwa für verschiedene Tierarten, führt dieses Vorgehen zu einem hohen Zeitaufwand sowie zu repetitiver, fehleranfälliger Arbeit. Zudem besteht die Gefahr, dass einzelne Knochen versehentlich ausgelassen werden, was zu Unvollständigkeiten in der Datenstruktur führt. Die

nachträgliche Identifikation und Korrektur solcher Fehler erfordert zusätzlichen Aufwand. Auch die spätere Anpassung der Werte einzelner Knochen wird durch dieses manuelle Vorgehen erheblich erschwert.

Editor-Utilities können diesen Arbeitsprozess automatisieren. Durch ein Editor Utility Widget lassen sich per Knopfdruck alle Knochen eines Skeletts in ein Data Asset einfügen und anhand definierter Parameter automatisch mit Schadensmultiplikatoren versehen.

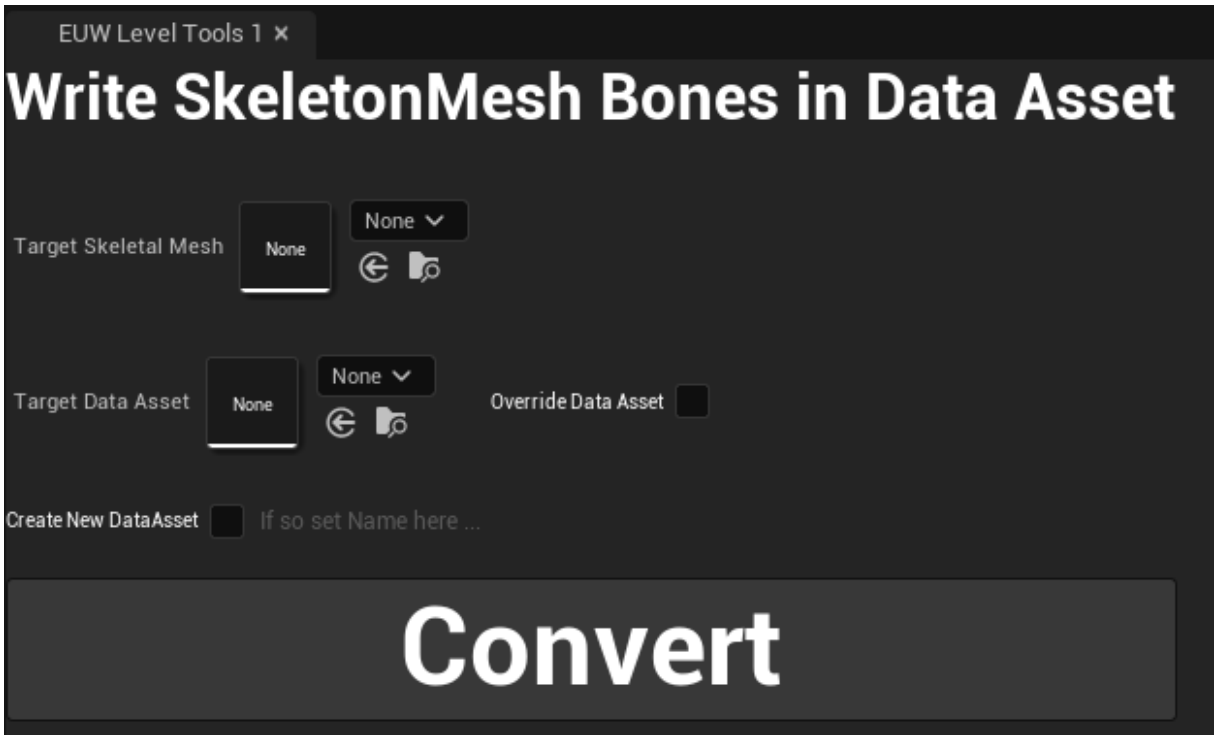


Abbildung 15: Knochen-Export-Utility

Dieses Verfahren funktioniert für alle Skelette, einschließlich tierischer Modelle. Potenzielle Fehlerquellen werden weitgehend eliminiert, da Anpassungen zentral im Code vorgenommen werden können, was die Wartung erheblich vereinfacht.

Darüber hinaus können bestehende Assets mit neuer Logik automatisch neu generiert werden, ohne dass manuelle Eingriffe erforderlich sind. Diese Methode führt zu einer deutlichen Zeitersparnis, reduziert die Fehleranfälligkeit und gewährleistet eine nachhaltige Anpassbarkeit der Datenstrukturen im weiteren Entwicklungsverlauf.

4.2.2. Optimierung des Level-Design Workflows

Beim Level-Design in der Unreal Engine 5 kann die Übersichtlichkeit des Arbeitsbereichs schnell abnehmen. Alle im Level platzierten Elemente werden im sogenannten *Outliner* aufgelistet (Abbildung 4).

Ohne strukturierte Sortierung, konsistente Benennung oder quantitative Informationen darüber, wie viele Objekte tatsächlich im Level vorhanden sind, besteht die Gefahr, den Überblick zu verlieren. Dies kann dazu führen, dass Assets versehentlich nicht platziert werden oder die räumliche Verteilung der Objekte unklar bleibt. Ein solcher Mangel an Struktur erschwert die Gestaltung kohärenter Levels und führt häufig zu zeitintensiven Iterationsprozessen.

Zwar bietet die Unreal Engine die Möglichkeit, den Outliner manuell zu organisieren, beispielsweise durch das Anlegen von Ordnern oder das Umbenennen einzelner Elemente. Dieser Prozess ist aber arbeitsintensiv, muss regelmäßig wiederholt werden und skaliert nicht mit der Größe des Levels. Mit zunehmender Komplexität des Projekts steigt der organisatorische Aufwand erheblich, wodurch wertvolle Entwicklungszeit verloren geht.

Darüber hinaus stellt Unreal Engine keine native Funktion zur Verfügung, um den Outliner nach konkreten Assets, etwa nach einem bestimmten Buch-Asset, zu durchsuchen, wenn es nicht entsprechend benannt ist. Das Auffinden spezifischer Objekte ist daher nur direkt im Level möglich. Dies erhöht die Wahrscheinlichkeit, einzelne Elemente zu übersehen, und führt zu ineffizienten Arbeitsabläufen.

Durch den Einsatz von Editor-Utilities lassen sich diese Probleme weitgehend automatisieren und somit effizient beheben. Die Fehleranfälligkeit wird reduziert, der Zeitaufwand erheblich verringert, und die entwickelten Tools bleiben unabhängig von der Levelgröße leistungsfähig und zuverlässig.

Mit dem entwickelten Editor Utility Tool können auf Knopfdruck alle im Level vorhandenen Klassentypen oder verwendeten Static Mesh-Assets automatisch ermittelt und in einer Liste angezeigt werden. Diese Klassen können anschließend direkt selektiert und bearbeitet werden. Ein Prozess, der ohne ein entsprechendes Tool mehrere Minuten in Anspruch nehmen würde, lässt sich dadurch in wenigen Sekunden durchführen.

In der folgenden Abbildung 16 ist zu erkennen, dass im gesamten Level genau ein Actor der Klasse *BP_Candle_01_C* vorhanden ist (Abbildung 16; 1). Dieser wurde über die Funktion *New Select* (Abbildung 16; 2) identifiziert und ausgewählt und kann nun bei Bedarf gezielt bearbeitet werden.

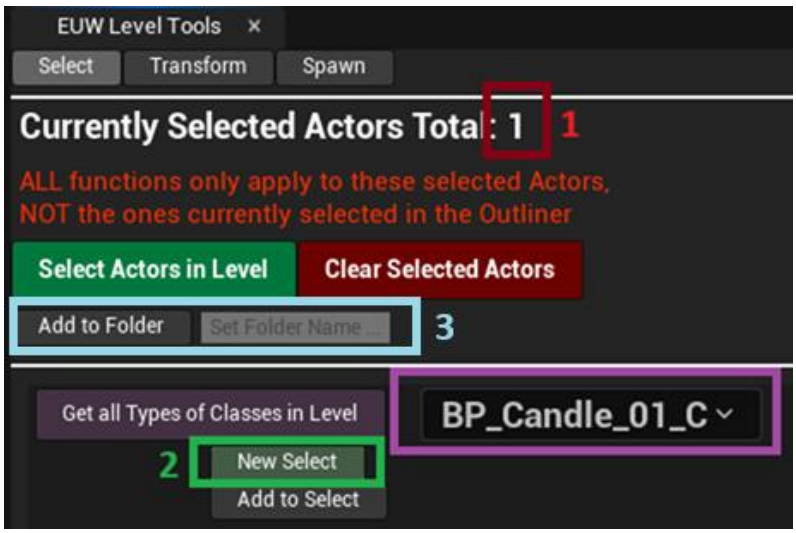


Abbildung 16: Selektion aller BP_Candle_01_C Actor im Level

Darüber hinaus ermöglicht das Tool die automatische Auflistung aller im Level verwendeten Static Mesh-Assets. Auf diese Weise wird unmittelbar ersichtlich, welche Assets im aktuellen Level eingesetzt wurden (Abbildung 17).

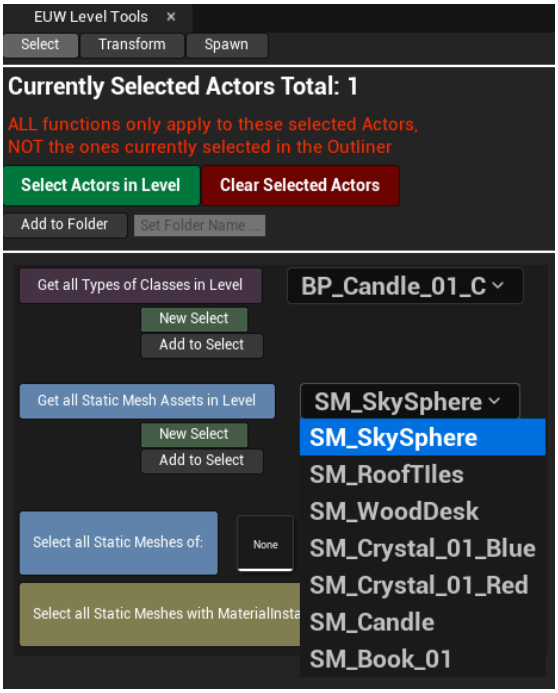


Abbildung 17: Alle platzierten Static Mesh Assets im Level

Im gezeigten Beispiel wurden alle Static Mesh Actors ausgewählt, die das Asset *SM_Crystal_01_Blue* verwenden. Mithilfe der Funktion *Add to Folder* (Abbildung 16; 3) können alle selektierten Objekte anschließend automatisch in einem neu erstellten Ordner organisiert werden, beispielsweise unter dem Namen „*Blue Crystals*“. Dadurch lässt sich das Level mit nur wenigen Arbeitsschritten strukturiert organisieren, während gleichzeitig eine konsistente Übersicht über alle platzierten Assets gewährleistet bleibt.

Das Tool stellt zwei spezialisierte Suchfunktionen zur Verfügung, die eine gezielte Analyse und Bearbeitung von Objekten im Level ermöglichen. Die erste Funktion erlaubt die Suche nach bestimmten Static Mesh Assets und identifiziert automatisch alle Objekte, die dieses Mesh referenzieren, sodass sie direkt selektiert werden können (Abbildung 3; 2).

Die zweite Funktion, *Select all Static Meshes with MaterialInstance (Abbildung 3; 4)*, ermittelt und markiert alle Objekte, die eine ausgewählte *Material Instance* verwenden. Dies ist insbesondere dann von Bedeutung, wenn *Materials* innerhalb einzelner Objekte gezielt angepasst oder ausgetauscht werden sollen, ohne dass die Änderungen global auf sämtliche Instanzen angewendet werden.

5. Implementierung

Im folgenden Abschnitt wird veranschaulicht, wie der Unreal Editor mithilfe von Editor-Utilities erweitert und an spezifische Arbeitsabläufe angepasst werden kann. Zudem werden die im Rahmen der empirischen Untersuchung eingesetzten Tools sowie der Versuchsleitfaden für die Teilnehmenden vorgestellt. Abschließend werden die Aufgaben beschrieben, die von den Teilnehmenden jeweils manuell und mithilfe der entwickelten Tools bearbeitet werden sollten.

5.1. Anpassung und Erweiterung des Unreal Editors

Im Rahmen dieser Arbeit wird der Unreal Editor gezielt erweitert, um den Zugriff auf entwickelte Editor-Utilities zu vereinfachen und deren Nutzung in den Arbeitsalltag zu integrieren. Eine besondere Schwierigkeit besteht darin, entwickelte Tools so bereitzustellen, dass sie nahtlos in die Benutzeroberfläche eingebunden sind und unabhängig von ihrer internen Struktur oder Benennung effizient verwendet werden können.

Um die erstellten Editor Utility Widgets nutzen zu können, müssen diese standardmäßig im Content Drawer manuell gesucht und ausgeführt werden. Dieses Vorgehen setzt voraus, dass die genauen Bezeichnungen oder Speicherorte der Widgets bekannt sind. In der Praxis ist dies jedoch häufig nicht der Fall, da viele Nutzende weder die Namen der vorhandenen Widgets kennen noch einen Überblick über deren Umfang haben.

Zu diesem Zweck werden die Editor Utility Widgets als Schaltflächen direkt in die Editoroberfläche integriert (Abbildung 19). Durch das Anklicken einer Schaltfläche öffnet sich das jeweilige Widget in einem separaten Fenster, das frei innerhalb der Editoroberfläche positioniert werden kann. Für die Nutzung ist somit lediglich bekannt, dass entsprechende Schaltflächen existieren. Detaillierte Kenntnisse über Namen oder Speicherorte der Widgets sind nicht erforderlich.

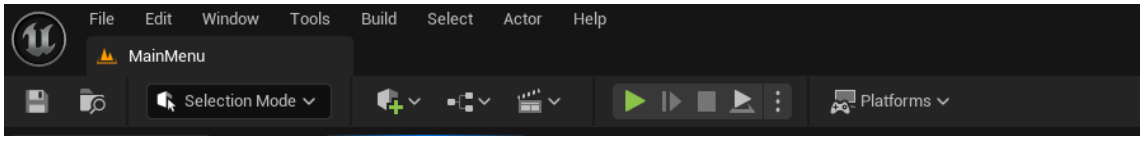


Abbildung 18: Editoroberfläche ohne Anpassung

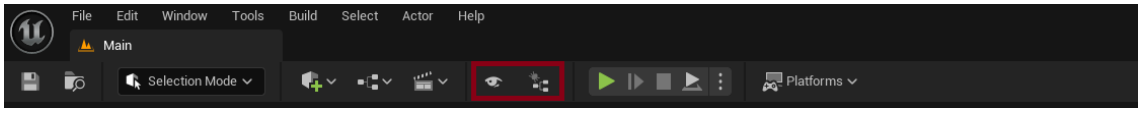


Abbildung 19: Editoroberfläche mit Anpassung

Diese Form der Integration stellt sicher, dass alle entwickelten Werkzeuge jederzeit zugänglich sind und ohne zusätzliche Einarbeitung verwendet werden können. Gleichzeitig wird der Workflow verbessert, da wiederkehrende Such- und Öffnungsschritte entfallen.

Diese Integration erfordert eine gezielte Anpassung der Editoroberfläche, die im nächsten Schritt technisch umgesetzt wird. Im Folgenden werden die verwendeten Wege und Tools vorgestellt, mit denen die Editor Utility Widgets als feste Bestandteile der Unreal-Editor-Oberfläche realisiert werden können.

Es bestehen mehrere Möglichkeiten, den Unreal Editor zu erweitern, unter anderem durch die Verwendung von C++, Python-Skripten oder Blueprint-basierten Ansätzen. In dieser Arbeit wurde ein Blueprint-basierter Ansatz gewählt, da dieser einen geringen Implementierungsaufwand erfordert und eine schnelle Iteration ermöglicht, ohne zusätzlichen Quellcode schreiben zu müssen.

