Hochschule Harz

Fachbereich Automatisierung und Informatik

Studiengang Medien- und Spielekonzeption

Masterarbeit zum Thema:

Umsetzung einer Anwendung zur Modellierung und Durchführung von Populationssimulationen mithilfe von Unity DOTS

Felix Altenhofen, m29641

Abgabe: 19.06.2023

Erstbetreuer: Prof. Jürgen Singer Ph.D.

Zweitbetreuer: Prof. Daniel Ackermann

I Inhaltsverzeichnis

II Abkürzungsverzeichnis

DOTS.....Data Oriented Technology Stack (Sammlung datenorientierter Technologien)

ECS..... Entity Component System (Entitäten-Komponenten-System)

FPS ..... Frames per Second (Bilder pro Sekunde)

GUI.....Graphical User Interface (grafische Benutzeroberfläche)

ID.....Identifikationsnummer

JSON .....JavaScript Object Notation (Objektnotation basierend auf JavaScript)

LTS..... Long Term Support (Unterstützung über lange Zeit)

III Glossar

Befehlsarchitektur
Äußere Architektur eines Prozessors, die festlegt, wie Software mit der Prozessorhardware interagieren kann. Auf Prozessoren mit gleicher Befehlsarchitektur kann dabei die gleiche Software verwendet werden, auch wenn die tatsächliche Hardware der Prozessoren sich stark unterscheidet.

Bildfrequenz
Anzahl an unterschiedlichen Bildern, welche in einer bestimmten Zeitspanne durch eine Anwendung bereitgestellt und auf einem Ausgabegerät, beispielsweise einem Bildschirm, dargestellt werden.

Compiler
Programm, welches Programmcode, der in einer bestimmten Programmiersprache geschrieben wurde, in eine andere Sprache – üblicherweise Maschinencode – überträgt und ihn dabei optimiert.

CPU-Cache
Flüchtiger Hardware-Speicher der physisch nah an den Kernen eines Prozessors platziert ist und Daten speichert, auf welche der Prozessor mit hoher Geschwindigkeit zugreifen muss.

Datenorientierung
Programmierungsansatz, bei dem die Anordnung von Daten im CPU-Cache optimiert werden soll, um bei den erstellten Anwendungen höhere Performanz zu erzielen. Wird hauptsächlich in der Entwicklung digitaler Spiele genutzt. [1]

Debugging
Vorgang, bei dem ein Programm, bei welchem unerwartetes Verhalten auftritt, auf Fehler untersucht wird, um diese zu beheben. [2]

Framework
Für die Programmentwicklung genutztes Grundgerüst, welches häufig benötigte Funktionen beinhaltet, sodass diese von auf dem Framework basierenden Anwendungen genutzt werden können, ohne dort neu umgesetzt werden zu müssen.

Instructions per Cycle
Maßeinheit für die Anzahl an Befehlen, welche ein Prozessor in einem Taktzyklus ausführen kann. [3]

Maschinencode
Programmiersprache, welche aus direkt von einem Prozessor ausführbaren Anweisungen besteht. Ihr Aufbau wird von der Befehlsarchitektur des jeweiligen Prozessors vorgegeben.

Multithreading	Aufteilung der durch den Code einer Anwendung vorgegebenen Befehle auf mehrere sogenannte „Threads“. Prozessorkerne können daraufhin jeweils einen oder mehrere dieser Threads voneinander unabhängig bearbeiten, sodass der Programmcode parallel und somit schneller ausgeführt werden kann.
Objektorientierung	Entwicklungsansatz für Programme, bei dem Programmbestandteile als Objekte mit durch Methoden manipulierbaren Eigenschaften definiert werden.
Performanz	Leistung eines beliebigen digitalen oder physischen informationstechnischen Systems, welche die Geschwindigkeit widerspiegelt, mit der Berechnungen in von diesem durchgeführt werden.
Prozessorkern	Recheneinheit eines Prozessors, die in Mehrkernprozessoren unabhängig von anderen Kernen durch Programmcode gegebene Befehle ausführt. [4]
Spiel-Engine	Echtzeit-Framework für digitale Spiele, welches grundlegende Anwendungskomponenten für den Spielablauf handhabt. Üblicherweise verbunden mit einer speziell auf die Engine ausgelegten Entwicklungsumgebung, die Werkzeuge zur Anwendungserstellung bereitstellt. [2]
Taktfrequenz	Geschwindigkeit, mit der Prozessoren Befehle ausführen. Pro Takt können je nach deren Art mehrere solcher Anweisungen umgesetzt werden; mitunter bedürfen Befehle auch einer Ausführung über mehrere Takte hinweg. [5]
Zeit pro Bild	Zeit in Millisekunden, welche ein Programm für die Bereitstellung eines konkreten Einzelbildes benötigte. [6]

IV Abbildungsverzeichnis

V Tabellenverzeichnis

1 Einleitung

Die letzten Jahre und Jahrzehnte brachten massiven technologischen Fortschritt mit sich, welcher sich in der gegenwärtigen digitalen Landschaft unter anderem in immer leistungsfähigeren Computersystemen niederschlägt. Gleichzeitig ist jedoch auch zu beobachten, dass die inkrementelle Verbesserung der aktuellen Spitzen-Hardware sich zunehmend verlangsamt. [7]

Gut wahrnehmbar ist dies beispielsweise bei Desktop-Prozessoren, deren Design statt schlichter Optimierungen zur Steigerung von *Taktfrequenz* und/oder *Instructions per Cycle* immer größere fundamentale Veränderungen erfährt, um *Performanz* und Effizienz merklich steigern zu können. So ist über die letzten Jahre hinweg beispielsweise die Menge an *Prozessorkernen* aktueller Systeme deutlich angestiegen; seit ihrer zwölften Generation verwenden Intel-Prozessoren zudem ein hybrides Design mit getrennten Performanz- und Effizienz-Kernen, um hohe Leistung wie auch Energieeffizienz erzielen zu können. [8] Aktuelle Prozessoren der Marke AMD setzen inzwischen hingegen auf ein modulares Chiplet-Design und nutzen zum Teil statt rein horizontal nebeneinander angeordnete *CPU-Cache*-Module auch vertikal übereinander gestapelte, um schnellere Zugriffszeiten zu ermöglichen und so höhere Performanz zu erzielen. [9] [10]

Andererseits gewinnen auch gänzlich neue *Befehlsarchitekturen* immer mehr an Bedeutung, welche mit den gegenwärtig noch dominanten x86-basierten CPUs konkurrieren. Auf der Arm-Architektur basierende Prozessoren sind auf Mobilgeräten längst Standard und werden beispielsweise bereits seit Ende 2020 auch bei Apple-PC-Systemen genutzt. [11] Auch RISC-V, eine in ihrer aktuellen Form erstmals 2015 vorgestellte und somit sehr moderne Befehlsarchitektur, wirbt immer erfolgreicher darum, in Endgeräten eingesetzt zu werden. [12] [13] [14] Dabei sind die größten Vorteile der Architektur, dass sie einen offenen Standard darstellt und somit gebührenfrei nutzbar ist. Durch ihre nur wenige Jahre zurückliegende Veröffentlichung ist RISC-V zudem enorm optimiert und auf diversen modernen Gerätefamilien performant und gleichzeitig ressourcenschonend einsatzfähig. [15]

Ein ähnliches Bild zeichnet sich auch bei Grafikkarten ab, bei denen direkte Leistungsverbesserungen immer häufiger von zusätzlichen Softwaresystemen überschattet werden, welche durch den Einsatz unterschiedlichster Algorithmen die Performanz der Hardware verbessern sollen. Nennenswert sind dabei beispielsweise AMDs FidelityFX Super Resolution, NVIDIAs Deep Learning Super Sampling und Intels X^e Super Sampling. Diese erhöhen die Auflösung der von der Grafikkarte bereitgestellten Bilder künstlich, um in Bezug auf die grafische Ausgabe rechenintensive Anwendungen auch auf hochauflösenden Displays flüssig nutzbar zu machen.

[16] [17] [18] Technologien wie Resizable BAR und AMD Smart Access Memory finden zudem Anwendung, um Zugriffszeiten bei der Kommunikation von Grafikarte und Prozessor zu minimieren. [19] [20] [21]

Jedoch sind nicht allein Hardwarehersteller in der Verantwortung, die Nutzung immer komplexerer Anwendungen zu ermöglichen. Auch besagte Anwendungen selbst müssen auf Performanz hin optimiert werden, um komplexe Funktionen bereitstellen zu können und/oder zu gewährleisten, dass ihre Verwendung auch auf weniger leistungsstarken PC-Systemen möglich ist. [7]

Ein gutes Beispiel für stark performanzgebundene Software stellen komplexe Echtzeitanwendungen dar, mit deren Inhalten Nutzer direkt und unverzögert interagieren können – so beispielsweise digitale Spiele. Die zur Erstellung eines Großteiles der entsprechenden Spiele verwendeten *Frameworks* sollten entsprechend stetig angepasst werden, um eine auf Performanz fokussierte Anwendungsentwicklung mit maximaler Kompatibilität zum aktuellen Hardware-Standard zu ermöglichen. Um auf ein vorheriges Beispiel zurückzukommen, wäre so im Zuge der Hardware-Entwicklung der letzten Jahre beispielsweise Unterstützung für *Multithreading* sinnvoll.

Die im Falle digitaler Spiele zum Entwickeln genutzten *Spiel-Engines*, beispielsweise Unreal Engine, Cry Engine oder Unity, können jedoch auch zur Umsetzung sonstiger leistungsstarker Echtzeitanwendungen genutzt werden. Entsprechend stellen sie ein weiteres relevantes Anwendungsgebiet in Echtzeit ablaufender Simulationen dar, welche beispielsweise zur möglichst greifbaren Veranschaulichung von Prozessen Anwendung finden und stark von höherer Leistungsfähigkeit profitieren können.

Diese Arbeit beschäftigt sich mit dem „Data-Oriented Technology Stack“ (DOTS). Bei diesem handelt es sich um eine Erweiterung der Unity-Entwicklungsumgebung, welche das unkomplizierte Erstellen komplexer, hochperformanter Anwendungen ermöglichen soll. Dabei wird untersucht, inwiefern Unity DOTS die Umsetzung von Echtzeit-Simulationen begünstigen und ob Unity entsprechend eine geeignete Entwicklungsumgebung für diesen Anwendungsfall darstellen kann.

Diese Überprüfung geschieht praxisnah in einem Projekt zur beispielhaften Umsetzung einer solchen simulationsbezogenen Echtzeit-Anwendung in prototypischer Form. So soll es ermöglicht werden, möglichst realitätsnahe Testdaten zur Effektivität von DOTS hervorzubringen. Die umzusetzende Anwendung soll es konkret ermöglichen, schlichte Populationsmodelle zu erstellen und diese innerhalb einer statischen Umgebung zu simulieren. Diese Simulation soll wahlweise mit oder ohne Verwendung von Unity DOTS ablaufen, um einen direkten Vergleich zu

ermöglichen und so die durch DOTS erzielten performanzbezogenen Unterschiede feststellen zu können.

Im Rahmen dieser Arbeit wird zunächst ein grundlegender Überblick über Simulationen und Populationssimulationen im Speziellen gegeben. Darauffolgend wird die Unity-Entwicklungsumgebung, sowie Unity DOTS vorgestellt, wobei die Komponenten von DOTS und ihre jeweiligen Funktionen genauer betrachtet werden. Auch werden in diesem Rahmen die angedachten Vorteile und Anwendungsgebiete von Unity DOTS näher erläutert. Im Anschluss folgt eine Darlegung des Projektaufbaus, wobei definiert wird, was genau die zu erstellende Anwendung beinhalten und wie sie funktionieren soll. Es werden zudem die zur Umsetzung der Anwendung nötigen Einzelschritte dargelegt und Bewertungskriterien für die Auswertung des Projekts bestimmt.

Im Anschluss daran wird der Ablauf der Projektdurchführung dargelegt und eine Bewertung des dadurch erzielten Resultats anhand der vorher definierten Kriterien durchgeführt. Am Ende der Arbeit stehen zudem ein Fazit zum Projekt in seiner Gesamtheit sowie ein Überblick über das Einsatzpotential von Unity DOTS im Rahmen des betrachteten Anwendungsgebiets sowie im Allgemeinen. Auch werden Möglichkeiten zur sinnvollen Weiterentwicklung des erstellten Anwendungsprototyps aufgezeigt.

2 Simulationen

Bei Simulation handelt es sich um ein verbreitetes Analyseinstrument. [22, p. 4] Konkret wird dabei im Rahmen einer Simulation ein für ein reales oder imaginäres System repräsentatives Modell erstellt, welches experimentell untersucht werden kann, um neue Erkenntnisse zu dem abgebildeten System zu gewinnen. [22, p. 4] [23, p. 3] Das Modell ist dabei ein vereinfachtes Abbild, welches nur die für die Untersuchung relevanten Variablen des Ursprungssystems beinhaltet. Neben diesen grundlegenden Eigenschaften imitiert ein Modell zudem die zeitlichen Abläufe von Vorgängen im jeweiligen System. [22, p. 4]

Methoden der Simulation sind zumeist einfacher anwendbar und weniger eingeschränkt als rein analytische Methoden, welche gleiche Ergebnisse erzielen können. Jedoch stellen die durch Simulationen gesammelten Daten stets Stichproben dar. Um relevante, belastbare Ergebnisse zu gewährleisten muss also sichergestellt werden, dass diese Stichproben tatsächlich statistische Aussagekraft besitzen. [22, pp. 4-5]

Im Rahmen von Simulationen bezeichnet ein System eine Menge an Objekten in einem in beliebigem Sinne abgegrenzten Bereich, welche zueinander in Beziehung stehen. [22, p. 77] [23, pp. 3, 10] Diese Objekte besitzen Eigenschaften und können vielfältige Verhaltensmuster aufweisen und Einfluss aufeinander respektive auf ihr jeweiliges Verhalten ausüben. Im Rahmen von Simulationen werden Systeme dabei zumeist auf einen zentralen Untersuchungsgegenstand reduziert betrachtet. [22, p. 77] Die Gesamtheit der in einem System befindlichen Variablen, also aller Objekte und ihrer Attribute, wird als Zustand des Systems bezeichnet. [22, p. 77] [23, p. 10]

Ein bei einer Simulation verwendetes Modell stellt immer ein vereinfachtes Abbild eines solchen Systems dar, das das zeitliche und strukturelle Verhalten des Systems widerspiegelt. [22, p. 78] [23, p. 3] Die Vereinfachung ist jeweils so gewählt, dass zu dem im Fokus der Simulation stehenden Sachverhalt relevante Erkenntnisse unverfälscht aus dem Modell abgeleitet werden können. [23, p. 3] Jedes Modell stellt dabei letztlich ebenfalls ein System dar.

Systeme sind häufig dynamisch, was sich dadurch auszeichnet, dass sich ihr Zustand im Laufe der Zeit verändert. Entsprechende Veränderungen können dabei beliebig oft wiederholt und in jeweils unterschiedlicher Ausprägung ablaufen. Ausgelöst werden diese Zustandsänderungen durch im System auftretende Ereignisse, durch die sich die Objektmenge, ihre Attribute oder die Beziehungen zwischen den Objekten ändern. Verschiedene Ereignisse sind durch Aktivitäten verbunden, welche von den Objekten des Systems ausgeführt werden. Dabei können diese Aktivitäten sowohl sequenziell als auch parallel zueinander ablaufen und einander beeinflussen;

etwa indem das am Ende einer Aktivität stehende Ereignis andere Aktivitäten auslöst. [23, pp. 3-4, 10-13]

Je nach den Charakteristiken des zu simulierenden Systems und der in ihm zu betrachtenden Vorgänge, gibt es verschiedene allgemeine Simulationstechniken, um eine möglichst effiziente Simulation des erstellten Modells zu ermöglichen. [23, p. 21] Sehr verbreitet ist dabei die ereignisorientierte Simulation, die das Verhalten eines Systems mit enorm hoher Genauigkeit nachspielt. Dabei werden alle eintretenden Ereignisse und alle durch sie ausgelösten Zustandsänderungen des Modells einzeln simuliert. Der Zustand der Simulation wird kontinuierlich aktualisiert, wodurch ein flüssiger zeitlicher Ablauf entsteht, der Untersuchungen der ermittelten Daten zu jeder Zeit zulässt. [22, pp. 79, 92] [23, p. 22]

Hingegen werden bei prozessorientierter Simulation zusammenhängende Ereignisse und ihre Folgen als in Prozessen vereint betrachtet, wodurch die Simulation und damit die erfassten Daten letztlich weniger granular sind. [22, p. 92] [23, pp. 26-27] Wiederum existiert bei periodenorientierter Simulation keine kontinuierliche Zustandsberechnung. Stattdessen wird der aktuelle Modellzustand in punktuell festen zeitlichen Abständen ermittelt, die erfassten Daten sind also auf diese Zeitpunkte beschränkt. [23, p. 30]

Der Begriff der Simulation ist enorm vielfältig nutzbar. Simulationskonzepte sind entsprechend ebenso vielfältig und in einer Vielzahl von Anwendungsfeldern einsetzbar, da sie ein Verstehen, Optimieren und Planen verschiedenster Systeme des echten Lebens und selbst die Beleuchtung rein fiktiver Konzepte ermöglichen. So finden sich Einsatzfälle für Simulationen bei der Planung von Versorgungsketten, der Erforschung physikalischer Effekte, der Optimierung von Menschen- sowie Verkehrsflüssen oder der Vorbereitung und Durchführung von Raumfahrt-Missionen, um einige wenige Beispiele zu benennen. [22, pp. 14-15] [23, pp. 3, 117]

Besonders interessant sind Simulationen dabei für Systeme, in denen stochastische Vorgänge ablaufen – in denen also der Zufall Einfluss auf die Zustandsänderungen des Systems hat. Bei Zufallseinflüssen stoßen theoretische Ansätze zum Feststellen von Gesetzmäßigkeiten aufgrund zu großer Varianz häufig an ihre Grenzen. Pseudo-zufällig generierte Zahlen können in Modellen entsprechende Varianz gut abbilden und so ein systematisches, experimentelles Ermitteln von Regeln des simulierten Systems ermöglichen. [23, p. V]

Im heutigen Zeitalter werden Simulationen zumeist in digitaler Form, als in Software abgebildetes, oft interaktives, Modell umgesetzt. [23, p. 3] Genauso ist jedoch auch eine rein analoge Verwirklichung in der Realität möglich. Ein Beispiel dafür findet sich im Modell der Bucht von San Francisco, welches in der kalifornischen Stadt Sausalito vom United States Army Corps of Engineers betrieben wird. Das 1957 konstruierte, 100 mal 120 Meter große Modell ist ein maß-

stabsgetreuer Nachbau der Bucht und ihrer Umgebung und bildet diese und die in ihr ablaufenden Vorgänge enorm akkurat ab. [24] [25] So bildet es die Gezeiten wie auch die in der Bucht herrschenden Strömungen präzise ab und kann dadurch komplexe Veränderungen an dem System realitätsnah simulieren und ihre Auswirkungen sichtbar machen. [25]

Ursprünglich sollte das Modell dazu dienen, die Folgen von Vorschlägen zum Umbau der Bucht zu testen, welche eine Trennung von Meeres- und Trinkwasser mit daraus resultierenden neuen Nutzungsmöglichkeiten des Gebiets erreichen sollten. [24] [25] Genutzt wurde es zudem zur Bestimmung von Strömungsrichtungen in verschiedenen Teilen der Bucht und der damit einhergehenden Verteilung diversen Materials in ihr sowie zur vorrausgehenden Überprüfung weiterer Bauvorhaben, welche beispielsweise ein Erweitern der Bucht durch Abbaggern beabsichtigten. Entsprechende Simulationen werden seit dem Jahr 2000 in digitaler Form durchgeführt. [24]

In den folgenden Abschnitten werden einige konkrete Anwendungsgebiete für Simulationen näher beleuchtet. Zudem werden für diese Gebiete relevanten Eigenschaften entsprechender Simulationen näher erläutert.

2.1 Nutzen zur Veranschaulichung

Bisher wurden Simulationen lediglich im Hinblick auf ihre Nutzung im Rahmen, üblicherweise wissenschaftlicher, Untersuchungen von Systemen zum Zweck des Erkenntnisgewinns und der Feststellung von systembezogenen Regeln dargestellt. Ein weiteres relevantes Anwendungsgebiet findet sich jedoch in der Veranschaulichung der jeweils abgebildeten Systeme. So dient beispielsweise das zuvor erwähnte Modell der Bucht San Francisco inzwischen der bildungsbezogenen Veranschaulichung. Dabei soll der Umfang des Buchtgebiets, seine ökologische Komplexität und in ihm ablaufende Vorgänge, beispielsweise die Gezeiten, dargestellt und erlebbar gemacht werden. [24] [25]

Generell können Simulationen ein wirksames Werkzeug in verschiedenen Bereichen des digitalen Lernens sein, wo sie im Rahmen von, oft interaktiven, Visualisierungen zum Einsatz kommen. Dabei können sie genutzt werden, um Ursache-Wirkungsbeziehungen betrachteter Systeme zu veranschaulichen und das Systemverhalten unter verschiedenen Bedingungen zu zeigen. So können Simulationen in Systemen ablaufende Vorgänge erfahrbar machen und, sofern die Parameter der Simulation interaktiv bearbeitbar sind, das Trainieren von Fähigkeiten ermöglichen. Zudem können sie an Stelle von Experimenten zum Einsatz kommen, deren tatsächliche

Durchführung zu gefährlich und/oder zu kostenintensiv wäre. Durch die Vielfältigkeit von Simulationen können dabei Qualität und Umfang, je nach Einsatzgebiet und verfügbaren Ressourcen, leicht variiert werden, um eine optimale Nutzung zu ermöglichen. [26]

Zur Anwendung kommt dies beispielsweise im 2004 gegründeten Mathematik-Labor der Julius-Maximilians-Universität in Würzburg. Dieses soll es Schülerinnen und Schülern ermöglichen, sich experimentell mit mathematischen Konzepten auseinanderzusetzen und so ein einfaches wie auch natürliches Verstehen dieser und damit zusammenhängender Phänomene möglich machen. Dabei sollen unter anderem real geschaffene Modelle wie auch in digitalen Anwendungen ablaufende Simulationen helfen. Letztere sollen Visualisierungsmöglichkeiten bieten sowie Interaktion mit den präsentierten Inhalten ermöglichen und so als virtuelle Umgebungen zum Experimentieren dienen. [27, pp. V-VI, 107, 132-133]

Ein weiteres Anwendungsgebiet findet sich im Bereich digitaler Spiele wieder, welche auf der vielfältigen Simulation virtueller Welten basieren. Dabei besteht für die Entwickelnden derartiger Spiele häufig der Anreiz, die Systeme der Spiele möglichst realitätsnah zu gestalten, um über den so erzielten Realismus als Merkmal potentielle Kunden für das Spiel zu überzeugen. Sogenannte Simulationsspiele sind sogar hauptsächlich auf die Imitation von Aktivitäten und Funktionen des realen Lebens auf möglichst akkurate Art und Weise ausgelegt. [2]

Entsprechend können digitale Spiele zum Teil als Lernressourcen fungieren, indem diverse reale Vorgänge visualisiert und dadurch verständlich gemacht werden. Die für das Medium zentrale Interaktivität ermöglicht zudem direkt handlungsorientiertes Lernen. [28] [29] Gerade die Verwendung von Videospielen im Rahmen des Schulunterrichts bietet sich an, um Sachverhalte zu verdeutlichen oder besser zugänglich zu machen; analog zu anderen Medien. Zum Teil werden entsprechende Spiele daher bereits von Lehrkräften in diesem Rahmen eingesetzt. [29] Letztlich können digitale Spiele so Laien Einblicke in verschiedene Tätigkeitsfelder und/oder allgemeiner in wissenschaftlich festgestellte Begebenheiten und auf diesen basierenden Annahmen gewähren.

