Bachelorarbeit
im Bachelorstudiengang
Game-Produktion und Management
an der Hochschule für angewandte Wissenschaften Neu-Ulm

Spieleerstellung lehren und lernen
Evaluation unterschiedlicher Game Engines bezüglich ihres Einsatzes in der Lehre

Erstkorrektor/-in: Prof. Dr. Erica Weilemann
Betreuer/-in: Prof. Michael Hebel

Verfasser/-in: Junes Albert (Matrikel-Nr.: 287839)

Thema erhalten: 20.02.2024
Arbeit abgegeben: 04.06.2024

Hochschule Neu-Ulm
University of Applied Sciences

Danksagung

Danke an Herrn Sebastian De Andrade, Herrn Professor Jirka Dell'Oro-Friedl, Herrn Professor Sylvius Lack, Herrn Professor Dr. Jochen Koubek, Herrn Professor Dr. Tobias Breiner, Herrn Professor Dr. Christof Rezk-Salama, Herrn Professor Ralf Hebecker, Herrn Dr. Stefan Göbel und Herrn Professor Dr. Markus Wacker für die Unterstützung dieser Arbeit durch Ihre aufschlussreiche Teilnahme an den durchgeführten Experteninterviews.

Darüber hinaus gilt ein besonderer Dank Frau Professorin Dr. Erica Weilemann und Herrn Professor Michael Hebel für die regelmäßige und sehr hilfreiche Unterstützung dieser Arbeit.

Gender-Hinweis

Zur besseren Lesbarkeit der vorliegenden Arbeit wird auf die Nutzung von Kurzbezeichnungen (Binnenmajuskel, Gendersternchen, Genderdoppelpunkte und Gendergaps) bei Personenbezeichnungen verzichtet. Stattdessen wird, wo möglich, die geschlechtsneutrale Bezeichnung für Personenbezeichnungen genutzt (beispielsweise **Spielende** statt **Spieler** oder **Spielerinnen**), die sich stets auf alle Geschlechter und Geschlechtsidentitäten bezieht.

Abstract

Die vorliegende Bachelorarbeit befasst sich mit der Evaluation unterschiedlicher Game Engines bezüglich ihres Einsatzes im Lehrumfeld für die Ausbildung von Spieleentwickelnden. Ausgelöst durch die Änderungen im Preismodell der *Unity* Game Engine im September 2023 hat sich an unterschiedlichen deutschen Hochschulen in den vergangenen Monaten die Meinung bezüglich der Nutzung dieser Game Engine geändert. Entsprechend sind hier neuerdings alternative Angebote im Einsatz. Allgemein ist es das Ziel der jeweiligen Lehrumfelder den Lernenden eine sinnvolle Ausbildung zu bieten, die im tatsächlichen Berufsumfeld auch Anwendung findet oder zumindest leicht auf die aktuell dort vorzufindenden Herausforderungen übertragbar ist. Welche Game Engines hierfür potenziell in Frage kommen und wie gut diese dann tatsächlich für den Einsatz in der Lehre der Spieleerstellung geeignet sind wird in der vorliegenden Bachelorarbeit beantwortet. Einer Aufstellung einzelner Qualifikations- und Evaluationskriterien, basierend auf Befragungen von Lehrenden, folgt die Evaluation von insgesamt zehn Game Engines. All diese Game Engines sind im Kontext der Lehre grundsätzlich sinnvoll einsetzbar, für bestimmte Anwendungsfälle sind einige davon aber besser geeignet als andere. Entsprechend können die Ergebnisse dieser Arbeit sowie die grundsätzliche Vorgehensweise der Evaluation Dozierenden als Orientierungshilfe zum zukünftigen Einsatz von Game Engines in der Lehre der Spieleerstellung dienen.

Hochschule Neu-Ulm
University of Applied Sciences

Inhaltsverzeichnis

Hochschule Neu-Ulm
University of Applied Sciences

Tabellenverzeichnis

Hochschule Neu-Ulm
University of Applied Sciences

Abbildungsverzeichnis

Glossar

API – **Application Programming Interface** (dt. etwa: Programmierschnittstelle) fasst Befehle beziehungsweise Funktionen zur Erstellung von oder Kommunikation mit Software(systemen) zusammen, sodass Entwickelnde für häufig verwendete Operationen keinen neuen Code schreiben müssen[1]

Blender – Software zur vollständigen Erstellung von 3D-Modellen[2]

Commit – Speichern einer Datei(änderung) in einer *Repository*

Craftomation 101 – Spiel, das in *Defold* erstellt wurde[3]

C# – Programmiersprache von Microsoft[4]

C++ – Programmiersprache von Bjarne Stroustrup[5]

Depthris – Spiel, das in *jMonkeyEngine* erstellt wurde[6]

Dreams – Plattform, auf der nutzende kreative Werke erstellen und mit anderen teilen können[7]

Epic Games Store – Vertriebsplattform für Videospiele von Epic Games[8]

Fortnite – Spiele-Plattform von Epic Games, auf der Nutzende Spiele erstellen und mit anderen teilen können sowie von Epic Games erstellte Spielmodi spielen können[9]

game Verband der deutschen Games-Branche – Zusammenschluss verschiedener Beteiligter der deutschen Games-Branche aus Entwicklungs-Studios, Publisher, E-Sport-Organisationen und weiteren Mitgliedern dieser Branche[10]

GDScript – Programmiersprache, die für *Godot* entwickelt wurde und darin eingesetzt wird[11]

Git – Open Source System zur Versionsverwaltung[12]

[1] vgl. Talend o. J.
[2] vgl. Blender o. J.
[3] vgl. SteamDB o. J.d
[4] vgl. ComputerWeekly o. J.
[5] vgl. W3Schools o. J.a
[6] vgl. codewalker o. J.
[7] Playstation o. J.
[8] vgl. McCall o. J.
[9] vgl. Epic Games o. J.u
[10] vgl. game o. J.
[11] vgl. Godot o. J.i
[12] vgl. Git-scm o. J.a

gitattributes – Datei, die bestimmte Attribute für einen bestimmten Pfad einer *Repository* definiert[13]

GitHub – Dienst zur Verwaltung von *Repositories* mit *Git*[14]

gitignore – Datei, die angibt, welche Dateien für eine Aufnahme in eine *Repository* nicht verfolgt werden sollen[15]

GMLCode – Programmiersprache, die für *GameMaker* entwickelt wurde und darin eingesetzt wird[16]

GMLVisual – visuelle Programmiersprache, die für *GameMaker* entwickelt wurde und darin eingesetzt wird[17]

GX.games – Vertriebsplattform für Videospiele von *GameMaker* und Opera GX[18]

(*GitHub*) *Issue* – Element zur Planung, Besprechung und Nachverfolgung von Arbeit an einem (*GitHub*)-Projekt[19]

Itch.io – Vertriebsplattform für Videospiele[20]

Java – Programmiersprache von Oracle[21]

JavaScript – Programmiersprache von Brendan Eich[22]

Lua – Programmiersprache von Roberto Ierusalimschy, Luiz Henrique de Figueiredo und Waldemar Celes[23]

Murders on the Yangtze River – Spiel, das in *Unity* erstellt wurde[24]

Mystery Society 2: Hidden Puzzles – Spiel, das in *Cocos Creator* erstellt wurde[25]

Obscured by the Night – Spiel, das in *Flax* erstellt wurde[26]

[13] vgl. Git-scm o. J.b
[14] vgl. Kinsta 2023
[15] vgl. Git-scm o. J.c
[16] vgl. GameMaker o. J.n
[17] vgl. GameMaker o. J.p
[18] vgl. kseniias 2022
[19] vgl. GitHub o. J.
[20] vgl. Rosenberg 2021
[21] vgl. GIGA 2021
[22] vgl. W3Schools o. J.c
[23] vgl. Ierusalimschy/Figueiredo/Celes o. J.
[24] vgl. SteamDB o. J.b
[25] vgl. SteamDB o. J.e
[26] vgl. Itch o. J.

Open Source – Sofware (oder Teile davon), die öffentlich zugänglich und damit beliebig bearbeitbar, erweiterbar und verwendbar ist[27]

Palworld – Spiel, das in *Unreal Engine* erstellt wurde[28]

Perforce – Entwicklungsfirma für unterschiedliche Software, unter anderem zur Versionsverwaltung[29]

Python – Programmiersprache von Guido van Rossum[30]

Repository – Ablage beziehungsweise Sammlung von Daten, bei *GitHub* konkret Quellcode[31]

Roblox – Plattform, auf der Nutzende Spiele erstellen und mit anderen teilen können[32]

Steam – Vertriebsplattform für Videospiele von Valve[33]

Subversion – *Open Source* System zur Versionsverwaltung[34]

Super Mario Maker – Baukasten-Spiel, das Nutzenden erlaubt Super Mario Level zu erstellen, zu spielen und miteinander zu teilen[35]

Trackmania – Free2Play-Rennspiel, das zusätzlich Nutzenden erlaubt, eigene Strecken zu erstellen und miteinander zu teilen[36]

Turnip Boy Robs a Bank – Spiel das in *GameMaker* erstellt wurde[37]

TypeScript – Programmiersprache, die *Java*Script um bestimmte Syntax erweitert[38]

20 Small Mazes – Spiel, das in *Godot* erstellt wurde[39]

[27] vgl. Red Hat o. J.
[28] vgl. SteamDB o. J.f
[29] vgl. Perforce o. J.
[30] vgl. W3Schools o. J.b
[31] vgl. Kinsta 2023
[32] vgl. schau-hin o. J.
[33] vgl. GIGA 2019
[34] vgl. Apache o. J.
[35] vgl. fandom 2024
[36] vgl. Klinge 2020
[37] vgl. SteamDB o. J.a
[38] vgl. W3Schools o. J.d
[39] vgl. SteamDB o. J.c

1. Einleitung

Game Engines haben inzwischen einen festen Platz in der Lehre der Spieleerstellung gefunden. Unabhängig des konkreten Studiengangs werden Game Engines immer dann benutzt, wenn die Erstellung eines Spiels den Hauptinhalt der zugehörigen Lehreinheit darstellt. Der wohl wichtigste Grund für die Arbeit mit Game Engines in diesem Umfeld ist ihre – im Vergleich zur Spieleprogrammierung von Grund auf – schnelle Erlern- und Anwendbarkeit, nicht nur bezüglich der Programmierung, sondern auch der Implementierung von Art- und Sound-Assets, sowie der Erstellung von einzelnen Spielelementen wie Levels, Zwischensequenzen und weiterem.

Darüber hinaus bietet das Erlenen von konkreten Programmen, die zur Spieleerstellung genutzt werden können – wobei hier nicht nur Game Engines, sondern auch Zeichenprogramme, 3D-Grafik-Programme oder Kompositionssoftware für Musik gemeint sind – eine gute Vorbereitung auf spätere Berufswege im Games-Umfeld. Viele Programme finden dort entweder direkten Einsatz, werden in abgewandelter Form verwendet oder einzelne Funktionen dieser sind in gleichwertigen Programmen ebenso vorzufinden.

Gleichzeitig gibt es inzwischen eine schwer überschaubare Menge an Game Engines, wodurch Lehrende mit der komplexen Aufgabe der Auswahl der richtigen Game Engine für den jeweiligen Lehrkontext konfrontiert sind. Hierbei stößt man teilweise aber auf das Problem, dass viele Game Engines, der Verfügung der jeweiligen Herausgebenden unterliegen. Dies kann dazu führen, dass zum Beispiel Preisänderungen, Verwendungseinschränkungen oder andere restriktive Änderungen an den Nutzungsbedingungen der jeweilige Game Engine die Nutzbarkeit dieser in der Lehre einschränken oder die Game Engine möglicherweise sogar komplett für den Einsatz in der Lehre unbrauchbar machen. Alternativen hierzu sind dann entweder *Open Source* oder kostenfrei verfügbare Game Engines, teilweise auch Custom Game Engines, die von einzelnen Lehrenden oder Lehrinstitutionen explizit für das eigene Lehrumfeld und den zugehörigen Lehrplan konzipiert und erstellt wurden.

Hieraus stellt sich für Lehrende die Frage, welche Game Engines potenziell für den Einsatz im eigenen Lehrkontext in Frage kommen und wie hieraus die bestmöglich passendste Game Engine ausgewählt werden. Diese Frage versucht die vorliegende Bachelorarbeit zu beantworten.

Hierfür wird zunächst ein Kriterienkatalog aufgestellt, der sich neben generellen Qualifikationskriterien, die die grundlegende Eignung einer Game Engine für den Einsatz in der Lehre bestimmen, vor allem auf die Erlernbarkeit, sowie darüber hinaus die

Anwendbarkeit und die Übertragbarkeit einer Game Engine konzentriert. Ergänzt wird dies durch eine analytische Zusammenfassung der aktuellen Spiele-Lehre in Deutschland, die auf qualitativen Experteninterviews mit Lehrenden basiert. Anschließend werden einzelne Game Engines aus folgenden drei Kategorien untersucht und evaluiert:

- Teilweise kostenpflichtige Game Engines: Die Nutzung dieser Game Engines ist von den jeweiligen Herausgebenden teilweise eingeschränkt, was sich zum Beispiel in umsatzbezogenen Lizenzgebühren äußern kann.
- Kostenfreie Game Engines: Diese Kategorie umfasst alle Game Engines, deren Nutzung grundsätzlich kostenfrei ist. Darüber hinaus lassen sich hier auch *Open Source* Game Engines finden, deren Quellcode öffentlich einsehbar ist, wodurch eine Nutzung und Änderung nicht durch die Herausgebenden eingeschränkt ist.
- Custom Game Engines: Hier werden alle Game Engines zusammengefasst, die eigens für ein spezifisches Lehrumfeld erstellt wurden und in diesem eingesetzt werden.

Hierauf folgt ein zusammenfassender Vergleich der evaluierten Game Engines, bevor jegliche Ergebnisse und mögliche Handlungsempfehlungen in einem Fazit zusammengefasst werden.

1.1 Forschungshintergrund

Während bereits einige Veröffentlichung zur Evaluation von Game Engines existieren, ist dabei keine konkret auf den Einsatz von Game Engines in der Lehre spezialisiert. Häufig werden Game Engines bezüglich ihrer Einsatzmöglichkeiten für die Erstellung von Spielen eines bestimmten Genres [40], bezüglich ihrer Nutzbarkeit auf oder für bestimmte Plattformen[41] oder ihrer generellen Einsatzmöglichkeiten für die Spieleproduktion und die zugehörige Skalierbarkeit miteinander verglichen.[42]

Darüber hinaus wird bei genannten Vergleichen entweder ein größeres Augenmerk auf die Aufstellung der jeweils verwendeten Kriterien als auf die tatsächliche Evaluation mit Hilfe dieser gelegt [43] oder Kriterien gewählt, die einen möglichst objektiven Vergleich ermöglichen.[44] Während ersteres für einen technisch detaillierten Vergleich bezüglich der Leistungsfähigkeit einzelner Game Engines sinnvoll ist, leistet dieser Ansatz keine Aussage

[40] vgl. Petridis/Dunwell/Panzoli 2012
[41] vgl. Halsas 2017
[42] vgl. Ong o. J.
[43] vgl. Pattrasitidecha 2014
[44] vgl. Halsas 2017

über den tatsächlichen Umgang mit der Game Engine – angefangen mit den ersten Schritten bis hin zu einem vollständig spielbaren Spiel. Bezüglich diesem konkreten „Hands on“ sind letztgenannte objektiven Vergleichsparametern wie Ladezeit und Performanz einer Game Engine zwar wichtig für den Einsatz dieser in einer Spieleproduktion, der Kontext der Erlernung wird dabei aber außen vorgelassen. Häufig werden in der genannten Evaluation auch Antwortskalen (teilweise auch in Form von Likert-Skalen[45]) verwendet, was zwar eine schnelle und leicht vergleichbare Evaluation ermöglicht, Details und empirische Forschungsergebnisse aber auslässt.[46] Diese sind für den sinnvollen Einsatz von Game Engines im Kontext der Lehre aber unabdingbar, da der konkrete Umgang mit dem Lehr- und Lernmaterial Game Engine den durch Lehrende und Lernende gestellten Erwartungen und Anforderungen gerecht werden muss.

1.2 Forschungsfrage und Forschungsziel

Ausgehend von diesen bereits existierenden Evaluationen von Game Engines bezüglich ihres Einsatzes in der Spieleproduktion versucht diese Arbeit konkret die Einsatzmöglichkeiten von Game Engines für die Lehre der Spielererstellung zu ergründen. Zusammengefasst wird dieses Vorhaben in der Forschungsfrage: *Welche Game Engine eignet sich (am besten) für den Einsatz in der Lehre der Spielerstellung?*

Die Einklammerung in der Forschungsfrage bezieht sich darauf, dass im Rahmen dieser Arbeit eigentlich zwei Fragen beantwortet werden sollen; zunächst soll herausgefunden werden, welche Game Engines sich grundsätzlich für den Einsatz in der Lehre eignen und in einem weiteren Schritt dann konkretisiert werden, welche von diesen Game Engines für bestimmte konkrete Einsatzbereiche besser oder sogar am besten im Vergleich zu allen anderen evaluierten Game Engines für die Lehre der Spieleerstellung geeignet ist.

Ziel dieser Arbeit ist es darüber hinaus, einen detaillierten Überblick über möglichst viele unterschiedliche Game Engines zu erschaffen und durch Vergleiche basierend auf einheitlichen Kriterien Vor- und Nachteile dieser Game Engines ihre Eignung für die Lehre im Allgemeinen sowie den Einsatz in unterschiedlichen Lehrkontexten und eventuelle Einschränkungen hierfür aufzuzeigen. Hierbei steht nicht die tatsächliche Leistungsfähigkeit für den konkreten Spieleerstellungsprozess im Vordergrund, sondern die Lehr- und Lernmöglichkeiten, die sich durch den Einsatz der Game Engine in der Lehre ergeben. Hieraus sollen Lehrende direkt ableiten können, welche Game Engine sich für den Einsatz im jeweiligen Lehrplan bestmöglich eignet.

[45] vgl. Qualtrics o. J.
[46] Pattrasitidecha 2014

1.3 Forschungsmethode

1.3.1 Literaturrecherche

Game Engines werden im Rahmen dieser Arbeit als interaktive und digitale Lehr- und Lernmittel angesehen, da sie sowohl in der Lehrvermittlung durch Dozierende als auch im Selbstlernprozess von Studierenden eingesetzt werden können und der Umgang damit stets in einer Interaktion zwischen Mensch und Software stattfindet.

In Deutschland gibt es bisher keine allgemeingültigen Kriterien zur Auswahl von ebensolchen digitalen Lehrmitteln für den Einsatz in der Spieleerstellung an Hochschulen, Universitäten und Ausbildungsstätten. Die Erstellung der Liste von Qualifikations- und Evaluationskriterien basiert deshalb auf einer Literaturrecherche mit besonderem Augenmerk auf der Evaluation von Lehrmitteln im Allgemeinen. Zusätzlich werden hier ebenfalls Kriterien aus den bereits referenzierten Veröffentlichungen zur Einsatzevaluation von Game Engines in der Spieleerstellung außerhalb von Lehrkontexten beleuchtet.

1.3.2 Qualitative Experteninterviews

Ergänzend hierzu werden Meinungen von Lehrenden in diesem Feld durch eine Datenerhebung in Form von **qualitativen Experteninterviews** nach Kaiser miteinbezogen und als Grundlage beziehungsweise Begründung der aufgestellten Kriterien verwendet. Die Form der Interviews ist dabei semistrukturiert, da ein Leitfaden in Form eines Fragebogens existiert, dieser aber nicht in jedem Interview vollständig abgearbeitet werden muss. Gleichzeitig kann je nach Inhalt der einzelnen Antworten vom Fragebogen abgewichen werden. Hierdurch können Besonderheiten in einzelnen Herangehensweisen der interviewten Experten hervorgehoben und näher beleuchtet werden, wodurch gleichzeitig Erkenntnisse auftreten können, die durch die strikte Abarbeitung eines standardisierten Fragebogens nicht angesprochen werden würden – diese Abweichung vom definierten Fragebogen muss auch nach Kaiser ausdrücklich möglich sein.[47] Die Aufstellung der einzelnen Interviewfragen leitet sich aus der von Kaiser beschriebenen Vorgehensweise ab[48]; zunächst wurden aus der zuvor genannten Forschungsfrage (*Welche Game Engine eignet sich (am besten) für den Einsatz in der Lehre der Spieleerstellung?*) entsprechende **Analysedimensionen** abgeleitet. Konkret handelt es sich hierbei um die folgenden Analysedimensionen: Spieleerstellung, Game Engines, Lehre, Lehrmitteleinsatz.

Aus diesen Analysedimensionen lassen sich wiederum **Fragenkomplexe** ableiten, die in der folgenden Tabelle zusammengefasst sind:

[47] vgl. Kaiser 2021 S. 66
[48] vgl. Kaiser 2021 S. 69

Analysedimension	Fragenkomplexe
Spieleerstellung	Kontexte der Spieleerstellung
	Herausforderungen der Spielerstellung
	Herangehensweisen an die Spieleerstellung
Game Engines	Definition von Game Engines
	Verfügbarkeit von Game Engines
	Einsatzgebiete von Game Engines
Lehre	Ziele der Lehre
	Kontext der Lehre
Lehrmitteleinsatz	Kriterien für die Lehrmittelauswahl
	Dimension von Lehrmitteln (Menge, Vielfalt, …)

Tabelle 1 Analysedimensionen und Fragekomplexe der qualitativen Experteninterviews

Abschließend wurden aus den aufgelisteten Fragenkomplexen konkrete **Interviewfragen** abgeleitet, die in der folgenden Tabelle aufgelistet sind:

Fragenkomplex	Nr.	Frage (sowie weiterführende und ergänzende Fragen)
Kontext der Lehre, Ziele der Lehre, Kontexte der Spieleerstellung	1	In welchem Lehrkontext (Studiengang bzw. Studiengangsmodul) setzen Sie Game Engine(s) ein?
	1.1	*Weiterführend:* Was sind die allgemeinen Inhalte und Ziele dieses Lehrkontexts?
	1.2	*Weiterführend:* Wofür werden Game Engines innerhalb dieses Lehrkontexts eingesetzt?
Dimension von Lehrmitteln, Einsatzgebiete von Game Engines	2	Welche Game Engine(s) setzen Sie aktuell in diesem Lehrkontext ein?
	2.1	*Weiterführend:* Haben Sie in der Vergangenheit andere Game Engines in diesem Lehrkontext eingesetzt?
	2.1.1	*Ergänzend:* Falls ja, welche?
	2.1.2	*Ergänzend:* Warum setzen Sie diese nicht mehr ein?
	2.2	*Weiterführend:* Planen Sie in Zukunft eine oder mehrere andere Game Engines in diesem Lehrkontext einzusetzen?
	2.2.1	*Ergänzend:* Falls ja, welche?
	2.2.2	*Ergänzend:* Falls nein, warum nicht?
	2.2.3	*Ergänzend:* Aus welchen Gründen würden Sie eine oder mehrere andere Game Engines in diesem Lehrkontext einsetzen?

Verfügbarkeit von Game Engines, Dimension von Lehrmitteln, Kriterien für die Lehrmittelauswahl	3	Welche Game Engine(s) standen für den Einsatz im aktuellen Semester in diesem Lehrkontext für Sie zur Auswahl?
	3.1	*Weiterführend:* Aus welchen Gründen standen diese Game Engine(s) für den Einsatz im aktuellen Semester in diesem Lehrkontext zur Auswahl?
	3.2	*Weiterführend:* Wie häufig reevaluieren Sie die Game Engine(s), die in diesem Lehrkontext eingesetzt wird/werden?
	3.2.1	*Ergänzend*: Was war das konkrete Ergebnis der letzten Reevaluation?
Kriterien für die Lehrmittelauswahl	4	Welche Kriterien haben Sie zur Auswahl der aktuell eingesetzten Game Engine(s) für diesen Lehrkontext aufgestellt?
	4.1	*Weiterführend:* Aus welchen Gründen haben Sie diese Kriterien zur Auswahl der aktuell eingesetzten Game Engine(s) für diesen Lehrkontext aufgestellt?
	4.2	*Weiterführend:* Wie genau sah der Auswahlprozess für die aktuell eingesetzten Game Engine(s) für diesen Lehrkontext aus? (Beteiligte Personen, Aufgewandte Zeit, Evaluiert Game Engines und weiteres)
Herangehensweisen an die Spieleerstellung	5	Wie sieht der konkrete Einsatz von Game Engine(s) in diesem Lehrkontext aus? (Aufgaben, Projekte, o.ä., die von Studierenden mithilfe dieser Game Engine(s) erfüllt werden sollen)
	5.1	*Weiterführend:* Welche konkreten Hilfsmaterialien setzen Sie in Verbindung hiermit ein? (Arbeitsblätter, *YouTube*-Tutorials, o.ä.)
Herausforderungen der Spieleerstellung	6	Mit welchen Einschränkungen und/oder Hürden sind Sie aufgrund der eingesetzten Game Engine(s) in diesem Lehrkontext konfrontiert?
	6.1	*Weiterführend:* Wie gehen sie mit diesen Einschränkungen beziehungsweise Hürden um?

Tabelle 2 Fragenkomplexe und Fragen der qualitativen Experteninterviews

Die Auswahl der Interviewpartner erfolgte über eine Recherche bezüglich deutscher Studiengänge und Ausbildungsplätze im Kontext der Spielerstellung. Hierfür wurde die

Gamesmap des *game Verbands der deutschen Games-Branche* betrachtet und Hochschulen, Universitäten sowie Ausbildungsstätten herausgesucht, die Studiengänge oder Ausbildungsplätze im Game Bereich oder mit Bezug zu diesem anbieten und hierbei Game Engines einsetzen.[49] Hieraus entstand eine Liste von insgesamt 40 Experten und möglichen Ansprechpersonen, die alle bezüglich ihrer Verfügbarkeit für ein Experteninterview angefragt wurden. Hiervon gaben zunächst 13 Personen eine positive Rückmeldung, letztendlich wurden zehn Personen interviewt. Die folgende Tabelle bietet eine Übersicht der interviewten Personen, sowie die zugehörige Hochschule, Universität oder Ausbildungsstätte an der und den Lehrkontext in dem sie tätig sind:

Hochschule, Universität, Ausbildungsstätte	**Lehrkontext**	**Name der Lehrperson**	**Referenz-kürzel**
Hochschule für angewandte Wissenschaften Neu-Ulm	Bachelorstudiengang Game-Produktion und Management	Sebastian De Andrade	I0
Anonym	Bachelorstudiengang Interaktive Medien und Masterstudiengang Interaktive Mediensysteme	*Anonym*	I1
Hochschule Furtwangen	Bachelorstudiengang Games & Immersive Media	Jirka Dell'Oro-Friedl	I2
Mediadesign Hochschule Berlin	Bachelorstudiengang Game Design	Sylvius Lack	I3
Universität Bayreuth	Bachelorstudiengang _ und Masterstudiengang Computerspielwissenschaften	Jochen Koubek	I4
Hochschule Kempten	Bachelorstudiengang Games Engineering und Masterstudiengang Games Engineering and Visual Computing	Tobias Breiner	I5
Hochschule Trier	Bachelorstudiengang Informatik – Digitale Medien und Spiele	Christof Rezk-Salama	I6

[49] vgl. Gamesmap o. J.

Hochschule für angewandte Wissenschaften Hamburg	Masterstudiengang Zeitabhängige Medien / Sound – Vision - Games	Ralf Hebecker	I7
Technische Universität Darmstadt	AG Serious Games	Stefan Göbel	I8
Hochschule Dresden	Bachelorstudiengang Medieninformatik	Markus Wacker	I9

Tabelle 3 Übersicht der befragten Dozenten sowie zugehörige Bildungsstätte und Lehrkontext

Die Interviews wurden über die Plattform *Zoom* abgehalten und aufgezeichnet. Anschließend wurden sie mithilfe der Software *Whisper* automatisch transkribiert. Die Transkription wurde daraufhin händisch auf eventuelle Fehler überprüft und gekürzt. Gekürzt wurden hierbei:

- Wortwiederholungen („[…] das, das, das ist […]“) und Ansammlungen von Füllwörtern („[…] ähm, äh, äh […]“), die den Lesefluss stark beeinträchtigen
- Abschweifungen, die nicht der Beantwortung der Frage dienen
- Nachfragen, Nebensätze oder ähnliches, die nicht der Beantwortung der Frage dienen oder den Lesefluss beeinträchtigen
- Namen anderer Personen und sonstige private Informationen beziehungsweise Informationen, die dem Datenschutz unterliegen könnten

Gekürzte Stellen sind in den Transkripten jeweils mit […] gekennzeichnet. Ebenfalls wurden an manchen Stellen Einfügungen vorgenommen, um den jeweiligen Inhalt verständlicher zu machen. Diese sind ebenfalls mit eckigen Klammern gekennzeichnet. Nach der Bearbeitung der Transkripte wurden diese den Interviewpartnern übergeben, von diesen erneut teilweise in Form von einzelnen Wortstreichungen oder -ergänzungen bearbeitet und ein entsprechendes schriftliches Einverständnis mit der Verwendung der Transkripte innerhalb dieser Arbeit gegeben.

Die transkribierten Interviews wurden daraufhin mithilfe der Software *MAXQDA* und der **induktiven Kategorienbildung** folgend einer **qualitativen Inhaltsanalyse** nach Mayring unterzogen. [50] Die induktive Kategorienbildung baut auf der zusammenfassenden

[50] vgl. Mayring 2022 S. 84 ff.

qualitativen Inhaltsanalyse auf, dient der Abbildung von Material[51] und wird im Rahmen dieser Arbeit verwendet, um in Kapitel 3 einen Überblick über die aktuell eingesetzten Game Engines, zugehörige Kriterien, eventuelle Auswahlprozesse und weitere Aspekte der momentanen Lehre der Spieleerstellung in Deutschland zusammenzufassen. Der erste Schritt der induktiven Kategorienbildung dient dem Aufstellen einer **Fragestellung**.[52] Diese lautet unter Berücksichtigung des zuvor genannten Verwendungsplans des Materials: *Welche Game Engines werden aufgrund welcher Kriterien aktuell in der Lehre der Spieleerstellung in Deutschland eingesetzt?*

Hierauf folgte die Aufstellung von Regeln zur Kategorienbildung in Form der **Kategoriendefinition** und des **Abstraktionsniveaus**. Die Kategoriendefinition dient als Selektionskriterium und bestimmt, welches Ausgangsmaterial zur weiteren Kategorienbildung verwendet werden soll. Im Rahmen dieser Arbeit wurden hierfür die Transkripte der durchgeführten qualitativen Experteninterviews vollständig verwendet, wobei ein besonderer Fokus auf die Beleuchtung der jeweiligen Antworten zu den Fragen 3 und 4 lag. Bezüglich des Abstraktionsniveaus wurde vor Beginn der Analyse festgelegt, dass der Einsatzkontext einzelner Game Engines, Erklärungen von Kriterien bezüglich ihrer Auswahl, eventuelle Einschränkungen von eingesetzten Game Engines und mögliche ungenutzte Vorteile von nicht eingesetzten Game Engines zur entsprechenden Kategorisierung beziehungsweise Codierung (die Begriffe werden im Folgenden synonym verwendet) verwendet werden. Entsprechend werden Detailerklärungen zum Einsatz von Game Engines nicht betrachtet und auch Details zu Inhalten und Zielen des jeweiligen Lehrkontexts, die über eine grobe Einordnung hinausgehen, werden nicht näher analysiert – der Fokus liegt also auf einzelnen Game Engines und ihrer Eignung für die Lehre. Basierend auf diesen Regeln wurde das verfügbare Material entsprechend mithilfe der Software *MAXQDA* durchgearbeitet sowie entspreche Codierungen herausgearbeitet und in Kapitel 3 analysiert und interpretiert.

1.3.3 Zusammenfassung

Die konkrete Evaluation der einzelnen Game Engines in Kapitel 4 basiert zusammenfassend also zunächst auf der Betrachtung und Auswertung bezüglich der in Kapitel 2 aufgestellten und mithilfe der Ergebnisse aus der Literaturrecherche und den qualitativen Experteninterviews gestützten Evaluationskriterien. Gleichzeitig werden, wo möglich, einzelne Meinungen der befragten Lehrenden zu der jeweils evaluierten Game Engine beleuchtet.

[51] vgl. Mayring 2022 S. 84 f.
[52] vgl. Mayring 2022 S. 85

1.4 Einschränkungen

Im Rahmen dieser Arbeit ist es nicht möglich, größer angelegte Untersuchungen mit Studierenden zum konkreten Einsatz einzelner Game Engines durchzuführen, da diese den gegebenen Zeitrahmen übersteigen würden. Entsprechend unterliegen die Auswertungsergebnisse zu großen Teilen der empirischen Forschung des Autors. Diese wurden so unvoreingenommen wie möglich generiert. Dennoch sei an dieser Stelle der Hinweis gegeben, dass der Autor in der Vergangenheit bereits mithilfe der Game Engines *Unity* und *Unreal Engine* an kleinen bis mittelgroßen Projekten mitgewirkt hat und darüber hinaus bereits Kenntnisse in den Programmiersprachen *C#*, *C++*, *Java* und *Python* besitzt.

1.5 Übersicht verwendeter Hilfsmittel

Gemäß der Eigenständigkeitserklärung der Hochschule für angewandte Wissenschaften Neu-Ulm wurden für die Erstellung dieser Arbeit ausschließlich entsprechend erlaubte Hilfsmitteln verwendet. Die vollständige Liste der verwendeten Hilfsmittel lautet:

- Die Software *Word*, die zum Verfassen dieser Arbeit verwendet wurde[53]
- Die Software *Zotero*, die zur Verwaltung der in dieser Arbeit referenzierten Literatur verwendet wurde[54]
- Die Software *Zoom*, die zur Durchführung und Aufnahme der qualitativen Experteninterviews verwendet wurde[55]
- Die KI-Software *Whisper*, die zur ersten Transkription der Interviews verwendet wurde[56]
- Die Software *MAXQDA*, die zur Codierung beziehungsweise Kategorisierung der Interviews verwendet wurde[57]

1.6 Hinweis zur Formatierung

Die Formatierung dieser Arbeit orientiert sich an den **Leitlinien für wissenschaftliche Arbeiten – Fakultät IM** der Hochschule für angewandte Wissenschaften Neu-Ulm[58], verwendet aber die Schriftart **Arial**. Zusätzlich werden Eigennamen und Begriffe, die im Glossar erklärt werden, *kursiv* sowie wichtige Stellen und Begriffe **fett** formatiert.

[53] vgl. Word o. J.
[54] vgl. Zotero o. J.
[55] vgl. Zoom o. J.
[56] vgl. OpenAI 2024
[57] vgl. MAXQDA o. J.
[58] HNU o. J.

2. Definition der Kriterien

Die im Folgenden aufgestellten Kriterien setzen sich aus einer Reihe von Untersuchungspunkten zusammen, die zur Evaluation anderer Lehrmittel oder zur Evaluation von Game Engines bereits berücksichtig werden. Hierzu zählt die spezifische Evaluation von Lehr- und Lernmaterialien für den Einsatz im Schulkontext sowie die Ergebnisse der qualitativen Experteninterviews. Dabei wird im Folgenden zunächst der Inhalt des jeweiligen Kriteriums dargelegt und die zugehörige Grundlage basierend auf den zuvor genannten Quellen aufgezeigt. Anschließend wird eine entsprechende Evaluationsvorgehensweise für das Kriterium beschrieben.

2.1 Qualifikationskriterien

Die folgenden Kriterien entscheiden darüber, ob eine Game Engine für einen Einsatz in der Lehre überhaupt in Frage kommt und entsprechend in Kapitel 4 bezüglich dessen evaluiert wird. Aus Platzgründen sind bei der eigentlichen Evaluation der Game Engine diese Qualifikationskriterien lediglich in Kurzform innerhalb einer Übersichtstabelle aufgelistet.

2.1.1 Definitionstreue

Als Game Engine wird im Folgenden jede Form von eigenständiger Software bezeichnet, die die Entwicklung von digitalen Spielen nicht nur ermöglicht, sondern diese in bestimmten Aspekten, zum Beispiel durch das Vorhandensein einer Physik-Berechnung, erleichtert[59] und darüber hinaus diese Entwicklung nicht auf eine bestimmte Entwicklungsumgebung, ein bestimmtes Setting oder Genre oder eine Darstellungsdimension beschränkt, beziehungsweise von einer anderen Software abhängig ist, auf dieser beruht oder damit in Verbindung steht. Gleichzeitig muss bereits eine vollständige Release-Version der Game Engine existieren, sie darf sich nicht noch in der Entwicklung befinden. Zusätzlich muss eine eigenständige und offiziell unterstützte visuelle Oberfläche (auch häufig als Editor beziehungsweise Editorfenster bezeichnet) vorhanden sein (ein Plugin oder ähnliches reicht nicht aus), die die Game Engine von einem einfachen Text-Editor, einem Ordner- beziehungsweise Dateienbrowser, einer integrierten Entwicklungsumgebung, einer Programmbibliothek oder ähnlichem, visuell abheben. Darüber hinaus muss die Game Engine entweder eine bereits bestehende Programmiersprache oder eine eigens für den Einsatz in Verbindung mit der Game Engine erstellte Programmiersprache für die Programmierung bereitstellen. Dies ist dadurch begründet, dass in der Lehre der Spieleerstellung auch häufig die zugehörige Programmierung gelehrt wird. Game Engines, die also komplett ohne Programmierung eine Spieleerstellung ermöglichen, sind zwar für

[59] vgl. Andrade 2015

einen leichten Einstieg in die Thematik sinnvoll nutzbar, für den Einsatz in der Lehre aber weniger gut geeignet.[60,61]

Durch das Kriterium der Definitionstreue werden unter anderem Sandbox-Plattformen wie *Roblox* oder *Fortnite* sowie Level-Editoren beziehungsweise Level-Editor-Videospiele wie *Super Mario Maker* oder *Trackmania* von einer weiteren Untersuchung ausgeschlossen. Darüber hinaus werden rein für die Erstellung von 2D-Spielen ausgelegte Game Engines und Game Engines ohne Programmiersprachen-Unterstützung sowie Game Engines, die sich in einem Entwicklungsstadium vor einem **Stable Release** (dt.: Stabile Veröffentlichung)[62] befinden ausgeschlossen.

Zusätzlich werden bezüglich dieses Kriteriums im Zuge der Evaluation die grundsätzlichen Möglichkeiten zur Spieleerstellung mit der jeweils untersuchten Game Engine aufgezeigt.

2.1.2 Aktualität

Die Game Engine sowie die damit erstellten Spiele sollten eine gewisse Aktualität aufweisen.[63] Hierdurch kann auf Seiten der Game Engine erkennbar werden, dass diese stets an aktuelle technische Herausforderungen und Neuheiten angepasst oder optimiert ist. Bezüglich der erstellten Spiele lässt sich wiederum die Relevanz der Game Engine im Gesamtkontext aller Möglichkeiten zur Spieleerstellung sowie die technische Leistungsfähigkeit der Game Engine ablesen.[64] Auch das Landesmedienzentrum Baden-Württemberg [65] und Franzke [66] nennen die Aktualität von Medien beziehungsweise Schulbüchern als Kriterium zur Evaluation der Eignung für den Unterrichtseinsatz, jedoch wird hierbei nicht klar definiert, was genau als aktuell kategorisiert werden kann. Das Landesinstitut für Lehrerbildung und Schulentwicklung geht dabei einen Schritt weiter und definiert Aktualität eines Lehrmittels in Form der Anregung „[…] zum Einbezug aktueller Interessen der Schülerinnen und Schüler […].“.[67]

Davon ausgehend gilt im Rahmen dieser Arbeit eine Game Engine als aktuell, wenn diese seit dem 01.01.2023 ein Update erhalten hat **und** seit diesem Datum ein Spiel über eine Vertriebsplattform (beispielsweise *Steam*, *Epic Games Store* oder *itch.io*) veröffentlicht

[60] vgl. I0 Z. 72 – 76
[61] vgl. I5 Z. 110 f.
[62] vgl. Pierce 2022
[63] vgl. I7 Z. 221 ff.
[64] vgl. I7 Z. 243 – 249
[65] vgl. Landesmedienzentrum Baden-Württemberg o. J.
[66] vgl. Franzke o. J.
[67] Yumpu.com o. J.

wurde, das mit der jeweiligen Game Engine entwickelt wurde – alle Game Engines, für die das nicht zutrifft, werden entsprechend ausgeschlossen.

Durch dieses Kriterium wird auch das zugehörige Spieleportfolio [68, 69] der jeweils untersuchten Game Engine kurz beleuchtet, indem bei der Evaluation des Kriteriums beispielhaft ein Spiel aufgelistet wird, das mit der jeweiligen Game Engine erstellt worden ist. Darüber hinaus wird die aktuellste Version der Game Engine selbst beleuchtet.

2.1.3 Preisgestaltung

Eine in der Lehre sinnvoll einsetzbare Game Engine sollte eine möglichst kleine finanzielle Zugänglichkeitsschwelle besitzen. [70, 71] So stellt zum Beispiel auch die Interkantonale Lehrmittelzentrale der Schweiz im Lehrmittelbewertungsinstrument *levanto* das Kriterium Wirtschaftlichkeit unter dem Statement „Das Lehrmittel weist ein gutes Kosten-Nutzenverhältnis auf, inklusive attraktivem Lizenzmodell." auf.[72] Auch Pattrasitidecha[73] oder Christopoulou und Xinogalos [74] nennen die Preisgestaltung als wichtiges Evaluationskriterium für Game Engines. Letztere zählen das Kriterium der Preisgestaltung auch unter dem Kriterienkomplex der **Accessibility** (dt.: Zugänglichkeit) und verweisen damit darauf, dass Lehrmaterialien ohne unverhältnismäßige Investitionen zugänglich sein sollten.

Da hier bei unterschiedlichen Game Engines eine gewisse Fluktuation stattfinden kann, wird bezüglich dieses Kriteriums in der Evaluation die Preisgestaltung der jeweils aktuellen Version der Game Engine aufgezeigt. Ausgeschlossen werden durch dieses Kriterium nur die Game Engines, die von vorneherein einen Festpreis zur Nutzung der gesamten Game Engine oder einzelner wichtiger Funktionen aufstellen.

2.1.4 Entwicklungsplattformen, Zielplattformen und Systemanforderungen

Als Entwicklungsplattform wird im Folgenden eine Plattform bezeichnet, **auf der** Spiele mithilfe der jeweiligen Game Engine erstellt werden können. Als Zielplattform wird im Folgenden eine Plattform bezeichnet, **für die** Spiele mithilfe der jeweiligen Game Engine erstellt werden können.

Die grundsätzlich für die Lehre geeignetste Entwicklungsplattform ist ein Computer mit *Windows*- oder *MacOS*-Betriebssystem, da diese auch außerhalb der

[68] vgl. I0 Z. 70 f.
[69] vgl. I7 Z. 243 – 249
[70] vgl. I1 Z. 113 ff.
[71] vgl. I3 Z. 70 ff.
[72] Interkantonale Lehrmittelzentrale o. J.
[73] vgl. Pattrasitidecha 2014
[74] vgl. Christopoulou/Xinogalos 2017

Spieleprogrammierung bei mehr als 90% der Bevölkerung in Deutschland vorhanden sind.[75] Entsprechend werden durch dieses Kriterium auch **Game Creation Systems** (dt.: Spielerstellungssysteme) wie beispielsweise *Dreams*, die die Spieleerstellung auf Konsolen ermöglichen, ausgeschlossen. Darüber hinaus werden durch dieses Kriterium auch alle Game Engines ausgeschlossen, die nicht für *Windows*- und *MacOS*-Systeme zur Verfügung stehen oder für diese Systeme nicht unterstützt werden.

Bezüglich den Zielplattformen muss eine Game Engine, die sinnvoll in der Lehre eingesetzt werden kann, mindestens die entsprechende Entwicklungsplattform bieten. Darüber hinaus werden durch dieses Kriterium keine Einschränkungen gesetzt und stattdessen im Folgenden ein Überblick bei der jeweils untersuchten Game Engine geboten.

Eine für die Lehre sinnvoll einsetzbare Game Engine sollte außerdem eine möglichst geringe systembezogene Zugänglichkeitsschwelle besitzen, damit Studierende ohne große zusätzliche finanzielle Investitionen einen sinnvollen Mehrwert daraus ziehen können.

Da sich Systemanforderungen von einzelnen Game Engines stets ändern können und technische Upgrades bei einzelnen Computersystemen jederzeit möglich sind, werden bezüglich dieses Kriteriums bei der folgenden Untersuchung der Game Engines jeweils nur die Systemanforderungen der aktuellen Version aufgelistet.

2.2 Qualifizierte Game Engines

Die Datenbank-Webseite EnginesDatabase.com [76] listet 188 Game Engines (Stand 20.02.2024) auf, von denen acht die hier beschriebenen Qualifikationskriterien erfüllen und entsprechend evaluiert werden. Diese werden in zwei Kategorien aufgeteilt betrachtet. Konkret geht es dabei um diese Game Engines:

- Teilweise kostenpflichtig (4): *Flax*, *GameMaker*, *Unity*, *Unreal Engine*
- Kostenfrei (4): *Cocos Creator*, *Defold*, *Godot*, *jMonkeyEngine*

Anhang 1 zeigt eine zusammenfassende Tabelle aller anderen, nicht qualifizierten, Game Engines sowie eine kurze Begründung, warum sie nach den hier aufgestellten Qualifikationskriterien nicht für einen Einsatz in der Lehre und eine weitere Evaluation geeignet sind.

[75] vgl. Statista o. J.
[76] vgl. Alves o. J.

2.3 Custom Game Engines

Darüber hinaus werden im Rahmen dieser Arbeit auch Game Engines evaluiert, die eigens für einen bestimmten Lehrkontext von Dozierenden und Studierenden angefertigt worden sind (im weiteren als Custom Game Engines beziehungsweise kurz als Custom Engines bezeichnet). Diese erfüllen zwar nicht zwingend alle der zuvor definierten Qualifikationskriterien, werden aber bereits aktiv in der Lehre der Spieleerstellung eingesetzt und sind für diesen Einsatz konzipiert worden. Entsprechend sollten hierdurch besonders gut einzelne Lerninhalte vermittelt werden können.

Konkret werden hierbei die folgenden zwei Custom Engines evaluiert:

- *FUDGE* – Furtwangen University Didactic Game Editor: Entwickelt von Jirka Dell'Oro-Friedl und Studierenden der Hochschule Furtwangen
- *Vektoria*: Entwickelt von Tobias Breiner und Studierenden der Hochschule Kempten

2.4 Evaluationskriterien

Die folgenden Kriterien dienen der konkreten Evaluation einzelner Game Engines und bieten dementsprechend Aussagen darüber, wie gut die jeweils untersuchte Game Engine in der Lehre eingesetzt werden kann. Hierbei wird zwischen den Kriterien Erlernbarkeit, Anwendbarkeit und Übertragbarkeit unterschieden und nach den folgenden Definitionen evaluiert. Abschließend zeigt eine kurze Zusammenfassung Vor- und Nachteile der jeweils untersuchten Game Engine bezüglich dieser Kriterien auf und summiert diese sowie einzelne Erkenntnisse beziehungsweise Meinungen aus den qualitativen Experteninterviews zu einer konkreten Nutzungsempfehlung für den Einsatz in der Lehre.

2.4.1 Erlernbarkeit

Unter dem Kriterium der Erlernbarkeit wird evaluiert, wie einfach und schnell einzelne Funktionen einer Game Engine beziehungsweise der Umgang mit dieser erlernt werden kann.