Das Hinzufügen von Schaltflächen zur Editor-Toolbar erfolgt über sogenannte *Editor Utility Tool Menu Entry* Blueprints. Um die gewünschte Position der Schaltfläche innerhalb der Editoroberfläche zu bestimmen, kann der Konsolenbefehl *ToolMenus.Edit 1* ausgeführt werden, welcher eine Übersicht der verfügbaren Menü- und Toolbar-Bereiche bereitstellt. Nach Auswahl des entsprechenden Bereichs wird die Bearbeitungsansicht mit *ToolMenus.Edit 0* wieder deaktiviert.

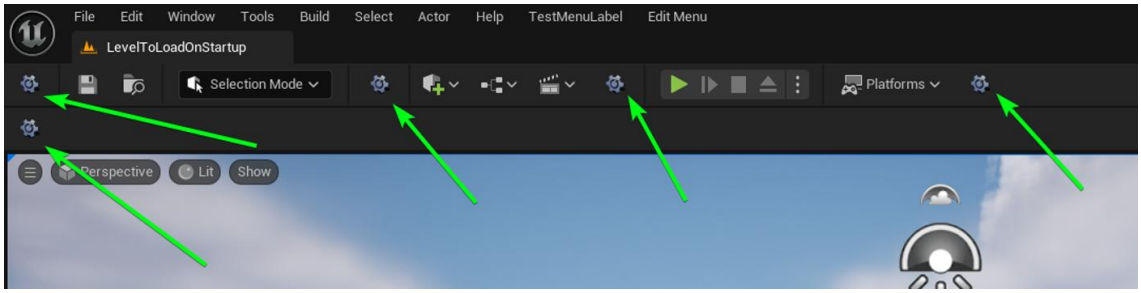


Abbildung 20: Editoroberfläche Anpassungsoptionen. Quelle: https://dev.epicgames.com/community/learning/tutorials/owYv/unreal-engine-getting-started-with-editor-utility-blueprints

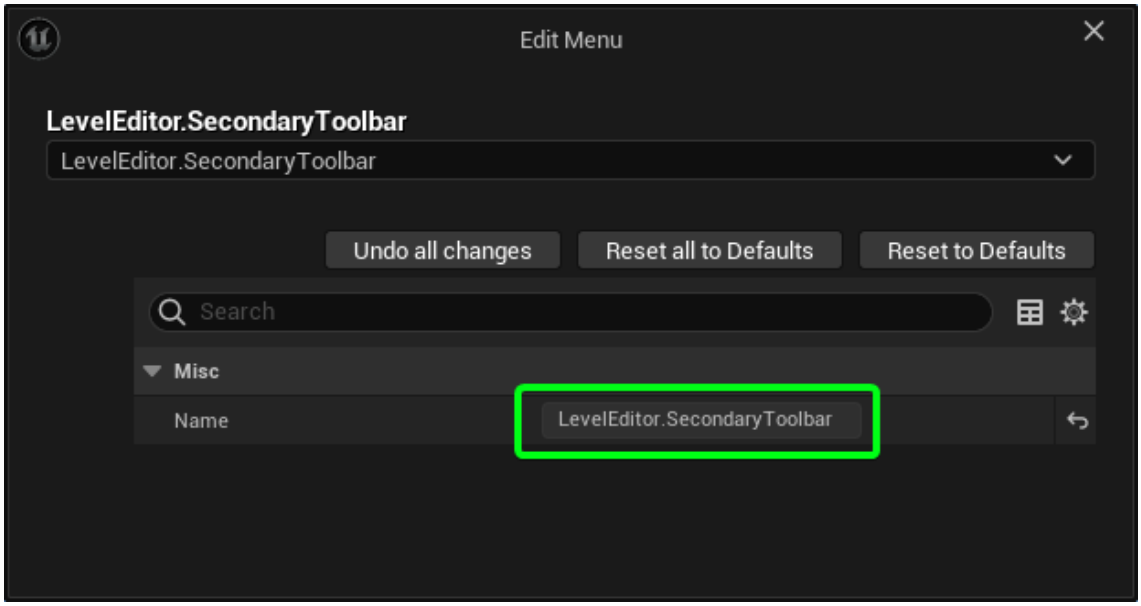


Abbildung 21 Name einer Schaltflächenposition im Editor. Quelle: https://dev.epicgames.com/community/learning/tutorials/owYv/unreal-engine-getting-started-with-editor-utility-blueprints

Anschließend wird ein neues *Editor Utility Tool Menu Entry* Blueprint erstellt, in dem die erforderlichen Einstellungen vorgenommen werden. In den *Class Defaults* wird unter anderem der Menübereich definiert, in dem die Schaltfläche erscheinen soll. Dieser Bereich wurde im vorherigen Schritt (Abbildung 21) ermittelt. Weitere Parameter wie Beschriftung, Tooltip und Icon können optional ergänzt werden, um die Bedienbarkeit weiter zu verbessern.

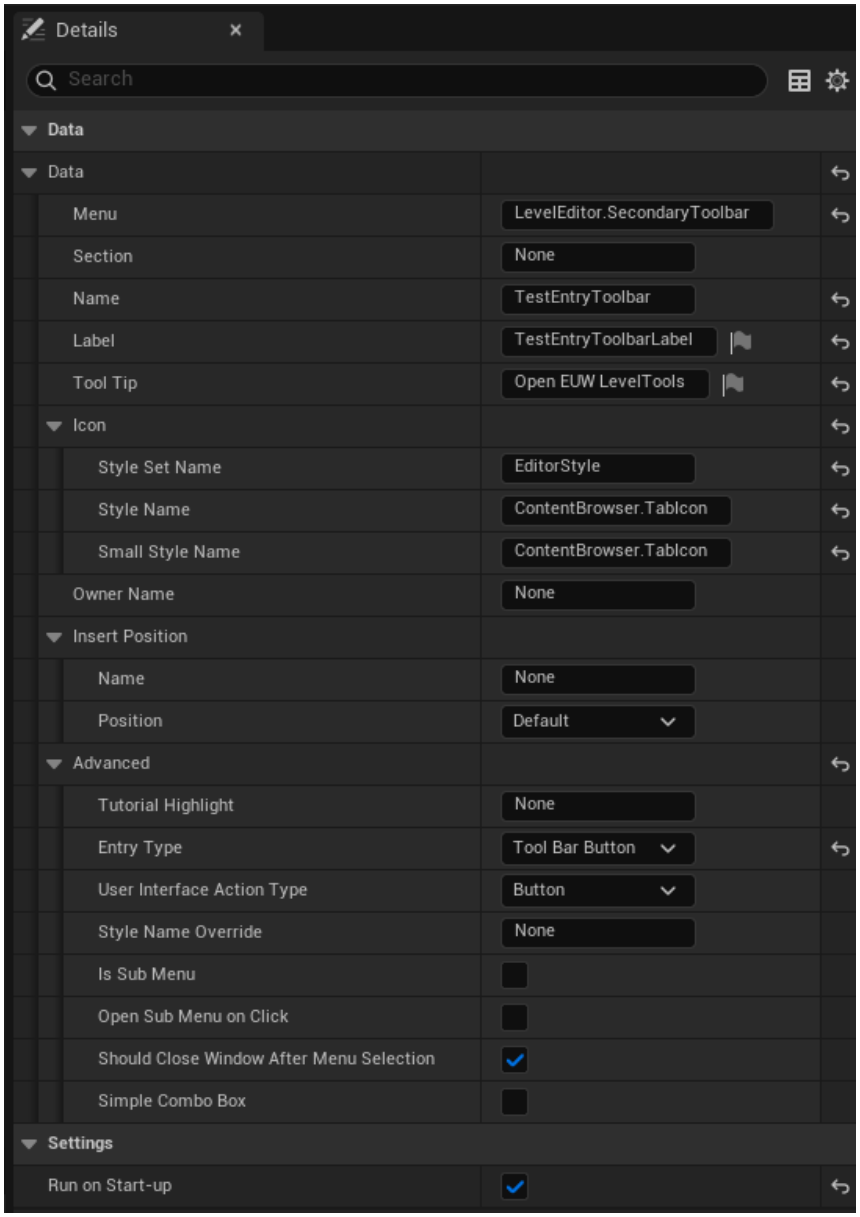


Abbildung 22: Editor Utility Tool Menu Entry

Nachdem die Schaltflächen erfolgreich in die Editoroberfläche integriert wurden, stellt sich im nächsten Schritt die Frage, welche Funktionalität durch diese ausgelöst wird. Die zentrale Logik eines EUTME wird innerhalb der überschreibbaren Funktionen des Blueprints definiert. Die Funktion *Run* wird beim Initialisieren des Menu Entry ausgeführt und sorgt dafür, dass die Schaltfläche im Editor verfügbar ist. Die eigentliche Aktion, die beim Anklicken des Buttons ausgelöst wird, wird über die Funktion *Event Execute* implementiert. Im vorliegenden Fall wird

beim Auslösen des Events das zugehörige Editor Utility Widget geöffnet. Dieses erscheint in einem separaten Fenster, das unabhängig von der Toolbar innerhalb der Editoroberfläche positioniert werden kann.

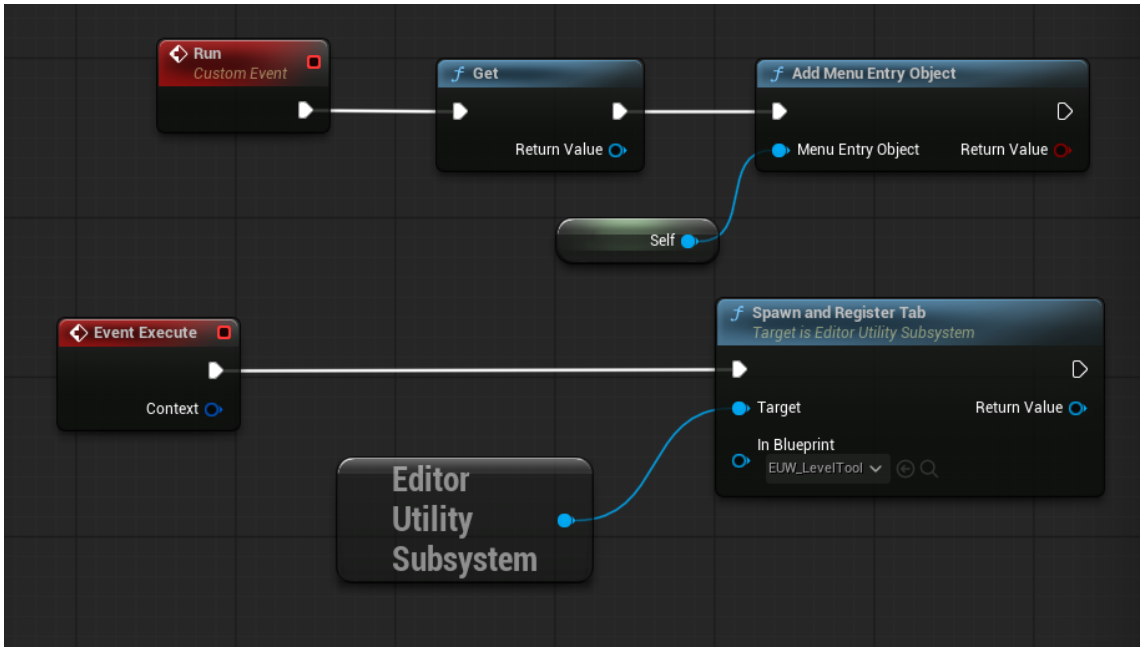


Abbildung 23: Beispiel-Blueprint eines Editor Utility Tool Menu Entry Blueprint

5.2. Leitfaden

Im Folgenden wird der praktische Einsatz der entwickelten Tools im Rahmen der empirischen Untersuchung dargestellt. Dazu werden zunächst die beiden implementierten Tools vorgestellt und anschließend die konkreten Aufgaben beschrieben, die die Teilnehmenden sowohl manuell als auch mithilfe der Tools bearbeiten sollen.

Die beiden in dieser Arbeit entwickelten Editor-Utilities (im Folgenden *Utility A* und *Utility B*) wurden in den Unreal Editor integriert und für die Evaluation in einem Template-Projekt bereitgestellt. *Utility A* ermöglicht den automatischen Export von Skelett- und Knochenhierarchien in Data Assets, *Utility B* dient der Massenplatzierung von Static Meshes zur Beschleunigung typischer Level-Design-Aufgaben.

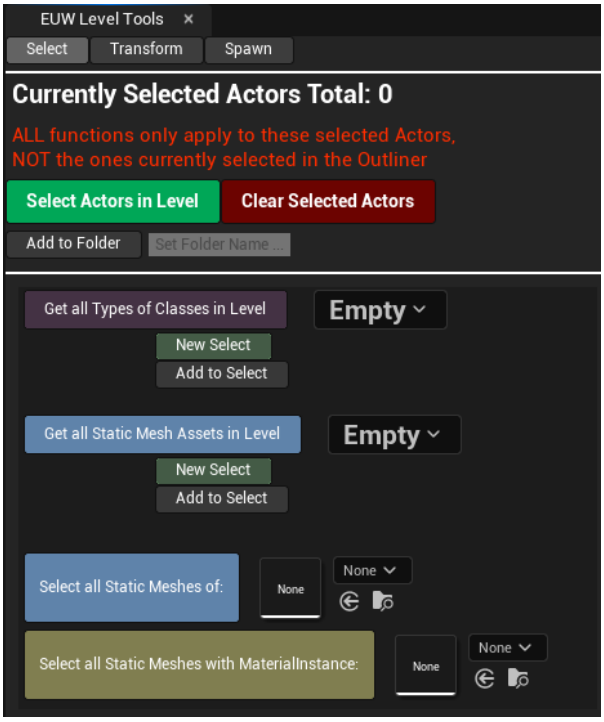


Abbildung 24: Utility B: Select-Tool

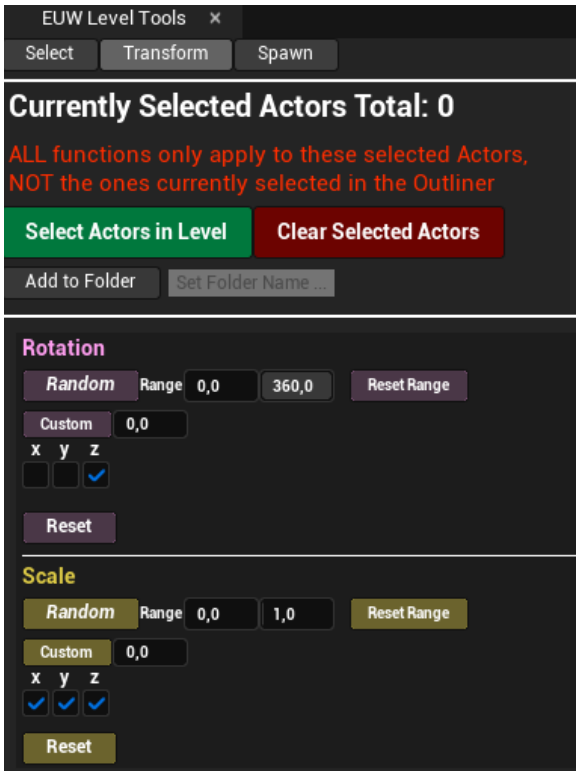


Abbildung 25: Utility B: Transform-Tool

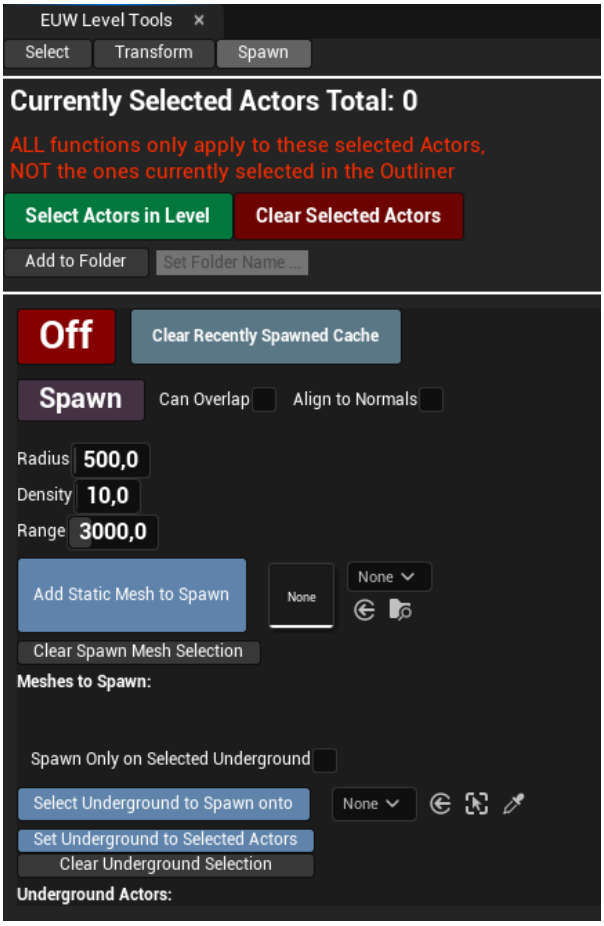


Abbildung 26: Utility B: Spawn-Tool

Für die Durchführung der Experimente wurden geeignete Assets in das Projekt importiert. Zur Bereitstellung verschiedener Skelette kam das kostenlose *Animal Variety Pack* aus dem FAB Marketplace zum Einsatz (Abbildung 27).