Als Beispiel für digitale Spiele mit entsprechenden Simulationen ist das von Klei Entertainment entwickelte „Oxygen Not Included“ zu nennen. Dabei handelt es sich um eine 2017 erstmals als Testversion veröffentlichte Kolonie-Simulation, in welcher Spielende damit beauftragt sind, ein Lager innerhalb eines Asteroiden zu errichten. [30] [31] [32] Dazu sind der Gruppe von Kolonisten, welche das Lager bewohnen sollen, Aufgaben zu erteilen. [31]

Oxygen Not Included baut sich dabei grundlegend auf einer Reihe an physikalischen wie auch chemischen Simulationssystemen auf. [32] So umfasst es beispielsweise eine Gas- und Flüssigkeitssimulation, welche den Fluss entsprechender Substanzen durch die konstruierten Räumlichkeiten sowie die Interaktionen zwischen ihnen steuert. [31] [32] Stoffe haben verschiedene

Anwendungsgebiete sowie unterschiedliche Auswirkungen auf die Kolonisten, die beispielsweise eine hohe Sauerstoffkonzentration zum Überleben benötigen. Entsprechend müssen verschiedene Flüssigkeiten und Gase durch Leitungssysteme transportiert sowie gefiltert und in Tanks gelagert werden, um eine Nutzung zu ermöglichen und das Überleben in der Kolonie sicherzustellen. [30] [31]

Darauf aufbauend sind auch die durch den Stofftransport erzielten Druckunterschiede zu beachten, welche räumlich simuliert werden und sich auf das Gedeihen von Lebewesen sowie Pflanzen auswirken. [32] Analog dazu agiert auch die ortsgebunden simulierte Temperatur, welche durch von Lebewesen und Maschinen produzierte Wärme erhöht wird und zur Schaffung eines lebensfreundlichen Umfelds abtransportiert und/oder abgebaut werden muss. [30] [31] Die eben erwähnten Maschinen sind zudem mit Strom zu versorgen, welcher von diversen in Betrieb zu haltenden Stromquellen erzeugt und gespeichert wird. Dazu ist der Aufbau von Stromnetzen nötig, welche wiederum instand gehalten und vor Überladung geschützt werden müssen. [31]

Alle in diesen Systemen vereinten Ressourcen können auf unterschiedliche Arten in jeweils andere Ressourcen umgewandelt werden. Dabei bleiben Masse und/oder Energie der Stoffe erhalten, wodurch der Kreislauf und die Verwaltung der begrenzten Ressourcen als weiteres Grundsystem eingebunden wird. [31] [32]

Diese verschiedenen, miteinander verbundenen Systeme kulminieren in Oxygen Not Included zu einer breit gefächerten, in ihren Bestandteilen aber eigentlich simplen Simulation. Dabei sollen die einander überlappenden Systeme das Spiel interessant machen und seine Herausforderung definieren. [32] Durch eine zum Teil zufällig generierte Spielwelt wird der Modellaufbau dabei zusätzlich variabel gestaltet, wodurch die dynamischen Interaktionen der Einzelsysteme weiter hervorgehoben werden. [31]

Johann Seidenz, einer der Designer des Spiels sieht dabei den Anreiz von Oxygen Not Included für Spielende hauptsächlich im Durchschauen der zusammenwirkenden Simulationssysteme und dem damit einhergehenden schrittweisen Verstehen des Simulationsmodells. [33] Auch wird das Beobachten der dargestellten Population sowie ihrer Entwicklung als Ansatz zum Erleben des Spiels betont. [30] [31]

Als weiteres Beispiel kann „Kerbal Space Program“ dienen, einer von Squad entwickelten und zuerst im Jahr 2011 als Testversion veröffentlichten Raumfahrt-Simulation mit enormen Detailgrad. [34] [35] [36] Konkret erlaubt das Spiel Nutzenden das Leiten eines Raumfahrtprogrammes, wobei die Konstruktion von Raumfahrzeugen und das Steuern ihres anschließenden Abschusses im Zentrum der spielerischen Interaktion stehen. [34] [35]

Der Bau besagter Raumfahrzeuge geschieht dabei durch das beliebige Kombinieren einer Vielzahl an Bauteilen, von denen jedes eigene Funktionen sowie Eigenschaften aufweist und sich entsprechend unterschiedlich auf das erstellte Raumschiff auswirkt. Das Start- und Flugverhalten der so erstellten Schiffe wird dabei möglichst realitätsgetreu simuliert. So verhält sich die erstellte Gesamtkonstruktion physisch korrekt und jede ihrer Komponenten wirkt sich, beispielsweise durch ihre aerodynamischen Eigenschaften, auf den Ablauf ihres Einsatzes und damit seinen letztlichen Erfolg oder Misserfolg aus. [34] [35]

Wurde so ein funktionsfähiges Raumfahrzeug erschaffen, können mit diesem der Weltraum bereist und die in ihm befindlichen astronomische Objekte erforscht werden. [34] [35] Dabei können auch fortschrittliche Manöver der Raumfahrt durchgeführt werden, wobei für eine erfolgreiche Durchführung stets das physikalische Verhalten der involvierten Objekte zu beachten ist. Beispiele dafür sind das Errichten frei im Raum schwebender Raumstationen sowie das Andocken an diese oder das Umkreisen von und Landen auf beliebigen astronomischen Objekten. [34] [35] [36] Als weitere Simulationsaspekte treten im Spiel zudem das Management der eingesetzten Crews sowie des Raumfahrtprogrammes an sich auf. [34] [35]

Laut Felipe Falanghe, dem Hauptentwickler von Kerbal Space Program, sollten Spielende sich dabei hauptsächlich damit befassen, zu verstehen, warum Fehlschläge auftreten und wie sie erstellte Konstruktionen entsprechend verbessern können. Um dies zu unterstützen ist das Spiel stark auf Experimentieren ausgelegt und eine tatsächlich erfolgreiche Raumfahrtmission folgt üblicherweise erst auf eine große Menge an Versuchen und gewonnenen Erkenntnissen. [34] [36]

Wegen des hohen Detailgrads des Spiels wurde dessen Entwicklerstudio im Jahr 2013 von der Nationalen Aeronautik- und Raumfahrtbehörde der Vereinigten Staaten für eine Zusammenarbeit kontaktiert. Dabei wurden reale Bauteile und Raummissionen der NASA in das Spiel integriert sowie Daten zur Verbesserung der Simulation zur Verfügung gestellt. Dies geschah im Rahmen der Öffentlichkeitsarbeit der NASA, welche in Kerbal Space Program und ähnlichen simulationsbasierten Spielen großes Potential sieht, der Bevölkerung Einblicke in ihr Tätigkeitsfeld und damit verbundene Karrierepfade zu gewähren. [36]

So werden entsprechende Spiele als Möglichkeiten gesehen, Menschen zu inspirieren und auf den Spielerfahrungen beruhende Zukunftspläne anzuregen. Auch wird ihre Anwendung zur angenehmen Wissensvermittlung als große Stärke digitaler Spiele angesehen. Mit „KerbalEdu“ entstand entsprechend inzwischen ein Projekt, in welchem eine auf Lehre ausgerichtete angepasste Version des Spiels in Schulen eingesetzt werden soll, um Schülerinnen und Schülern die behandelten Konzepte näherzubringen. [36]

Weitere Beispiele lassen sich beim Entwickler „CodeParade“ finden, welcher digitale Spiele umsetzt, die verschiedenste hypothetische Konzepte behandeln. So simuliert sein 2022 veröffentlichtes Spiel „Hyperbolica“ Welten mit nicht-euklidischen Raumgeometrien; also solche, in denen die geometrischen Regeln, welche auf unsere Weltwahrnehmung zutreffen, durch andere ersetzt sind. Dabei beschäftigt sich Hyperbolica spielerisch mit den dadurch hervorgerufenen Effekten und lässt Anwendende erleben, wie die verschiedenen Geometrien funktionieren, deren Existenz den wenigsten Menschen überhaupt bekannt sein dürfte. [37]

Ein ähnliches Bild zeigt sich beim aktuell in Entwicklung befindlichen „4D Golf“, welches eine Welt mit vier wahrnehmbaren Raumdimensionen simuliert. Diese Vierdimensionalität wird so erfahrbar und besser verständlich gemacht, wobei verschiedene Einstellungsmöglichkeiten eine vielfältige Anpassung der Spielerfahrung erlauben, um die simulierten Inhalte möglichst gut verständlich zu machen. [38] CodeParade veröffentlicht dabei zusätzliche Videos über seinen YouTube-Kanal, in welchen er unter anderem die in beiden Spielen behandelten Konzepte näher erläutert, um sie Zuschauenden zugänglicher zu machen. Die Spiele im Speziellen finden in diesem Rahmen Anwendung zur Visualisierung. [39]

Abseits digitaler Spiele stellt der YouTube-Kanal „Primer“ ein weiteres Beispiel für die Nutzung von Simulationen zur Veranschaulichung von Sachverhalten dar. Auf diesem stellt Justin Helps diverse Konzepte, hauptsächlich aus der Biologie sowie der Wahrscheinlichkeitsrechnung, vor. Diese demonstriert er mithilfe visueller Simulationen, deren erzielte Daten er auswertet und mit in den jeweiligen Fachrichtungen etablierten Grundsätzen vergleicht. Zusätzlich stellt er Zuschauenden die Simulationen zum Teil zur Verfügung, sodass diese sie ausprobieren und so die behandelten Effekte selbst erfahren können. So beleuchtet Primer die Hintergründe wissenschaftlicher Gesetze und macht die behandelten Themen der Öffentlichkeit zugänglicher. [40]

Neben ihrem großen forschungsbezogenen Nutzen sind Simulationen also gerade im Bildungsbereich gut einsetzbar und bieten durch ihre im Fokus stehende starke Erlebbarkeit eine Vielzahl an Möglichkeiten zur Wissenswiedergabe und -vermittlung. Im Gegensatz zur wissenschaftlichen Nutzung rückt dabei die Genauigkeit der Modelle oft weiter in den Hintergrund und muss lediglich grundlegend gegeben sein, um die simulierten Sachverhalte korrekt abzubilden.

2.2 Populationssimulationen

Durch die Vielfältigkeit von Simulationen und ihrer Anwendungszwecke finden sie im Rahmen verschiedenster Felder Anwendung. Ein im Rahmen dieser Arbeit zentrales Anwendungsgebiet

stellt die Simulation von Populationen dar, welche genutzt wird, um das Verhalten und/oder die Entwicklung verschiedenförmiger Gruppen von Individuen darzustellen.

Einige Videos des im vorhergehenden Abschnitt beleuchteten YouTube-Kanals Primer stellen ein gutes Beispiel für solche Populationssimulationen dar. In diesem Fall kommen sie zum Einsatz, um biologische Konzepte, beispielsweise die natürliche Selektion, altruistisches Verhalten oder die Ausbreitung von Krankheiten darzustellen und die Funktionsweisen besagter Konzepte zu erläutern. [40]

Ein ähnlicher Ansatz findet sich im Webvideo „Coding Adventure: Simulating an Ecosystem“ von Sebastian Lague. In diesem erschafft er eine virtuelle Welt, welche als Modell der Realität dienen soll. Diese lässt er von zunächst einem und später zwei verschiedenen Kreaturenarten mit je unterschiedlichen Verhaltensweisen bevölkern. Im Verlauf des Videos passt Lague die Bedürfnisse und Fähigkeiten der so geschaffenen Kreaturen an und erweitert sie, wobei er dokumentiert, wie sich die entsprechenden Änderungen auf den Simulationsablauf auswirken. Dazu protokolliert er die Entwicklung der Populationsgrößen über verschiedene Durchläufe der Simulation hinweg. [41]

Die Eigenschaften der in dem simulierten Modell genutzten Kreaturen gestaltet Lague dabei als eingeschränkte Größen, die bei den individuellen Kreaturen unterschiedlich stark ausgeprägt sein können. Durch einen zusätzlich implementierten schlichten Fortpflanzungsvorgang wird der Simulation das evolutionäre Konzept der natürlichen Selektion hinzugefügt. Im Video wird dieses Konzept durch Protokollierung der konkreten Attributwerte und ihrer Entwicklung über Simulationsdurchgänge hinweg näher beleuchtet. Dabei werden insgesamt Rückschlüsse zu den Auswirkungen der verschiedenen Anpassungen des simulierten Modells gezogen, um Zuschauenden einen grundlegenden Einblick in die stark vereinfachten biologischen Prozesse zu gewähren. [41]

Bei bereits erwähnten Simulationsspielen finden sich ebenfalls verschiedenste Populationssimulationen. So ist zum Beispiel Oxygen Not Included in seinen Grundlagen eine Populationssimulation der im Zentrum stehenden Kolonisten. Diese haben schlichte Persönlichkeiten sowie grundlegende physische und psychische Bedürfnisse, deren Erfüllung ihnen ein zufriedenstellendes Leben ermöglicht. [31] [32] Diese Attribute wirken sich in begrenztem Umfang auf das Verhalten der Kolonisten aus; in ähnlicher Weise tun dies auch äußere Einflüsse, die beispielsweise durch Aufbau und lokale Merkmale des errichteten Lagers eingebracht werden.

Im Folgenden soll allerdings stärker auf einige Spiele eingegangen werden, in deren Zentren tiefergreifenden Populationssimulationen stehen; angefangen mit dem im Jahr 2006 erstmals veröffentlichten „Dwarf Fortress“. [42] [43] [44] Bei diesem von 12 Bay Games entwickelten Spiel handelt es sich um eine im Fantasy-Genre angesiedelte Siedlungsbau-Simulation, welche

versucht, jeden Aspekt der Spielwelt und ihrer Systeme möglichst detailliert zu simulieren. So sollen Spielenden möglichst vielfältige Interaktionen mit tiefgreifenden spielinternen Abläufen ermöglicht werden. [42]

Dabei präsentiert Dwarf Fortress eine vollständig simulierte, persistente Spielwelt, bestehend aus Ozeanen wie auch Kontinenten mit verschiedenen Landformen und erbauten Strukturen. [43] [45] Verschiedene Spielsysteme dienen dazu, Wetter, Kreaturen und ähnliche Inhalte der virtuellen Welt teilweise zufallsbasiert zu kreieren und so eine hohe Varianz im Spielablauf zu schaffen. Diese Varianz setzt sich fort in der großen Menge verschiedenster Ressourcen und Gegenständen, die in der Welt verteilt sind und unterschiedlichste Eigenschaften besitzen, welche die Interaktionen beeinflussen, die mit ihnen durchgeführt werden können. [46] [45]

Die in der Spielwelt heimischen Charaktere weisen detaillierte Persönlichkeiten, Fähigkeiten und Eigenschaften auf, wobei alle Kreaturen in ihre Körperteile unterteilt betrachtet werden. So wird jedes Körperteil separat simuliert, wodurch sie individuell von verschiedenen Systemen beeinflusst werden und dadurch ebenso individuelle Eigenschaften aufweisen können. Dabei bestimmt letztlich die Beschaffenheit der Gesamtheit der Körperteile einer Kreatur, ob und wie diese agiert beziehungsweise überhaupt lebt. [44] [46] [45]

Die Spielwelt bewohnende Charaktere und Kreaturen leben zum Teil vereint in stark unterschiedlich ausgeprägten Zivilisationen, welche auf verschiedene Arten miteinander aufeinander einwirken können. Der Spielende selbst hat die Aufgabe, eine solche Zivilisation in einem kleinen Ausschnitt der Spielwelt aufzubauen, wobei er mit der umliegenden Welt und ihren Bewohnern frei interagieren kann. [43] [45] Derjenige hat dabei keine direkte Kontrolle über das Spielgeschehen, sondern regt die ihm unterstellten Charaktere lediglich zu Handlungen an, bei denen die Simulation selbst entscheidet, wann und wie sie tatsächlich ausgeführt werden. [44]

Zusammen mit dem Fehlen einer festen Siegbedingung, wird der Spielende so motiviert, die verschiedenen Abläufe des Spiels zu beobachten und sich zu erschließen, wie diese von den Interaktionen der diversen Spielsysteme beeinflusst werden. [44] So können selbst gesteckte Ziele im Rahmen des Erlernens und Meisterns der verschiedenen Spielkomponenten gesteckt und verfolgt werden. Für dieses als herausragend bezeichnete Interaktionsdesign wird Dwarf Fortress seit 2012 im New Yorker Museum of Modern Art ausgestellt. [43] [44] [46] [47]

Ein ähnliches Beispiel findet sich im von Ludeon Studios erstmals 2013 veröffentlichten Spiel „Rimworld“, welches sich ebenfalls als Kolonie-Simulation versteht. [48] [49] [50] [51] Konkret werden Spielende hier damit beauftragt, auf einem fremden Planeten eines Science-Fiction-basierten Universums mithilfe einiger Kolonisten ein Lager zu errichten und die Lebensverhältnisse in diesem zu verbessern. [49] [51]

Die besagten Kolonisten haben dabei individuelle Persönlichkeiten mit unterschiedlichen Bedürfnissen, Launen, Stärken und Schwächen sowie diversen geistigen und/oder körperlichen Merkmalen. Zusammen beeinflussen diese Attribute das Verhalten sowie die Gefühle der Charaktere in verschiedenen Situationen und wirken sich unterschiedlich auf die Beziehungen zwischen ihnen sowie ihren Einfluss aufeinander aus. [48] [49] [50] [51]

Analog zu Dwarf Fortress werden dabei die Körperbestandteile von Menschen und Kreaturen individuell betrachtet und können durch Krankheiten, Verletzungen oder medizinische Eingriffe beschädigt, entfernt, verbessert oder ersetzt werden. Diese Aktionen haben entsprechenden Auswirkungen auf Fähigkeiten, Verhalten und Moral der zugehörigen Kreatur sowie der Kolonie in ihrer Gesamtheit. Errichtete Strukturen sowie Ausrüstungsgegenstände wirken sich gleichermaßen aus. [48] [49]

RimWorld hebt sich dabei durch ein Event-System hervor, durch welches eine Vielzahl unterschiedlicher Ereignisse dynamisch eintreten können. Diese wirken sich auf die Kolonie und ihre Bewohner in verschiedener Weise aus und bewirken so variable physische wie auch psychischen Veränderungen. [48] [49] [50] Ereignisse sind zudem offen gestaltet, sodass sie einander überlappen und so dynamisch miteinander interagieren können. [51] Der als Kulisse für diese Ereignisse dienende Planet, auf welchem sich das Spiel abspielt, wird in RimWorld vollständig generiert und besteht aus verschiedenen Klimazonen, welche diverse Ressourcen, erbaute Strukturen sowie miteinander interagierende Bevölkerungsgruppen mit je eigenen Kulturen beinhalten. Verschiedene Klimaareale heben sich zudem durch unterschiedliche Temperaturen sowie Wetterphänomene voneinander ab. [48] [49]

Spielenden bietet RimWorld dabei große Freiheit bei der Gestaltung ihres persönlichen Spielablaufs. So ist für die Kolonie zunächst lediglich ein kleines umliegendes Gebiet der generierten Spielwelt relevant, welches frei gewählt werden kann und sich durch sein Klima und die in ihm befindlichen Kreaturen und Kulturen direkt auf die Siedlung und ihre Bewohner auswirkt. Die Kolonie selbst ist sehr frei formbar und wird wie auch die umgebende Welt durch verschiedene kulturbezogene Entscheidungen beeinflusst, welche von Spielenden getroffen werden können. [48] [49]

Auf verschiedene Geschehnisse in und um die Kolonie weist das Spiel dabei zwar stets hin, ein Handlungszwang existiert jedoch nie, wodurch ermöglicht wird, das Spielgeschehen sich frei entfalten zu lassen und Interaktionen sowie die Auswirkungen von Ereignissen zu beobachten. [49] Tynan Sylvester, der Schöpfer von RimWorld sieht die Interaktion mit den maßgeschneiderten Spielsystemen als den größten Spaßfaktor des Spiels. Dabei machten den Reiz von RimWorld aus seiner Sicht hauptsächlich Simulationen aus, welche nicht zu komplex sind und von Spielenden beobachtet und verstanden werden können. [51]

Sowohl Dwarf Fortress als auch RimWorld simulieren im Kern also eine kleine Menge an komplexen Individuen sowie eine Vielzahl von Verhaltensweisen und deren Interaktionen. Während das Hauptaugenmerk bei Ersterem aber auf der Zusammenführung vieler enorm detaillierter Zusatzsysteme liegt, fokussiert sich RimWorld auf makroskopischere Interaktionen und auf die zentrale Kolonie in ihrer Gesamtheit betreffende Effekte.

Ein Beispiel für ein Populationssimulationsspiel mit deutlich anderer Ausrichtung stellt hingegen das von Entwickler Karl Whimble erstellte Spiel „Equilinox“ dar, welches 2018 als „Natursimulationsspiel“ veröffentlicht wurde. [52] [53] [54] In diesem erschaffen Spielende unterschiedlich ausgeprägte Ökosysteme, wofür eine große Auswahl an Pflanzen sowie Tieren zur Verfügung stehen. Dabei bringt jede der Pflanzen und jedes der Tiere eigene Verhaltensmuster sowie individuelle Vorlieben in Bezug auf den bereitgestellten Lebensraum ein. [52] [53] [54] [55]

Platzierte Lebewesen interagieren selbstständig miteinander und führen laut Whimble in Equilinox je nach Art eine mehr oder weniger einzigartige Funktion aus, wodurch das Spiel eine sehr dynamische, variable Simulation darstellt. [52] [53] Basierend auf der platzierten Flora und Fauna entstehen in der Spielwelt zudem unterschiedliche Biome, die diese Variabilität weiter erhöhen. [53]

Das hauptsächliche Ziel des Spiels findet sich darin, ein ausgeglichenes Ökosystem zu erstellen. [52] [54] Zu diesem Zweck müssen die diversen Unter-Systeme der Spielinhalte verstanden und aufeinander abgestimmt werden, wofür Equilinox Spielende zum Experimentieren motiviert. Es lässt Spieler zudem die verschiedenen Interaktionen wie auch die generelle Entwicklung der Spielwelt ungestört erleben, ohne Eingriffe zu fordern. Insgesamt sollen Spieler dabei angeregt werden, sich auf ein schlichtes Genießen der erstellten virtuellen Welt und ein Beobachten ihrer Kreaturen und deren Routinen einzulassen. [52]

Laut Whimble sei Equilinox dabei nicht speziell auf eine bildungsbezogene Funktion ausgelegt. So basiere das Spiel zwar auf der Realität entnommenen Gegebenheiten, lege bei deren Umsetzung jedoch keinen zu großen Fokus auf Genauigkeit. Dennoch erkenne er es als Ziel von Equilinox, Menschen Einblicke in natürliche Systeme zu geben und so Interesse an der Natur in Gänze zu wecken. [52]

Abseits dieser publikumsorientierten Anwendungsbeispiele finden Populationssimulationen ebenfalls Verwendung in wissenschaftlichen Feldern; so zum Beispiel in der Systembiologie. Diese hat zur Hauptaufgabe, biologische Organismen sowie die in ihnen ablaufenden Prozesse im Sinne des Erkenntnisgewinns vollständig zu untersuchen. [56, p. 1] Dabei nutzt sie einen interdisziplinären Ansatz, bei dem Biologie, Mathematik und Informationstechnik in theoretischen wie auch experimentellen Methoden kombiniert werden. [56, pp. 1-2] [57, p. 3] An dieser

Stelle speziell hervorzuheben ist die Bedeutung der Informatik, welche die Entwicklung sowie effiziente Implementierung von Analyse- und Simulationsmethoden zu verantworten hat. Gleichwohl kommt diesem Bereich hohe Bedeutung im Rahmen der Modellbildung sowie der Umsetzung von Visualisierungswerkzeugen zu. [56, pp. V, 2-3]

Wie aus der Relevanz dieser Konzepte abzuleiten, stellt das Erstellen von Modellen und die Durchführung und Analyse von Simulationen einen der Hauptansätze zur Beantwortung der Kernfragen der Systembiologie dar. [56, p. 1] So sollen Modellanalysen ein Verständnis der modellierten Systeme schaffen und Simulationsstudien zur Überprüfung von Hypothesen dienen. [57, p. 3] Simulationen kommen in diesem Feld also eine große Bedeutung zu, da sie häufig die Grundlage zum Verstehen systembiologisch betrachteter Prozesse bilden. [56, pp. 5, 153] Dabei kommen sie hauptsächlich dann zum Einsatz, wenn in der für eine Untersuchung begrenzt zur Verfügung stehenden Zeit eine sichere mathematische Herleitung fester Aussagen nicht möglich ist. [57, p. 36]

2.3 Statistische Aussagekraft

Die statistische Datenanalyse, also das Auswerten zuvor ermittelter Daten, stellt einen wichtigen Bestandteil diverser Untersuchungsmethoden dar und ist damit fundamental für die Forschung vieler wissenschaftlicher Felder. Da die Datenauswertung auf diverse Weisen geschehen kann, gewährt sie Forschern großen Freiraum, erschwert jedoch auch eine korrekte Anwendung, die aussagekräftige Ergebnisse erzielt. [58, pp. 1, 4] Eine möglichst vollständig akkurate Datenanalyse baut dabei auf den zugrunde liegenden Daten auf – eine hohe Qualität der genutzten Datenquelle stellt also eine essenzielle Grundlage dar.

Wissenschaftliche Untersuchungen versuchen üblicherweise, Unterschiede beziehungsweise Abweichungen festzustellen, welche die Versuchsobjekte betreffende Effekte charakterisieren. Besagte Abweichungen können jedoch auch rein zufällig auftreten, ohne durch einen tatsächlich relevanten Effekt ausgelöst worden zu sein. Entsprechend ist es wichtig, eine hohe statistische Signifikanz beobachteter Effekte sicherzustellen. Dabei kommt der Größe eines vermeintlich aufgetretenen Effekts zunächst keine Relevanz zu; vielmehr muss eine hohe Wahrscheinlichkeit seines Auftretens abseits von Zufällen bestätigt werden. [58, pp. 7, 9]

Einen ausschlaggebenden Faktor stellt dabei die sogenannte Teststärke dar, welche angibt, wie wahrscheinlich bei einer konkreten Untersuchung ein tatsächlicher Effekt von einem Zufall unterschieden werden und somit ein statistisch signifikantes Ergebnis erzielt werden kann. Eine zu geringe Teststärke führt zu Ergebnissen ohne realen Aussagewert und stellt dadurch ein in

Untersuchungen vieler wissenschaftlicher Felder häufig auftretendes Problem dar. [58, pp. 15, 17-19]

Der konkrete Wert der Teststärke variiert stark je nach Untersuchungssubjekt und den zur Datenerfassung angewandten Methoden; ein für den Wert konstant wichtiger Faktor findet sich jedoch im Stichprobenumfang. So erhöht sich die Aussagekraft einer Untersuchung stets mit der Größe des Stichprobenumfangs, also der Menge an erfassten Daten. Tatsächliche Effekte sind also grundsätzlich leichter erkennbar, wenn die gesammelte Datenmenge, jedoch nicht zwingend die Menge unterschiedlicher Daten, erhöht wird. Je nach Untersuchung kann dies etwa durch eine Erhöhung der Menge an Testsubjekten oder eine Erweiterung des betrachteten Zeitraums geschehen. [58, pp. 8, 17-18, 23]

Ein dabei relevantes Problem ergibt sich dadurch, dass ein für eine Untersuchung aussagekräftiger Stichprobenumfang zumeist weit größer ist, als von Beteiligten intuitiv angenommen. Zudem steigt die benötigte Stichprobengröße exponentiell weiter an, je kleiner der betrachtete Effekt ist, da der Einfluss des Zufalls entsprechend stark steigt. Auf der anderen Seite ist das Erhöhen der tatsächlich verwendeten Stichprobengröße meist nur eingeschränkt möglich und stößt so oft an die praktischen Grenzen des Experiments. [58, pp. 18, 20, 25]

Beim Einsatz von Simulationen ist, wie auch bei anderen Datenerfassungsmethoden, stets eine hohe statistische Sicherheit erwünscht. [57, p. 36] Einzelne Simulationsabläufe oder solche, die über einen nur sehr geringen Zeitraum laufen, sind dabei zumeist nicht ausreichend, um aussagekräftige Informationen zu gewinnen. [22, p. 6] Die Laufzeit der betrachteten Simulationen, wie auch die Menge an Simulationsdurchläufen sollte entsprechend möglichst hoch angesetzt werden. Zudem bietet sich häufig ein Vergrößern des Simulationsumfangs selbst an, um die Stichprobengröße zu erhöhen und zusätzlich ein gleichförmigeres Simulationsverhalten zu erzielen. Entsprechend profitieren Simulationen in vielen Fällen von hoher Skalierbarkeit und daran gebundener guter Performanz.