Hierfür werden einerseits Lernmaterialien betrachtet, die die Herausgeber der Game Engine selbst zur Verfügung stellen (im weiteren als **offizielle Lernmaterialien** bezeichnet). Dabei wird vor der tatsächlichen Reproduzierbarkeit dieser Tutorials und Guides zunächst auch die Verfügbarkeit für unterschiedliche Versionen der Game Engine, die Verfügbarkeit in unterschiedlichen Sprachen sowie die Aktualität dieser berücksichtigt. Darüber hinaus werden in diesem Zusammenhang einige Ansprüche an Lehrmittel nach Adamina bezüglich ihres Vorhandenseins bei den verglichenen Game Engines überprüft. Hierzu gehört die Auslegung von Lernmaterialien und -aufgaben auf „[…] das situierte und […] eigenständige

Lernen […]“ [77] , sowie die Möglichkeit der „[…] Selbst- und Fremdbeurteilung von Kompetenzentwicklungen [beziehungsweise] Rückmeldungen zum Lernprozess und zu Lernergebnissen unter Bezugnahme auf […] Kompetenzerwartungen […].“.[78] Konkret wird hiervon abgeleitet also beleuchtet, inwiefern eine Game Engine von Studierenden eigenständig mit den verfügbaren **offiziellen Lernmaterialien** erlernt werden kann, oder ob die Erlernung ausschließlich im Kontext von durch Lehrpersonal betreuten Vorlesungs- oder Seminar-Situationen möglich ist. Bezüglich der Selbst- und Fremdbeurteilung und Rückmeldungen zum Lernprozess wird betrachtet, ob und inwieweit diese Aspekte innerhalb der **offiziellen Lernmateralien** – beispielsweise durch konkrete Abfragen der erlernten Inhalte – abgedeckt werden. Auch innerhalb der qualitativen Experteninterviews war die Verfügbarkeit von vielen und qualitativen Lernmaterialien ein häufig angesprochenes Kriterium. [79 , 80] Um bezüglich dieses Aspekts der Erlernbarkeit eine möglichst sinnvolle Vergleichbarkeit zu schaffen werden für jede Game Engine – soweit verfügbar – die Lernmaterialien betrachtet, die sich explizit mit den ersten Schritten innerhalb dieser Engine befassen.

Im Anschluss an die Analyse der **offiziellen Lernmaterialien** wird ein Blick auf weitere Lernmaterialien gelegt, die durch Drittanbietende zur Verfügung gestellt werden. Diese werden dabei als **inoffizielle Lernmaterialien** bezeichnet. In diesem Zug wird auch die Community der jeweiligen Game Engine am Beispiel der Social-News-Aggregations-Webseite *Reddit* kurz beleuchtet sowie *YouTube* als Plattform zum Teilen unterschiedlicher videogestützter Erklärungen betrachtet. Durch diesen Blick auf Menge und Relevanz weiterer Hilfestellungen können erneut die Möglichkeiten zum Selbststudium mit der jeweiligen Game Engine beleuchtet werden.

Bei Christopoulou und Xinogalos wird unter dem Kriterienkomplex der Zugänglichkeit auch die **Usability** (dt.: Benutzerfreundlichkeit) einer Game Engine aufgefasst.[81] Entsprechend wird diesbezüglich eine **Heuristische Evaluation** der jeweiligen Benutzeroberfläche vorgenommen. Diese wird Nielsen folgend durchgeführt, nach dessen Vorgehensweise für eine angeschaute Benutzeroberfläche eine Meinung zu positiven und negativen Aspekten dieser Oberfläche entwickelt wird[82] – in der folgenden Evaluation wird dies in Form eines empirischen Ersteindrucks getan. Neben dieser allgemeinen ersten Meinung zu der jeweils untersuchten Benutzeroberfläche werden darüber hinaus einzelne konkrete Aspekte

[77] Adamina 2014 S. 364
[78] Adamina 2014 S. 366
[79] vgl. I4 Z. 69 – 74
[80] vgl. I8 Z. 50 ff.
[81] vgl. Christopoulou/Xinogalos 2017
[82] vgl. Nielsen 1993 S. 155

überprüft. Diese basieren auf den von Nielsen aufgestellten zehn Heuristiken zur Bewertung der Benutzerfreundlichkeit von User Interface Design [83] und wurden für die spezielle Untersuchung auf typische Benutzeroberflächen von Game Engines angepasst, wie in der folgenden Tabelle zu sehen ist:

Heuristik nach Jakob Nielsen	**Davon abgeleitete Bewertungsfragen übertragen auf eine Evaluation von Game Engines**	**Nr. der Frage bzw. Fragensammlung**
„Visibility of System Status“ (dt.: Sichtbarkeit des Systemstatus)	Inwieweit können Nutzende den Überblick über einzelne Änderungen innerhalb der Game Engine behalten? Wie konkret äußern sich die Erstellung und Änderungen von einzelnen Spielobjekten in der Sichtbarkeit?	1
„User Control and Freedom“ sowie **„Flexibility and Efficiency of Use“** (dt.: Benutzerkontrolle und Freiheit sowie Flexibilität und Effizienz der Nutzug)	Inwieweit können Nutzende ihre eigene Kontrolle über das System ausüben; Gibt es Möglichkeiten zum Rückgängigmachen von falschen Inputs? Gibt es Möglichkeiten zum Definieren eigener Shortcuts? Gibt es Sicherungen durch die Game Engine, die den Datenverlust verhindern?	2
„Help Users Recognize, Diagnose, and Recover from Errors“ (dt.: Unterstützung der Benutzer bei der Erkennung, Diagnose und Behebung von Fehlern)	Inwieweit existieren Fehlermeldungen zu geschriebenem Programmcode, geben Aussagen über die konkreten Fehler und stellen konkrete Vorschläge zur Fehlerlösung zur Verfügung?	3
„Help and Dokumentation“ (dt.: Hilfe und Dokumentation)	Inwieweit gibt es eine Dokumentation für die einzelnen Menüpunkte, Teilfenster und weitere Elemente der Benutzeroberfläche?	4

Tabelle 4 Übersicht Bewertungsfragen Benutzeroberfläche abgeleitet von Heuristiken nach Jakob Nielsen

[83] siehe Nielsen Norman Group o. J.

Die ausgefüllten Kriterientabellen zur Heuristischen Analyse der Benutzeroberfläche der einzelnen Game Engines finden sich gesammelt in **Anhang 3**.

2.4.2 Anwendbarkeit

Unter dem Kriterium der Anwendbarkeit wird evaluiert, wie einfach und schnell eine tatsächliche Spieleerstellung mithilfe der Game Engine stattfinden kann.

Diesbezüglich wird der genaue Umgang mit einer Game Engine beleuchtet: Christopoulou und Xinogalos sowie Petridis et al. untersuchen unter dem Kriterienkomplex **Composability** (dt.: Zusammensetzbarkeit), inwiefern Inhalte, die mit einer Game Engine erstellt wurden, wiederverwendet werden können und inwieweit die Game Engine das Importieren und Exportieren von häufig für die Spieleerstellung verwendeten Dateiformaten unterstützt.[84,85] In der Evaluation dieses Aspekts wird entsprechend ein Blick auf die von der jeweils untersuchten Game Engine unterstützten Dateiformate gelegt und eventuelle Besonderheiten in der Arbeit mit diesen aufgezeigt. Die Möglichkeiten zum Import einzelner Elemente werden dabei konkret an einem einfachen 3D-Objekt (im **fbx**-Format) und einem 2D-Bild (im **png**-Format) überprüft – eine Darlegung dieser beiden Dateien ist in **Anhang 2** zu finden. Darüber hinaus wird überprüft, inwieweit einzelne Inhalte, die in einer Game Engine erzeugt wurden, aus dieser exportiert werden können und dadurch in anderen Projekten der gleichen Game Engine oder sogar in anderen Game Engines wiederverwendet werden können.

Anschließend wird die von Pattrasitidecha unter dem Kriterienkomplex **Usability** (dt.: Benutzerfreundlichkeit) betrachtete **Efficiency of Use** (dt.: Effizienz der Nutzung) der jeweils untersuchten Game Engine beleuchtet. Hierbei werden nicht die zu erwartenden Funktionen wie eine Programmierumgebung beziehungsweise -schnittstelle, ein Physiksystem sowie verschiedene Komponenten, die einzelnen Spielobjekten unterschiedliche Eigenschaften verleihen oder Editorfenster zur Zusammensetzung, Strukturierung und Ausführung von Spielszenen betrachtet, sondern ein Blick auf weitere Funktionen gelegt, die die Engine über diese Mindestanforderungen hinaus anbietet – hierzu können beispielsweise Möglichkeiten zur Erstellung von Licht-Objekten, Animationen, UI-Elementen oder Werkzeuge zur Optimierung der Leistungsfähigkeit des Spiels gehören. Unter diesem Aspekt wird auch ein besonderes Augenmerk auf Möglichkeiten zur Versionsverwaltung mit der jeweiligen Game Engine gelegt, da hierdurch vor allem auch die Kollaborationsmöglichkeiten für Nutzende der Game Engine aufgezeigt werden können. Hier wird entsprechend betrachtet, ob und inwieweit die Game Engine

[84] vgl. Christopoulou/Xinogalos 2017
[85] vgl. Petridis/Dunwell/Panzoli 2012

Nutzenden ein eigenes System zur Versionsverwaltung zur Verfügung stellt, oder ob auf bereits bestehende Verwaltungssysteme, wie beispielsweise *Git*[86], zurückgegriffen werden muss und wie die Schnittstelle zwischen der jeweiligen Game Engine und einzelnen Versionsverwaltungssystemen verwendet werden kann.

Abschließend werden unter dem Kriterium der Anwendbarkeit noch **Besonderheiten** der einzelnen Game Engines aufgezeigt, die ihre tatsächliche Anwendbarkeit zur Spieleerstellung beeinflussen und sie dadurch im Vergleich zu anderen Game Engines hervorstechen lassen.

2.4.3 Übertragbarkeit

Unter dem Kriterium der Übertragbarkeit wird evaluiert, wie einfach und schnell der erlernte Umgang mit einer Game Engine auf die Nutzung anderer Game Engines beziehungsweise die Nutzung der erlernten Game Engine auf das berufliche Umfeld in der Games Branche übertragen werden kann.

Gestützt wird dieser Anspruch auch innerhalb der qualitativen Experteninterviews; viele Lehrkontexte haben hierbei klar das Ziel, Studierende auf das zugehörige Berufsumfeld vorzubereiten, wodurch die Übertragbarkeit einzelner erlernte Fertigkeiten entsprechend hohe Relevanz erhält.[87,88] Ausgehend hiervon werden konkret die folgenden übertragbaren Konzepte der Spieleprogrammierung beziehungsweise Spieleerstellung mithilfe von Game Engines betrachtet:

Programmierparadigmen – Programmierparadigmen (auch Programmierstile) umfassen die Art und Herangehensweise einer Lösungsumsetzung mithilfe einer Programmiersprache. Hierbei wird grundsätzlich zwischen imperativen (wie soll das Programm arbeiten), deklarativen (was ist das Ergebnis, das das Programm liefern soll) oder modernen (beispielsweise objektorientierte Programmierung, die versucht das zu lösende Problem zu abstrahieren) Programmierparadigmen unterschieden.[89] Viele Programmiersprachen und damit auch Game Engines ermöglichen oder empfehlen die Verwendung einer oder weniger dieser Vorgehensweisen. Bei der nachfolgenden Evaluation werden unter diesem Aspekt die mit der jeweiligen Game Engine erlernbaren und sinnvoll einsetzbaren Programmierparadigmen betrachtet und eine vergleichende Übertragbarkeit evaluiert.

Schlüsselkonzepte – **Scenes**, **Level** oder **Maps** sind nur einige der vielen unterschiedlichen Bezeichnung, die in Game Engines für das Konzept einzelner zusammenhängender

[86] vgl. Git-scm o. J.a
[87] vgl. I3 Z. 22 f.
[88] vgl. I9 Z. 52 f.
[89] vgl. Mathes/Seufert 2022

Spielabschnitte genutzt werden. Unter dem Aspekt der Schlüsselkonzepte wird in der nachfolgenden Evaluation der Aufbau dieser sowie die jeweilige Herangehensweise beleuchtet und betrachtet, inwiefern eine Übertragung der jeweils eingesetzten Konzepte auf andere Game Engines möglich ist. Der Aufbau von Spielabschnitten beziehungsweise von zusammenhängenden Spielobjekten lässt sich dabei grob in zwei Strukturen einteilen. Einerseits gibt es die **Freie-Szenen-Struktur**, bei der Spielabschnitte frei aus einzelnen Objekten zusammengesetzt werden können. Andererseits gibt es die **Wurzel-Knoten-Struktur**, bei der Spielabschnitte ausgehend von einem Wurzelknoten zusammengesetzt werden, an den einzelne Objekte in Form von weiteren Knoten angehängt werden können. Auf der technischen Seite sind diese beiden Systeme je nach betrachteten Game Engines sehr ähnlich und eine Unterscheidung ist hier eher anhand der visuellen Präsentation dieser Systeme innerhalb der Game Engine möglich. Unter dem Aspekt der möglicherweise übertragbaren Schlüsselkonzepte sollen auch einzelne Spielelemente, die häufig **GameObjects**, **Nodes** oder **Actors** genannt werden und innerhalb einer Game Engine zur Darstellung einzelner Spielelemente und der Zusammenstellung von Spielabschnitten verwendet werden können, vergleichend gegeneinander aufgestellt und evaluiert werden. Darüber hinaus wird auch die Wiederverwendung einzelner Spielelemente, die dann häufig als **Prefabs**, **Blueprints** oder **Templates** bezeichnet werden, unter diesem Punkt beleuchtet.

2.4.4 Zusammenfassung

Nach der jeweiligen Evaluation einer Game Engine werden bezüglich dieser noch einzelne Meinungen von Lehrenden aus den durchgeführten qualitativen Experteninterviews ergänzt. Diese werden dann mit den Erkenntnissen aus der durchgeführten Evaluation verknüpft und in einer abschließenden Zusammenfassung werden hiervon Einsatzmöglichkeiten der jeweiligen Game Engine in Lehrkontexten abgeleitet.

3. Aktuelle Situation

Bevor die eigentliche Evaluation der qualifizierten Game Engines durchgeführt wird, wird an dieser Stelle zunächst ein Überblick über die aktuelle Situation in Deutschland in Bezug auf die Lehre der Spieleerstellung gegeben. Die hier zusammengefassten Daten beruhen auf den qualitativen Experteninterviews, die mithilfe der Software *MAXQDA* unter Verwendung der qualitativen Inhaltsanalyse und induktiven Kategorienbildung nach Mayring codiert wurden.

Ein Thema, das innerhalb jedes einzelnen Interviews aufgekommen ist, obwohl es nicht im Fragebogen verankert war und entsprechend vom jeweiligen Interviewten selbst in das Gespräch miteingebracht wurde oder durch den natürlichen Gesprächsverlauf angesprochen wurde, ist die Lizenzpolitik von **Unity Technologies**, dem Unternehmen hinter der *Unity* Game Engine. Dieses kündigte im September 2023 an, ab dem 1. Januar 2024 ein neues Lizenzmodell einführen zu wollen, nach dem Spieleentwickelnde einen bestimmten Geldbetrag für jeden Download eines mit *Unity* erstellten Spiels bezahlen müssten.[90] Bevor hierbei entsprechende Preise anfielen – die sich je nach verwendeter *Unity* Lizenz unterschieden – musste eine bestimmte Umsatzschwelle und Installationsmenge überschritten werden. Auch diese Schwelle unterschied sich je nach verwendeter Version der Game Engine, was die genaue Höhe der potenziellen Kosten, die für Entwickelnde entstanden wären, sehr undurchsichtig machte. Darüber hinaus kamen verschiedene Fragen auf, beispielsweise, wie genau die Anzahl der Downloads überwacht werden sollen und inwiefern dabei Daten von Nutzenden weiterhin geschützt wären. Auch die Tatsache, dass ein Spiel nach dem einmaligen Kauf potenziell auf mehreren Geräten installiert werden könnte (dies erlaubt beispielsweise die Spieleplattform *Steam*) und dadurch mehrfach zum Erreichen beziehungsweise Überschreiten der benötigten Installationsmenge beitragen könnte, ließen die angekündigten Änderungen weiter unüberlegt und gerade für kleine Entwicklungsstudios sehr einschränkend wirken. Entsprechend äußerten sich viele Entwickelnde enttäuscht über die angekündigten Änderungen und kündigten an, so bald wie möglich von *Unity* auf eine andere Game Engine umzusteigen.[91] **Unity Technologies** ruderte entsprechend zurück, löschte den Beitrag zu den ursprünglich angekündigten Änderungen [92] und veröffentlichte neben einer Entschuldigung [93] eine überarbeitete Version der Lizenzänderungen – die die Umsatzschwelle deutlich höher ansetzte und für einige Lizenzen die vorherigen

[90] vgl. Parrish 2023
[91] vgl. Gamerant o. J.
[92] vgl. Unity Technologies 2023a
[93] vgl. Unity Technologies 2023b

Bedingungen in Kraft ließ.[94] Gleichzeitig wurde ein **Runtime Fee Estimator** (dt.: Laufzeitgebühren-Schätzer) veröffentlicht, mit dem potenziell entstehende Kosten schon im Vorhinein schätzungsweise berechnet werden konnten.[95] Dennoch hinterließ dieser Prozess – vor allem durch die schlechte Kommunikation und die undurchsichtige Präsentation – bleibende Schäden am Image der Game Engine und dem Unternehmen dahinter sowie Ungewissheit, ob solche oder schlimmere Lizenzänderungen nicht in Zukunft wieder angekündigt und dann möglicherweise noch härter umgesetzt werden könnten.

Hieraus abgeleitet ist es nicht verwunderlich, dass zwei der vier Lehrkontexte, bei denen *Unity* in der Vergangenheit in den Lehrplan integriert war, durch diese Ungewissheit über die Produktpolitik auf andere Game Engines umgestiegen sind[96,97] – die anderen beiden sind bereits zuvor auf eine andere Game Engine gewechselt. Abgesehen hiervon zählt *Unity* aktuell dennoch zu den meisteingesetzten Game Engines in den befragten Lehrkontexten, wie an der folgenden Grafik zu erkennen ist:

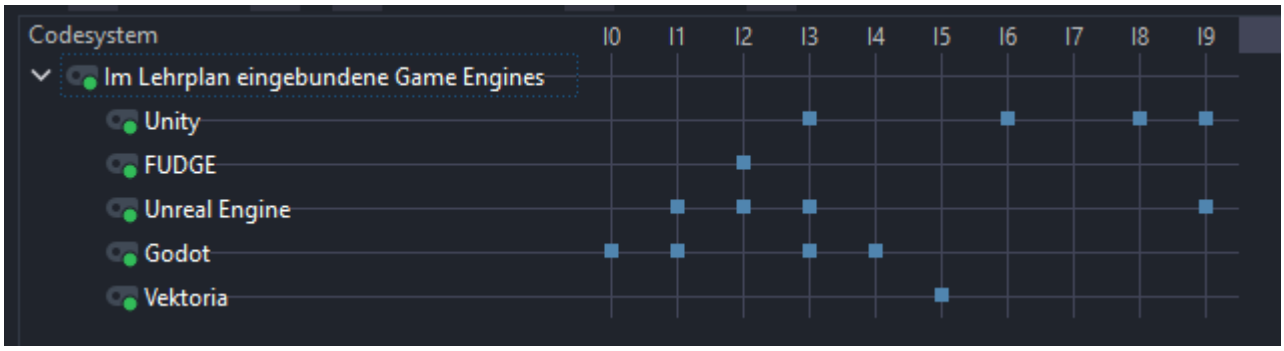

Abb. 1 Übersicht aller Game Engines, die in die befragten Lehrpläne eingebunden sind

Neben *Unity* sind auch *Godot* und *Unreal Engine* jeweils in vier der befragten zehn Lehrkontexten fest in den Lehrplan eingebunden – thematisieren den Umgang mit der Game Engine also in Seminaren, Übungen oder ähnlichem. Auffällig ist beim Blick auf diese Kategorie aber, dass neben *Unity*, *Godot* und *Unreal Engine* tatsächlich nur noch *FUDGE* und *Vektoria* in den Lehrplan eingebunden sind – die Diversität an in der Lehre eingesetzten Game Engines beschränkt sich also auf die „großen Drei“ sowie die Custom Engines. Erst beim Blick auf Game Engines, die in Projekten eingesetzt werden (wobei hier sowohl studentische Projekte mit freier Game Engine Wahl als auch Forschungsprojekte mit festen Vorgaben zu verwendeten Game Engines gemeint sind), zeigt sich eine etwas größere Vielfalt. Dennoch ist *Unity* die einzige Game Engine, die zusammengefasst in allen

[94] vgl. Unity Technologies o. J.b
[95] vgl. Unity Technologies o. J.f
[96] vgl. I0 Z. 42 f.
[97] vgl. I4 Z. 86 ff.

Lehrkontexten Verwendung findet. Bezüglich der in dieser Arbeit evaluierten Game Engines ist hierbei *GameMaker* die einzige Game Engine, die innerhalb der Interviews noch angesprochen wurde (siehe **Abb. 2**). Diese wurde zumindest in einem Fall auch bezüglich einer Aufnahme in den Lehrplan evaluiert[98] und war in einem weiteren Fall immerhin in der Vergangenheit Teil des Lehrplans.[99]

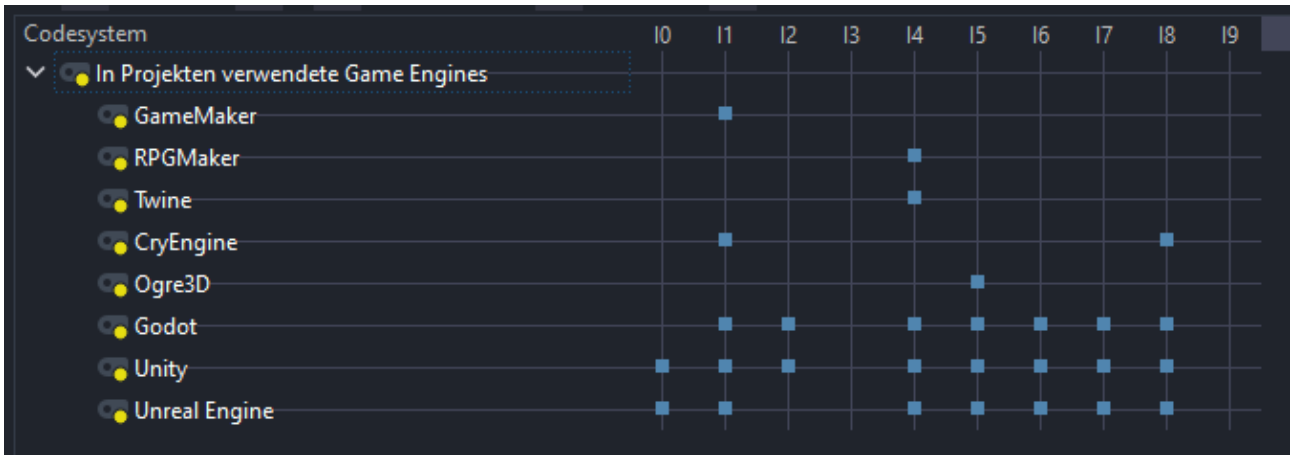

Abb. 2 Übersicht aller Game Engines, die in den befragten Lehrkontexten in Projekten eingesetzt werden

Alle anderen in dieser Arbeit evaluierten Game Engines sind in den untersuchten Lehrkontexten weder aktuell im Einsatz noch in der Vergangenheit im Einsatz gewesen, noch wurden sie bei einer eventuellen Evaluation bezüglich ihrer Einsetzbarkeit berücksichtigt.

Warum hauptsächlich *Unity*, *Godot* und *Unreal* eingesetzt werden, lässt sich beim Blick auf die innerhalb der Lehrkontexte aufgestellten Kriterien für Game Engines ableiten (siehe **Abb. 3**). Die beiden am häufigsten genannten Kriterien für die Aufnahme einer Game Engine in den Lehrplan sind einerseits die geringe Einstiegshürde beziehungsweise eine gewisse Einsteigerfreundlichkeit sowie andererseits die Möglichkeit schnelle Ergebnisse zu produzieren und hierdurch entsprechend schnell motiviert zu werden, wie die folgende Grafik zeigt.

[98] vgl. I0 Z. 60
[99] vgl. I4 Z. 122 ff.

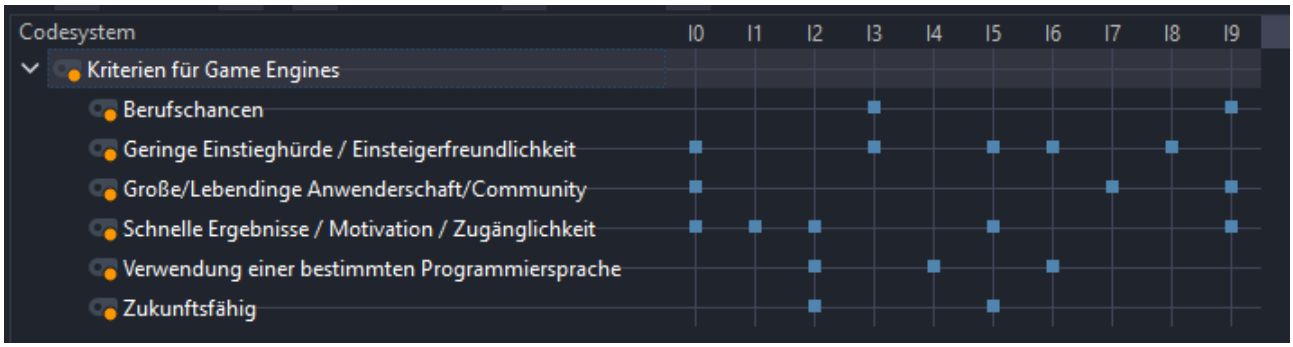

Abb. 3 Übersicht der sechs am häufigsten genannten Kriterien für den Einsatz von bestimmten Game Engines

Diese Umstände sind bei *Unity* und *Unreal* vor allem anhand ihrer sehr langen Existenz und der entsprechend großen Menge an Nutzenden und der zugehörigen großen Menge an verfügbaren Hilfestellungen erklärbar. *Godot* reiht sich hier ein, weil die Game Engine einerseits ebenfalls eine geringe Einstiegshürde bietet[100] und darüber hinaus auch mit schnellen Ergebnissen glänzen kann[101] sowie zusätzlich *Open Source* ist.[102]

Abgesehen von den sechs meistgenannten Kriterien (in mindestens 30% der Interviews vertreten) für den Einsatz einer Game Engine sind weitere Kriterien je nach betrachtetem Lehrkontext sehr unterschiedlich. Insgesamt wurden innerhalb der Interviews 27 Kriterien aufgestellt, die eine Game Engine für die Aufnahme in den Lehrplan qualifizieren könnten. Hieraus lässt sich leicht ableiten, dass eine Game Engine auch möglichst gut zum jeweiligen Lehrkontext passen muss und die hierfür berücksichtigten Schwerpunkte sich mitunter stark voneinander unterschieden. Einig waren sich ein Großteil der Befragten nur in den oben gezeigten Kriterien, alle weiteren Nennungen von Kriterien fanden einzeln verteilt statt und wirkliche Überschneidungen lassen sich hierbei nur schlecht herausarbeiten.

Dies setzt sich auch in den Kriterien fort, durch die eine Game Engine nicht in den Lehrplan aufgenommen wurde oder durch die sie mitunter sogar aus dem Lehrplan gestrichen wurden (siehe **Abb. 4**). Neben den in der Grafik aufgelisteten vier meistgenannten Kriterien (in mindestens 30% der Interviews vertreten) wurden elf weitere Kriterien genannt, die sich ebenfalls zwischen den einzelnen Lehrkontexten stark unterschieden. Dennoch gibt es bezüglich der hier am häufigsten genannten Kriterien ein paar Auffälligkeiten; in der Hälfte der Interviews wurden fehlende Kenntnisse des Lehrpersonals oder fehlendes Lehrpersonal

[100] vgl. I0 Z. 107 f.
[101] vgl. I0 Z. 45 f.
[102] vgl. I3 Z. 26 f.

im Kontext des Einsatzes von Game Engines genannt, wodurch Lehrkontexte entsprechend auf den Einsatz einer einzelnen Game Engine beschränkt sind.[103,104,105,106,107]

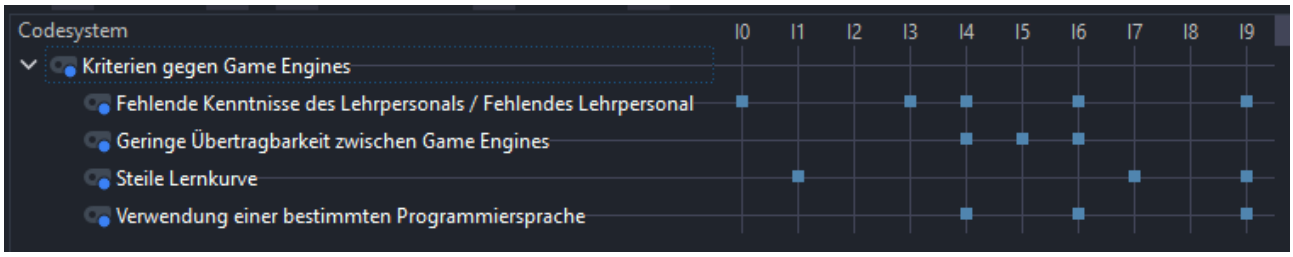

Abb. 4 Übersicht der vier am häufigsten genannten Kriterien gegen den Einsatz von bestimmten Game Engines

Auffällig ist auch, dass unter dem Code **Verwendung einer bestimmten Programmiersprache** in allen drei Fällen jeweils die *Unreal Engine* genannt wurde und die darin verwendete Programmiersprache *C++* als zu schwer erlernbar im Kontext der jeweiligen Studierendenschaft eingestuft wurde.[108,109,110] Hieran anknüpfend lässt sich zwar eine grobe Einteilung der jeweils beleuchteten Lehrkontexte in zwei unterschiedliche Lehransätze vornehmen – die informatiknahen auf der einen Seite, die informatikferneren auf der anderen – generalisierende Aussagen über hierbei jeweils fokussierte Kriterien für oder gegen den Einsatz von Game Engines lassen sich aber nicht treffen. Zusammengefasst zeigen die Interviews, dass Studiengänge, Studiengangsmodule oder sonstige Lehrkontexte, die sich in Deutschland mit der Spieleerstellung beschäftigen, teilweise sehr unterschiedlich sind und entsprechend sehr unterschiedliche Kriterien für die jeweils verwendeten Game Engines aufstellen. 16 der 27 Kriterien für die Aufnahme einer Game Engine in den Lehrplan wurden jeweils nur ein Mal innerhalb der zehn Interviews genannt, was diesen Punkt unterstützt. Trotz dieser großen Unterschiede in der Begründung für den Einsatz einer bestimmten Game Engine finden sich aber immer wieder die gleichen Antworten auf die Frage, welche Game Engine denn im jeweiligen Lehrkontext eingesetzt wird. Nur fünf der zehn in dieser Arbeit evaluierten Game Engines werden aktuell in der Lehre der Spieleerstellung in Deutschland eingesetzt, wodurch hier möglicherweise Erweiterungen der Lehrpläne um weniger bekannte Game Engines vorgenommen werden können, falls diese sich im folgenden Kapitel als sinnvoll einsetzbar herausstellen.

[103] vgl. I0 Z. 103 f.
[104] vgl. I3 Z. 76 f.
[105] vgl. I4 Z. 214 – 220
[106] vgl. I6 Z. 124 f.
[107] vgl. I9 Z. 41 – 48
[108] vgl. I4 Z. 139 – 146
[109] vgl. I6 Z. 132 – 137
[110] vgl. I9 Z. 28 ff.

4. Evaluation

4.1 Teilweise kostenpflichtige Game Engines

Unter dieser Kategorie sind alle Game Engines zusammengefasst, die nur begrenzt kostenlos zum Spielevertrieb einsetzbar sind. Während die Nutzung der Game Engines zur Spieleerstellung kostenlos ist, fallen bei einem Verkauf der Spiele unter bestimmten Umständen Gebühren an. Dies äußert sich beispielsweise durch feste beziehungsweise intervallbezogene Nutzungsgebühren oder umsatzbezogene Lizenzgebühren.

4.1.1 Flax

Kriterium	**Ergebnis**
Definitionstreue	*Flax* ermöglicht die Erstellung von 2D- und 3D-Spielen mithilfe der Programmiersprachen *C#* oder *C++*.[111]
Aktualität	Version 1.7.2 wurde am 20. Dezember 2023 veröffentlicht[112], Auf *Itch.io* wurde zuletzt (18. Dezember 2023) *Obscured by the Night* veröffentlicht.[113]
Preisgestaltung	*Flax* ist kostenfrei bei einem Gewinn von weniger als 250.000$ pro Kalenderquartal, für größere Gewinne werden 4% Lizenzgebühren berechnet.[114]
Entwicklungsplattformen	*Windows*[115], *MacOS*[116], *Linux*[117]
Zielplattformen	*Windows, MacOS, Linux, Android, iOS, Playstation 4, Playstation 5, Xbox One, Xbox Series X/S, Nintendo Switch*[118]
Systemanforderungen	Allgemein: 2 GHz CPU, 4GB RAM, 1GB Festplattenspeicherplatz *Windows*: *Windows* Vista SP1 oder neuer, 512MB und DX10+-kompatible GPU[119] *Linux*: Ubuntu 22 LTS, Vulkan-kompatible GPU[120]

[111] vgl. Flax o. J.h
[112] vgl. Figat 2023
[113] vgl. Itch o. J.
[114] vgl. Flax o. J.l
[115] Flax o. J.v
[116] Flax o. J.g
[117] Flax o. J.f
[118] Flax o. J.h
[119] Flax o. J.p
[120] Flax o. J.f

	MacOS: *MacOS* 11.7 oder neuer, arm64 Prozessor (M1/M2/M3)[121]

Tabelle 5 Evaluation der Qualifikationskriterien – Flax

Erlernbarkeit

Offizielle Lernmaterialien

Flax bietet Neueinsteigenden online ein Manual sowie eine zugehörige *C#* und *C++ API*. In ersterem gibt es unterschiedliche Tutorials zu ersten Schritten im Umgang mit der Game Engine, spezifische Tutorials zu einzelnen Funktionen der Game Engine (beispielsweise der Benutzeroberfläche und einzelnen Fenstern dieser) und drei Migrierungs-Guides zum Umstieg von *Unreal Engine 4*, *Unity* oder *Godot* zu *Flax*. Alle Tutorials stehen nur auf Englisch zur Verfügung und eine Anpassung dieser an eine bestimmte Version der Game Engine ist nicht möglich. Auch bezüglich der Aktualität der Tutorials lassen sich keine Aussagen treffen, da ein entsprechendes Veröffentlichungsdatum oder ähnliches nicht einsehbar ist.

Da die bisher aufgezählten Tutorials von *Flax* recht übersichtlich ausfallen und sie jeweils nur die wichtigsten Informationen für Neueinsteigende zur Installation und Projekterstellung mit der Game Engine zusammenfassen, wurde zur Überprüfung der Reproduzierbarkeit das Tutorial **How to: Change Scene from Script**[122] aus dem **Samples and Tutorials**[123] Teil des Manuals ausgewählt und dieses anhand der *Flax*-Version 1.7 evaluiert. Im Tutorial wird erklärt, wie zwischen einzelnen Spielszenen durch die Verwendung eines Scripts gewechselt werden kann. Das Tutorial beschränkt sich auch nur auf diesen Inhalt und lässt entsprechend das nötige Setup eines Projekts aus. Dies ist zwar nicht weiter schlimm, wenn man sich vorher das Tutorial zum Erstellen eines Projekts angeschaut hat, eine Verlinkung hierzu wäre aber gerade für Neueinsteigende sehr praktisch. Darüber hinaus kommen bereits beim ersten Schritt des Tutorials Fragen auf. Während hier beschrieben wird, dass man im **Source**-Ordner per Rechtsklick und Auswahl der Option **New → Script** ein neues Script erstellen soll, funktioniert das in der Game Engine selbst nicht direkt. Erst wenn man in den **Game**-Ordner, der sich im **Source**-Ordner befindet, navigiert, erhält man die Option ein neues Script anzulegen. Gleichzeitig steht man hier noch vor der Auswahl zwischen einem *C#*-Script und einem *C++*-Script. Letztendlich ist es egal, für welche dieser beiden Optionen man sich entscheidet, da im nächsten Schritt des Tutorials sowohl *C#*-, als auch *C++*-Code gegeben ist. Im Zuge der Evaluation wurde zunächst die *C#*-Version gewählt.

[121] Flax o. J.g
[122] vgl. Flax o. J.i
[123] vgl. Flax o. J.s

Nachdem das Script vervollständigt wurde, soll es im nächsten Schritt zu einem **Actor** in der **Scene** hinzugefügt werden. Dass zunächst einmal überhaupt erst eine solche **Scene** erstellt werden und dieser ein **Actor** hinzugefügt werden muss, wird weder erwähnt noch näher erklärt. Mit ein wenig Ausprobieren ist aber auch dieser Schritt erfolgreich abgeschlossen und der Rest des Tutorials lässt sich ohne weitere Probleme vervollständigen und reproduzieren. Bei der alternativen Wahl des *C++*-Scripts stolpert man aber über das Problem, dass das Script nicht wie beschrieben einem **Actor** hinzugefügt werden kann, wodurch das Tutorial über diesen Weg nicht abgeschlossen werden kann.

Zusammengefasst bietet *Flax* also zunächst relativ wenige konkrete Tutorials zu Themen an, die über die ersten wenigen Minuten im Umgang mit der Game Engine hinausgehen. Entsprechend ist eigenständiges Lernen mit der Game Engine sehr schwer und situiertes Lernen im Kontext eines Seminars oder ähnlichem mit entsprechend tiefgreifender Betreuung nötig. Auch eine eventuelle Fremdbeurteilung des Lernenden entfällt im Kontext der zur Verfügung gestellten offiziellen Lernmaterialien. Die Selbstbeurteilung eines Lernenden ist lediglich durch die Betrachtung und eigenständige Evaluation der erzielten Ergebnisse möglich, wobei diese im evaluierten Tutorial durch die unerklärlicherweise nicht funktionierende Implementierung des *C++*-Scripts quasi unmöglich ist. Abgesehen hiervon existieren auch keine einzelnen Guides zum Erstellen kleiner Spiele, die bei anderen Game Engines häufig vorhanden sind. Auch die gegebenen Migrierungs-Guides fallen relativ kurz aus, geben aber immerhin einen guten Überblick, um einen Umstieg zu gewährleisten.

Inoffizielle Lernmaterialien

Auch auf *YouTube* gibt es wenige Tutorials, die sich mit *Flax* beschäftigen. Gleichzeitig gehen diese teilweise zwischen Tutorials zur gleichnamigen neural network library[124] unter. Lediglich auf dem *YouTube*-Kanal **Abra** [125] finden sich unter den 69 Videos (Stand 04.04.2024) einige Tutorials zu *Flax*. Der Bereich für *Flax* auf der Plattform *Reddit* fällt mit seinen 176 Mitgliedern ebenfalls sehr klein aus. Zusätzlich wurde hier zuletzt vor drei Jahren ein Post erstellt. Auch ein Forum zu *Flax* existiert leider nicht. Insgesamt stehen Lernenden also sehr wenige gut verwendbare Ressourcen zur Erlernung der Game Engine zur Verfügung.

[124] vgl. Google 2024
[125] vgl. Abra o. J.

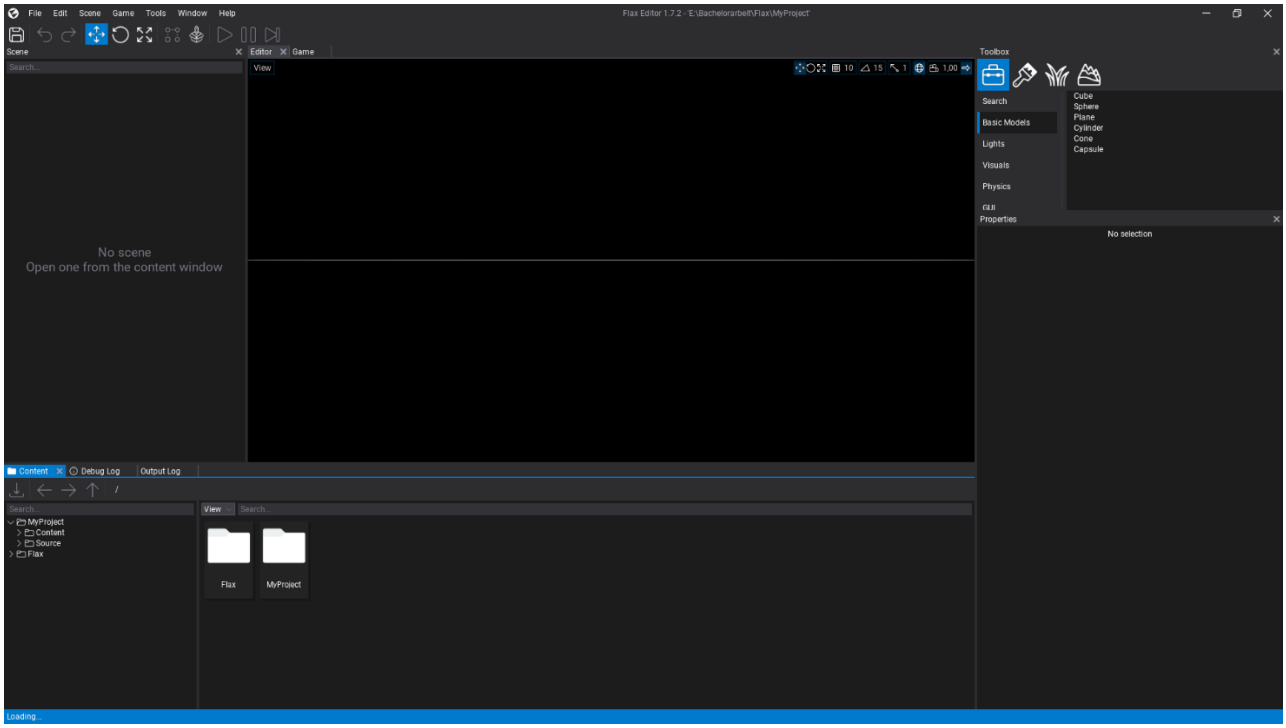

Abb. 5 Benutzeroberfläche Flax

Die Benutzeroberfläche von *Flax* ist beim ersten Öffnen recht schlicht gehalten und in einzelne Fenster aufgeteilt (siehe **Abb. 5**). Auch für Neueinsteigende wird relativ schnell ersichtlich, dass das **Content**-Fenster zur Übersicht einzelner Dateien im Projekt dient. Im **Scene**-Fenster links werden Nutzende darauf hingewiesen, dass noch keine **Scene** geöffnet ist und diese erst im **Content**-Fenster geöffnet werden muss – durch diese kurze Erklärung werden Nutzenden bereits erste Abläufe innerhalb der Game Engine klar kommuniziert. Auch die mitunter recht großen Icons unter der Menüleiste vereinfachen die Benutzung wichtiger Funktionen der Game Engine. Auf der rechten Seite findet man das **Toolbox**-Fenster, durch das **Basic Models**, **Lights**, **Visuals** und weiteres erstellt und bearbeitet werden können. Über die Menüleiste können einzelne Änderungen an der **Scene** und am **Game** vorgenommen werden, sowie weitere Fenster geöffnet werden. Über den **Play**-Knopf unter der Menüleiste kann die aktuell geöffnete **Scene** gestartet werden. Insgesamt ist die Benutzeroberfläche von *Flax* sehr aufgeräumt und sinnvoll sortiert. Menüpunkte sind leicht nachvollziehbar benannt und entsprechend benutzbar. Der Ersteindruck bezüglich der Benutzerfreundlichkeit ist also zusammengefasst sehr positiv. Einzig negativer Aspekt der ersten Erfahrung mit der Benutzeroberfläche sind die fehlenden Erklärungen zu den mitunter sehr vielen unterschiedlichen Ordnern im **Content**-Fenster.

Anwendbarkeit

Zusammensetzbarkeit

Flax unterstützt viele gängigen Dateiformate für Bild-[126] und 3D-Model-Dateien.[127] Auch Audio-Dateien werden unterstützt, wobei hier keine genaue Liste der unterstützten Dateiformate vorliegt, aber mindestens **mp3**, **wav** und **ogg** unterstützt werden.[128] Einzelne Assets können per Drag-and-Drop in das jeweilige *Flax*-Projekt importiert werden. Sowohl für Bilddateien als auch 3D-Modeldateien gibt es beim Import noch weitere spezifische Einstellungsmöglichkeiten. Für das 3D-Objekt können hier beispielsweise Anpassungen in der Skalierung, Rotation und Position gemacht werden, sodass der Unterschied zwischen dem ursprünglichen Referenz-Koordinatensystem in *Blender* (rechtshändig, mit Z-Achse als Höhenanzeige) und dem neuen Koordinatensystem in *Flax* (rechtshändig, mit Y-Achse als Höhenanzeige) von vorneherein ignoriert werden kann. Das 3D-Objekt kann darüber hinaus direkt als solches einer Spielszene hinzugefügt werden und es wird automatisch ein eigener Ordner für das Objekt angelegt, in dem die zugehörigen Materials gespeichert sind. Assets können aus *Flax* entweder einzeln oder zusammengefasst leicht exportiert werden und somit in anderen *Flax*-Projekten oder an anderer Stelle wiederverwendet werden – diese Funktion ist aber nur für Assets verfügbar, die nicht spezifisch innerhalb der Game Engine erzeugt wurden. Skripte und Spielabschnitte können also nicht exportiert werden.

Effizienz der Nutzung

Zu den Funktionen, die *Flax* Nutzenden bietet, gehört unter anderem die Implementierung von Licht-[129] und Audio-Elementen[130], die Erstellung von Animationen[131], die Erstellung von Graphen innerhalb des Visual Scripting Systems[132], verschiedene Werkzeuge zur Analyse des Spiels auf unterschiedlichen Leistungsebenen[133] und viele weitere. Einen eigenen Asset Store besitzt die Game Engine aber nicht. Bezüglich der Versionsverwaltung mit *Git* stellt *Flax* lediglich Beispiele für nützliche *gitignore*- als auch *gitattributes*-Dateien zur Verfügung.[134] Eine direkte Verwendung von *Git* oder ähnlichem innerhalb der Game Engine ist nicht möglich und ein eigenes Versionsverwaltungssystem existiert auch nicht.

[126] vgl. Flax o. J.r
[127] vgl. Flax o. J.k
[128] vgl. Flax o. J.j
[129] vgl. Flax o. J.m
[130] vgl. Flax o. J.c
[131] vgl. Flax o. J.b
[132] vgl. Flax o. J.u
[133] vgl. Flax o. J.o
[134] vgl. Flax o. J.t

Besonderheiten

Die größte Besonderheit von *Flax* ist die Möglichkeit Programmcode sowohl in *C#* als auch in *C++* zu schreiben und den Code eines Spiels beliebig aus den beiden Sprachen zusammenzusetzen. Entsprechend kann hier leicht, wo nötig, auf die höhere Performanz von *C++* gesetzt werden. Eine weitere Besonderheit, die damit in Zusammenhang steht, ist das sogenannte **Hot-reloading** beider Sprachen im Editor, wodurch dieser nicht geschlossen werden muss, während die Scripts kompiliert werden.[135]

Übertragbarkeit

Programmierparadigmen

Flax verwendet die objektorientierte Programmiersprachen *C#* und *C++*, weswegen auch alle in *Flax* verwendeten Skripte der Objektorientierung folgen. Dies äußert sich unter anderem darin, dass zunächst jedes Skript von der Basisklasse **Script**[136] erbt und hierdurch bereits einige grundlegende Funktionen zur Verfügung gestellt bekommt. Geschriebene Skripte können an **Actor** angehängt werden, wodurch hier für Nutzende leicht der Zusammenhang zwischen dem Konzept eines Objekts auf Seiten der Programmierung mit dem einzelnen Spielelement in *Flax* dargestellt wird.