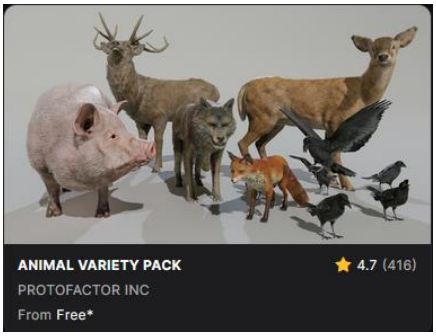


Abbildung 27: Asset-Pack für Tier-Skelette

Für die Level-Editing-Tests wurde das Paket Dreamscape: Stylized Environment – Tower / Stylized Nature aus dem FAB Marketplace verwendet, da es qualitativ hochwertige und praxisnahe Assets enthält, die sich gut für die Platzierungs- und Effizienztests eignen (Abbildung 28).

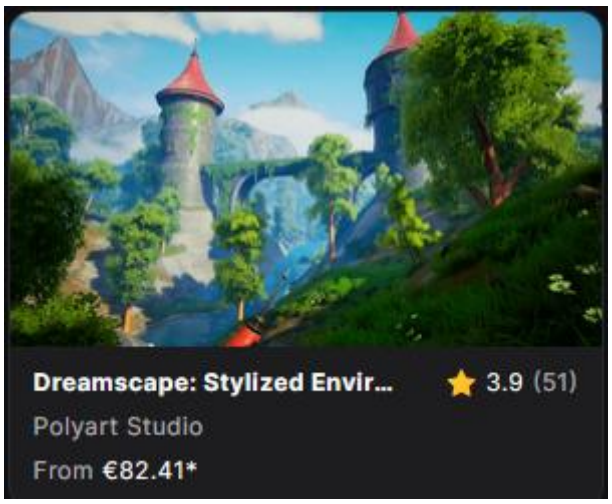


Abbildung 28: Asset-Pack Dreamscape
Quelle: https://www.fab.com/listings/c4e83f22-369a-4f5c-8f86-d53ccd716ae6

5.2.1. Utility A: Automatische Befüllung von Data Assets

Ziel: Die Knochenhierarchien ausgewählter Skeletal Meshes in jeweils ein Data Asset des Typs *PDA_Bones* zu transferieren, wobei das Feld der Schadensmultiplikation unverändert bleiben soll.

Manuell:

Für drei ausgewählte Skeletal Meshes (Pig, Crow, Fox) wird ein neues Data Asset vom Typ PDA_Bones erzeugt und nach der Konvention „*DA_<AssetName>_Bones*“ benannt. Anschließend werden alle Knochen des jeweiligen Skelettes manuell in das Data Asset übertragen.

Mit Utility:

Das zugehörige EUW (Abbildung 15) wird über die integrierte Schaltfläche im Editor gestartet. Anschließend soll mit diesem Tool die entsprechenden Data Assets befüllt werden. Die Funktionen und Bedienung des Tools werden nicht erläutert.

5.2.2. Utility B: Automatische Platzierung von Static Meshes

Utility B wurde für zwei Szenarien getestet:

1. Platzieren zahlreicher Kristall-Assets um eine Turmspitze in einer kugel-ähnlichen Form

   Ziel: Erzeugung einer kugelähnlichen Verteilung von schwebenden Kristallen um eine Turmspitze, ohne Kollisionen mit dem Turm oder untereinander. Insgesamt sollen mindestens 240 Kristalle platziert werden, mit annähernd gleicher Verteilung auf die verschiedenen Kristall-Assets.

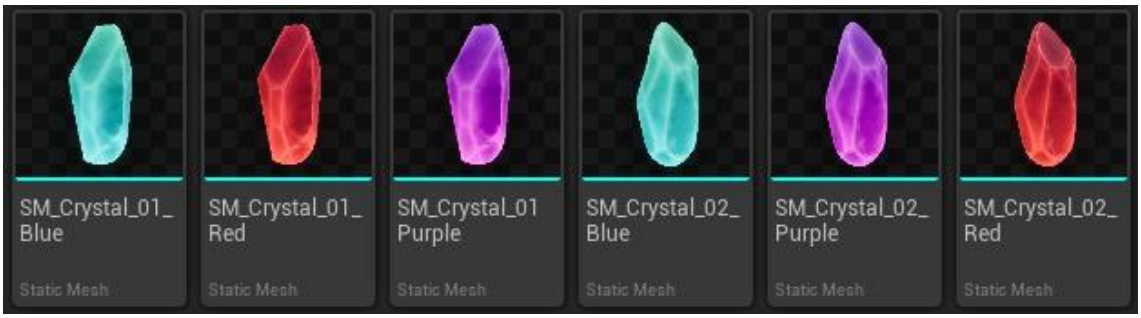


Abbildung 29: Kristall-Assets

   Weitere Vorgaben:

   - Alle platzierten Objekte eines Assets werden im Outliner in einem separaten Ordner organisiert
   - Blaue Kristalle: zufällige Rotation um alle Achsen & konstante Skalierung
   - Rote Kristalle: Rotation nur um die Z-Achse & konstante Skalierung
   - Lila Kristalle: Rotation nur um die Z-Achse & zufällige Skalierung im Bereich 0,7-1,3

Manuell:

Alle Kristalle sollen ohne Utility Tool von der Testperson nach eigener Methode platziert werden. Dabei müssen die oben genannten Anforderungen erfüllt sein. Fehler nach Abschluss werden vermerkt.

Mit Utility:

Das entsprechende EUW (Abbildung 24, Abbildung 25, Abbildung 26) wird über die integrierte Schaltfläche im Editor geöffnet. Die Testperson muss nun mit Hilfe dieses Tools die oben genannten Anforderungen erfüllen und die Kristalle um den Turm platzieren. Es darf händisch nachgebessert werden. Fehler nach Abschluss werden vermerkt.

2. Platzieren von Assets auf bestimmten Oberflächen (z.B. Tische) mit definierten inhaltlichen Regeln

   Ziel: Versehen einer Reihe von Tischen gemäß konkreten Regeln.

Abbildung 30: Beispiel einer Ausstattung der Tische

Vorgaben:

- Jeder Tisch muss zwischen 2 und 10 Kerzen besitzen.
- Jeder Tisch muss zwischen 1 und 4 zufällige Bücher mit variierenden Rotationen erhalten.
- Genau ein Tisch soll einen Globus und ein offenes Buch erhalten.
- Kein Tisch darf identisch zu einem anderen sein. Die Belegung muss eindeutig variieren.

Manuell:

Alle Objekte werden per Hand platziert und rotiert, sodass die Anforderungen erfüllt sind. Fehler nach Abschluss werden vermerkt.

Mit Utility:

Mit dem Tool sollen die Objekte gemäß den Vorgaben platziert und rotiert werden. Anpassungen per Hand sind erlaubt, um etwaige Kollisionen oder ästhetische Feinheiten zu korrigieren.

6. Pilotentest

Im folgenden Kapitel wird die Verständlichkeit und Usability der entwickelten Tools überprüft. Der Pilotentest diente dazu, die Versuchsdurchführung, Aufgabenstellung und Messinstrumente vor der eigentlichen Studie zu erproben und gezielt zu optimieren.

6.1. Aufgabenstellung

Es zeigte sich, dass die Testperson die neuen Tools zunächst im nicht im Editor fand, obwohl sie auf zusätzliche Schaltflächen hingewiesen wurde. Zur Verbesserung der Auffindbarkeit wurde den Teilnehmenden kurz vor Start der Aufgabe die Tool-Buttons gezeigt.

Die inhaltliche Präzisierung betraf darüber hinaus mehrere Stellen. So wurde die Anweisung zur Sortierung der Kristalle in jeweilige Outliner-Ordner konkretisiert. Bei der Tischbestückung wurde klargestellt, wie viele Tische jeweils ein offenes Buch und einen Globus erhalten sollen. Außerdem wurde die Forderung so erweitert, dass alle Tische sowohl Bücher als auch Kerzen erhalten sollen.

6.2. Feedback zur Durchführung

Der Pilottest lieferte konkrete Hinweise zur Bedienbarkeit:

- Utility A: Keine nennenswerten Probleme. Funktionalität und Bedienungsanforderungen wurden verstanden.
- Utility B: Probleme traten insbesondere bei der Actor-Selektion auf. Die im Editor markierten Objekte stimmten nicht immer mit der internen Auswahl des Tools überein, was zu Verwirrung führte. Daraufhin wurden zusätzliche Hinweise und Tooltips ins UI integriert.
- Der Begriff „Density“ im Spawn-Reiter von Utility B wurde als irreführend empfunden, da das Tool eine maximale Anzahl generierter Objekte steuert. Die Bezeichnung wurde in „max. Amount“ umbenannt.
- Das UI-Layout war bei dem Spawn-on-Selected-Underground-Feature mangelhaft. Die räumliche Trennung zwischen Checkbox und Untergrundauswahl war verwirrend. Das Layout wurde so überarbeitet, dass zusammengehörige Bedienelemente näher beieinander liegen.

- Im Select-Reiter von Utility B wurde das Eingabefeld für den Zielordner anfangs nicht als solches erkannt. Es wurde visuell stärker hervorgehoben. Der Knopf „Populate“ erzeugt Unsicherheit und wurde umbenannt in aussagekräftigere Bezeichnungen wie „Get all Classes“ bzw. „Get all Static Mesh Assets“. Insgesamt erforderte Utility B eine kurze Einarbeitungszeit. Nach dieser Phase konnten die Funktionen jedoch selbstständig und erfolgreich angewendet werden.

6.3. Fragebogen

Als Ergebnis des Pilotlaufs wurden zudem die Fragen des Fragebogens überarbeitet. Die Kategorie „Effektivität“ erhielt stärker aufgabenspezifisch formulierte Fragen, um die Eigenschaften der einzelnen Tools präziser abzufragen. Diese Anpassungen wurden vor Beginn der Hauptstudie umgesetzt.

7. Ergebnisse

Im folgenden Kapitel werden die Ergebnisse der empirischen Untersuchung dargestellt. Ziel war es, die Effizienz und Benutzerfreundlichkeit der entwickelten Editor-Utilities zu bewerten. Die Analyse umfasst sowohl quantitative Messungen der Bearbeitungsdauer als auch qualitative Einschätzungen der Teilnehmenden.

7.1. Stichprobe

Die Teilnehmerzahl betrug sechs Teilnehmer. Alle verfügten über ein bis vier Jahre Erfahrung in der Nutzung der Unreal Engine, während nur ein Drittel bereits Erfahrung mit Editor-Utilities hatte. Die unterschiedlichen Erfahrungsstände beeinflussen sowohl die allgemeine Vertrautheit mit der Engine als auch den Umgang mit neuen Werkzeugen.

7.2. Übersicht Aufgaben und Messgrößen

Die Teilnehmenden hatten drei Aufgaben zu bewältigen. Die erste Aufgabe zielte darauf ab, eine repetitive Tätigkeit im Bereich der Asset-Interaktion zu vereinfachen. Sie bestand darin, die Knochenhierarchie verschiedener Tier-Skelette in ein Data Asset zu übertragen.

Die zweite Aufgabe befasste sich mit dem Level-Design-Prozess, insbesondere mit der Platzierung einer großen Anzahl von Assets in der Luft. Es sollten zahlreiche Kristall-Assets um die Spitze eines Turms positioniert werden. Dabei durfte es zu keinen Überschneidungen mit anderen Assets kommen. Zudem verfügten die Kristalle über spezifische Vorgaben hinsichtlich Skalierung und Rotation. Die zu erzeugende „Kristallwolke“ hatte aus mindestens 240 Assets zu bestehen, verteilt auf sechs verschiedene Kristallvarianten.

Die dritte Aufgabe behandelte ebenfalls den Bereich des Level-Designs, jedoch mit Fokus auf eine deutlich präzisere Platzierung von Objekten. Hierfür mussten bestimmte Assets unter festgelegten Bedingungen auf vorgegebenen Tischen positioniert werden.

Alle drei Aufgaben wurden sowohl mithilfe der Tools als auch ohne diese durchgeführt. Die für die jeweilige Bearbeitung benötigte Zeit wurde erfasst und dient als primäre Messgröße. Nach Abschluss der Aufgaben dokumentierte ich die aufgetretenen Fehler.

7.3. Zeitmessungen

Im Folgenden werden die Zeitmessungen für jede Aufgabe vorgestellt und deren Einfluss auf die Nutzung von Editor-Utilities aufgezeigt.

7.3.1. Knochen-Export in Data Asset

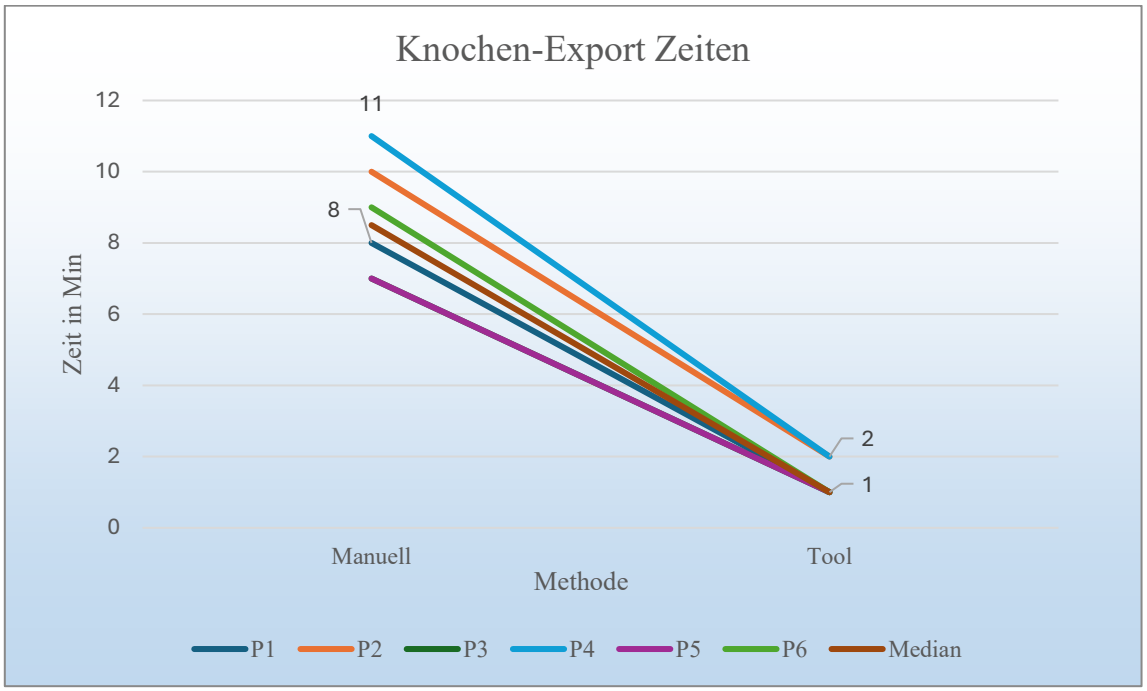


Abbildung 31: Aufgabe: Knochen-Export: Bearbeitungszeiten

Der Median der Bearbeitungsdauer bei der Knochen-Export Aufgabe beträgt bei manueller Durchführung 8,5 Minuten (IQR = 3, n = 6) (Abbildung 31). Mit dem entwickelten Tool betrug der Median der Zeit nur 1,0 Minuten (IQR = 1, n = 6). Die Zeitersparnis im Median lag damit bei 7,5 Minuten (IQR = 2, n = 6), was einer Reduktion von 86 % entspricht. Die Streuung der Differenzen war gering, sodass der positive Effekt bei allen Testpersonen beobachtet wurde (Abbildung 31).