3 Unity DOTS

Unity DOTS ist eine in Entwicklung befindliche Erweiterung der Unity-Entwicklungsumgebung. In den nachfolgenden Abschnitten wird zunächst Unity an sich sowie Unity DOTS im Speziellen vorgestellt.

3.1 Die Unity-Entwicklungsumgebung

Bei Unity handelt es sich um eine Entwicklungsplattform für Echtzeitanwendungen, welche im Jahr 2005 veröffentlicht wurde und seither regelmäßig aktualisiert wird. [59]. Die Anwendungsentwicklung mit Unity ist leicht zugänglich gestaltet und erlaubt so ein agiles Umsetzen von Ideen. Auch dadurch ist Unity als Entwicklungsumgebung stark verbreitet und es existiert eine Vielzahl an Ressourcen zum Erlernen ihrer diversen Komponenten und zur generellen Unterstützung Entwickelnder. [60]

Im Folgenden werden zunächst einige der Konzepte der Unity-Entwicklungsumgebung näher erläutert.

3.1.1 Einsatzgebiet

Ein zentrales Merkmal der Unity-Entwicklungsplattform ist es, dass in ihr erstellte Anwendungen unkompliziert für eine Vielzahl an digitalen Plattformen bereitgestellt werden können, ohne je nach Plattform starke Anpassungen vornehmen zu müssen. Konkret wird die Entwicklung von Anwendungen für Desktop-Systeme, Mobilgeräte, Spiel-Konsolen, Webseiten und diverse XR-Plattformen unterstützt. [60] [61] [62] Diese hohe Kompatibilität mit geringem einhergehenden Aufwand auf Seiten Anwendender stellt einen der hauptsächlichen Faktoren für die verbreitete Nutzung der Entwicklungsumgebung dar.

Laut eigener Aussage findet Unity dabei Anwendung in über 50 Prozent der für Computer, Konsolen und Mobilgeräte entwickelten Spiele. [63] Die Menge an jährlich veröffentlichten Unity-basierten Spielen und Anwendenden der Entwicklungsplattform steigt zudem kontinuierlich an. [64]

War Unity zunächst als reine Spiel-Engine zur Entwicklung digitaler Spiele ausgelegt, dient es inzwischen als Grundlage zur Erstellung diverser Anwendungen mit unterschiedlichsten Einsatzgebieten. Dabei bietet sich die Nutzung gerade dann an, wenn umfangreiche Programme mit

dreidimensionalen Inhalten und/oder komplexen Interaktionen umgesetzt werden sollen. Entsprechend findet Unity beispielsweise in der Automobilkonzeption, der Entwicklung künstlicher Intelligenz, der Filmproduktion sowie diversen weiteren Anwendungsfeldern Verwendung in unterschiedlichem Ausmaß. [59] [65] [66] [67] [68]

Durch die bereits angesprochene Nähe digitaler Spiele und Simulationen ist Unity zudem sehr gut zur Umsetzung von simulationsbezogenen Projekten geeignet. So verwendet der zuvor vorgestellte YouTube-Kanal Primer zur Umsetzung seiner Simulationen Unity und auch Sebastian Lague nutzte die Entwicklungsumgebung für das erwähnte Video „Coding Adventure: Simulating an Ecosystem“. Als Spiel-Engine kommt Unity des Weiteren in den behandelten Spielen Kerbal Space Program sowie RimWorld zum Einsatz und dient ebenfalls als Entwicklungsplattform für die von CodeParade erstellten Spiele.

3.1.2 Aktualisierung der Entwicklungssoftware

Unity wird regelmäßig aktualisiert, wobei der Plattform neue Funktionen hinzugefügt sowie Bestehende überarbeitet und/oder erweitert werden. Zudem werden in diesem Rahmen in Unity auftretende Fehler behoben und das Nutzungserlebnis im Allgemeinen verbessert. Diese Aktualisierungen erscheinen je nach Umfang mehrmals jährlich oder sogar mehrmals monatlich, wobei es Nutzenden freisteht, welche Version sie verwenden sowie ob und wann sie Aktualisierungen durchführen möchten. Organisiert sind die Unity-Versionen dabei grob nach dem Jahr ihrer Entwicklung, welches sich prominent in ihrer Versionsnummer wiederfindet.

Für jedes Jahr wird dabei letztlich eine Version mit „Long Term Support“ (LTS) veröffentlicht. Diese soll ein sehr stabiles Nutzungserlebnis gewährleisten, da ihre Funktionen nicht verändert und keine neuen Funktionen zu ihr hinzugefügt werden. Stattdessen erhalten diese Sonderversionen über zwei Jahre hinweg regelmäßige Aktualisierungen, welche lediglich zum Großteil aus neueren Unity-Versionen stammende Korrekturen für Fehler beinhalten. So soll eine möglichst lange Nutzung für Projekte möglich gemacht werden, ohne dass durch Aktualisierungen Inkompatibilitäten auftreten. Gleichzeitig soll die Menge an auftretenden Fehlern minimiert werden, um eine saubere Entwicklung zu ermöglichen. [69] Somit sind LTS-Versionen für den Einsatz in zur Veröffentlichung vorgesehenen Projekten mit großem Umfang vorgesehen.

3.1.3 Konzept der Anwendungserstellung

Unity-Anwendungen sind aus beliebig vielen Szenen aufgebaut, wobei es sich um dreidimensionale Räume handelt, in welchen verschiedenste Objekte beliebig platziert werden können. Die

Eigenschaften dieser Objekte können daraufhin manipuliert werden, wofür sogenannte Skripte zum Einsatz kommen. Bei diesen Skripten handelt es sich um kleine Programme, welche in der Programmiersprache C# geschrieben werden. Diese können eine Vielzahl vorgefertigter Code-Komponenten nutzen, welche von Unity für häufige Anwendungsfälle zur Verfügung gestellt werden. In den erstellten Szenen werden zudem üblicherweise Kameras platziert, welche den Szeneninhalt einfangen und als Ansicht für die die Anwendung nutzende Person ausgeben. [2]

Des Weiteren bietet Unity diverse optionale Komponenten, um mit der erstellten Anwendung möglichst vielfältige Verwendungszwecke zu unterstützen. So existieren beispielsweise vorgefertigte Objekte für Audio, Nutzereingaben oder Interkation mit Internetinhalten. Durch einen in neueren Versionen verstärkt eingesetzten modularen Aufbau ist es zudem möglich, zusätzliche Erweiterungen von Unity selbst oder Drittanbietern einzubinden, welche den Funktionsumfang erweitern und so die Entwicklungsumgebung weiter auf den eigenen Anwendungsfall zuschneiden. [70]

Von Unity werden zudem verschiedene Dienste angeboten, die in Projekte eingebunden werden können, um zusätzliche Funktionalitäten zu integrieren. So werden unter anderem Werkzeuge zur Software-Nutzungsanalyse, Einbindung von Werbeanzeigen und Kommunikation über Netzwerke bereitgestellt. [71]

3.1.4 Der Unity-Editor

Zum Erstellen von Anwendungen dient der Unity-Editor, ein Programm zum Anfertigen und Bearbeiten von Szenen sowie Skripten und zum Festlegen diverser Verhaltensparameter besagter Anwendung. Der Editor, wie er in Abbildung 1 zu sehen ist, beinhaltet dafür zunächst eine Szenenansicht, welche die aktuell bearbeitete Szene aus Sicht einer Kamera darstellt, die in ihr frei bewegt werden kann. Alle in der Szene befindlichen Objekte sind in einem Hierarchie-Fenster aufgelistet, das zudem die Eltern-Kind-Beziehungen, bei denen ein oder mehrere Objekte einem anderen untergeordnet sind, in der Szene darstellt. Über die Szenenansicht oder das Hierarchie-Fenster können die in der aktuellen Szene befindlichen Objekte ausgewählt werden, wodurch ihre jeweiligen Eigenschaften im Inspektor-Fenster in Listenform dargestellt werden. Besagte Eigenschaften können dabei entweder über den Inspektor oder direkt in der Szenenansicht manipuliert werden. [2]

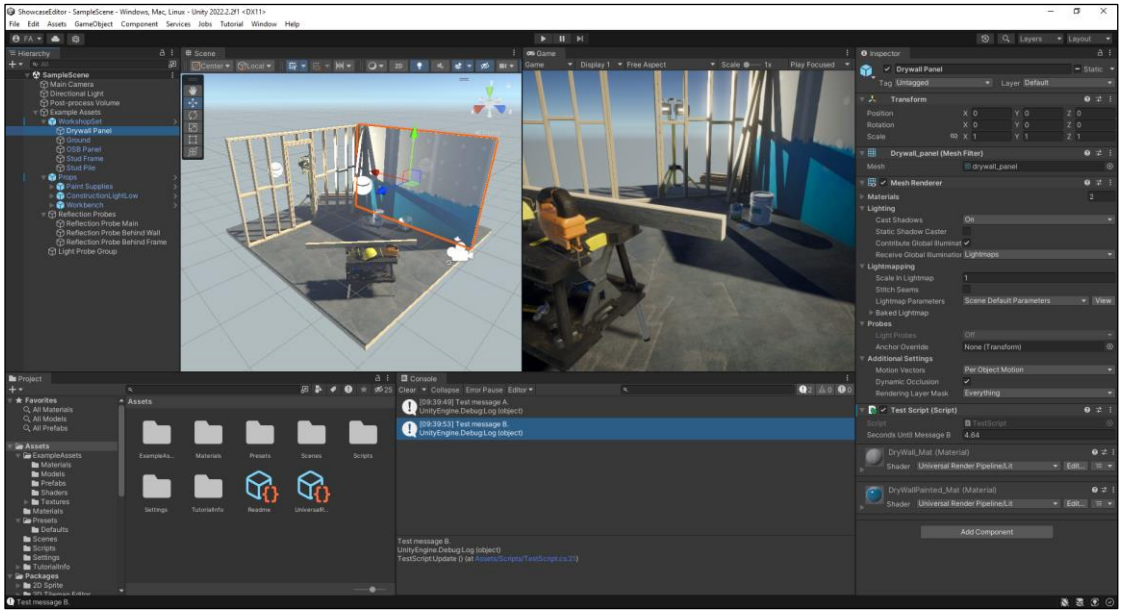

Abbildung 1: Der Unity-Editor mit seinen grundlegenden Werkzeugen

In der Anwendung benötigte Dateien können über das Projektfenster, welches die Ordnerstruktur des Projekts auf dem verwendeten PC darstellt, in das Projekt geladen werden. Dabei ist zum Teil eine Bearbeitung der Dateien über den Inspektor möglich; etwa um Bilddateien für die Grafikkarte zu optimieren. Der Editor enthält zudem die sogenannte Spielansicht, in welcher eine Vorschau der letztlich erzeugten Anwendung gestartet werden kann, um Interkationen mit ihr zu testen. Zur Unterstützung solcher Tests dient zudem die Konsole – ein weiteres Fenster, in welchem Fehlermeldungen, Warnungen und generelles Feedback des Programmcodes ausgegeben werden. Die entsprechenden Meldungen beinhalten dabei einen Zeitstempel sowie einen Link zu dem für sie jeweils relevanten Codeabschnitt, um als *Debugging*-Hilfe zu agieren. [2]

Neben diesen grundlegenden Komponenten bietet der Unity-Editor noch diverse weitere Fenster und Dialoge, mit denen verschiedene Komponenten der Anwendung eingerichtet werden können. Diese können je nach Bedarf für das aktuelle Projekt jederzeit aktiviert beziehungsweise deaktiviert und, analog zu den grundlegenden Editorbestandteilen, frei angeordnet werden. [2] So kann er an Projektanforderungen sowie persönliche Arbeitsabläufe angepasst werden.

Um die erstellte Anwendung zu optimieren, stellt Unity zudem verschiedene in den Editor integrierte Debugging-Werkzeuge bereit, welche die Auswertung diverser performanzbezogener Parameter ermöglichen. Besonders herauszustellen ist dabei der Unity Profiler, der beim Ausführen der Anwendung Aufzeichnungen erstellt und so beispielsweise für jeden Zeitpunkt festhält, wie viel Rechenzeit für verschiedene Aufgaben benötigt wurde. [72] [73]

3.2 Unity DOTS und seine Komponenten

Beim Data Oriented Technology Stack für Unity handelt es sich um eine Kombination verschiedener Technologien. Diese basieren dabei namensgebend auf *datenorientiertem* Design, im Gegensatz zur in der modernen Programmierung vorherrschenden *Objektorientierung*. [74] DOTS soll es so ermöglichen, mit Unity komplexere Anwendungen zu erstellen und diese mühelos zu skalieren. Entsprechende Anwendungen sollen dadurch etwa auf unterschiedlich leistungsfähiger Hardware ohne performanzbezogene Probleme lauffähig sein. [74] [75] Dabei soll Entwickelnden zudem mehr Kontrolle über ihre Projekte und ihren Programmcode im Speziellen gegeben werden. [75]

DOTS ist in drei wesentliche Komponenten gegliedert, die unterschiedliche Bestandteile des Unity-Entwicklungsprozesses auf unterschiedliche Arten verbessern respektive optimieren sollen. [74] Diese sind unabhängig voneinander nutzbar, profitieren aber von einer gemeinsamen Verwendung durch zusätzliche Performanzverbesserung. [76] Namentlich handelt es sich bei den Bestandteilen um das „C# Job System“, den „Burst Compiler“ und das „Entity Component System“ (ECS), welche nachfolgend jeweils näher vorgestellt werden. [74]

3.2.1 C# Job System

Das C# Job System soll Entwickelnden eine einfachere Nutzung von Multithreading erlauben, wodurch Programmcode auf einem oder mehreren Prozessorkernen parallel ausgeführt werden kann. [74] [77] So sollen starke Performanzverbesserungen ermöglicht werden, ohne dass die im Code ausgeführten Aktionen an sich optimiert werden müssen. [77] Auch soll das System aussagekräftigeres Feedback geben, wenn Probleme auftreten, um deren Beseitigung möglichst einfach und effektiv zu gestalten. [78]

Zur Nutzung des C# Job System müssen Entwickelnde ihren Programmcode in voneinander unabhängige Jobs aufteilen, welche dann automatisiert auf die verfügbaren Prozessorkerne aufgeteilt werden. [78] [79] Dabei können zudem Kerne, die ihre zugewiesenen Aufgaben bereits abgeschlossen haben, weitere Jobs, welche ursprünglich anderen Kernen zugewiesen wurden, übernehmen und so Wartezeiten weiter verringern. Dadurch ist für eine schnelle Ausführung des Programmcodes auch die relative Größe einzelner Jobs unerheblich, solange die von ihnen ausgeführten Aktionen nicht voneinander abhängig sind. [77]

Das C# Job System ist zudem grundlegend darauf ausgelegt, Probleme zu vermeiden, welche durch parallel laufenden Code auftreten können. [74] [77] Greifen beispielsweise verschiedene

parallel laufende Systeme auf gleiche Daten zu, können sich Resultate ändern, je nachdem welches System die geteilten Daten zuerst bearbeitet hat. Unter Umständen kann es sogar zu gleichzeitigen Zugriffen kommen, welche Ergebnisse unvorhersehbar machen und verfälschen können. Um solche Probleme zu vermeiden muss sichergestellt werden, dass Daten gleichzeitig von mehreren Systemen gelesen, aber lediglich von je einem System auf einmal bearbeitet werden können. Um dies zu gewährleisten, forciert das C# Job System stets eine Spezifizierung aller Datenzugriffe; es muss also für alle verwendeten Daten festgelegt werden, ob diese nur gelesen oder auch bearbeitet werden. Das C# Job System verhindert daraufhin mögliche Konflikte automatisch, indem der Zugriff auf bereits von Jobs verwendete Daten gesperrt wird. Andere Jobs, welche auf die Daten zugreifen möchten, werden dann zunächst pausiert, bis diese Sperre aufgebhoben ist. [78]

3.2.2 Burst Compiler

Bei einem *Compiler* handelt es sich um ein Programm, welches in einer bestimmten Programmiersprache geschriebenen Programmcode in Code einer anderen Programmiersprache umwandelt. Üblicherweise wird der Code so in eine Hardware-nähere Sprache übersetzt, sodass er als Programm ausgeführt werden kann. Der Compiler nimmt dabei zudem Optimierungen am Code vor, um dessen Performanz nach Möglichkeit zu verbessern.

In regulären Unity-System wird der in C# geschriebene Code zunächst in performanteren C++-Code umgewandelt, welcher daraufhin wiederum in nativen *Maschinencode* übertragen wird. Diese Aufteilung geschieht, da C++ bessere Hardware-Optimierung ermöglicht, jedoch auch Kenntnis fortgeschrittenerer Programmierkonzepte voraussetzt und höhere Compiler-Wartezeiten mit sich bringt. Das weniger performante sowie einfacher zugängliche C# wird daher als Oberflächen-Sprache verwendet, während C++ als Zwischenschritt zur Optimierung Verwendung findet. [78]

Der Burst Compiler stellt eine neue Compiler-Technologie dar, welche die Performance von DOTS-basierten Unity-Projekten steigern soll. [80] Anders als beim klassischen System, wandelt er dabei den von Unity Nutzenden geschriebenen C#-Code direkt in Maschinencode um. [74] [80] Mit diesem eigens für Unity entwickelten Compiler soll auf die Umwandlung des Programmcodes mehr Einfluss genommen werden können, um ihn so besser für die letztlich verwendete Hardware zu optimieren und so die für seine Ausführung benötigte Rechenzeit zu reduzieren. [77]

Um noch höhere Kontrolle zu ermöglichen, limitiert der Burst Compiler zudem die durch Entwickelnde nutzbaren C#-Bestandteile auf das sogenannte High-Performance-C#. [78] [80]

Dadurch werden die Arten, auf die Code geschrieben werden kann eingeschränkt, wodurch der Programmcode besser einschätzbar wird. So kann der Burst Compiler letztlich in Fällen, in denen normalerweise aufgrund von fehlendem Wissen zum Code bestimmte Optimierungen übersprungen worden wären, diese doch vornehmen. [78] [81]

Nutzenden von Unity wird des Weiteren der letztlich erzeugte Maschinencode leichter zugänglich gemacht, um es ihnen zu erleichtern, Optimierungen an ihrem ursprünglichen Programmcode vorzunehmen. [82] Unity selbst soll zudem überarbeitet werden, um selbst ebenfalls den Burst Compiler nutzendes C# statt C++ zu verwenden. Da Unity so letztlich die gleiche Programmiersprache verwendet, wie die darin entwickelten Anwendungen, soll es so für Nutzende auch einfacher werden, Systeme der Entwicklungsumgebung anzupassen, zu erweitern oder sie gänzlich neu zu implementieren, um sie besser auf Projekte zuschneiden zu können. [78]

3.2.3 Entity Component System

Wie vorhergehend erläutert, nutzt der Unity-Editor für die Erstellung von Anwendungen standardmäßig ein objektorientiertes System, in dem alle Anwendungsinhalte als mit Komponenten versehene Objekte erstellt werden. Dieser Ansatz spiegelt sich ebenfalls bei der Programmierung von Skripten wider, bei welcher der Programmcode hauptsächlich Objekte mit modifizierbaren Eigenschaften erstellt, die von anderem Code bearbeitet werden können. Solche objektorientierten Ansätze sind in der modernen Programmierung durchaus verbreitet, da sie ein für Menschen einfach verständliches, als natürlich empfundenes Konzept darstellen. Dem Vorteil der Zugänglichkeit steht dabei entgegen, dass die durch dieses System erzeugten Daten sehr unorganisiert und, etwa für Compiler oder die den Code ausführende Hardware, entsprechend schwer verwaltbar sind, wodurch Performanzprobleme auftreten können. [83]

Das Entity Component System soll eine Alternative zum klassischen Entwicklungssystem darstellen, welche auf die Beseitigung der benannten Nachteile ausgelegt ist. Es handelt sich dabei um ein datenorientiertes Framework, welches durch Fokussierung auf bessere Verwaltung der den Anwendungen zugrundeliegenden Daten Entwickelnden mehr Kontrolle über ihren Code geben und ambitioniertere Projekte ermöglichen soll. [74] [84]

Die größte Verbesserung soll sich dabei bei der Nutzung des CPU-Cache einstellen. So soll die für die Ausführung von Programmcode benötigte Rechenzeit stark sinken, indem Daten Cachefreundlicher gestaltet werden. [77] Aus diesem extrem performanten temporären Speicher bezieht der Prozessor alle für die aktuelle Code-Ausführung benötigten Daten. Dabei lädt er stets eine feste Menge hintereinander im Speicher liegender Daten, beginnend mit den aktuell benötigten. [83] [85] Sind diese, wie im klassischen Unity-System üblich, unsortiert, stellt dabei der

Großteil der zusätzlich geladenen Daten momentan nicht benötigten Ballast dar, welcher nicht genutzt und wieder entladen wird. Auf weitere tatsächlich gebrauchte Daten muss entsprechend zumeist mit vielen weiteren Ladevorgängen separat zugegriffen werden, was hohe Wartezeiten zur Folge hat. [83]

Alle Objekte und Komponenten im klassischen System werden ohne größere Logik völlig separat voneinander im Speicher platziert. Beim Ausführen einer mit Unity erstellten Anwendung werden zudem die Skripte der Komponenten aller Objekte einzeln nacheinander bearbeitet. In der dabei größtenteils zufällig gewählten Reihenfolge kommt es häufig vor, dass aufeinanderfolgend völlig verschiedene Skripte ausgeführt werden, welche entsprechend unterschiedliche Daten benötigen und so bedingen, dass der Prozessor enorm häufig neue Anfragen an den Cache stellen muss. Besonders bei großen Objekt- und Komponentenmengen ist dieser Prozess entsprechend sehr ineffizient und führt zu schlechter Performanz. [83]

Anders als dieses klassische System ist ECS darauf ausgelegt, die Objekte, welche hier als „Entitäten“ bezeichnet werden, nach den ihnen zugewiesenen Komponenten in Archetypen zu sortieren. Dabei werden alle Entitäten eines Archetyps im Speicher direkt hintereinander platziert. Die Komponenten-Skripte selbst beinhalten zudem keinen Programmcode zum Manipulieren ihrer Daten mehr; stattdessen werden sogenannte Systeme genutzt. Dabei handelt es sich um separate Skripte, welche die in ihrem Code definierten Aktionen auf eine Menge an Komponenten, und somit Entitäten, anwenden. Sie lassen dabei aus dem Cache alle Entitäten laden, welche bestimmte Komponentenarten besitzen, und manipulieren diese dann nacheinander, statt einzelne Komponenten zu bearbeiten. [83]

Beispielhaft kann man dies an einem System erklären, welches Entitäten physisch simuliert zu einem Punkt anziehen soll. Dieses könnte alle Entitäten aus dem Speicher laden lassen, welche eine Positions-Komponente und eine Physikeinfluss-Komponente besitzen. Basierend auf den Daten der Positions-Komponente würde es dann in der zugehörigen Physikeinfluss-Komponente die auf die Entität wirkende Kraft anpassen. Ein separates Physik-System würde daraufhin alle Entitäten entsprechend der gesetzten Kraft bewegen.

Der Cache-Ladeprozess wäre dabei enorm effizient, da die relevanten Entitäten dank übereinstimmender Archetypen im Speicher direkt nebeneinander positioniert gewesen wären. Die jeweils aus dem Cache geladene Datenmenge würde so also kaum Ballast-Daten beinhalten, wodurch nur wenige Cache-Abfragen zum vollständigen Ausführen des Systems notwendig wären. Die Entitäten nehmen zudem deutlich weniger Speicherplatz ein als die klassischen Unity-Objekte, wodurch eine einzelne Cache-Anfrage generell deutlich größere Mengen an Entitäten laden kann. [83]

Dieser Ablauf ist dabei gut für eine Verwendung mit Multithreading geeignet und vom Compiler stark optimierbar. ECS ist entsprechend sehr gut mit dem C# Job System sowie dem Burst Compiler kompatibel und kann in Verbindung mit ihnen noch höhere Performanzgewinne erzielen. ECS-Systeme müssen zudem stets definieren, ob Komponenten-Daten nur gelesen oder auch bearbeitet werden, wodurch die Zusammenarbeit mit dem C#-Job-System zusätzlich optimiert wird. [83]

Durch die Sortierung der Entitäten in Archetypen ergibt sich zudem ein weiterer Leistungsvorteil beim Erstellen neuer Entitäten. Der Speicher ist stets in gleichgroße Abschnitte unterteilt, welche mit Entitäten befüllt werden, wobei nach der letzten Entität eines Archetyps der Rest des aktuellen Abschnitts ungenutzt gelassen wird. Dadurch können neue Entitäten des Archetyps meist einfach in diesen Leerraum eingefügt werden, statt für sie neuen Speicherplatz zuweisen zu müssen. [85]

Letztlich sind Entitäten so vielfältiger nutzbar als klassische Objekte und können in weit größeren Mengen verwendet werden. [83] So ist es optimiert für komplexe, große Szenen, welche einfach ge- und entladen werden können, um auf Systemen unterschiedlichen Leistungsniveaus performant nutzbar zu sein. Simulationen im Speziellen können durch ECS enorm groß skaliert werden und laufen deterministisch ab, wodurch bei Entwicklung oder Einsatz störende Varianz vermieden wird. [84]

ECS ist dabei mit dem klassischen Unity-System beliebig kombinierbar, um besser auf den jeweiligen Einsatzzweck abstimmbar zu sein. [83] [84] [85] So kann sein Einsatz beispielsweise auf Anwendungsteile beschränkt werden, welche hohe Performanz erfordern, während der Großteil eines Projekts weiter im leichter zugänglichen Standardsystem entwickelt wird. Auch können bereits bestehende Systeme, welche auf klassische Weise entwickelt worden, weiter genutzt werden, ohne auf ECS-Optimierungen an anderer Stelle verzichten zu müssen. [83] [86]

Der Großteil der für die Umstellung auf dieses datenorientierte System notwendigen Anpassungen wird von ECS bereitgestellt, wodurch für die Nutzung lediglich eine angepasste Struktur für Skripte und Szenen implementiert werden muss. [83] Um die Entwicklung mit ECS weiter zu vereinfachen, ist das Framework zudem in den Unity-Editor direkt integriert und kann dort mit den Werkzeugen des klassischen Entwicklungssystems verwendet werden. [74] [75] [84] So können Entitäten beispielsweise ausgewählt und ihre Eigenschaften daraufhin im Inspektor bearbeitet werden – analog zu regulären Objekten. Auch können Entitäten zunächst als Objekte erstellt und erst beim Anwendungsstart umgewandelt werden, um gänzlich den vertrauten Arbeitsprozess nutzen zu können.