Hieraus abgeleitet lassen sich jegliche Konzepte, die in *Flax* bezüglich der Objektorientierung erlernt und angewendet werden können, leicht auf andere objektorientierte Programmiersprachen und entsprechende Game Engines übertragen. Bezüglich der in dieser Arbeit evaluierten Game Engines trifft das auf *Unity*, *Unreal Engine*, *Cocos Creator*, *Godot*, *jMonkeyEngine*, *FUDGE* und *Vektoria* zu. Darüber hinaus lassen sich erlernte Kenntnisse bezüglich der Programmiersprachen *C#* und *C++* direkt auf *Unity*, *Unreal Engine*, *Godot* und *Vektoria* übertragen.

Schlüsselkonzepte

Zusammenhängende Spielabschnitte werden in *Flax* als **Scenes**[137] bezeichnet. Diese setzen sich – der Freie-Szenen-Struktur folgend – aus einzelnen Elementen, den sogenannten **Actors**[138] zusammen, an die wiederum **Components** angehängt werden, durch die sie unterschiedliche Eigenschaften und Funktionen erhalten. **Actors** sind in einer **Scene** unabhängig voneinander angeordnet, können darin aber auch in sogenannten **Parent-Child-Verbindungen** zusammengesetzt werden, wodurch Position, Rotation und

[135] vgl. Flax o. J.d
[136] vgl. Flax o. J.e
[137] vgl. Flax o. J.q
[138] vgl. Flax o. J.a

Skalierung des untergeordneten **(Child-)Actors** von der des übergeordneten **(Parent-) Actors** abhängig gemacht wird. **Actors**, die zur Wiederverwendung benutzbar sein sollen, können in sogenannte **Prefabs**[139] umgewandelt werden. Dabei werden Änderungen an dem ursprünglichen **Prefab** direkt auf alle Kopien beziehungsweise Instanzen dieses Elements übertragen. Änderungen an Kopien eines **Prefabs** werden erst auf das ursprüngliche und alle weiteren Kopien übertragen, wenn dies explizit im Editor bestätigt wird.

Die Freie-Szenen-Struktur aus frei zusammensetzbaren **Scenes** und unabhängigen **Actors** lässt sich auf viele andere Game Engines, die den gleichen Ansatz verfolgen, übertragen. Von den in dieser Arbeit evaluierten Game Engines trifft das auf *GameMaker*, *Unity*, *Unreal Engine* und *Cocos Creator* zu.

Meinungen von Lehrenden und Zusammenfassung

Da *Flax* in keinem der befragten Lehrkontexte entweder im Lehrplan eingebunden ist oder in Projekten eingesetzt wird, entfällt an dieser Stelle die Beleuchtung der Meinungen von Lehrenden bezüglich der Game Engine.

Während die ersten Schritte im Umgang mit der Game Engine noch recht leicht ausfallen, wird der allgemeine Einstieg in komplexere Funktionen von *Flax* durch die wenigen und überschaubaren Tutorials sowie der kaum vorhandenen Community entscheidend erschwert. Bezüglich der Anwendbarkeit weist *Flax* aber viele sinnvolle Konzepte auf, die im Rahmen der Lehre für einen Allround-Einsatz gut geeignet sind. Vor allem die Unterstützung von *C#* und *C++* sowie die in Verbindung damit oder der Game Engine im Allgemeinen erlernten Konzepte machen die Übertragbarkeit auf andere Game Engines sehr einfach. Zusammengefasst eignet sich *Flax* entsprechend gut für Lehrkontexte, die eine tiefgreifende Unterstützung der Lernenden über die Grundsätze hinaus ermöglichen können und gleichzeitig eine gute Übertragbarkeit der erlernten Fähigkeiten auf andere Game Engines gewährleisten wollen.

[139] vgl. Flax o. J.n

4.1.2 GameMaker

Kriterium	Ergebnis
Definitionstreue	*GameMaker* ermöglicht die Erstellung von 2D- und 3D-Spielen mithilfe der eigenen Programmiersprache GML Code oder GML Visual.[140]
Aktualität	Version 2024.2 wurde am 04. März 2024 veröffentlicht.[141] Auf Steam wurde zuletzt (18.02.2024) unter anderem *Turnip Boy Robs a Bank* veröffentlicht.[142]
Preisgestaltung	*GameMaker* ist kostenfrei für die nichtkommerzielle Nutzung Die kommerzielle **Professional**-Lizenz kostet einmalig 99,99$, die **Enterprise**-Lizenz (die zusätzlich Konsolenexporte erlaubt) kostet monatlich 79,99$.[143]
Entwicklungsplattformen	*Windows, MacOS*
Zielplattformen	*GX.games*, *Windows, MacOS, Web, Mobile*[144]
Systemanforderungen	Allgemein: Dual-Core-CPU, 2GB RAM, OpenGL-4-kompatible GPU, HDD *Windows*: 7 mit SP1 oder neuer *MacOS*: Big Sur oder neuer[145]

Tabelle 6 Evaluation der Qualifikationskriterien – GameMaker

Erlernbarkeit

Offizielle Lernmaterialien

GameMaker bietet zwei Anlaufstellen für Neueinsteigende, um erste Informationen über die Game Engine zu erhalten; einerseits gibt es eine Reihe an **Tutorials**, die bereits auf der Startseite leicht findbar verlinkt sind. Und andererseits gibt es noch ein Manual, das jedoch nur klein in der Fußzeile der Webseite verlinkt ist. Über erstere Option findet man insgesamt 92 Tutorials, die über Filter weiter aufgeteilt werden können – letztendlich sind aber 82 der 92 Tutorials für das Anfänger-Level geeignet[146] wodurch sehr wenige der Tutorials für weiterführende Themen ausgelegt sind.

[140] vgl. GameMaker o. J.b
[141] vgl. Matharoo 2024
[142] vgl. SteamDB o. J.a
[143] vgl. GameMaker o. J.l
[144] vgl. GameMaker o. J.l
[145] vgl. GameMaker o. J.l
[146] vgl. GameMaker o. J.m

Das Manual auf der anderen Seite gibt zusätzlich konkrete Informationen zu den ersten Schritten im Umgang mit der Game Engine, erklärt also unter anderem, wie das Editorfenster aufgebaut ist und wie einzelne erste Spielobjekte erstellt werden können. Sowohl die Tutorials als auch das Manual stehen nur auf Englisch zur Verfügung und eine Anpassung beider an eine bestimmte Version der Game Engine ist nicht möglich. Bezüglich der Aktualität findet sich zu jedem Tutorial ein entsprechendes Veröffentlichungsdatum, wobei die meisten der 92 Tutorials innerhalb der letzten zwei Jahre veröffentlicht wurden.

Die Reproduzierbarkeit wurde anhand des Tutorials **Make Your Own Arcade Space Shooter** mithilfe der Version 2024.2 untersucht. Das Tutorial erklärt Schritt für Schritt und sowohl unter der Verwendung von *GMLCode* als auch *GMLVisual* die Erstellung eines kleinen Arcade-Spiels. Jeder Schritt ist ausführlich mit entsprechenden Erklärungen und Bildern ausgestattet, wodurch sich das gezeigte sehr leicht und ohne Probleme replizieren lässt. Lediglich die Tatsache, dass das recht lange Tutorial auf einer einzigen Seite ohne Navigationsoptionen abgebildet ist und *GMLCode*- beziehungsweise *GMLVisual*-Abschnitte nicht je nach präferierter Wahl getrennt voneinander ausgeblendet werden können, macht das Ganze recht unübersichtlich.

Der Einstieg in den Umgang mit *GameMaker* fällt zusammengefasst durch die vielen verfügbaren Tutorials und das ausführliche Manual recht leicht und wird nur dadurch getrübt, dass weiterführende Themen eher weniger abgedeckt werden und der Fokus stark auf der grundsätzlichen Erlernung der Game Engine liegt. Die Game Engine ist zusammenfassend definitiv für das eigenständige Lernen geeignet, kann aber auch vom situierten Lernen profitieren, da keine Möglichkeiten zur Selbstbeurteilung (über die bloße eigenständige Betrachtung und Evaluation der erzielten Ergebnisse hinaus) existieren und eine Fremdbeurteilung durch eine Lehrperson entsprechend sinnvoll wäre. Neben Tutorials und dem Manual auf der Webseite der Game Engine gibt es auf dem offiziellen *YouTube*-Kanal noch eine relativ große Menge an Video-Tutorials, die neben Englisch auch teilweise in Spanisch verfügbar sind.[147]

Inoffizielle Lernmaterialien

Neben den über 200 Videos auf dem offiziellen *GameMaker YouTube*-Kanal finden sich auf *YouTube* recht wenige Videos, die sich mit einzelnen Aspekten der Game Engine beschäftigen. Häufiger zu finden sind dagegen Tutorial-Reihen, die sich dann entsprechend um die komplette Erstellung einzelner kleiner Spiele drehen. Darüber hinaus existiert auf der Plattform *Reddit* ein eigener Bereich für *GameMaker*, in dem die Community bestehend

[147] vgl. GameMaker o. J.a

aus über 80.000 Mitgliedern[148] regelmäßig Tutorials miteinander teilt. Insgesamt stehen Lernenden besonders hierdurch also recht viele Ressourcen für unterschiedliche Kompetenzstufen und Kontexte zur Verfügung.

Heuristische Analyse der Benutzeroberfläche

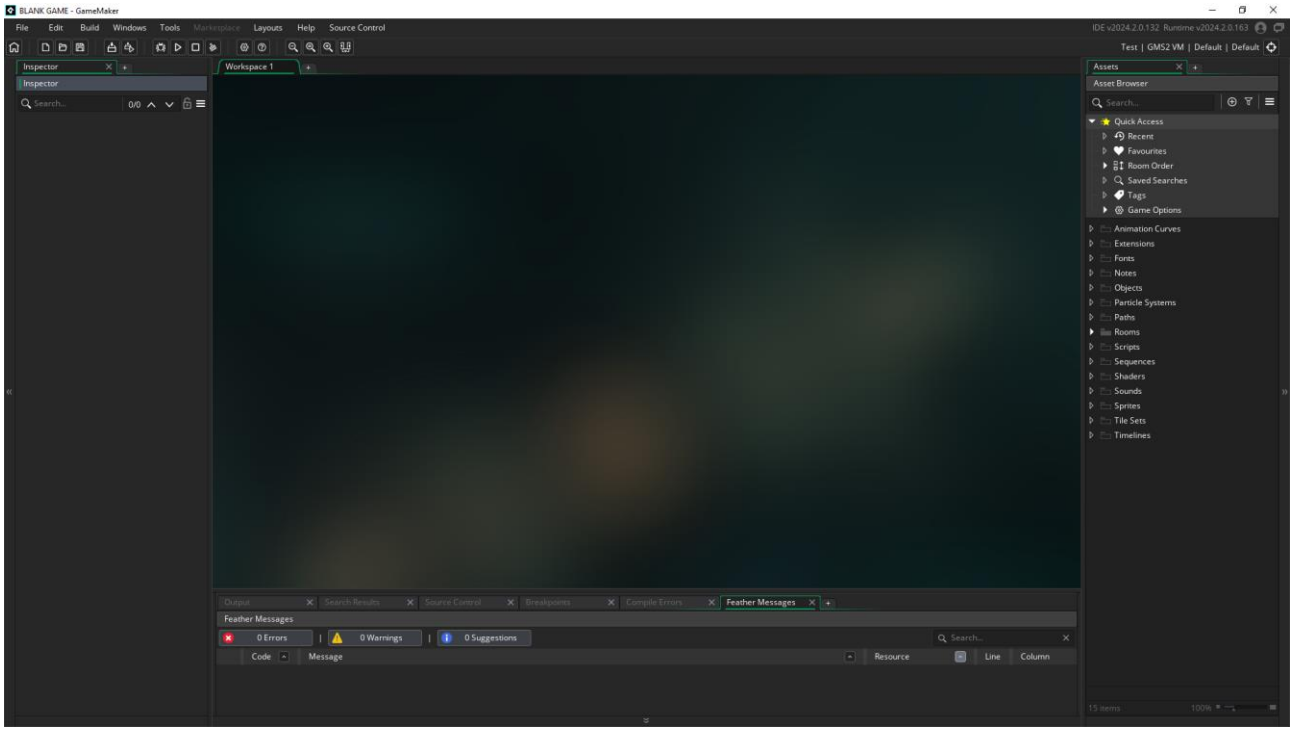

Abb. 6 Benutzeroberfläche GameMaker

Die Benutzeroberfläche von *GameMaker* wirkt beim ersten Öffnen sehr schlicht gehalten und ist in einzelne Fenster aufgeteilt (siehe **Abb. 6**). Auch für Neueinsteigende wird relativ schnell ersichtlich, dass das **Assets**-Fenster zur Übersicht einzelner Dateien im Projekt dient und das **Workspace**-Fenster vermutlich den Hauptarbeitsplatz darstellt. Dieses ist jedoch leer und lässt entsprechend nicht direkt ableiten, was genau darin gemacht werden kann. Das **Inspector**-Fenster auf der linken Seite dient darüber hinaus zum Erhalt weiterer Informationen über einzeln ausgewählte Assets. Einzelne Assets, die über ein entsprechendes Menü im **Assets**-Fenster erstellt werden können, werden über ein Icon repräsentiert und damit leicht voneinander unterscheidbar dargestellt. Durch die Menüleiste können leicht weitere Fenster geöffnet werden. Über den **Play**-Knopf direkt unter der Menüleiste kann das aktuelle Spiel gestartet werden, wodurch sich automatisch ein eigenständiges Spielfenster öffnet. Insgesamt ist die Benutzeroberfläche von *GameMaker* sehr aufgeräumt und sinnvoll sortiert. Menüpunkte sind leicht nachvollziehbar benannt und

[148] vgl. /r/gamemaker o. J.

entsprechend benutzbar. Zu den einzigen negativen Aspekten der ersten Erfahrung mit der Benutzeroberfläche gehören die fehlenden Erklärungen zu den mitunter sehr vielen Menüunterpunkten und das etwas abschreckende große leere **Workspace**-Fenster in der Mitte des Bildschirms, das zunächst nicht sonderlich zum Arbeiten einlädt, weil es auch durch den unscharfen Hintergrund etwas erzwungen platziert wird.

Anwendbarkeit

Zusammensetzbarkeit

GameMaker unterstützt viele gängigen Dateiformate für Bild-[149] und Audio-Dateien.[150] Diese können per Rechtsklick im **Assets**-Fenster oder alternativ per Drag-and-Drop entsprechend in die Game Engine importiert werden. Für 3D-Modell-Dateien stellt *GameMaker* von Haus aus keine Möglichkeiten zum Import zur Verfügung und diese Funktionalität muss entsprechend manuell nachgerüstet werden.[151] Nach dem Import gibt es für Bilddateien weitere Einstellungsmöglichkeiten zu Filtern, Auflösung und weiterem. Gleichzeitig können sie über den integrierten Sprite-Editor direkt bearbeitet werden. Assets können aus *GameMaker* zusammengebündelt in Form von sogenannten **Asset Packages** exportiert werden.[152] Diese können dann in anderen *GameMaker*-Projekten wiederum importiert und somit wiederverwendet werden.

Effizienz der Nutzung

Die Funktionen, die *GameMaker* Nutzenden bietet, beziehen sich hauptsächlich auf die Entwicklung von 2D-Spielen. Hierzu gehören unter anderem die Erstellung von Sprites[153], die Implementierung von Licht-[154] und Audio-Elementen[155], die Erstellung von Animationen[156], die Erstellung von Programmabläufen mithilfe von *GMLVisual* und viele weitere. Bezüglich der Versionsverwaltung unterstützt *GameMaker* von Haus aus *Git* und besitzt einen entsprechenden Menüpunkt zur Verbindung des Projekts mit einer *Repository*.[157] Sobald das Projekt entsprechend verbunden wurde, werden Änderungen an

[149] vgl. GameMaker o. J.j
[150] vgl. GameMaker o. J.d
[151] vgl. DragoniteSpam 2024
[152] vgl. GameMaker o. J.g
[153] vgl. GameMaker o. J.b
[154] vgl. GameMaker o. J.f
[155] vgl. GameMaker o. J.d
[156] vgl. GameMaker o. J.c
[157] vgl. GameMaker o. J.e

einzelnen Dateien jeweils durch ein Icon angezeigt[158], was die Zusammenarbeit an einem *GameMaker*-Projekt innerhalb eines Teams sehr vereinfacht.

Besonderheiten

Die größte Besonderheit von *GameMaker* ist die Möglichkeit zur Anlegung mehrerer **Workspaces** innerhalb des Editorfensters zur Strukturierung einzelner Spielobjekte. Einzelne Spielobjekte können innerhalb des **Workspace** erstellt, angepasst und um Funktionen wie Code oder anderes erweitert werden. Gleichzeitig lassen sich mit mehreren **Workspaces** zusammengehörige Objekte einfach und überschaubar sortieren. Dieser Ansatz sorgt dafür, dass *GameMaker*, sobald man sich mit den **Workspaces** zurechtgefunden hat, sehr gut für visuelles Arbeiten geeignet ist und einzelne Zusammenhänge, Funktionen und Spielelemente sehr übersichtlich präsentiert und entsprechend bearbeitet werden können.

Übertragbarkeit

Programmierparadigmen

GameMaker verwendet die Programmiersprache *GMLCode*, die im Kontext der Game Engine eher imperativ eingesetzt wird und keine Möglichkeiten zur Objektorientierung bietet.[159]

Entsprechend lassen sich damit erlernte Fähigkeiten eher schlecht auf andere in dieser Arbeit evaluierte Game Engines übertragen, da sich die meisten auf die Objektorientierung stützen. Am ehesten kann hier für eine Übertragbarkeit auf *Defold* argumentiert werden, da diese ebenfalls auf die imperative Programmierung setzt.

Schlüsselkonzepte

Zusammenhängende Spielabschnitte werden in *GameMaker* als **Rooms**[160] bezeichnet. Diese setzen sich – der Freie-Szenen-Struktur folgend – aus einzelnen Elementen, den sogenannten **Objects**[161] zusammen, die nach einer Erstellung mithilfe von **Properties** und **Options** Eigenschaften und Funktionen erhalten und gleichzeitig mit Programmcode erweitert werden können. **Objects** sind in einem **Room** unter dem Reiter **Instances** unabhängig voneinander angeordnet, können aber auch in sogenannten **Parent-Child-Verbindungen** zusammengesetzt werden, wodurch Programmfunktionen des

[158] vgl. GameMaker o. J.o
[159] vgl. GameMaker o. J.n
[160] vgl. GameMaker o. J.i
[161] vgl. GameMaker o. J.k

übergeordneten **(Parent-)Objects** auf das untergeordnete **(Child-)Object** übertragen werden.[162] Innerhalb von **Rooms** können mehrere sogenannte **Layer** erstellt werden, um Instanzen und weitere Assets übersichtlich voneinander zu trennen. **Objects** sind standardmäßig zur Wiederverwendung geeignet und alle in einem Room instanziierten **Objects** sind vom jeweiligen Hauptobjekt im Assets-Ordner abhängig. Entsprechend werden Änderungen an dem ursprünglichen **Object** direkt auf alle Instanzen dieses Elements übertragen. Änderungen an einzelnen Instanzen beziehen sich dabei nur auf die jeweilige Instanz.

Die Freie-Szenen-Struktur aus frei zusammensetzbaren **Rooms** und unabhängigen **Objects** lässt sich auf viele andere Game Engines, die den gleichen Ansatz verfolgen, übertragen. Von den in dieser Arbeit evaluierten Game Engines trifft das auf *Flax*, *Unity*, *Unreal Engine* und *Cocos Creator* zu.

Meinungen von Lehrenden und Zusammenfassung

Da *GameMaker* von keinem der befragten Dozierenden im Lehrplan eingesetzt wird, entfällt an dieser Stelle die Beleuchtung der Meinungen von Lehrenden bezüglich der Game Engine. *GameMaker* wurde innerhalb der qualitativen Experteninterviews lediglich als Game Engine angesprochen, die teilweise von Studierenden in Projekten verwendet wird[163], entsprechend insgesamt eine gewisse Bekanntheit hat und hierdurch für den Einsatz im Lehrplan immerhin evaluiert wurde – im direkten Vergleich mit *Godot* aber nicht vollständig überzeugen konnte.[164]

Zusammengefasst ist *GameMaker* eine Game Engine, die sich vor allem gut für einen Einstieg im Bereich der 2D-Spieleerstellung eignet. Die Erlernung der Game Engine ist durch die vielen Tutorials recht gut möglich. Bezüglich des Kriteriums der Anwendbarkeit überzeugt GameMaker vor allem durch die eingebaute Versionsverwaltung und die sehr sinnvollen Möglichkeiten zum visuellen Arbeiten mithilfe der **Workspaces**. Die Übertragbarkeit der erlernten Konzepte mit *GameMaker* auf andere Game Engines ist aber bezüglich der Programmiersprache etwas erschwert. *GameMaker* ist also besonders für Lehrkontexte geeignet, die einen ersten leichten Einstieg in die Spieleerstellung bieten wollen oder die 2D-Spieleentwicklung in den Vordergrund setzen.

[162] vgl. GameMaker o. J.h
[163] vgl. I1 Z. 45 f.
[164] vgl. I0 Z. 60

4.1.3 Unity

Qualifikationskriterien

Kriterium	Ergebnis
Definitionstreue	*Unity* (auch Unity3D oder **Unity Editor**) ermöglicht die Erstellung von 2D- und 3D-Spielen mithilfe der Programmiersprache *C#* oder *Unity* Visual Scripting.[165]
Aktualität	LTS-Version 2022.3.20f1 wurde am 14. Februar 2024 veröffentlicht. Auf Steam wurde zuletzt (31. Januar 2024) unter anderem *Murders on the Yangtze River* veröffentlicht.[166]
Preisgestaltung	**Student**- und **Personal**-Version kostenlos bis 100.000$ Umsatz innerhalb eines Jahres, **Pro**-Version ab $185 pro Monat, **Industry**-Version ab $450 pro Monat, **Enterprise**-Version individueller Preis.[167]
Entwicklungsplattformen	*Windows, MacOS, Linux*[168]
Zielplattformen	*iOS, android, WebGL, Playstation 4, Playstation 5, Nintendo Switch, Xbox One, Xbox Series S, Xbox Series X, XR-Plattformen* und weitere[169]
Systemanforderungen	CPU mit X64 Architektur mit SSE2 Unterstützung und DX10-, DX11- oder DX12-fähige GPU (*Windows*), Metal-fähige Intel- oder AMD-GPU (*MacOS*) oder OpenGL 3.2+ oder Vulkan-fähige Nvidia- oder AMD-GPU (*Linux*)[170]

Tabelle 7 Evaluation der Qualifikationskriterien - Unity

Erlernbarkeit

Offizielle Lernmaterialien

Unity bietet mit **Unity Learn** eine eigene umfassende Lernplattform für die Erlernung des Umgangs mit der Game Engine.[171] Zur Nutzung des Dienstes muss man eine **Unity ID** erstellen und sich mit dieser anmelden. Anschließend stehen Nutzenden insgesamt fünf sogenannte **Pathways** (dt.: Pfad) zur Verfügung, durch die man innerhalb verschiedener

[165] vgl. Unity Technologies o. J.c
[166] vgl. SteamDB o. J.b
[167] vgl. Unity Technologies o. J.a
[168] Unity Technologies o. J.x
[169] Jead 2024
[170] Unity Technologies o. J.v
[171] vgl. Unity Technologies o. J.e

Spezialisierungen Einblicke in *Unity* erlangen kann. Unterschieden wird hierbei zwischen **Unity Essentials** (hier wird eine generelle Einleitung in den Umgang mit *Unity* gegeben), **Junior Programmer** (hier wird ein besonderer Fokus auf erste Schritte in der Programmierung gelegt), **Creative Core** (hier wird ein tieferer Einblick in die Game Engine selbst gegeben), **VR Development** (hier wird ein besonderer Fokus auf die VR-Entwicklung gelegt) und **Mobile AR Development** (hier wird ein besonderer Fokus auf die Entwicklung von AR-Apps für iOS- und Android-Geräte gelegt).[172] Jeder dieser Pfade besteht aus mehreren Missionen, die wiederum in einzelne Projekte und Tutorials aufgeteilt sind. Darüber hinaus lässt sich auf der Übersichtseite für einen einzelnen Pfad die Bearbeitungsdauer, das **Level** (also der Schwierigkeitsgrad), sowie ein **Completion bonus** (dt.: Abschlussprämie) und konkrete **Skills** (dt.: Fertigkeiten) die durch die Bearbeitung des Pfades erlernt werden können, einsehen.[173] Zusätzlich stellt **Unity Learn** ein **Learning Dashboard** (dt.: Lerntafel) zur Verfügung, auf dem eine Übersicht über den allgemeinen Lernfortschritt einsehbar ist. Hierdurch wird gleichzeitig eine gute Möglichkeit zur Selbstbeurteilung des eigenen Lernfortschritts geboten. Die einzelnen Pfade stehen nur auf Englisch zur Verfügung und eine Anpassung des Pfades an eine bestimmte Version der Game Engine ist nicht möglich. Auch bezüglich der Aktualität der Pfade lassen sich keine Aussagen treffen, da ein entsprechendes Veröffentlichungsdatum oder ähnliches nicht einsehbar ist.

Die tatsächliche Reproduzierbarkeit wurde anhand des **Unity Essentials** Pfades untersucht, da dieser sich explizit an Neueinsteigende richtet. Der Pfad besteht aus insgesamt drei Missionen und gibt eine Bearbeitungsdauer von zwei Wochen bei täglicher Arbeit von vier bis fünf Stunden an.[174] Die erste Mission leitet Nutzende durch die Installation von **Unity Hub** und dem **Unity Editor**, erklärt die Anlegung und Verwaltung von Spieleprojekten und gibt einen Überblick über das **Unity Learning Ecosystem**. Die einzelnen Projekte werden teilweise mit einem Übersichtsvideo eingeleitet, einzelne Schritte sind in Textform übersichtlich gestaltet und können einzeln als abgeschlossen markiert werden – darüber hinaus können Nutzende Kommentare hinterlassen. Am Ende eines Projekts erhält man eine Abschlussprämie und wird nach einer Bewertung der bisherigen Erklärungen gefragt. Für die Bearbeitung einzelner Aspekte des Tutorials wird die jeweils aktuelle LTS-Version von *Unity* empfohlen.[175] Im Rahmen der Evaluation wurde stattdessen die älteste LTS-Version verwendet (2017.1.0f1[176]), mit der die Tutorials größtenteils auch

[172] vgl. Unity Technologies o. J.e
[173] vgl. Unity Technologies o. J.g
[174] vgl. Unity Technologies o. J.g
[175] vgl. Unity Technologies o. J.d
[176] vgl. Unity Technologies o. J.h

reproduzierbar waren – einzelne Menüpunkte waren in unterschiedlichen *Unity*-Versionen an unterschiedlichen Stellen platziert, dementsprechend ist für die Anwendung der Tutorials auf ältere Versionen teilweise eine entsprechende Suche nötig. Alle weiteren Schritte des Pfades konnten dennoch auch mit dieser älteren Version vervollständigt werden. Hierbei wurde neben den ersten Schritten in der 3D-Umgebung von *Unity* und darauffolgender Objektmanipulation auch die Programmierung und Beeinflussung von einzelnen Spielobjekten kurz erklärt. Zusätzlich wird auch die generelle Herangehensweise an *Unity*-Projekte beleuchtet. Jede Mission wird mit einem Checkpoint abgeschlossen, an dem erlernte Fertigkeiten durch ein Quiz abgeschlossen werden, wodurch quasi eine Fremdbeurteilung simuliert wird. Der erfolgreiche Abschluss einer Mission schaltet je nach Inhalt auch einzelne Belohnungen frei, wie zum Beispiel **Badges** (dt.: Abzeichen) oder Assets.

Neben **Unity Learn** können weitere Informationen, Hilfestellungen und Tutorials über die Game Engine innerhalb der ausführlichen Dokumentation, dem Forum und auf dem offiziellen *YouTube*-Kanal abgerufen werden. Auf letzterem können auch Einblicke in weiterführende Themen erlangt werden, beispielsweise mit der Videoreihe zu **Programming Design Patterns** (dt.: Entwurfsmuster für die Programmierung).[177] Durch die ausführlichen Pfade in **Unity Learn** und die große Menge an weiteren offiziellen Lernmaterialien ist *Unity* sehr gut sowohl für das eigenständige als auch das situierte Lernen geeignet.

Inoffizielle Lernmaterialien

Neben den über 2.500 Videos auf dem offiziellen *Unity YouTube*-Kanal[178] finden sich auf *YouTube* unzählige Videos, die sich einzelnen Aspekten der *Unity* Game Engine widmen und viele Kanäle, die explizit auf Tutorials zu *Unity* ausgelegt sind. Zu den Kanälen mit der größten Menge an Lernmaterialien zählen unter anderem **Code Monkey**[179], **Game Dev Guide**[180] und **Jason Weimann**[181]. Darüber hinaus existiert auf der Plattform *Reddit* ein sehr großer Bereich für *Unity*, in dem die Community bestehend aus über 100.000 Mitgliedern[182] regelmäßig Tutorials miteinander teilt. Insgesamt stehen Lernenden also sehr viele verschiedene und leicht benutzbare Ressourcen für unterschiedliche Kompetenzstufen und Kontexte zur Verfügung.

[177] vgl. Unity 2024
[178] vgl. Unity o. J.
[179] vgl. Code Monkey o. J.
[180] vgl. Game Dev Guide o. J.
[181] vgl. Weimann o. J.
[182] vgl. /r/unity o. J.

Heuristische Analyse der Benutzeroberfläche

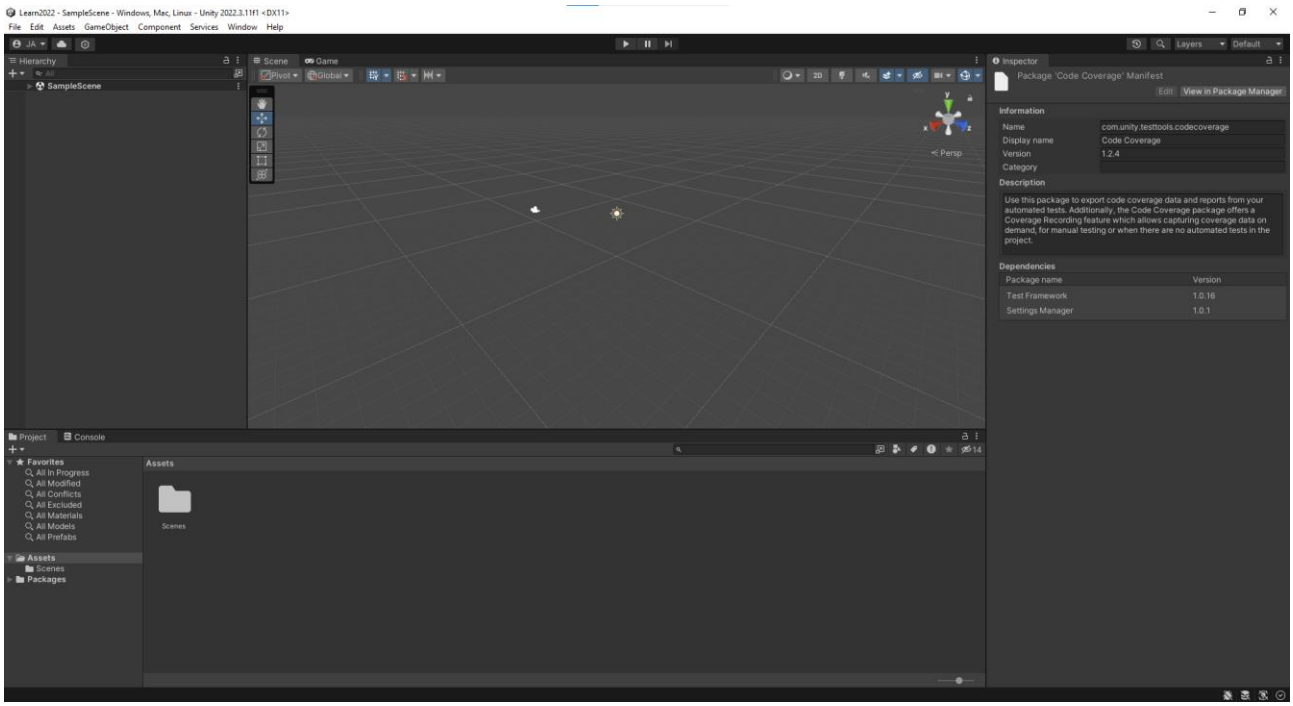

Abb. 7 Benutzeroberfläche Unity

Die Benutzeroberfläche von *Unity* ist beim ersten Öffnen recht schlicht gehalten und in einzelne Fenster aufgeteilt (siehe **Abb. 7**). Auch für Neueinsteigende wird relativ schnell ersichtlich, dass das **Project**-Fenster zur Übersicht einzelner Dateien im Projekt dient und das **Scene**-Fenster den Inhalt der aktuell bearbeiteten Spielszene enthält. Durch Klicks auf die in der **SampleScene** bereits platzierten Objekte **Main Camera** und **Directional Light** wird der entsprechende Eintrag im **Hierarchy**-Fenster geöffnet und die Details des Objekts im **Inspector**-Fenster angezeigt. Gleichzeitig ist für das ausgewählte Objekt das **Move Tool** aktiviert, wodurch Pfeile sichtbar werden, mit denen man das Objekt verschieben kann. Jegliche Änderungen, die man dann an einzelnen Objekten im **Scene**-Fenster durchführt, werden auch im **Inspector**-Fenster angezeigt – und umgekehrt. Durch die Menüleiste können leicht verständlich **GameObjects** und andere Assets erzeugt und **Components** an erstere angehängt werden, sowie weitere Fenster geöffnet werden. Über den **Play**-Knopf in der oberen Mitte kann das aktuelle Spiel gestartet werden, wodurch sich automatisch das **Game**-Fenster öffnet. Insgesamt ist die Benutzeroberfläche von *Unity* sehr aufgeräumt und sinnvoll sortiert. Menüpunkte sind leicht nachvollziehbar benannt und entsprechend benutzbar. Beim Verweilen des Mauszeigers über einzelne Knöpfe des **Scene**-Fensters werden kurze Erklärungen der einzelnen Funktionen gegeben. Der Ersteindruck bezüglich

der Benutzerfreundlichkeit ist also insgesamt sehr positiv. Einzig negativer Aspekt der ersten Erfahrung mit der Benutzeroberfläche sind die fehlenden Erklärungen zu den mitunter sehr vielen Menüunterpunkten.

Anwendbarkeit

Zusammensetzbarkeit

Unity unterstützt viele gängige Dateiformate für Bild-, 3D-Model-, Audio- und Text-Dateien.[183] Diese können über den Menüpunkt **Assets → Import New Asset** oder per Drag-and-Drop in das jeweilige *Unity*-Projekt importiert werden. Nach dem Import gibt es für Bilddateien weitere Einstellungsmöglichkeiten zu Filtern, Auflösung und weiterem. 3D-Modeldateien werden als ein gebündeltes Asset importiert, das sich einerseits aus dem 3D-Mesh und andererseits aus den darin verwendeten Materialien zusammensetzt und können somit direkt als **GameObject** in eine Spielszene importiert werden. Besonderheiten finden sich hier in den Rotationen des Objekts, die sich zwischen den ursprünglichen Einstellungen in *Blender* und dem letztendlichen **GameObject** in *Unity* unterscheiden. Dies liegt an der Ausrichtung der jeweiligen Koordinatensysteme; während *Blender* ein rechtshändiges Koordinatensystem verwendet, bei dem die Z-Achse der Höhenanzeige dient, besitzt *Unity* ein linkshändiges Koordinatensystem mit der Y-Achse als Höhenanzeige. Dies führt zu ungleichen Rotationen des importierten 3D-Models bei nicht weiter angepassten Export-Einstellungen in *Blender*. Assets können aus *Unity* entweder einzeln oder zusammengebündelt in Form von sogenannten **Packages** exportiert werden. Diese können dann in anderen *Unity*-Projekten wiederum importiert und somit wiederverwendet werden.

Effizienz der Nutzung

Zu den Funktionen, die *Unity* Nutzenden bietet, gehört unter anderem die Erzeugung von Textelementen [184], die Implementierung von Licht- [185] und Audio-Elementen [186], die Erstellung von Animationen [187], die Erstellung von Graphen innerhalb des *Unity* Visual Scripting Systems [188], verschiedene Werkzeuge zur Analyse des Spiels auf unterschiedlichen Leistungsebenen [189] und viele weitere. Besonders praktisch sind hierbei der Asset Store, über den direkt in der Game Engine Assets gekauft werden können und der **Package Manager**, durch den gekaufte und weitere von *Unity* zur Verfügung gestellte

[183] vgl. Unity Technologies o. J.w
[184] vgl. Unity Technologies o. J.s
[185] vgl. Unity Technologies o. J.m
[186] vgl. Unity Technologies o. J.k
[187] vgl. Unity Technologies o. J.j
[188] vgl. Unity Technologies o. J.u
[189] vgl. Unity Technologies o. J.p

Packages installiert werden können. Bezüglich der Versionsverwaltung ist die Verwendung von *Git* für *Unity*-Projekte von vielen Problemen geplagt, denen man zwar mit entsprechender Vorarbeit entgegnen kann, die den allgemeinen Arbeitsablauf der Versionsverwaltung aber dennoch erschweren.[190] Ein häufig genannter Problempunkt hierbei ist die Verwendung von meta-Dateien, die besondere Vorsicht beim Umgang mit *Git* erfordern; meta-Dateien werden allgemein verwendet, um die Objektverwaltung innerhalb der Game Engine zu verifizieren. Falls hierbei aber beispielsweise ein Branch-Switch in *Git* durchgeführt wird, ohne dass sämtliche meta-Dateien entsprechend *commited* wurden, kann es schnell zu größeren Problemen mit dem gesamten Spieleprojekt kommen. Eine Alternative zu *Git* bietet *Unity* selbst durch die sogenannte **Unity Version Control** (früher **Unity Collab** oder **Plastic SCM**). Hiermit können einzelne Dateien während der Bearbeitung gesperrt werden, wodurch Merge-Problemen einzelner Dateien von vorneherein vorgebeugt wird. Der Service ist für drei Personen mit einem Speicherplatz von 5GB kostenfrei nutzbar, für alles darüber wird in Abhängigkeit der Teamgröße, des verwendeten Speicherplatzes und weiterer Parameter ein individueller Preis gesetzt, der monatlich bezahlt werden muss.[191]

Besonderheiten

Unity stellt von Haus aus bereits viele praktische Funktionen und Werkzeuge zur Spieleerstellung zur Verfügung. Darüber hinaus können über den **Package Manager** weitere Werkzeuge nachinstalliert werden. Diese befinden sich teilweise in unterschiedlichen Entwicklungsstadien[192], auf welche Nutzende vor der jeweiligen Installation hingewiesen werden. Etwas unübersichtlich sind teilweise die Dokumentationen der einzelnen **Packages**, die in einem eigenständigem – von der restlichen *Unity* Manual unabhängigem – Teil der *Unity* Dokumentation untergebracht sind. Letztendlich sind hierdurch viele Packages doppelt erwähnt (einerseits in einer kurzen Zusammenfassung in der *Unity* Manual und andererseits in einer ausführlichen Version auf der Dokumentationsseite) und es benötigt entsprechenden Mehraufwand die genauen Details zu finden, die man gerade sucht. Darüber hinaus kommt hinzu, dass manche **Packages** einen Ersatz für bereits in der Game Engine vorhandene Funktionen darstellen sollen und entsprechend Nutzenden überlassen wird, welche Version dieser Funktion sie nutzen wollen. Ein weiteres kleines Durcheinander eröffnet sich beim Blick auf die **Render Pipelines** von *Unity*. Hier gibt neben der standardmäßig enthaltenen **Built-In Render Pipeline** auch eine **Universal Render Pipeline**, eine **High Definition Render Pipeline** und

[190] vgl. thoughbot 2017
[191] vgl. Unity Technologies o. J.i
[192] vgl. Unity Technologies o. J.n

eine **Scriptable Render Pipeline**. Die genauen Unterschiede dieser einzelnen **Pipelines** zu durchdringen und die passende Pipeline für das jeweilige Projekt zu finden ist auch aufgrund der hier ebenfalls verteilten beziehungsweise doppelt erwähnten Erklärungen etwas schwierig.

Übertragbarkeit

Programmierparadigmen

Unity verwendet die objektorientierte Programmiersprache *C#*, weswegen auch alle in *Unity* verwendeten Skripte der Objektorientierung folgen. Dies äußert sich unter anderem darin, dass zunächst jedes Skript von der Basisklasse **MonoBehaviour**[193] erbt und hierdurch bereits einige grundlegende Funktionen zur Verfügung gestellt bekommt. **MonoBehaviour**-Skripte können an **GameObjects** angehängt werden, wodurch hier für Nutzende leicht der Zusammenhang zwischen dem Konzept eines Objekts auf Seiten der Programmierung mit dem einzelnen Spielelement in *Unity* dargestellt wird. *Unity* stellt darüber hinaus mit der Klasse **ScriptableObject**[194] eine weitere Struktur zum Einsatz von objektorientierter Programmierung zur Verfügung, die hauptsächlich zur Speicherung von Daten genutzt werden kann.

Hieraus abgeleitet lassen sich jegliche Konzepte, die in *Unity* bezüglich der Objektorientierung erlernt und angewendet werden können, leicht auf andere objektorientierte Programmiersprachen und entsprechende Game Engines übertragen. Bezüglich der in dieser Arbeit evaluierten Game Engines trifft das auf *Flax*, *Unreal Engine*, *Cocos Creator*, *Godot*, *jMonkeyEngine*, *FUDGE* und *Vektoria* zu. Darüber hinaus lassen sich erlernte Kenntnisse bezüglich der Programmiersprache *C#* direkt auf *Flax* und *Godot* übertragen.

Schlüsselkonzepte

Zusammenhängende Spielabschnitte werden in *Unity* als **Scenes**[195] bezeichnet. Diese setzen sich – der Freie-Szenen-Struktur folgend – aus einzelnen Elementen, den sogenannten **GameObjects**[196] zusammen, die wiederum durch das Hinzufügen von **Components** unterschiedliche Eigenschaften und Funktionen erhalten. **GameObjects** sind in einer **Scene** in der Hierarchie unabhängig voneinander angeordnet, können darin aber auch in sogenannten **Parent-Child-Verbindungen** zusammengesetzt werden, wodurch

[193] vgl. Unity Technologies o. J.t
[194] vgl. Unity Technologies o. J.r
[195] vgl. Unity Technologies o. J.q
[196] vgl. Unity Technologies o. J.l

Position, Rotation und Skalierung des untergeordneten **(Child-)GameObjects** von der des übergeordneten **(Parent-)GameObjects** abhängig gemacht wird. **GameObjects**, die zur Wiederverwendung benutzbar sein sollen, können in sogenannte **Prefabs**[197] umgewandelt werden. Dabei werden Änderungen an dem ursprünglichen **Prefab** direkt auf alle Kopien beziehungsweise Instanzen dieses Elements übertragen. Änderungen an Kopien eines **Prefabs** werden erst auf das ursprüngliche und alle weiteren Kopien übertragen, wenn dies explizit im Editor bestätigt wird.

Die Freie-Szenen-Struktur aus frei zusammensetzbaren **Scenes** und unabhängigen **GameObjects** lässt sich auf viele andere Game Engines, die den gleichen Ansatz verfolgen, übertragen. Von den in dieser Arbeit evaluierten Game Engines trifft das auf *Flax*, *GameMaker*, *Unreal Engine* und *Cocos Creator* zu.

Meinungen von Lehrenden und Zusammenfassung

Unity ist in vier der befragten zehn Lehrkontexten fest in den Lehrplan eingebunden[198,199,200,201] und wird in allen weiteren Lehrkontexten von Studierenden für Spieleprojekte eingesetzt.[202,203,204,205,206,207] Die große Beliebtheit der Game Engine lässt sich einerseits durch ihre berufsvorbereitende Funktion[208] und andererseits durch ihre angenehme Lernkurve[209], die wiederum auf die große Community zurückzuführen ist[210], erklären. Alle diese Aspekte konnten auch in der vorangegangenen Evaluation bestätigt werden; es gibt eine große Anzahl an Lernmaterialien, die für unterschiedliche Lernsituationen geeignet sind und die Übertragbarkeit ist aufgrund der Objektorientierung und der in der Spieleerstellung vergleichsweise häufig verwendeten Programmiersprache *C#* sowohl in Richtung der Industrie als auch in Richtung anderer Game Engines (bezüglich der „drei Großen“ vor allem in Richtung *Godot*[211]) sehr gut. Gleichzeitig ist die Game Engine bezüglich der Anwendbarkeit gut für die Erstellung von sowohl 2D-, als auch 3D-Spielen geeignet. Abstriche muss die Game Engine vor allem im „Außenrum“ einstecken,

[197] vgl. Unity Technologies o. J.o
[198] vgl. I3 Z. 21
[199] vgl. I6 Z. 59
[200] vgl. I8 Z. 26 f.
[201] vgl. I9 Z. 5
[202] vgl. I0 Z. 38
[203] vgl. I1 Z. 45 f.
[204] vgl. I2 Z. 61 f.
[205] vgl. I4 Z. 14 ff.
[206] vgl. I5 Z. 26 ff.
[207] vgl. I7 Z. 89 f.
[208] vgl. I3 Z. 22 f.
[209] vgl. I8 Z. 50
[210] vgl. I9 Z. 50 ff.
[211] vgl. I6 Z. 145 – 153

beispielsweise vor allem durch die unübersichtliche Dokumentation einzelner Packages, die schlechte Verwendbarkeit mit Versionsverwaltungssoftware und nicht zuletzt auch durch die sehr schlechte Kommunikation mit Nutzenden im Rahmen der letzten Lizenzänderungen. Dass aufgrund dieser nicht mehr Nutzende in der Lehre auf eine andere Game Engine umgestiegen sind, lässt sich hauptsächlich daran festmachen, dass bei einer Änderung der im Lehrplan eingebundenen Game Engine zunächst das zugehörige Lehrpersonal diese Umstellung gut verinnerlichen müsste, damit eine sinnvolle Betreuung überhaupt möglich ist – und dies ist Lehrkontext-übergreifend nicht einfach so möglich.[212] Zusammengefasst ist *Unity* eigentlich sehr gut für jegliche Lehrkontexte geeignet die sich mit der Spieleerstellung beschäftigen – sowohl für komplett Neueinsteigende als auch für erfahrenere Lehr- beziehungsweise Lernumfelder – und momentan auch zurecht in diesen vertreten. Durch die aktuelle Firmenpolitik, die aber bereits viele langjährige Unterstützende vertrieben hat, bleibt abzusehen, wie attraktiv die Game Engine auch in Zukunft noch sein wird, wenn bereits jetzt schon andere Game Engines ähnlich gute Spieleprojekte hervorbringen und möglicherweise auch durch andere Aspekte überzeugen können.