Diese deskriptiven Kennzahlen bilden die Grundlage für die folgende ökonomische Bewertung der Berechnung des Break-Even-Points.

Die Ergebnisse zeigen deutlich, dass die Bearbeitungszeit durch den Einsatz des Tools erheblich reduziert wird. Alle Teilnehmenden waren in der Lage, das Tool ohne vorherige Einweisung effizient zu nutzen. Insgesamt konnte eine durchschnittliche Zeitersparnis von 86 % festgestellt

werden. Diese Effizienzsteigerung skaliert mit der Größe des Projekts bzw. mit der Häufigkeit der Nutzung. Je mehr Skeletthierarchien exportiert werden, desto größer fällt der Nutzen des Tools aus. Während der Evaluation zeigten sich keinerlei Nachteile gegenüber der manuellen Übertragung. Stattdessen konnte die Fehleranfälligkeit vollständig eliminiert werden. Wie in Abbildung 32 erkenntlich, konnte die Aufgabe von allen Personen fehlerfrei durchgeführt werden.

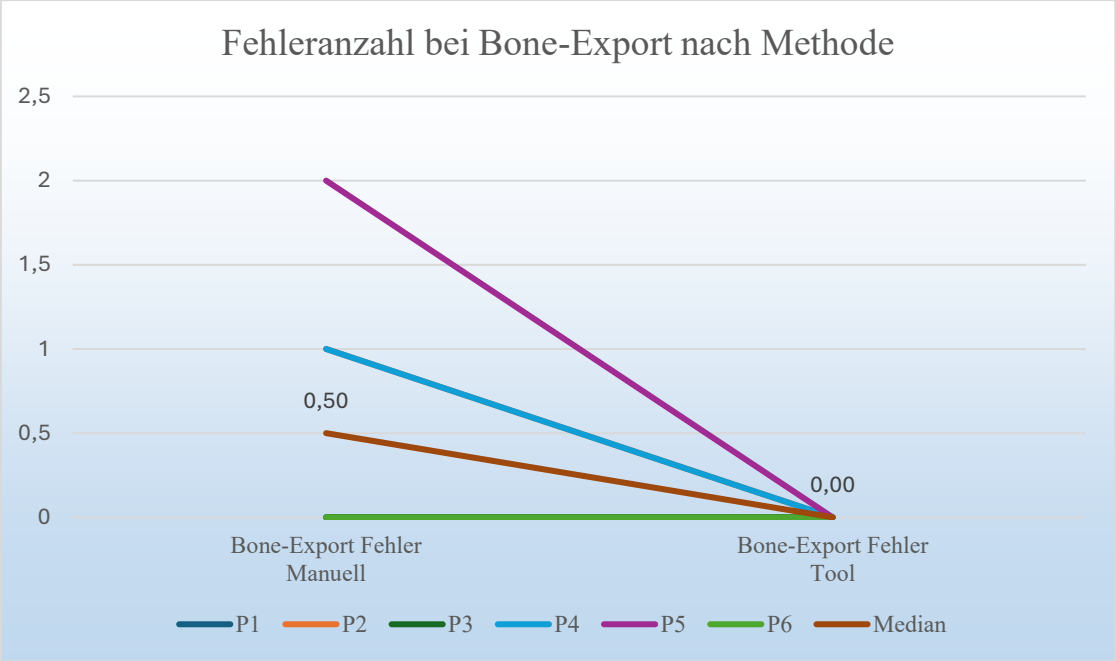


Abbildung 32: Aufgabe: Knochen-Export: Fehleranzahl

7.3.2. Level-Design: Generierung von Assets in der Luft

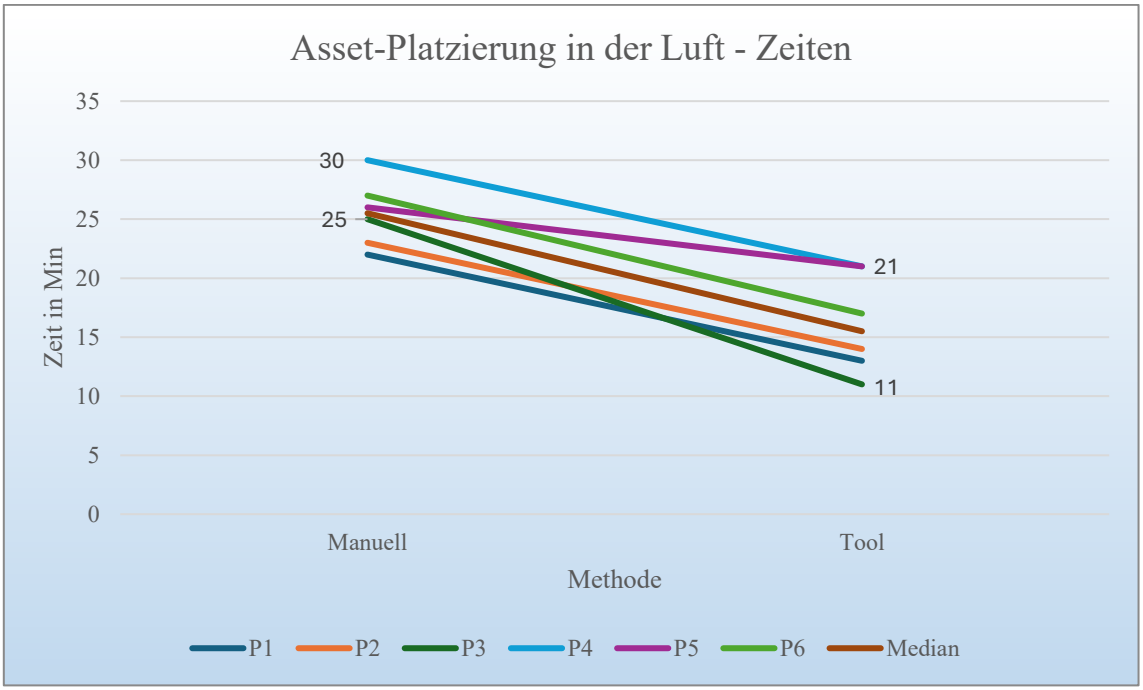


Abbildung 33: Aufgabe: Generierung von Assets in der Luft: Bearbeitungszeit

Der Median der Bearbeitungsdauer bei der Generierung von Assets um eine Turmspitze beträgt bei manueller Durchführung 25,5 Minuten (IQR = 4, n = 6) (Abbildung 33). Mit dem entwickelten Tool betrug der Median der Zeit nur 15,5 Minuten (IQR = 8, n = 6) (Abbildung 33). Die Zeitersparnis im Median lag damit bei 9 Minuten (IQR = 1, n = 6) (Abbildung 43), was einer durchschnittlichen Reduktion von 36 % entspricht (Abbildung 37). Die Streuung der Differenzen war bei der Nutzung des Tools hoch, da sich Einarbeitungszeit und Verständnis des Tools zwischen den Testpersonen deutlich unterschieden. Grund dafür sind Erfahrung und Vertrautheit mit der Unreal Engine. Der positive Effekt des Editor-Utilities lässt sich zwar nachweisen, tritt jedoch mit uneinheitlicher Effizienz auf (Abbildung 33).

Die Ergebnisse zeigen, dass sich die Bearbeitungszeit durch den Einsatz des Editor Utility Tools insgesamt reduziert lässt. Im Median wurde eine Zeitersparnis von 36 % erzielt, wobei die Einsparung zwischen den Teilnehmenden deutlich variiert. Diese Streuung ist vor allem auf unterschiedliche Erfahrungen im Umgang mit der Unreal Engine allgemein zurückzuführen.

Die Effektivität des Tools steigt mit der Nutzungshäufigkeit, sodass bei wiederholter Anwendung eine kumulative Zeitersparnis zu erwarten ist.

Bei der Anwendung zeigte sich, dass alle Teilnehmenden anfangs eine Eingewöhnungsphase benötigten, da keine Einführung bereitgestellt wurde. Die Dauer dieser Phase variiert zwischen den Personen, letztlich konnten jedoch alle Teilnehmenden das Tool vollständig verstehen und effizient einsetzen. Vor dem Hintergrund dieses Lerneffekts ist bei der Wiederholung des Experiments mit denselben Personen mit einer höheren Zeitersparnis zu rechnen, da die Orientierungsphase dann entfiele.

Die Hypothese 1, wonach Editor-Utilities die Bearbeitungszeit typischer Aufgaben gegenüber manuellen Verfahren reduzieren, wird durch die vorliegenden Daten gestützt. Im Median lag die Zeitersparnis bei 36 %, was den praktischen Nutzen des Tools in diesem Anwendungsgebiet belegt. Die Fehleranalyse ergibt hingegen ein differenzierteres Bild. Sowohl bei manueller Bearbeitung als auch bei der Verwendung des Tools traten Fehler auf. Der Median der Fehleranzahl bei der manuellen Methode lag bei Median = 0 (IQR = 2, n = 6) und bei der Tool gestützten Methode bei Median = 1 (IQR = 0, n = 6), womit die Fehleranzahl mit Tool höher war. Zudem ist die Streuung bei der manuellen Methode moderat hoch, bei der Methode mithilfe des Tools jedoch sehr konzentriert um den Median (0), da fast alle Teilnehmenden gleich viele Fehler machten (Abbildung 34).

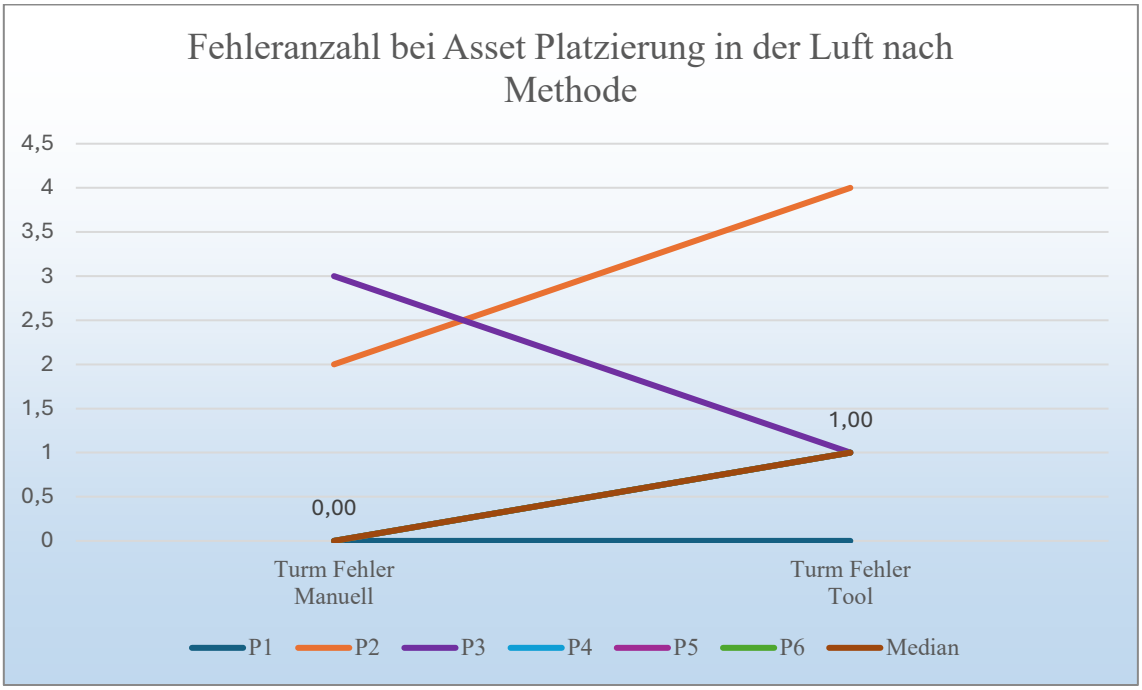


Abbildung 34: Aufgabe: Generierung von Assets in der Luft: Fehleranzahl

Vor diesem Hintergrund lässt sich Hypothese 2, dass Editor-Utilities die Fehlerquote verringern und zu konsistenteren Ergebnissen führen, nicht allgemein bestätigen. Es zeigt sich jedoch, dass einzelne Teilnehmer von der Tool Nutzung profitierten und dadurch weniger Fehler machten, sodass die Wirkung auf die Fehlerquote situationsabhängig ist.

7.3.3. Level-Design: Massenplatzierung von Assets auf bestimmten Oberflächen

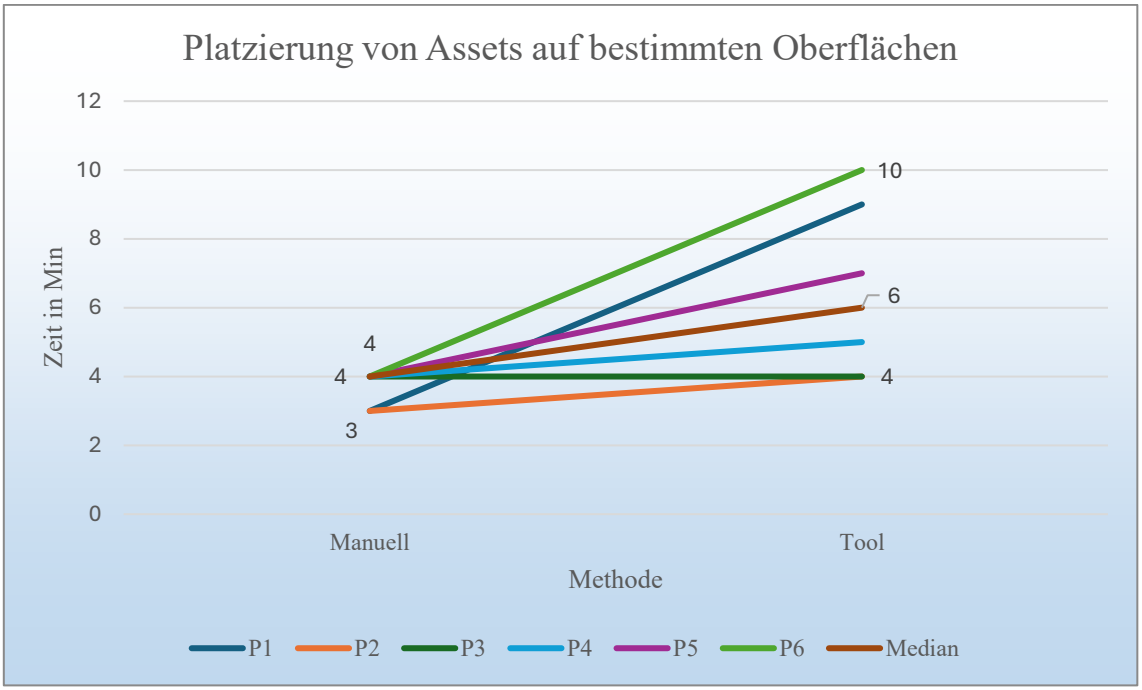


Abbildung 35: Aufgabe: Massenplatzierung von Assets auf bestimmten Oberflächen: Bearbeitungszeiten

Der Median der Bearbeitungsdauer bei der Platzierung von Assets auf bestimmten Untergründen beträgt bei manueller Durchführung 5 Minuten (IQR = 1, n = 6). Mit dem entwickelten Tool betrug der Median der Bearbeitungsdauer 6 Minuten (IQR = 5, n = 6). Die Zeitersparnis im Median lag damit bei -2 Minuten (IQR = 5, n = 6), was einer durchschnittlichen Erhöhung der Bearbeitungszeit von 54 % entspricht. Die Streuung der Zeitdifferenzen war bei der Nutzung des Tools fast so hoch wie der Mittelwert (6,5), was auf die unterschiedliche Vertrautheit mit dem Tool zurückzuführen ist. Ein positiver Effekt des Tools lässt sich nicht nachweisen. Im Umkehrschluss ist das Tool für diesen konkreten Anwendungsfall nicht geeignet, da die Bearbeitungszeit mit Tool höher ist als ohne Tool (Abbildung 35).