3.3 Anwendungsgebiet

Die Leistung rechenintensiver Anwendungen, gerade im für Unity besonders relevanten Bereich der digitalen Spiele, wird häufig durch einzelne Hardwarekomponenten limitiert, welche als Engpass agieren. So reizen grafisch intensive Programme häufig die maximale Leistungsfähigkeit der verwendeten Grafikkarte aus. Andererseits kann jedoch auch die Leistung des genutzten Prozessors einen Engpass darstellen. Vorrangig ist dies der Fall bei Anwendung mit einer großen Menge an Inhalten, für welche regelmäßige Berechnungen erforderlich sind – beispielsweise für Interaktivität und/oder eigenständiges Verhalten, vor allem im Rahmen künstlicher Intelligenz. [87] Besonders bei Simulationen mit vielen involvierten Bestandteilen ist die Prozessorleistung entsprechend stark ausschlaggebend für die erzielte Performanz.

Gerade die bereits erwähnten digitalen Spiele stellen dabei eine klassische Anwendungsgruppe dar, bei der gewisse Techniken zur Performanzverbesserung kaum Anwendung finden. Dies ist zum Beispiel der Fall bei Multithreading, da die in den Spielen verwendeten Systeme häufig schlecht separierbar sind und ihre Parallelisierung meist nicht einfach skalierbar ist. [88] Zusätzlich stellt die Performanzoptimierung einen durchaus aufwendigen Prozess dar, der bei der Anwendungsplanung von Grund auf bedacht werden sollte und an welchem über die Entwicklungszeit hinweg kontinuierlich gearbeitet werden muss. Dies hat eine häufige Vernachlässigung dieses Optimierungsprozesses zur Folge.

Durch fehlende Optimierung entstehende Leistungsengpässe können sich dabei für Endnutzer durchaus bemerkbar machen. Dabei ist zumeist die von der Anwendung erzielte *Bildfrequenz* der wahrnehmbarste Indikator für entsprechende Performanzprobleme. Diese wird durch die Einheit „Bilder pro Sekunde“ (FPS) angegeben, wobei für Echtzeitanwendungen üblicherweise eine stabile Bildfrequenz von 30 FPS als Minimum und eine Frequenz von 60 FPS als Richtwert für eine angenehme Nutzungserfahrung angesehen werden. Leistungsprobleme machen sich durch die Bildfrequenz im schlimmsten Fall durch merkliches Stocken der visuellen Ausgabe des Programms bemerkbar. Auch bei auditiven Inhalten können Leistungsengpässe ähnliches Stocken bewirken und bei Eingabegeräten Verzögerungen mit sich bringen und/oder dazu führen, dass Eingaben von der Anwendung teilweise gänzlich ignoriert werden.

Entsprechend wurde Unity DOTS hauptsächlich zur Vereinfachung dahingehend entwickelt, auf verschiedenster Hardware angenehm lauffähige Anwendungen mit minimalen Leistungsengpässen zu erstellen. So sollen komplexe Inhalte auch auf leistungsschwachen Systemen, beispielsweise Mobilgeräten, angenehmer nutzbar werden. [75] [76] [77] [80] [83] Zusätzlich soll DOTS es ermöglichen, ambitioniertere Projekte mit größerem Umfang und mehr Inhalten umsetzen zu können. [75] [76] [77] [86] [89] Auch sollen Simulationen, beispielsweise für Physikinteraktionen, durch DOTS komplexer wie auch deterministischer umsetzbar werden und

weit besser skalierbar sein, als es in klassischen Unity-Anwendungen der Fall wäre. [75] [76] [77] [89] Nutzenden soll zudem mehr Kontrolle über ihren Programmcode gegeben werden, um bessere Optimierung zu ermöglichen und so die erzielte Performanz noch weiter verbessern zu können. [75] [89]

Durch den Fokus auf Parallelisierung und Ordnung profitieren dabei besonders Projekte mit einer großen Menge an ähnlich agierenden Inhalten. So nutzt beispielsweise das von Stunlock Studios entwickelte Spiel „V Rising“ DOTS, um eine sehr große Spielwelt mit enorm vielen Inhalten umzusetzen und durch effizientes Laden und Entladen mit guter Performanz nutzbar zu machen. [75] Wiederum nutzt Far North Entertainment die Systeme für ein digitales Spiel, in welchem Spielende sich mit enormen Mengen an Gegnern konfrontiert sehen, welche dank ihres ähnlichen Verhaltens besonders effektiv mit DOTS simuliert werden können. [85]

3.4 Entwicklung und aktueller Stand

Unity DOTS wurde erstmals 2018 der Öffentlichkeit als Entwicklungsversion zugänglich gemacht und kann seither in Projekten genutzt werden. Als noch in Entwicklung befindliche Technologien sollten die DOTS-Komponenten laut Empfehlung von Seiten Unitys dabei jedoch nach Möglichkeit nicht in zur Veröffentlichung vorgesehenen Anwendungen integriert werden. Da durch die fortlaufende Entwicklung und die damit einhergehenden größeren Änderungen an verschiedenen Systemen zunächst wenig Stabilität zu erwarten war, sollten sie vielmehr zu Testzwecken genutzt werden, um Feedback für die weitere Entwicklung geben zu können. [90] [91]

Seit dieser ersten Publizierung von Unity DOTS wurden seine Bestandteile regelmäßig aktualisiert, um ihnen weitere Funktionalitäten hinzuzufügen und/oder Arbeitsabläufe mit den Technologien zu überarbeiten. So ergaben sich über den Entwicklungszeitraum hinweg weitere große Performanzverbesserungen und die den Nutzenden des Systems zugänglichen Werkzeuge wurden regelmäßig erweitert. [82] Der Hauptfokus lag dabei auf der Weiterentwicklung von ECS, während der Burst Compiler sowie das C# Job System kleinere Aktualisierungen erhielten, welche zusammen mit den größeren ECS-Anpassungen veröffentlicht wurden. Diese Priorisierung basierte darauf, dass die beiden letzteren Systeme schon in einer frühen Entwicklungsversion von DOTS als grundlegend abgeschlossen angesehen wurden. So wurde auch ihre Anwendung in über den Rahmen von Tests der Technologie hinausgehenden Projekten bereits empfohlen. [76]

Zunächst wurden im Rahmen dieser Weiterentwicklung regelmäßig kleinere Aktualisierungen der ursprünglichen Testversion von DOTS veröffentlicht. So wurden die zunächst umgesetzten Grundlagen nach und nach erweitert und die Nutzungserfahrung mit den DOTS-Bestandteilen

angepasst an von Nutzenden gegebenes Feedback verbessert. So wurde beispielsweise bereits im April 2018 eine Integration von zu DOTS gehörenden Werkzeugen in den Unity-Editor umgesetzt. [92] Diese rege Weiterentwicklung setzte sich bis zu Version 0.17 von ECS fort, welche am zehnten Dezember 2020 veröffentlicht wurde und zunächst die letzte publizierte Version von DOTS darstellte. [93]

Ab diesem Zeitpunkt entschied man sich von Seiten Unitys dazu, neue Versionen deutlich seltener zu publizieren. Begründet wurde dies durch die zum Teil unerwartet hohe Komplexität der Erweiterung von DOTS, die vermehrt die Anpassung und Überarbeitung einer Vielzahl an zu DOTS und/oder Unity allgemein gehörender Systeme voraussetzte. Die veröffentlichten Vorschauversionen sollten dennoch stets eine grundlegende Stabilität hinsichtlich ihrer Funktionen sowie einer fehlerarmen Nutzung aufweisen. Das Gewährleisten dieses Ziels im Rahmen regelmäßiger Veröffentlichungen hätte aufgrund der großen nötigen Umstrukturierungen also einen hohen Mehraufwand vorausgesetzt und die letztliche Entwicklungsgeschwindigkeit stark eingeschränkt. Stattdessen sollte von diesem Zeitpunkt an eine fokussiertere Umsetzung auf Seiten Unitys ermöglicht werden, welche wenige, deutlich stabilere Versionen von DOTS hervorbringen sollte. Konkret sollte so zunächst mit Version 0.50 ein deutlich fortgeschrittener Zwischenstand erarbeitet werden. Daraufhin sollte direkt Version 1.0 als erstmalig abgeschlossene und zur Verwendung jenseits von Testprojekten empfohlene Version folgen. [86]

Entsprechend dieses Plans wurde im März 2022 Entwicklungsversion 0.50 von Unity DOTS öffentlich gemacht und brachte viele größere Verbesserungen mit sich. [76] [90] So wurde unter anderem die Integration in den Unity-Editor erweitert, Systeme zum leichteren Debugging umgesetzt sowie viele DOTS-spezifische Arbeitsprozesse überarbeitet. Insgesamt wurden dadurch Leistungssteigerung sowie eine bessere Kompatibilität zu bestehenden Unity-Systemen erzielt. [90] Im Rahmen dieses großen Versionssprungs wurde zudem die zur Nutzung von DOTS benötigte Unity-Version aktualisiert. So sollte die direkte Nutzung neuer Unity-Bestandteile mit DOTS ermöglicht werden, während die Unterstützung inzwischen entfernter Funktionen aus Effizienzgründen eingestellt werden konnte. Konkret wurde so für die Verwendung von DOTS 0.50 eine Version von Unity 2020 vorausgesetzt. [76] [90] DOTS 0.50 wurde nach kurzer Zeit auf Version 0.51 aktualisiert, welche die Kompatibilität mit neueren Unity-Versionen weiter erhöhen und den DOTS-Programmcode im Sinne der Effizienzsteigerung aufräumen sollte. Diese Aktualisierung erhöhte zudem die vorausgesetzte Mindestversion von Unity weiter auf 2021. [76]

Ebenfalls plangemäß wurde im September 2022 eine erste experimentelle Version von DOTS 1.0 veröffentlicht. [89] In dieser wurden erneut Anpassungen an ECS-Arbeitsabläufen vorgenommen, um das Entwickeln und Testen ECS-basierter System performanter sowie schneller

zu gestalten. Des Weiteren wurden diverse bestehende Komponenten von ECS überarbeitet, um sie leistungsstärker und einfacher nutzbar zu machen. Insgesamt wurde die Verwendung von ECS dabei auch so angepasst, dass Abläufe dem Standardsystems von Unity ähnlicher sind, um die Einstiegshürde zur Anwendung des Systems zu senken. [89] [94] Weitere Verbesserungen wurden an den Debugging-Werkzeugen sowie der Integration von DOTS in den Unity-Editor vorgenommen. [86] [91] Auch wurde die Nutzbarkeit des Burst Compilers für unterschiedlichen Programmcode ausgeweitet, die Darstellung umfangreicher virtueller Welten auf performante Art und Weise optimiert und diverse DOTS-basierte Komponenten von Unity überarbeitet. [89] [94]

Neben diesen sehr sichtbaren Anpassungen wurde auch der zugrundeliegende DOTS-Programmcode erneut aufgeräumt, indem nicht mehr verwendete Systeme entfernt und der verbleibende Code optimiert wurde, womit zudem diverse Fehlerkorrekturen einhergingen. [89] [94] Die verfügbare Dokumentation wurde aktualisiert und stark ausgeweitet, um Entwickelnden einen möglichst guten Einstieg in die Nutzung von DOTS zu gewährleisten. [86] [89] Zusätzlich wurde zu diesem Zweck die Bereitstellung verschiedener Lernangebote für die Technologien in Aussicht gestellt. [86]

DOTS 1.0 ging wie die vorhergehenden Versionen mit einer Aktualisierung der mindestens benötigten Unity-Version, in diesem Fall Unity 2022, einher. [76] Zum jetzigen Zeitpunkt besteht DOTS 1.0 dabei weiterhin als Entwicklungsversion, an welcher Test durchgeführt und kleinere Änderungen vorgenommen werden. [91] Nach Abschluss dieser Testphase sollen die DOTS-Systeme in die offizielle Funktionspalette von Unity aufgenommen werden, womit eine Integration in die darauffolgend veröffentlichten LTS-Versionen einhergeht. Zudem soll DOTS so einen höheren Qualitätssicherungs-Standard erfahren, wobei statt möglichst rapider Weiterentwicklung mehr Fokus auf die Wartung der DOTS-Bestandteile gelegt werden soll. [76]

Somit befinden sich aktuell sowohl das C# Job System, wie auch der Burst Compiler und das Entity Component System in einem zunächst abgeschlossenen Zustand, an welchem keine grundlegenden Änderungen mehr vorgenommen werden sollen. Dadurch ist anzunehmen, dass DOTS vermehrt Anwendung in Projekten finden sollte – spätestens nachdem die momentan ablaufende Testphase abgeschlossen ist und DOTS 1.0 als offiziell fertige Version bereitgestellt wird. [75] Aktuell bestehende Probleme, welche den Einstieg in die Entwicklung mit DOTS erschweren, finden sich vornehmlich in Form der knappen, an vielen Stellen unfertigen Dokumentation der Systeme und der stark begrenzten Menge an Hilfsressourcen und Diskurs unter Nutzenden. Diese stellen bei den regulären Unity-Systemen einen großen Vorteil, vor allem in Bezug auf Einsteigerfreundlichkeit, dar.

Viele von Unity bereitgestellte Basiskomponenten, beispielsweise für Physiksimulation, Kommunikation über Online-Netzwerke oder grafische Darstellung der Anwendungen, wurden unter Nutzung von DOTS neu geschrieben. [75] [83] Dabei handelt es sich um einen weiter andauernden Prozess, durch den immer größere Teile der Entwicklungsumgebung von den neuen Technologien profitieren sollen. So sollen möglichst viele Anwendungsfälle abgedeckt und die durch DOTS erzielten Performancegewinne Nutzenden einfacher zugänglich gemacht werden. [75] Unity DOTS besteht dabei auf absehbare Zeit parallel zum klassischen Unity-System und stellt für Entwickelnde eine von vielen Möglichkeiten dar, Unity auf ihr konkretes Projekt zuzuschneiden. [76] [83]

Zukünftig soll gerade ECS zudem weiterentwickelt werden, um das Potential der Technologie noch besser ausschöpfen zu können. [76] Die durch die Veröffentlichung mögliche breitere Nutzung von DOTS soll zudem als Orientierungspunkt dienen, um Verbesserungen entsprechend der dabei dominierenden Anwendungsfälle vorzunehmen. [89] Unabhängig davon sind für die Weiterentwicklung von DOTS zudem diverse Ansatzpunkte bereits geplant beziehungsweise werden in Erwägung gezogen. [89] [95]

4 Projektaufbau

Im Rahmen dieser Arbeit wird ein praktisches Projekt zur Überprüfung der Anwendbarkeit von Unity DOTS durchgeführt. In den folgenden Abschnitten werden zu diesem Zweck die Hauptaspekte dieses Vorhabens näher erläutert.

4.1 Grundidee

Im Rahmen des Projekts soll eine Desktop-Anwendung erstellt werden, welche die Vorbereitung und Durchführung divers ausgeprägter Populationssimulationen erlaubt. Dabei soll ein Vergleich zwischen der Nutzung des etablierten Unity-Entwicklungssystems und Unity DOTS im Rahmen dieses Anwendungsfalls ermöglicht werden.

In der Anwendung sollen die für die Simulationen genutzten Modelle frei erzielbar sein. So soll die den Rahmen des Modells bildende virtuelle Welt anpassbar sein. Auch sollen die in ihr befindlichen Simulationsobjekte sowie ihre Verteilung in besagter Welt frei wählbar sein. Die Simulationsobjekte selbst sollen analog zu den Modellen ebenfalls mit großer Freiheit erstellbar sein. So sollen Nutzende sie aus vorgefertigten Bauteilen zusammenstellen, durch welche Attribute und Verhalten der so erzeugten Objekte bestimmt werden.

Bei der Durchführung von Simulationen mit den so erstellten Modellen sollen Anwendenden Werkzeuge zum Unterstützen des Beobachtens von Ereignissen sowie der Gesamtentwicklung des Modells und der darin beinhalteten Objekte zur Verfügung gestellt werden. Der die Simulation steuernde Programmcode soll zum einen auf Grundlage des klassischen Unity-Systems umgesetzt, zusätzlich jedoch auch separat mithilfe von DOTS implementiert werden. Nutzenden soll dadurch bei der Durchführung von Simulationen die Wahl zwischen beiden Systemen gegeben werden, um einen möglichst direkten Vergleich zwischen ihrem jeweiligen Einfluss auf die Simulation zu erlauben.

4.2 Anwendungskonzept

Die Anwendung ist grundlegend in drei Bereiche unterteilt, welche größtenteils getrennt voneinander agieren, jedoch aufeinander aufbauen. Konkret handelt es sich bei diesen Bereichen um das Erstellen von Simulationsobjekten, das Anlegen von Modellen mithilfe dieser Simulationsobjekte und das Durchführen von Simulationen anhand der so erzeugten Modelle. Verbunden

sind diese einzelnen Anwendungskomponenten dabei durch eine schlichte grafische Menüstruktur, welche potenziell weitere Anwendungsfunktionen, wie etwa ein Optionsmenü, beinhaltet.

In den nachfolgenden Abschnitten wird jeder der Hauptbereiche in Hinblick auf Aufbau sowie Funktion näher betrachtet und anschließend definiert, was die aus ihnen aufgebaute Anwendung in ihrer Gesamtheit leisten soll.

4.2.1 Erstellen von Simulationsobjekten

Die im ersten Bereich angelegten Simulationsobjekte werden der Einfachheit halber als „Kreaturen“ bezeichnet. Sie müssen jedoch nicht zwingend dem verbreiteten Verständnis des Begriffs der Kreatur als Lebewesen entsprechen, sondern können zum Beispiel auch eine Pflanze oder ein gänzlich unbelebtes Objekt, beispielsweise einen Felsen, repräsentieren.

Als Grundlage für diesen Bereich dient ein Editor mit grafischer Benutzeroberfläche (GUI), in welchem neue Kreaturen angelegt und Bestehende verwaltet werden können. Konkret werden dabei verschiedene vorgegebene Bausteine zur Verfügung gestellt, welche zu einer Kreatur hinzugefügt werden können. Für jeden Baustein ist zudem eine zusätzliche Anpassung über diverse ausfüllbare GUI-Elemente, wie Eingabefelder, Dropdown-Listen oder Kontrollkästchen möglich. So können beispielsweise Verhaltensmuster ausgewählt, Bezeichnungen festgelegt oder Wertebereiche angepasst werden.

Die verfügbaren Bausteine sind nach ihrer Funktion in Attribute, Fähigkeiten und Aktionen aufgeteilt. Attribute stellen dabei Eigenschaften der Kreatur dar, welche als numerische Werte ausgedrückt werden; so beispielsweise die Laufgeschwindigkeit eines Tieres. Nutzer können den Startwert dieser Eigenschaften, ihren möglichen Wertebereich sowie einen sie identifizierenden Titel festlegen. Bei Fähigkeiten handelt es sich um weit fester definierte, vordefinierte Verhaltensmuster, welche von Kreaturen eingesetzt werden können; so beispielsweise das Bewegen durch Laufen, das Angreifen anderer Kreaturen oder die Fortpflanzung. Jede dieser Fähigkeiten besitzt dabei verschiedene für die jeweilige Verhaltensweise relevante Variablen, welche in ihren GUI-Feldern angepasst werden können. Aktionen dienen zum Verbinden von Attributen und Fähigkeiten. Sie besitzen eine wählbare Voraussetzung, beispielsweise das Fallen eines Attributwerts in einen bestimmten Bereich oder die Ausführung einer Fähigkeit. Wird diese Voraussetzung erfüllt, wird eine definierte Handlung durchgeführt, wobei zum Beispiel eine Fähigkeit genutzt oder ein Attributwert angepasst werden kann.

Gerade Attribute und Aktionen sind dabei möglichst offen angelegt und nutzen einen einzigen vielfältig anpassbaren Grundbaustein. Die Funktionsweise von Fähigkeiten muss hingegen individuell implementiert werden; die erstellbaren Simulationen sind in Hinblick auf sie also

durch den Editor-Umfang beschränkt. Um dieser Limitierung entgegenzuwirken sollen Fähigkeiten jedoch möglichst vielfältig einsetzbar gestaltet und durch in der GUI offengelegte Variablen auf möglichst starke Anpassung ausgelegt werden.

Die mit diesem System letztlich erschaffenen Kreaturen sollen durch den Editor gespeichert werden, um eine beliebige Überarbeitung zu späteren Zeitpunkten zu ermöglichen. Dabei werden die Kreaturen in sogenannte „Ökosysteme“ unterteilt, welche eine grundlegende Ordnerstruktur zum besseren Verwalten der Kreaturen darstellen. So soll der Bereich insgesamt dazu dienen, beliebige Mengen möglichst divers aufgebauter Kreaturen zu erstellen.

4.2.2 Anlegen von Modellen

Analog zur Erstellung von Kreaturen dient in diesem Bereich ein ähnlich aufgebauter GUI-basierter Editor zum Anlegen und Verwalten von Modellen. Wie beim Kreaturen-Editor werden in diesem auch grundlegend vorgefertigte Bausteine, die hier die gespeicherten Kreaturen repräsentieren, zu einem ausgewählten Modell hinzugefügt. Für jede so eingefügte Kreatur ist dabei festzulegen, wie sie im Ursprungszustand des Modells platziert wird, indem die Kreaturenmenge und die Art ihrer Verteilung festgelegt werden.

Zusätzlich existieren für jedes Modell festzulegende allgemeine Einstellungsmöglichkeiten. So können der Aufbau der vom Modell genutzten Welt sowie einige Startparameter für den Ablauf von Simulation am Modell festgelegt werden; so beispielsweise ein Startwert für die Berechnung der für die Simulation genutzten Zufallszahlen. Auch kann das für die Simulationen genutzte Programmiersystem, also klassischer Unity-Code oder auf Unity DOTS basierender, ausgewählt werden.

Analog zu den im vorherigen Bereich erstellten Kreaturen werden die angelegten Modelle gespeichert, wobei aufgrund der vermutlich kleineren Menge an erstellten Modellen zunächst keine Ordnerstruktur zum Einsatz kommt. So soll ein mehrmaliges Durchführen von Simulationen am gleichen Modell ohne großen Zeitaufwand erleichtert werden. Zusätzlich wird so das unkomplizierte Anpassen der verschiedenen Modellparameter möglich.

4.2.3 Durchführen von Simulationen

Der finale Bereich der Anwendung dient zur tatsächlichen Wiedergabe von Simulationsabläufen an den zuvor angelegten Modellen. Dabei wird die im Modell definierte Welt dreidimensional dargestellt und die hinzugefügten Kreaturen werden nach den im Modell getroffenen Einstel-

lungen in dieser platziert, woraufhin sie ihre festgelegten Verhaltensweisen frei ausleben können. Durch eine frei im Raum bewegliche Kamera sind die so entstehenden Prozesse dabei beliebig beobachtbar.

Nutzenden werden zusätzlich einige Optionen gegeben, um den Simulationsablauf zu beeinflussen. So sollen verschiedene Merkmale der Simulation, zum Beispiel bestimmte einzelne Interaktionen zwischen Kreaturen oder die Gesamtentwicklung des Modells über einen längeren Zeitraum, besser beobachtbar gemacht werden. Zu diesem Zweck kann beispielsweise die Ablaufgeschwindigkeit der Simulation erhöht oder gesenkt werden.

Die Kernfunktionen dieses Bereichs werden dabei zweimal implementiert. So kommt einmal regulärer Unity-Code zum Einsatz und einmal auf Unity DOTS basierende Skripte, um den Performanzvergleich zwischen beiden Systemen zu ermöglichen. Visuell soll die Simulation insgesamt schlicht gehalten werden, um einen Leistungsengpass auf Seiten der grafischen Ausgabe zu vermeiden. Ein solcher Engpass würde die Simulationsleistung aufgrund der grafischen Leistungsfähigkeit des sie ausführenden Geräts begrenzen, was eine Einschätzung der prozessorbasierten Performanz behindern würde.

4.2.4 Gesamtfunktion der Anwendung

Die durch Kombination der unterschiedlichen Bereiche umgesetzte Anwendung soll zunächst im Rahmen dieser Arbeit dazu dienen, einen Performanzvergleich der beiden genutzten Skriptsysteme vorzunehmen. Abseits davon soll sie es Nutzenden erlauben, möglichst einfach vielfältige Populationsmodelle mit unterschiedlichsten Schwerpunkten zu erstellen. Dabei soll ein möglichst angenehmes Erleben von an diesen Modellen durchgeführten Simulationen ermöglicht werden.

Die so erzeugten Simulationen sollen hauptsächlich der Visualisierung beliebig tiefgreifender Konzepte in Bezug auf Populationen und deren Verhalten dienen, aber keinen explizit wissenschaftlichen Anspruch haben. So bedingt die ermöglichte Vielfältigkeit, dass die Möglichkeiten zur perfekten Abstimmung eines angelegten Modells auf das von ihm abgebildete System begrenzt sind. Gerade die zur Erstellung von Kreaturen verfügbaren Verhaltensweisen sind begrenzt, gleichwohl jedoch möglichst offen gestaltet. Dadurch ist anzunehmen, dass ein Abbilden von Vorlagen durch sie nicht immer perfekt möglich sein wird. Ein Erstellen für wissenschaftliche Untersuchungen einsetzbarer Simulationen ist somit zwar potenziell möglich, jedoch nicht der für das Programm vorgesehene Anwendungszweck.

4.3 Umfang des Prototyps

Die im Projekt erstellte Anwendung ist explizit zunächst als Prototyp konzipiert und entsprechend nicht als fertiges Produkt anzusehen. Konkret ist ihr Funktionsumfang voraussichtlich limitiert, beziehungsweise Funktionen des am Ende des Projekts stehenden Programms sind lediglich oberflächlich respektive unvollständig implementiert. Konkret soll der letztlich abgeschlossene Teil der Anwendung dabei auf die im Rahmen dieser Arbeit relevantesten Funktionen zugeschnitten sein, um das Projekt zeitlich effizient abschließen zu können. Gleichzeitig soll das erläuterte Anwendungskonzept in seiner Gesamtheit zumindest grundlegend abgebildet und auf eine spätere Erweiterung respektive Vervollständigung ausgelegt werden.