[212] vgl. I9 Z. 41 – 48

4.1.4 Unreal Engine

Kriterium	Ergebnis
Definitionstreue	*Unreal Engine* ermöglicht die Erstellung von 2D- und 3D-Spielen mithilfe der Programmiersprache *C++* oder Unreal Blueprints.[213]
Aktualität	Version 5.3 wurde am 06. September 2023 veröffentlicht.[214] Auf Steam wurde zuletzt (19. Februar 2024) unter anderem *Palworld* veröffentlicht.[215]
Preisgestaltung	*Unreal Engine* ist kostenfrei bis zu Gesamtbruttoeinnahmen von mehr als 1.000.000$, dann fallen 5% Lizenzgebühren an (**Standard**-Lizenz). [216]
Entwicklungsplattformen	*Windows*, *MacOS*, *Linux*[217]
Zielplattformen	*Windows, MacOS, Linux, iOS, Android, iPadOS, tvOS, XR* und weitere[218]
Systemanforderungen	*Windows* 10 64-bit, Quad-Core von Intel oder AMD, Taktfrequenz von 2,5 GHz oder höher, 8 GB RAM *MacOS* Big Sur, Quad-Core von Intel, Taktfrequenz von 2,5 GHz oder höher, 8 GB RAM *Linux* Ubuntu 18.04, Quad-Core von Intel oder AMD, Taktfrequenz von 2,5 GHz oder höher, 32 GB RAM[219, 220]

Tabelle 8 Evaluation der Qualifikationskriterien - Unreal Engine

Erlernbarkeit

Offizielle Lernmaterialien

Unreal Engine bietet Neueinsteigenden eine große Zahl an Tutorials und Lernmaterialien an. Diese sind aufgeteilt in eine **Learning Library** und verschiedene Einführungen unter dem Namen **Getting Started**. Erstere lassen sich nach Typ, Inhalt und Sprache filtern, wobei von den über 1.300 aufgelisteten offiziellen Lernmaterialien aber alle nur in Englisch verfügbar sind. Dies trifft auch auf die Guides unter **Getting Started** zu. Bezüglich der

[213] vgl. Epic Games o. J.i
[214] vgl. Epic Games o. J.t
[215] vgl. SteamDB o. J.f
[216] Epic Games o. J.v
[217] Epic Games o. J.s
[218] vgl. Epic Games o. J.r
[219] siehe Epic Games o. J.s
[220] siehe Epic Games o. J.k

Verfügbarkeit für unterschiedliche Versionen der Game Engine wird es hier etwas unübersichtlich; während der **Course** (dt.: Kurs) namens **Introduction To Your First Hour in Unreal Engine 5.2** auf die genannte Version 5.2 ausgelegt ist, ist der **Learning Path** namens **BeginPlay()** auf Version 5.0 ausgelegt. Entsprechend ist es etwas umständlich, zunächst ein passendes Tutorial für die aktuell verwendete Version der Game Engine zu wählen. Bezüglich der Aktualität findet sich zu jedem Tutorial ein entsprechendes Veröffentlichungsdatum, wobei die meisten offiziellen Tutorials innerhalb der letzten zwei Jahre veröffentlicht wurden.

Um zu überprüfen, ob diese Umständlichkeit der Versionszuordnung möglicherweise entfällt, weil Tutorials auch für naheliegende Versionen benutzt werden können, wurde der erstgenannte Kurs **Introduction To Your First Hour In Unreal Engine 5.2** mit der Version 5.3.2 bezüglich seiner Reproduzierbarkeit evaluiert. Der Kurs bietet einen Überblick über die Erstellung von neuen Projekten mithilfe von **Templates** und der Erweiterung dieser durch verschiedene Elemente. Gleichzeitig wird auch eine Übersicht über die Verwendung von **Blueprints** gegeben.[221] Neben einem **Overview**-Kapitel besteht der Kurs aus acht weiteren Kapiteln, in denen wiederum ausschließlich Videos die einzelnen Schritte beleuchten. Eine Textversion oder ein Transkript existieren nicht, wodurch es schwer ist, einzelne Teile des Tutorials im Nachhinein noch einmal nachzuschlagen. Auch entsprechende Zeitstempel innerhalb der Videos gibt es nicht, was diesen Umstand erschwert. Das erste Video des Kurses erklärt, wie einzelne Versionen von *Unreal Engine* installiert werden können und wie der zugehörige **Marketplace** genutzt werden kann. Darüber hinaus wird verwirrenderweise erklärt, dass das Projekt **Online Learning Kit** aus dem **Marketplace** installiert werden soll, obwohl dieses im weiteren Verlauf des Kurses quasi nicht genutzt wird. Stattdessen wird im darauffolgenden Video erklärt, dass ein Projekt mit dem **Third Person Template** erstellt werden soll. Der Rest des Kurses erklärt darüber hinaus noch die Bewegung im und Benutzung des Editors sowie das Hinzufügen von weiteren Assets vom **Marketplace**. Dies wird konkret am Beispiel des erwähnten **Online Learning Kits** durchgeführt, das dann in das zuvor erstellt Projekt migriert wird – wobei dieser Schritt etwas zu fortgeschritten für den Kontext des Kurses wirkt. Anschließend wird noch die Erstellung eines eigenen Levels thematisiert, die durch eine sehr detaillierte Erklärung des Lighting-Systems abgeschlossen wird. Die darauffolgende Erklärung der **Blueprints** (wiederverwendbare Objekte, die über Code, einen **Event Graph** oder **Components** um Funktionen erweitert werden können) fällt dagegen vergleichsweise sehr kurz aus. Abschließend wird noch das **Packaging** des Projects demonstriert. All diese Schritte sind ohne Probleme reproduzierbar, dennoch kommt teilweise das Gefühl auf, dass

[221] vgl. Epic Online Learning/Wadstein 2023

die erklärten Thematiken recht komplex für jemanden sein könnten, der die Game Engine zum allerersten Mal geöffnet hat. Eine Möglichkeit zur Selbstbeurteilung (über die bloße eigenständige Betrachtung und Evaluation der erzielten Ergebnisse hinaus) existiert nicht und auch eine Fremdbeurteilung des Lernenden entfällt im Kontext der zur Verfügung gestellten offiziellen Lernmaterialien. Entsprechend ist auch das eigenständige Erlernen der Game Engine – nicht zuletzt auch durch die recht komplexen Themen innerhalb der Einstiegstutorials – erschwert, wodurch die *Unreal Engine* vermutlich eher für situiertes Lernen mit entsprechender Betreuung durch Lehrpersonal geeignet ist.

Neben den Tutorials, Kursen und anderweitigen Einstiegshilfen gibt es noch die Dokumentation, in der weitere Informationen über einzelne bestimmte Funktionen von *Unreal Engine* erhalten werden können. Darüber hinaus finden sich auf dem offiziellen *YouTube*-Kanal unter den über 2.600 Videos auch einige Tutorials zur Weiterbildung.

Inoffizielle Lernmaterialien

Auch außerhalb der offiziellen Lernmaterialien finden sich einige Tutorials und Hilfestellungen. Auf *YouTube* gibt es sogar einige Kanäle, die sich explizit mit *Unreal Engine* beschäftigen und Tutorials oder News hierzu bereitstellen – dazu gehören beispielsweise **RubaDev**[222] oder **UnrealSensei**.[223] Darüber hinaus zählt die Community auf *Reddit*, in der regelmäßig Tutorials geteilt werden, über 245.000 Mitglieder[224], wodurch Neueinsteigenden eine sehr große Menge an Hilfestellungen zur Verfügung steht. Eine Besonderheit, die erst bei genauerem Hinsehen auffällt, ist die Tatsache, dass zwischen **offiziellen Lernmaterialien** auf der im vorherigen Abschnitt thematisierten Seite von *Unreal Engine* auch Beiträge der Community eingestreut sind. Auch wenn es nicht näher erklärt wird, sind diese vermutlich entsprechend kuratiert und so finden auch deutschsprachige Neueinsteigende immerhin 56 Tutorials zu *Unreal Engine* in diesem Bereich.[225] Insgesamt stehen Lernenden also sehr viele verschiedene Ressourcen zur Verfügung.

[222] vgl. RubaDev o. J.
[223] vgl. Unreal Sensei o. J.
[224] vgl. /r/unrealengine o. J.
[225] vgl. Epic Games o. J.j

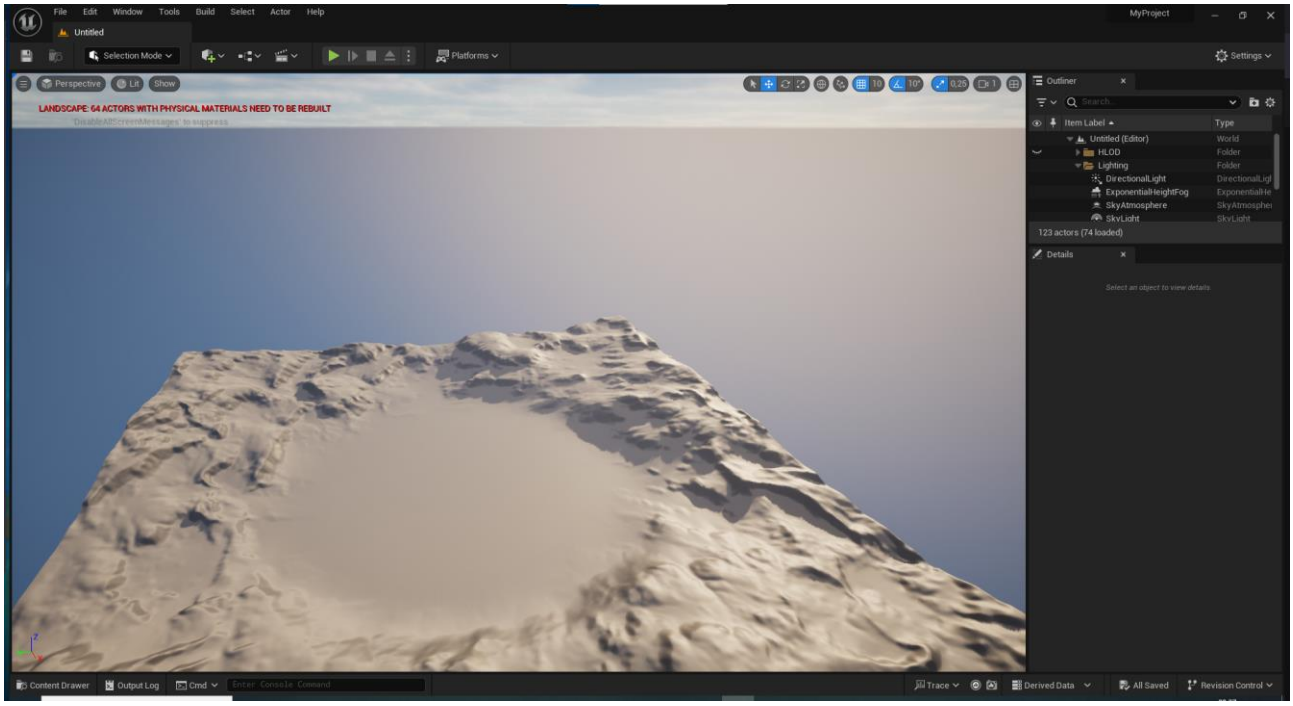

Abb. 8 Benutzeroberfläche Unreal Engine

Die Benutzeroberfläche von *Unreal Engine* wirkt beim ersten Öffnen recht aufgeräumt (siehe **Abb. 8**), wirft gleichzeitig aber ein paar Fragen auf. Zwar wird relativ schnell ersichtlich, dass das **Outliner**-Fenster alle Objekte der Spielszene anzeigt, im **Details**-Fenster weitere Informationen zu ausgewählten Objekten erhalten werden können und auch das große **Untitled**-Fenster lässt sich schnell als Arbeitsplatz für den aktuelle Spielabschnitt erkennen. Warum aber in einem als **Blank Project** (dt.: leeres Projekt) erstellten Projekt bereits eine Spielszene und mehrere Objekte im **Outliner**-Fenster existieren, wird nicht erklärt. Auch ist die Benutzeroberfläche zwar sehr aufgeräumt, lässt aber hierdurch etwas mehr Kontext vermissen. Wo beispielsweise Assets gefunden werden können, die zwar im Projekt aber noch nicht in der aktuellen Spielszene existieren, wird nicht schnell ersichtlich. Dagegen ist der **Play**-Knopf, mit dem die aktuelle Spielszene gestartet werden kann, direkt unter der Menüleiste sehr schnell erkennbar. Über die Menüleiste können weitere Fenster geöffnet und **Tools** angewendet werden. Einige der Unterpunkte sind hier zwar mit Icons zur besseren Erkennbarkeit gekennzeichnet, insgesamt verliert man aber trotzdem sehr schnell den Überblick über die Unmengen an Menüpunkten. Während die Benutzeroberfläche also auf den ersten Blick nicht überladen ist, kommt spätestens beim Versuch der Zurechtfindung in allen einzelnen Fenstern und Menüpunkten etwas Überforderung auf. Zusätzlich stiften die bereits aufgekommenen Fragen weitere Verwirrung, die durch eigenständiges Explorieren der Benutzeroberfläche nur bedingt geklärt werden.

Anwendbarkeit

Zusammensetzbarkeit

Unreal Engine besitzt keine leicht auffindbare vollständige Liste mit unterstützten Dateiformaten. Für 2D-Bilddateien und 3D-Model-Dateien muss ein Import für jedes Dateiformat entsprechend getestet werden, wobei die im Rahmen der Evaluation getesteten Dateiformate **png** und **fbx** ohne Probleme importiert werden können. Bezüglich Audio-Dateien werden recht viele gängige Formate unterstützt – mit Ausnahme von **mp3**.[226] Der Import von Dateien kann über das Fenster **Content Drawer** und den darin enthaltenen **Import**-Button ausgeführt werden. Alternativ ist auch ein Import per Drag-and-Drop möglich. 2D-Bilddateien werden beim Import automatisch in eine **Texture** umgewandelt, die im Editorfenster noch weiter bearbeitet werden kann. Für 3D-Model-Dateien öffnet sich beim Import ein eigenes Fenster, in dem recht viele Einstellungen vorgenommen werden können. Das Objekt wird bei Standardeinstellungen als **Static Mesh**, das zugehörige Material als **Material** getrennt importiert. Anschließend kann das Objekt direkt in einer Spielszene platziert und dort verwendet werden. In dieser wird dann eine entsprechende **StaticMeshComponent** angehängt, über die weitere Einstelllungen vorgenommen werden können. Einzelne oder mehrere Assets können zwischen einzelnen *Unreal Engine* Projekten mitsamt ihren Referenzen und Abhängigkeiten migriert werden, wodurch sie leicht in mehreren Projekten wiederverwendet werden können.[227]

Effizienz der Nutzung

Zu den Funktionen, die *Unreal Engine* ihren Nutzenden bietet, gehört die Erzeugung von UI-Systemen[228], die Implementierung von Licht-[229] und Audio-Elementen[230], die Erstellung von Animationen[231], die Erstellung von Terrain[232] und viele weitere. Darüber hinaus gibt es mit dem **Marketplace** eine große Auswahl an verschiedensten Assets, die direkt in *Unreal Engine* eingebunden werden können und der Service **Quixel Bridge** bietet hochauflösende 3D-Assets, die mithilfe eines eigenen Fensters direkt in der Game Engine verwendet werden können. Bezüglich der Versionsverwaltung bietet *Unreal Engine* die Unterstützung verschiedener Systeme, darunter neben *Git* auch *Subversion* oder *Perforce*. Die Nutzung dieser kann direkt im Editorfenster vorgenommen werden.[233] Gleichzeitig können hierdurch

[226] vgl. Epic Games o. J.l
[227] vgl. Epic Games o. J.p
[228] vgl. Epic Games o. J.f
[229] vgl. Epic Games o. J.n
[230] vgl. Epic Games o. J.c
[231] vgl. Epic Games o. J.b
[232] vgl. Epic Games o. J.g
[233] vgl. Muir 2022

mehrere Personen gleichzeitig eine Spielszene editieren[234], was die Zusammenarbeit an einem Projekt innerhalb eines Teams sehr vereinfacht.

Besonderheiten

Unreal Engine besitzt zwei große Besonderheiten, die an dieser Stelle kurz beleuchtet werden: Einerseits bietet die Game Engine eine unglaubliche Vielzahl an Möglichkeiten zur Erstellung von Spielen. Hierbei gibt es unter anderem teilweise sehr detaillierte Einstellungsmöglichkeiten zu einzelnen Objekten, unabhängig ihres Typs; *Unreal Engine* unterstützt nicht nur die Bearbeitung, sondern auch die tiefgreifende und detaillierte Aufbereitung von 2D-, 3D- und Audiodateien und weiterem direkt innerhalb der Game Engine. Andererseits zählen Spiele, die mit *Unreal Engine* erstellt werden, auch wegen der zugrundeliegenden Techniken wie beispielsweise *Niagara*[235] und *Lumen*[236] entsprechend häufig zu Vorzeigebeispielen von schönen, detaillierten und performanten Spielerfahrungen. Der Preis hierfür ist aber die entsprechend hohe Einstiegshürde in den Umgang mit der Game Engine.

Übertragbarkeit

Programmierparadigmen

Unreal Engine verwendet die Programmiersprache *C++* objektorientiert. Dies äußert sich unter anderem darin, dass jedes Script von unterschiedlichen Basisklassen erben kann und hierdurch bereits einige grundlegende Funktionen zur Verfügung gestellt bekommt – diese äußern sich je nach Basisklasse unterschiedlich.[237] Skripte können dann an die entsprechende Objektrepräsentation innerhalb einer Spielszene oder eines **Blueprints** angehängt werden, wodurch für Nutzende leicht der Zusammenhang zwischen dem Konzept eines Objekts auf Seiten der Programmierung mit dem einzelnen Spielelement in *Unreal Engine* dargestellt wird.

Hieraus abgeleitet lassen sich jegliche Konzepte, die in *Unreal Engine* bezüglich der Objektorientierung erlernt und angewendet werden können, leicht auf andere objektorientierte Programmiersprachen und entsprechende Game Engines übertragen. Bezüglich der in dieser Arbeit evaluierten Game Engine trifft das auf *Flax*, *Unity*, *Cocos Creator*, *Godot*, *jMonkeyEngine*, *FUDGE* und *Vektoria* zu. Darüber hinaus lassen sich

[234] vgl. Gomez 2024
[235] vgl. Epic Games o. J.h
[236] vgl. Epic Games o. J.o
[237] vgl. Doctor 2022

erlernte Kenntnisse bezüglich der Programmiersprache *C++* direkt auf *Flax* und *Vektoria* übertragen.

Schlüsselkonzepte

Zusammenhängende Spielabschnitte werden in *Unreal Engine* als **Level**[238] bezeichnet. Diese setzen sich – der Freie-Szenen-Struktur folgend – aus einzelnen Elementen zusammen, die in **Actors**[239] und **Pawns**[240] eingeteilt werden. Diese können wiederum mit **Components**[241] um unterschiedliche Funktionen erweitert werden. Spielobjekte sind in einem **Level** zunächst unabhängig voneinander angeordnet, können darin aber mit anderen Spielobjekten in sogenannten **Parent-Child-Verbindungen** zusammengesetzt werden, wodurch Position, Rotation und Skalierung des untergeordneten **(Child-)Spielobjekts** von der des übergeordneten **(Parent-)Spielobjekts** abhängig gemacht wird. Spielobjekte, die zur Wiederverwendung verfügbar sein sollen, können in sogenannten **Blueprints**[242] kombiniert werden. Dabei werden Änderungen an dem ursprünglichen **Blueprint** direkt auf alle Instanzen dieses Elements übertragen.

Die Freie-Szenen-Struktur aus frei zusammensetzbaren **Levels** und unabhängigen Spielobjekten lässt sich auf viele andere Game Engines übertragen, die den gleichen Ansatz verfolgen. Von den in dieser Arbeit evaluierten Game Engines trifft das auf *Flax*, *GameMaker*, *Unity* und *Cocos Creator* zu.

Meinungen von Lehrenden und Zusammenfassung

Unreal Engine ist in vier der befragten zehn Lehrkontexte fest in den Lehrplan eingebunden[243,244,245,246] und wird in allen weiteren Lehrkontexten von Studierenden für Spieleprojekte eingesetzt.[247,248,249,250,251,252] Die große Beliebtheit der Game Engine lässt sich an vielen Punkten festmachen; einerseits bieten die **Blueprints** einen einfachen Code-

[238] vgl. Epic Games o. J.m
[239] vgl. Epic Games o. J.a
[240] vgl. Epic Games o. J.q
[241] vgl. Epic Games o. J.e
[242] vgl. Epic Games o. J.d
[243] vgl. I1 Z. 59 ff.
[244] vgl. I2 Z. 22
[245] vgl. I3 Z. 21
[246] vgl. I9 Z. 26
[247] vgl. I0 Z. 39
[248] vgl. I4 Z. 14 ff.
[249] vgl. I5 Z. 26 ff.
[250] vgl. I6 Z. 62
[251] vgl. I7 Z. 89 f.
[252] vgl. I8 Z. 37

freien Einstieg in die Spieleerstellung[253, 254] und entsprechend können schnell erste Ergebnisse erzielt werden, die zur Weiterarbeit motivieren.[255,256] Andererseits ist die Game Engine gleichzeitig zukunftsfähig[257] und bietet entsprechende Berufschancen.[258] Dies ist hauptsächlich in der hohen Leistungsfähigkeit der Game Engine erklärt.[259] Vor allem dieser letzte Aspekt konnte auch in der vorangegangenen Evaluation bestätigt werden; überzeugend ist die *Unreal Engine* primär bezüglich ihrer sehr vielen Möglichkeiten in der Anwendung und ihrer sehr guten Übertragbarkeit auf andere Game Engines oder in das Berufsfeld. Die vielfältigen Anwendungsmöglichkeiten werden etwas dadurch eingeschränkt, dass eine gewisse Einarbeitung nötig ist, wobei hierfür aber recht viele Tutorials und Hilfestellungen für Neueinsteigende angeboten werden. Die teilweise recht komplexen Themen, die darin abgedeckt werden, sowie die Benutzeroberfläche der Game Engine, die schnell unübersichtlich werden kann, und die Programmiersprache *C++*, die eher schwer als erste Sprache erlernbar ist, trüben die ersten Schritte in der Game Engine aber etwas. Entsprechend ist die *Unreal Engine* wohl am besten für Lehrkontexte geeignet, die entweder eine gute informatiknahe Ausbildung gewährleisten können und ihre Studierenden in diesem Rahmen mit einem der leistungsfähigsten Tools ausbilden wollen oder Lehrkontexte, die über die Verwendung von **Blueprints** ihren Studierenden einen erleichterten Einblick in die recht komplexe Game Engine ermöglichen wollen. In beiden Fällen ist eine starke Betreuung durch entsprechendes Lehrpersonal sinnvoll.

[253] vgl. I1 Z. 97 f.
[254] vgl. I2 Z. 23 – 26
[255] vgl. I1 Z. 100
[256] vgl. I2 Z. 104 – 112
[257] vgl. I2 Z. 91 f.
[258] vgl. I3 Z. 57 ff.
[259] vgl. I1 Z. 105 ff.

4.2 Kostenfreie Engines

Unter dieser Kategorie sind alle Game Engines zusammengefasst, die komplett kostenfrei zur Spieleerstellung und zum Spielevertrieb einsetzbar sind. Gleichzeitig sind einige der Game Engines auch *Open Source*, wodurch sie frei anpassbar und erweiterbar sind.

4.2.1 Cocos Creator

Kriterium	**Ergebnis**
Definitionstreue	*Cocos Creator* ermöglicht die Erstellung von 2D- und 3D-Spielen mithilfe den Programmiersprachen TypeScript und *Java*Script.[260,261]
Aktualität	Version 3.8.2 wurde am 25. Januar 2024 veröffentlicht.[262] Auf Steam wurde zuletzt (18. Januar 2024) unter anderem *Mystery Society 2: Hidden Puzzles* veröffentlicht.[263]
Preisgestaltung	*Cocos Creator* ist kostenfrei nutzbar und *Open Source*.[264]
Entwicklungsplattformen	*Windows, MacOS*[265]
Zielplattformen	*Windows, MacOS, Web, iOS, Android* und weitere[266]
Systemanforderungen	Cocos Dashboard: *Windows* 7 64-bit oder neuer *MacOS* X 10.9 oder neuer[267]

Tabelle 9 Evaluation der Qualifikationskriterien - Cocos Creator

Erlernbarkeit

Offizielle Lernmaterialien

Zunächst ist zur Nutzung von *Cocos Creator* ein entsprechender **Cocos Developer Account** nötig. Nach Erstellung dessen kann dieser zur Anmeldung im **Cocos Dashboard** genutzt werden. Hier gibt es dann Möglichkeiten, eigene Projekte zu erstellen, unterschiedliche Versionen der Game Engine zu installieren und neben weiterem auch einen Tab für Tutorials. Das dort aufgelistete Tutorial namens **The Beginners Guide to Cocos Creator 3.x** mit einer angegebenen **Difficulty** (dt.: Schwierigkeit) von 1 war zum Zeitpunkt der Evaluation aber **Unavailable** (dt.: Nicht verfügbar). Entsprechend wurde für

[260] vgl. Cocos Creator o. J.a
[261] vgl. Cocos Creator o. J.m
[262] vgl. Cocos Creator o. J.q
[263] vgl. SteamDB o. J.e
[264] Cocos Creator o. J.a
[265] Cocos Creator o. J.r
[266] vgl. Cocos Creator o. J.i
[267] vgl. Cocos Creator o. J.k

die weitere Evaluation auf den **Beginner's Guide** in der Manual von *Cocos Creator* zurückgegriffen. Dieser ist zwar nur in Englisch, dafür aber in unterschiedlichen Versionen, angepasst an die jeweilige Version der Game Engine, verfügbar. Bezüglich der Aktualität dessen lassen sich aber keine Aussagen treffen, da ein entsprechendes Veröffentlichungsdatum oder ähnliches nicht einsehbar ist.

Zur Überprüfung der Reproduzierbarkeit wurde Version 3.1 des **Beginner's Guide**[268] und die entsprechende Version der Game Engine, die am 13. April 2021 veröffentlicht wurde[269], verwendet. Nach einem kurzen Überblick zu der Game Engine an sich und entsprechenden Workflows, sowie einer Erklärung zur Installation des Editors, werden erste Schritte mit der Game Engine anhand eines **Hello World Projects** gezeigt. Hier wird nach einer entsprechenden Erstellung des Projekts erklärt, wie **Scenes**, **Objects** und **Scripts** erstellt werden. Darüber hinaus wird ein kurzer Blick auf die Programmierung in *Cocos Creator* gelegt, bevor das **Script** dann als **Component** an das erzeugte **Object** angehängt wird. Anschließend wird erklärt, dass die erstellte **Scene** sowohl im Standardbrowser als auch in einem **Game View** sowie einem **Simulator** abgespielt werden kann – was genau die letzten beiden dieser Optionen bewirken, wird jedoch ausgelassen. Abschließend wird noch gezeigt, wie einzelne **Objects** innerhalb der **Scene** verändert werden können. Der **Beginner's Guide** ist insgesamt ohne Probleme reproduzierbar, bietet aber keine Fremdbeurteilung und auch die Selbstbeurteilung eines Lernenden ist lediglich durch die Betrachtung und eigenständige Evaluation der erzielten Ergebnisse möglich. Diese Einleitung ist dementsprechend zwar ausreichend für einen ersten Einstieg in die Game Engine im Rahmen von eigenständigem Lernen geeignet, wäre aber im Kontext von situiertem Lernen und unter der Betreuung einer entsprechenden Lehrperson effizienter.

Neben dem Einstiegstutorial bietet das Manual noch unterschiedliche Erklärungen zu den wichtigsten Funktionen von *Cocos Creator*. Das Vorhandensein dieser unterscheidet sich nur leider sehr stark zwischen einzelnen Versionen des Manuals; während die Manual-Version 3.1 beispielsweise noch einen **Basic Workflow** zur **Scene Creation** erklärt, fehlt dieser in der Manual-Version 3.8 vollständig, stattdessen findet sich hier eine **Editor Interface Introduction**, die in der Manual-Version 3.1 wiederum nicht vorhanden ist. Nutzende müssen sich also selbst die jeweils benötigten Erklärungen zusammensuchen, finden im Manual aber insgesamt recht viele Erklärungen zu der Game Engine. Abgesehen

[268] vgl. Cocos Creator o. J.f
[269] vgl. Cocos Creator o. J.q

hiervon gibt es zwar noch einen eigenen *YouTube*-Kanal, die Menge der Tutorials ist hier aber recht klein.[270]

Inoffizielle Lernmaterialien

Auch **inoffizielle Lernmaterialien** lassen sich nicht sonderlich viele finden. Auf *YouTube* gibt es vereinzelt kleine Kanäle, die sich mit wenigen Videos *Cocos Creator* widmen[271,272] , im Forum der Game Engine gibt es 134 Beiträge in der sogenannten **Knowledge Base**[273] und auf der Plattform *Reddit* sind 98 Mitglieder versammelt, die jedoch nicht sehr aktiv Tutorials teilen. [274] Insgesamt sind also wenige einzelne Hilfestellungen neben den **offiziellen Lernmaterialien** verfügbar, was den Einstieg in die Game Engine zusätzlich erschwert.

Heuristische Analyse der Benutzeroberfläche

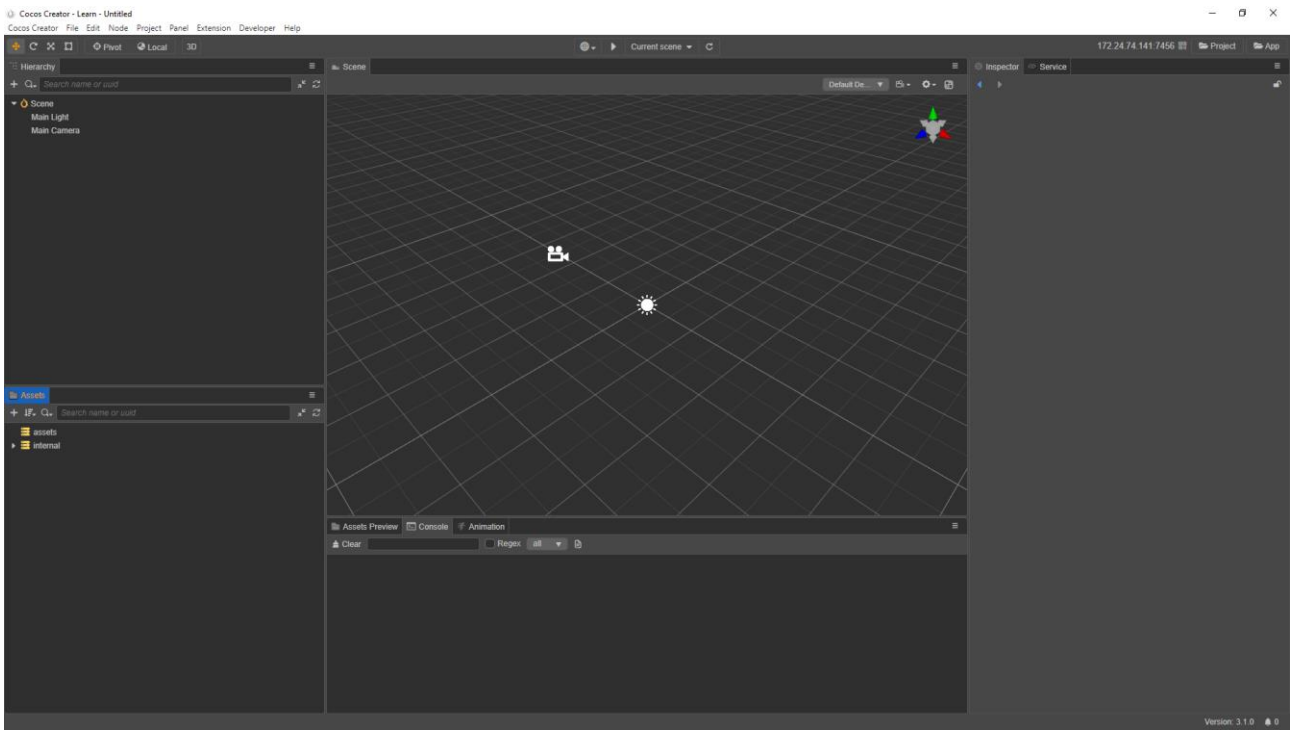

Abb. 9 Benutzeroberfläche Cocos Creator

Die Benutzeroberfläche von *Cocos Creator* ist beim ersten Öffnen recht schlicht gehalten und in einzelne Fenster aufgeteilt (siehe **Abb. 9**). Auch für Neueinsteigende wird relativ schnell ersichtlich, dass das **Assets**-Fenster zur Übersicht einzelner Dateien im Projekt

[270] vgl. Cocos o. J.a
[271] vgl. Cyan o. J.
[272] vgl. NitzzSea o. J.
[273] vgl. Cocos o. J.b
[274] vgl. /r/cocoscreator o. J.

dient und das **Scene**-Fenster den Inhalt der aktuell bearbeiteten Spielszene enthält. Durch Klicks auf die in der **SampleScene** bereits platzierten Objekte **Main Camera** und **Main Light** werden der entsprechende Eintrag im **Hierarchy**-Fenster geöffnet und die Details des Objekts im **Inspector**-Fenster angezeigt. Gleichzeitig ist für das ausgewählte Objekt das **Move Tool** aktiviert, wodurch Pfeile sichtbar werden mit denen man das Objekt verschieben kann. Jegliche Änderungen, die man dann an einzelnen Objekten im **Scene**-Fenster durchführt, werden auch im **Inspector**-Fenster angezeigt – und umgekehrt. Neue Spielobjekte können leicht über die **Create**-Icons (ein Plus-Symbol) im **Hierarchy**- oder **Assets**-Fenster erzeugt werden. Über den **Play**-Knopf in der oberen Mitte kann das aktuelle Spiel gestartet werden, wodurch sich automatisch der Standardbrowser öffnet. Insgesamt ist die Benutzeroberfläche von *Cocos Creator* sehr aufgeräumt und sinnvoll sortiert. Menüpunkte sind leicht nachvollziehbar benannt, sowie mit einem entsprechenden Icon zur besseren Erkennbarkeit und einem Shortcut zur schnelleren Benutzbarkeit gekennzeichnet. Der Ersteindruck ist also insgesamt sehr positiv.

Anwendbarkeit

Zusammensetzbarkeit

Cocos Creator unterstützt viele gängigen Dateiformate für Bild-[275] und Audio-Dateien.[276] Für 3D-Objekte werden nur die Dateiformate **fbx** und **glTF** unterstützt. Dateien können per Drag-and-Drop leicht importiert werden. Bilddateien bestehen nach dem Import aus dem importierten Bild sowie einem automatisch erzeugten Kind-Objekt, über das verschiedene Einstellungen für das Bild vorgenommen werden können.[277] 3D-Objekte dagegen setzen sich aus insgesamt drei Kind-Objekten zusammen; dem eigentlichen Objekt, das automatisch die Dateiendung **prefab** erhalten hat, dem Mesh des Objekts, das automatisch die Dateiendung **mesh** erhalten hat und das Material des Objekts, das automatisch die Dateiendung **material** erhalten hat. Letzteres ist aber auf den Modus **Read only** geschaltet und kann entsprechend nicht bearbeitet werden. Das Objekt kann direkt in die Spielszene eingebunden werden. In dieser besteht das Objekt dann nur noch aus einem Eltern-Objekt, an das lediglich das automatisch erzeugte **prefab** Objekt als Kind-Objekt angehängt ist. Letzteres ist im Vergleich zur Rotation in *Blender* unterschiedlich rotiert, was an der Ausrichtung der jeweiligen Koordinatensysteme liegt; während *Blender* ein rechtshändiges Koordinatensystem verwendet, bei dem die Z-Achse der Höhenanzeige dient, besitzt *Cocos Creator* ein rechtshändiges Koordinatensystem mit der Y-Achse als Höhenanzeige. Assets

[275] vgl. Cocos Creator o. J.j
[276] vgl. Cocos Creator o. J.h
[277] vgl. Cocos Creator o. J.j

können aus *Cocos Creator* entweder einzeln oder zusammengebündelt in Form von sogenannten **Asset Packages** exportiert werden. Eine Besonderheit ist hierbei, dass die einzelnen Assets als **zip** verpackt werden, wodurch sie auch leicht an anderer Stelle und nicht nur innerhalb eines anderen *Cocos Creator* Projekts wiederverwendet werden können.

Effizienz der Nutzung

Zu den Funktionen, die *Cocos Creator* Nutzenden bietet, gehört die Erzeugung von UI-Systemen[278], die Implementierung von Licht-[279] und Audio-Elementen[280], die Erstellung von Animationen[281], die Erstellung von Terrain[282] und viele weitere. Besonders praktisch ist hierbei der **Store**, der über das **Cocos Dashboard** geöffnet werden kann und über welchen Assets gekauft werden können. Bezüglich der Versionsverwaltung wird die Verwendung von *Git* unterstützt. Hierfür generiert *Cocos Creator* automatisch entsprechend benötigte *gitignore* Dateien bei der Erstellung eines Projekts. Darüber hinaus wird darauf hingewiesen, dass bei anderen Versionsverwaltungssystemen nur einige Dateien *commited* werden dürfen.[283] Gleichzeitig erzeugt *Cocos Creator* aber meta-Dateien für einzelne Assets, wodurch besondere Vorsicht diesbezüglich geboten ist, da meta-Dateien allgemein zwar verwendet werden, um die Objektverwaltung innerhalb der Game Engine zu verifizieren. Falls hierbei aber beispielsweise ein Branch-Switch in *Git* durchgeführt wird, ohne dass sämtliche meta-Dateien entsprechend *commited* wurden, kann es schnell zu größeren Problemen mit dem gesamten Spieleprojekt kommen.

Besonderheiten

Eine der Besonderheiten von *Cocos Creator* ist die Bereitstellung von drei unterschiedlichen Optionen zum Testen der erstellten Spiele. Neben dem Standardbrowser können Spiele auch in einem einzelnen Fenster sowie über den **Simulator** getestet werden. Letzterer ist leider schlecht dokumentiert, ermöglicht aber das Testen auf unterschiedlichen Geräten durch eine entsprechende Simulation dieser. Bei jeder dieser Optionen können einzelne Informationen über die Performanz des Spiels abgelesen werden. Darüber hinaus ermöglicht die Darstellung im Browser das Festlegen unterschiedlicher Darstellungsformate für gängige Smartphones, wodurch gerade das Testen von Mobile Games sehr vereinfacht wird und das Vorhandensein des tatsächlichen Geräts nicht nötig ist.

[278] vgl. Cocos Creator o. J.p
[279] vgl. Cocos Creator o. J.l
[280] vgl. Cocos Creator o. J.h
[281] vgl. Cocos Creator o. J.g
[282] vgl. Cocos Creator o. J.o
[283] vgl. Cocos Creator o. J.e

Übertragbarkeit

Programmierparadigmen

Cocos Creator verwendet die Programmiersprachen *JavaScript* und *TypeScript* im Zusammenhang mit der Objektorientierung. Dies äußert sich unter anderem darin, dass jedes Script zunächst die Basisklasse **Component** erweitert und hierdurch bereits einige grundlegende Funktionen zur Verfügung gestellt bekommt. Skripte können an **Nodes** angehängt werden, wodurch hier für Nutzende leicht der Zusammenhang zwischen dem Konzept eines Objekts auf Seiten der Programmierung mit dem einzelnen Spielelement in *Cocos Creator* dargestellt wird.

Hieraus abgeleitet lassen sich jegliche Konzepte, die in *Cocos Creator* bezüglich der Objektorientierung erlernt und angewendet werden können, leicht auf andere objektorientierte Programmiersprachen und entsprechende Game Engines übertragen. Bezüglich der in dieser Arbeit evaluierten Game Engines trifft das auf *Flax*, *Unity*, *Unreal Engine*, *Godot*, *jMonkeyEngine*, *FUDGE* und *Vektoria* zu. Lediglich der Umstieg von *JavaScript* oder *TypeScript* auf andere Programmiersprachen, wie *C#* oder *C++*, die in der Spieleerstellung eher verwendet werden, erschwert die Übertragbarkeit der in *Cocos Creator* erlernten Fähigkeiten etwas. Am ehesten lassen sich erlernte Kenntnisse der Programmiersprache *TypeScript* aber direkt auf *FUDGE* übertragen.

Schlüsselkonzepte

Zusammenhängende Spielabschnitte werden in *Cocos Creator* als **Scenes**[284] bezeichnet. Diese setzen sich – der Freie-Szenen-Struktur folgend – aus einzelnen Elementen, den sogenannten **Nodes** [285] zusammen. Jede **Node** kann mit **Components** [286] um unterschiedliche Funktionen erweitert werden. **Nodes** sind in einer Scene unabhängig voneinander angeordnet, können darin aber mit anderen **Nodes** in sogenannten **Parent-Child-Verbindungen** zusammengesetzt werden, wodurch Position, Rotation und Skalierung der untergeordneten **(Child-)Nodes** von der der übergeordneten **(Parent-) Nodes** abhängig gemacht wird. **Nodes**, die zur Wiederverwendung verwendbar sein sollen, können in sogenannte **Prefabs**[287] umgewandelt werden. Dabei werden Änderungen an dem ursprünglichen **Prefab** direkt auf alle Kopien dieses Elements übertragen. Änderungen an

[284] vgl. Cocos Creator o. J.n
[285] vgl. Cocos Creator o. J.b
[286] vgl. Cocos Creator o. J.c
[287] vgl. Cocos Creator o. J.d

Kopien eines **Prefabs** werden erst auf das ursprüngliche und alle weiteren Kopien übertragen, wenn dies explizit im Editor bestätigt wird.

Die Freie-Szenen-Struktur aus frei zusammensetzbaren **Scenes** durch **Nodes** lässt sich auf viele andere Game Engines, die den gleichen Ansatz verfolgen, übertragen. Von den in dieser Arbeit evaluierten Game Engines trifft das auf *Flax*, *GameMaker*, *Unity* und *Unreal Engine* zu.

Meinungen von Lehrenden und Zusammenfassung

Da *Cocos Creator* von keinem der befragten Dozierenden eingesetzt wird, entfällt an dieser Stelle die Beleuchtung der Meinungen von Lehrenden bezüglich der Game Engine. Auch wird die Game Engine in keinem der befragten Lehrkontext von Studierenden innerhalb entsprechender Projekte eingesetzt.

Die ersten Schritte im Umgang mit *Cocos Creator* fallen durch ausführliche Tutorials in diesem Bereich sehr leicht, unterschiedliche Versionen des Manuals und die geringe Verfügbarkeit von weiteren Lernmaterialien außerhalb dieser erschweren aber die Erlernung des weiteren Umgangs mit *Cocos Creator*. Unter dem Kriterium der Anwendbarkeit besitzt *Cocos Creator* einige sinnvolle Konzepte, die im Rahmen der Lehre für einen Allround-Einstieg und besonders für die Entwicklung für Mobilgeräte eingesetzt werden können. Die Übertragbarkeit der erlernten Konzepte mit *Cocos Creator* auf andere Game Engines ist recht einfach möglich – lediglich *JavaScript* beziehungsweise *TypeScript* als Programmiersprache sind in anderen Game Engines eher weniger zu finden und entsprechend schlecht übertragbar. *Cocos Creator* eignet sich deswegen also gut für Lehrkontexte, die eine weiterführende Unterstützung der Lernenden über die Grundsätze hinaus ermöglichen können, die Engine nur für die ersten Schritte in diesem Umfeld verwenden wollen oder einen besonderen Fokus auf die Spieleerstellung für Web- oder Mobile-Plattformen legen.

4.2.2 Defold

Kriterium	Ergebnis
Definitionstreue	*Defold* ermöglicht die Erstellung von 2D- und 3D-Spielen mithilfe der Programmiersprache *Lua*.[288]
Aktualität	Version 1.6.4 wurde am 13. Februar 2024 veröffentlicht.[289] Auf Steam wurde zuletzt (19. Februar 2024) unter anderem *Craftomation 101* veröffentlicht.[290]
Preisgestaltung	*Defold* ist kostenfrei nutzbar und *Open Source*.[291]
Entwicklungsplattformen	*Windows, MacOS, Linux*[292]
Zielplattformen	*Windows, MacOS, Linux, iOS, Android, Nintendo Switch, Playstation 4* und weitere[293]
Systemanforderungen	*Windows*: Vista oder neuer *MacOS*: 11 Big Sur oder neuer *Linux*: Ubuntu 18.04 oder neuer[294]

Tabelle 10 Evaluation der Qualifikationskriterien - Defold

Erlernbarkeit

Offizielle Lernmaterialien

Für *Defold* existieren eine Reihe an **offiziellen Lernmaterialien**, die sich den ersten Schritten in der Game Engine widmen und in Manuals (die sich dem Einstieg und einem ersten Überblick bezüglich des Umgangs mit *Defold* widmen) und Tutorials (die zu spezifischen Erklärungen detailliertere Anleitungen bereitstellen) aufgeteilt werden. Innerhalb der Manuals, die im Weiteren einer näheren Betrachtung unterliegen, wird ausgehend von einer kurzen Einleitung, der Installations- und Projekterstellungsprozess von *Defold* erklärt. In der Einleitung werden alle wichtigen Aspekte rund um *Defold* zusammengefasst und auf entsprechende Erklärungsseiten zum Editor, einfachen Beispielen, der Programmiersprache *Lua*, den Engine-eigenen **Building Blocks** und zum Forum weitergeleitet.[295] Während einzelne Tutorials zwar in unterschiedlichen Sprachen verfügbar sind (neben Englisch auch Spanisch, Französisch oder Ukrainisch), trifft das auf weitere Erklärungsseiten nicht zu. Gleichzeitig gibt es keine Möglichkeit einzelne Tutorials

[288] vgl. Defold Foundation o. J.a
[289] vgl. Westerdahl 2024
[290] vgl. SteamDB o. J.d
[291] vgl. Defold Foundation o. J.a
[292] vgl. Defold Foundation o. J.d
[293] vgl. Defold Foundation o. J.a
[294] vgl. Defold Foundation o. J.c
[295] vgl. Defold Foundation o. J.h

an eine bestimmte Version der Game Engine anzupassen. Auch bezüglich der Aktualität der Lernmaterialien lassen sich keine Aussagen treffen, da ein entsprechendes Veröffentlichungsdatum oder ähnliches nicht einsehbar ist.

Die tatsächliche Reproduzierbarkeit wurde anhand des **Editor overview** Tutorials und mit der Version 1.2.184, die am 29. Juni 2021 erschien[296], untersucht. Im Tutorial wird erklärt, wie der Editor gestartet wird, sowie ein kurzer Überblick über dessen einzelne Fenster gegeben. Anschließend wird erklärt, wie einzelne Spielobjekte ausgewählt, bewegt, rotiert und skaliert werden können. Dass erst nach diesem Schritt erklärt wird, wie neue Dateien (zu denen auch die Spieleobjekte gehören) überhaupt erstellt werden können, lässt den Aufbau dieses Tutorials etwas unüberlegt wirken. Insgesamt lassen sich die einzelnen Schritte zunächst auch nur teilweise reproduzieren, da zwar in entsprechenden Vergleichsbildern gezeigt wird, wie ein Spielobjekt, das in *Defold* als **Collection** bezeichnet wird, auf unterschiedliche Weisen manipuliert wird, jedoch nicht konkret erklärt wird, wie genau diese spezifische **Collection** erzeugt werden kann – hierfür müssen entsprechend weitere Tutorials zurate gezogen werden. Am Ende der **Editor overview** werden eventuelle Probleme und weitere Fragen durch das angehängte FAQ beantwortet. Eine Selbstbeurteilung ist für dieses Tutorial nur durch die Betrachtung der tatsächlich erzeugten Ergebnisse möglich, jegliche Fremdbeurteilung entfällt im Kontext des eigenständigen Lernens mithilfe dieses Tutorials. Hierdurch wäre ein Einstieg in die Game Engine im Kontext von situiertem Lernen und unter entsprechender Betreuung durch Lehrpersonal deutlich einfacher zu handhaben. Insgesamt ist das Tutorial zwar ein guter erster Überblick über die Game Engine, lässt aber an entscheidenden Stellen Details vermissen, die dazu führen, dass weitere Erklärungen in weiteren Tutorials der Manuals gesucht werden müssen.