Im Median benötigten die Teilnehmenden 54 % mehr Zeit, um die Aufgabe zu erledigen, obwohl sie zu diesem Zeitpunkt bereits mit dem Tool vertraut waren. Tritt eine solche Aufgabe im Projekt häufiger auf, ist daher der Verzicht auf das Tool und die Wahl einer alternativen Methode zu empfehlen.

Der wesentliche Unterschied zur vorherigen Aufgabe liegt in der erforderlichen Feinjustierung. In diesem Fallbeispiel mussten wenige, dafür jedoch besonders präzise, Assets platziert werden. Die Ergebnisse verdeutlichen, dass eine exakte Positionierung mit dem Tool nicht effizient umsetzbar ist und die manuelle Platzierung deutlich schneller zum Ziel führt. Zudem wurden mit dem Tool wiederholt Assets an Positionen generiert, die zwar formal Teil des Untergrundes waren, sich jedoch nicht für eine korrekte Platzierung eigneten. Solche Fehlplatzierungen traten bei der manuellen Methode nicht auf (Abbildung 36).

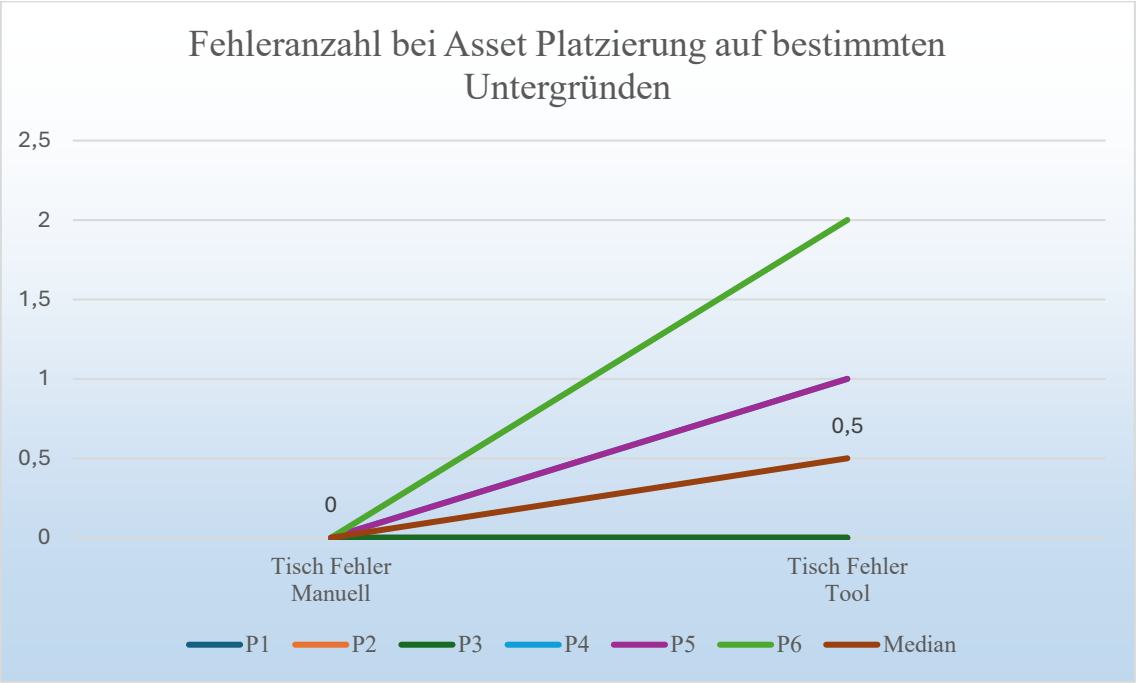


Abbildung 36: Aufgabe: Massenplatzierung von Assets auf bestimmten Oberflächen: Fehleranzahl

Obwohl die Teilnehmenden mit dem Tool besser vertraut waren als zu Beginn, hatten sie Schwierigkeiten, es effektiv in einem räumlich stark begrenzten Bereich einzusetzen. Die Aufgabe ließ sich manuell schnell und gezielt lösen, während das Tool zahlreiche alternative, aber weniger effiziente Vorgehensweisen eröffnete. Lediglich eine Person konnte die Aufgabe mithilfe des Tools in etwa der gleichen Zeit bewältigen wie bei der manuellen Durchführung (Abbildung 35).

Hypothese 1, dass der Einsatz von Editor-Utilities die Bearbeitungszeit typischer Aufgaben im Vergleich zur manuellen Durchführung reduziert, trifft in diesem konkreten Anwendungsfall

nicht zu. Stattdessen führte die Nutzung des Tools zu einer Verlängerung der Bearbeitungszeit. Für Aufgaben dieser Art empfiehlt sich daher entweder eine gezielte Anpassung bzw. Weiterentwicklung des Tools oder die Ausführung der Arbeitsschritte ohne Tool.

Die Analyse zeigt zudem, dass die Aufgabe ohne Tool fehlerfrei durchgeführt werden konnte (Median = 0), während beim Einsatz des Tools Fehler auftraten (Median = 0,5). Daraus lässt sich schließen, dass die Fehleranfälligkeit des Tools für diese spezifische Aufgabe erhöht ist (Abbildung 35).

Somit ist Hypothese 2 in diesem Anwendungsfall widerlegt.

7.4. Zusammenfassender Vergleich aller Aufgaben

In diesem Abschnitt werden die Ergebnisse der drei Aufgaben hinsichtlich Bearbeitungszeit und Fehleranfälligkeit direkt gegenübergestellt, um den tatsächlichen Einfluss der entwickelten Editor-Utilities auf die Arbeitsabläufe zu bewerten.

7.4.1. Vergleich Bearbeitungszeiten pro Aufgabe

Die Analyse der Bearbeitungszeit zeigt ein insgesamt gemischtes Bild. Während zwei der drei untersuchten Aufgaben von einer deutlichen Reduktion der Bearbeitungsdauer durch den Einsatz der entwickelten Tools profitierten, führte eines der Utilities nicht zu einer schnelleren Bearbeitung und war im Schnitt etwas langsamer als die manuelle Durchführung.

Für den Knochen-Export (7.3.1) ergab sich eine klare Verbesserung der Effizienz des Arbeitsvorgangs. Die Bearbeitungszeit im Median lag bei 8,5 Minuten, während die Durchführung mit dem entsprechenden Tool im Median bei 1 Minute lag.

Auch bei der Aufgabe, bei der die Nutzer Assets um eine Turmspitze platzieren mussten (7.3.2), zeigte sich ein vergleichbarer Effekt. Die manuelle Bearbeitung dauerte im Median 25,5 Minuten, während der Einsatz des Tools die Bearbeitungszeit im Median auf 15,5 Minuten reduzierte.

Ein abweichendes Ergebnis zeigte die dritte Aufgabe (7.3.3), „Platzierung von Assets auf bestimmten Oberflächen“. Hier lag die manuelle Bearbeitungszeit im Median bei 4 Minuten, während die Tool-gestützte Variante im Median 6 Minuten benötigte. Die Beobachtung deutet darauf hin, dass das Editor Utility Widget in dieser konkreten Aufgabe keine Effektivitätssteigerung erzielte und möglicherweise durch zusätzlichen Interaktionsaufwand oder eine unzureichende Automatisierungslogik ausgebremst wurde.

Zusammenfassend zeigen die Ergebnisse, dass der Einsatz der entwickelten Tools in zwei von drei Fällen zu einer deutlichen Workflow-Beschleunigung führt. Gleichzeitig verweist die ineffizientere Bearbeitungszeit der dritten Aufgabe darauf, dass der Nutzen von Editor-Utilities stark aufgabenspezifisch ausfallen kann und Optimierungspotenzial für bestimmte Anwendungsfälle besteht.

7.4.2. Zeitersparnis insgesamt

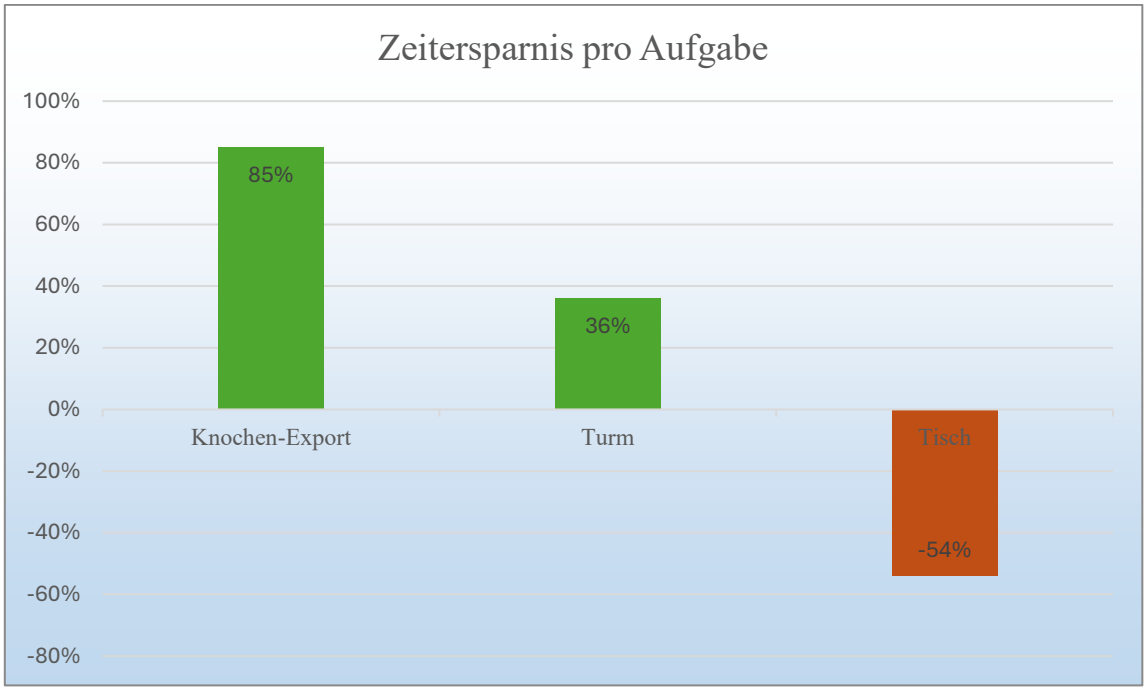


Abbildung 37: Zeitersparnis pro Aufgabe

Die relative Betrachtung der Zeitersparnis zeigt, dass der Einsatz der Tools trotz unterschiedlicher Ergebnisse pro Aufgabe insgesamt zu einer deutlichen Effizienzsteigerung führen kann. Für den Knochen Export ergab sich eine Reduktion von 86 % (IQR = 6, n = 6) im Median, während das Utility zum Platzieren vieler Assets in der Luft eine Zeitersparnis von 36% (IQR = 26, n = 6) im Median erreichte. Beide Aufgaben verdeutlichen, dass automatisierte Arbeitsschritte in diesen Bereichen einen spürbaren praktischen Nutzen bieten (Abbildung 37).

Demgegenüber führte die Aufgabe zur Platzierung von Assets auf bestimmten Oberflächen zu einer negativen Zeitbilanz. Hier verlängerte der Einsatz des Tools den Arbeitsprozess um 54 % (IQR = 125) im Median, was zeigt, dass der Grad der Automatisierung in diesem Fall nicht ausreichte, um den zusätzlichen Bedien- und Konfigurationsaufwand auszugleichen (Abbildung 37).

Trotz dieses Ausreißers ergibt sich über alle Aufgaben hinweg eine durchschnittliche relative Zeitersparnis von insgesamt 68 %. Dieser Wert verdeutlicht, dass der überwiegende Teil der

untersuchten Tätigkeiten klar von den entwickelten Utilities profitiert, wenngleich der Nutzen nicht für jeden Arbeitsschritt gleichermaßen gegeben ist.

7.4.3. Fehleranfälligkeit

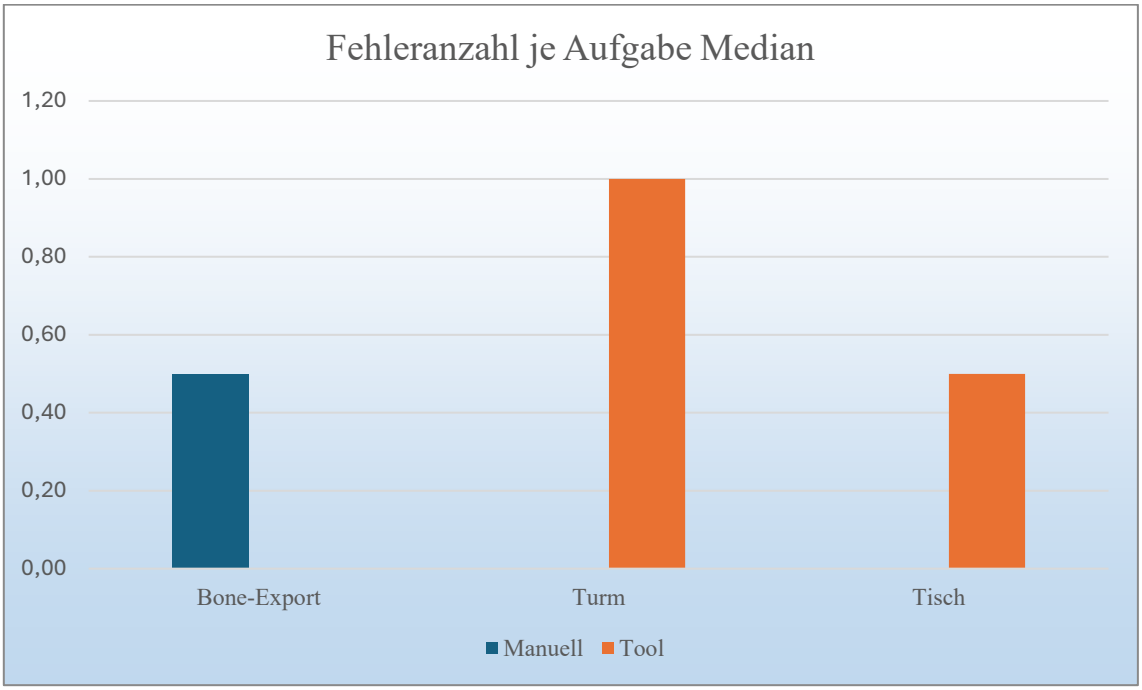


Abbildung 38: Fehleranzahl nach Aufgabe

Die quantitative Auswertung der protokollierten Fehlerfälle liefert ein differenziertes Bild. Während das Tool für eine Aufgabe zu einer Reduktion der Fehlerrate führte, erhöhte es in zwei weiteren Aufgaben die Zahl der gemachten Fehler teilweise deutlich.

Für den Knochen-Export wurde im manuellen Ablauf eine Fehleranzahl im Median von 0,5 (IQR = 1, n = 6) dokumentiert. Durch den Einsatz des Tools sank die Fehleranzahl im Median auf 0 (IQR = 0, n = 6), was auf eine Verbesserung der Prozesssicherheit in diesem Bereich hinweist (Abbildung 38).

Im Gegensatz dazu erhöhte sich bei der zweiten Aufgabe die Fehleranzahl leicht. Bei manuellem Vorgehen lag die Fehleranzahl im Median bei 0 (IQR = 2, n = 6), die Streuung war jedoch sehr hoch. Daran lässt sich erkennen, dass dennoch Fehler auftreten. Bei Nutzung des Tools betrug die Fehleranzahl im Median 1 (IQR = 0, n = 6). Die Differenz ist geringfügig,

weist jedoch darauf hin, dass die Automatisierung in diesem Fall neue Fehlerquellen eingeführt oder bestehende verstärkt haben könnte (Abbildung 38).