Durch das so zunächst abgeschlossene Programm ergibt sich zudem ein entwicklungsbezogener Vorteil, da es als Orientierungspunkt für eine mögliche Weiterentwicklung genutzt werden kann. So kann das Anwendungskonzept, beziehungsweise die konkrete Umsetzung der verschiedenen Programmfunktionen, gestützt auf vollumfängliche Tests der Anwendung angepasst werden. Eine Veröffentlichung der Anwendung ist dabei im Rahmen dieses Projekts sowie generell vor einer solchen Weiterentwicklung nicht geplant.

Zu den im Prototyp nicht abgeschlossenen Anwendungsteilen zählen unter anderem die in den Editoren verfügbaren Anpassungsoptionen. Besonders hervorzuheben ist dabei die Menge unterschiedlicher Fähigkeiten im Kreaturen-Editor, welche zunächst in deutlich begrenztem Umfang umgesetzt werden wird. Zudem wird die Verwaltung gespeicherter Kreaturen sowie Modelle zunächst nur in begrenztem Umfang möglich sein und kein Sortieren respektive Strukturieren zu Übersichtszwecken ermöglichen. Gleichfalls wird die Menge an bei der Durchführung von Simulationen verfügbaren Werkzeugen eingeschränkt sein, wodurch der Einblick in die Simulation sowie die in ihr verorteten Kreaturen begrenzt sein wird.

Abseits davon werden die Menüs, gerade in Hinblick auf Einstellungsmöglichkeiten der Anwendung, einen lediglich grundlegenden Umfang aufweisen. Eine Dokumentation des Programms sowie seiner verschiedenen Funktionen, beispielsweise in Form einer integrierten Nutzungsanleitung, wird zudem, abseits der kurz gefassten Inhalte dieser Arbeit, zunächst nicht vorhanden sein. Dies ergibt sich daraus, dass Umfang wie auch Implementierung der Programmfunktionen durch die nicht abgeschlossene Entwicklung noch stark variabel sein werden und eine Nutzung der Anwendung in dieser Form durch Dritte nicht vorgesehen ist. Entsprechend sind auch ähnliche Maßnahmen zur Optimierung von Nutzungserfahrung zunächst nicht geplant. Visuelle sowie auditive Gestaltungsaspekte des Programms stehen gleichfalls nicht im Fokus des Projekts.

4.4 Resultatsbewertung

Die fertige Anwendung soll letztlich anhand von drei verschiedenen Aspekten ausgewertet werden. Konkret sind dabei die Performanz der beiden für die Simulationen genutzten Skriptsysteme, der jeweils für ihre Implementierung nötige Entwicklungsprozess und das Nutzungserlebnis der Anwendung in Gänze zu betrachten. In den folgenden Abschnitten soll jeder dieser Aspekte und die Art ihrer Untersuchung kurz näher erläutert werden.

4.4.1 Vergleich der Simulationsperformanz

Im Zentrum der Bewertung der letztlich umgesetzten Anwendung steht ein Vergleich der verwendeten Skriptsysteme im Hinblick auf die von ihnen erzielte Performanz. Die dafür benötigten Daten sollen an Simulationen erfasst werden, welche jeweils unter Nutzung der DOTS-basierten wie auch der im Unity-Standardsystem erfolgten Implementierung durchgeführt werden. So sollen verschiedene, nachfolgend näher erläuterte Parameter ausgewertet werden.

Die verbreitetste performanzbezogene Metrik, welche unabhängig von der betrachteten Anwendung eine Einordnung der von ihr erzielten Leistung ermöglicht, stellt die zuvor erwähnte Bildfrequenz dar. Höhere Performanz geht üblicherweise mit einer gesteigerten Bildfrequenz einher; eine beständige Bildfrequenz mit wenigen merklichen Abfällen wird zudem als angenehm empfunden und stellt somit ebenso eine positive Performanzcharakteristik dar.

Mit der Bildfrequenz verbunden ist zudem die Maßeinheit der *Zeit pro Bild*, welche angibt, wie lange die Ausgabe eines einzelnen Bildes dauert. Dieser kommt gerade bei der Entwicklung von Echtzeitanwendungen große Bedeutung zu, da sie neben der Feststellung der Bildfrequenz auch genauere Aussagen zur Flüssigkeit der ausgegebenen Bildfolge ermöglicht. [6] Entsprechend stellen die erzielten Zeiten pro Bild während des Simulationsablaufs die Hauptmetrik zur Einschätzung der erzielten Performanz dar.

Für den Performanzvergleich ist in der Anwendung ein Modell zu erstellen, an dem daraufhin mit beiden Skriptsystemen eine Simulation über einen festgelegten Zeitraum hinweg durchzuführen ist. Das getestete Modell soll die umgesetzten Simulationsfunktionen vielseitig abbilden, um der Gegenüberstellung ein möglichst allgemeines Abbild des Anwendungsinhalts zugrunde zu legen. Während dieser Simulationsdurchführungen sind leistungsbezogene Daten, mit Fokus auf den erzielten Zeiten pro Bild, zu erfassen. Anhand der gesammelten Daten ist daraufhin eine Auswertung der jeweils erzielten Bildfrequenzen sowie ihrer Stabilität vorzunehmen. Sollten zusätzlich beständige Unterschiede in weiteren erfassten Metriken auftreten, sind diese ebenso zu erwähnen und für die Auswertung des jeweils erzielten Performanzabbildes heranzuziehen.

Daten sollen auf verschiedenen PC-Systemen mit je unterschiedlichen Leistungsgraden erfasst werden, um die Performanzcharakteristiken von Unity DOTS auf verschiedenen Leistungsniveaus feststellen und das Spektrum der von System zu System variierenden Leistungsunterschiede besser darstellen zu können. Zur Minimierung von Abweichungen ist es vorgesehen, für alle durchgeführten Simulationen den gleichen Zufallsgenerator-Startwert zu nutzen und die Datenerfassung in einem identischen Zeitraum direkt nach dem Start der jeweiligen Simulation vorzunehmen.

Die Erfassung der für diese Auswertung nötigen Daten ist über den zuvor kurz erwähnten Profiler von Unity vorzunehmen, welcher vielfältige Daten zur Leistung der erstellten Anwendung während ihrer Nutzung kontinuierlich aufzeichnen kann. Zusätzlich soll eine den Funktionsumfang des Profilers ergänzende Erweiterung, der „Profile Analyser“, zum Einsatz kommen. Der Profile Analyser ermöglicht dabei eine vielfältigere Auswertung der gesammelten Daten. Etwa ergänzt er die vom Profiler bereitgestellten Statistiken zu einzelnen von einer Anwendung ausgegebenen Bildern mit solchen zu frei wählbaren Zeiträumen, wodurch etwa die einfache Ermittlung von Durchschnittswerten erleichtert wird.

Da das Ausführen der erstellten Anwendung innerhalb des Unity-Editors deren Leistungsfähigkeit deutlich einschränkt, ist sie für diese Datenerfassung als alleinstehendes Programm zu exportieren, sodass sie ohne den Editor lauffähig ist. Die Anwendung soll zudem so erweitert werden, dass während durchgeführten Simulationen optional Leistungsdaten erfasst und in Dateien gespeichert werden können. Um dies zu ermöglichen, ist der Export als sogenannter „Development Build“ vorzunehmen, wodurch das Programm mit zusätzlichen Werkzeugen zum Debugging und zur Datenerfassung erweitert wird. Die so gespeicherten Leistungsmetriken können daraufhin im Unity-Editor in den Profiler respektive den Profile Analyzer importiert und so ausgewertet werden.

4.4.2 Vergleich der Entwicklungsprozesse

Auf die Gegenüberstellung hinsichtlich der Performanz folgend soll eine subjektive Einschätzung zum Ablauf der Entwicklung der im Projekt erstellten Anwendung gegeben werden. Hauptsächlich ist dabei die Umsetzung des für die Simulationsdurchführung zuständigen Anwendungsteils mit und ohne Verwendung von Unity DOTS vergleichsweise zu beleuchten.

Durch diesen Vergleich soll eine grobe Einordnung des Arbeitsprozesses mit der aktuellen Version von Unity DOTS ermöglicht werden. Konkreter ist dabei vorgesehen, die Dokumentation des Systems, das nötige Umdenken bezüglich des Programmieransatzes und die Benutzerfreundlichkeit der Umsetzung im Unity-Editor in den Fokus zu stellen.

4.4.3 Nutzungserlebnis der Anwendung

Zuletzt ist eine kurz gefasste, subjektive Einschätzung zum von der erstellten Anwendung in ihrer aktuellen Form erzielten Nutzungserlebnis zu treffen. Dabei soll konkreter betrachtet werden, wie angenehm sich die Nutzung des Programms gestaltet und wie gut es bereits für den zuvor definierten Anwendungszweck einsetzbar ist.

4.5 Aussagegehalt der erzielten Ergebnisse

Im Voraus der Projektdurchführung sind noch einige Hinweise zu den letztlich erzielten Ergebnissen und der Aussagekraft der aus ihnen gezogenen Schlüsse zu geben. So ist zunächst festzuhalten, dass sich die Resultate auf den konkreten Anwendungsfall beziehen, der in dieser Arbeit erläutert wurde und möglichst stark auf vermeintliche Vorteile der Entwicklung mit Unity DOTS fokussiert sein soll. Da verschiedene Anwendungen bedingt durch ihre unterschiedlichen Inhalte stark unterschiedliche Leistungsanforderungen haben, kann die Größe, beziehungsweise das generelle Vorhandensein von Performanzgewinnen durch DOTS stark variieren.

Die im Rahmen des Projekts umzusetzende Anwendung ist zudem, wie zuvor beschrieben, aufgrund des begrenzten Entwicklungsrahmens, zunächst nicht als vollständiges Produkt anzusehen. Viele fehlende oder nur grundlegend umgesetzte Bestandteile des Programms führen so zu einer insgesamt verringerten Komplexität. Aufgrund dessen ist davon auszugehen, dass der Umfang beobachteter Performanzunterschiede entsprechend geringer ausfallen wird, als es in einer umfangreicher umgesetzten Version der Anwendung der Fall wäre.

Daneben ist auch rein für den behandelten Anwendungsfall das mögliche Auftreten einiger Varianzen zu beachten. So hat vor allem das zur Ausführung der Anwendung genutzte PC-System einen großen Einfluss auf durch Technologien wie DOTS erbrachte Leistungsvorteile. Dabei können selbst bei baugleichen Rechnern standardmäßig Abweichungen auftreten; etwa durch die stets vorhandenen Leistungsunterschiede zwischen Prozessoren des gleichen Typs sowie unterschiedlichen zusätzlich zur Anwendung laufenden Hintergrundprozessen der Systeme. Um den Effekten dieser verschiedenen Abweichungen in gewissem Maße entgegenzuwirken, werden für die Datenerfassung im Rahmen dieses Projekts mehrere PCs unterschiedlicher Leistungsklassen genutzt. Eine Repräsentation des vollen Spektrums unterschiedlicher Rechner ist dadurch nicht möglich; ein Vergleich der relativen Performanz der unterschiedlichen Implementierungen auf dem jeweils gleichen System sollte aber dennoch aussagekräftig sein.

Ein weiterer Varianzfaktor findet sich im Programmcode selbst. Dieser kann sehr frei geschrieben werden, wodurch verschiedene Codeabschnitte mit gleicher Funktion stark unterschiedliche

Abläufe aufweisen können. So können sich die gewählten Algorithmen und der generelle Aufbau des Programmcodes stark auf die vom Programm erzielte Performanz auswirken. Gerade im Rahmen dieses zeitlich und personell sehr eingeschränkten Projekts ist dabei eine möglichst perfekte Optimierung des Codes nur bedingt möglich. Die dafür jeweils nötige Expertise im Umgang mit dem Entwicklungssystem ist im Fall von Unity DOTS aufgrund von dessen Neuartigkeit zudem schwerlich aufzubringen. Es ist also damit zu rechnen, dass der erzielte Leistungsunterschied mit und ohne Nutzung von DOTS nicht gänzlich repräsentativ für eine optimale Implementierung ist.

Für das konkret im Rahmen dieser Arbeit entwickelte Programm sollte zudem herausgestellt werden, dass die beiden zu vergleichenden Versionen des für die Simulation zuständigen Programmteils auf völlig unterschiedlichen Unity-Systemen basieren. Der Ablauf gleich aufgebauter Simulationen wird zwischen beiden Versionen daher kaum identisch sein und entsprechende Varianz in die betrachteten Leistungsdaten einbringen; so beispielsweise durch die Unterschiede in den jeweils verwendeten Zufallszahlengeneratoren oder den verwendeten Physiksystemen. Dennoch sollten die Daten insgesamt ein im Durchschnitt repräsentatives Leistungsabbild des jeweiligen Entwicklungssystems hervorbringen.

Der begrenzte mögliche Umfang der im Rahmen dieses Projekts machbaren Untersuchungen, gerade in Bezug auf die verfügbare Zeit und den Zugang zu Testsystemen, lässt eine Reduzierung des Einflusses dieser Umstände nur in bedingtem Maß zu. Die durch die Varianzen möglicherweise hervorgerufenen Auswirkungen auf die letztlich festgestellten Resultate sollten entsprechend beachtet werden.

Die Ermittlung einer zum Treffen fundierter Aussagen ausreichend großen Datenmenge ist vor dem Hintergrund dieser Limitierungen, gerade bei einer möglichst sorgfältigen Arbeitsweise, kaum erzielbar. Entsprechend wird bei der Ergebnisauswertung davon abgesehen, eine detaillierte statistische Untersuchung durchzuführen. Vielmehr sollen die Resultate als Orientierungspunkt dienen und eine potentielle Grundlage für vertiefende Untersuchungen darstellen. Die Ergebnisse sollen dabei eher als grundlegendes Werkzeug zur Entscheidungsfindung in Bezug auf die Verwendung von Unity DOTS dienen, statt als feste Größen angesehen zu werden. Es empfiehlt sich zudem, stets eine eigene Auswertung vorzunehmen, um die Sinnhaftigkeit der Nutzung von DOTS für ein konkretes Projekt einschätzen zu können.

5 Projektdurchführung

Bevor mit der Umsetzung des Projekts begonnen werden konnte, mussten zunächst einige vorbereitende Entscheidungen getroffen werden. In den folgenden Abschnitten werden diese Vorbereitungsschritte zunächst erläutert, bevor die Umsetzung der verschiedenen Komponenten der Anwendung in ihrer Grundform näher dokumentiert wird.

5.1 Vorbereitung

Zunächst war die für das Projekt zu nutzende Unity-Version festzulegen, wobei sich die Verwendung einer LTS-Version empfohlen hätte. Die neuste dieser LTS-Versionen stellte dabei Unity 2021.3.18f1 dar; für die Nutzung der aktuellen Version von Unity DOTS ist jedoch als Mindestanforderung die Nutzung von Unity 2022.2.0b8 oder einer neueren Version vorgegeben. [96] Entsprechend musste das Projekt in einer regulären Version von Unity angelegt werden; konkret in Version 2022.2.5f1. Diese stellte zum Zeitpunkt des Projektstarts die aktuellste als abgeschlossen veröffentlichte Version von Unity dar. DOTS stand mit der neusten veröffentlichten Version 1.0.0-pre.15 zwar kurz vor der Fertigstellung, befand sich jedoch noch in einer Testphase. Wahrscheinlich würde es sich daher positiv auswirken, eine möglichst aktuelle Editor-Version zu nutzen, um eine hohe Fehlerfreiheit und damit für DOTS bestmögliche Funktion zu gewährleisten.

Entsprechend wurde ein neues Unity-Projekt basierend auf Version 2022.2.5f1 erstellt. Der Paketmanager, über den zusätzliche Editor-Module installiert werden können, kam daraufhin zum Einsatz, um alle für DOTS 1.0.0-pre.15 relevanten Komponenten zum Projekt hinzuzufügen.

Zur Umsetzung des Projekts sollte zudem ein Versionskontrollsystem zum Einsatz kommen, wobei die Entscheidung auf das stark verbreitete „Git“ fiel. Bei solchen Systemen handelt es sich um Software, welche es ermöglicht, Projektdateien zentral zu speichern und ihre Weiterentwicklung zu dokumentieren. Wird ein Versionskontrollsystems genutzt können und sollten am Projekt vorgenommene Änderungen in regelmäßigen Abständen an dieses gemeldet werden. Daraufhin werden die Änderungen an der zentralen Projektversion übernommen und ihre vorherige Fassung wird archiviert. Dadurch können alle Änderungen chronologisch sowie inhaltlich nachvollzogen und frühere Versionen des Projekts wiederhergestellt werden. So wird es etwa möglich, durch Änderungen auftretende Fehler besser nachvollziehen und beheben zu können. Auch können Kompatibilitätstest, bei Nutzung von Unity beispielsweise zur Verwendung neuerer Editor-Versionen, durchgeführt und bei Problemen leicht wieder rückgängig gemacht werden. [97] [98]

Ein weiterer Vorteil entsprechender Systeme findet sich beispielsweise bei kollaborativer Arbeit. So können mit Versionskontrollsystemen mehrere Personen gleichzeitigt ein Projekt bearbeiten. Entstehen dabei Konflikte, etwa durch unterschiedliche Änderungen, die simultan an gleichen Daten vorgenommen werden, weist das System auf diese hin und assistiert bei ihrer Behebung. [97] [98] Diesen und einigen weiteren, hier nicht näher erläuterten Vorzügen der Nutzung eines Versionskontrollsystems, kamen im Rahmen dieses Projekts dabei keine große Bedeutung zu.

5.2 Kreaturenerstellung

Den ersten tatsächlich umgesetzten Bereich des Programms stellte der Editor zum Erstellen von Kreaturen dar, welcher in Abbildung 2 zu sehen ist. Dieser wird gänzlich durch eine GUI repräsentiert, welche in drei Hauptbereiche aufgeteilt ist. Besagte Bereiche sind farblich sowie durch Weißraum räumlich getrennt und so für Nutzende voneinander abgehoben wahrnehmbar.

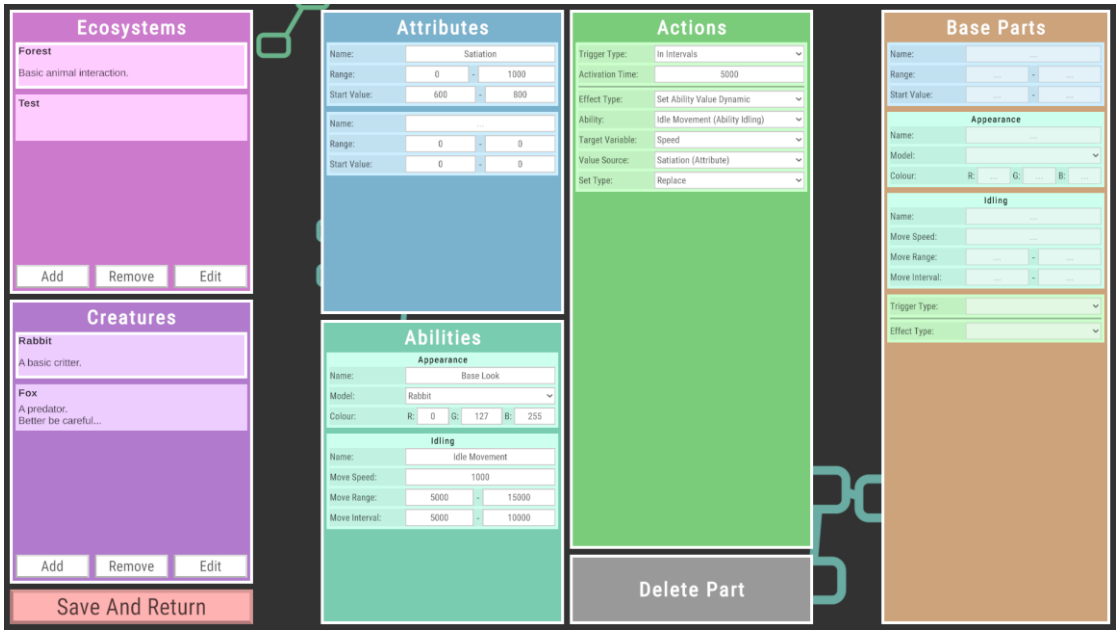

Abbildung 2: Der Kreatureneditor

Auf der linken Seite befindet sich eine Übersicht der bestehenden Ökosysteme sowie Kreaturen. Im oberen Teil dieser Übersicht ist dabei eine scrollbare Liste aller bestehenden Ökosysteme hinterlegt, welche als rechteckige Blöcke dargestellt werden. Über die Knöpfe „Add“, „Edit“ und „Remove“ können neue Ökosysteme erstellt sowie Bestehende bearbeitet oder gelöscht werden. Beim Erstellen sowie Bearbeiten wird, wie in Abbildung 3 sichtbar, ein Dialog geöffnet, in welchem über Eingabefelder Name und Beschreibung des jeweiligen Ökosystems fest-

gelegt werden können. Diese Eigenschaften werden im Block, welcher das Ökosystem repräsentiert, angezeigt, um eine Unterscheidung zu ermöglichen. Unter dieser ersten Liste befindet sich eine Zweite, welche analog aufgebaut ist und die zu einem Ökosystem gehörenden Kreaturen darstellt. Funktional gleicht sie dabei der Ökosystem-Liste.

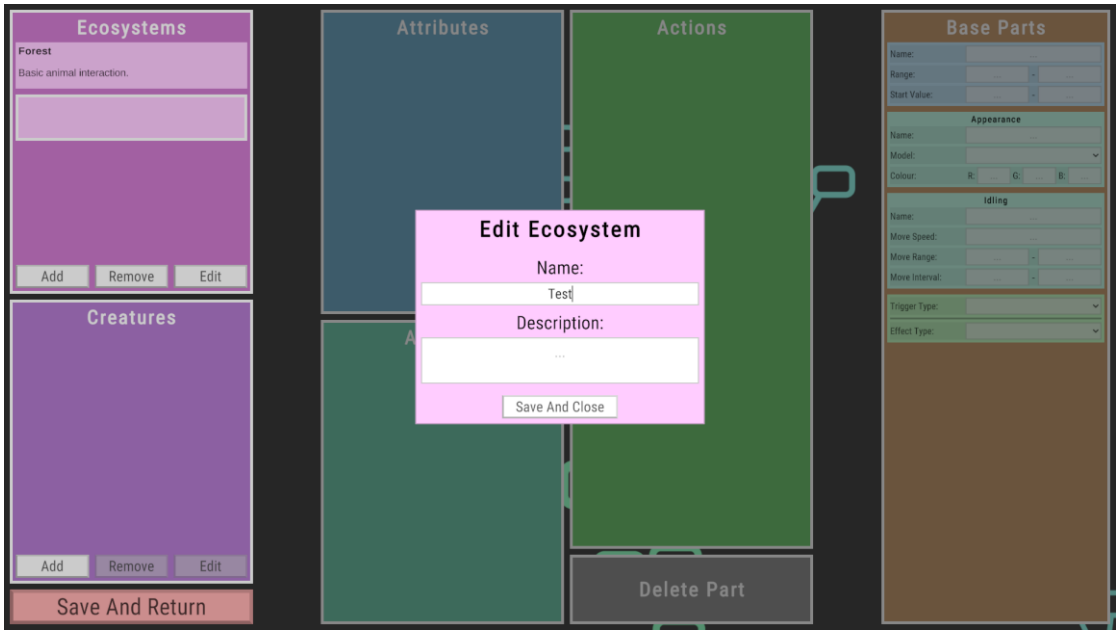

Abbildung 3: Dialog zum Erstellen oder Bearbeiten eines Ökosystems

Die Einträge beider Listen können durch einen Klick ausgewählt werden, wodurch das jeweilige Element hervorgehoben und die Knöpfe für die Lösch- sowie Bearbeitungsfunktion aktiviert werden. Die Kreaturenliste zeigt dabei stets nur die Kreaturen an, die zu dem aktuell ausgewählten Ökosystem gehören. Wird eine Kreatur ausgewählt, kann diese in einem anderen GUI-Bereich bearbeitet werden, wie im später folgenden Text näher erläutert. Unter den beiden Auflistungen findet sich zudem ein Knopf zum Schließen des Editors.

Der rechte Bereich des Editors beinhaltet eine weitere Liste, welche alle für die Kreaturenerstellung verfügbaren Bausteine beinhaltet. Besagte Bausteine werden dabei analog zu den Ökosystemen sowie Kreaturen als Blöcke dargestellt, in welchen sich verschiedene GUI-Felder befinden. Über diese können die Eigenschaften der Bausteine, wie zuvor beschrieben, angepasst werden – diese Möglichkeit ist jedoch zunächst deaktiviert, solange sich die Bausteine im rechten Bereich befinden. Relevante Interaktionsmöglichkeiten bieten sich hier erst in Zusammenwirkung mit dem dritten Bereich.

Der mittig platzierte Teil der GUI dient zum tatsächlichen Bearbeiten ausgewählter Kreaturen, wobei er in verschiedene Zonen mit unterschiedlicher Funktionalität unterteilt ist. Zunächst finden sich dabei drei Zonen für die verschiedenen Baustein-Kategorien, also Attribute, Fähigkeiten und Aktionen. Des Weiteren gibt es eine Zone zum Löschen von Bausteinen.

Die Interaktion mit besagten Zonen erfolgt über ein Drag-and-Drop-System, wie in Abbildung 4 zu sehen ist. Dabei können Bausteine angeklickt und durch Bewegung der Maus verschoben werden. Beim Loslassen der Bausteine über einer validen Zone werden sie daraufhin in dieser platziert. Jeder Baustein ist dabei einer der zuvor erwähnten Kategorien zugeordnet und kann nur in der jeweils zugehörigen Zone platziert werden. Um dies besser zu visualisieren, sind die Zonen farblich leicht unterschiedlich gestaltet, wobei die jeweils zugehören Bausteine analog farbkodiert sind. Dadurch soll Nutzenden die jeweilige Zusammengehörigkeit signalisiert und eine richtige Zuordnung vereinfacht werden. Für einige Bausteine ist das Hinzufügen zudem insofern begrenzt, dass bei einer Kreatur nur jeweils einer verwendet werden kann. So sollen Funktionskonflikte vermieden werden; etwa bei konstant aktiven Fähigkeiten, welche einander überlagern würden. Ein Beispiel dafür ist das Aussehen der Kreatur, welches nur einmal gesetzt werden kann.