Inoffizielle Lernmaterialien

Bezüglich konkreter Lernmaterialien finden sich vor allem auf *YouTube* ein paar Kanäle und Videos, die sich mit Tutorials für *Defold* beschäftigen. Neben dem offiziellen *Defold YouTube*-Kanal, auf dem sich nur wenige der 100 Videos der tatsächlichen Erklärung der Spieleentwicklung mit *Defold* widmen, gibt es die Kanäle **Defold Tutorials** [297] und **Unfolding Gamedev**[298]. Während auf ersterem nur 14 Videos existieren, wovon das letzte vor drei Jahren veröffentlicht wurde, glänzt letzterer durch konkrete Tutorials zu einzelnen Konzepten der Game Engine. Darüber hinaus soll an dieser Stelle auch das Video **Defold**

[296] Westerdahl 2021
[297] vgl. Defold Tutorials o. J.
[298] vgl. Unfolding Gamedev o. J.

for Unity Developers von **Gamefromscratch** hervorgehoben werden, das *Defold* im Vergleich zu *Unity* erklärt und dabei Gemeinsamkeiten, Unterschiede und konkrete Schritte zur Übertragung von *Unity*-Kenntnissen auf *Defold* hervorhebt.[299] Abschließend gibt es auch eine *Reddit*-Community, in der sich knapp 1.500 Mitglieder über Fragen zu *Defold* regemäßig austauschen[300] und ein Forum zu *Defold*, dessen **Questions**-Teil über 7.500 Beiträge umfasst.[301] Insgesamt steht Lernenden also eine recht überschaubare Menge an Ressourcen für unterschiedliche Kompetenzstufen und Kontexte zur Verfügung.

Heuristische Analyse der Benutzeroberfläche

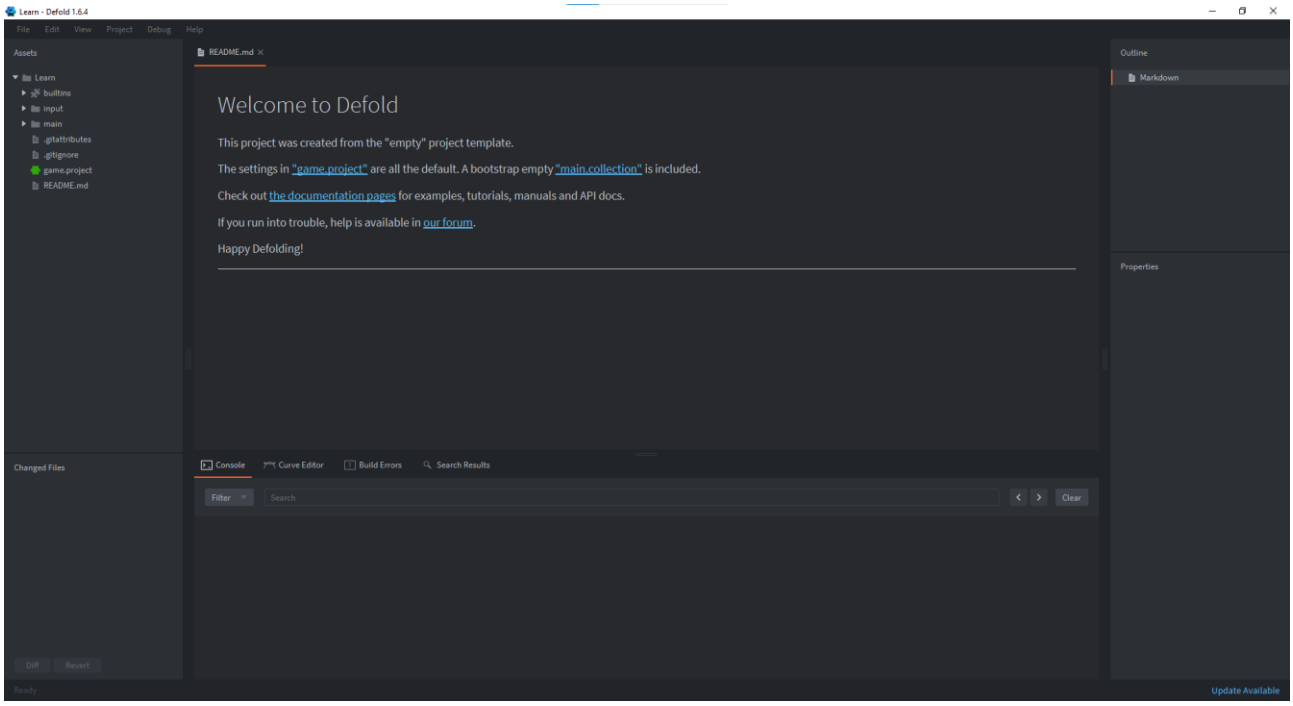

Abb. 10 Benutzeroberfläche Defold

Die Benutzeroberfläche von *Defold* ist beim ersten Öffnen recht schlicht gehalten und in einzelne Fenster aufgeteilt (siehe **Abb. 10**). Auch für Neueinsteigende wird relativ schnell ersichtlich, dass im **Asset**-Fenster alle Dateien des aktuellen Projekts aufgelistet sind, diese in der Mitte des Bildschirms angezeigt werden können und Details dazu im **Outline**-Fenster auf der linken Seite eingesehen werden können. Weitere Dateien können leicht über das **Asset**-Fenster geöffnet werden. Das Spiel kann (leicht versteckt) über den Menüpunkt **Debug → Start/Attach** gestartet werden. Für jegliche Menüunterpunkte wird, falls vorhanden, ein entsprechender Shortcut angezeigt. Spielobjekte können über das **Outline**-

[299] vgl. Gamefromscratch 2023
[300] vgl. /r/defold o. J.
[301] vgl. Defold Foundation o. J.i

Fenster oder im **Asset**-Fenster erstellt werden, wobei jede erstellbare Datei mit einem entsprechenden Icon gekennzeichnet ist – Detailerklärungen fehlen aber und die Liste ist insgesamt recht lang und unübersichtlich. Hier wäre eine Kategorisierung mit Unterpunkten oder ähnliches hilfreich. Skripte können direkt im Editor der Game Engine geöffnet werden, wodurch kein zusätzlicher Codeeditor benötigt wird. Für Neueinsteigende stellt sich hier aber die Frage, welche der neu erstellbaren Dateien überhaupt zur Programmierung genutzt werden kann. Hier existieren nämlich die Optionen **Gui Script**, ***Lua* Module**, **Render Script** und **Script**, die auch alle mit dem gleichen Icon gekennzeichnet sind. Insgesamt wirkt die Benutzeroberfläche bezüglich dieses Ersteindrucks recht übersichtlich und ordentlich, an manchen Stellen aber zu kurz erklärt, wodurch auf die entsprechende Dokumentation zurückgegriffen werden muss.

Anwendbarkeit

Zusammensetzbarkeit

Defold besitzt keine vollständige Auflistung aller unterstützten Dateiformate, unterstützt für 2D-Bilddateien aber **png** und **jpeg**[302] und für 3D-Modell-Dateien **glTF** und **dae**.[303] Durch die fehlende Dokumentation weiterer unterstützter Dateiformate müssen für jede Art von Assets zunächst entsprechende Tests durchgeführt werden. Der Import von Assets kann leicht per Drag-and-Drop vorgenommen werden und wurde aufgrund der fehlenden Unterstützung von **fbx**-Dateien testweise nur mit der **png**-Datei durchgeführt. Diese muss nach dem Import in ein sogenanntes **Defold asset** überführt werden, konkret ist hier für einzelne Bilder sowie Spritesheets der **Atlas** geeignet. Dieser muss erstellt werden, dann das gewünschte Bild verknüpft werden und erst dann kann das importierte Bild durch diesen **Atlas** in einem Spielobjekt referenziert werden. Dieser Umweg erschwert die Einbindung von Dateien in *Defold* im Vergleich zu anderen in dieser Arbeit untersuchten Game Engines. Darüber hinaus gibt es in *Defold* auch keine Möglichkeiten zum Export einzelner Spielobjekte zur gezielten Wiederverwendung in anderen Projekten.

Effizienz der Nutzung

Neben Standardfunktionen bietet *Defold* zusätzlich die Möglichkeit zur Erstellung von Partikeleffekten, tiefgreifenden Animationen einzelner Spielobjekte, Native Extensions[304], die die Game Engine um weitere Funktionen erweitern und die Möglichkeit zum Live-Update einzelner Assets. Hierbei können während der Laufzeit des Spiels Assets geladen und

[302] vgl. Defold Foundation o. J.f
[303] vgl. Defold Foundation o. J.g
[304] vgl. Defold Foundation o. J.n

verwendet werden, die bei der Erstellung des ursprünglichen Builds nicht miteingebunden wurden.[305] Darüber hinaus integriert *Defold* direkt *Git* zur Versionsverwaltung von Projekten.[306] Sobald ein Projekt mit einer entsprechenden *Repository* verbunden wurde, werden Änderungen an einzelnen Dateien direkt im Editor im **Changed Files**-Fenster angezeigt. Eine Synchronisierung mit der *Repository* kann über den Menüpunkt **File → Synchronize** ebenfalls innerhalb des Editors durchgeführt werden. Hierbei können **Remote changes** (dt. etwa: Fernänderungen) geladen, Konflikte gelöst sowie Dateien *committed* werden, was die Zusammenarbeit an einem *Defold*-Projekt innerhalb eines Teams sehr vereinfacht.

Besonderheiten

Defold besitzt trotz der vergleichsweise geringen Popularität der Game Engine und der kleinen Community eine Vielzahl an Assets, die alle kostenlos über das **Defold Asset Portal** verfügbar sind.[307] Besonders hervorgehoben werden können an dieser Stelle die **Steamworks** Integration zur Implementierung verschiedener Funktionen der Spieleplattform *Steam*, die **Google AdMob** Integration zur Arbeit mit Werbeanzeigen innerhalb eines Spiels und die **Ink** Integration, die die **narrative scripting language** (dt.: narrative Skriptsprache) **ink** in Verbindung mit *Defold* nutzbar macht.

Übertragbarkeit

Programmierparadigmen

Defold verwendet die Programmiersprache *Lua*, die zwar Objektorientierung und andere Programmierparadigmen grundsätzlich unterstützt, im Rahmen von *Defold* aber rein imperativ verwendet wird, wodurch auch jegliche in diesem Kontext erstellten Skripte der imperativen Programmierung unterliegen.

Entsprechend lassen sich damit erlernte Fähigkeiten eher schlecht auf andere in dieser Arbeit evaluierte Game Engines übertragen, da die meisten sich auf die Objektorientierung stützen. Am ehesten kann hier für eine Übertragbarkeit auf *GameMaker* argumentiert werden, da diese ebenfalls auf die imperative Programmierung setzt.

Schlüsselkonzepte

Einzelne Spielobjekte werden in *Defold* **GameObjects** genannt. Diese können mit **Components** um verschiedene Funktionen erweitert werden und in sogenannten

[305] vgl. Defold Foundation o. J.j
[306] vgl. Defold Foundation o. J.m
[307] vgl. Defold Foundation o. J.b

Collections strukturiert werden. Eine **Collection** ist dabei in Form einer Baumstruktur aufgebaut und folgt entsprechend der Wurzel-Knoten-Struktur. Das besondere hierbei ist, dass **Collections** auch mit anderen **Collections** verknüpft werden können. Während zusammenhängende Spielabschnitte in anderen Game Engines also in unabhängigen **Szenen**, **Levels** oder **Maps** voneinander getrennt sind, können in *Defold* durch die Verknüpfung mehrere **Collections** leichter einzelne häufig verwendete Spielobjekte und -abschnitte wiederverwendet werden.[308] Sowohl **Collection**-, **GameObject**-, als auch **Component**-Dateien werden im Kontext von *Defold* als **Prototype** bezeichnet und können an beliebiger Stelle instanziiert und somit überall wiederverwendet werden.[309] Wenn die ursprüngliche **Prototype**-Datei verändert wird, wirkt sich das entsprechend auch auf alle Instanzen dieser aus. Die jeweilige Basisdatei wird dabei stets im **Outline**-Fenster angezeigt, wodurch die Arbeit mit **Prototypes** übersichtlich bleibt.

Der Ansatz des Aufbaus zusammenhängender Spielabschnitte, die der Wurzel-Knoten-Struktur folgen, lässt sich leicht auf andere Game Engines übertragen, die den gleichen Ansatz verfolgen. Von den in dieser Arbeit evaluierten Game Engines trifft das auf *Godot*, *jMonkeyEngine*, *FUDGE* und *Vektoria* zu.

Meinungen von Lehrenden und Zusammenfassung

Da *Defold* von keinem der befragten Dozierenden eingesetzt wird, entfällt an dieser Stelle die Beleuchtung der Meinungen von Lehrenden bezüglich der Game Engine. Auch wird die Game Engine in keinem der befragten Lehrkontext von Studierenden innerhalb entsprechender Projekte eingesetzt.

Zusammenfassend ist *Defold* als Game Engine mit einigem Potenzial, aber mit entsprechend großen Einschränkungen beschreibbar. Die Erlernung der Game Engine ist durch die bereitgestellten Manuals zwar grundsätzlich gut möglich, insgesamt aber recht oberflächlich gehalten und aufgrund fehlender zugehöriger Beurteilungen eher schlecht für das eigenständige Lernen geeignet. Unter dem Kriterium der Anwendbarkeit besitzt *Defold* aber einige sehr sinnvolle Konzepte, die im Rahmen der Lehre für einen Allround-Einstieg eingesetzt werden können. Besonders sprechen hierfür die integrierte Programmierschnittstelle und Versionsverwaltung. Die Übertragbarkeit der erlernten Konzepte mit *Defold* auf andere Game Engines ist bezüglich der Programmiersprache etwas erschwert. *Defold* ist also besonders für Lehrkontexte geeignet, die eine beständige Unterstützung der Lernenden für den Einstieg in die Verwendung der Game Engine

[308] vgl. Defold Foundation o. J.k
[309] vgl. Defold Foundation o. J.l

gewährleisten können und gleichzeitig einen großen Wert darauf legen, kleine Teams mit geeigneten Tools an die Spieleerstellung heranzuführen.

4.2.3 Godot

Kriterium	**Ergebnis**
Definitionstreue	*Godot* ermöglicht die Erstellung von 2D- und 3D-Spielen mithilfe der Programmiersprachen *GDScript* oder *C#*.[310]
Aktualität	Version 4.2.1 wurde am 12. Dezember 2023 veröffentlicht.[311] Auf Steam wurde zuletzt (16. Februar 2024) unter anderem *20 Small Mazes* veröffentlicht.[312]
Preisgestaltung	*Godot* ist kostenfrei nutzbar und *Open Source*.[313]
Entwicklungsplattformen	*Windows, MacOS, Linux*[314]
Zielplattformen	*Windows, MacOS, Linux, iOS, Android, Web*[315]
Systemanforderungen	OpenGL 3.3 oder OpenGL ES 3.0 kompatible Hardware[316]

Tabelle 11 Evaluation der Qualifikationskriterien - Godot

Erlernbarkeit

Offizielle Lernmaterialien

Godot bietet mit **Godot Docs** eine Übersicht über die Game Engine und verschiedene Tutorials und Anleitungen zu einzelnen Aspekten dieser an.[317] Hierzu gehören neben der Rubrik **Getting Started**[318], in der die Game Engine als Ganzes vorgestellt wird und dann sowohl die Erstellung eines 2D-, als auch eines 3D-Spiels erklärt wird, auch die Rubrik Manual[319], in der weiterführende Konzepte von *Godot* erklärt werden. **Godot Docs** ist in der Gesamtheit in mehreren Sprachen verfügbar, darunter neben Englisch auch Deutsch, Spanisch, Französisch und weitere. Gleichzeitig können alle Erklärungen jeweils an eine bestimmte Version der Game Engine angepasst werden, wobei hier neben der letzten Veröffentlichung und der letzten LTS-Veröffentlichung auch frühere Versionen – bis zurück zur Version 2.1 – ausgewählt werden können.

[310] vgl. Godot o. J.p
[311] vgl. Godot o. J.g
[312] vgl. SteamDB o. J.c
[313] vgl. Godot o. J.h
[314] vgl. Godot o. J.g
[315] vgl. Godot o. J.f
[316] vgl. Godot o. J.g
[317] vgl. Godot o. J.j
[318] vgl. Godot o. J.o
[319] vgl. Godot o. J.e

Die tatsächliche Reproduzierbarkeit wurde anhand des Tutorials **Your first 2D game** mit der Version 4.2.1 untersucht.[320] Hierin wird erklärt, wie ein Projekt in *Godot* erstellt werden kann, wie darin einzelne Spielobjekte erstellt werden können, wie diese mit Logik zu einem Spiel zusammengesetzt werden können und wie das Spiel um ein **Heads-up-Display** erweitert werden kann. Einzelne Schritte des Tutorials sind gut bebildert und entsprechend leicht umsetzbar. Code, der geschrieben werden soll, ist jeweils in sowohl *GDScript*, als auch *C#* angegeben. Darüber hinaus sind in manchen Teilen des Tutorials auch Erklärungen und weiterleitende Links zu einzelnen angesprochenen Themen vorhanden. Gleichzeitig gibt es auf jeder Seite des Tutorials den Hinweis, dass die Seite aktuell gehalten wird und Nutzende werden gebeten, bei auffallender veralteter Information einen entsprechenden *Issue* über *GitHub* zu öffnen. Während der durchgeführten Evaluation war dies nicht nötig, weil das Tutorial ohne Probleme vollständig reproduzierbar war. Somit ist der Einstieg in *Godot* gut für eigenständiges Lernen geeignet. Lediglich die Selbstbeurteilung des eigenen Lernfortschritts ist nur durch die Betrachtung und eigenständige Evaluation der erzielten Ergebnisse möglich. Hier würde also im Kontext des situierten Lernens eine anleitende Lehrperson, die eine entsprechende Fremdbeurteilung ermöglicht, den Lernprozess sinnvoll ergänzen.

Abseits diverser Tutorials gibt es in den **Godot Docs** auch Weiterleitungen zur **Asset Library**, dem Forum, einzelnen Chats und anderen digitalen Orten, an denen weitere Hilfestellungen bezüglich der Game Engine gefunden werden können. Hierzu zählt auch der offizielle *YouTube*-Kanal, auf dem ein Großteil der über 110 Videos Aufnahmen von Vorträgen zu verschiedenen Themen rund um die Game Engine sind.[321]

Inoffizielle Lernmaterialien

Auch außerhalb der **offiziellen Lernmaterialien** finden sich einige Tutorials und Hilfestellungen. Auf *YouTube* gibt es sogar einige Kanäle, die sich explizit mit *Godot* beschäftigen und Tutorials hierzu bereitstellen – hierzu gehören beispielsweise **GDQuest**[322] oder **StayAtHomeDev**.[323] Abseits hiervon zählt die Community auf der Plattform *Reddit*, in der regelmäßig Tutorials geteilt werden, über 175.000 Mitglieder.[324] Insgesamt stehen Lernenden also sehr viele verschiedene und leicht benutzbare Ressourcen für unterschiedliche Kompetenzstufen und Kontexte zur Verfügung.

[320] vgl. Godot o. J.u
[321] vgl. Godot Engine o. J.
[322] vgl. GDQuest o. J.
[323] vgl. StayAtHomeDev o. J.
[324] vgl. /r/godot o. J.

Heuristische Analyse der Benutzeroberfläche

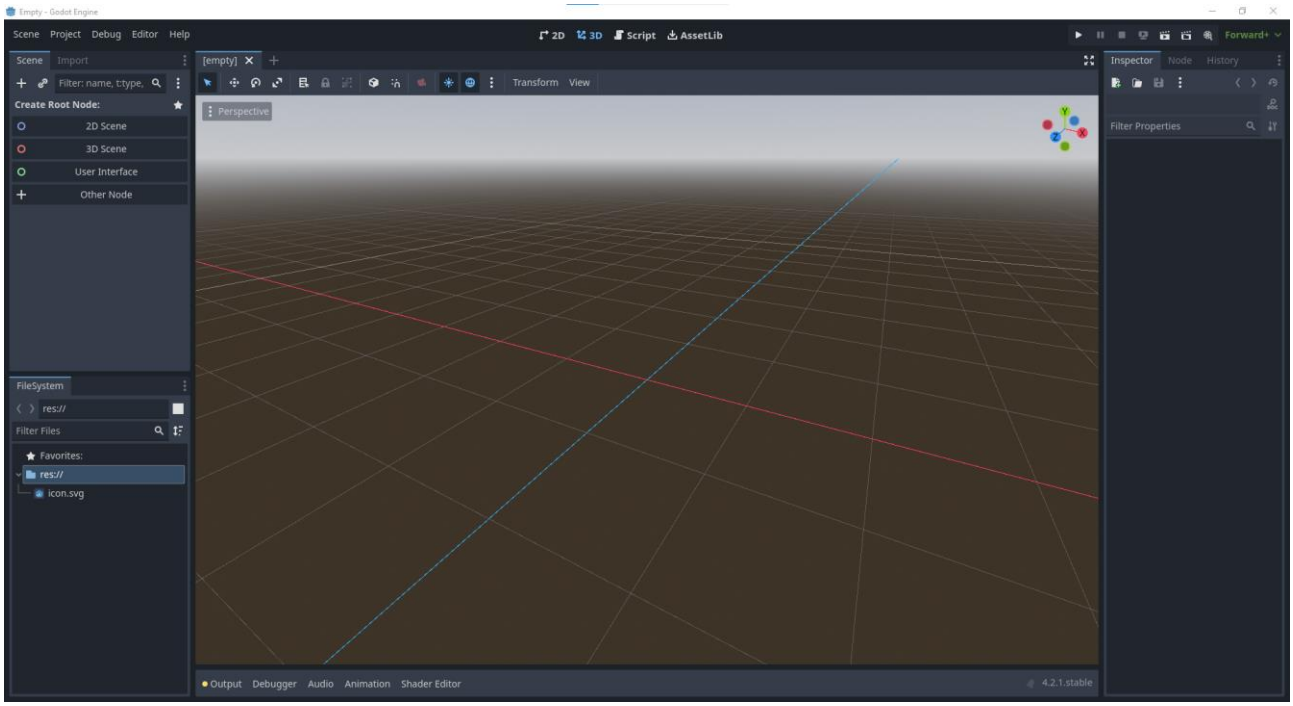

Abb. 11 Benutzeroberfläche Godot

Die Benutzeroberfläche von *Godot* ist beim ersten Öffnen recht schlicht gehalten und in einzelne Fenster aufgeteilt (siehe **Abb. 11**). Auch für Neueinsteigende wird relativ schnell ersichtlich, dass das **FileSystem**-Fenster zur Übersicht einzelner Dateien im Projekt dient und das mittlere Hauptfenster den Inhalt der aktuell bearbeiteten Spielszene enthält. Eine weitere Repräsentation dieser Spielszene befindet sich entsprechend im **Scene**-Fenster. Darüber hinaus wird auch schnell ersichtlich, dass nähere Informationen zu einzelnen Spielobjekten nach einer entsprechenden Auswahl dem **Inspector**-Fenster entnommen werden können. In der oberen Bildschirmmitte kann leicht zwischen den vier verschiedenen Ansichten **2D** (die die 2D-Version der aktuellen Spielszene zeigt), **3D** (die die 3D-Version der aktuellen Spielszene zeigt und zum Start des Editorfensters standardmäßig ausgewählt ist), **Script** (die den integrierten Script-Editor zeigt) und **AssetLib** (die den **Asset Store** öffnet) umgeschaltet werden. Durch die Menüleiste können unter anderem weitere Einstellungen bezüglich des Projekts oder Editorfensters vorgenommen werden. Über den **Play**-Knopf in der oberen rechten Ecke kann das aktuelle Spiel gestartet werden, wodurch sich automatisch ein eigenständiges Fenster öffnet. Insgesamt ist die Benutzeroberfläche von *Godot* sehr aufgeräumt und sinnvoll sortiert. Beim Verweilen des Mauszeigers über einzelne Knöpfe des Hauptfensters werden kurze Erklärung der einzelnen Funktionen gegeben. Der Ersteindruck bezüglich der Benutzerfreundlichkeit ist also insgesamt sehr positiv.

Anwendbarkeit

Zusammensetzbarkeit

Godot unterstützt eine Vielzahl an gängigen Dateiformaten für 2D-[325], Audio-[326] und Text-Dateien.[327] Bezüglich 3D-Modell-Dateien unterstützt Godot zwar neben **glTF 2.0** auch ältere Dateiformate wie **dae**, **fbx** wird aber nur durch eine zugehörige Integration unterstützt, die zusätzlich zur Game Engine installiert werden muss.[328] Entsprechend entfällt im Folgenden die Evaluation des 3D-Imports. Der Import von Assets kann generell aber leicht per Drag-and-Drop vorgenommen werden. Anschließend können für 2D-Bilddateien unterschiedliche Einstellungen im **Import**-Fenster vorgenommen werden. Darüber hinaus gibt es keine Möglichkeit, einzelne oder mehrere Assets aus *Godot* über das Editorfenster zu exportieren, um sie in anderen Projekten wiederverwenden zu können.

Effizienz der Nutzung

Neben Standardfunktionen einer Game Engine bietet *Godot* zusätzlich die Möglichkeit zur Erstellung von UI-Elementen[329], Animationen einzelner Spielobjekte[330], das Hinzufügen von Audio[331], der einfachen Lokalisierung von Spielen[332] und weiterem. Darüber hinaus gibt es ein offizielles Plugin zur Versionsverwaltung mit *Git* für *Godot*. Gleichzeitig kann bei der Erstellung eines Projekts eingestellt werden, dass eine passende *gitignore* und eine *gitattributes* Datei direkt miterstellt wird. Über das Plugin zur Versionsverwaltung können dann direkt im Editorfenster Dateien mit einer Repository synchronisiert werden[333], was die Zusammenarbeit an einem *Godot*-Projekt innerhalb eines Teams sehr vereinfacht.

Besonderheiten

Die größte Besonderheit von *Godot* liegt in der leichten aber gleichzeitig vielseitigen Benutzbarkeit der Game Engine. Die Integration eines Code-Editors im Editorfenster und die recht einfach erlern- und einsetzbare Programmiersprache *GDScript* erlaubt auch Neueinsteigenden, einen Großteil der Handgriffe für eine Spieleprojekt direkt im Editor durchführen zu können, wodurch mögliche Ablenkungen von vorneherein ausgeschlossen werden. Auch der starke Fokus auf das **Node**-System von *Godot* macht die Game Engine

[325] vgl. Godot o. J.l
[326] vgl. Godot o. J.k
[327] vgl. Godot o. J.m
[328] vgl. Godot o. J.d
[329] vgl. Godot o. J.s
[330] vgl. Godot o. J.a
[331] vgl. Godot o. J.c
[332] vgl. Godot o. J.n
[333] vgl. Godot o. J.t

sehr versatil; Dadurch, dass jede einzelne **Node** als entsprechende Spielszene aufgesetzt und getestet werden kann, ist es sehr einfach, große Szenen zusammenzusetzen, indem sich zunächst um einzelne kleine Aspekte dieser gekümmert wird und diese individuell getestet werden.

Übertragbarkeit

Programmierparadigmen

Godot verwendet die imperative und objektorientierte Programmiersprache *GDScript*[334], die im Zusammenhang mit der Game Engine auch sinnvoll objektorientiert verwendet werden kann.[335] Darüber hinaus unterstützt *Godot* auch die objektorientierte Programmierspache *C#*, wodurch ebenfalls Konzepte der Objektorientierung in der Game Engine verwendet werden können. Dies äußert sich beispielsweise darin, dass Skripte die Klasse **Node** erweitern können und hierdurch bereits einige grundlegende Funktionen zur Verfügung gestellt bekommen. Darüber hinaus können Skripte dann an einzelne Spielobjekte angehängt werden, wodurch hier für Nutzende leicht der Zusammenhang zwischen dem Konzept eines Objekts auf Seiten der Programmierung mit dem einzelnen Spielelement in *Godot* dargestellt wird.

Hieraus abgeleitet lassen sich jegliche Konzepte, die in *Godot* bezüglich der Objektorientierung erlernt und angewendet werden können, leicht auf andere objektorientierte Programmiersprachen und entsprechende Game Engines übertragen. Bezüglich der in dieser Arbeit evaluierten Game Engines trifft das auf *Flax*, *Unity*, *Unreal Engine*, *Cocos Creator*, *jMonkeyEngine*, *FUDGE* und *Vektoria* zu. Darüber hinaus lassen sich erlernte Kenntnisse bezüglich der Programmiersprache *C#* direkt auf *Flax* und *Unity* übertragen.

Schlüsselkonzepte

Zusammenhängende Spielabschnitte werden in *Godot* als **Scenes**[336] bezeichnet. Diese setzen sich – der Wurzel-Knoten-Struktur folgend – aus einzelnen Elementen, den sogenannten **Nodes**[337] zusammen. Jede **Node** besitzt selbst jeweils eine bestimmte Funktion und andere **Nodes** können angehängt werden, um ein Spielobjekt um weitere Funktionen zu erweitern. In einer **Scene** können **Nodes** aber auch unabhängig voneinander angeordnet werden. Darüber hinaus können einzelne **Scenes** in andere **Scenes** importiert

[334] vgl. Godot o. J.i
[335] vgl. Godot o. J.b
[336] vgl. Godot o. J.q
[337] vgl. Godot o. J.r

und darin verwendet werden.[338] Hierdurch ist jede einzelne **Scene** grundsätzlich leicht als wiederverwendbares Objekt angelegt; sobald eine **Scene** erstellt und gespeichert wurde, kann sie in andere **Scenes** implementiert werden. Änderungen an einer **Scene** spiegeln sich darüber hinaus automatisch an allen Verwendungen dieser **Scene** in anderen **Scenes** wider.

Der Ansatz des Aufbaus zusammenhängender Spielabschnitte, die der Wurzel-Knoten-Struktur folgen, lässt sich leicht auf andere Game Engines übertragen, die den gleichen Ansatz verfolgen. Von den in dieser Arbeit evaluierten Game Engines trifft das auf *Defold*, *jMonkeyEngine*, *FUDGE* und *Vektoria* zu.

Meinungen von Lehrenden und Zusammenfassung

Godot ist in vier der befragten zehn Lehrkontexte fest in den Lehrplan eingebunden[339,340,341,342] und wird in fünf weiteren Lehrkontexten von Studierenden für Spieleprojekte eingesetzt.[343,344,345,346,347] Die große Beliebtheit der Game Engine lässt sich an vielen Aspekten festmachen. Eines der am häufigsten genannten Argumente für *Godot* ist die sehr geringe Einstiegshürde[348] und die Tatsache, dass damit schnell Ergebnisse erzielt werden können, wodurch Lernende schnell motiviert werden.[349,350] Auch der *Open Source* Aspekt der Game Engine überzeugt – gerade im direkten Vergleich zu den anderen beiden häufig genannten Game Engines *Unity* und *Unreal Engine*.[351,352] Vor allem der hier erstgenannte Aspekt – die geringe Einstiegshürde – konnte auch in der vorangegangenen Evaluation bestätigt werden; *Godot* präsentiert sich insgesamt als gute Game Engine für den Einstieg in die Spieleerstellung. Während die Erlernbarkeit der Game Engine durch die recht hohe Menge an Tutorials und Hilfestellungen insgesamt gut ausfällt finden sich aber vor allem auch in der Besonderheit der Anwendbarkeit viele Punkte, die die Spieleerstellung auch für komplett Neueinsteigende sehr einfach machen. Darüber hinaus können auch erste Konzepte der Objektorientierung mit der Game Engine erlernt werden, wodurch die

[338] vgl. Godot o. J.r
[339] vgl. I0 Z. 31
[340] vgl. I1 Z. 45 f.
[341] vgl. I3 Z. 21
[342] vgl. I4 Z. 9 f.
[343] vgl. I2 Z. 62 f.
[344] vgl. I5 Z. 26 ff.
[345] vgl. I6 Z. 64
[346] vgl. I7 Z. 111 ff.
[347] vgl. I8 Z. 26 f.
[348] vgl. I0 Z. 107 f.
[349] vgl. I0 Z. 45 f.
[350] vgl. I1 Z. 129 – 134
[351] vgl. I3 Z. 26 f.
[352] vgl. I7 Z. 111 ff.

Übertragbarkeit trotz der Engine-eigenen Programmiersprache gegeben ist. Entsprechend eignet sich *Godot* aufgrund der genannten Aspekte gut für allerlei Lehrkontexte, die sich mit der Spieleerstellung befassen – besonders aber für solche, die sich auf den Einstieg in dieses Feld konzentrieren und dabei optional auch einen größeren Fokus auf die 2D-Spieleentwicklung setzen.

4.2.4 jMonkeyEngine

Kriterium	**Ergebnis**
Definitionstreue	*jMonkeyEngine* ermöglicht die Erstellung von 2D- und 3D-Spielen mithilfe der Programmiersprache *Java*.[353]
Aktualität	Version 3.6.1 wurde am 23. Juni 2023 veröffentlicht.[354] Auf *Itch.io* wurde zuletzt (07. August 2023) *Depthris* veröffentlicht.[355]
Preisgestaltung	*jMonkeyEngine* ist kostenfrei nutzbar und *Open Source.*[356]
Entwicklungsplattformen	*Windows, MacOS, Linux*[357]
Zielplattformen	*Windows, MacOS, Linux, iOS, Android, Web*[358]
Systemanforderungen	unbekannt

Tabelle 12 Evaluation der Qualifikationskriterien - jMonkeyEngine

Offizielle Lernmaterialien

jMonkeyEngine bietet eine Reihe an **offiziellen Lernmaterialien** um Neueinsteigenden die ersten Schritte mit der Game Engine, spezifische Konzepte und einzelne Aspekte konkret beizubringen. Letztere sind in sogenannten **Beginner Tutorials** zusammengefasst und bieten einen Einstieg in die wichtigsten Systeme der Game Engine. Die einzelnen Erklärungen sind allesamt ausschließlich auf Englisch verfügbar. Gleichzeitig gibt es zwar die Möglichkeit, einzelne Tutorials an eine bestimmte Version der Game Engine anzupassen, dies ist aber auf die Versionen 3.2, 3.3 und 3.4 beschränkt. Für die aktuelle Version 3.6.1 gibt es keine eigenen Tutorialseiten – möglicherweise aber auch, weil sich in der grundlegenden Herangehensweise hieran nichts geändert hat. Auch bezüglich der Aktualität der Tutorials lassen sich keine Aussagen treffen, da ein entsprechendes Veröffentlichungsdatum oder ähnliches nicht einsehbar ist.

[353] vgl. jMonkeyEngine 2023
[354] vgl. jMonkeyEngine o. J.k
[355] vgl. codewalker o. J.
[356] vgl. jMonkeyEngine 2023
[357] vgl. jMonkeyEngine o. J.l
[358] vgl. jMonkeyEngine o. J.a

Die tatsächliche Reproduzierbarkeit wurde anhand des **Beginner Tutorials Hello SimpleApplication** mit der Version 3.6.1 untersucht. Zunächst leitet das Tutorial Nutzende an, wie ein neues Projekt über das Editorfenster erstellt werden kann. Nach Befolgung der Schritte erschien im Rahmen dieser Evaluation ein Warnhinweis, dass eine oder mehrere Projekt-Ressourcen nicht gefunden werden konnten, mit der Frage, ob dieses Problem behoben werden soll. Das Problem konnte durch ein entsprechendes Dialogfenster kurze Zeit später gelöst werden – das Auftauchen des Problems lässt jedoch die Frage aufkommen, inwiefern Neueinsteigende hiervon nicht schon verunsichert werden könnten. Der nächste Stolperstein kam daraufhin beim Ausführen eines im Tutorial vorgegebenen Programmcodes auf; in der Anleitung stand hierzu lediglich, dass über Rechtsklick auf das Skript die Option **Run** ausgeführt werden konnte. Im Editorfenster gab es stattdessen nur die Optionen **Run File** (die ausgegraut war), **Run Appstate** und **Run Gradle**. Nach kurzem Herumprobieren war die richtige Lösung für dieses Problem zunächst **Run Gradle** auszuführen, wodurch die Option **Run File** ausführbar wurde und das Spiel, wie im Tutorial beschrieben, gestartet werden konnte. Der Rest des Tutorials konzentriert sich auf die Erklärung des erstellten Skripts und bietet damit einen guten Überblick über einzelne Vorgehensweisen im Umgang mit *jMonkeyEngine*. Dabei gibt es aber weder gute Möglichkeiten zur Selbstbeurteilung des Fortschritts (dies ist lediglich durch die Betrachtung und eigenständige Evaluation der erzielten Ergebnisse möglich), noch eine Form der Fremdbeurteilung. Entsprechend ist der Einstieg in den Umgang mit der Game Engine in diesem Kontext eher für situiertes Lernen, das die Betreuung durch eine Lehrperson ermöglicht, ausgelegt.

Auch die anderen Tutorials der **Beginner Tutorials** Reihe setzen diesen Lernprozess fort, wodurch Neueinsteigenden relativ schnell und größtenteils verständlich verschiedene Aspekte der Game Engine beigebracht werden. Insgesamt bietet *jMonkeyEngine* also einen guten, wenn auch etwas erschwerten, Einstieg, und einen groben Überblick über mögliche weiterführende Schritte im Umgang mit der Game Engine.

Inoffizielle Lernmaterialien

Bezüglich der inoffiziellen Lernmaterialien finden sich zwar auf *YouTube* ein paar Videos, die sich mit der Game Engine beschäftigen, diese sind aber jeweils recht alt.[359] Auch die Community auf der Plattform *Reddit* ist mit ca. 130 Mitgliedern sehr klein und darüber hinaus auch nur unregelmäßig aktiv im Teilen von Hilfestellungen und Erklärungen.[360] Da auch im Forum von *jMonkeyEngine* keine Kategorie zum Sammeln und Teilen von Tutorials existiert,

[359] vgl. Charles Anderson 2013
[360] vgl. /r/jMonkeyEngine o. J.

sind inoffizielle Lernmaterialien zusammengefasst also sowohl schwer zu finden als auch in ihrer Gesamtanzahl recht überschaubar.

Heuristische Analyse der Benutzeroberfläche

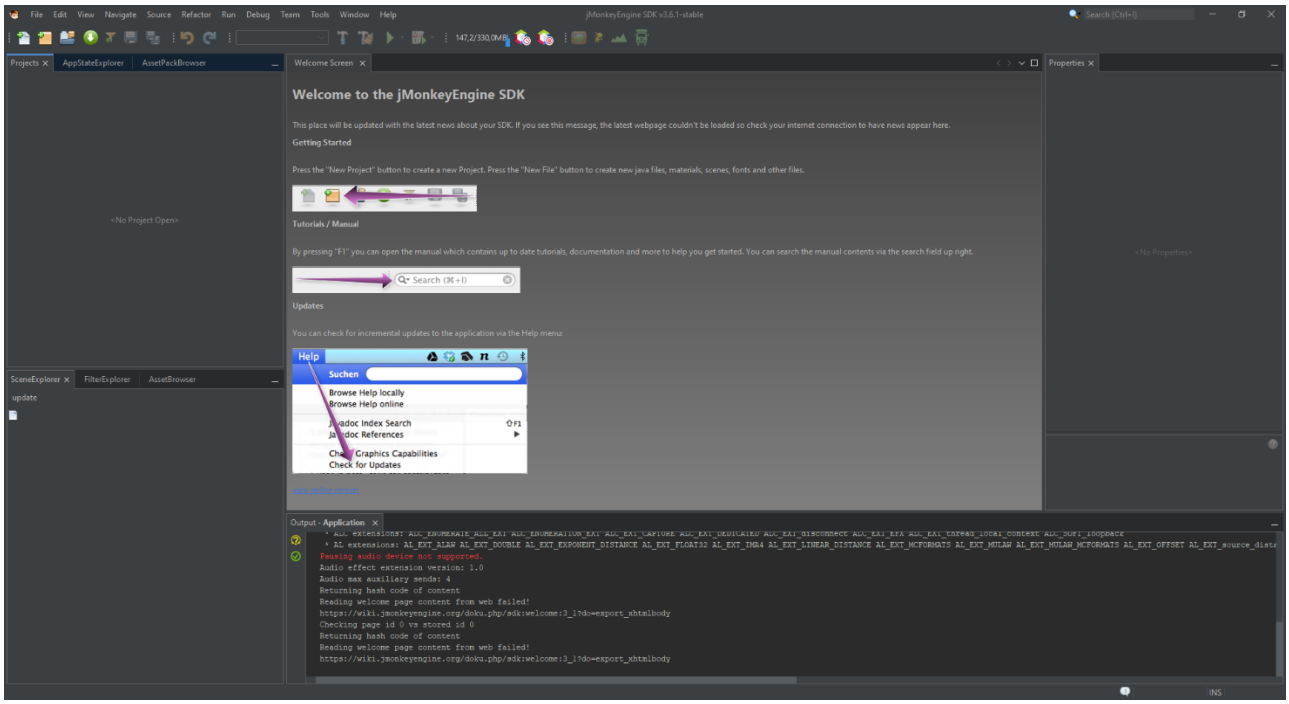

Abb. 12 Benutzeroberfläche jMonkeyEngine

Beim ersten Öffnen von *jMonkeyEngine* fällt direkt der **Welcome Screen** ins Auge, auf dem die Erstellung eines Projekts und das Öffnen von Tutorials und Hilfestellungen kurz sowohl textlich als auch bildlich erklärt wird (siehe **Abb. 12**). Darüber hinaus ist die Benutzeroberfläche recht aufgeräumt und nach kurzem Überlegen wird auch ersichtlich, wofür die einzelnen Fenster genutzt werden können; im **Projects**-Fenster findet man alle Dateien des momentanen Projects, das **SceneExplorer**-Fenster zeigt alle Elemente der momentan geöffneten Szene, im **Properties**-Fenster können weitere Informationen über ausgewählte Elemente erhalten werden und das **Output**-Fenster zeigt aktuelle Statusmeldungen des Spiels und der Game Engine. Assets können leicht über einen entsprechenden Menüpunkt und ein ausführliches Popup-Menü im **Projects**-Fenster erstellt werden. Die relativ große Anzahl an Menüpunkten und ihre vielen Unterpunkte, die nur teilweise mit Icons versehen sind, erschweren weitere Schritte zunächst. Skripte können direkt im Editor der Game Engine geöffnet werden, wodurch kein zusätzlicher Codeeditor benötigt wird. Der Menüpunkt **Help** bietet die Möglichkeit, die Dokumentation der Game Engine zu öffnen, sowie eine ***Java*doc Index Search** vorzunehmen. Was genau dies ist und wie es funktioniert wird jedoch nicht erklärt – entsprechend war diese Funktion der

Game Engine während dieses Ersteindrucks nicht sinnvoll nutzbar. Insgesamt wirkt die Benutzeroberfläche bezüglich zwar recht übersichtlich und ordentlich, an manchen Stellen aber zu kurz erklärt, wodurch auf die entsprechende Dokumentation zurückgegriffen werden muss.

Anwendbarkeit

Zusammensetzbarkeit

jMonkeyEngine unterstützt viele gängige Dateiformate für 2D-Bilddateien und 3D-Modell-Dateien. **fbx**-Dateien werden hierbei aber nicht offiziell unterstützt, weswegen ein Importtest dieser im Nachfolgenden entfällt. Bezüglich Audio-Dateien werden nur die Dateiformate **wav** und **ogg** unterstützt.[361] Der Import von Assets kann leicht per Drag-and-Drop vorgenommen werden. Nach dem Import können 2D-Bilddateien innerhalb der Game Engine beispielsweise zugeschnitten oder andersweitig bearbeitet werden. Einzelne Assets können in **Templates** zusammengeführt werden und damit für die Wiederverwendung in anderen *jMonkeyEngine*-Projekten gespeichert werden.

Effizienz der Nutzung

Neben Standardfunktionen bietet *jMonkeyEngine* zusätzlich die Möglichkeit zur Erstellung von Partikeleffekten[362], Animationen einzelner Spielobjekte[363], die Erstellung von Terrain[364] und weiterem. Darüber hinaus integriert *jMonkeyEngine* direkt *Git* zur Versionsverwaltung von Projekten, wodurch Projekte innerhalb der Game Engine direkt mit einer *Repository* verbunden werden können. Sobald ein Projekt mit einer entsprechenden *Repository* verbunden wurde, werden Änderungen an einzelnen Dateien direkt im Editorfenster mit einzelnen Icons angezeigt[365], was die Zusammenarbeit an einem *jMonkeyEngine*-Projekt innerhalb eines Teams sehr vereinfacht.

Besonderheiten

Die größte Besonderheit von *jMonkeyEngine* sind die gut dokumentierten Möglichkeiten zur tiefergreifenden Entwicklung von Spielen mithilfe der Game Engine. Hierzu zählt einerseits **Spidermonkey**, eine client-server networking *API* zur Erstellung von Multiplayer-Spielen[366] sowie andererseits die geringe Abstraktion von **OpenGL** innerhalb der Game Engine,

[361] vgl. jMonkeyEngine o. J.b
[362] vgl. jMonkeyEngine o. J.d
[363] vgl. jMonkeyEngine o. J.c
[364] vgl. jMonkeyEngine o. J.f
[365] vgl. jMonkeyEngine o. J.j
[366] vgl. jMonkeyEngine o. J.i

wodurch erfahrenen Entwickelnden viele Möglichkeiten zur effizienten Arbeit geboten werden.

Übertragbarkeit

Programmierparadigmen

jMonkeyEngine verwendet die objektorientierte Programmiersprache *Java*, weswegen auch alle in *jMonkeyEngine* verwendeten Skripte der Objektorientierung folgen. Dies äußert sich unter anderem darin, dass zunächst jedes Skript von der Basisklasse **SimpleApplication**[367] erbt und hierdurch bereits einige grundlegende Funktionen zur Verfügung gestellt bekommt.

Hieraus abgeleitet lassen sich jegliche Konzepte, die in *jMonkeyEngine* bezüglich der Objektorientierung erlernt und angewendet werden können, leicht auf andere objektorientierte Programmiersprachen und entsprechende Game Engines übertragen. Bezüglich der in dieser Arbeit evaluierten Game Engines trifft das auf *Flax*, *Unity*, *Unreal Engine*, *Cocos Creator*, *Godot*, *FUDGE* und *Vektoria* zu. Darüber hinaus lassen sich erlernte Kenntnisse bezüglich der Programmiersprache *Java* am ehesten noch auf *Flax*, *Unity* und *Godot* übertragen, da die dort verwendete Programmiersprache *C#* recht viele Ähnlichkeiten zu *Java* aufweist.

Schlüsselkonzepte

jMonkeyEngine basiert auf der Wurzel-Knoten-Struktur, wodurch alle Objekte an einen Eintrittspunkt (der sogenannten **Root**) angehängt werden. Einzelne Objekte, die an die **Root** angehängt werden, werden entsprechend **Nodes** genannt.[368] Einzelne **Nodes** können darüber hinaus sogenannten **Pivot Nodes**[369] untergeordnet werden, um über diese gesteuert zu werden.

Der Ansatz des Aufbaus zusammenhängender Spielabschnitte, die der Wurzel-Knoten-Struktur folgen, lässt sich leicht auf andere Game Engines übertragen, die den gleichen Ansatz verfolgen. Von den in dieser Arbeit evaluierten Game Engines trifft das auf *Defold*, *Godot*, *FUDGE* und *Vektoria* zu.

Meinungen von Lehrenden und Zusammenfassung

Da *jMonkeyEngine* von keinem der befragten Dozierenden eingesetzt wird, entfällt an dieser Stelle die Beleuchtung der Meinungen von Lehrenden bezüglich der Game Engine. Auch

[367] vgl. jMonkeyEngine o. J.g
[368] vgl. jMonkeyEngine o. J.e
[369] vgl. jMonkeyEngine o. J.h

wird die Game Engine in keinem der befragten Lehrkontext von Studierenden innerhalb entsprechender Projekte eingesetzt.

jMonkeyEngine ist insgesamt als ziemlich durchschnittliche Game Engine anzusehen. Zwar gibt es einige offizielle Lernmaterialien, mit denen der Einstieg recht leicht erfolgen kann, darüber hinaus ist die Erlernbarkeit aber durch die sehr kleine Community entsprechend erschwert. Auch bezüglich der Anwendbarkeit gibt es keine Aspekte, die *jMonkeyEngine* besonders herausstechen lassen. Dennoch sind im Umgang mit der Game Engine erlernte Kenntnisse leicht auf andere Programmiersprachen und Game Engines übertragbar. Zusammenfassend ist *jMonkeyEngine* also am ehesten für Lehrkontexte geeignet, die sich ausführlich mit einer bestimmten Game Engine oder der Programmiersprache *Java* beschäftigen wollen, können oder müssen und entsprechende Rahmenbedingungen durch eine sinnvolle Betreuung der Lernenden dafür bieten – dann kann mit *jMonkeyEngine* ein entsprechend tiefer Einblick in die Spieleentwicklung geboten werden.