Besonders auffällig ist das Ergebnis der dritten Aufgabe. Während im manuellen Ablauf keine Fehler protokolliert wurden (Median = 0, IQR = 0, n = 6), stieg die Fehleranzahl bei Nutzung des Tools im Median auf 0,5 (IQR = 1, n = 6) an. Dieses deutliche Anwachsen der Fehlerrate zeigt, dass das Tool in dieser Aufgabe nicht die erwünschte Robustheit erzielt hat (Abbildung 38).

Zusammenfassend zeigen die Fehlerdaten, dass Automatisierung nicht zwangsläufig Fehler reduziert. Sie kann in manchen Anwendungsfällen zu weniger Fehlern führen oder diese sogar gänzlich vermeiden (Abbildung 38). In anderen Fällen wiederum kann es zu einer Zunahme von Fehlern führen. Die zugrundeliegenden Ursachen für die erhöhten Fehlerraten werden in Abschnitt 7.4.4 näher untersucht und diskutiert.

7.4.4. Zusammenfassung

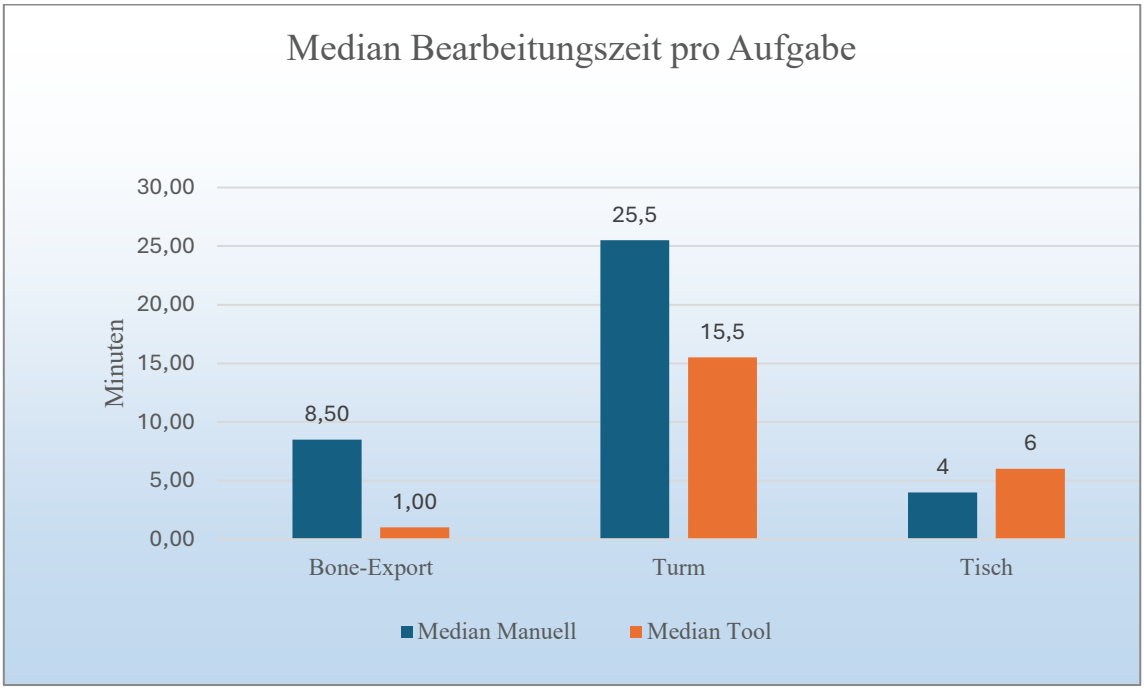


Abbildung 39: Bearbeitungszeit pro Aufgabe im Median

Zusammenfassend zeigen die vorliegenden Messdaten, dass die Automatisierung durch die entwickelten Editor-Utilities unter den hier getesteten Bedingungen überwiegend positive Effekte erzielt. In zwei von drei Aufgaben verringerte die Nutzung der Tools die Bearbeitungszeit deutlich gegenüber der manuellen Durchführung (Abbildung 39). Aufgrund der Stichprobengröße von n = 6 sind diese Ergebnisse als indikativ zu verstehen und erlauben vorrangig Aussagen über Tendenzen. Da diese Tools in der Praxis meist nur von einem kleinen Team benutzt werden, ist das Ergebnis dennoch als aussagekräftig einzuordnen.

Die Analyse der Fehlerfälle macht deutlich, dass die Effektivität der Tools anwendungsspezifisch ist. Bei einfachen, klar definierten Aufgabenstellungen, wie dem Export von Knochen-Hierarchien in ein Data Asset (siehe 5.2.1), sank sowohl der Zeitaufwand als auch die Fehlerrate. Bei räumlich oder konfigurationsabhängigen Aufgaben (z. B. Kollisionsfälle bei der Turmspitze; Platzierung auf speziellen Untergründen) hingegen stieg die Fehlerrate unter Tool-Nutzung an. Solche Effekte sind auf eine fehlende Feinjustierung der Funktionsweise der Tools und auf Bedienfehler zurückzuführen.

Praktisch bedeuten die Befunde, dass Editor-Utilities besonders dann empfehlenswert sind, wenn die Aufgaben wiederkehrend und gut formalisiert sind. Komplexere oder stark kontextabhängige Aufgaben erfordern hingegen zusätzliche Validierungsmechanismen oder eine gezielte Schulung der Nutzenden. Eine ökonomische Bewertung unter Berücksichtigung der Entwicklungszeit folgt in Abschnitt 7.6.

7.5. Usability Bewertung

Zur Ergänzung der quantitativen Messergebnisse wurden die Teilnehmenden gebeten, die Bedienbarkeit der entwickelten Tools anhand von drei Likert-Fragen zu bewerten:

- Intuitivität der Nutzung
- Notwendigkeit von Hilfestellung
- Wahrscheinlichkeit einer zukünftigen Nutzung

Alle Fragen wurden auf einer fünfstufigen Skala (1 = stimme nicht zu, 5 = stimme voll zu) beantwortet.

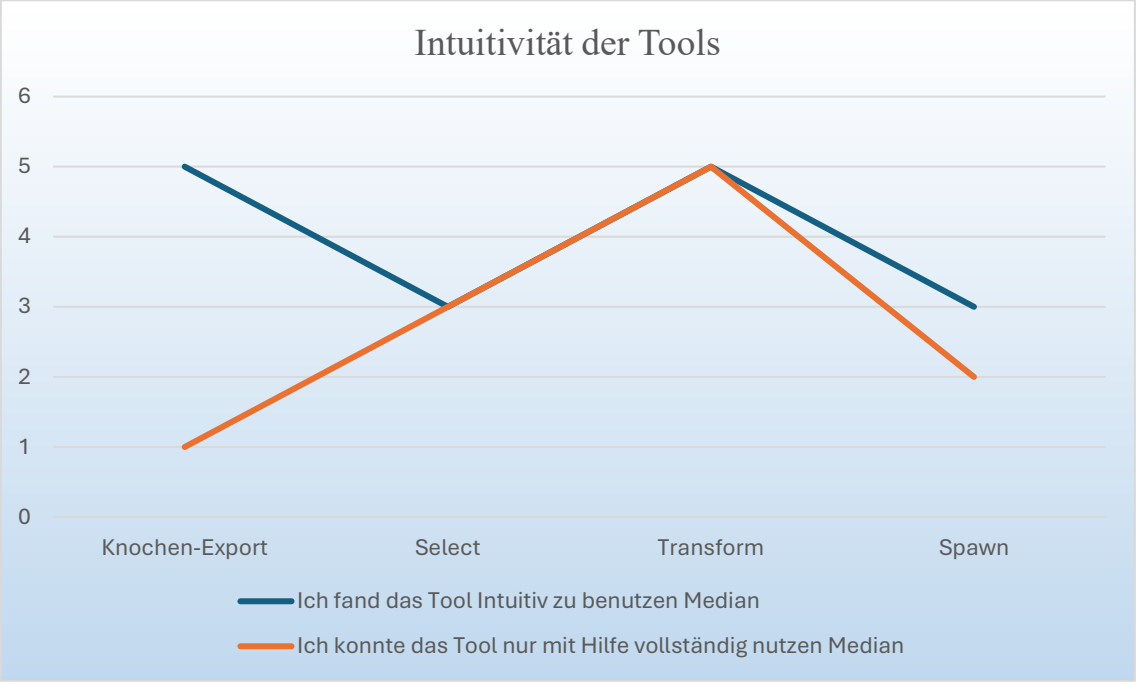


Abbildung 40: Usability Fragebogen: Intuitives Benutzen der Tools

Die Ergebnisse zeigen, dass alle Tools im Durchschnitt als intuitiv wahrgenommen wurden. Die Bewertungen reichen von einem Maximum von 5 beim Transform- und Knochen-Export-Tool (Abbildung 25, Abbildung 26, Abbildung 40) bis zu einem Minimum von 3 beim Spawn- und Select-Tool (Abbildung 26, Abbildung 24, Abbildung 40). Erkennbar ist, dass komplexere

Tools wie Select und Spawn, die vielfältig einsetzbar sind, als weniger intuitiv eingeschätzt wurden als die einfachen Tools wie das Knochen-Export und Transform-Tool (Abbildung 40).

Das Knochen-Export-Tool und das Transform-Tool erzielten beide sehr hohe Werte in Bezug auf die Intuitivität (Abbildung 40). Dies lässt sich darauf zurückführen, dass diese Tools einen klar begrenzten Funktionsumfang besitzen und ihr Nutzen leicht erfassbar ist. Sie reduzieren den Arbeitsaufwand wiederkehrender, einfacher Aufgaben durch Automatisierung oder Massenoperationen, die ohne Tool nicht möglich oder erheblich zeitaufwendiger wären. Der geringe Interquartilsabstand unterstreicht die einheitliche Wahrnehmung dieser beiden Tools.

Das Select-Tool und das Spawn-Tool hingegen sind komplexer in ihrer Anwendungsweise und flexibel einsetzbar. Dadurch vergrößert sich das User Interface (UI), was die Übersichtlichkeit verringert, und die intuitive Bedienbarkeit erschwert. Nicht allen Teilnehmenden war die vollständige Funktionsweise direkt ersichtlich. Erst durch eine nachträgliche Einweisung konnten alle Funktionen korrekt angewandt werden. Die höheren IQR Werte zeigen zudem, dass die Tools je nach Erfahrung der Teilnehmenden mit der Unreal Engine und Editor-Utilities unterschiedlich wahrgenommen wurden.

Da insbesondere die Erfahrungswerte unter Anfängern in der UE5 stark variieren, unterscheiden sich auch die Wahrnehmung und Handhabung der Tools. Das Ziel, alle Tools möglichst intuitiv zu gestalten, sodass keine Einweisung nötig ist, konnte daher nur begrenzt erreicht werden.

Trotz der Komplexität mancher Tools und trotz zusätzlicher Zeitkosten bei einer der Aufgaben äußerten alle Teilnehmenden, dass sie die Tools in zukünftigen Arbeitsprozessen wahrscheinlich nutzen würden (Abbildung 41).

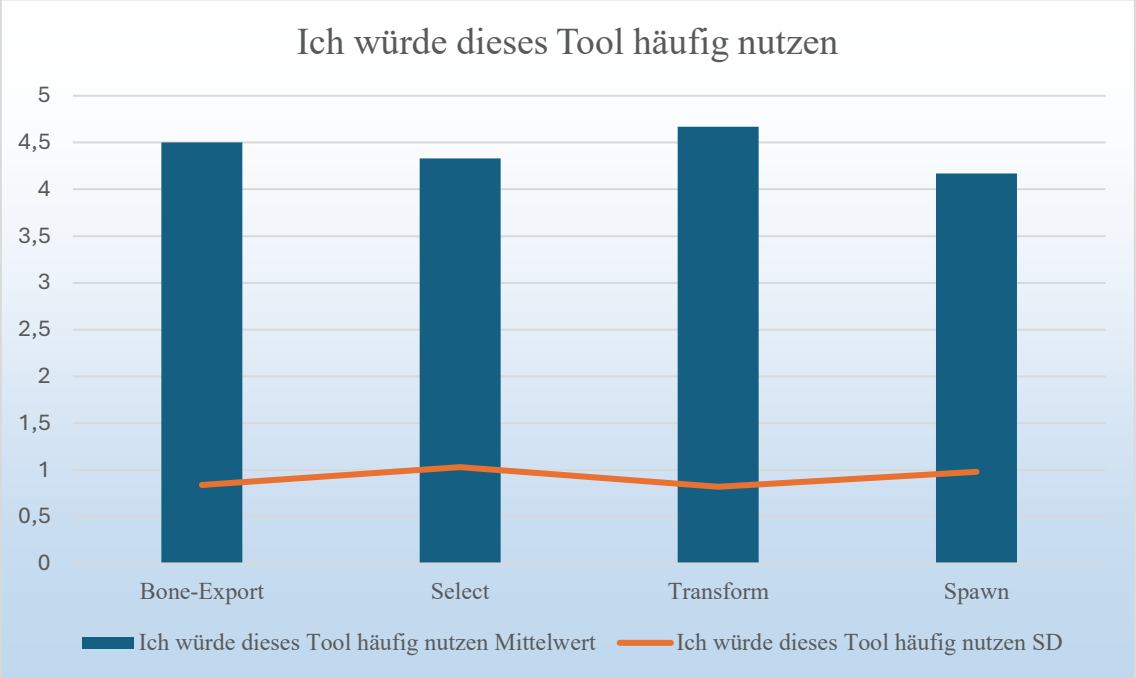


Abbildung 41: Häufigkeit der Nutzung des Utilities

Dies zeigt, dass der allgemeine Nutzen der Editor-Utilities von den Teilnehmenden klar erkannt wurde. Insbesondere die Tools für Level-Design-Aufgaben waren nicht optimal auf die gestellten Aufgabenstellungen abgestimmt, was die geringere Effizienz in diesen Testfällen erklärt. In anderen Anwendungsszenarien würden die Teilnehmenden das Tool dennoch verwenden.

7.5.1. Qualitativ

Vier der sechs Teilnehmenden hoben das Knochen-Export-Tool besonders positiv hervor:

„... super, einfach und effektiv“

Ebenfalls positiv wurde erwähnt, dass die Editor Utility Widgets bequem per Knopfdruck im Editor geöffnet werden konnten.