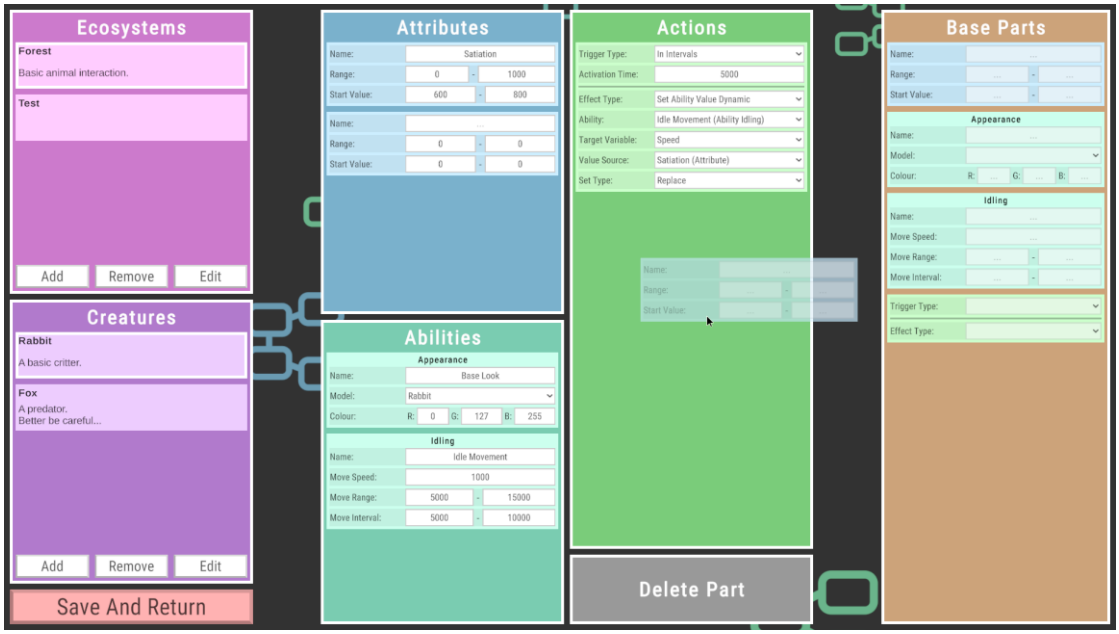

Abbildung 4: Verschieben von Bausteinen per Drag-and-Drop

So korrekt platzierte Bausteine werden in der jeweiligen Zone entsprechend ihrer Position in einer scrollbaren Liste eingeordnet; bei einer inkorrekten Platzierung kehren sie zu ihrem Ursprungsort zurück. Wurde ein Baustein in einer Liste der Zonen platziert, wird er der aktuell ausgewählten Kreatur hinzugefügt. Auch wird die Interaktion mit den von ihm beinhalteten GUI-Felder aktiviert, sodass seine Eigenschaften angepasst werden können. Werden Bausteine in der Lösch-Zone platziert, werden sie hingegen stets, unabhängig von ihrer Kategorie, entfernt. Aus der Baustein-Übersicht herausgezogene Blöcke werden bei korrekter Platzierung nicht verschoben, sondern lediglich kopiert; die Übersicht dient also als unbegrenzte Quelle für weitere Bausteine.

Trotz der, wie bereits erwähnt, zunächst fehlenden Dokumentation zur Anwendung, sollten Nutzende bei ihrer Verwendung bereits grundlegend geführt werden, um eine intuitivere Bedienung zu ermöglichen. Zu diesem Zweck wurde für die GUI ein schlichtes Fokussystem integriert, durch das GUI-Bereiche, mit denen im Moment nicht sinnvoll interagiert werden kann, deaktiviert und visuell verdunkelt werden, um dies für Nutzende zu verdeutlichen. Beispielhaft ist dies in Abbildung 5 zu sehen.

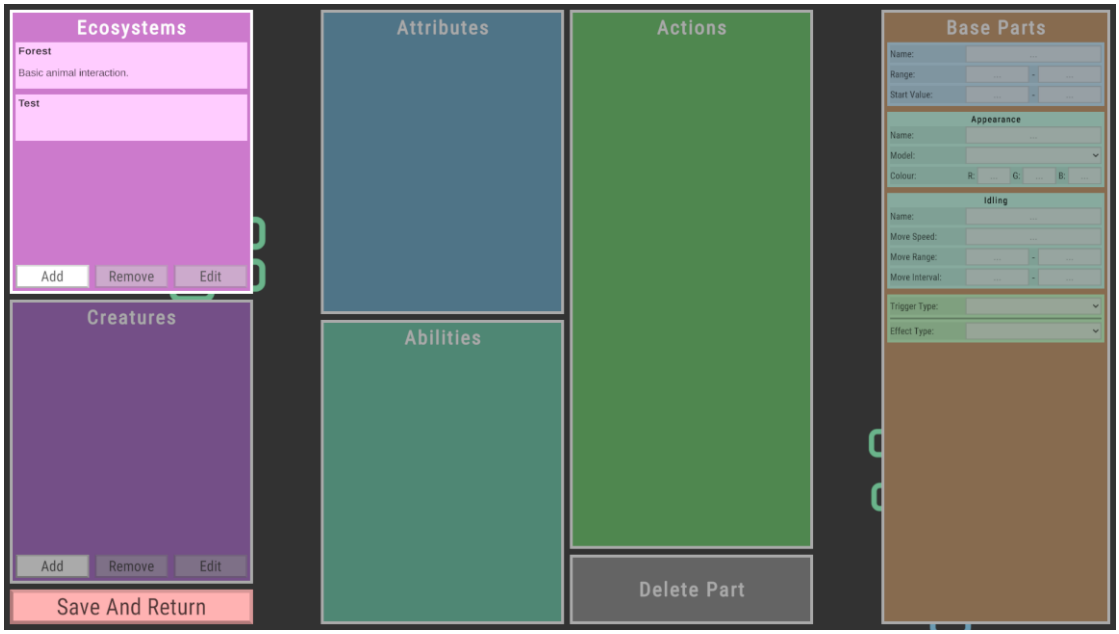

Abbildung 5: Fokussystem im Kreatureneditor

Beim Erstellen von Kreaturen sind dabei zunächst nur die Liste der Ökosysteme sowie der Knopf zum Verlassen des Editors aktiviert. Durch die Erstellung und Auswahl eines Ökosystems wird zusätzlich die Kreaturen-Liste aktiviert. Wird wiederum eine Kreatur erstellt und ausgewählt, werden der verbleibende Bereich zum Bearbeiten der Kreatur sowie die Baustein-Liste aktiviert. Wird die Kreatur respektive das Ökosystem wieder abgewählt, tritt der gegenteilige Effekt ein, indem die zugehörigen anderen GUI-Bereiche wieder deaktiviert werden.

Das Fokussystem kommt zudem beim Öffnen von Dialogen zum Einsatz, wobei es die Interaktion mit der restlichen Szene deaktiviert und Eingaben nur in dem jeweiligen Dialog erlaubt. Analog zur sonstigen Anwendung des Systems werden dabei alle anderen Szenenelemente visuell verdunkelt.

Zentral für den Kreatureneditor ist neben diesen GUI-Systemen eine Speicherfunktion für die erstellten Modellkomponenten. Diese speichert alle Daten persistent ab, welche die angelegten Ökosystemen und Kreaturen sowie die den Kreaturen angefügten Bausteine definieren. Dabei

werden alle Instanzen dieser drei Komponentenarten separat und möglichst unabhängig voneinander gesichert, um ein möglichst granulares Laden respektive Speichern jeweils kleiner Datenmengen zu ermöglichen.

Für jede in der GUI erstellte Komponenteninstanz steht dabei im Hintergrund ein Datenobjekt, welches die jeweiligen Inhalte aller bearbeitbaren GUI-Felder beinhaltet. Zudem verfügt jedes dieser Datenobjekte über eine Identifikationsnummer (ID) sowie Auflistungen mit ihr verbundener Instanzen. So ist für jedes Ökosystem eine Auflistung mit den IDs der ihm untergeordneten Kreaturen hinterlegt. Jede Kreatur beinhaltet wiederum Listen mit den IDs der ihr angefügten Bausteine.

Zum Speichern dieser Datenobjekte wird eine Unity-eigene Funktion genutzt, welche aus ihren Daten Text in „JavaScript Object Notation“ (JSON) generiert. Dabei handelt es sich um ein kompaktes, für Menschen leicht verständliches, Datenformat. Mithilfe einer weiteren von Unity bereitgestellten Funktion werden die so erstellten Textabschnitte daraufhin automatisiert auf dem Rechner, auf welchem die Anwendung ausgeführt wird, abgespeichert. Das Laden der Daten erfolgt daraufhin über den gleichen Prozess in umgekehrter Reihenfolge, wobei basierend auf den Daten GUI-Objekte angelegt und im Editor platziert werden.

Um das Speichern und Laden effizient abzuwickeln, erfolgen beide Vorgänge nur dann, wenn die Daten der jeweiligen Komponente tatsächlich benötigt werden. Informationen zu Kreaturen und Bausteinen werden so nur geladen und als GUI-Elemente präsentiert, wenn das zugehörige Ökosystem beziehungsweise die zugehörige Kreatur ausgewählt wird. Wird diese jeweils übergeordnete Komponente wieder abgewählt, werden die zugehörigen Kreaturen respektive Bausteine aus den zugehörigen Listen entfernt und ihre Daten gespeichert. Analog werden die zu einzelne Komponenten gespeicherten Daten stets aktualisiert, wenn Änderungen an ihrer GUI-Repräsentation vorgenommen werden. Auch werden beim Verlassen des Editors alle zu diesem Zeitpunkt geladenen Ökosysteme, Kreaturen und Bausteine gespeichert.

5.3 Modellerstellung

Der zum Anlegen von Modellen vorgesehene Editor, wie er in Abbildung 6 zu sehen ist, wurde als nächstes umgesetzt, wobei er sich stark an dem für die Kreaturenerstellung genutzten orientiert. So wird auch er vollständig durch eine GUI dargestellt, welche in drei grundlegende Bereiche unterteilt ist. Diese werden dabei analog durch ihre farbliche Gestaltung und räumliche Trennung voneinander abgehoben.

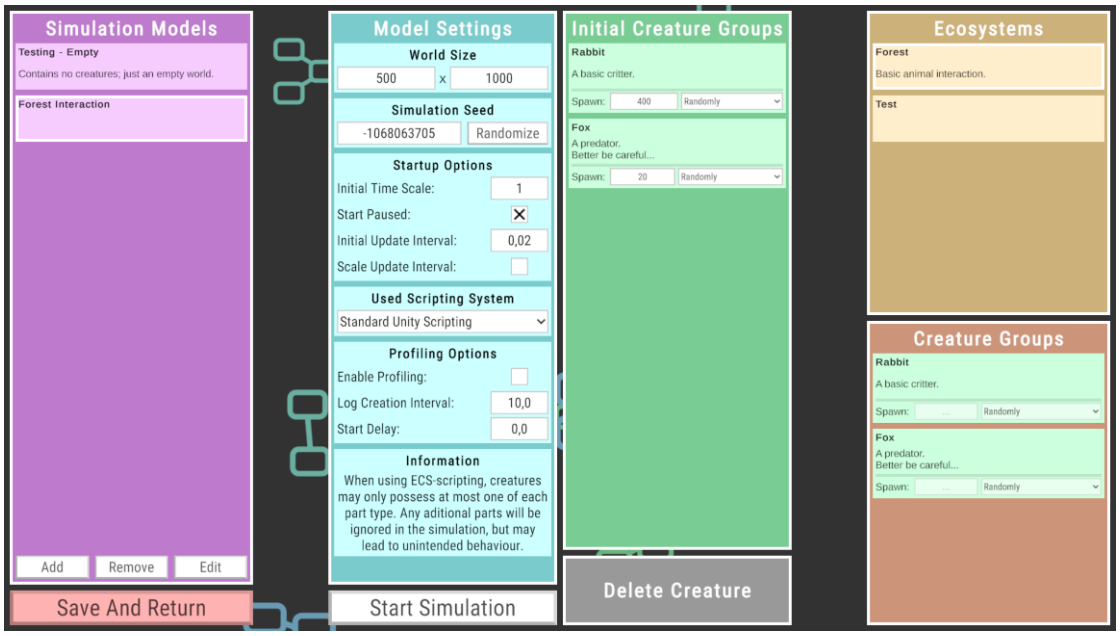

Abbildung 6: Der Modelleditor

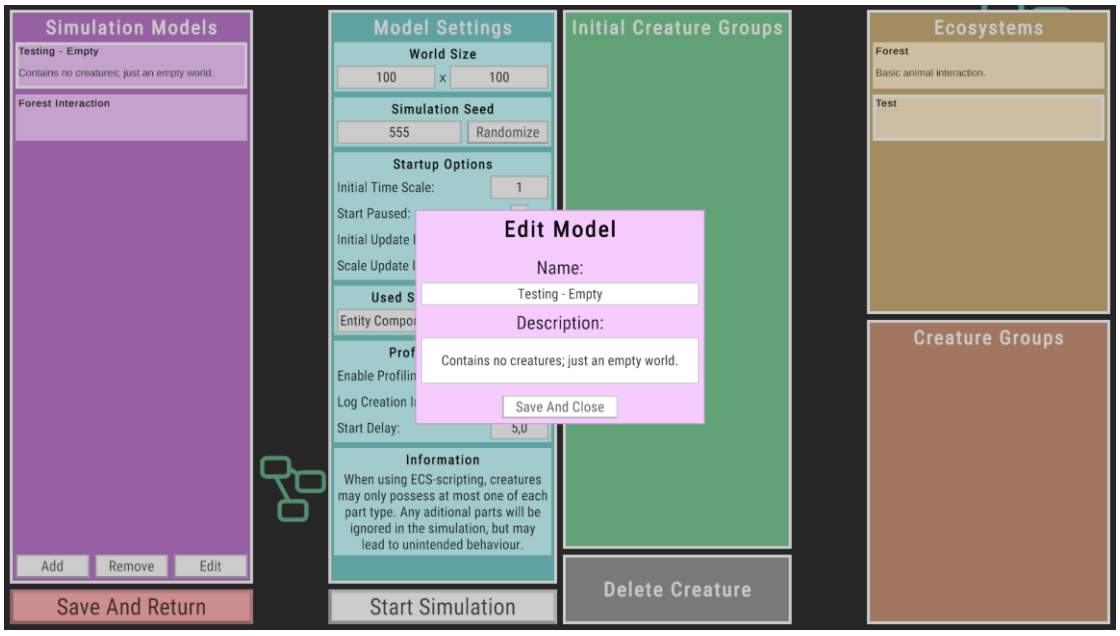

Abbildung 7: Dialog zum Erstellen oder Bearbeiten eines Modells

Im linken Bereich findet sich zunächst eine Übersicht aller erstellten Modelle in Form einer scrollbaren Liste. Diese ist funktional identisch zu den für die Ökosysteme sowie Kreaturen genutzten Übersichten und ermöglicht entsprechend das dialoggestützte Erstellen und Bearbeiten sowie das Löschen von Modellen, wie in Abbildung 7 zu sehen. Diese werden als Blöcke dargestellt, welche Name und Beschreibung des jeweiligen Modells beinhalten und per Klick ausgewählt werden können. Unter dieser Auflistung findet sich zudem ein Knopf zum Schließen des Editors.

Auf der rechten Seite des Editors finden sich zwei Übersichten, welche jeweils die zuvor erstellten Ökosysteme und die in diese eingeordneten Kreaturen beinhalten. Die Darstellung dieser erfolgt dabei wie im Kreatureneditor als schlichte Blöcke innerhalb scrollbarer Listen. Anders als in diesem Editor, fehlt hier jedoch die Möglichkeit zum Anpassen, Erstellen oder Löschen. Ökosysteme können ausgewählt werden, um die ihnen zugeordneten Kreaturen zu laden; diese sind jedoch selbst nicht auswählbar. Zudem werden die Kreaturen hier als Gruppen bezeichnet und erweitert dargestellt, wobei sie GUI-Felder zur Einstellung verschiedener, für die Simulation relevanter, Attribute beinhalten. Die Bearbeitung dieser Felder ist zunächst deaktiviert und kommt erst im nächsten Bereich zum Tragen.

Im mittleren Teil des Editors findet sich eine Übersicht zur Anpassung des aktuell ausgewählten Modells. Sie beinhaltet zunächst eine Auflistung allgemeiner Optionen, bei denen über GUI-Felder Einstellungen für die Simulationsdurchführung an dem Modell angepasst werden können. Konkret kann so ein Startwert für die Zufallszahlenberechnung angegeben und die Größe der simulierten Umgebung angepasst werden. Es kann ebenfalls festgelegt werden, mit welcher Geschwindigkeit die Simulation zu Beginn ablaufen und ob sie zunächst pausiert starten soll. Zudem finden sich hier Einstellungen für das Aktualisierungsintervall der Simulation. Das Konzept dieses Intervalls sowie die Auswirkungen der es betreffenden Einstellungen werden an späterer Stelle erläutert, wenn die tatsächliche Simulationsdurchführung beschrieben wird.

Abseits dieser Optionen lässt sich das für die Simulation genutzte Skriptsystem festlegen und einstellen, ob und wie während der Simulation Leistungsdaten erfasst werden sollen. Dabei ist auswählbar, in welchen Zeitabständen erfasste Daten in Dateien gespeichert werden sollen und, wie lange nach Simulationsstart die Datenerfassung beginnen soll. Zuletzt beinhaltet die Übersicht ein Informations-Panel, welches Hinweise zur Funktionalität der Simulationsdurchführung gibt.

Neben diesen allgemeinen Modelloptionen finden sich im mittleren Bereich des Editors zudem zwei Zonen zur Nutzung mit den zuvor erwähnten Gruppen. Die erste stellt dabei eine Übersicht der Startgruppen des Modells dar, während die zweite zum Löschen von Gruppen dient. Analog zum Kreaturen-Editor können die Gruppen hier per Drag-and-Drop-System über die Zonen gezogen werden, um in diesen platziert respektive aus ihnen entfernt zu werden. Um dieses Konzept zu verdeutlichen und intuitiv erschließbar zu machen, sind auch hier die Zone für Startgruppen sowie die Gruppen an sich farblich aneinander angepasst.

Die aus der Gruppenübersicht gezogenen Blöcke werden stets kopiert und können entsprechend beliebig oft in Modellen verwendet werden. Sind sie in der Startgruppen-Übersicht platziert, können ihre GUI-Felder bearbeitet werden. Konkret kann so die Menge an Kreaturen in der Gruppe bestimmt und festgelegt werden, wie diese Kreaturen in der simulierten Welt verteilt

werden sollen. Der mittlere Bereich des Modelleditors beinhaltet neben diesen Elementen zudem einen Knopf, mit welchem am ausgewählten Modell eine Simulation mit den aktuell gewählten Einstellungen gestartet werden kann.

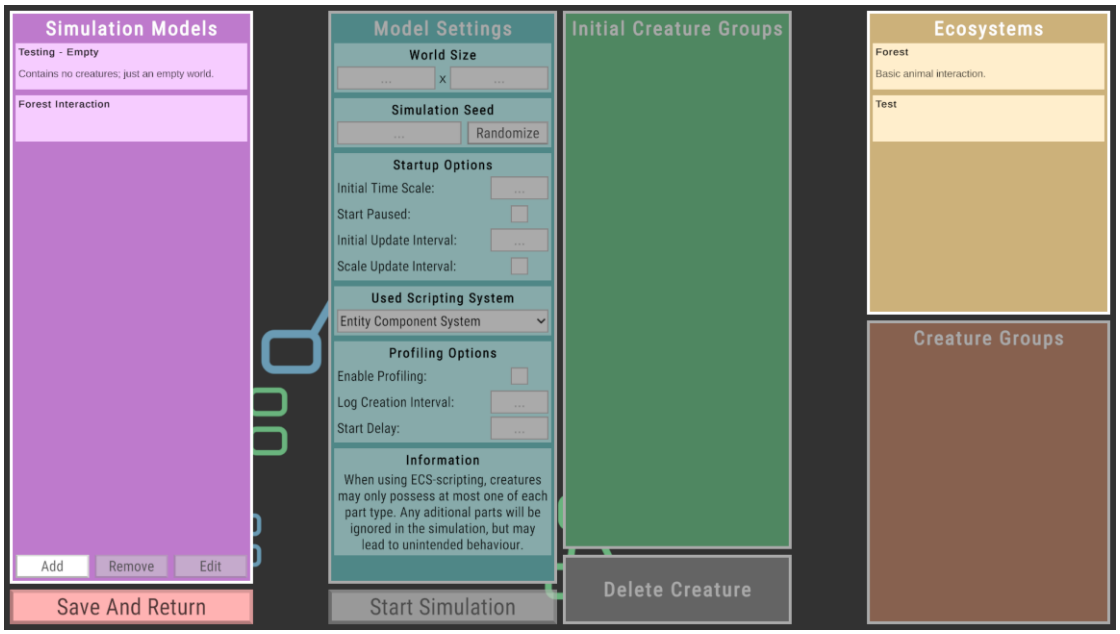

Abbildung 8: Fokussystem im Modelleditor

Das im vorigen Abschnitt erläuterte Fokussystem kommt auch in diesem Editor zum Einsatz, wie in Abbildung 8 sichtbar. So ist eine Interaktion zunächst nur mit der Modell-Liste, der Ökosystem-Liste sowie dem Knopf zum Schließen des Editors möglich. Durch Auswahl eines Modells wird zusätzlich der Bereich zur Modellbearbeitung aktiviert, während die Auswahl eines Ökosystems die Interaktion mit der Gruppenübersicht möglich macht. Die Abwahl eines dieser Elemente führt wiederrum zur erneuten Deaktivierung des jeweils zugehörigen Bereichs. Der Dialog zur Modellerstellung nutzt das System ebenfalls, analog zu den bei der Kreaturenerstellung genutzten Dialogen.

Für die Modelle sowie Startgruppen kommt erneut das zuvor erläuterte Speichersystem zum Einsatz, um diese und ihre Eigenschaften in Form von JSON-Text persistent festzuhalten. Für jedes Modell werden dabei Name und Beschreibung sowie die Werte der allgemeinen Optionen in einem Datenobjekt gespeichert; für die Startgruppen werden jeweils die für Menge und Verteilung gewählten Werte sowie die ID der zugehörigen Kreatur festgehalten. Jedes Modell verfügt zusätzlich über eine Liste der IDs aller ihm zugeordneten Startgruppen.

Auch hier werden Objekte der Effizienz halber nur geladen oder gespeichert, wenn sie tatsächlich verwendet werden. Die jeweiligen Gruppen und Startgruppen werden somit nur geladen, wenn das ihnen jeweils übergeordnete Ökosystem beziehungsweise Modell ausgewählt wird. Gespeichert werden diese Daten, wenn die ihnen übergeordnete Komponente abgewählt wird.

Geladene Modelle sowie Gruppen werden zudem ebenfalls gespeichert, wenn Änderungen an ihren GUI-Repräsentationen vorgenommen werden und wenn der Editor verlassen wird.

5.4 Simulationsdurchführung

Der für die Simulationsdurchführung umgesetzte Bereich der Anwendung beinhaltet zunächst einige Basiskomponenten. Diese wurden unabhängig vom für eine konkrete Simulation genutzten Skriptsystem umgesetzt, da sie sich bei separater Umsetzung pro System aufgrund ihrer geringen Komplexität kaum aussagekräftig auf das jeweils erzielte Leistungsresultat ausgewirkt hätten.

So wurde zunächst eine Steuerung für die die Simulation darstellende Kamera umgesetzt. Besagte Kamera kann im für die Simulationen genutzten dreidimensionalen Raum frei bewegt werden, wobei ihre Bewegung auf einen Bereich entsprechend der im Modell festgelegten Weltgröße begrenzt ist. Über die Tasten „W“, „A“, „S“ sowie „D“ oder die Pfeiltasten kann die Kamera dabei vor und zurück sowie nach links und rechts bewegt werden. Durch Halten der Leertaste kann sie aufsteigen und über die Shift- oder Steuerung-Taste wieder abgesenkt werden. Bei Klicken und Halten der rechten Maustaste oder des Mausrads kann durch gleichzeitige Bewegung der Maus zudem die Ausrichtung der Kamera angepasst werden. Die Kamerageschwindigkeit kann des Weiteren durch Bewegen des Mausrads erhöht respektive gesenkt werden, wobei die „R“- oder die Tabulator-Taste die Geschwindigkeit auf ihren Standardwert zurücksetzen.

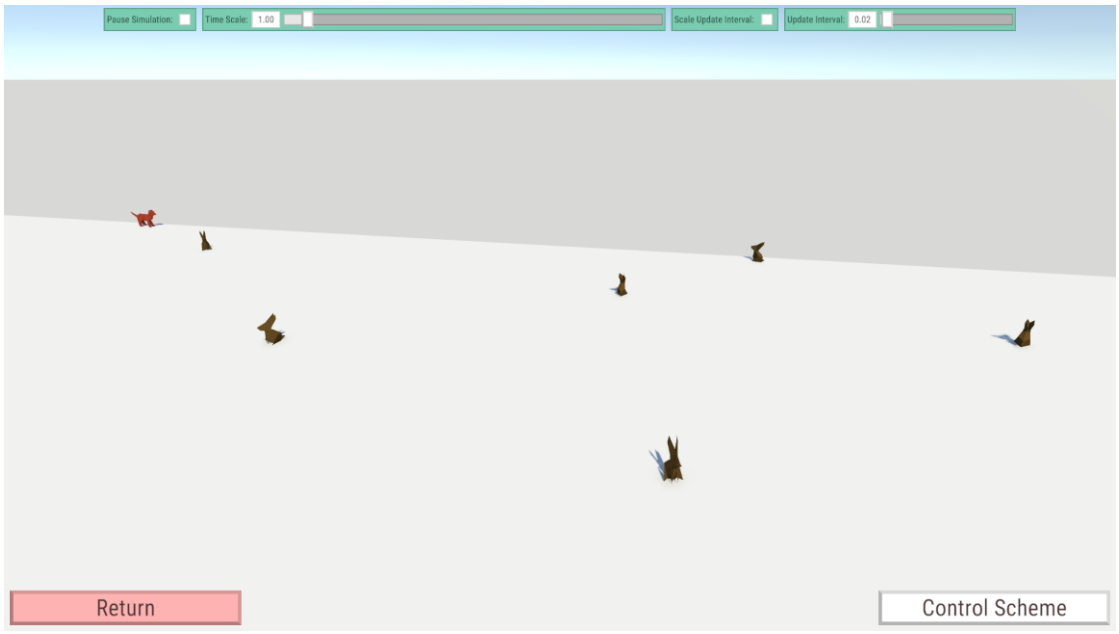

Abbildung 9: Kameraansicht und GUI bei der Durchführung einer Simulation

Zudem wurde auch hier eine grundlegende GUI umgesetzt, wie in Abbildung 9 zu sehen ist. Diese beinhaltet eine am oberen Bildschirmrand befindliche Leiste, über welche Simulationsparameter angepasst werden können. So kann über einen Schalter die ablaufende Simulation pausiert respektive fortgesetzt werden. Zudem ermöglicht ein Panel die begrenzte Anpassung der Simulationsgeschwindigkeit. Dies geschieht entweder über einen Schieberegler oder durch Eintragen des Multiplikators für die Geschwindigkeit in ein separates Textfeld. Analog zu dieser Geschwindigkeitseinstellung lässt sich zudem das zuvor bereits erwähnte Aktualisierungsintervall der Simulation wahlweise über ein Textfeld oder über einen Schieberegler anpassen.