4.3 Custom Engines

Unter dieser Kategorie sind alle Game Engines zusammengefasst, die eigens für den Einsatz in einem bestimmten Lehrkontext (beispielsweise für einen bestimmten Studiengang) entwickelt worden sind.

4.3.1 FUDGE

Kriterium	**Ergebnis**
Definitionstreue	*FUDGE* ermöglicht die Erstellung von 2D- und 3D-Spielen mithilfe der Programmiersprache TypeScript.[370]
Aktualität	Version 0.1.3 wurde am 29. Juni 2023 veröffentlicht.[371] Eine Übersichtsseite aktueller Spieleprojekte existiert nicht öffentlich, dennoch wird die Game Engine in jedem Semester in der Lehre für Spieleprojekte eingesetzt.
Preisgestaltung	*FUDGE* ist kostenfrei nutzbar und *Open Source*.[372]
Entwicklungsplattformen	*Windows, MacOS, Linux*[373]
Zielplattformen	*Browser*[374]
Systemanforderungen	WebGL2, WebAudio[375]

Tabelle 13 Evaluation der Qualifikationskriterien - FUDGE

[370] vgl. Dell'Oro-Friedl o. J.a
[371] vgl. Dell'Oro-Friedl o. J.c
[372] vgl. Dell'Oro-Friedl o. J.a
[373] vgl. I2 Z. 162 – 166
[374] vgl. I2 Z. 167 – 171
[375] vgl. I2 Z. 172 – 175

Erlernbarkeit

Offizielle Lernmaterialien

Für *FUDGE* gibt es eine über eine *Repository* gehostete Webseite mit einem Tutorial zur Installation und zum Starten der Game Engine, sowie eine Übersicht an **Fundamentals**, in der die ersten Schritte innerhalb der Game Engine und deren wichtigste Funktionsweisen erklärt werden.[376] Die Tutorials sind alle in Englisch gehalten und es gibt keine Möglichkeit einzelne Tutorials an die verwendete Version der Game Engine anzupassen. Auch bezüglich der Aktualität der Tutorials lassen sich keine Aussagen treffen, da ein entsprechendes Veröffentlichungsdatum oder ähnliches nicht einsehbar ist.

Die tatsächliche Reproduzierbarkeit wurde anhand des letztgenannten Tutorials zu **Fundamentals** mit Version 0.1.3 der Game Engine untersucht. Im Tutorial wird zunächst erklärt, wie über den Editor ein Projekt erstellt und geöffnet werden kann. Darüber hinaus werden weitere Informationen zu einzelnen Menüpunkten und eine kurze Erklärung zur Änderung des Editor-Themes gegeben – gefolgt von einer Erklärung zur Ordnerstruktur eines *FUDGE*-Projekts. Anschließend wird die Erstellung eines **Graphs** thematisiert. Hierzu wird erklärt, dass man diesen benötigt, damit man darin einzelne Objekte darstellen lassen kann. Um das Projekt dann im Standardbrowser ausführen zu können werden drei Lösungsansätze genannt, die jedoch nicht näher erklärt werden. Auch wird im Tutorial gezeigt, dass die **Interal.json** Datei im Projektordner alle von *FUDGE* generierten Ressourcen des Projekts anzeigt. Die Erklärung, dass diese dort aber erst angezeigt werden, sobald man das Projekt speichert, fehlt. Die nächsten Schritte, die das hinzufügen neuer **Materials**, **Lights** und **Transforms** erklären waren dennoch alle reproduzierbar. Auch die Erklärungen zu einzelnen Dateien im *FUDGE*-Projektordner waren hilfreich, lediglich die Erklärung zur Datei **Main.ts** ist zum Zeitpunkt der Evaluation (Stand 05.04.2024) unvollständig und entsprechend wenig hilfreich.

Insgesamt bieten die gegebenen Tutorials von *FUDGE* zwar einen guten ersten Überblick, die die ersten Schritte im Umgang mit der Game Engine erleichtern. Allerdings erschweren die teilweise unvollständigen Erklärungen und das Fehlen weiterer Guides beispielsweise zum Erstellen kleiner Spiele oder ähnlichem, sowie eine generelle Erklärung zu Zusammenhängen einzelner Spielelemente den weiteren Einstieg in *FUDGE*.

Inoffizielle Lernmaterialien

[376] vgl. Dell'Oro-Friedl o. J.b

Für *FUDGE* existieren neben den **offiziellen Lernmaterialien** keine weiteren Hilfestellungen zur Erlernung der Game Engine. Entsprechend entfällt hier eine Beleuchtung dieser.

Heuristische Analyse der Benutzeroberfläche

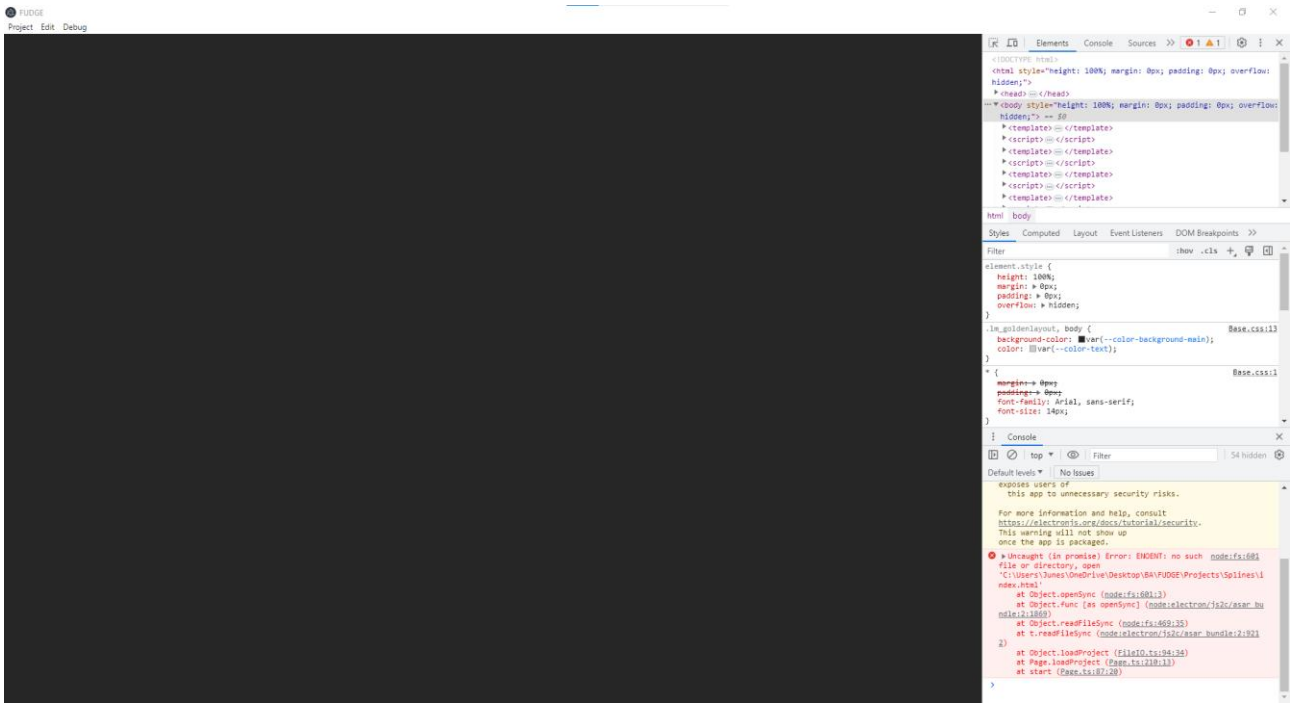

Abb. 13 Benutzeroberfläche FUDGE

Die Benutzeroberfläche von *FUDGE* ist beim ersten Öffnen recht leer und enthält lediglich das geöffnete **DevTool**-Fenster, in dem die **Console** angezeigt wird (siehe **Abb. 13**). Erst nach dem Erstellen und Öffnen eines Projekts über die entsprechenden Menüpunkte werden weitere Fenster der Game Engine sichtbar (siehe **Abb. 14**).

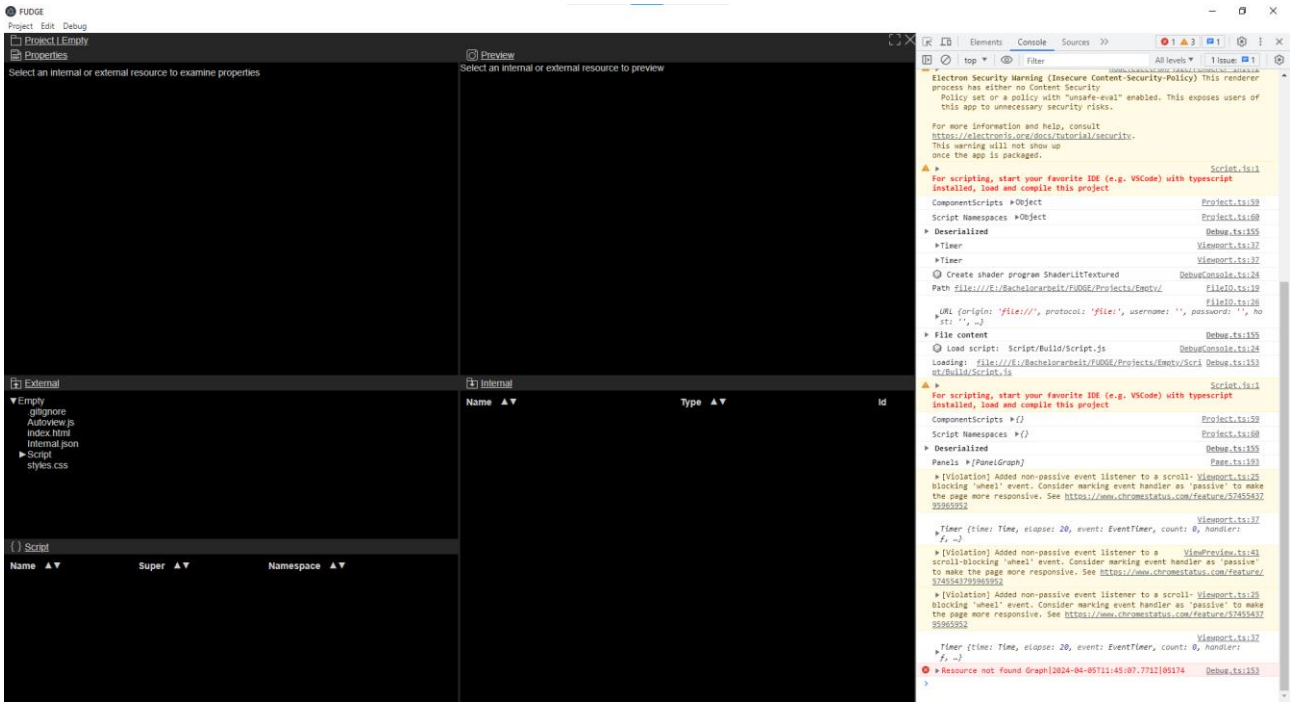

Abb. 14 Benutzeroberfläche FUDGE mit geöffnetem Projekt

Die dort angezeigten Fenster **Properties** und **Preview** besitzen jeweils einen kurzen Text, der beschreibt wie sie zu nutzen sind. Darüber hinaus wird schnell klar, dass das **External**-Fenster die Ordnerstruktur des Projekts anzeigt. Dass das **Internal**-Fenster zur Erstellung und Bearbeitung einzelner Elemente benutzt wird ist zwar nicht weiter erklärt, aber durch etwas Herumprobieren und einen entsprechenden Rechtsklick in das Fenster schnell eigenständig herausgefunden. Wofür das **Script**-Fenster jedoch genutzt werden soll, wird nicht ersichtlich. Die Auswahl eines Scripts im **External**-Fenster zeigt dessen Inhalt auch nur im **Preview**-Fenster an. Durch die Menüleiste können weitere Fenster sowie ein **Help**-Fenster mit generellen Erklärungen und Details zum **Graph**- und **Project**-Panel sowie das **Wiki** für weitere Hilfestellungen geöffnet werden. Insgesamt ist die Benutzeroberfläche von *FUDGE* relativ aufgeräumt und sehr simpel gehalten. Fehlende Erklärungen machen zwar einzelne Fenster und deren Zusammenhänge nicht ersichtlich, die sehr überschaubare Anzahl an Menüpunkten und die wenigen Optionen innerhalb der einzelnen Fenster überfordern aber auch nicht und können Neugierige zum ersten Ausprobieren einladen. Die fast gleiche farbliche und formliche Gestaltung der einzelnen Fenster, die teilweise nur durch ein kleines Icon voneinander unterscheidbar sind, trüben den ersten Eindruck dennoch weiter, weil ein entsprechendes Zurechtfinden innerhalb der Game Engine dadurch eine gewisse Eingewöhnungszeit erfordert.

Anwendbarkeit

Zusammensetzbarkeit

FUDGE besitzt keine vollständige Auflistung aller unterstützten Dateiformate, unterstützt für 2D-Bilddateien aber mindestens **png**. Durch die fehlende Dokumentation weiterer unterstützter Dateiformate müssen für jede Art von Assets zunächst entsprechende Tests durchgeführt werden. Der Import von Assets erfolgt über den Projektordner und wurde aufgrund der fehlenden Unterstützung von **fbx**-Dateien nur mit der **png**-Datei durchgeführt. Diese muss nach dem Import noch in den entsprechenden **Internal**-Ordner gezogen werden, bevor sie in *FUDGE* verwendet werden kann. Abseits davon besitzt *FUDGE* die Möglichkeit, leicht Objekte zur Wiederverwendung zu verwenden, da diese einfach aus dem **Internal.json** Script – in dem jedes einzelne Spielobjekt vollständig in Programmcode abgebildet ist – herauskopiert und in andere *FUDGE*-Projekte hineinkopiert werden können.

Effizienz der Nutzung

FUDGE bietet Nutzenden die Möglichkeit zur Implementierung von Audio-[377] und Licht-Elementen[378], die Erzeugung von Animationen für einzelne Objekte[379] und weitere einfache Funktionen und setzt dabei den Fokus eher auf die Erlernung dieser einzelnen Funktionen einer Game Engine als auf die tatsächlichen Spieleerstellung. Die Versionsverwaltung über *Git* oder ähnliche Systeme funktioniert sehr leicht, da *FUDGE* sehr code-nahe strukturiert ist und alle wichtigen Aspekte des jeweiligen Projekts vollständig im Code abgebildet werden.

Besonderheiten

Die größte Besonderheit von *FUDGE* ist der einzigarte Fokus der Game Engine. Wie bereits in **Effizienz der Nutzung** kurz angerissen verfolgt die Game Engine nicht das Hauptziel, besonders gut für die Spieleerstellung geeignet zu sein. Stattdessen wird durch die sehr code-nahe Struktur ein leichter und übersichtlicher Einblick in die technischen Hintergründe einer Game Engine geboten. So können einzelne **Components** beliebig stark individualisiert und entsprechend direkt im Editorfenster verwendet werden. Auch die Abbildung der gesamten Projektzusammensetzung in der **Interal.json** Datei macht es sehr leicht, die Abläufe der Game Engine nachzuvollziehen und hieraus entsprechende Kenntnisse über die Funktionsweise von Game Engines im Allgemeinen zu erlangen.

[377] Dell'Oro-Friedl 2020
[378] Dell'Oro-Friedl 2022a
[379] Dell'Oro-Friedl 2022b

Übertragbarkeit

Programmierparadigmen

FUDGE verwendet vorrangig *TypeScript* in Verbindung mit der Objektorientierung. Dies äußert sich unter anderem darin, dass jedes Script zunächst die Basisklasse **Component** erweitert und hierdurch bereits einige grundlegende Funktionen zur Verfügung gestellt bekommt. Diese Skripte können dann an **Nodes** angehängt werden, wodurch hier für Nutzende leicht der Zusammenhang zwischen dem Konzept eines Objekts auf Seiten der Programmierung mit dem einzelnen Spielelement in *FUDGE* dargestellt wird.

Hieraus abgeleitet lassen sich jegliche Konzepte, die in *FUDGE* bezüglich der Objektorientierung erlernt und angewendet werden können, leicht auf andere objektorientierte Programmiersprachen und entsprechende Game Engines übertragen. Bezüglich der in dieser Arbeit evaluierten Game Engines trifft das auf *Flax*, *Unity*, *Unreal Engine*, *Cocos Creator*, *Godot*, *jMonkeyEngine* und *Vektoria* zu. Lediglich der Umstieg von *TypeScript* auf andere Programmiersprachen, wie *C#* oder *C++*, die in der Spieleerstellung eher verwendet werden, erschwert die Übertragbarkeit der in *FUDGE* erlernten Fähigkeiten etwas. Am ehesten lassen sich erlernte Kenntnisse der Programmiersprache *TypeScript* aber direkt auf *Cocos Creator* übertragen.

Schlüsselkonzepte der Game Engine

FUDGE basiert auf einem **Graph**-System, das der Wurzel-Knoten-Struktur folgt, wodurch alle Objekte an einen Eintrittspunkt (dem sogenannten **Graph**-Node) angehängt werden. An diesem Graph können weitere **Nodes** angehängt werden. Diese wiederum setzen sich aus einzelnen **Components** zusammen und erhalten hierdurch einzelne Funktionsweisen.

Der Ansatz des Aufbaus zusammenhängender Spielabschnitte, die der Wurzel-Knoten-Struktur folgen, lässt sich leicht auf andere Game Engines übertragen, die den gleichen Ansatz verfolgen. Von den in dieser Arbeit evaluierten Game Engines trifft das auf *Defold*, *Godot*, *jMonkeyEngine* und *Vektoria* zu.

Meinungen von Lehrenden und Zusammenfassung

Da *FUDGE* nur von einem der befragten Dozierenden eingesetzt wird und dieser an der Entwicklung der Game Engine beteiligt ist, entfällt an dieser Stelle die Beleuchtung der Meinungen von Lehrenden bezüglich der Game Engine.

Zusammenfassend ist *FUDGE* bereits recht gut für das entsprechende Einsatzfeld – der Lehre von Hintergründen einzelner Funktionen von Game Engines – geeignet. Die etwas

zu simple visuelle Benutzeroberfläche und die quasi kaum vorhandenen offiziellen und inoffiziellen Lernmaterialien machen eine Erlernung im Kontext von eigenständigem Lernen und ohne viele Möglichkeiten der Selbstbeurteilung aber sehr schwer. Auch die Anwendung der Game Engine wird durch die bereits erwähnte Ausrichtung der Game Engine erschwert – eine effiziente Spieleerstellung ist mit *FUDGE* nicht so leicht möglich wie mit anderen Game Engines. Dennoch können erlernte Konzepte bezüglich der Objektorientierung leicht auf andere Game Engines übertragen werden. Entsprechend ist *FUDGE* besonders für Lehrkontexte geeignet, in denen sehr viel Wert auf die Erlernung tieferer Konzepte bezüglich der Spieleprogrammierung gelegt wird und entsprechend der besondere Fokus der Game Engine sinnvoll genutzt werden kann.

4.3.2 Vektoria

Kriterium	**Ergebnis**
Definitionstreue	*Vektoria* ermöglicht die Erstellung von 2D- und 3D-Spielen mithilfe der Programmiersprache *C++*[380]
Aktualität	Version 23.06 wurde am 27.01.2024 veröffentlicht[381] Eine Übersichtsseite aktueller Spieleprojekte existiert nicht öffentlich, dennoch wird die Game Engine in jedem Semester in der Lehre für Spieleprojekte eingesetzt.
Preisgestaltung	*Vektoria* ist unter bestimmten Bedingungen kostenfrei nutzbar.[382]
Entwicklungsplattformen	*Windows*[383]
Zielplattformen	*Windows*[384]
Systemanforderungen	DirectX11+[385]

Tabelle 14 Evaluation der Qualifikationskriterien - Vektoria

Erlernbarkeit

Offizielle Lernmaterialien

Für *Vektoria* gibt es ein **Manual** [386], das über die Seite der Hochschule Kempten heruntergeladen werden kann und jeweils für die aktuelle Hauptversion zur Verfügung steht – das aktuelle Manual entsprechend für Version 23 der Game Engine. Dieses besteht aus

[380] Vektoria o. J.
[381] vgl. I5 Z. 169
[382] vgl. Breiner o. J.c
[383] vgl. I5 Z. 170
[384] vgl. I5 Z. 171
[385] vgl. I5 Z. 172 – 175
[386] vgl. Breiner o. J.b

30 Kapiteln, die von der Installation und einer Einführung über erste einfache Programme bis hin zur Shaderprogrammierung einen Überblick über den Umgang und die Möglichkeiten mit der Game Engine bietet. Das Manual steht nur auf Deutsch zur Verfügung und eine Anpassung dessen an eine bestimmte Version der Game Engine ist nicht möglich. Auch bezüglich der Aktualität des Manuals lassen sich keine spezifischen Aussagen treffen, da ein entsprechendes Veröffentlichungsdatum oder ähnliches nicht einsehbar ist. Hierbei ist aber davon auszugehen, dass eine aktuelle Version des Manual jeweils mit einer neuen Version der Game Engine veröffentlicht wird.

Da das Manual hauptsächlich für den Einsatz in Vorlesungen beziehungsweise Seminaren entwickelt wurde, ist es außerhalb dieses entsprechenden Kontexts eher schlecht für das eigenständige Lernen optimiert. Eine detaillierte Beschreibung der Evaluation der Reproduzierbarkeit entfällt an dieser Stelle dementsprechend. Einzelne beschriebene und entsprechend getestete Vorgehensweisen können aber wie zu erwarten mit der aktuellen Version der Game Engine vollständig reproduziert werden. Eine Selbstbeurteilung ist darüber hinaus nur durch die Betrachtung der tatsächlich erzeugten Ergebnisse möglich, jegliche Fremdbeurteilung entfällt im Kontext des eigenständigen Lernens. Hierdurch ist zur sinnvollen Erlernung der Game Engine das situierte Lernen mit Betreuung durch eine entsprechende Lehrperson nötig. Besonders hilfreich sind aber die Vergleiche einzelner Konzepte von *Vektoria* zu ähnlichen Konzepten in anderen Game Engines innerhalb des Manuals. Diese Vergleiche beschränken sich aber bezüglich der in dieser Arbeit evaluierten Game Engines nur auf *Unity*. Ansonsten sind weitere Begriff, die in anderen Game Engines verwendet werden als Beispiele genannt.[387] Bezüglich unterschiedlicher Kompetenzstufen richtet sich die Game Engine sowie das Manual hauptsächlich an Studierende informatiknaher Studiengänge.

Inoffizielle Lernmaterialien

Für *Vektoria* existieren neben den offiziellen Lernmaterialien keine öffentlich einsehbaren weiteren Hilfestellungen zur Erlernung der Game Engine. Entsprechend entfällt hier eine Beleuchtung dieser.

Heuristische Analyse der Benutzeroberfläche

Da *Vektoria* keine visuelle Benutzeroberfläche besitzt, entfällt hier eine Evaluation dieser.

Anwendbarkeit

[387] Breiner o. J.b Dokument V02_HalloWeltProgramme_Act S. 6

Zusammensetzbarkeit

Vektoria unterstützt den Import von jeglichen Bilddateiformaten, **obj** für 3D-Modelle sowie **mp3** und **wav** für Audiodateien. Weitere Formate und der Import von Videodateien werden nicht unterstützt.[388] Der Import des Test-Modells im Dateiformat **fbx** entfällt an dieser Stelle entsprechend. Für den Import von 2D-Bilddateien muss zunächst eine entsprechende Umgebung bestehend aus einer **Root**, einem **Frame**, einem **Viewport**, einer Kamera und weiteren Elementen aufgesetzt, diese alle initialisiert und abschließend zusammengefügt werden. All dies passiert aufgrund einer fehlenden visuellen Benutzeroberfläche rein auf Ebene des Codes. Anschließend kann das Programm innerhalb der integrierten Entwicklungsumgebung ausgeführt werden und das Bild angezeigt werden. Einzelne **Roots** und jegliche damit zusammenhängende Objekte können darüber hinaus vollständig oder in Teilen exportiert und dadurch sehr leicht und effizient in anderen *Vektoria*-Projekten wiederverwendet werden.[389]

Effizienz der Nutzung

Aufgrund der fehlenden visuellen Benutzeroberfläche ist die Nutzung von *Vektoria* im Vergleich zu anderen in dieser Arbeit evaluierten Game Engines zunächst nicht sehr effizient. Anstatt einzelne Spielobjekte mit wenigen Klicks zu erstellen, müssen sie in *Vektoria* zunächst definiert, dann unter Berücksichtigungen eventueller Abhängigkeiten initialisiert und abschließend einem Kontext hinzugefügt werden. Auch bezüglich einzelner Funktionen beziehungsweise Werkzeugen ist *Vektoria* im Vergleich zu anderen Game Engines teilweise eingeschränkt. So existiert zum Beispiel keine Möglichkeit zur einfachen Erstellung von Charakteranimationen.[390] Abgesehen hiervon deckt *Vektoria* aber viele Funktionen ab, wie die Unterstützung unterschiedlicher Eingabegeräte (auch VR-Brillen), die Möglichkeit zur Erzeugung komplexer Partikeleffekte und die Möglichkeit zur Erstellung einzelner Module, die die Game Engine insgesamt um weitere Funktionen erweitern können.[391] Darüber hinaus besitzt die Game Engine weitere tiefgreifende Funktionen, die fraktale Terrainerzeugung, ein Vegetationssystem[392] und Fahrsimulationen[393] unterstützen. Aufgrund der Tatsache, dass *Vektoria* rein auf geschriebenem Programmcode basiert, ist eine Versionsverwaltung über *Git* oder andere Verwaltungssysteme sehr leicht möglich. Eine explizite Integration von *Git* in die Game Engine oder eine anderweitig besondere

[388] vgl. I5 Z. 176 – 179
[389] vgl. I5 Z. 180 – 192
[390] vgl. I5 Z. 164 f.
[391] vgl. Breiner o. J.a
[392] vgl. Breiner o. J.b Dokument V01_InstallationUndEinfuehrung_Act S. 16
[393] vgl. Breiner o. J.b Dokument V01_InstallationUndEinfuehrung_Act S. 17

Schnittstelle existiert nicht, ist aber aufgrund des zuvor genannten Umstands auch nicht nötig.

Besonderheiten

Zu den Besonderheiten der *Vektoria* Game Engine zählt vor allem der starke Fokus auf prozedurale Generierung von Spieleumgebungen.[394] Darüber hinaus sind die sogenannten *Pluralklassen* eine sogar rechtlich geschützte Besonderheit der *Vektoria* Game Engine. Diese existieren für jede vordefinierte Objektklasse und dienen als Container für diese, wodurch leicht, schnell und vielseitig mehrere Objekte desselben Typs verwaltet werden können.[395]

Übertragbarkeit

Programmierparadigmen

In *Vektoria* wird die objektorientierte Programmiersprache *C++* verwendet, wodurch Nutzende auch leicht jegliche Konzepte der Objektorientierung in Verbindung mit der Game Engine anwenden können. Dies äußert sich vor allem darin, dass jede Klasse in *Vektoria* von einer der insgesamt 23 Basisklassen erben kann und hierdurch unterschiedliche spezifische Funktionen zur Verfügung gestellt bekommt. Hierbei gibt es beispielsweise eigene Klassen für Audio, Spielszenen, Licht und weitere.[396]

Hieraus abgeleitet lassen sich jegliche Konzepte, die mit *Vektoria* bezüglich der Objektorientierung erlernt und angewendet werden können, leicht auf andere objektorientierte Programmiersprachen und entsprechende Game Engines übertragen. Bezüglich der in dieser Arbeit evaluierten Game Engines trifft das auf *Flax*, *Unity*, *Unreal Engine*, *Cocos Creator*, *Godot*, *jMonkeyEngine* und *FUDGE* zu. Darüber hinaus lassen sich erlernte Kenntnisse bezüglich der Programmiersprache *C++* direkt auf *Flax* und *Unreal Engine* übertragen.

Schlüsselkonzepte der Game Engine

Vektoria basiert auf einem **Root**-System, das der Wurzel-Knoten-Struktur folgt, wodurch alle Objekte an einen Eintrittspunkt (der sogenannten **Root**) angehängt werden. Neben der **Root** existiert zusätzlich noch die **Node**-Klasse, welche die Basisklasse für alle Spielobjekte in *Vektoria* darstellt. Objekte, die auf dieser Basisklasse basieren, können dann an die **Root** angehängt werden. Es existieren unterschiedliche vorgefertigte Objekte (sogenannte

[394] vgl. I5 Z. 120 f.
[395] vgl. Breiner o. J.b Dokument V30_Pluralklassen_Act
[396] vgl. Breiner o. J.b Dokument V02_HalloWeltProgramme_Act S. 87

Knotenobjekte), die so an die Root angehängt werden können. Hierbei existieren 21 **Knotenobjekte**, die jeweils unterschiedliche Funktionen umfassen. Alle **Nodes** besitzen darüber hinaus Methoden zur Änderung von **Parent-** oder **Child-Nodes**. Hierdurch können leicht weitere **Nodes** aneinandergehängt werden, um eine Hierarchie aufzuspannen.

Der Ansatz des Aufbaus zusammenhängender Spielabschnitte, die der Wurzel-Knoten-Struktur folgen, lässt sich leicht auf andere Game Engines übertragen, die den gleichen Ansatz verfolgen. Von den in dieser Arbeit evaluierten Game Engines trifft das auf *Defold*, *Godot*, *jMonkeyEngine* und *FUDGE* zu.

Meinungen von Lehrenden und Zusammenfassung

Da *Vektoria* nur von einem der befragten Dozierenden eingesetzt wird und dieser an der Entwicklung der Game Engine beteiligt ist, entfällt an dieser Stelle die Beleuchtung der Meinungen von Lehrenden bezüglich der Game Engine.

Zusammenfassend zeigt sich *Vektoria* eher geeignet für informatiknahe Studiengänge. Die fehlende visuelle Benutzeroberfläche und die vergleichsweise wenigen verfügbaren **offiziellen** und **inoffiziellen Lernmaterialien** machen eine Erlernung im Kontext von eigenständigem Lernen und ohne viele Möglichkeiten der Selbstbeurteilung sehr schwer. Auch die Anwendung wird durch diese Umstände erschwert – eine effiziente Spieleerstellung ist mit *Vektoria* nicht so leicht möglich wie mit anderen Game Engines. Dennoch verfolgt *Vektoria* viele sinnvolle Ansätze und besitzt einige Besonderheiten, die den Einsatz der Game Engine rechtfertigen. Auch können erlernte Konzepte leicht übertragen werden – vorrangig beispielsweise auf *Unreal Engine*, die auch in der Gameindustrie vor allem in kleinen bis mittelgroßen Spielestudios regen Einsatz findet. Entsprechend ist *Vektoria* besonders für Lehrkontexte geeignet, in denen sehr viel Wert auf die Erlernung tieferer Konzepte bezüglich der Spieleprogrammierung gelegt wird. Sobald zukünftig zusätzlich eine entsprechende visuelle Benutzeroberfläche für *Vektoria* vorhanden ist, spricht zunächst auch nichts gegen einen Einsatz der Game Engine in Lehrkontexten, die sich eher mit der Spieleerstellung im Allgemeinen befassen und dabei gleichzeitig die Möglichkeit zu einer informatiknahen Spezialisierung in der Spieleprogrammierung bieten.

4.4 Vergleichende Zusammenfassung der Evaluation

Zur abschließenden Zusammenfassung der Evaluation werden im Folgenden die zuvor einzeln betrachteten Game Engines in Verbindung miteinander gebracht und bezüglich der Evaluationskriterien miteinander verglichen.

Zunächst lässt sich an dieser Stelle ein vergleichender Blick auf die Kosten der jeweiligen Game Engines wagen. Hier haben vorerst alle evaluierten Game Engines die Gemeinsamkeit, dass die grundsätzliche Spieleerstellung mit ihnen immer kostenfrei ist. Lediglich sobald ein entwickeltes Spiel kommerzialisiert und entsprechend verkauft werden soll, fallen unterschiedliche Lizenzgebühren beziehungsweise umsatzabhängige Abgaben an. Bezüglich dieser teilweise kostenpflichtigen Game Engines kann vor allem *GameMaker* mit dem einmaligen und vergleichsweise günstigen Preis von 99,99$ überzeugen. Aber auch *Flax* und *Unreal Engine* besitzen mit 4% beziehungsweise 5% Lizenzgebühren ab einem gewissen hohen Schwellenwert ein recht faires Preismodell. Unübersichtlicher und teurer ist da *Unity*, weil hier schon ab einem Umsatz von 100.000$ innerhalb eines Jahres der Umstieg auf eine der recht teuren kostenpflichtigen Lizenzen nötig ist. Auf der Seite der kostenfreien Game Engines erübrigt sich dieses Problem. Gleichzeitig gibt es hier bei allen den Vorteil, dass sie zusätzlich *Open Source* sind und damit eine freie Änderung und Erweiterung des zugrundeliegenden Programmcodes ermöglichen. Dies bietet zusätzliche Freiheiten für Nutzende der Game Engines.

Übertragen auf das Lehrumfeld macht die Kostenfrage dabei zunächst keinen Unterschied bezüglich des Einsatzes von Game Engines. Möchte man aber den jeweiligen Studierenden die Möglichkeit bieten, direkt nach dem Studium selbständig zu werden und dabei mögliche anfallende Kosten gering zu halten, ist natürlich der Einsatz von kostenfreien Game Engines eine präferierte Vorbereitung. Gleichzeitig kann man aufgrund des *Open Source* Aspekts hier gezielt eine technische Vertiefung in die Programmierung beziehungsweise Erweiterung von Game Engines anbieten. Auch *Unreal Engine* ist *Open Source*, wodurch dieser Weg auch mit dieser kommerziell orientierten Game Engine möglich wäre. Gleichzeitig würde man hierdurch eine Game Engine lehren, die momentan auch vielerorts in der Spielebranche – und damit im späteren Berufsfeld der Studierenden – zur Entwicklung von Spielen eingesetzt wird. Eine ähnliche „Marktnähe“ kann man dabei gleichzeitig aber auch mit *Unity* und teilweise (vor allem im Indie-Bereich) auch mit *Godot* abdecken. Ob sich zusammenfassend die möglicherweise entstehenden Kosten einer Game Engine im Vergleich zu einer komplett kostenfreien Game Engine bezüglich der jeweiligen Funktionen für Nutzende lohnen, lässt sich nicht pauschal sagen und eher durch einen gezielten direkten Vergleich bewerten. Hierbei unterstützen möglicherweise die folgenden Vergleiche.

4.4.1 Vergleich bezüglich Erlernbarkeit

Mit Blick auf die Menge und Verfügbarkeit der **offiziellen Lernmaterialien** sowie die mögliche Hilfe durch **inoffizielle Lernmaterialien**, die sich durch die Aktivität und Größe der jeweiligen Community sowie die Anzahl an sonstigen Hilfestellungen äußern, wurde die folgende Übersichtsskala als Vergleich aller evaluierten Game Engines gemäß ihrer Erlernbarkeit erstellt (siehe **Abb. 15**). Die Skala stellt dabei keine absolute Aussage über die Erlernbarkeit der Game Engines dar, sondern ordnet alle untersuchten jeweils in Abhängigkeit der Orientierungspunkte links (vergleichsweise am einfachsten erlernbar), mittig (vergleichsweise durchschnittlich erlernbar) und rechts (vergleichsweise am schwersten erlernbar) ein.

Abb. 15 Vergleichende Skala bezüglich der Erlernbarkeit evaluierter Game Engines

Die jeweiligen Orientierungspunkte beziehungsweise die jeweils zugeordneten Game Engines wurden aus folgenden Gründen so gewählt: *Unity* ist von allen untersuchten Game Engines am einfachsten erlernbar, da hier mit **Unity Learn** eine extensive Lernplattform für Einsteigende existiert, die gleichzeitig die Möglichkeit zur umfassenden Selbst- und Fremdbeurteilung ermöglicht. Darüber hinaus besitzt *Unity* eine sehr große und aktive Community, die vor allem auf *YouTube* sehr viele verschiedene Hilfestellungen zu unterschiedlichen Themen anbieten. *FUDGE* wird hier dagegen als am schwersten erlernbare Game Engine eingestuft, da das einzig frei verfügbare Lernmedium teilweise unvollständig und gleichzeitig wenig tiefgreifend ist. Auch **inoffiziellen Lernmaterialien** sind hierbei nicht verfügbar. Genau zwischen diesen beiden Extrempunkten wird *Unreal Engine* als durchschnittlich erlernbare Game Engine eingestuft. Während es hier zwar recht viele offizielle und inoffizielle Lernmaterialien gibt, sind diese teilweise sehr schwer für komplett Neueinsteigende nachzuvollziehen, was den Einstieg entsprechend erschwert. Die große Community bietet dafür sehr viele Hilfestellungen für unterschiedliche Themen. Dagegen erschwert aber wiederum die Programmiersprache *C++* einen einfacheren Einstieg. Es gibt zusammenfassend also viele Argumente, die sich für eine Einordnung in Richtung eher einfacherer oder eher schwererer Erlernbarkeit gegenüberstehen, weswegen *Unreal Engine* als mittleren Orientierungspunkt am passendsten ist.

Links dieser Mitte finden sich *Godot* und *GameMaker*. Erstere weil sie mit vielen einfach verständlichen offiziellen Lernmaterialien glänzt und gleichzeitig eine immer größer werdende Community besitzt, die den Einstieg in den Umgang mit der Game Engine aus vielen Perspektiven heraus unterstützt. Dieser letzte Aspekt erklärt darüber hinaus die Einordung von *GameMaker* weiter rechts; die Community hier ist im direkten Vergleich deutlich kleiner. Dafür sind aber viele der **offiziellen Lernmaterialien** explizit auf Neueinsteigende ausgelegt, was die Game Engine insgesamt betrachtet eher einfach erlernbar macht.

Auf der anderen Seite der Skala finden sich entsprechend alle Game Engines, die eher wenige **offizielle Lernmaterialien** besitzen und eine kleine Community vorzuweisen haben, was sie verglichen mit *Unreal Engine* eher schwerer erlernbar macht. Direkt links neben *FUDGE* reiht sich hier die zweite Custom Engine dieser Untersuchung ein. Im direkten Vergleich zu der am schwersten erlernbaren Game Engine besitzt *Vektoria* deutlich mehr **offizielle Lernmaterialien**, in denen nicht nur der erste Einstieg in den Umgang mit der Game Engine, sondern auch weiterführende Themen übersichtlich abgedeckt werden. Hierauf folgt *jMonkeyEngine*, die ebenfalls ausreichend viele **offizielle Lernmaterialien** zur Verfügung stellt und gleichzeitige eine kleine Community besitzt, über die zumindest der Zugang zu weiteren **inoffiziellen Lernmaterialien** möglich ist. Da diese aber vergleichsweise überschaubar und größtenteils auch recht alt sind, ist *Flax* im direkten Vergleich entsprechend als einfacher erlernbar einzuordnen. Noch einfacher sind dabei *Cocos Creator* und *Defold* erlernbar. Hierbei stellen beide recht viele **offizielle Lernmaterialien** zur Verfügung, *Cocos Creator* unterliegt *Defold* im direkten Vergleich aber durch die etwas unübersichtliche Zusammensetzung dieser und der vergleichsweise kleineren Community, aufgrund welcher der Zugang zu weiteren Hilfestellungen erschwert ist.

4.4.2 Vergleich bezüglich Anwendbarkeit

Mit Blick auf die unterstützen Dateiformate und Import- und Export-Möglichkeiten, die sich in der **Zusammensetzbarkeit** der Game Engine äußern, sowie die Funktionen und Werkzeuge, die die untersuchten Game Engines unter dem Aspekt der **Effizienz der Nutzung** zur Verfügung stellen, wurde die folgende Übersichtsskala als Vergleich aller evaluierten Game Engines gemäß ihrer Anwendbarkeit bezüglich einer bestimmten Spieledimension erstellt (siehe **Abb. 16**). Die Skala stellt dabei keine absolute Aussage über die Eignung der Game Engines für eine bestimmte Spieledimension dar, sondern ordnet alle untersuchten jeweils in Abhängigkeit der Orientierungspunkte links (vergleichsweise am

besten für 2D anwendbar), mittig (vergleichsweise gleich gut für 2D und 3D anwendbar) und rechts (vergleichsweise am besten für 3D anwendbar) ein.

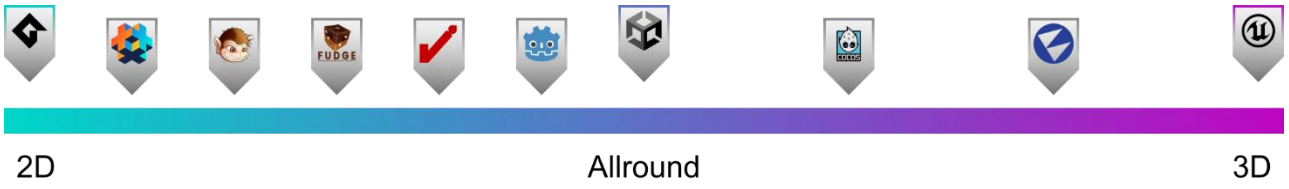

Abb. 16 Vergleichende Skala bezüglich der Anwendbarkeit in bestimmten Darstellungsdimensionen evaluierter Game Engines

Die jeweiligen Orientierungspunkte beziehungsweise die jeweils zugeordneten Game Engines wurden aus folgenden Gründen so gewählt: *GameMaker* ist am meisten auf die 2D-Spieleerstellung spezialisiert, was sich vor allem darin äußert, dass standardmäßig kein Import von 3D-Modell-Dateien möglich ist. Darüber sind auch einige Funktionen, wie beispielsweise die Sprite-Erstellung innerhalb der Game Engine klar auf die 2D-Entwicklung ausgelegt. Auf der anderen Seite der Skala liegt dagegen *Unreal Engine*. Die Game Engine ermöglicht zwar die Erstellung von Spielen in 2D, setzt aber den Fokus ganz klar auf 3D-Spiele. Dies äußert sich vor allem in den sehr detaillierten Funktionen zum Umgang mit 3D-Modell-Dateien und dem Niagara-System. Als Game Engine, die gleich gut für die Erstellung von 2D- sowie 3D-Spielen eingesetzt werden kann, ist *Unity* in der Mitte der Skala platziert. Dies ist sowohl an der Menge an unterstützen Dateiformaten für beide Ausprägungen als auch an der Tatsache, dass einzelne Funktionen nicht eine der beiden Darstellungsdimensionen bevorzugen, festgemacht. *Unity* ist also als Allround-Engine für beide Spieledimensionen sehr sinnvoll anwendbar.

Ausgehend hiervon sind *Defold*, *jMonkeyEngine*, *FUDGE*, *Vektoria* und Godot eher in Richtung der Zweidimensionalität ausgelegt. *Defold* und *jMonkeyEngine* sind sich dabei sehr ähnlich, da sie beide das **fbx**-Dateiformat für 3D-Modell-Dateien nicht unterstützen. Im direkten Vergleich ist *Defold* noch eher auf 2D spezialisiert, da ein Großteil der Erklärungen im zugehörigen Manual eher in diese Richtung ausgelegt ist. Auch *FUDGE* und *Vektoria* befinden sich aufgrund der geringen Unterstützung von 3D-Dateiformaten eher auf der Seite der 2D-Spieleerstellung. Da beide – und vor allem *Vektoria* – im direkten Vergleich zu anderen Game Engines auf dieser Seite aber mehr Funktionen für die 3D-Spieleerstellung bereitstellen, sind sie entsprechend eher mittig orientiert. Abschließend ist Nahe der Skalenmitte auch *Godot* platziert. Diese Game Engine stellt zwar Erklärungen für beide Darstellungsdimensionen dar, besitzt aber ebenfalls keine standardmäßig verfügbaren Optionen für den Import von **fbx**-Dateien. Zusammenfassend ist die Game Engine also zwar

eher in Richtung der Allround-Spieleerstellung ausgelegt, besitzt aber einen etwas leichteren Umgang für 2D-Spieleprojekte.

Auf der eher in Richtung 3D-Spieleentwicklung ausgelegten rechten Seite der Mitte findet sich zunächst *Cocos Creator*. Ähnlich wie *Unity* unterstützt auch diese Game Engine viele Formate für 3D-Modell-Dateien und lässt sich auch aufgrund des ähnlichen Aufbaus der Benutzeroberfläche eher im Allround-Bereich mit Tendenz zum stärkeren Fokus auf 3D verorten. Abschließend findet sich noch etwas weiter rechts auf der Skala *Flax*. Begründet ist dies einerseits durch die große Anzahl an unterstützten Dateiformaten für 3D-Modelle und andererseits durch die eher für die 3D-Spieleerstellung ausgelegte Benutzeroberfläche.

Den Vergleich bezüglich der Anwendbarkeit abschließend zeigt die folgende Tabelle mit einem letzten Blick auf die unterschiedlichen Möglichkeiten zur Versionsverwaltung eine Gegenüberstellung verschiedener Lösungsansätze:

Game Engine	**Versionsverwaltung im Editor möglich**	**Eigenes System existiert**	**Keine explizite Lösung**
Flax			X
GameMaker	X		
Unity		X	
Unreal Engine	X		
Cocos Creator			X
Defold	X		
Godot	(X)		
jMonkeyEngine	X		
FUDGE			X
Vektoria			X

Tabelle 15 Übersicht einzelner Lösungsansätze zur Versionsverwaltung innerhalb der evaluierten Game Engines

Hierzu lässt sich vergleichend feststellen, dass Game Engines, die eine Versionsverwaltung direkt im Editor (im Fall von *Godot* erst mithilfe eines Plugins) ermöglichen, die entsprechende Effizienz der Spieleerstellung fördern. Bezüglich der Game Engines die keine explizite Lösung zur Versionsverwaltung zur Verfügung stellen, lässt sich hervorheben, dass *FUDGE* und *Vektoria* im direkten Vergleich zu *Flax* und *Cocos Creator* aufgrund ihrer Codenähe noch einfacher mit Systemen wie *Git* benutzbar sind. Inwiefern das eigene System von *Unity* hier im Gegensatz zu anderen Lösungen überzeugen kann, müsste in einem direkten Vergleich getestet werden.

4.4.3 Vergleich bezüglich Übertragbarkeit

Bezüglich der Übertragbarkeit einzelner Kenntnisse, die im Umgang mit den evaluierten Game Engines erlernt wurden, lassen sich konkrete Vergleiche bezüglich der verwendeten Programmiersprachen, Schlüsselkonzepten und Schlüsselbegriffen aufstellen.