Drei Teilnehmende merkten an, dass die Einarbeitungszeit insbesondere ohne Einweisung etwas länger dauerte, die Tools nach dieser Phase jedoch gut nutzbar waren:

„Spawn Tool komplex, also ein bisschen Einarbeitungszeit von Nöten, aber sonst keine Probleme“

Alle Teilnehmenden gaben den Verbesserungsvorschlag, das UI-Design zu überarbeiten, um eine intuitivere und klarere Verständlichkeit zu gewährleisten:

„Mesh Placement Tool UX-freundlicher aufbauen und gestalten, um es besser zu verstehen“

7.6. Entwicklungsaufwand und praktische Bewertung

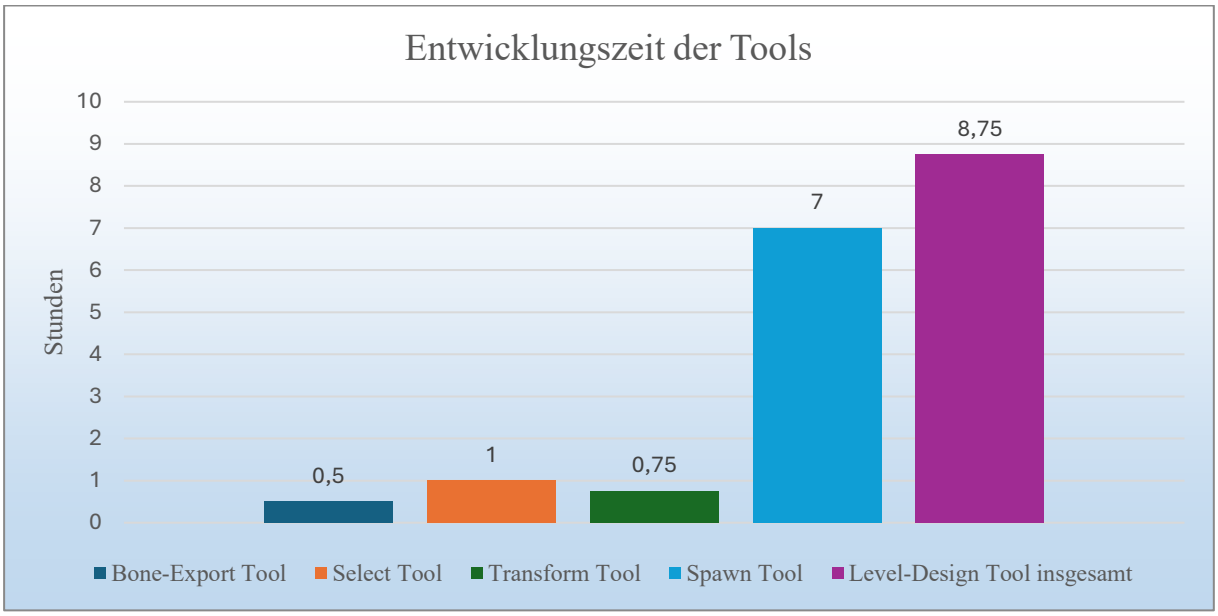


Abbildung 42: Entwicklungszeit der Editor-Utilities

Die Entwicklungszeit der jeweiligen Tools variierte deutlich in Abhängigkeit von der Komplexität. Das Knochen-Export-Tool konnte bereits in 30 Minuten erstellt werden (Abbildung 42). Die Entwicklung des Select-Tools benötigte eine Stunde, während das Transform-Tool innerhalb von 45 Minuten fertiggestellt werden konnte (Abbildung 42). Das komplexeste der entwickelten Editor-Utilities, das Spawn-Tool, erforderte insgesamt sieben Stunden Entwicklungszeit, obwohl dieses noch weiter verbessert und angepasst werden könnte (Abbildung 42). Es handelt sich also nur um die minimale Entwicklungszeit, die benötigt wurde.

Damit beträgt die gesamte Entwicklungsdauer des Level-Design-Toolsets (Utility B, siehe 5.2) 8 Stunden und 45 Minuten (Abbildung 42). Die im Vergleich deutlich längere Entwicklungszeit des Spawn-Tools ist auf dessen umfangreiche Funktionalität und hohe technische Komplexität zurückzuführen.

Die erfassten Zeiten umfassen den vollständigen Entwicklungsprozess, bestehend aus Implementierung, Testphasen sowie der Gestaltung der Benutzeroberflächen. Die Zeitmessung

erfolgte präzise mittels Stoppuhr und wurde für jedes Tool separat durchgeführt, da alle Werkzeuge nacheinander entwickelt wurden.

Die Ermittlung der Zeitersparnis pro Nutzung der Tools gestaltet sich als unterschiedlich schwierig. Beim Knochen-Export-Tool ist eine Nutzung eindeutig definierbar, da das Tool ausschließlich zur Befüllung eines Data Assets dient. Die Fertigstellung dieses Vorgangs markiert somit eine abgeschlossene Nutzung. Im Median wird pro Anwendung eine Zeitersparnis von 7,5 Minuten erzielt, was gerundet etwa acht Minuten entspricht (Abbildung 43).

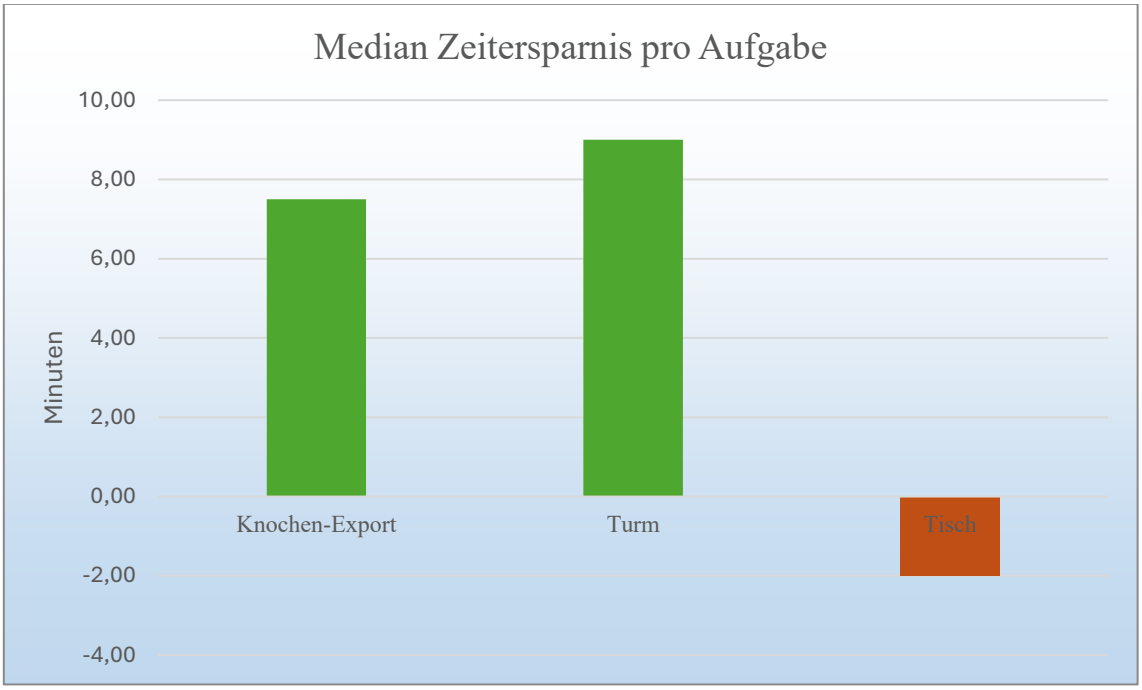


Abbildung 43: Zeitersparnis pro Aufgabe in Minuten

Die Zeitersparnis des Level-Design-Toolsets pro Nutzung lässt sich nur bedingt bestimmen. Die drei Werkzeuge (Utility B, siehe 5.2) sind flexibel einsetzbar und unterstützen den Entwicklungsprozess an vielen unterschiedlichen Stellen. Aufgrund dieser hohen Variabilität kann eine einzelne Nutzung nicht klar definiert werden. Stattdessen kann im Rahmen des Experiments jeweils die vollständige Bearbeitung einer Aufgabe als abgeschlossene Nutzung betrachtet werden.

Für die Testaufgaben ergibt sich im Median eine Zeitersparnis von neun Minuten bei der Platzierung von Assets in der Luft um die Turmspitze (Abbildung 43).

Bei der Aufgabe zur präzisen Platzierung von Assets auf bestimmten Untergründen zeigte sich hingegen eine Zeitverzögerung von zwei Minuten (Abbildung 43).

Diese Ergebnisse verdeutlichen, dass die Tools nicht in allen Anwendungsfällen zwangsläufig eine Zeitersparnis bieten. Insbesondere bei Aufgaben, die eine hohe Präzision erfordern, kann die manuelle Bearbeitung schneller sein als die Nutzung des Tools. Entscheidend ist die Verknüpfung zwischen der Entwicklungszeit und der tatsächlichen Zeitersparnis durch Nutzung der Tools.

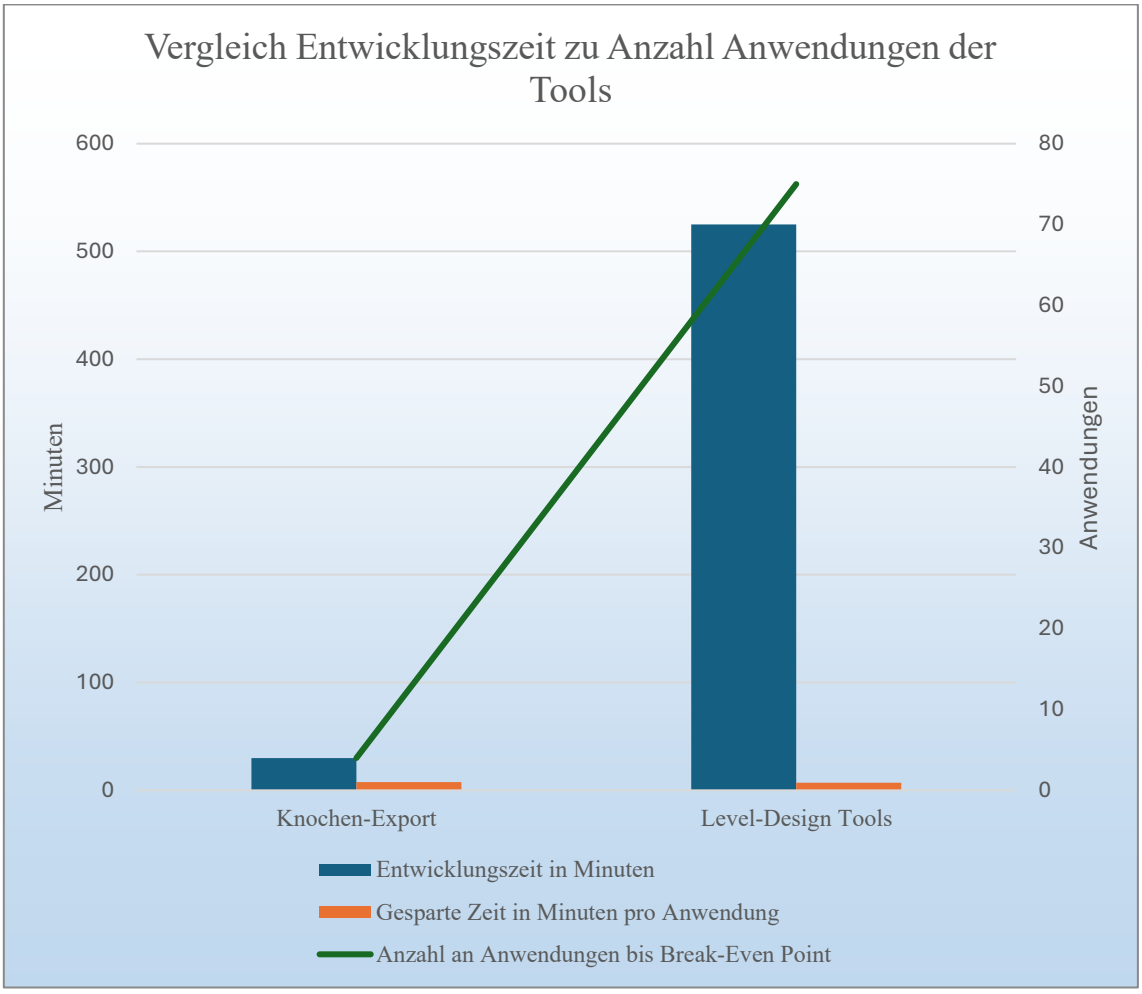


Abbildung 44: Vergleich Entwicklungszeit zu Anzahl Anwendungen

Der Break-Even-Point (BEP) beschreibt den Zeitpunkt, ab dem sich die Entwicklung eines Tools durch die eingesparte Bearbeitungszeit amortisiert. Ein Tool lohnt sich also dann, wenn die durch seine Nutzung eingesparte Zeit die zur Entwicklung benötigte Zeit übersteigt. Für die Berechnung wird die Entwicklungszeit des Tools und die durchschnittliche Zeitersparnis pro Nutzung benötigt.

BEP = Entwicklungszeit (Minuten) / Zeitersparnis pro Nutzung (Minuten)

Für das Knochen-Export-Tool ergibt sich:

30 / 7,5 = 4

Nach vier Nutzungen ist die eingesparte Zeit mindestens so hoch wie die Entwicklungszeit (Abbildung 44). In der praktischen Anwendung wird dieser Wert problemlos erreicht, da mehr als vier Skelette verarbeitet werden. Zusätzlich spricht die vollständige Eliminierung von Fehlern (Abbildung 32) sowie die hohe Zeitersparnis (Abbildung 37) für einen klaren praktischen Nutzen. Auch die Ergebnisse des SUS-Fragebogens bestätigen die einfache Bedienbarkeit und die intuitive Nutzung des Tools (Abbildung 40).

Das Level-Design-Toolset wurde einmal entwickelt, aber in zwei Aufgaben eingesetzt. Deshalb können die Zeitersparnisse (bzw. Verluste) beider Aufgaben addiert werden.

Aufgabe 2: +9 Minuten (Abbildung 43)

Aufgabe 3: -2 Minuten (Abbildung 43)

525 / (9 + (-2)) = 75

Daraus ableitend ergibt sich, dass erst nach 75 Nutzungen die eingesparte Zeit die Entwicklungszeit ausgleicht (Abbildung 44).

In der Praxis ist eine derart hohe Anzahl an Anwendungen für dieses spezifische Toolset eher unrealistisch, insbesondere da die Spawn-Funktion den größten Anteil an Entwicklungszeit beanspruchte (Abbildung 42), jedoch gleichzeitig der Teil ist, der in der Analyse den geringsten Nutzen zeigte bzw. sogar zu Zeitverlust führte (Abbildung 43). Hinzu kommt, dass die Nutzerfreundlichkeit, wie in der SUS-Auswertung deutlich wurde, viel Optimierungspotential

hat, wodurch weitere Entwicklungszeit für Verbesserungen notwendig wäre (Abbildung 40). Dies würde den BEP zusätzlich erhöhen.

Wären lediglich die Select- und Transform-Funktionen entwickelt worden, könnten diese eine sinnvolle Ergänzung des Level-Design-Prozesses darstellen. Sie lassen sich flexibel einsetzen, sind wenig komplex und beschleunigen wiederkehrende Arbeitsschritte. Je nach Präferenz der Nutzer könnten sie gezielt in den Arbeitsprozess integriert werden.

Das Knochen-Export-Tool amortisiert sich schnell, ist zuverlässig, eliminiert Fehler vollständig und weist eine hohe Nutzerfreundlichkeit auf. Das Level-Design-Toolset hingegen amortisiert sich nur bei sehr häufiger Nutzung und müsste sowohl funktional als auch in Bezug auf die Usability weiter optimiert werden, um einen klaren praktischen Mehrwert zu bieten. Vor diesem Hintergrund stellt sich die Frage, ob der zusätzliche Entwicklungsaufwand in seiner derzeitigen Form gerechtfertigt ist.

7.7. Gesamteinschätzung

Insgesamt lässt sich festhalten, dass der Arbeitsprozess durch den Einsatz von Editor-Utilities verbessert werden kann, dies jedoch nicht zwingend der Fall ist. Ähnlich verhält es sich mit den aufgestellten Hypothesen.

H1 ist in zwei von drei Fällen bestätigt. Die erzielte Zeitersparnis überwiegt jedoch deutlich die Zeitverluste, sodass die Hypothese insgesamt als zutreffend angesehen werden kann. Unter Berücksichtigung der Entwicklungszeit ergibt sich, dass wenig komplexe Tools mit geringer Entwicklungszeit den Arbeitsprozess verkürzen, während komplexe Tools dazu führen können, dass der tatsächliche Nutzen den Entwicklungsaufwand nicht rechtfertigt.

H2 kann teilweise als zutreffend und teilweise als nicht zutreffend bewertet werden. Einfache Tools wie das Knochen-Export-Tool führen zu konsistenten Ergebnissen und reduzieren die Fehlerquote bis auf null. Tools mit mehreren Anwendungsfällen, die stärker von der korrekten Nutzung durch den Anwender abhängen, führen hingegen nicht zwingend zu einer geringeren Fehlerquote.

H3 lässt sich ähnlich wie H2 beantworten. Einfache Tools mit begrenztem Funktionsumfang sind leichter verständlich und intuitiver zu bedienen als komplexe Tools. Besonders ohne vorherige Einweisung sind komplexe Tools kaum effektiv nutzbar.