In Unity müssen alle in einer Szene existierenden Objekte regelmäßig aktualisiert werden, beispielsweise um ihre Position über die Zeit hinweg zu verändern oder sie auf Interaktionen reagieren zu lassen. Solche Aktualisierungen können dabei grundlegend auf zwei verschiedene Arten geschehen. So können sie einerseits immer dann vorgenommen werden, wenn die Anwendung ein neues Bild ausgibt, andererseits ist es aber auch möglich, Aktualisierungen in regelmäßigen, von der Bildausgabe unabhängigen, Abständen durchzuführen. Die erste Methode eignet sich dabei besser für Aktionen, welche hohe Aktualität erfordern; so beispielsweise für Objektbewegungen, welche mit jedem generierten Bild fortschreiten sollen, um möglichst flüssig zu erscheinen, oder Interaktionssysteme, welche möglichst zeitnah auf Eingaben regieren sollen. Die zweite Methode eignet sich hingegen besser für einheitlich gleichmäßig auszuführende Aktionen, beispielsweise bei Physikkalkulationen, welche unabhängig von der Bildfrequenz in konstanten Intervallen geschehen sollen.

Für die Durchführung der Simulationen in der hier behandelten Anwendung kommt ebenfalls das zweite System zum Einsatz, damit die Simulationen möglichst beständig ablaufen und in ihrem Ablauf nicht von der an die Leistungsfähigkeit des sie ausführenden Systems gekoppelten Bildfrequenz beeinflusst werden. Die Größe der dabei zugrundeliegenden Zeitintervalle kann über die zuvor benannte Einstellung angepasst werden, um den Simulationsablauf individueller auf die konkrete Simulation zuzuschneiden. Durch Verkleinern des Intervalls wird dabei der Detailgrad der Simulation erhöht, wodurch, gerade bei gleichzeitiger Verringerung der Simulationsgeschwindigkeit, Abläufe deutlich genauer erfassbar werden. Eine Erhöhung des Intervalls erlaubt hingegen eine bessere Darstellung makroskopischer Prozesse bei hoher Performanz, da die Menge an zu berechnenden Einzelzuständen der Simulation stark verringert wird.

Ergänzt wird diese Option durch einen Schalter, welcher die Skalierung des genutzten Intervalls entsprechend der gewählten Simulationsgeschwindigkeit aktiviert respektive deaktiviert. Bei Aktivierung dieser Einstellung wird das Intervall von der Geschwindigkeit entkoppelt; in einer festen Echtzeitspanne geschieht also stets die gleiche Menge an Aktualisierungen. Bei Verlangsamung der Simulation erhöht sich durch diese Option also automatisch der Detailgrad, während

er bei Beschleunigung sinkt. Dafür bleibt die Performanz der Simulation unabhängig der Geschwindigkeit beständiger.

Am unteren Bildschirmrand findet sich rechts ein Knopf zur Einsicht der Tastenbelegung der Kamerasteuerung. Durch Anklicken dieses Knopfes öffnet sich ein Dialog mit den entsprechenden Informationen, wie in Abbildung 10 dargestellt. Dieser nutzt das bereits mehrfach angesprochenes Fokus-System, um Interaktion mit der restlichen Szene zu verhindern. Links am unteren Bildschirmrand findet sich schlussendlich ein weiterer Knopf, mit dem zum Modellerstellungsbereich zurückgekehrt werden kann.

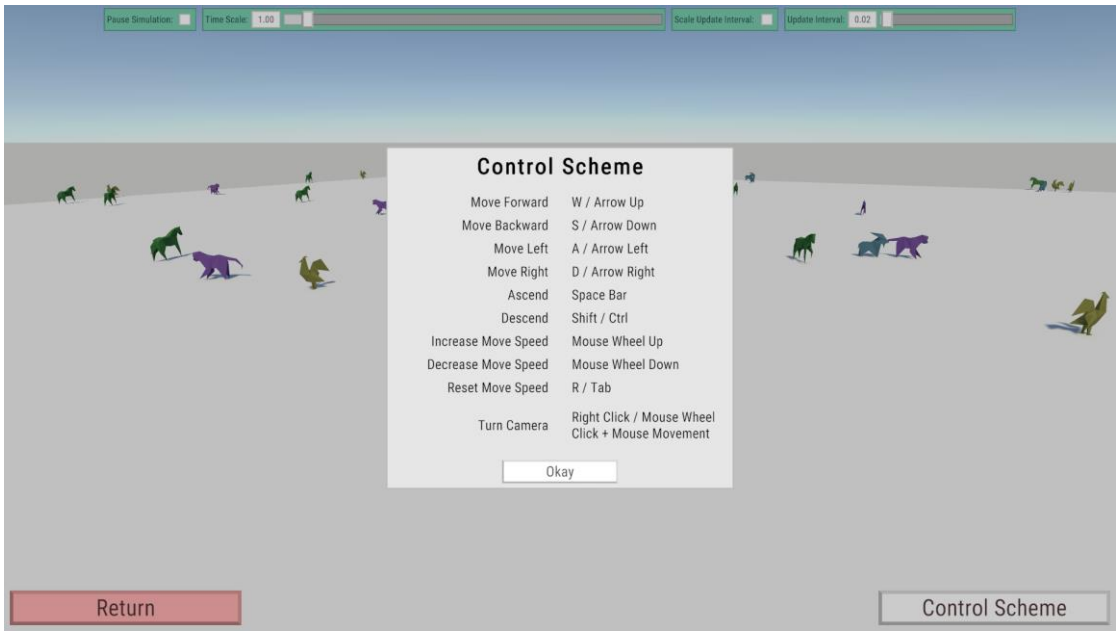

Abbildung 10: Dialog zur Erläuterung der Kamerasteuerung

Die Ausführung der Simulation selbst wird nach dem Laden der Szene direkt eingeleitet. Dabei kommt ein Initiierungsskript zum Einsatz, welches zunächst die gespeicherten Daten des zuvor ausgewählten Models sowie der in ihm genutzten Kreaturen lädt. Je nach im Modell gewähltem Skript-System bereitet diese Initialisierung mithilfe der so geladenen Daten daraufhin unterschiedliche, für die Einleitung der Simulation zuständige, Skripte vor und stößt deren Ausführung an. In den folgenden Abschnitten wird für beide umgesetzte Skript-Systeme die Funktion dieser Skripte sowie der durch sie jeweils initiierte Simulationsprozess näher erläutert.

5.4.1 Simulationsablauf mit Standardsystem

Der durch das Initiierungsskript angestoßene Prozess baut zunächst die für die Simulation genutzte Welt entsprechend der im Modell gesetzten Parameter auf. Dabei werden als Boden agie-

rende Objekte erstellt, welche entsprechend der gewählten Weltgröße skaliert und platziert werden. Zudem werden Kollisionsobjekte angelegt und platziert, welche als Begrenzung der Welt agieren, die von simulierten Kreaturen nicht überwunden werden kann.

Ist dieser Prozess abgeschlossen, werden nachfolgend die im Modell definierten Kreaturen instanziiert. Dabei wird zunächst für jeden Kreaturentyp eine Basiskreatur erstellt. Diese wird zunächst lediglich durch ein vordefiniertes Objekt repräsentiert, welches grundlegend benötigte Unity-Komponenten zur dreidimensionalen Darstellung und Physikinteraktion beinhaltet. Zusätzlich haftet diesem Grundobjekt ein Kernskript an, welches später das Zusammenwirken der Bausteine der jeweiligen Kreatur verwaltet. Daraufhin werden die Bausteine der aktuell zu erstellenden Kreatur abgearbeitet, wobei der Basiskreatur für jeden davon ein dem Bausteintyp zugehöriges Skript hinzugefügt wird. Auf diese als Simulationskomponenten bezeichneten Skripte werden darauffolgend die im Kreatureneditor gesetzten Werte des Bausteins übertragen.

Ist so eine Basiskreatur in ihrer Gesamtheit erstellt wird sie zunächst deaktiviert, bevor alle Basiskreaturen im nächsten Schritt weiter bearbeitet werden. Dabei wird jede Basiskreatur entsprechend der im Modell festgelegten Menge der jeweiligen Kreatur kopiert. Jede der so erstellten Kopien wird daraufhin initialisiert, wodurch jede ihr anhaftende Simulationskomponente einen Einrichtungsprozess durchläuft. Bei diesem werden beispielsweise Zufallswerte mithilfe eines zentralen Zufallszahlengenerators für jede Kreatur individuell gesetzt. Auch werden Referenzen auf die verschiedenen Simulationskomponenten sowie Unity-Komponenten der Kreatur eingerichtet, um spätere Zugriffe auf sie performanter zu gestalten.

Daraufhin werden alle so kopierten und eingerichteten Kreaturen entsprechend der Modelleinstellungen in der Simulationswelt platziert. Ist dies geschehen, werden alle Simulationskomponenten der Kreaturen gestartet, wodurch Abläufe, wie zum Beispiel das Bewegen durch die Welt, eingeleitet werden. Somit beginnt die Simulation, wobei alle weiteren Abläufe von den individuellen Simulationskomponenten der Kreaturen abhängen. Die verschiedenen Simulationskomponenten speichern dabei entweder nur Daten oder beinhalten zusätzliche Funktionen, welche Änderungen an Verhalten, Aussehen und ähnlichen Eigenschaften der Kreaturen durchführen können. Dabei interagieren sie zum Teil, entsprechend der im Kreatureneditor getroffenen Einstellungen, miteinander.

Die konkrete Funktion der Simulationskomponenten lässt sich dabei beispielhaft an der Leerlauf-Komponente erklären. Diese sorgt dafür, dass die ihr zugewiesene Kreatur sich zufällig bewegt. An ihr einstellbare Parameter umfassen ein Intervall für die Weite der Bewegung, einen Wert für die Geschwindigkeit der Kreatur sowie ein Zeitintervall für die Festlegung eines neuen Bewegungsziels. Während die Simulation abläuft, ruft die Simulationskomponente in durch das Zeitintervall bestimmten Abständen eine Funktion auf, welche entsprechend der eingestellten

Bewegungsreichweite ein neues Bewegungsziel festlegt. Daraufhin berechnet die Komponente bei jeder Aktualisierung der Simulation, wie die Kreatur sich entsprechend ihrer Geschwindigkeit bewegen müsste, um sich dem Ziel weiter anzunähern. Diese Bewegungsinteraktion teilt sie daraufhin dem Kernskript der Kreatur mit.

Besagtes Kernskript nimmt bei jeder Aktualisierung zentral verschiedene Intentionen der Simulationskomponenten entgegen und führt diese nach Möglichkeit aus. So kombiniert es beispielsweise verschiedene Bewegungsintentionen, wie die der Leerlauf-Komponente, und wirkt eine entsprechende physische Kraft auf die Kreatur aus, um die Bewegung umzusetzen. Durch diese und ähnliche Abläufe agiert jede Kreatur individuell entsprechend der ihr angefügten Bausteine.

5.4.2 Simulationsablauf mithilfe von Unity DOTS

Wird ECS als Skript-System genutzt, gleicht die Einleitung der Simulationsprozesse stark der zuvor dargestellten, wobei sich Abweichungen hauptsächlich durch die von DOTS geforderten Datenstrukturen sowie Funktionsaufteilung ergeben. Zunächst werden dabei, analog zu den vorherigen Ausführungen, Boden- sowie Kollisionsobjekte erstellt, wobei nun Entitäten statt Unity-Objekten zum Einsatz kommen.

Die Kreaturenerstellung läuft ebenfalls grundlegend wie zuvor erläutert ab, indem für jeden Kreaturentyp eine Basiskreatur als Entität erstellt wird. Diese enthält ebenfalls ein Kernskript und wird durch Simulationskomponenten erweitert, welche hier jedoch in ihrer Form als ECS-Komponenten nur Daten speichern und keinerlei Logik beinhalten. Auf diese datenbasierten Simulationskomponenten werden darauffolgend die im Kreatureneditor getroffenen Einstellungen übertragen.

Nach Erstellung aller Basiskreaturen werden diese, wie zuvor auch, deaktiviert und kopiert, wobei jedoch kein Einrichtungsprozess der Simulationskomponenten eingeleitet wird. Stattdessen werden die Kreaturen direkt in der Simulationswelt platziert, womit die Funktion des Einleitungsskripts endet.

Von hier an weicht der Prozess entsprechend des ECS-Ansatzes stärker ab. So existiert für jede Art von Simulationskomponente ein System, welches bei jeder Aktualisierung der Simulation auf alle Entitäten mit der entsprechenden Komponente zugreift. Beim ersten Zugriff führt es dabei den zuvor übersprungenen Einrichtungsprozess der Simulationskomponente durch. Für die Initialisierung kommt dabei ein pro Kreatur gesetzter Zufallszahlengenerator zum Einsatz, um Parallelisierung sowie eine variable Bearbeitungsreihenfolge für die Komponente zu ermöglichen. Auch werden die der Simulationskomponente zugehörigen Abläufe direkt gestartet. Andere Komponenten der Kreatur, welche für die Ausführung besagter Abläufe benötigt werden,

werden dabei nicht mehr gespeichert, sondern vom System bei jeder Aktualisierung bereitgestellt, was durch die Speicherverwaltung von DOTS auf effiziente Weise geschieht.

Ist so der reguläre Simulationsablauf eingeleitet worden, werden alle darin verorteten Abläufe von den verschiedenen Systemen gehandhabt. Diese Prozesse lassen sich erneut an der Leerlauf-Simulationskomponente beispielhaft erläutern. Diese speichert hier lediglich die im Editor gesetzten Daten sowie das aktuelle Bewegungsziel und den Zeitpunkt, an welchem ein neues Bewegungsziel gesetzt werden wird. Bei jeder Aktualisierung der Simulation iteriert das zugehörige System über alle Leerlauf-Komponenten. Dabei prüft es zunächst für jede der Komponenten, ob der Zeitpunkt für die Festlegung eines neuen Bewegungsziels erreicht wurde und ermittelt, sofern dies zutrifft, ein neues Ziel, welches es zusammen mit einem neuen Zeitpunkt in der Komponente speichert.

Daraufhin berechnet das System die auszuführende Bewegung für die Annährung der jeweiligen Kreatur an ihr Ziel für das nächste Aktualisierungsintervall und speichert diese in der Kernskript-Komponente. Ein separates Bewegungssystem iteriert später über alle Kernskript-Komponenten und führt die so jeweils gespeicherten Bewegungsintentionen aus. So werden die Berechnungen für alle Kreaturen nach Komponenten geordnet ausgeführt und Abhängigkeiten minimiert.

Bei der DOTS-basierten Simulation besteht die Einschränkung, dass Kreaturen jeweils nur mit einer Simulationskomponente jeder Art ausgestattet werden können. Aufgrund der beschränkten Bearbeitungszeit hätte diese Restriktion vor Projektende nicht in zufriedenstellender Qualität aufgehoben werden können. Bei der Projektauswertung wurde dieser Sachverhalt beachtet; es wurde bei der späteren performanzbezogenen Datenerfassung insofern darauf Rücksicht genommen, dass die getesteten Simulationen in beiden Systemen dieser Einschränkung unterlagen und sich somit kein Vorteil basierend darauf ergab.

5.5 Sonstige Anwendungsfunktionen

Um die bisher separat erstellten Anwendungsbereiche zu verbinden, wurde im nächsten Schritt zunächst ein GUI-basiertes Menü als separate Szene umgesetzt, wie es in Abbildung 11 zu sehen ist. Dieses wird beim Start der Anwendung geöffnet und ermöglicht das Öffnen des Kreaturen- sowie des Modell-Editors über entsprechende Knöpfe. Zwei weiterer Knöpfe dienen dazu, ein Einstellungsmenü zu öffnen, auf welches nachfolgend noch näher eingegangen werden wird, respektive die Anwendung zu beenden. Die in den Editoren jeweils bereits vorhandenen Knöpfe zum Schließen dieser wurden entsprechend angepasst, sodass die Anwendung bei Interaktion mit ihnen zum Hauptmenü zurücknavigiert.

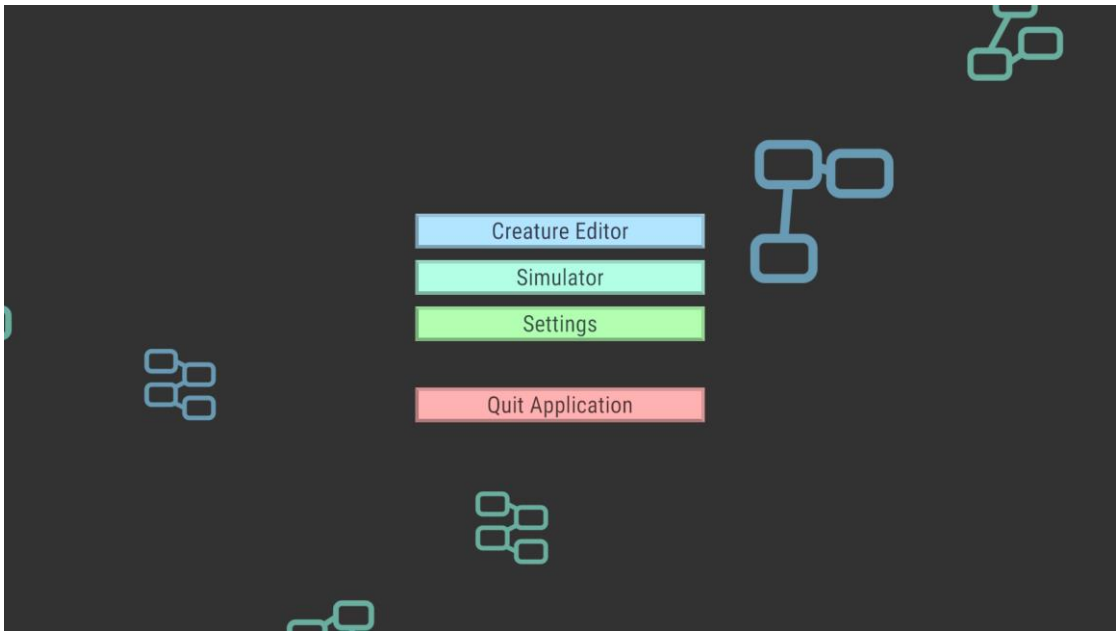

Abbildung 11: Das Hauptmenü

Abbildung 12: Das Einstellungsmenü

Das Einstellungsmenü, wie in Abbildung 12 dargestellt, ist in seiner aktuellen Form sehr rudimentär aufgebaut und beinhaltet lediglich eine einzelne Option, ist jedoch auf eine beliebige Erweiterung ausgelegt. Besagte Option ist das Löschen aller von den Editoren gespeicherten Daten. Diese Aktion muss stets über ein Dialogfenster, wie es in Abbildung 13 zu sehen ist, bestätigt werden, um eine versehentliche Ausführung mit unbeabsichtigtem Datenverlust zu vermeiden. Das Dialogfenster nutzt dabei das in vorherigen Abschnitten erläuterte Fokussystem.

Abbildung 13: Bestätigungsdialog zur Datenlöschung

Ein einfacher Zugang zu dieser Option ist gerade im Rahmen des Entwicklungsprozesses der Editoren relevant, da während diesem zu Testzwecken häufig unterschiedlich geartete Kreaturen wie auch Modelle erstellt werden müssen. Durch die Option ist ein einfaches Entfernen der Testdaten zur Gewährleistung der Übersichtlichkeit ohne großen Zeitaufwand möglich. Zudem können so speziell Daten entfernt werden, welche in früheren Iterationen der Editoren erstellt wurden und von den aktualisierten Systemen nicht mehr erkannt werden.

6 Ergebnisbewertung

Die Anwendung konnte insofern umgesetzt werden, dass der geplanten Funktionsumfang in grundlegender Form abgeschlossen abgebildet wurde. Entsprechend konnte im nächsten Schritt eine Auswertung des erstellten Programms anhand der zuvor festgelegten Kriterien erfolgen.

6.1 Vergleich der Performanz

Für die Auswertung der Leistungsfähigkeit der ermöglichten Simulationen wurden drei unterschiedliche PC-Systeme verwendet. Diesen werden zur Differenzierung entsprechend ihrer Art die Bezeichnungen „Desktop T“, „Desktop M“ sowie „Laptop M“ zugewiesen. Ihre wichtigsten, für die Performanz der Anwendung potentiell ausschlaggebenden, technischen Daten können Tabelle 1 entnommen werden. Für den vorliegenden Performanzvergleich sollte sich hauptsächlich der jeweilige Prozessor in relevantem Maße auf die erzielten Ergebnisse auswirken.

Tabelle 1: Technische Daten der für die Leistungserfassung genutzten Rechner

	Desktop T	**Desktop M**	**Laptop M**
Art	Desktop Topsegment	Desktop Mittelsegment	Laptop Mittelsegment
Prozessor	AMD Ryzen 9 5950X 16 Kerne; 32 Threads 3,4 - 4,9 GHz Taktfrequenz	AMD Ryzen 5 1600 6 Kerne; 12 Threads 3,2 - 3,6 GHz Taktfrequenz	Intel Core i7-8750H 6 Kerne; 12 Threads 2,2 - 4,1 GHz Taktfrequenz
Arbeitsspeicher	64 GB DDR4-Speicher 3600 MHz Taktfrequenz Dual Channel	16 GB DDR4-Speicher 2666 MHz Taktfrequenz Dual Channel	16 GB DDR4-Speicher 2666 MHz Taktfrequenz Dual Channel
Grafikkarte	AMD Radeon RX 6950 XT 16 GB VRAM 1,89 - 2,44 GHz Taktfrequenz	AMD Radeon RX 580 8 GB VRAM 1,34 - 1,41 GHz Taktfrequenz	NVIDIA GeForce GTX 1060 Mobile 6 GB VRAM 1,40 - 1,67 GHz Taktfrequenz
Betriebssystem	Windows 10 Pro Version 22H2, 19045.2965	Windows 10 Pro Version 22H2, 19045.2965	Windows 10 Home Version 22H2, 19045.2965

Die Anwendung wurde für die Datenerfassung wie geplant als Development Build exportiert und auf die PC-Systeme gebracht. Daraufhin wurde auf allen Rechnern im Programm selbst ein identisches Modell erstellt, an welchem die Leistungserfassung vorgenommen werden sollte. Das Modell umfasst dabei fünf verschiedene Kreaturenarten, welche alle im Rahmen des Projekts umgesetzten Verhaltensbausteine möglichst vielseitig abbilden und zu je gleicher Anzahl im Modell vertreten sind.

Die Gesamtzahl der zu simulierenden Kreaturen wurde an die Leistungsfähigkeit des jeweiligen Rechners angepasst. Die höchste genutzte Kreaturenmenge wurde dabei ungefähr so angesetzt, dass die Simulation ungeachtet des genutzten Skriptsystems noch als insgesamt flüssig wahrgenommen wurde und auf Nutzereingaben angenehm reagierte. So sollte eine realistische Anwendung des Programms abgebildet werden.

Pro PC wurden daraufhin je drei Datenaufzeichnungen pro Skriptsystem vorgenommen – mit der festgelegten Höchstmenge an Kreaturen sowie mit zwei Abstufungen dieser. Die ermittelte Datenmenge sollte so erhöht werden; zudem sollte die Möglichkeit eingeräumt werden, Rückschlüsse zur Auswirkung der Menge an simulierten Kreaturen auf die Leistungscharakteristik der Simulation zu ziehen. Jede einzelne Simulation wurde in Gänze hinsichtlich ihrer Leistungsmetriken protokoliert, während sie ohne externe Eingaben laufen gelassen wurde. Da die Simulation inhaltsbedingt recht statisch ablaufen würde – mit wenigen signifikanten Veränderungen über die Zeit hinweg – wurde ein Zeitraum von zehn Minuten pro Durchlauf gewählt. Dieser sollte ein umfassendes Gesamtbild des Simulationsablaufs aussagekräftig abdecken. Vor jeder durchgeführten Simulation wurde die Anwendung neu gestartet, um einen komplett unbeeinflussten Ablauf zu gewährleisten.

Die während dieser Simulationen jeweils erreichten Zeiten pro Bild sowie die diesen entsprechenden Bildfrequenzen sind nach Rechnern aufgeteilt in Tabelle 2, Tabelle 3 sowie Tabelle 4 festgehalten. Besagte Tabellen umfassen pro Simulation den Median sowie das obere und untere Quartil aller während ihrer Durchführung erfassten Werte. Letztere stellen dabei den Mittelwert zwischen dem Median sowie dem niedrigsten beziehungsweise höchsten aufgezeichneten Wert dar und gewähren somit Einblick in die Schwankungen der Performanz.