Game Engine	***C#***	***C++***	***JavaScript / TypeScript***	***Java***	***Lua***	**Eigene Sprache(n)**
Flax	X	X				
GameMaker						X
Unity	X					
Unreal Engine		X				
Cocos Creator			X			
Defold					X	
Godot	X					X
jMonkeyEngine				X		
FUDGE			X			
Vektoria		X				

Tabelle 16 Übersicht einzelner Programmiersprachen der evaluierten Game Engines

Beim Blick auf die verwendeten Programmiersprachen zeigt sich hier einerseits eine recht große Vielfalt von insgesamt sechs unterschiedlichen Sprachen bei den insgesamt zehn evaluierten Game Engines (siehe **Tabelle 16**). Andererseits finden sich aber auch viele Übertragungsmöglichkeiten durch einige Überschneidungen. Am praktischsten ist dieser Umstand bei *Flax* und *Godot*, die beide zwei unterschiedliche Programmiersprachen abdecken und dadurch leicht auf *Unity* und/oder *Unreal Engine* übertragbar sind. Auch lässt sich durch diese Übersicht leicht erkennen, dass *C#* und *C++* die dominanten Programmiersprachen sind, wenn es um die Spieleerstellung geht. Die Erlernung einer Game Engine, die eine oder beide dieser Sprachen unterstützt, ist also entsprechend sinnvoll, um die erlernten Fertigkeiten später auf das Berufsumfeld übertragen zu können.

Game Engine	**Freie-Szenen-Struktur**	**Wurzel-Knoten-Struktur**
Flax	X	
GameMaker	X	
Unity	X	
Unreal Engine	X	
Cocos Creator	X	
Defold		X

Godot		X
jMonkeyEngine		X
FUDGE		X
Vektoria		X

Tabelle 17 Überschrift der verwendeten Strukturen zur Präsentation der Zusammensetzung einzelner Spielabschnitte innerhalb der evaluierten Game Engines

Bezüglich der Präsentation der Zusammensetzung einzelner Spielabschnitte innerhalb der Game Engines lässt sich leicht die in der Definition des Evaluationskriteriums aufgestellte Zweiteilung erkennen (siehe **Tabelle 17**). Im direkten Vergleich der beiden Strukturen lässt sich aber keine direkte Aussage darüber treffen, ob eine der beiden Strukturen bestimmte Vorteile zur Verständlichkeit oder Benutzbarkeit bezüglich der Spieleerstellung aufweist. Entsprechend kann hier zusammenfassend festgehalten werden, dass es durchaus sinnvoll sein kann, beide Strukturen einmal anhand von konkreten Game Engines kennenzulernen.

Game Engine	**Spielabschnitt**	**Einzelnes Spielobjekt**	**Wiederverwendbares Objekt**
Flax	Scene	Actor	Prefab
GameMaker	Room	Object	Object
Unity	Scene	GameObject	Prefab
Unreal Engine	Level	Actor, Pawn	Blueprint
Cocos Creator	Scene	Node	Prefab
Defold	Collection	GameObject	Prototype
Godot	Scene	Node	Scene
jMonkeyEngine	-	Node	Node
FUDGE	Graph	Node	Graph
Vektoria	-	Node	Node

Tabelle 18 Übersicht verwendeter Schlüsselbegriffe innerhalb der evaluierten Game Engines

In der Benennung einzelner Spielabschnitte, Spielobjekte und wiederverwendbarer Objekte lassen sich ebenfalls ein paar Gemeinsamkeiten der unterschiedlichen Game Engines erkennen (siehe **Tabelle 18**); Fast alle Game Engines, die der **Wurzel-Knoten-Struktur** folgen, verwenden entsprechend auch die Bezeichnung **Node** für einzelne Spielobjekte – mit Ausnahme von *Defold*. Bei Game Engines die dagegen der **Freie-Szenen-Struktur** folgen gibt es solche Gemeinsamkeiten nicht. Auch die Bezeichnungen für Spielabschnitte unterscheiden sich teilweise sehr. Lediglich der Begriff **Scene** ist in vier der evaluierten Game Engines im Einsatz. Auch bezüglich der Bezeichnung für wiederverwendbare Objekte lassen sich nur über den Begriff **Prefab** ein paar Gemeinsamkeiten finden. Darüber hinaus

besitzen *jMonkeyEngine* und *Vektoria* keine eigenen Begriffe für zusammenhängende Spielabschnitte, wodurch diese quasi als Ansammlung von einzelnen **Nodes** aufgefasst werden können. Zusammenfassend zeigt diese Übersicht also, dass eine Übertragbarkeit einzelne Begriffe zwischen unterschiedlichen Game Engines eher schwierig ist und nur teilweise der generellen Strukturierung der Game Engine folgend angepasst ist. Da einzelne Begriffe bei der (Neu-)Erlernung einer Game Engine recht schnell erlernt werden können und dabei auch auf bisherigen Erfahrungen mit anderen Game Engines leicht aufgebaut werden kann, stellt dieser Umstand aber kein Hindernis dar.

4.4.3 Einsatz in der Lehre

Um die aufgestellten Vergleiche abschließend zusammenzufassen werden an dieser Stelle noch einige konkrete Einsatzvorschläge der evaluierten Game Engines aufgestellt. Diese unterscheiden sich von den Einsatzmöglichkeiten am Ende der jeweiligen Kapitel in dem Punkt, dass sie die einzelnen Game Engines nicht isoliert, sondern im Kontext der anderen Game Engines betrachten.

Wie die vorausgegangene Evaluation zeigt, ist grundsätzlich jede in dieser Arbeit evaluierte Game Engine für einen Einsatz in der Lehre geeignet. Somit wird zumindest ein Großteil der aufgestellten Forschungsfrage beantwortet. Die Detailfrage, welche Game Engine denn am besten für die Lehre der Spieleerstellung geeignet ist, lässt sich dagegen nicht so leicht beantworten, da eine Antwort sehr von den jeweiligen Inhalten und Zielen einzelner Lehrkontexte abhängig ist. Ein Studiengang, der sich beispielsweise nur oberflächlich mit der technischen Seite der Spieleerstellung beschäftigt und vielleicht eher einen Fokus auf Designaspekte von Spielen legt, braucht eine deutlich leichter zu erlernende und effizienter einzusetzende Game Engine als ein Studiengang, der sich eher mit der Spieleprogrammierung im Besonderen befasst. Hier wäre es entsprechend vermutlich sinnvoller eine Game Engine einzusetzen, die viele Möglichkeiten zur technisch detaillierten Spieleerstellung und -optimierung bietet – auch wenn diese vielleicht zunächst schwerer zu erlernen ist.

Im Gesamtkontext aller untersuchten Game Engines bieten jedoch *Unity*, *Unreal Engine* und *Godot* sowohl einzeln als auch vor allem in Kombination, das größte Potenzial am passendsten für die meisten Lehrkontexte einsetzbar und gleichzeitig sinnvoll auf das spätere Berufsfeld übertragbar zu sein. Eine Inkludierung aller drei Game Engines in einen Lehrkontext könnte also die wichtigsten Inhalte der Spieleerstellung gut abdecken und gleichzeitig eine vielfältige Übersicht über zugehörige Tools bieten. Am sinnvollsten wäre hier wahrscheinlich, den grundsätzlichen Einstieg über *Godot* stattfinden zu lassen, da diese Game Engine von den genannten drei am schnellsten zu ersten Ergebnissen führen kann,

die die weitere Motivation der Studierenden sicherstellen. Anschließend wäre ein Umstieg auf *Unity* sinnvoll, um einzelne Bereiche der Spieleerstellung tiefer zu erkunden. Gerade durch die große Menge an qualitativ hochwertigen Lernmaterialien wäre dies auch sehr einfach machbar. Den Abschluss kann daraufhin *Unreal Engine* bilden, um einerseits noch detailliertere und performantere Spiele zu ermöglichen und gleichzeitig gut auf das Berufsfeld vorzubereiten. Andererseits hätte man mit diesem Dreigestirn die weit verbreitetsten und damit wichtigsten Game Engines der aktuellen Zeit abgedeckt.

Der Ansatz, die jeweilige Lehre nicht nur auf eine Game Engine sondern stattdessen auf mindestens zwei oder sogar drei zu fokussieren, bietet, wie an diesem Beispiel zu erkennen ist, also unterschiedliche Vorteile. Als Alternative zu dieser Vorgehensweise könnte statt *Godot* beispielsweise auch *GameMaker* oder *Defold* und statt *Unity* auch *Flax* oder *Cocos Creator* genutzt werden. Diese sind zwar vergleichsweise weniger bekannt und Studierende finden außerhalb entsprechender Vorlesungen und Seminare eher weniger Hilfestellungen, dennoch eignen sie sich trotzdem gut, um ähnliche Aspekte der Spieleerstellung an ihnen erlernen zu können, teilweise sogar mit einzelnen Besonderheiten im direkten Vergleich wie beispielsweise der großen Asset-Datenbank von *Defold* oder der Fokussierung von *Cocos Creator* auf Web- und Mobile-Spiele.

FUDGE und *Vektoria* bilden hierbei als Custom Engines die zusätzliche spannende Möglichkeit, einzelne Konzepte und Hintergründe von und um Game Engines näher zu beleuchten. Entsprechend wäre ihr Einsatz über ihre aktuelle Unterbringung in Lehrkontexte hinaus sicher auch für Spezialisierungsmöglichkeiten oder Wahlmodule in anderen Studiengängen sinnvoll. Auch *jMonkeyEngine* füllt hier eine bestimmte Nische durch die geringe Abstraktion einzelner Aspekte innerhalb der Game Engine aus und könnte entsprechend ebenfalls in einem zusätzlichen Wahlmodul sinnvoll eingesetzt werden.

Generell lässt sich hier zusammenfassend sagen, dass eine größere Diversität im Einsatz einzelner Game Engines generell und gleichzeitig der Einsatz von mehreren Game Engines innerhalb eines Lehrkontexts wünschenswerte Ansätze wären. Hiermit würden einerseits Studierende viele unterschiedliche Einblicke in verschiedene Herangehensweisen an die Spieleerstellung erfahren und andererseits auch die Communities der einzelnen Game Engines wachsen – also eine Win-win-Situation für alle Beteiligten.

5. Fazit

So vielfältig die Landschaft bezüglich unterschiedlicher Game Engines eigentlich ist, so monoton ist sie aber mit Blick auf den konkreten Einsatz dieser in den beleuchteten Lehrplänen. Neben *Unity*, *Unreal Engine* und *Godot* finden sich in den befragten Lehrkontexten nur Custom Engines. Hierbei zeigt die vorliegende Arbeit definitiv Ausbaupotenzial beziehungsweise Erweiterungspotenzial anhand der untersuchten Game Engines. Alle innerhalb dieser Arbeit evaluierten Game Engines eignen sich für die Spieleentwicklung und einen entsprechenden Einsatz in Lehrplänen. Letztendlich bleibt der Einsatz einer bestimmten Game Engine momentan aber von vielen unterschiedlichen und teilweise sehr spezifischen Kriterien abhängig und wird darüber hinaus meistens davon bestimmt, welche Game Engine denn tatsächlich auch im Berufsfeld direkte Anwendung finden kann oder ohne größere Probleme leicht dorthin übertragbar ist.

Dennoch würde mehr Diversität bezüglich der eingesetzten Game Engines sicher nicht schaden. Es ist zwar durchaus sinnvoll, mindestens eine der „großen Drei“ (*Unity*, *Unreal Engine* und *Godot*) abzudecken, aber Studierende sollten auch betreute Möglichkeiten erhalten, um Alternativen näher kennenzulernen. Gründe dafür gibt es genug: So wird man nur Tool-agnostisch, wenn man tatsächlich auch andere Tools ausprobiert und dadurch ein breites Verständnis von Game Engines an sich erlernt – und weniger die konkrete Anwendung einer spezifischen Game Engine. Darüber hinaus besitzen einzelne Game Engines möglicherweise Besonderheiten, die in anderen gar nicht vertreten sind und so die Spielentwicklung mit neuen Augen betrachtbar und erlernbar machen. Auch der Aspekt *Open Source* beziehungsweise kostenfrei nutzbare Game Engines im Allgemeinen ist im Kontext von undurchschaubaren Lizenzänderungen einzelner Game Engines definitiv nicht zu unterschätzen. Wer weiß schließlich, wie lange kommerzielle Game Engines überhaupt noch sinnvoll in der Lehre einsetzbar sind.

Bezüglich der zu Beginn dieser Arbeit aufgestellten Forschungsfrage lässt sich an dieser Stelle klar sagen: Eine beste Game Engine für die Lehre gibt es nicht – grundsätzlich eignen sich sehr viele Game Engines für die Erlernung der Spieleerstellung oder einzelner Aspekte dieser. *Unity*, *Unreal Engine* und *Godot* kommen aber einer möglichst optimalen Lösung sehr nahe und können gerade in Kombination alle wichtigen Aspekte der Spieleerstellung sinnvoll abbilden. Eine optimale Einbindung von Game Engines in den Lehrplan sollte also nicht singulär passieren. Es ist zwar definitiv möglich, einzelne Konzepte an einer spezifischen Game Engine so zu erklären, dass die erlernten Kenntnisse dann auch sinnvoll auf andere Game Engines übertragbar sind. Noch viel besser wäre es aber, wenn man direkt die Übertragbarkeit mit mindestens einer zweiten Game Engine selbst auch lehrt und

entsprechend erfährt. Für die Motivation einzelner Studierenden kann es auch fördernd sein, zunächst mit einer leichter erlernbaren Game Engine anzufangen und dann später auf komplexere Tools zu wechseln. Bezüglich der Auswahl von konkret einzusetzenden Game Engines kann man mit den in dieser Arbeit beleuchteten Optionen keine falsche Entscheidung treffen. Die einzige falsche Entscheidung beim Einsatz von Game Engines für die Lehre ist, eine entsprechende Vielfalt der zu lehrenden Game Engines auszulassen.

5.1 Kritische Würdigung

Anknüpfend an das Fazit dieser Arbeit wird an dieser Stelle noch ein Blick auf die Entstehung dieser, entsprechende Vorgehensweisen und zugehörige Arbeitsschritte gelegt und diese kritisch reflektiert. Hierbei liegt der Fokus weniger auf der Hervorhebung gut gelungener Aspekte dieser Arbeit, sondern vielmehr auf der Betrachtung einzelner Verbesserungsmöglichkeiten für zukünftige Forschungen in diesem Themenbereich.

Vorrangig sei hierbei die Definition und die zugehörige Evaluation des Kriteriums der Anwendbarkeit genannt. Durch das Evaluationskriterium wird im Rahmen der Arbeit zwar ein grober Überblick darüber gegeben, inwieweit die jeweils untersuche Game Engine tatsächlich praktisch anwendbar ist, die Grundlage auf die sich die Untersuchung dieses Aspekts stützt, fällt aber etwas schwach aus; die Überprüfung von Import- und Export-Möglichkeiten, sowie Optionen zur Versionsverwaltung beziehungsweise kollaborativen Arbeit innerhalb einer Game Engine decken hierbei nur einzelne Teilperspektiven der praktischen Spieleerstellung ab. Sinnvoller wäre hier vermutlich eine tatsächliche praktische Erstellung eines Spiels mit der jeweils untersuchten Game Engine gewesen. Dabei hätten einfacher einzelne Eigenheiten sowie Vor- und Nachteile der Game Engine für bestimmte Anwendungsfälle beleuchtet werden können. Alternativ wäre das Kriterium der Anwendbarkeit auch durch eine größere Studie, in der mehrere Studierende ohne viele Vorerfahrungen in der Spieleerstellung mithilfe der einzelne Game Engines einfache Spiele erstellen und im Nachhinein vergleichend ihre Arbeit mit den einzelnen Engines evaluieren, sinnvoll überprüfbar gewesen. Leider konnten diese beiden Ansätze aufgrund des detaillierten Fokus auf andere Evaluationskriterien im Kontext des Zeitrahmens dieser Arbeit praktisch nicht durchgeführt werden.

Bezüglich der Auswertung der qualitativen Experteninterviews gibt es auch eine Verbesserungsmöglichkeit. So hätte noch vor der Evaluation einzelner Game Engines bei der Durchführung und Codierung der qualitativen Experteninterviews eine effizientere Vorgehensweise verfolgt werden können; Anstatt zunächst alle Interviews durchzuführen und im Anschluss mit der vollständigen Codierung dieser zu beginnen, hätte bereits nach dem ersten Interview eine grobe Codierung dessen stattfinden können. Hierbei hätten

einzelne Aspekte bei zukünftigen Interviews aufgegriffen werden können, was möglicherweise zu einer vollständigeren Abdeckung aller potenziell wichtigen Aspekte des Themenkomplexes beigetragen hätte.

Während die Forschungsfrage dieser Arbeit zwar ausführlich beantwortet werden konnte, bleibt dennoch abzuwarten, inwiefern die konkreten Forschungsergebnisse, die hierdurch aufgestellt werden konnten, auch längerfristig Relevanz behalten. Mit einem letzten Blick auf die zuletzt chaotische Lizenzpolitik von *Unity* wird ersichtlich, dass das Gebiet der Game Engines schnell durch einzelne disruptive Ereignisse verändert werden kann und sich damit quasi dauerhaft im Wandel befindet. Ob also in einem Jahr die in dieser Arbeit evaluierten Game Engines noch genauso gut für die Lehre geeignet sind, vielleicht durch eine anwachsende Community noch besser geeignet sein werden, oder durch einschränkende Nutzungsänderungen für die Lehre unbrauchbar werden, lässt sich aus dieser Arbeit resultierend nicht hervorsagen. Neben den Erkenntnissen der einzelnen Evaluationen hat diese Arbeit aus Sicht des Autors aber vor allem auch ein stabiles Grundgerüst an Qualifikations- und Evaluationskriterien geliefert, durch das auch in Zukunft noch Game Engines auf ihre Einsatzfähigkeit für einen Einsatz in der Lehre der Spieleerstellung überprüft werden können.

5.2 Ausblick

Aufbauend auf der vorliegenden Arbeit gibt es zukünftig noch einige weitere Möglichkeiten, den konkreten Einsatz von Game Engines in der Lehre der Spieleerstellung aus unterschiedlichen Perspektiven zu beleuchten. An vorderster Stelle wäre hier anzuführen, dass einfach mehr unterschiedliche Game Engines evaluiert und verglichen werden könnten. Hierfür wäre zwar eine entsprechende Lockerung der aufgestellten Qualifikationskriterien nötig, eine Untersuchung einzelner Game Engines, die sich beispielsweise aber auf eine bestimmte Darstellungsdimension, ein Genre oder eine Plattform fokussieren würde aber einen breiten Blick auf besondere Tools bieten, die in einzelnen Wahlmodulen in der Lehre sicher sinnvoll aufgehoben wären.

Anknüpfend an die vorausgegangene kritische Würdigung wäre darüber hinaus zusätzlich eine großflächig angelegte Überprüfung der tatsächlichen Anwendbarkeit sehr hilfreich, um ein möglichst differenziertes und detailliertes Bild einzelner Game Engines zu erhalten.

Abschließend stellt vor allem auch die regelmäßige Neuevaluation einzelner Game Engines unter Verwendung der in dieser Arbeit aufgestellten Kriterien und eventuell einzelner weiterer – möglicherweise auch kontextspezifischer – Kriterien eine gute Möglichkeit für weitere Forschungsarbeiten in diesem komplexen Themenbereich dar.

Hochschule Neu-Ulm
University of Applied Sciences

Literaturverzeichnis

Abra (o. J.): Abra - YouTube. URL: https://www.youtube.com/@abradotcs/videos (21.05.2024).

Adamina, Marco (2014): Lehr- und Lernmaterialien im kompetenzorientierten Unterricht. In: Beiträge zur Lehrerinnen- und Lehrerbildung, 32. Jg., S. 359–372.

Alves, Henrique L. (o. J.): Engines Database. URL: https://enginesdatabase.com/ (21.05.2024).

Andrade, A. (2015): Game engines: a survey. In: EAI Endorsed Transactions on Game-Based Learning, 2. Jg. (6), S. 150615.

Apache (o. J.): Apache Subversion. URL: https://subversion.apache.org/ (27.05.2024).

Blender (o. J.): Introduction - Blender 4.1 Manual. URL: https://docs.blender.org/manual/en/latest/getting_started/about/introduction.html (27.05.2024).

Breiner, Tobias (o. J.a): Features. URL: https://www.hs-kempten.de/fakultaet-informatik/zentrale-einrichtungen/computerspiel-zentrum-games/vektoria-features (22.05.2024).

Breiner, Tobias (o. J.b): Link - LRZ Sync+Share. URL: https://syncandshare.lrz.de/getlink/fi8QgkqCfJxjZCpuVPN3Ly/ (22.05.2024).

Breiner, Tobias (o. J.c): Rechtliches. URL: https://www.hs-kempten.de/fakultaet-informatik/zentrale-einrichtungen/computerspiel-zentrum-games/vektoria/rechtliches (22.05.2024).

Charles Anderson (2013): Part 1: Java Tutorial Series using jMonkeyEngine - Hello World. URL: https://www.youtube.com/watch?v=Aa7_R-4gWdo (22.05.2024).

Christopoulou, Eleftheria/Xinogalos, Stelios (2017): Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. In: International Journal of Serious Games, 4. Jg. (4), S. 21–36.

Cocos (o. J.a): Cocos - YouTube. URL: https://www.youtube.com/@CocosEngine/videos (22.05.2024).

Cocos (o. J.b): Latest Knowledge base topics. URL: https://discuss.cocos2d-x.org/c/knowledgebase/41 (22.05.2024).

Cocos Creator (o. J.a): Cocos Creator - Efficient and lightweight cross-platform 3D/2D graphics engine. URL: https://www.cocos.com//en (22.05.2024).

Cocos Creator (o. J.b): Cocos Creator 3.0 Manual - Node. URL: https://docs.cocos.com/creator/3.0/manual/en/concepts/scene/node.html (22.05.2024).

Cocos Creator (o. J.c): Cocos Creator 3.0 Manual - Nodes and Components. URL: https://docs.cocos.com/creator/3.0/manual/en/concepts/scene/node-component.html#components (22.05.2024).

Cocos Creator (o. J.d): Cocos Creator 3.0 Manual - Prefab. URL: https://docs.cocos.com/creator/3.0/manual/en/asset/prefab.html (22.05.2024).

Cocos Creator (o. J.e): Cocos Creator 3.0 Manual - Project Structure. URL: https://docs.cocos.com/creator/3.0/manual/en/getting-started/project-structure/#version-control (22.05.2024).

Cocos Creator (o. J.f): Cocos Creator 3.1 Manual - Basic Usage. URL: https://docs.cocos.com/creator/3.1/manual/en/getting-started/ (22.05.2024).

Cocos Creator (o. J.g): Cocos Creator 3.8 Manual - Animation. URL: https://docs.cocos.com/creator/manual/en/animation/ (22.05.2024).

Cocos Creator (o. J.h): Cocos Creator 3.8 Manual - Audio. URL: https://docs.cocos.com/creator/manual/en/asset/audio.html (22.05.2024).

Cocos Creator (o. J.i): Cocos Creator 3.8 Manual - Cross-platform Publishing. URL: https://docs.cocos.com/creator/manual/en/editor/publish/ (22.05.2024).

Cocos Creator (o. J.j): Cocos Creator 3.8 Manual - Images. URL: https://docs.cocos.com/creator/manual/en/asset/image.html (22.05.2024).

Cocos Creator (o. J.k): Cocos Creator 3.8 Manual - Install and Launch. URL: https://docs.cocos.com/creator/manual/en/getting-started/install/ (22.05.2024).

Cocos Creator (o. J.l): Cocos Creator 3.8 Manual - Lighting. URL: https://docs.cocos.com/creator/manual/en/concepts/scene/light.html (22.05.2024).

Cocos Creator (o. J.m): Cocos Creator 3.8 Manual - Programming Language Support. URL: https://docs.cocos.com/creator/manual/en/scripting/language-support.html (22.05.2024).

Cocos Creator (o. J.n): Cocos Creator 3.8 Manual - Scene. URL: https://docs.cocos.com/creator/manual/en/editor/scene/ (22.05.2024).

Cocos Creator (o. J.o): Cocos Creator 3.8 Manual - Terrain System. URL: https://docs.cocos.com/creator/manual/en/editor/terrain/ (22.05.2024).

Cocos Creator (o. J.p): Cocos Creator 3.8 Manual - UI System. URL: https://docs.cocos.com/creator/manual/en/2d-object/ui-system/ (22.05.2024).

Cocos Creator (o. J.q): Cocos Creator Download - Lightweight and efficient development engine. URL: https://www.cocos.com/en/creator-download (22.05.2024).

Cocos Creator (o. J.r): Installing and Launching | Cocos Creator. URL: https://docs.cocos.com/creator/3.7/manual/en/getting-started/install/index.html (28.05.2024).

Code Monkey (o. J.): Code Monkey - YouTube. URL: https://www.youtube.com/@CodeMonkeyUnity (21.05.2024).

codewalker (o. J.): Depthris™ by codewalker. URL: https://codewalker.itch.io/depthris (22.05.2024).

ComputerWeekly (o. J.): Was ist C# (C-Sharp)? - Definition von Computer Weekly. URL: https://www.computerweekly.com/de/definition/C-C-Sharp (27.05.2024).

Cyan (o. J.): Cyan - YouTube. URL: https://www.youtube.com/@cyangineer/videos (22.05.2024).

Defold Foundation (o. J.a): Defold - Official Homepage - Cross platform game engine. URL: https://defold.com/ (22.05.2024).

Defold Foundation (o. J.b): Defold Asset Portal. URL: https://defold.com/assets/ (22.05.2024).

Defold Foundation (o. J.c): Defold engine and editor FAQ. URL: https://defold.com/faq/faq/ (22.05.2024).

Defold Foundation (o. J.d): Download Defold. URL: https://defold.com/download/ (22.05.2024).

Defold Foundation (o. J.e): Editor keybindings. URL: https://defold.com/manuals/editor-keyboard-shortcuts/ (22.05.2024).

Defold Foundation (o. J.f): Importing and using 2D graphics. URL: https://defold.com/manuals/importing-graphics/ (22.05.2024).

Defold Foundation (o. J.g): Importing models. URL: https://defold.com/manuals/importing-models/ (22.05.2024).

Defold Foundation (o. J.h): Introduction to Defold. URL: https://defold.com/manuals/introduction/ (22.05.2024).

Defold Foundation (o. J.i): Latest Questions topics. URL: https://forum.defold.com/c/questions/5 (22.05.2024).

Defold Foundation (o. J.j): Live update content in Defold. URL: https://defold.com/manuals/live-update/ (22.05.2024).

Defold Foundation (o. J.k): The building blocks of Defold. URL: https://defold.com/manuals/building-blocks/ (22.05.2024).

Defold Foundation (o. J.l): The building blocks of Defold. URL: https://defold.com/manuals/building-blocks/#objects-added-in-place-or-by-reference (22.05.2024).

Defold Foundation (o. J.m): Version control. URL: https://defold.com/manuals/version-control/ (22.05.2024).

Defold Foundation (o. J.n): Writing native extensions for Defold. URL: https://defold.com/manuals/extensions/ (22.05.2024).

Defold Tutorials (o. J.): Defold Tutorials - YouTube. URL: https://www.youtube.com/@DefoldTutorials/videos (22.05.2024).

Dell'Oro-Friedl, Jirka (2020): Audio · JirkaDellOro/FUDGE Wiki · GitHub. URL: https://github.com/JirkaDellOro/FUDGE/wiki/Audio (29.05.2024).

Dell'Oro-Friedl, Jirka (2022a): Light. URL: https://github.com/JirkaDellOro/FUDGE/wiki/Light (29.05.2024).

Dell'Oro-Friedl, Jirka (2022b): Animation · JirkaDellOro/FUDGE Wiki · GitHub. URL: https://github.com/JirkaDellOro/FUDGE/wiki/Animation (29.05.2024).

Dell'Oro-Friedl, Jirka (o. J.a): FUDGE. URL: https://jirkadelloro.github.io/FUDGE/ (22.05.2024).

Dell'Oro-Friedl, Jirka (o. J.b): FUDGE Tutorial. URL: https://jirkadelloro.github.io/FudgeTutorial/Page/fundamentals.html (22.05.2024).

Dell'Oro-Friedl, Jirka (o. J.c): Releases · JirkaDellOro/FUDGE. URL: https://github.com/JirkaDellOro/FUDGE/releases (22.05.2024).

Doctor, Moth (2022): Basic Class Structure | Community tutorial. URL: https://dev.epicgames.com/community/learning/tutorials/7xWm/unreal-engine-basic-class-structure (22.05.2024).

DragoniteSpam (2024): Importing 3D Object Files in GameMaker. URL: https://www.youtube.com/watch?v=rKLp38gI1HI (21.05.2024).

Epic Games (o. J.a): Actors and Geometry. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/actors-and-geometry-in-unreal-engine (22.05.2024).

Epic Games (o. J.b): Animation. URL: https://www.unrealengine.com/en-US/uses/animation (22.05.2024).

Epic Games (o. J.c): Audio Engine Overview. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/audio-engine-overview-in-unreal-engine (22.05.2024).

Epic Games (o. J.d): Blueprint Visual Scripting. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine (22.05.2024).

Epic Games (o. J.e): Components Window. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/components-window-in-unreal-engine (22.05.2024).

Epic Games (o. J.f): Creating and Displaying UI. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/creating-widgets-in-unreal-engine (22.05.2024).

Epic Games (o. J.g): Creating Landscapes. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/creating-landscapes-in-unreal-engine (22.05.2024).

Epic Games (o. J.h): Creating Visual Effects In Niagara For Unreal Engine | Unreal Engine 5.4 Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/creating-visual-effects-in-niagara-for-unreal-engine (22.05.2024).

Epic Games (o. J.i): Epic Developer Community. URL: https://dev.epicgames.com/community/unreal-engine/getting-started/games (22.05.2024).

Epic Games (o. J.j): Epic Developer Community Learning | Tutorials, Courses, Demos & More. URL: https://dev.epicgames.com/community/unreal-engine/learning?languages=de-de (22.05.2024).

Epic Games (o. J.k): Hardware and Software Specifications for Unreal Engine | Unreal Engine 5.0 Dokumentation | Epic Developer Community. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/hardware-and-software-specifications-for-unreal-engine?application_version=5.0 (22.05.2024).

Epic Games (o. J.l): Importing Audio Files | Unreal Engine 5.3 Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/audio-files-in-unreal-engine (22.05.2024).

Epic Games (o. J.m): Levels In Unreal Engine | Unreal Engine 5.4 Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/levels-in-unreal-engine (22.05.2024).

Epic Games (o. J.n): Lighting The Environment In Unreal Engine | Unreal Engine 5.4 Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/lighting-the-environment-in-unreal-engine (22.05.2024).

Epic Games (o. J.o): Lumen Global Illumination And Reflections In Unreal Engine | Unreal Engine 5.0 Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-global-illumination-and-reflections-in-unreal-engine (22.05.2024).

Epic Games (o. J.p): Migrating Assets In Unreal Engine | Unreal Engine 5.4 Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/migrating-assets-in-unreal-engine (22.05.2024).

Epic Games (o. J.q): Pawn. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/pawn-in-unreal-engine (22.05.2024).

Epic Games (o. J.r): Sharing And Releasing Projects For Unreal Engine | Unreal Engine 5.4 Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/sharing-and-releasing-projects-for-unreal-engine (22.05.2024).

Epic Games (o. J.s): The most powerful real-time 3D creation tool. URL: https://www.unrealengine.com/de/download (22.05.2024).

Epic Games (o. J.t): Unreal Engine 5.3 jetzt verfügbar – alle Neuigkeiten hier! URL: https://www.unrealengine.com/de/blog/unreal-engine-5-3-is-now-available (22.05.2024).

Epic Games (o. J.u): Was ist Fortnite? Leitfaden für Einsteiger. URL: https://www.fortnite.com/news/what-is-fortnite-beginners-guide (27.05.2024).

Epic Games (o. J.v): Unreal Engine (UE5) licensing options. URL: https://www.unrealengine.com/en-US/license (28.05.2024).

Epic Online Learning/Wadstein, Mathew (2023): Your First Hour in Unreal Engine 5.2 | Course. URL: https://dev.epicgames.com/community/learning/courses/3ke/your-first-hour-in-unreal-engine-5-2 (22.05.2024).

fandom (2024): Super Mario Maker. URL: https://mario.fandom.com/de/wiki/Super_Mario_Maker (27.05.2024).

Figat, Wojciech (2023): Flax 1.7.2 released. URL: https://flaxengine.com/blog/flax-1-7-2-released/ (21.05.2024).

Flax (o. J.a): Actors | Flax Documentation. URL: https://docs.flaxengine.com/manual/get-started/scenes/actors.html (21.05.2024).

Flax (o. J.b): Animation | Flax Documentation. URL: https://docs.flaxengine.com/manual/animation/index.html (21.05.2024).

Flax (o. J.c): Audio | Flax Documentation. URL: https://docs.flaxengine.com/manual/audio/index.html (21.05.2024).

Flax (o. J.d): C++ Scripting | Flax Documentation. URL: https://docs.flaxengine.com/manual/scripting/cpp/index.html#c-scripting-with-flax (21.05.2024).

Flax (o. J.e): Class Script | Flax Documentation. URL: https://docs.flaxengine.com/api/FlaxEngine.Script.html (21.05.2024).

Flax (o. J.f): Editor on Linux | Flax Documentation. URL: https://docs.flaxengine.com/manual/get-started/linux.html (21.05.2024).

Flax (o. J.g): Editor on Mac | Flax Documentation. URL: https://docs.flaxengine.com/manual/get-started/mac.html (21.05.2024).

Flax (o. J.h): Features. URL: https://flaxengine.com/features/ (21.05.2024).

Flax (o. J.i): HOWTO: Change scene from script | Flax Documentation. URL: https://docs.flaxengine.com/manual/scripting/tutorials/change-scene.html?tabs=code-csharp (21.05.2024).

Flax (o. J.j): Importing audio | Flax Documentation. URL: https://docs.flaxengine.com/manual/audio/importing.html (21.05.2024).

Flax (o. J.k): Importing models | Flax Documentation. URL: https://docs.flaxengine.com/manual/graphics/models/import.html (21.05.2024).

Flax (o. J.l): Licensing. URL: https://flaxengine.com/licensing/ (21.05.2024).

Flax (o. J.m): Lighting | Flax Documentation. URL: https://docs.flaxengine.com/manual/graphics/lighting/index.html (21.05.2024).

Flax (o. J.n): Prefabs | Flax Documentation. URL: https://docs.flaxengine.com/manual/get-started/prefabs/index.html (21.05.2024).

Flax (o. J.o): Profiling | Flax Documentation. URL: https://docs.flaxengine.com/manual/editor/profiling/index.html (21.05.2024).

Flax (o. J.p): Requirements | Flax Documentation. URL: https://docs.flaxengine.com/manual/get-started/requirements.html (21.05.2024).

Flax (o. J.q): Scenes | Flax Documentation. URL: https://docs.flaxengine.com/manual/get-started/scenes/index.html (21.05.2024).

Flax (o. J.r): Sprites | Flax Documentation. URL: https://docs.flaxengine.com/manual/graphics/sprites/index.html (21.05.2024).

Flax (o. J.s): Tutorials | Flax Documentation. URL: https://docs.flaxengine.com/manual/samples-tutorials/tutorials/index.html (21.05.2024).

Flax (o. J.t): Version Control Systems | Flax Documentation. URL: https://docs.flaxengine.com/manual/get-started/version-control.html (21.05.2024).

Flax (o. J.u): Visual Scripting | Flax Documentation. URL: https://docs.flaxengine.com/manual/scripting/visual/index.html (21.05.2024).

Flax (o. J.v): Windows | Flax Documentation. URL: https://docs.flaxengine.com/manual/platforms/windows.html?q=editor%20on%20windows (28.05.2024).

Franzke, Dr Reinhard (o. J.): Kriterien zur Analyse und Beurteilung von Schulbüchern. URL: https://opendata.uni-halle.de/bitstream/1981185920/94461/1/sachunterricht_volume_0_5867.pdf.

game (o. J.): Wer wir sind. URL: https://www.game.de/ueber-den-game/ (27.05.2024).

Game Dev Guide (o. J.): Game Dev Guide - YouTube. URL: https://www.youtube.com/@GameDevGuide (21.05.2024).

Gamefromscratch (2023): Defold For Unity Developers. URL: https://www.youtube.com/watch?v=-3CzCbd4QZ0 (22.05.2024).

GameMaker (o. J.a): GameMaker - YouTube. URL: https://www.youtube.com/@GameMakerEngine/videos (21.05.2024).

GameMaker (o. J.b): GameMaker Funktionen und Werkzeuge. URL: https://gamemaker.io/de/features (21.05.2024).

GameMaker (o. J.c): GameMaker Manual - Animation. URL: https://manual.gamemaker.io/monthly/en/index.htm?#t=GameMaker_Language%2FGML_Reference%2FAsset_Management%2FSprites%2FSkeletal_Animation%2FAnimation%2FAnimation.htm (21.05.2024).

GameMaker (o. J.d): GameMaker Manual - Audio. URL: https://manual.gamemaker.io/monthly/en/#t=GameMaker_Language%2FGML_Reference%2FAsset_Management%2FAudio%2FAudio.htm (21.05.2024).

GameMaker (o. J.e): GameMaker Manual - Cloning A Repository. URL: https://manual.gamemaker.io/monthly/en/index.htm?#t=IDE_Tools%2FSource_Control%2FCloning_A_Repository.htm (21.05.2024).

GameMaker (o. J.f): GameMaker Manual - Lighting. URL: https://manual.gamemaker.io/monthly/en/index.htm?#t=GameMaker_Language%2FGML_Reference%2FDrawing%2FLighting%2FLighting.htm (21.05.2024).

GameMaker (o. J.g): GameMaker Manual - Local Asset Packages. URL: https://manual.gamemaker.io/monthly/en/index.htm?#t=IDE_Tools%2FLocal_Asset_Packages.htm (21.05.2024).

GameMaker (o. J.h): GameMaker Manual - Parent Objects. URL: https://manual.gamemaker.io/monthly/en/index.htm?#t=The_Asset_Editors%2FObject_Properties%2FParent_Objects.htm (21.05.2024).

GameMaker (o. J.i): GameMaker Manual - Rooms. URL: https://manual.gamemaker.io/monthly/en/#t=GameMaker_Language%2FGML_Reference%2FAsset_Management%2FRooms%2FRooms.htm (21.05.2024).

GameMaker (o. J.j): GameMaker Manual - Sprites. URL: https://manual.gamemaker.io/monthly/en/index.htm?#t=GameMaker_Language%2FGML_Reference%2FAsset_Management%2FSprites%2FSprites.htm (21.05.2024).

GameMaker (o. J.k): GameMaker Manual - The Object Editor. URL: https://manual.gamemaker.io/monthly/en/index.htm?#t=The_Asset_Editors%2FObjects.htm (21.05.2024).

GameMaker (o. J.l): GameMaker Preisgestaltung. URL: https://gamemaker.io/de/get (21.05.2024).

GameMaker (o. J.m): GameMaker-Tutorials, lernen, wie man ein Spiel macht. URL: https://gamemaker.io/de/tutorials?page=1&levels=%5B%225fdcb7f12becb500065ed443%22%5D&topics=%5B%5D&versions=%5B%5D&types=%5B%5D (21.05.2024).

GameMaker (o. J.n): GML Code Overview. URL: https://manual.gamemaker.io/monthly/en/GameMaker_Language/GML_Overview/GML_Overview.htm (21.05.2024).

GameMaker (o. J.o): Standard Workflow. URL: https://manual.gamemaker.io/monthly/en/IDE_Tools/Source_Control/Standard_Workflow.htm (21.05.2024).

GameMaker (o. J.p): GML Visual. URL: https://manual.gamemaker.io/monthly/en/Drag_And_Drop/Drag_And_Drop_Index.htm (27.05.2024).

Gamerant (o. J.): Unity Changes How Fees Work After Backlash. URL: https://gamerant.com/unity-licensing-fee-backlash-change/ (21.05.2024).

Gamesmap (o. J.): gamesmap.de. URL: https://www.gamesmap.de/?v=1&c=51-9-10-8-12-14-56-13 (21.05.2024).

GDQuest (o. J.): GDQuest - YouTube. URL: https://www.youtube.com/@Gdquest/videos (22.05.2024).

GIGA (2019): Was ist Steam? Einfach erklärt. URL: https://www.giga.de/downloads/steam/specials/was-ist-steam/ (27.05.2024).

GIGA (2021): Was ist Java? – einfach erklärt. URL: https://www.giga.de/artikel/was-ist-java-einfach-erklaert/ (27.05.2024).

GitHub (o. J.): Informationen zu Issues - GitHub-Dokumentation. URL: https://docs.github.com/de/issues/tracking-your-work-with-issues/about-issues (27.05.2024).

Git-scm (o. J.a): Git. URL: https://git-scm.com/ (21.05.2024).

Git-scm (o. J.b): Git - gitattributes Documentation. URL: https://git-scm.com/docs/gitattributes (27.05.2024).

Git-scm (o. J.c): Git - gitignore Documentation. URL: https://git-scm.com/docs/gitignore (27.05.2024).

Godot (o. J.a): Animation. URL: https://docs.godotengine.org/en/stable/tutorials/animation/tutorials/animation/index.html (22.05.2024).

Godot (o. J.b): Applying object-oriented principles in Godot. URL: https://docs.godotengine.org/en/4.0/tutorials/best_practices/tutorials/best_practices/what_are_godot_classes.html (22.05.2024).

Godot (o. J.c): Audio. URL: https://docs.godotengine.org/en/stable/tutorials/audio/tutorials/audio/index.html (22.05.2024).

Godot (o. J.d): Available 3D formats. URL: https://docs.godotengine.org/en/stable/tutorials/assets_pipeline/importing_3d_scenes/tutorials/assets_pipeline/importing_3d_scenes/available_formats.html (22.05.2024).

Godot (o. J.e): Best practices. URL: https://docs.godotengine.org/en/stable/tutorials/best_practices/tutorials/best_practices/index.html (22.05.2024).

Godot (o. J.f): Building from source. URL: https://docs.godotengine.org/en/stable/contributing/development/compiling/contributing/development/compiling/index.html (22.05.2024).

Godot (o. J.g): Download for Windows. URL: https://godotengine.org/download/windows/ (22.05.2024).

Godot (o. J.h): Frequently asked questions. URL: https://docs.godotengine.org/en/stable/about/about/faq.html (22.05.2024).

Godot (o. J.i): GDScript reference. URL: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/tutorials/scripting/gdscript/gdscript_basics.html (22.05.2024).

Godot (o. J.j): Godot Docs – 4.2 branch. URL: https://docs.godotengine.org/en/stable/index.html (22.05.2024).

Godot (o. J.k): Importing audio samples. URL: https://docs.godotengine.org/en/stable/tutorials/assets_pipeline/tutorials/assets_pipeline/importing_audio_samples.html (22.05.2024).

Godot (o. J.l): Importing images. URL: https://docs.godotengine.org/en/stable/tutorials/assets_pipeline/tutorials/assets_pipeline/importing_images.html (22.05.2024).

Godot (o. J.m): Importing translations. URL: https://docs.godotengine.org/en/stable/tutorials/assets_pipeline/tutorials/assets_pipeline/importing_translations.html (22.05.2024).

Godot (o. J.n): Internationalizing games. URL: https://docs.godotengine.org/en/stable/tutorials/i18n/tutorials/i18n/internationalizing_games.html (22.05.2024).

Godot (o. J.o): Introduction. URL: https://docs.godotengine.org/en/stable/getting_started/introduction/getting_started/introduction/index.html (22.05.2024).

Godot (o. J.p): List of features. URL: https://docs.godotengine.org/en/stable/about/about/list_of_features.html (22.05.2024).

Godot (o. J.q): Overview of Godot’s key concepts. URL: https://docs.godotengine.org/en/4.0/getting_started/introduction/getting_started/introduction/key_concepts_overview.html (22.05.2024).

Godot (o. J.r): Overview of Godot’s key concepts. URL: https://docs.godotengine.org/en/4.0/getting_started/introduction/key_concepts_overview.html#nodes (22.05.2024).

Godot (o. J.s): User interface (UI). URL: https://docs.godotengine.org/en/stable/tutorials/ui/tutorials/ui/index.html (22.05.2024).

Godot (o. J.t): Version control systems. URL: https://docs.godotengine.org/en/stable/tutorials/best_practices/tutorials/best_practices/version_control_systems.html (22.05.2024).

Godot (o. J.u): Your first 2D game. URL: https://docs.godotengine.org/en/stable/getting_started/first_2d_game/getting_started/first_2d_game/index.html (22.05.2024).

Godot Engine (o. J.): Godot Engine - YouTube. URL: https://www.youtube.com/@GodotEngineOfficial/videos (22.05.2024).

Gomez, Juan S. (2024): Collaborating with Multi-User Editing in Unreal Engine | Tutorial. URL: https://dev.epicgames.com/community/learning/tutorials/7Jx6/collaborating-with-multi-user-editing-in-unreal-engine (22.05.2024).

Google (2024): google/flax. URL: https://github.com/google/flax (21.05.2024).

Halsas, Rasmus (2017): Comparison of HTML5 game engines used in game development. URL: https://lutpub.lut.fi/handle/10024/130917 (20.02.2024).

HNU (o. J.): Abschlussarbeiten - HNU intern. URL: https://intern.hnu.de/einrichtungen/referate-studium-pruefung/pruefung-praktikum/abschlussarbeiten (27.05.2024).

Ierusalimschy, Roberto/Figueiredo, Luiz Henrique de/Celes, Waldemar (o. J.): The evolution of an extension language: a history of Lua. URL: https://www.lua.org/history.html (27.05.2024).

Interkantonale Lehrmittelzentrale (o. J.): Lehrmittelevaluation | ilz. URL: https://www.ilz.ch/lehrmittel/lehrmittelevaluation (21.05.2024).

Itch (o. J.): Obscured by the Night by mrcobalt124. URL: https://mrcobalt124.itch.io/obscured-by-the-night (21.05.2024).

Jead (2024): What platforms are supported by Unity? URL: https://support.unity.com/hc/en-us/articles/206336795-What-platforms-are-supported-by-Unity (28.05.2024).

jMonkeyEngine (2023): jMonkeyEngine. URL: https://jmonkeyengine.org/ (22.05.2024).

jMonkeyEngine (o. J.a): jMonkeyEngine Docs. URL: https://wiki.jmonkeyengine.org/docs/3.4/sdk/application_deployment.html (22.05.2024).

jMonkeyEngine (o. J.b): jMonkeyEngine Docs - File Formats. URL: https://wiki.jmonkeyengine.org/docs/3.4/getting-started/features.html (22.05.2024).

jMonkeyEngine (o. J.c): jMonkeyEngine Docs - Hello Animation. URL: https://wiki.jmonkeyengine.org/docs/3.4/tutorials/beginner/hello_animation.html (22.05.2024).

jMonkeyEngine (o. J.d): jMonkeyEngine Docs - Hello Effects. URL: https://wiki.jmonkeyengine.org/docs/3.4/tutorials/beginner/hello_effects.html (22.05.2024).

jMonkeyEngine (o. J.e): jMonkeyEngine Docs - Hello Node. URL: https://wiki.jmonkeyengine.org/docs/3.4/tutorials/beginner/hello_node.html (22.05.2024).

jMonkeyEngine (o. J.f): jMonkeyEngine Docs - Hello Terrain. URL: https://wiki.jmonkeyengine.org/docs/3.4/tutorials/beginner/hello_terrain.html (22.05.2024).

jMonkeyEngine (o. J.g): jMonkeyEngine Docs - SimpleApplication. URL: https://wiki.jmonkeyengine.org/docs/3.4/core/app/simpleapplication.html (22.05.2024).

jMonkeyEngine (o. J.h): jMonkeyEngine Docs: Pivot Node. URL: https://wiki.jmonkeyengine.org/docs/3.4/tutorials/beginner/hello_node.html#what-is-a-pivot-node (22.05.2024).

jMonkeyEngine (o. J.i): jMonkeyEngine Docs: SpiderMonkey. URL: https://wiki.jmonkeyengine.org/docs/3.4/networking/networking.html (22.05.2024).

jMonkeyEngine (o. J.j): jMonkeyEngine Docs: Version Control. URL: https://wiki.jmonkeyengine.org/docs/3.4/sdk/version_control.html (22.05.2024).

jMonkeyEngine (o. J.k): Release jMonkeyEngine 3.6.1-stable · jMonkeyEngine/jmonkeyengine. URL: https://github.com/jMonkeyEngine/jmonkeyengine/releases/tag/v3.6.1-stable (22.05.2024).

jMonkeyEngine (o. J.l): Releases · jMonkeyEngine/sdk. URL: https://github.com/jMonkeyEngine/sdk/releases (22.05.2024).