8. Fazit

Editor-Utilities bieten ein hohes Potenzial zur Automatisierung und Vereinfachung von Arbeitsprozessen. Ihr tatsächlicher Nutzen hängt jedoch maßgeblich von einer sorgfältigen und zielgerichteten Entwicklung ab. Zentral sind dabei eine geringe Einarbeitungszeit, eine intuitive Bedienbarkeit sowie ein angemessenes Verhältnis zwischen Entwicklungsaufwand und funktionalem Nutzen. Besonders effektiv sind Editor-Utilities dann, wenn sie präzise auf eine klar definierte Aufgabe zugeschnitten sind. In solchen Fällen ermöglichen sie eine zuverlässige, schnelle und wiederholbare Bearbeitung ohne große Fehlerquote.

Komplexe Editor-Utilities, die für eine Vielzahl unterschiedlicher Anwendungsfälle ausgelegt sind, können ebenfalls einen Mehrwert bieten, allerdings meist erst ab einer sehr hohen Nutzungshäufigkeit und in Verbindung mit einer gründlichen Einweisung durch die Entwickelnden. Ohne diese Voraussetzungen übersteigen die benötigte Bearbeitungszeit und die potenzielle Fehleranfälligkeit häufig den tatsächlichen Nutzen. Für bestimmte Aufgaben, insbesondere solche, die hohe Präzision erfordern, bleibt die manuelle Bearbeitung weiterhin die effizientere Lösung.

Grundsätzlich ist das Potenzial von Editor-Utilities enorm. Nahezu jeder Arbeitsschritt kann in irgendeiner Form automatisiert oder zumindest vereinfacht werden. Davon profitieren sowohl erfahrene Entwickler als auch Personen mit geringer Erfahrung. Besonders groß ist der Nutzen in Szenarien, in denen viele Assets gleichzeitig bearbeitet werden müssen. Die Skalierbarkeit mit der Projektgröße ist ebenfalls ein großer Vorteil von Editor-Utilities. Je öfter die Tools eingesetzt werden, desto stärker wirkt sich die eingesparte Arbeitszeit aus und desto schneller amortisiert sich die Entwicklung.

Am effektivsten erweisen sich jedoch jene Utilities, die bewusst einfach gehalten sind und eine klar begrenzte Funktion erfüllen. Je fokussierter ein Tool und je unkomplizierter die damit verbundene Aufgabe, desto höher ist der praktische Nutzen. Das Knochen-Export-Tool stellt hierfür ein typisches Beispiel dar. Es ist funktional schlank, exakt auf seine Aufgabe ausgerichtet und erzielt dadurch eine hohe Effizienz bei minimaler Einarbeitungszeit.

9. Quellenverzeichnis

Accelerating Your In-Editor Workflows with Editor-Utilities | GDC 2024 | Talks and demos. (2024, April 26). Epic Games Developer. https://dev.epicgames.com/community/learning/talks-and-demos/5J9b/unreal-engine-accelerating-your-in-editor-workflows-with-editor-utilities-gdc-2024

Byshonkov, D. (2025, Februar 10). Video Game Insights: Game Engines on Steam in 2025 [Substack newsletter]. *GameDev Reports - powered by Xsolla*. https://gamedevreports.substack.com/p/video-game-insights-game-engines

CD Projekt will swap REDengine for Unreal Engine 5 to create the next Witcher saga. (2022, März 22). Game Developer. https://www.gamedeveloper.com/production/cd-projekt-will-swap-redengine-for-unreal-engine-5-to-create-the-next-i-witcher-i-saga

Scripting and Automating the Unreal Editor | Unreal Engine 5.6 Documentation | Epic Developer Community. (o. J.). Epic Games Developer. Abgerufen 13. Oktober 2025, von https://dev.epicgames.com/documentation/en-us/unreal-engine/scripting-and-automating-the-unreal-editor

Steam annual game releases 2024. (2025, Januar). Statista. https://www.statista.com/statistics/552623/number-games-released-steam/

Steam Steam Charts (App 753). (o. J.). SteamDB. Abgerufen 13. Oktober 2025, von https://steamdb.info/app/753/charts/

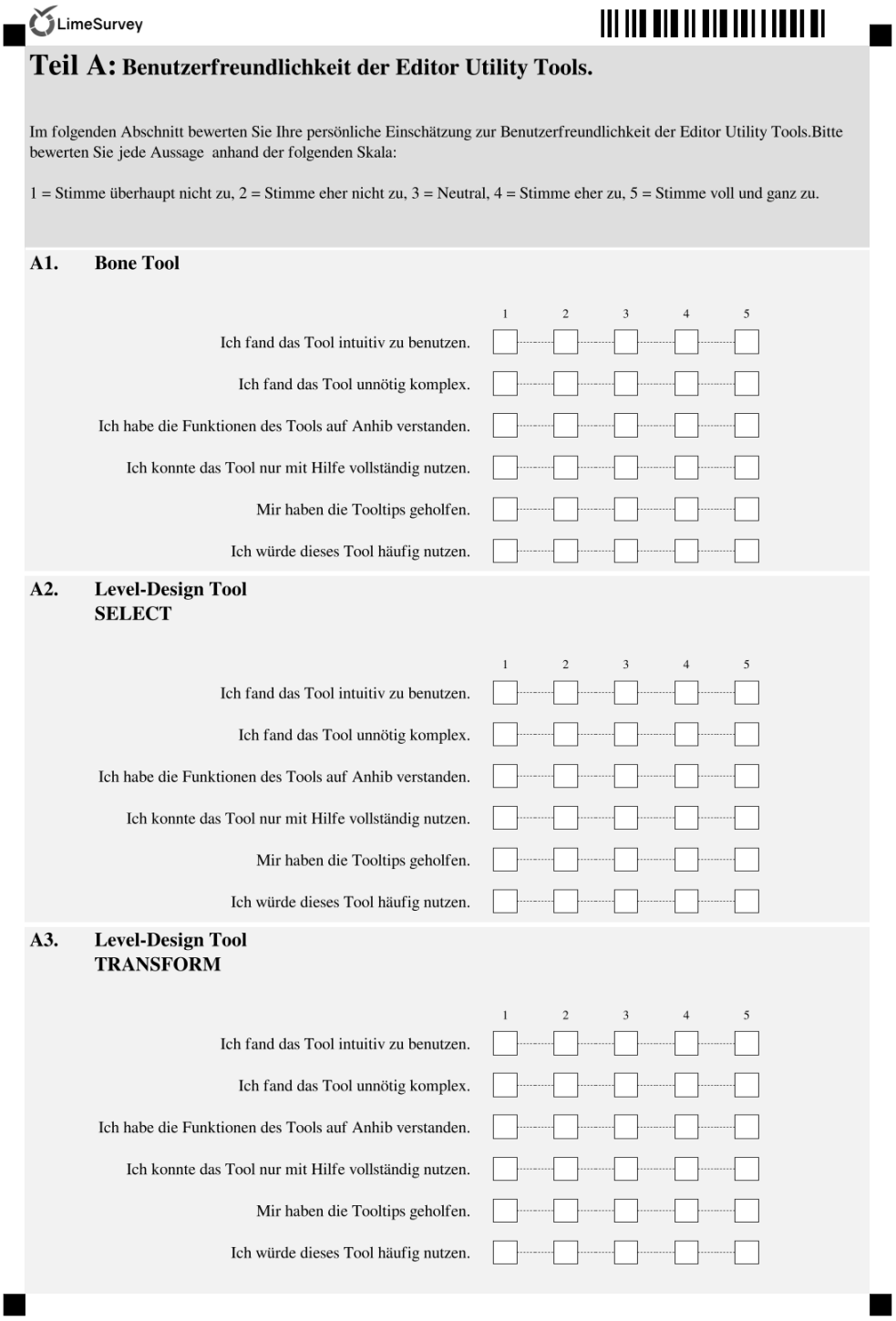
Unreal Engine 5.4 Release Notes | Unreal Engine 5.4 Documentation | Epic Developer Community. (o. J.). Epic Games Developer. Abgerufen 15. Oktober 2025, von https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-release-notes

10. Anhang

10.1. Verwendete Hilfsmittel

ChatGPT 4	Rechtschreib- und Grammatikprüfung
LimeSurvey	Durchführung der Umfrage
Microsoft Excel	Erstellung & Visualisierung der Diagramme
Microsoft Paint	Bearbeitung von Bildern
Zotero	Literaturverwaltung & Quellenverwaltung

10.2. Workflow-Optimierung in der Spieleentwicklung: Entwicklung und Evaluation von Editor-Utilities in Unreal Engine 5



LimeSurvey

Teil A: Benutzerfreundlichkeit der Editor Utility Tools.

Im folgenden Abschnitt bewerten Sie Ihre persönliche Einschätzung zur Benutzerfreundlichkeit der Editor Utility Tools.Bitte bewerten Sie jede Aussage anhand der folgenden Skala:

1 = Stimme überhaupt nicht zu, 2 = Stimme eher nicht zu, 3 = Neutral, 4 = Stimme eher zu, 5 = Stimme voll und ganz zu.

A1. Bone Tool

	1	2	3	4	5
Ich fand das Tool intuitiv zu benutzen.	☐	☐	☐	☐	☐
Ich fand das Tool unnötig komplex.	☐	☐	☐	☐	☐
Ich habe die Funktionen des Tools auf Anhib verstanden.	☐	☐	☐	☐	☐
Ich konnte das Tool nur mit Hilfe vollständig nutzen.	☐	☐	☐	☐	☐
Mir haben die Tooltips geholfen.	☐	☐	☐	☐	☐
Ich würde dieses Tool häufig nutzen.	☐	☐	☐	☐	☐

A2. Level-Design Tool SELECT

	1	2	3	4	5
Ich fand das Tool intuitiv zu benutzen.	☐	☐	☐	☐	☐
Ich fand das Tool unnötig komplex.	☐	☐	☐	☐	☐
Ich habe die Funktionen des Tools auf Anhib verstanden.	☐	☐	☐	☐	☐
Ich konnte das Tool nur mit Hilfe vollständig nutzen.	☐	☐	☐	☐	☐
Mir haben die Tooltips geholfen.	☐	☐	☐	☐	☐
Ich würde dieses Tool häufig nutzen.	☐	☐	☐	☐	☐

A3. Level-Design Tool TRANSFORM

	1	2	3	4	5
Ich fand das Tool intuitiv zu benutzen.	☐	☐	☐	☐	☐
Ich fand das Tool unnötig komplex.	☐	☐	☐	☐	☐
Ich habe die Funktionen des Tools auf Anhib verstanden.	☐	☐	☐	☐	☐
Ich konnte das Tool nur mit Hilfe vollständig nutzen.	☐	☐	☐	☐	☐
Mir haben die Tooltips geholfen.	☐	☐	☐	☐	☐
Ich würde dieses Tool häufig nutzen.	☐	☐	☐	☐	☐

LimeSurvey

A4. Level-Design Tool SPAWN

	1	2	3	4	5
Ich fand das Tool intuitiv zu benutzen.	☐	☐	☐	☐	☐
Ich fand das Tool unnötig komplex.	☐	☐	☐	☐	☐
Ich habe die Funktionen des Tools auf Anhib verstanden.	☐	☐	☐	☐	☐
Ich konnte das Tool nur mit Hilfe vollständig nutzen.	☐	☐	☐	☐	☐
Mir haben die Tooltips geholfen.	☐	☐	☐	☐	☐
Ich würde dieses Tool häufig nutzen.	☐	☐	☐	☐	☐

Teil B: Funktion und Effektivität der Editor Utility Tools

Im folgenden Abschnitt bewerten Sie Ihre persönliche Einschätzung zur Funktionsweise und Effektivität der Editor Utility Tools.Bitte bewerten Sie jede Aussage anhand der folgenden Skala:

1 = Stimme überhaupt nicht zu, 2 = Stimme eher nicht zu, 3 = Neutral, 4 = Stimme eher zu, 5 = Stimme voll und ganz zu.

B1. Mesh-Placement Utility

	1	2	3	4	5
Das Tool hat die Aufgabe schneller erledigt als die manuelle Methode.	☐	☐	☐	☐	☐
Das Tool hat die Zahl der Fehler reduziert.	☐	☐	☐	☐	☐
Ich habe das Tool und seine Parameter schnell verstanden.	☐	☐	☐	☐	☐
Die Platzierungen wirken natürlich und sind gut brauchbar für Level-Design.	☐	☐	☐	☐	☐
Ich würde dieses Tool in einem zukünftigen Projekt verwenden.	☐	☐	☐	☐	☐

B2. Bone-Export Utility

	1	2	3	4	5
Das Tool hat die Aufgabe schneller erledigt als die manuelle Methode.	☐	☐	☐	☐	☐
Das Tool hat die Zahl der Fehler reduziert.	☐	☐	☐	☐	☐
Ich habe das Tool und seine Parameter schnell verstanden.	☐	☐	☐	☐	☐
Ich empfand das Tool als zuverlässig.	☐	☐	☐	☐	☐
Ich würde dieses Tool in einem zukünftigen Projekt verwenden.	☐	☐	☐	☐	☐

Hochschule Neu-Ulm
University of Applied Sciences

LimeSurvey

Teil C: Vergleich von Tool vs. Manuell

Im folgenden Abschnitt geben Sie Ihre Bewertung zum Vergleich beider Methoden ab.Bitte bewerten Sie jede Aussage anhand der folgenden Skala:

1 = Stimme überhaupt nicht zu, 2 = Stimme eher nicht zu, 3 = Neutral, 4 = Stimme eher zu, 5 = Stimme voll und ganz zu.

C1.

Mesh-Placement Vergleich Tool vs. Manuell

Assets massenhaft in der Luft generieren

	1	2	3	4	5
Das Tool war insgesamt schneller als die manuelle Methode.	☐	☐	☐	☐	☐
Die Qualität des Ergebnisses war besser mit dem Tool als manuell.	☐	☐	☐	☐	☐
Die manuelle Methode gab mir mehr Kontrolle.	☐	☐	☐	☐	☐
Ich würde in Zukunft für diese Aufgabe eher das Tool als die manuelle Methode verwenden.	☐	☐	☐	☐	☐

C2.

Mesh-Placement Vergleich Tool vs. Manuell

Assets massenhaft auf bestimmten Oberflächen generieren

	1	2	3	4	5
Das Tool war insgesamt schneller als die manuelle Methode.	☐	☐	☐	☐	☐
Die Qualität des Ergebnisses war besser mit dem Tool als manuell.	☐	☐	☐	☐	☐
Die manuelle Methode gab mir mehr Kontrolle.	☐	☐	☐	☐	☐
Ich würde in Zukunft für diese Aufgabe eher das Tool als die manuelle Methode verwenden.	☐	☐	☐	☐	☐

C3.

Bone-Export Vergleich Tool vs. Manuell

	1	2	3	4	5
Das Tool war insgesamt schneller als die manuelle Methode.	☐	☐	☐	☐	☐
Die Vollständigkeit/Korrektheit war besser mit dem Tool als manuell.	☐	☐	☐	☐	☐
Die manuelle Methode gab mir mehr Kontrolle.	☐	☐	☐	☐	☐
Ich würde in Zukunft für diese Aufgabe eher das Tool als die manuelle Methode verwenden.	☐	☐	☐	☐	☐

Hochschule Neu-Ulm
University of Applied Sciences

LimeSurvey

Teil D: Offene Fragen

D1. **Was hat Ihnen an den Tools am besten gefallen? (Kurz)**

D2. **Welche Probleme oder Missverständnisse sind aufgetreten? (Kurz)**

D3. **Welche Verbesserungen würden Sie vorschlagen? (Kurz)**

D4. **Wie viele Jahre Erfahrung haben Sie insgesamt mit der Unreal Engine?**

- [ ] Gar keine
- [ ] Weniger als 1 Jahr
- [ ] Zwischen 1 und 2 Jahre
- [ ] Zwischen 2 und 4 Jahren
- [ ] Mehr als 4 Jahre

D5. **Haben Sie bereits Erfahrung mit Editor-Utilities gehabt?**

- [ ] Nein, noch keine.
- [ ] Wenn ja, bitte im Kommentar beschreiben

Hochschule Neu-Ulm
University of Applied Sciences

LimeSurvey

D6.

Ich bin damit einverstanden, dass meine anonymisierten Antworten für die Auswertung in der Bachelorarbeit verwendet werden. ☐