Kaiser, Robert (2021): Qualitative Experteninterviews - Konzeptionelle Grundlagen und praktische Durchführung. 2. Auflage. Wiesbaden: Springer VS.

Kinsta (2023): Was ist GitHub? Eine Einführung in GitHub für Einsteiger. URL: https://kinsta.com/de/wissensdatenbank/was-ist-github/ (27.05.2024).

Klinge, Heiko (2020): Trackmania im Test: Warum der Free2Play-Neustart (noch) nicht aufgeht. URL: https://www.gamestar.de/artikel/trackmania-test,3359446.html (27.05.2024).

kseniias (2022): Opera presents GX.games, a new platform to create and publish mobile games to Mobile Web on iOS and Android at once, for free. URL: https://press.opera.com/2022/07/29/opera-presents-gx-games/ (27.05.2024).

Landesmedienzentrum Baden-Württemberg (o. J.): Medienbegutachtung. URL: https://www.lmz-bw.de/medienbegutachtung-1 (21.05.2024).

Matharoo, Gurpreet S. (2024): GameMaker Update 2024.2 Now Available | GameMaker. URL: https://gamemaker.io/de/blog/release-2024-2 (21.05.2024).

Mathes, Markus A./Seufert, Jochen (2022): Programmierparadigmen. In: Mathes, Prof. Dr. Markus A./Seufert, Prof. Dr. Jochen (Hrsg.): Programmieren in C++ für Elektrotechniker und Mechatroniker: Das Lern- und Übungsbuch. Wiesbaden: Springer Fachmedien, S. 7–10.

MAXQDA (o. J.): MAXQDA | Die #1 Software für Qualitative & Mixed-Methods-Forschung. URL: https://www.maxqda.com/de/ (21.05.2024).

Mayring, Philipp (2022): Qualitative Inhaltsanalyse - Grundlagen und Techniken. 13. Auflage. Weinheim ; Basel: Beltz.

McCall, Vivian (o. J.): What is Epic Games? Here's what you need to know about the game developer and distributor behind the success of „Fortnite". URL: https://www.businessinsider.com/guides/tech/what-is-epic-games (27.05.2024).

Muir, Michael (2022): Versioning and Source Control | Tutorial. URL: https://dev.epicgames.com/community/learning/tutorials/jO2m/unreal-engine-versioning-and-source-control (22.05.2024).

Nielsen, Jakob (1993): Usability Engineering. Boston: Academic Press.

Nielsen Norman Group (o. J.): 10 Usability Heuristics for User Interface Design. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (21.05.2024).

NitzzSea (o. J.): NitzzSea - YouTube. URL: https://www.youtube.com/@nitzzsea4444/videos (22.05.2024).

Ong, Jeremy (o. J.): How I Evaluate Game Engines – Jeremy's Blog. URL: https://www.jeremyong.com/game%20engines/2023/09/15/how-i-evaluate-game-engines/ (20.02.2024).

OpenAI (2024): openai/whisper. URL: https://github.com/openai/whisper (21.05.2024).

Parrish, Ash (2023): Unity has changed its pricing model, and game developers are pissed off. URL: https://www.theverge.com/2023/9/12/23870547/unit-price-change-game-development (21.05.2024).

Pattrasitidecha, Akekarat (2014): Comparison and evaluation of 3D mobile game engines.

Perforce (o. J.): Perforce Software | Development Tools For Innovation at Scale. URL: https://www.perforce.com/ (27.05.2024).

Petridis, Panagiotis/Dunwell, Ian/Panzoli, David (2012): Game Engines Selection Framework for High-Fidelity Serious Applications. URL: https://ibimapublishing.com/articles/IJIW/2012/418638/418638.pdf (20.02.2024).

Pierce, Michael (2022): Software Release Life Cycle (SRLC): Understand The 6 Main Stages. URL: https://theproductmanager.com/topics/software-release-life-cycle/ (21.05.2024).

Playstation (o. J.): Dreams™. URL: https://www.playstation.com/de-de/games/dreams (30.05.2024).

Qualtrics (o. J.): Likert-Skala: Definition, Beispiel und Vorteile. URL: https://www.qualtrics.com/de/erlebnismanagement/marktforschung/likert-skala/ (21.05.2024).

/r/cocoscreator (o. J.): Cocos Creator Game Development. URL: https://www.reddit.com/r/CocosCreator/ (22.05.2024).

/r/defold (o. J.): Defold Game Engine. URL: https://www.reddit.com/r/defold/new/ (22.05.2024).

Red Hat (o. J.): Was ist Open Source? Definition, Funktionsweise und Vorteile. URL: https://www.redhat.com/de/topics/open-source/what-is-open-source (27.05.2024).

/r/gamemaker (o. J.): /r/gamemaker. URL: https://www.reddit.com/r/gamemaker/new/ (21.05.2024).

/r/godot (o. J.): Godot: your free, open-source Game Engine. URL: https://www.reddit.com/r/godot/new/ (22.05.2024).

/r/jMonkeyEngine (o. J.): Subreddit dedicated for jMonkeyEngine. URL: https://www.reddit.com/r/jMonkeyEngine/new/ (22.05.2024).

Rosenberg, Adam (2021): What is Itch.io? This oddball indie games store is a vital source of creativity. URL: https://mashable.com/article/itchio-what-is-it (27.05.2024).

RubaDev (o. J.): RubaDev - YouTube. URL: https://www.youtube.com/@RubaDev/videos (22.05.2024).

/r/unity (o. J.): Unity 3D, 2D, VR/AR & Video Engine | News, Chats, Questions, Projects & Info. URL: https://www.reddit.com/r/unity/ (21.05.2024).

/r/unrealengine (o. J.): Unreal Engine. URL: https://www.reddit.com/r/unrealengine/new/ (22.05.2024).

schau-hin (o. J.): Roblox: Was steckt hinter dem beliebten Online-Spiel? – SCHAU HIN! URL: https://www.schau-hin.info/grundlagen/roblox-was-steckt-hinter-dem-beliebten-online-spiel (27.05.2024).

Statista (o. J.): Betriebssysteme - Marktanteile in Deutschland 2024. URL: https://de.statista.com/statistik/daten/studie/158102/umfrage/marktanteile-von-betriebssystemen-in-deutschland-seit-2009/ (21.05.2024).

StayAtHomeDev (o. J.): StayAtHomeDev - YouTube. URL: https://www.youtube.com/@stayathomedev/videos (22.05.2024).

SteamDB (o. J.a): Turnip Boy Robs a Bank Price history. URL: https://steamdb.info/app/2097230/ (21.05.2024).

SteamDB (o. J.b): 山河旅探 Murders on the Yangtze River · Murders on the Yangtze River Price history. URL: https://steamdb.info/app/1746030/ (21.05.2024).

SteamDB (o. J.c): 20 Small Mazes. URL: https://steamdb.info/app/2570630/ (22.05.2024).

SteamDB (o. J.d): Craftomation 101 · Craftomation 101: Programming & Craft Price history. URL: https://steamdb.info/app/1724140/ (22.05.2024).

SteamDB (o. J.e): Mystery Society 2: Hidden Puzzles · SteamDB. URL: https://steamdb.info/app/2367650/ (22.05.2024).

SteamDB (o. J.f): Palworld Price history. URL: https://steamdb.info/app/1623730/ (22.05.2024).

Talend (o. J.): Was ist eine API? Einfach erklärt! | Talend. URL: https://www.talend.com/de/resources/was-ist-eine-api/ (27.05.2024).

thoughbot (2017): How to Git with Unity. URL: https://thoughtbot.com/blog/how-to-git-with-unity (22.05.2024).

Unfolding Gamedev (o. J.): Unfolding Gamedev - YouTube. URL: https://www.youtube.com/@unfolding_gamedev/videos (22.05.2024).

Unity (2024): Level up your code with game programming design patterns: Object pool | Tutorial. URL: https://www.youtube.com/watch?v=U08ScgT3RVM (21.05.2024).

Unity (o. J.): Unity - YouTube. URL: https://www.youtube.com/@unity (21.05.2024).

Unity Technologies (2023a): Unity plan pricing and packaging updates. URL: https://blog.unity.com/news/plan-pricing-and-packaging-updates (21.05.2024).

Unity Technologies (2023b): An open letter to our community. URL: https://blog.unity.com/news/open-letter-on-runtime-fee (21.05.2024).

Unity Technologies (o. J.a): Abonnements und Preise. URL: https://unity.com/products (21.05.2024).

Unity Technologies (o. J.b): Changes to pricing and Unity plans FAQ | Unity. URL: https://unity.com/pricing-updates (21.05.2024).

Unity Technologies (o. J.c): Echtzeit-Entwicklungsplattform von Unity | 3D, 2D, VR- und AR-Engine. URL: https://unity.com/ (21.05.2024).

Unity Technologies (o. J.d): Get started with the Unity Hub. URL: https://learn.unity.com/tutorial/get-started-with-the-unity-hub (21.05.2024).

Unity Technologies (o. J.e): Learn game development w/ Unity | Courses & tutorials in game design, VR, AR, & Real-time 3D | Unity Learn. URL: https://learn.unity.com/ (21.05.2024).

Unity Technologies (o. J.f): Runtime Fee Estimator. URL: https://unity.com/runtime-fee-estimator (21.05.2024).

Unity Technologies (o. J.g): Unity Essentials Pathway. URL: https://learn.unity.com/learn/pathway/unity-essentials (21.05.2024).

Unity Technologies (o. J.h): Unity QA - LTS Releases - Unity. URL: https://unity.com/releases/editor/qa/lts-releases?version=2017.1 (21.05.2024).

Unity Technologies (o. J.i): Scalable DevOps Services & Solutions | Unity. URL: https://unity.com/products/unity-devops (22.05.2024).

Unity Technologies (o. J.j): Unity - Manual: Animation. URL: https://docs.unity3d.com/Manual/AnimationSection.html (22.05.2024).

Unity Technologies (o. J.k): Unity - Manual: Audio. URL: https://docs.unity3d.com/Manual/Audio.html (22.05.2024).

Unity Technologies (o. J.l): Unity - Manual: GameObjects. URL: https://docs.unity3d.com/Manual/GameObjects.html (22.05.2024).

Unity Technologies (o. J.m): Unity - Manual: Lighting. URL: https://docs.unity3d.com/Manual/LightingOverview.html (22.05.2024).

Unity Technologies (o. J.n): Unity - Manual: Packages and feature sets. URL: https://docs.unity3d.com/Manual/PackagesList.html (22.05.2024).

Unity Technologies (o. J.o): Unity - Manual: Prefabs. URL: https://docs.unity3d.com/Manual/Prefabs.html (22.05.2024).

Unity Technologies (o. J.p): Unity - Manual: Profiler overview. URL: https://docs.unity3d.com/Manual/Profiler.html (22.05.2024).

Unity Technologies (o. J.q): Unity - Manual: Scenes. URL: https://docs.unity3d.com/Manual/CreatingScenes.html (22.05.2024).

Unity Technologies (o. J.r): Unity - Manual: ScriptableObject. URL: https://docs.unity3d.com/Manual/class-ScriptableObject.html (22.05.2024).

Unity Technologies (o. J.s): Unity - Manual: TextMeshPro. URL: https://docs.unity3d.com/Manual/com.unity.textmeshpro.html (22.05.2024).

Unity Technologies (o. J.t): Unity - Scripting API: MonoBehaviour. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html (22.05.2024).

Unity Technologies (o. J.u): Unity Visual Scripting | Unity. URL: https://unity.com/features/unity-visual-scripting (22.05.2024).

Unity Technologies (o. J.v): Unity - Manual: System requirements for Unity 2022.3. URL: https://docs.unity3d.com/Manual/system-requirements.html (28.05.2024).

Unity Technologies, Unity (o. J.w): Unity - Manual: Supported Asset Types. URL: https://docs.unity3d.com/Manual/AssetTypes.html (22.05.2024).

Unity Technologies, Unity (o. J.x): Start Your Creative Projects and Download the Unity Hub | Unity. URL: https://unity.com/download (28.05.2024).

Unreal Sensei (o. J.): Unreal Sensei - YouTube. URL: https://www.youtube.com/@UnrealSensei/videos (22.05.2024).

Vektoria (o. J.): Projekte mit der Game-Engine „Vektoria“ - YouTube. URL: https://www.youtube.com/playlist?list=PL3p2XiE_ark7ZGT3GUDlqwLmb3pFP0sUH (29.05.2024).

W3Schools (o. J.a): C++ Introduction. URL: https://www.w3schools.com/cpp/cpp_intro.asp (27.05.2024).

W3Schools (o. J.b): Introduction to Python. URL: https://www.w3schools.com/python/python_intro.asp (27.05.2024).

W3Schools (o. J.c): JavaScript Introduction. URL: https://www.w3schools.com/js/js_intro.asp (27.05.2024).

W3Schools (o. J.d): TypeScript Introduction. URL: https://www.w3schools.com/typescript/typescript_intro.php (27.05.2024).

Weimann, Jason (o. J.): Jason Weimann - YouTube. URL: https://www.youtube.com/@Unity3dCollege (21.05.2024).

Westerdahl, Mathias (2021): Defold 1.2.184 has been released - Release notes. URL: https://forum.defold.com/t/defold-1-2-184-has-been-released/68738 (30.05.2024).

Westerdahl, Mathias (2024): Defold 1.6.4 has been released - Release notes. URL: https://forum.defold.com/t/defold-1-6-4-has-been-released/75979 (22.05.2024).

Word (o. J.): Kostenlose Onlinebearbeitung von Dokumenten mit Microsoft Word | Microsoft 365. URL: https://www.microsoft.com/de-de/microsoft-365/word (23.05.2024).

Yumpu.com (o. J.): Kriterien zur Beurteilung von Lernmitteln - Landesinstitut für ... URL: https://www.yumpu.com/de/document/read/7692000/kriterien-zur-beurteilung-von-lernmitteln-landesinstitut-fur- (11.02.2024).

Zoom (o. J.): Zentrale Verbindungsplattform. URL: https://zoom.us/de (23.05.2024).

Zotero (o. J.): Zotero | Your personal research assistant. URL: https://www.zotero.org/ (23.05.2024).

Anhang

Anhang 1 – Nicht-Qualifizierte Game Engines

Die folgende Tabelle zeigt alle Game Engines, die auf EnginesDatabase.com gelistet sind, aber eines oder mehrere der in dieser Arbeit aufgestellten Qualifikationskriterien nicht erfüllen, sowie eine zugehörige Begründung. Falls eine Game Engine mehr als ein Qualifikationskriterium nicht erfüllt ist dennoch nur das erste aufgelistet.

Name	**Nicht erfülltes Qualifikationskriterium**	**Grund**
001 Game Creator	Definitionstreue	Keine Programmiersprachen-Unterstützung
Arc80	Preisgestaltung	Festpreis
Arcade	Definitionstreue	Auf 2D beschränkt
Armory3D	Definitionstreue	Keine eigenständige Software
Axmol Engine	Definitionstreue	Keine visuelle Benutzeroberfläche
Babylon.js	Definitionstreue	Keine eigenständige Software
BennuGD	Definitionstreue	Auf 2D beschränkt
Bevy	Definitionstreue	Kein Stable Release verfügbar
Bitsy	Definitionstreue	Keine eigenständige Software
Bitty Engine	Definitionstreue	Auf 2D beschränkt
Blitz3D	Definitionstreue	Keine Game Engine
Buildbox	Definitionstreue	Keine Programmiersprachen-Unterstützung
C4 Engine	Preisgestaltung	Festpreis
Calico	Definitionstreue	Auf Genre beschränkt (Interactive Fiction)
Castle Game Engine	Definitionstreue	Noch in der Entwicklung
Clickteam Fusion	Definitionstreue	Auf 2D beschränkt
Cocos2D-X	Definitionstreue	Auf 2D beschränkt
Construct3	Preisgestaltung	Festpreis
CopperCube	Definitionstreue	Keine Programmiersprachen-Unterstützung
Core	Definitionstreue	Noch in der Entwicklung
Crown	Definitionstreue	Noch in der Entwicklung
CryEngine	Entwicklungsplattform	Nicht für *MacOS* verfügbar

Name	**Nicht erfülltes Qualifikationskriterium**	**Grund**
ct.js	Definitionstreue	Auf 2D beschränkt
Dagor Engine	Definitionstreue	Kein Stable Release verfügbar
Dark Basic	Definitionstreue	Keine Game Engine
Decker	Definitionstreue	Keine Game Engine
DOME	Definitionstreue	Auf 2D beschränkt
DragonRuby Game Toolkit	Definitionstreue	Auf 2D beschränkt
DungeonScript	Definitionstreue	Keine eigenständige Software
Dust Engine	Definitionstreue	Kein Stable Release verfügbar
Ebitengine	Definitionstreue	Auf 2D beschränkt
Echo	Definitionstreue	Kein Stable Release verfügbar
Eldiron	Definitionstreue	Auf Genre beschränkt (RPG)
Emerald	Definitionstreue	Auf 2D beschränkt
Engine.lol	Definitionstreue	Keine eigenständige Software
ENIGMA	Definitionstreue	Auf 2D beschränkt
Evergine	Entwicklungsplattform	Nicht für *MacOS* verfügbar
Excalibur.js	Definitionstreue	Auf 2D beschränkt
ezEngine	Entwicklungsplattform	Nicht für *MacOS* verfügbar
FalcoEngine	Entwicklungsplattform	Nicht für *MacOS* verfügbar
Flame	Definitionstreue	Auf 2D beschränkt
FlatRedBall	Definitionstreue	Auf 2D beschränkt
Flowlab	Definitionstreue	Keine Programmiersprachen-Unterstützung
FNA	Entwicklungsplattform	Nicht für *MacOS* verfügbar
FWK	Aktualität	Keine aktuelle Veröffentlichung
Fyrox	Aktualität	Keine aktuelle Veröffentlichung
Game Guru MAX	Definitionstreue	Keine Programmiersprachen-Unterstützung
GameSalad	Definitionstreue	Keine Programmiersprachen-Unterstützung
GB Studio	Definitionstreue	Keine Programmiersprachen-Unterstützung
GBDK-2020	Definitionstreue	Keine Game Engine

Name	Nicht erfülltes Qualifikationskriterium	Grund
GDevelop	Definitionstreue	Keine Programmiersprachen-Unterstützung
GDevelop	Definitionstreue	Keine Programmiersprachen-Unterstützung
GGEZ	Definitionstreue	Auf 2D beschränkt
Gideros Mobile	Zielplattform	Auf Mobile beschränkt
Gosu	Definitionstreue	Auf 2D beschränkt
Harfang	Definitionstreue	Keine Game Engine
HaxeFlixel	Definitionstreue	Auf 2D beschränkt
Heaps.io	Defintionstreue	Keine Game Engine
Hot Soup Processor	Definitionstreue	Keine Game Engine
Hotham	Definitionstreue	Auf VR beschränkt
Ikemen GO	Definitiontreue	Auf Genre beschränkt (Fighting Games)
Indigo	Verfügbarkeit	Noch in der Entwicklung
Inform	Definitionstreue	Keine Game Engine
Intersect Engine	Definitionstreue	Auf Genre beschränkt (MMORPG)
Irrlicht Engine	Aktualität	Keine aktuelle Veröffentlichung
Isogenic	Definitionstreue	Auf 2D beschränkt
Ji-Xpress	Definitionstreue	Auf 2D beschränkt
Kaboom.js	Definitionstreue	Keine Game Engine
Kirikiri Z	Definitionstreue	Kein Stable Release verfügbar
Kohi	Verfügbarkeit	Noch in der Entwicklung
KorGE	Definitionstreue	Auf 2D beschränkt
Leadwerks	Preisgestaltung	Festpreis
LibGDX	Definitionstreue	Auf 2D beschränkt
LIKO-12	Definitionstreue	Auf 2D beschränkt
Love2D	Definitionstreue	Auf 2D beschränkt
LÖVR	Definitionstreue	Keine Game Engine
luxe	Definitionstreue	Kein Stable Release verfügbar
LWJGL	Definitionstreue	Keine Game Engine
Mach	Definitionstreue	Kein Stable Release verfügbar
Macroquad	Definitionstreue	Auf 2D beschränkt

Name	**Nicht erfülltes Qualifikationskriterium**	**Grund**
MANU	Definitionstreue	Keine Programmiersprachen-Unterstützung
Maratis	Aktualität	Keine aktuelle Veröffentlichung
Meg-4	Definitionstreue	Auf 2D beschränkt
MelonJS	Definitionstreue	Auf 2D beschränkt
Microsoft MakeCode	Definitionstreue	Keine Game Engine
microStudio	Definitionstreue	Auf 2D beschränkt
miniGDX	Aktualität	Keine aktuelle Veröffentlichung
MiniGL	Definitionstreue	Auf 2D beschränkt
Modd.io	Definitionstreue	Keine eigenständige Software
MonoGame	Definitionstreue	Keine eigenständige Software
Murder Engine	Definitionstreue	Keine eigenständige Software
NarratEngine	Definitionstreue	Auf Genre beschränkt (Narrative Games)
NeoAxis Engine	Entwicklungsplattform	Nicht für *MacOS* verfügbar
NESMaker	Definitionstreue	Auf 2D beschränkt
Next2D	Definitionstreue	Auf 2D beschränkt
Nu Game Engine	Entwicklungsplattform	Nicht für Mac
O3DE	Entwicklungsplattform	Nicht für Mac
Octo	Definitionstreue	Auf 2D beschränkt
Ogre3D	Definitionstreue	Keine Game Engine
OpenFrameworks	Definitionstreue	Keine Game Engine
ORX	Definitionstreue	Auf 2D beschränkt
Oxyengine	Definitionstreue	Auf 2D beschränkt
Panda3D	Entwicklungsplattform	Nicht für *MacOS* verfügbar
Phaser	Definitionstreue	Auf 2D beschränkt
PICO-8	Definitionstreue	Auf 2D beschränkt
Pixel	Definitionstreue	Keine Game Engine
Pixel Game Maker MV	Definitionstreue	Auf 2D beschränkt
Pixelbox.js	Definitionstreue	Auf 2D beschränkt
PixiJS	Definitionstreue	Auf 2D beschränkt
Planimeter	Definitionstreue	Auf 2D beschränkt
PlayCanvas	Definitionstreue	Keine eigenständige Software

Name	**Nicht erfülltes Qualifikationskriterium**	**Grund**
Polycode	Aktualität	Keine aktuelle Veröffentlichung
PuzzleScript	Definitionstreue	Auf 2D beschränkt
PX8	Definitionstreue	Auf 2D beschränkt
PyGame	Definitionstreue	Keine Game Engine
Pyglet	Definitionstreue	Keine Game Engine
Pyxel	Definitionstreue	Auf 2D beschränkt
QX82	Definitionstreue	Auf 2D beschränkt
Range Engine	Entwicklungsplattform	Nicht für *MacOS* verfügbar
Rav Engine	Definitionstreue	Kein Stable Release verfügbar
Raylib	Definitionstreue	Keine Game Engine
RealityKit	Entwicklungsplattform	Nur für *MacOS* verfügbar
Rebel Fork	Definitionstreue	Noch in der Entwicklung
Ren'Py	Definitionstreue	Auf 2D beschränkt
Retro Puzzle Maker	Definitionstreue	Keine Game Engine
Rogue Engine	Definitionstreue	Noch in der Entwicklung
Rootex	Entwicklungsplattform	Nicht für *MacOS* verfügbar
RPG Bakin	Definitionstreue	Auf Genre beschränkt (RPG)
RPG in a box	Definitionstreue	Keine Programmiersprachen-Unterstützung
RPG Maker MZ	Definitionstreue	Auf Genre beschränkt (RPG)
RPG Paper Maker	Definitionstreue	Auf Genre beschränkt (RPG)
Scratch	Definitionstreue	Keine eigenständige Software
seccia.dev	Definitionstreue	Auf 2D beschränkt
Simple	Entwicklungsplattform	Nicht für *MacOS* verfügbar
Snax	Entwicklungsplattform	Nicht für *MacOS* verfügbar
Solar2D	Definitionstreue	Auf 2D beschränkt
Solarus	Definitionstreue	Auf 2D beschränkt
Source	Aktualität	Keine aktuelle Veröffentlichung
SpriteKit	Entwicklungsplattform	Nicht für *MacOS* verfügbar
SpriteWidget	Definitionstreue	Keine Game Engine
Stencyl	Definitionstreue	Auf 2D beschränkt
Storm Engine	Aktualität	Keine aktuelle Veröffentlichung
Stride	Entwicklungsplattform	Nicht für *MacOS* verfügbar

Name	**Nicht erfülltes Qualifikationskriterium**	**Grund**
SUGAR	Definitionstreue	Auf 2D beschränkt
Superpowers	Aktualität	Keine aktuelle Veröffentlichung
Taylor	Definitionstreue	Auf 2D beschränkt
T-Engine4	Definitionstreue	Auf Genre beschränkt (Roguelikes)
Tetra	Definitionstreue	Auf 2D beschränkt
The Mirror	Definitionstreue	Noch in der Entwicklung
The Sandbox	Definitionstreue	Keine Programmiersprachen-Unterstützung
Three.js	Definitionstreue	Keine Game Engine
TIC-80	Definitionstreue	Keine Game Engine
Tinychoice	Definitionstreue	Auf Genre beschränkt (Text-Adventure)
Titan Engine	Definitionstreue	Keine Game Engine
Torque3D	Aktualität	Keine aktuelle Veröffentlichung
Trial Game Engine	Entwicklungsplattform	Nicht für *MacOS* verfügbar
Turbulenz	Definitionstreue	Nicht verfügbar
Turso3D	Definitionstreue	Noch in der Entwicklung
Twine	Definitionstreue	Keine Game Engine
UltraEngine	Definitionstreue	Noch in der Entwicklung
Unigine	Entwicklungsplattform	Nicht für *MacOS* verfügbar
UPBGE	Definitionstreue	Keine eigenständige Software
Urho3D	Aktualität	Keine aktuelle Veröffentlichung
Ursina Engine	Entwicklungsplattform	Nicht für *MacOS* verfügbar
Verge3D	Definitionstreue	Keine eigenständige Software
WASM-4	Definitionstreue	Auf 2D beschränkt
Welder Engine	Entwicklungsplattform	Nicht für *MacOS* verfügbar
Wick Editor	Definitionstreue	Auf 2D beschränkt
Wicked Engine	Entwicklungsplattform	Nicht für *MacOS* verfügbar
Wonderland Engine	Definitionstreue	Auf VR beschränkt
Yahaha	Definitionstreue	Keine Game Engine
Yami RPG Editor	Definitionstreue	Auf 2D beschränkt
Zero Engine	Aktualität	Keine aktuelle Veröffentlichung

Tabelle 19 Übersicht nicht-qualifizierter Game Engines

Hochschule Neu-Ulm
University of Applied Sciences

Anhang 2 – Objekte zur Überprüfung der Anwendbarkeit

3D-Objekt

Das 3D-Objekt zur Überprüfung der Import- und Export-Möglichkeiten einzelner Game Engines wurde in *Blender* erstellt und besitzt folgende Maße:

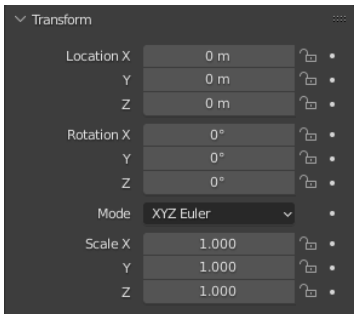

Abb. 17 Maße des 3D-Objekts zur Überprüfung der Import- und Export-Möglichkeiten einzelner Game Engines

Darüber hinaus wurden folgende Exporteinstellungen in *Blender* verwendet:

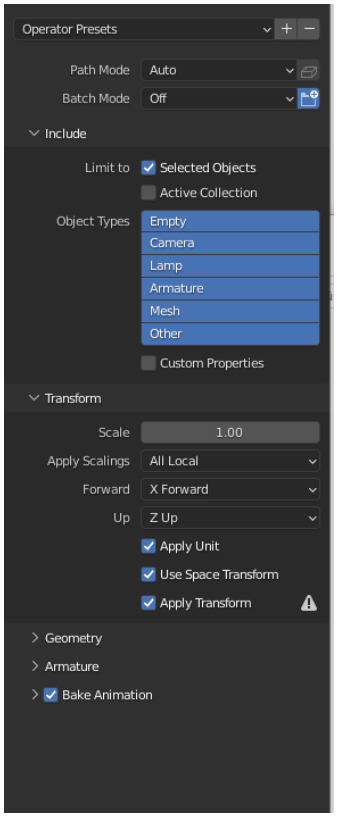

Abb. 18 Exporteinstellungen des 3D-Objekts

Das 3D-Objekt selbst ist dabei ein einfacher grauer Würfel:

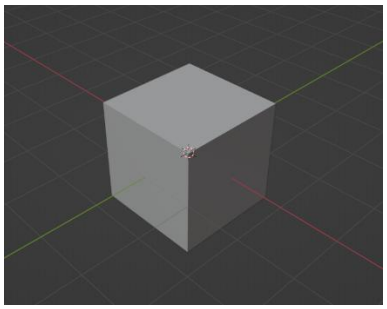

Abb. 19 3D-Objekt zur Überprüfung der Import- und Export-Möglichkeiten einzelner Game Engines

2D-Objekt

Das 2D-Objekt zur Überprüfung der Import- und Export-Möglichkeiten einzelner Game Engines wurde in *Figma* erstellt und besitzt folgende Eigenschaften:

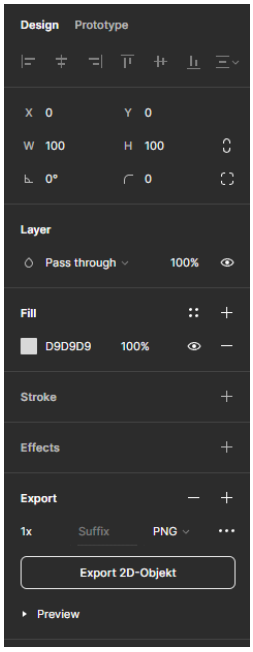

Abb. 20 Eigenschaften des 2D-Objekts zur Überprüfung der Import- und Export-Möglichkeiten einzelner Game Engines

Das 2D-Objekt selbst ist dabei ein graues Quadrat mit einem roten Schriftzug:

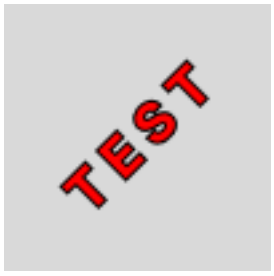

Abb. 21 2D-Objekt zur Überprüfung der Import- und Export-Möglichkeiten einzelner Game Engines

Anhang 3 – Heuristische Analyse, Kriterientabellen

Flax

Bewertungsfragen	Antworten
1	Einzelne Änderungen innerhalb des Editors werden durch das standardmäßig sichtbaren Fenster **Editor** Nutzenden direkt sichtbar gemacht. Über die Fenster **Scene** und **Properties** können zusätzliche Informationen über einzelne Spielobjekte erhalten werden, sobald diese ausgewählt wurden. Änderungen an einzelnen Spielobjekten sind sowohl im **Scene**- als auch im *Properties*-Fenster möglich und werden in beiden direkt angezeigt.
2	Einzelne Änderungen an Spielszenen oder einzelnen Spielobjekten können durch den Shortcut STRG+Z rückgängig gemacht werden. Weitere Shortcuts können innerhalb des Editors weder eingesehen, geändert noch ergänzt werden. Sobald ungespeicherte Änderungen vorliegen, werden diese (wenn auch sehr klein) im Namen des **Scene**-Fensters mit einem * angezeigt. Versucht man das Editorfenster zu schließen, während ungespeicherte Änderungen vorliegen, erhält man eine entsprechende Meldung mit den Möglichkeiten Änderungen zu speichern, nicht zu speichern und das Schließen des Editorfensters abzubrechen.
3	Sobald ein geschriebener *C#*-Programmcode nicht ausführbar ist, wird eine entsprechende Fehlermeldung klein am unteren linken Rand des Editorfensters angezeigt. Im Fenster **Output Log** wird diese ebenfalls angezeigt. Fehlermeldungen enthalten Zeile und Zeilenposition des Fehlers, sowie einen entsprechenden *C#* Compiler Error Code und je nach Fehler eine kurze Erklärung des Problems sowie einen Lösungsvorschlag. Je nach Kenntnissen der Nutzenden sind diese leichter oder schwerer zu verstehen, können aber durch das Nachschlagen des gegebenen Error Codes relativ schnell zu einer Lösung führen.
4	Unter dem Menüpunkt **Help** kann sowohl die **Documentation** sowie weitere potenzielle Hilfs-Quellen im Standardbrowser geöffnet werden.

Tabelle 20 Kriterientabelle zur Heuristischen Analyse von Flax

GameMaker

Bewertungsfragen	Antworten
1	Einzelne Änderungen an Objekten werden durch das standardmäßig sichtbaren Fenster **Workspace** Nutzenden direkt sichtbar gemacht. Über das Fenster **Inspector** können zusätzliche Informationen über einzelne Spielobjekte erhalten werden, sobald diese ausgewählt wurden. Änderungen an einzelnen Spielobjekten sind sowohl im **Workspace**- als auch im *Inspector*-Fenster möglich und werden in beiden direkt angezeigt.
2	Einzelne Änderungen an Spielszenen oder einzelnen Spielobjekten können nur bedingt den Shortcut STRG+Z rückgängig gemacht werden – während Codeänderungen hierrüber rückgängig gemacht werden können, funktioniert der Shortcut für Objektänderungen im **Workspace**-Fenster nicht. Darüber hinaus können weitere Shortcuts über die Menüpunkte **Edit → Redefine Keys** eingesehen und geändert werden. Sobald ungespeicherte Änderungen vorliegen, werden diese (wenn auch sehr klein) im Namen des Editorfensters mit einem * angezeigt. Versucht man das Editorfenster zu schließen erhält man zunächst eine Abfrage, ob dies tatsächlich geplant war. Falls darüber hinaus zusätzlich ungespeicherte Änderungen vorliegen, erhält man eine entsprechende Meldung mit den Möglichkeiten Änderungen zu speichern, nicht zu speichern und das Schließen des Editorfensters abzubrechen.
3	Sobald ein geschriebener Programmcode nicht ausführbar ist, öffnet sich ein entsprechendes Fenster in dem das Objekt, Skript sowie die Zeile, in der der Fehler gefunden wurde, angezeigt wird. Je nach Kenntnissen der Nutzenden sind diese leichter oder schwerer zu verstehen, können aber durch das Nachschlagen des gegebenen Error Codes relativ schnell zu einer Lösung führen.
4	Unter dem Menüpunkt **Help** kann sowohl das **Manual** als auch die **Knowledge Base**, das **Forum** oder weitere Hilfsangebote im Standardbrowser geöffnet werden.

Tabelle 21 Kriterientabelle zur Heuristischen Analyse von GameMaker

Unity

Bewertungsfragen	Antworten
1	Einzelne Änderungen innerhalb des Editors werden durch das standardmäßig sichtbaren Fenster **Scene** Nutzenden direkt sichtbar gemacht. Über die Fenster **Hierarchy** und **Inspector** können zusätzliche Informationen über einzelne Spielobjekte erhalten werden, sobald diese ausgewählt wurden. Änderungen an einzelnen Spielobjekten sind sowohl im **Scene**- als auch im **Inspector**-Fenster möglich und werden in beiden direkt angezeigt.
2	Einzelne Änderungen an Spielszenen oder einzelnen Spielobjekten können durch den Shortcut STRG+Z rückgängig gemacht werden. Darüber hinaus können weitere Shortcuts über die Menüpunkte **Edit → Shortcuts** eingesehen, geändert und ergänzt werden. Sobald ungespeicherte Änderungen vorliegen, werden diese (wenn auch sehr klein) im Namen des Editorfensters mit einem * angezeigt. Versucht man das Editorfenster zu schließen, während ungespeicherte Änderungen vorliegen, erhält man eine entsprechende Meldung mit den Möglichkeiten Änderungen zu speichern, nicht zu speichern und das Schließen des Editorfensters abzubrechen.
3	Sobald ein geschriebener Programmcode nicht ausführbar ist, wird eine entsprechende Fehlermeldung klein am unteren linken Rand des Editorfensters angezeigt. Im **Console**-Fenster wird diese ebenfalls angezeigt. Fehlermeldungen enthalten Zeile und Zeilenposition des Fehlers, sowie einen entsprechenden *C#* Compiler Error Code und je nach Fehler eine kurze Erklärung des Problems sowie einen Lösungsvorschlag. Je nach Kenntnissen der Nutzenden sind diese leichter oder schwerer zu verstehen, können aber durch das Nachschlagen des gegebenen Error Codes relativ schnell zu einer Lösung führen.
4	Unter dem Menüpunkt **Help** kann sowohl die **Unity Manual** als auch die **Scripting Reference** im Standardbrowser geöffnet werden.

Tabelle 22 Kriterientabelle zur Heuristischen Analyse von Unity

Unreal

Bewertungsfragen	Antworten
1	Einzelne Änderungen innerhalb des Editors werden durch das standardmäßig sichtbare Level-Fenster Nutzenden direkt sichtbar gemacht. Über die Fenster **Outliner** und **Details** können zusätzliche Informationen über einzelne Spielobjekte erhalten werden, sobald diese ausgewählt wurden. Änderungen an einzelnen Spielobjekten sind sowohl im **Details**- als auch im Level-Fenster möglich und werden in beiden direkt angezeigt.
2	Einzelne Änderungen an Spielszenen oder einzelnen Spielobjekten können durch den Shortcut STRG+Z rückgängig gemacht werden. Darüber hinaus können weitere Shortcuts über die Menüpunkte **Edit → Editor Preferences → Keyboard Shortcuts** eingesehen, geändert und ergänzt werden. Sobald ungespeicherte Änderungen an einem Objekt vorliegen, werden diese im Fenster **Content Browser** mit einem * angezeigt. Versucht man das Editorfenster zu schließen, während ungespeicherte Änderungen vorliegen, erhält man eine entsprechende Meldung sowie eine Übersicht, in welchen Dateien Änderungen vorliegen. Hierbei hat man die Möglichkeiten einzelne Änderungen zu speichern, alle Änderungen zu speichern oder zu verwerfen oder das Schließen des Editorfensters abzubrechen.
3	Sobald ein geschriebener Programmcode nicht ausführbar ist, wird eine entsprechende Fehlermeldung im Fenster **Output Log** angezeigt. Je nach Kenntnissen der Nutzenden sind diese leichter oder schwerer zu verstehen, können aber durch das Nachschlagen des gegebenen Error Codes relativ schnell zu einer Lösung führen.
4	Unter dem Menüpunkt **Help** kann sowohl die **Level Editor Documentation**, als auch die *C*++ API Reference und weitere Hilfsquellen im Standardbrowser geöffnet werden.

Tabelle 23 Kriterientabelle zur Heuristischen Analyse von Unreal Engine

Cocos Creator

Bewertungsfragen	Antworten
1	Einzelne Änderungen innerhalb des Editors werden durch das standardmäßig sichtbaren Fenster **Scene** Nutzenden direkt sichtbar gemacht. Über die Fenster **Hierarchy** und **Inspector** können zusätzliche Informationen über einzelne Spielobjekte erhalten werden, sobald diese ausgewählt wurden. Änderungen an einzelnen Spielobjekten sind sowohl im **Scene**- als auch im **Inspector**-Fenster möglich und werden in beiden direkt angezeigt.
2	Einzelne Änderungen an Spielszenen oder einzelnen Spielobjekten können durch den Shortcut STRG+Z rückgängig gemacht werden. Darüber hinaus können weitere Shortcuts über die Menüpunkte **Cocos Creator → Shortcuts** eingesehen und geändert werden. Sobald ungespeicherte Änderungen vorliegen, werden diese (wenn auch sehr klein) im Namen des Editorfensters mit einem * angezeigt. Versucht man das Editorfenster zu schließen, während ungespeicherte Änderungen vorliegen, erhält man eine entsprechende Meldung mit den Möglichkeiten Änderungen zu speichern, nicht zu speichern und das Schließen des Editorfensters abzubrechen.
3	Fehler im Programmcode werden im **Console**-Fenster angezeigt. Diese Meldungen sind aber je nach Fehler weniger hilfreich (SyntaxError: unknown, falls bei der Programmzeile `console.log(„Hello World!“);` das letzte Anführungszeichen vergessen wird), oder gar nicht erst vorhanden (beispielsweise, wenn bei der Programmzeile `console.log(„Hello World!“);` das `g` von `log` vergessen wird). Eine entsprechende Fehlerfindung und -verbesserung ist also mitunter erschwert.
4	Unter dem Menüpunkt **Help** kann sowohl das **User Manual** als auch die **API Reference** sowie das **Forum** im Standardbrowser geöffnet werden.

Tabelle 24 Kriterientabelle zur Heuristischen Analyse von Cocos Creator

Defold

Bewertungsfragen	Antworten
1	Änderungen an einzelnen Dateien sind stets mit einem * gekennzeichnet. Über das **Outline**-Fenster können zusätzliche Informationen über einzelne Spielobjekte erhalten werden, sobald die zugrundeliegende **Collection** ausgewählt wurde. Änderungen an einzelnen Spielobjekten sind sowohl im Hauptfenster als auch im **Outline**-Fenster möglich und werden in beiden direkt angezeigt.
2	Einzelne Schritte (zu denen auch das Aus- und Abwählen bestimmter Elemente gezählt werden) können über den Shortcut STRG+Z rückgängig gemacht werden. Eigene Shortcuts können zwar gesetzt werden, das muss aber über eine edn-Datei außerhalb der Game Engine geschehen, die dann unter dem Menüpunkt **File → Preferences** verlinkt werden muss.[397] Versucht man das Editorfenster zu schließen während ungespeicherte Änderungen vorliegen, erhält man eine entsprechende Meldung mit den Möglichkeiten Änderungen zu speichern, nicht zu speichern und das Schließen des Editorfensters abzubrechen.
3	Sobald ein geschriebener Programmcode nicht ausführbar ist, wird eine entsprechende Fehlermeldung im **Console**-Fenster angezeigt. Fehlermeldungen enthalten Name des Skripts, sowie die Zeile des Fehlers und eine entsprechende Fehlermeldung. Je nach Kenntnissen der Nutzenden sind diese leichter oder schwerer zu verstehen.
4	Über den Menüpunkt **Help** oder alternativ über den Shortcut F1 kann die Dokumentation im Standardbrowser geöffnet werden in der die konkret gesuchten Hilfestellungen dann noch gesucht werden müssen, weil hier zunächst unter anderem spezifische **Tutorials** und **Examples** aufgelistet sind.

Tabelle 25 Kriterientabelle zur Heuristischen Analyse von Defold

[397] vgl. Defold Foundation o. J.e

Godot

Bewertungsfragen	Antworten
1	Änderungen an einzelnen Dateien sind stets mit einem * gekennzeichnet. Über das **Inspector**-Fenster können zusätzliche Informationen über einzelne Spielobjekte erhalten werden, sobald diese ausgewählt wurde. Änderungen an einzelnen Spielobjekten sind sowohl im Hauptfenster als auch im **Inspector**-Fenster möglich und werden in beiden direkt angezeigt.
2	Einzelne Schritte können über den Shortcut STRG+Z rückgängig gemacht werden. Weitere Shortcuts können über **Editor → Editor Settings → Shortcuts** eingesehen und geändert werden. Versucht man das Editorfenster zu schließen, während ungespeicherte Änderungen vorliegen, erhält man eine entsprechende Meldung mit den Möglichkeiten Änderungen zu speichern, nicht zu speichern und das Schließen des Editorfensters abzubrechen.
3	Sobald ein geschriebener Programmcode nicht ausführbar ist, wird eine entsprechende Fehlermeldung im **Debugger**-Fenster angezeigt. Fehlermeldungen enthalten die Zeile des Fehlers und eine entsprechende Fehlermeldung. Je nach Kenntnissen der Nutzenden sind diese leichter oder schwerer zu verstehen. Darüber hinaus können einzelne Programmschritte durch das Fenster **Stack Trace** einzeln durchschritten und auf Fehler überprüft werden.
4	Über den Menüpunkt **Help** oder alternativ über den Shortcut F1 kann das Fenster **Search Help** geöffnet werden, mit den zu verschiedenen Aspekten der Game Engine Informationen gesucht werden können. Alternativ kann auch die Dokumentation oder das Forum im Standardbrowser geöffnet werden.

Tabelle 26 Kriterientabelle zur Heuristischen Analyse von Godot

jMonkeyEngine

Bewertungsfragen	Antworten
1	Änderungen an einzelnen Dateien sind nur dadurch gekennzeichnet, dass der entsprechende Dateiname fett gedruckt ist. Über das **Properties**-Fenster können zusätzliche Informationen über einzelne Spielobjekte erhalten werden.
2	Einzelne Schritte können über den Shortcut STRG+Z rückgängig gemacht werden. Weitere Shortcuts können über den Menüpunkt **Tools → Options → Keymap** eingesehen, geändert und ergänzt werden. Versucht man das Editorfenster zu schließen, während ungespeicherte Änderungen vorliegen, erhält man eine entsprechende Meldung sowie eine Übersicht, in welchen Dateien Änderungen vorliegen. Hierbei hat man die Möglichkeiten einzelne Änderungen zu speichern, alle Änderungen zu speichern oder zu verwerfen oder das Schließen des Editorfensters abzubrechen.
3	Sobald ein geschriebener Programmcode nicht ausführbar ist, wird eine entsprechende Fehlermeldung im jeweiligen Script bei der entsprechenden Zeile angezeigt. Fehlermeldungen erhalten je nach Fehler eine hilfreiche Erklärung zur Behebung des Problems. Andere Fehlermeldungen sind je nach Kenntnissen der Nutzenden leichter oder schwerer zu verstehen.
4	Über den Menüpunkt **Help** kann die Dokumentation im Standardbrowser geöffnet werden in der die konkret gesuchten Hilfestellungen dann noch gesucht werden müssen. Darüber hinaus kann über den Menüpunkt auch die Javadoc Index Search geöffnet werden, die jedoch zum Zeitpunkt der Evaluation nicht funktioniert hat.

Tabelle 27 Kriterientabelle zur Heuristischen Analyse von jMonkeyEngine

FUDGE

Bewertungsfragen	Antworten
1	Einzelne Änderungen an Spielobjekten können über das Fenster **Graph** nachvollzogen werden – sofern diesem ein **Graph mit Components** hinzugefügt wurde. Über die Fenster **Preview** und **Properties** können zusätzliche Informationen über einzelne Spielobjekte erhalten werden, sobald diese im **Internal**-Fenster ausgewählt wurden. Änderungen an einzelnen Spielobjekten sind im **Components**-Fenster und je nach Art des Objekts auch im **Properties**-Fenster möglich und werden in beiden direkt angezeigt.
2	Einzelne Änderungen an Spielszenen oder einzelnen Spielobjekten können nicht durch einen Shortcut rückgängig gemacht werden. Auch können innerhalb des Editors nur Shortcuts zu den jeweiligen Menüpunkten eingesehen werden. Möglichkeiten zur Änderung dieser oder Erstellung neuer Shortcuts gibt es nicht. Ungespeicherte Änderungen werden Nutzenden nicht angezeigt und beim Schließen des Editorfensters wird auch nicht darauf hingewiesen.
3	Im Editorfenster werden keine Fehler in geschriebenem Programmcode angezeigt.
4	Unter dem Menüpunkt **Edit → Help** kann sowohl die **Documentation** sowie weitere potenzielle Hilfs-Quellen im Standardbrowser geöffnet werden.

Tabelle 28 Kriterientabelle zur Heuristischen Analyse von FUDGE