Tabelle 2: Bei der Leistungserfassung erzielte Zeiten pro Bild auf Desktop T

	Unteres Quartil	**Median**	**Oberes Quartil**
Standard 2000 Kreaturen	2,17 ms (461 FPS)	2,25 ms (444 FPS)	2,38 ms (420 FPS)
Standard 4000 Kreaturen	4,30 ms (233 FPS)	15,71 ms (64 FPS)	16,17 ms (62 FPS)
Standard 6000 Kreaturen	40,50 ms (25 FPS)	41,47 ms (24 FPS)	54,19 ms (18 FPS)
DOTS 2000 Kreaturen	1,08 ms (926 FPS)	1,20 ms (833 FPS)	1,37 ms (730 FPS)
DOTS 4000 Kreaturen	1,2 ms (833 FPS)	1,57 ms (637 FPS)	2,47 ms (405 FPS)
DOTS 6000 Kreaturen	16,61 ms (60 FPS)	16,72 ms (60 FPS)	16,84 ms (59 FPS)

Tabelle 3: Bei der Leistungserfassung erzielte Zeiten pro Bild auf Desktop M

	Unteres Quartil	Median	Oberes Quartil
Standard 1000 Kreaturen	3.58 ms (279 FPS)	3,78 ms (265 FPS)	8,16 ms (123 FPS)
Standard 2000 Kreaturen	5,41 ms (185 FPS)	14,94 ms (67 FPS)	15,37 ms (65 FPS)
Standard 3000 Kreaturen	23,00 ms (43 FPS)	34,55 ms (29 FPS)	37,32 ms (27 FPS)
DOTS 1000 Kreaturen	3,62 ms (276 FPS)	3,75 ms (267 FPS)	6,54 ms (153 FPS)
DOTS 2000 Kreaturen	3,80 ms (263 FPS)	3,92 ms (255 FPS)	10,98 ms (91 FPS)
DOTS 3000 Kreaturen	14,85 ms (67 FPS)	15,11 ms (66 FPS)	15,28 ms (65 FPS)

Tabelle 4: Bei der Leistungserfassung erzielte Zeiten pro Bild auf Laptop M

	Unteres Quartil	Median	Oberes Quartil
Standard 1000 Kreaturen	5,26 ms (190 FPS)	5,93 ms (169 FPS)	10,39 ms (96 FPS)
Standard 2000 Kreaturen	6,69 ms (149 FPS)	10,78 ms (93 FPS)	14,48 ms (69 FPS)
Standard 3000 Kreaturen	18,91 ms (53 FPS)	19,19 ms (52 FPS)	19,66 ms (51 FPS)
DOTS 1000 Kreaturen	6,53 ms (153 FPS)	6,93 ms (144 FPS)	8,83 ms (113 FPS)
DOTS 2000 Kreaturen	6,93 ms (144 FPS)	7,23 ms (138 FPS)	9,75 ms (103 FPS)
DOTS 3000 Kreaturen	8,16 ms (123 FPS)	12,78 ms (78 FPS)	12,99 ms (77 FPS)

Über fast alle Simulationen hinweg ist eine merkliche Leistungssteigerung bei der Nutzung von DOTS ersichtlich. Bei Simulationen mit einer geringen Menge an Kreaturen fällt dieser Unterschied noch eher gering aus; im Falle der 1000 Kreaturen umfassenden Simulation auf Laptop M liegt die von DOTS erzielte Leistung sogar leicht unter der des Standardsystems. Bei Erhöhung der Kreaturenmenge sinkt die durch das Standardsystem erzielte Performanz schnell ab; die Leistung der DOTS-basierten Simulationen sinkt dabei ebenfalls, jedoch in geringerem Maße.

Mit der höchsten simulierten Anzahl an Kreaturen erreicht oder übersteigt die Bildfrequenz der DOTS-Simulation durchweg den Richtwert von 60 FPS. Im Standardsystem wird dies nicht erreicht – bei beiden Desktop-Rechnern liegt die erzielte Bildfrequenz bei der anspruchsvollsten Simulation zudem unter den als Untergrenze für ein flüssiges Nutzungserlebnis gesehenen 30 FPS.

Die im Mittel erzielte Zeit pro Bild steigt im Standardsystem deutlich mit beiden Erhöhungen der Kreaturenanzahl, wobei sich exponentielles Wachstum abzeichnet. In den DOTS-basierten Simulationen tritt diese exponentielle Steigerung ebenfalls auf, fällt jedoch bei der ersten Erhöhung zunächst weit marginaler aus. Der zweiten Erhöhung der Anzahl an Kreaturen folgt im DOTS-System in jedem Fall ein stärkeres Wachstum der Zeit pro Bild, als es im Standardsystem der Fall ist.

Beim Betrachten der Quartile zeigt sich, dass die erfassten Schwankungen der Performanz im Standardsystem insgesamt höher ausfallen als bei Nutzung von DOTS, wobei die Abweichungen vom Median sich mit steigender Kreaturenmenge weiter vergrößern. Das DOTS-basierte Simulationssystem weist dadurch insgesamt eine stabilere Bildfrequenz auf. Die im Mittel erzielten Zeiten pro Bild und ihre Abweichungen blieben über den Verlauf aller Simulationen hinweg jeweils konstant.

6.2 Vergleich des Entwicklungsprozesses

Die Umsetzung der Anwendung mithilfe des klassischen Unity-Entwicklungssystems ging insgesamt sehr gut von der Hand. Das Anwendungskonzept in seiner Gesamtheit ließ sich mit wenigen Hürden auf die Entwicklungsumgebung konkretisieren sowie in ihr zeiteffizient implementieren. Im Speziellen verlief dabei die Umsetzung der Simulationsdurchführung ohne größere Probleme. Zurückführen lässt sich dies vor allem auf die hohe Menge von Unity zur Verfügung gestellter Standardsysteme, welche gut auf den Anwendungsfall zugeschnitten werden konnten, sowie die quantitativ wie auch qualitativ stark ausgeprägten Hilfsressourcen.

Die Implementierung der Simulationsdurchführung mithilfe von Unity DOTS gestaltete sich, wie zuvor angenommen, wesentlich problematischer. Die zunächst größte Hürde stellte dabei das Umdenken, weg von der verbreiteten objektorientierten Programmierung sowie die damit einhergehende grundlegend unterschiedliche Code-Struktur, dar. Aufgrund damit einhergehender Änderungen mussten bei der Implementierung zudem häufig Umwege im Vergleich mit der klassischen Umsetzung genommen werden. Gerade die Einschränkung der nutzbaren Datentypen bedingte dabei das Verwenden von umständlicher gestalteten Alternativen, deren Verwendung sich komplexer gestaltet und leichter zu Fehlern bei der Ausführung des Programmcodes führen kann.

Während klassische Unity-Skripte Klassen darstellen, sind die in ECS genutzten Systeme und Komponenten Strukturen. Dies bedingt bei der Entwicklung mit diesem System, dass das in der objektorientierten Programmierung verbreitete Konzept der Vererbung, durch das einander ähnelnde Klassen auf von einer Elternklasse bereitgestellten Code zugreifen und diesen so teilen

können, nicht nutzbar ist. Das stattdessen einzusetzende Konzept der Aufteilung in möglichst generische Komponenten brachte während des Projekts erheblichen Mehraufwand sowie deutlich verminderte Übersichtlichkeit bei der Codeentwicklung mit sich und verlangsamte diese entsprechend merklich.

Insgesamt ergab sich durch die deutlich starrere Struktur von ECS in Hinblick auf Code-Aufbau und -Ausführung eine höhere Fehleranfälligkeit und damit einhergehend ein häufigeres Verhindern korrekter Code-Ausführung. Entsprechende Fehler werden dabei zumeist erst während der Programmausführung ersichtlich, da der Unity-Editor durch Fehlermeldungen auf ihre Auswirkungen verweist. Besagte Fehlermeldungen sind in Bezug auf die DOTS-Funktionen jedoch häufig sehr vage formuliert, wodurch die Fehlersuche erschwert wurde. Konträr dazu traten bei der klassischen Unity-Entwicklung, aufgrund der höheren Freiheit beim Strukturieren des Programmcodes, solche Fehler weit seltener auf und wurden aussagekräftiger ausgewertet, was ihre Behebung erleichterte. Rein algorithmische Fehler, bei welchen die Funktion des geschriebenen Codes nicht der gewünschten Anwendungsfunktion entspricht, sind vom Systemwechsel kaum betroffen.

Ein ähnliches Problem ergab sich bei der Nutzung des Burst Compilers, welcher automatisiert auf alle Skriptteile angewandt wird, welche im Programmcode dafür markiert wurden. Einige von Unity bereitgestellte Funktionen werden dabei jedoch nicht unterstützt; das Markieren von Programmteilen, in denen entsprechende Funktionen verwendet werden, führt also zu Fehlern und verhindert ein Exportieren der Anwendung aus dem Editor. Leider ist im Vorhinein nicht ersichtlich, welche Funktionen davon betroffen sind und auch mit Burst kompatible alternative Ansätze sind höchstens indirekt auffindbar. Entsprechend findet sich die Lösung dieser Probleme meist darin, die entsprechenden Programmteile schlicht aus der Burst-Kompilierung auszuschließen und so auf deren leistungsbezogene Vorteile zu verzichten. Dieser Prozess gestaltete sich zudem sehr ineffizient, da die von Unity zu diesen Burst-Problemen gegebenen Fehlermeldungen nicht persistent sowie weniger aussagekräftig sind als das übliche von Unity zu Code-Fehlern gegebene Feedback.

Die Nutzung von ECS machte zudem die Performanz von geschriebenem Code weniger ersichtlich. So gestaltet sich die Ausführung des Codes komplexer und weniger transparent, da stark spezialisierte und ausschließlich auf Leistung zugeschnittene Strukturen verwendet werden. Die Auswertung der Funktion dieser Strukturen im Editor gestaltet sich schwierig und konkrete Abläufe sind weniger nachvollziehbar, als es in Unity klassischerweise der Fall wäre. Zudem legt Unity DOTS viele Ansatzpunkte offen, welche sich zum Teil erheblich auf die Leistung der umgesetzten Programmfunktionen auswirken. Diese sind, zusammen mit der Größe besagter Auswirkungen, jedoch oft nicht direkt ersichtlich.

Letzteres Problem ist hauptsächlich auf den sehr begrenzten Umfang der verfügbaren Dokumentation zurückzuführen. Diese schlüsselt zwar die grundsätzliche Nutzung der DOTS-Systeme gut auf, beleuchtet aber die meisten zugehörigen Teilkomponenten nur sehr oberflächlich oder gar nicht. Somit müssen viele Möglichkeiten der Systeme manuell entdeckt und getestet werden. Gerade in Verbindung mit den sehr starren Vorgaben an Struktur und Ausführungsreihenfolge des Programmcodes gestaltet sich dies meist sehr aufwändig und wenig intuitiv. Durch die vergangene, kontinuierliche Entwicklung von DOTS sind viele Dokumentationsbestandteile noch nicht für den aktuellen Stand angepasst und nur in für ältere DOTS-Versionen ausgelegten Versionen der Dokumentation zu finden. Diese vermitteln jedoch entsprechend nicht mehr zwingend optimale Ansätze und/oder behandeln mittlerweile gar nicht mehr verfügbare DOTS-Bestandteile.

Zum Ausgleichen der unausgereiften Dokumentation werden von Unity Beispielprojekte zur Verfügung gestellt, welche verschiedene DOTS-Funktionen praktisch demonstrieren und ihre konkrete Anwendung näher erläutern, als die textbasierte, sehr theoretische Dokumentation. Verständlicherweise demonstrieren sie jedoch ebenfalls nicht jede erdenkliche Funktion. Das Finden von Anwendungsfällen spezifischer Funktionen in den verschiedenen Beispielprojekten gestaltet sich zudem weit umständlicher als das schlichte Nutzen der Suchfunktion in der Dokumentation.

Generell sind auch externe Hilfsressourcen für DOTS gerade quantitativ weitaus begrenzter als für die reguläre Unity-Entwicklung üblich, da die Menge sachkundiger Entwickelnder aufgrund des Entwicklungsstatus der Technologie sehr begrenzt ist. In diesem Rahmen dennoch behandelte Themen gehen zudem selten über Grundlagen hinaus und sind für konkretere Anwendungsfälle wenig hilfreich. Auch unter diesem Aspekt gestaltet sich zusätzlich die lang andauernde Weiterentwicklung von DOTS problematisch, da vorhandene Ressourcen häufig schlicht nicht mehr auf aktuelle Versionen anwendbar sind.

6.3 Beurteilung des Nutzungserlebnisses der Anwendung

Das im Rahmen des Projekts erstellte Programm bietet in seiner aktuellen Form bereits eine als gut zu bewertende Nutzungserfahrung. So konnte eine Anwendung umgesetzt werden, in welcher alle in der Planung vorgesehenen Funktionen mindestens in grundlegendem Umfang implementiert sind.

Aufgrund der größtenteils fehlenden Anleitung dürfte sich die Verwendung durch Dritte dabei jedoch zunächst als umständlich erweisen und mit einem stark experimentellen Lernprozess

verbunden sein. Sind Kenntnisse zu den grundlegenden Arbeitsabläufen jedoch vorhanden, gestaltet sich die Nutzung des Programms insgesamt unkompliziert. Das vielfältige Erstellen von Modellen sowie das Durchführen von Simulationen ist spielerisch-experimentell möglich und dadurch leicht anwendbar. Die aktuell genutzten Konzepte für Interaktion, Visualisierung und Ähnliches funktionieren dabei gut, wenngleich zum Teil etwas umständlich und erzielen ein rundes Anwendungserlebnis.

Insgesamt ist die Materie der Populationssimulationen anhand der Anwendung gut zugänglich und ohne thematische Vorkenntnisse erkundbar. Das Programm entspricht also der dargelegten Grundidee und ist für den definierten Nutzungszweck gut zu verwenden.

7 Fazit

Insgesamt konnte das im Mittelpunkt dieser Arbeit stehende Projekt erfolgreich abgeschlossen und die Nutzung von Unity DOTS im gewählten Anwendungsgebiet vielfältig untersucht werden. In den folgenden Abschnitten werden die Bedeutung respektive die Implikationen der erzielten Resultate näher beleuchtet.

7.1 Zukunft der Technik

Es wurde festgestellt, dass durch die Nutzung von Unity DOTS im in dieser Arbeit betrachteten Anwendungsfall eine merkliche Leistungssteigerung erreicht werden kann. Dieses Ergebnis entspricht dem durch das System verfolgten Ziel und stützt ähnliche Resultate anderer in Entwicklung befindlicher oder bereits veröffentlichter Projekte. Als besonderer Vorteil von DOTS ist dabei der flexible Implementierungsansatz herauszuheben, durch den dessen einzelnen Bestandteile in beliebiger Kombination genutzt werden und auf relevante Anwendungsteile, welche von Performanzverbesserungen stark profitieren, beschränkt werden können. Durch diese Flexibilität und gleichzeitige Effektivität kann DOTS für Entwickelnde einen durchaus praktikablen Ansatzpunkt für die Realisierung ihrer Projekte darstellen.

Jedoch ist gleichwohl anzumerken, dass der aktuelle Stand des Systems, wie er von den veröffentlichten Entwicklungsversionen widergespiegelt wird, noch nicht optimal ausfällt. So bringt die Nutzung von Unity DOTS, wie zuvor näher erläutert, einige Hürden mit sich, da Arbeitsprozesse im Vergleich zu denen länger bestehender Systeme nicht optimiert sind. Gerade aufgrund des nötigen Umdenkens weg von verbreiteteren Programmieransätzen ist der Einsatz der Technologie zudem zunächst mit einigem Aufwand verbunden, welcher durch fehlende oder unzulängliche Lernressourcen weiter erhöht wird.

Eine Etablierung von DOTS setzt daher grundsätzlich eine fortgeführte Entwicklung beziehungsweise verstärkte Unterstützung des Systems seitens Unity voraus. Neben fundamentaler Wartung bietet sich dabei etwa eine weitere Optimierung der für DOTS genutzten Arbeitsprozesse an, um eine angenehmere Nutzung durch Entwickelnde und dadurch eine einfachere Integrierung in Projekte möglich zu machen. Gerade eine Verbesserung der zur Verfügung stehenden Werkzeuge und Dokumentation im Sinne des Erleichterns einer optimalen Nutzung würde sich entsprechend der aktuellen Unzulänglichkeiten des Systems positiv auswirken.

Unity DOTS als Softwarelösung zur Leistungssteigerung stellt insgesamt also einen guten Ansatzpunkt dar, welcher Anwendungen verbessern oder gar erst vollumfänglich ermöglichen

kann. Bei Erfüllung der eben dargestellten Voraussetzung der Weiterentwicklung ist eine Nutzung des Systems für passende Anwendungsgebiete entsprechend zu empfehlen.

Dabei können die Ansätze des Systems auch außerhalb von Unity DOTS sinnvoll genutzt werden, gerade in Falle des zentralen datenorientierten Designs. Dieses bietet universell großes Potential für enorm komplexe Programme wie auch für Anwendungen, welche auf Leistungsschwachen Endgeräten möglichst performant ausgeführt werden sollen.

7.2 Bedeutung für verwandte Anwendungsgebiete

Wie eingangs erwähnt passen die von DOTS versprochenen performanzbezogenen Vorteile gut zum Leistungsprofil komplexer Simulationen, was sich auch praktisch anhand des durchgeführten Projekts bestätigte. Bei einer intensiveren Beschäftigung, als sie im Rahmen dieser Arbeit möglich gewesen wäre, könnten durch Optimierung der Anwendung die Auswirkungen der festgestellten Vorzüge dabei potentiell noch verstärkt werden.

Unity im Allgemeinen und besonders DOTS eignen sich also gut für die Umsetzung von Populationssimulationen. Dabei ist anzunehmen, dass diese Eignung sich auch bei Nutzung anders gearteter Simulationen positiv auswirken würde. Gerade bei einer visuellen, interaktiven Darstellungsart bietet sich dabei die Nutzung von Unity im Speziellen an, um vielfältige, ansprechende Anwendungen mit vergleichsweise geringem Aufwand zu entwickeln. Durch die Nutzung von DOTS kann dabei zusätzlich eine gleichzeitig hohe Leistungsfähigkeit erzielt werden.

Um das Bild des Nutzens von DOTS weiter zu konkretisieren, bieten sich weitergehende Untersuchungen an, welche entsprechende Nachweise anhand anderer Simulationsarten zum Ziel haben und/oder das System auf vielfältigere Weise betrachten.

7.3 Zukunft des erstellten Prototyps

Das Konzept eines Editors für Populationssimulationen konnte durch das Projekt gut abgebildet werden. Selbst mit den durch den Projektrahmen sehr eingeschränkten Funktionen erzielt die entwickelte Anwendung interessante Resultate und regt zum Ausprobieren der vorhandenen Möglichkeiten an. Der erstellte Prototyp stellt also eine erfolgreiche Umsetzung des eingangs formulierten konzeptionellen Ansatzpunktes dar.

Wie bereits zuvor erwähnt, stellt die umgesetzte Anwendung eine abgeschlossene, für den vorgesehenen Nutzungszweck verwendbare Basis dar. Eine Weiterentwicklung dieser Basis bietet

sich an; gerade in Anbetracht der Tatsache, dass die Limitierungen vieler Bestandteile der Anwendung schlicht durch die im Projektrahmen stark begrenzte Entwicklungszeit bedingt sind. Dabei sind diverse mögliche Ansatzpunkte in Betracht zu ziehen.

So könnten zunächst Funktionen des Programms, welche im Rahmen des Projekts nur in eingeschränkter Form umgesetzt wurden, verbessert werden. Allen voran stünde dabei die Erweiterung des DOTS-basierten Simulationssystems, sodass in dieser Kreaturen mehrere Simulationskomponenten gleicher Art besitzen können, wie ursprünglich vorgesehen. Dies stellt die einzige Einschränkung dar, welche im alternativen Simulationssystem nicht besteht – ihre Aufhebung wäre also Voraussetzung für eine Funktionsangleichung. So würden nicht nur die Möglichkeiten der DOTS-basierten Simulationen deutlich erweitert, sondern es würden auch weitergehende Vergleiche beider Systeme ermöglicht werden.

Darüber hinaus böten sich zunächst verschiedene Ansatzpunkte bei der Überarbeitung der Nutzungserfahrung. Grundlegende Verbesserungen würden sich dabei hinsichtlich der Verwaltung von Kreaturen sowie Modellen anbieten; etwa indem eine leichtere Verwaltung durch Kopieren sowie Anpassen der Reihenfolge dieser ermöglicht wird. Analog dazu würde das Ermöglichen des Verschiebens von Kreaturen zwischen Ökosystemen eine sinnvolle Ergänzung darstellen. Die Umsetzung eines Export- respektive Importsystems für Daten von Kreaturen, Ökosystemen, Bausteinen und Modellen könnte ebenfalls in Betracht gezogen werden.

Das aktuelle System zum Erstellen von Kreaturen ließe sich zudem insofern überarbeiten, dass Bausteine statt in schlichten Listen beispielsweise in zweidimensionalen Diagrammen platziert werden könnten. So würde eine bessere Übersichtlichkeit und Ordnung, gerade bei komplexeren Kreaturen, ermöglicht werden. Zum Festlegen von Interaktionen von Bausteinen würde es sich dann anbieten, Nutzende visuelle Verbindungen zwischen ihnen ziehen zu lassen, um diese ebenso klarer darzustellen.

Auch würde sich eine deutliche Erweiterung der während der Simulationsdurchführung verfügbaren Werkzeuge anbieten. So wäre es sinnvoll, einzelne Kreaturen auswählen zu können, um beispielsweise eine GUI-basierte Übersicht ihrer Attributwerte zu erhalten. Ähnliche Übersichten könnten zudem für alle Kreaturen eines Typs oder anderweitig festgelegte Gruppen in Betracht gezogen werden. Generell würde erweitertes Feedback zu den aktuellen Handlungen und/oder dem Status von Kreaturen einen Ansatzpunkt darstellen. Anbieten würden sich dabei etwa Animationen, im Umfeld der Kreatur platzierte Symbole oder Audio-Einblendungen.

Um ein detaillierteres Auswerten der Inhalte durchgeführter Simulationen möglich zu machen, würde es sich zudem anbieten, Simulationsdaten automatisiert aufzuzeichnen; etwa Attributwerte für Kreaturen sowie Kreaturengruppen und deren jeweilige Veränderungen über den Simulationsablauf hinweg. So wäre, gerade in Verbindung mit einem während und/oder nach der

jeweiligen Simulation aktivierbaren Visualisierungssystem, ein besseres Nachvollziehen der Entwicklung verschiedener Werte beziehungsweise von deren Auswirkungen möglich.

Ebenfalls könnten manuelle Eingriffsmöglichkeiten während der Simulation geschaffen werden. So etwa indem Kreaturen durch Nutzereingaben beliebig entfernt, erschaffen, bewegt oder anderweitig manipuliert werden könnten. Die Möglichkeit des manuellen Anpassens von Attributwerten wäre ebenso in Betracht zu ziehen.

Einen weiteren übergeordneten Ansatzpunkt stellt die Erweiterung der momentan bestehenden rudimentären Inhalte dar. Allem voran wäre in diesem Rahmen die Erweiterung der für die Kreaturenerstellung verfügbaren Fähigkeits-Bausteine sinnvoll. Entsprechend der Programm-Prämisse, Populationssimulationen mit unterschiedlichst gearteten Kreaturen zu ermöglichen, könnten dabei verschiedene der vielfältigen in der Natur vorfindbare Verhaltensweisen abgebildet werden. Beispiele für Fähigkeiten mit interessanten Effekten wären etwa das Aufnehmen und Speichern von Stoffen, das Kämpfen oder die geschlechtliche respektive ungeschlechtliche Fortpflanzung. Auch böte sich eine Erweiterung bestehender Fähigkeiten an; für die Anpassung des Aussehens erstellter Kreaturen könnten beispielsweise weitere Modelle zur Verfügung gestellt und die Bearbeitung zusätzlicher Parameter, wie zum Beispiel der Kreaturengröße, ermöglicht werden.

Neben der direkten Erweiterung der verfügbaren Fähigkeiten wäre es auch sinnvoll, die Funktion von Aktionen, welche das Verhalten von Kreaturen zusätzlich stark beeinflussen können, zu erweitern. Sinnvoll wäre es beispielsweise, pro Aktionsblock mehrere Auslöser sowie Effekte auswählen zu können. So könnten die definierten Aktionen übersichtlicher gestaltet und das Erstellen komplexerer Aktionen vereinfacht werden. Damit einhergehend böte sich auch die Integration logischer Operatoren an, durch die etwa mehrere definierte Auslöser aktiviert werden müssen, um die zugehörigen Effekte zu aktivieren. Auch eine generelle Erweiterung der verfügbaren Auslöser- sowie Effekttypen und ein Ausbauen der für sie jeweils gebotenen Einstellungsmöglichkeiten wäre sinnvoll.

Abseits der Ausdehnung der möglichen Kreaturenvielfalt böte es sich des Weiteren an, die aktuell sehr einfach aufgebaute Welt, in welcher die Simulationen ablaufen, durch stärkere Anpassungsmöglichkeiten auszugestalten. Statt der aktuellen flachen, rechteckigen Ebene ohne weitere Merkmale könnte beispielsweise aus verschiedenen Parametern ein dreidimensionales Terrain erzeugt werden. Eine solche Umgebung könnte sich auf das Kreaturenverhalten auswirken, beispielsweise indem Bewegungsabläufe auf bestimmte Steigungen begrenzt werden würden. Zusätzlich könnte die Simulationsumgebung durch natürliche Phänomene erweitert werden; etwa Niederschlag oder global synchronisierte Zyklen. So könnten zum Beispiel Jahreszeiten umgesetzt werden, welche jeweils unterschiedliches Kreaturenverhalten begünstigen. Diese

Erweiterungen der Simulationswelt könnten dabei zusätzlich als regional unterschiedlich geartet implementiert werden, wodurch in verschiedenen Regionen der Welt beispielsweise unterschiedlich geformtes Terrain auftreten könnte. Dies würde etwa Vergleiche der Auswirkung unterschiedlicher Simulationsbedingungen innerhalb einer einzelnen Simulation ermöglichen.

In der nur in sehr begrenztem Umfang verfügbaren Dokumentation der Anwendung findet sich ein weiterer relevanter Ansatzpunkt. Um Nutzenden einen unkomplizierten Einstieg in ihre Nutzung zu ermöglichen würde sich dabei etwa eine interaktive Anleitung innerhalb des Programms anbieten. Sinnvollerweise würde dieses System durch eine ausführlichere, textbasierte Dokumentation ergänzt werden.

Einer etwaigen Veröffentlichung sollte zuletzt auch eine generelle Optimierung der verschiedenen Systeme des Programms sowie des allgemeinen Nutzungserlebnisses vorangehen. So böte sich eine Erweiterung der in der Anwendung verfügbaren Einstellungen an; zum Beispiel durch Optionen für Barrierefreiheit. Auch Anpassungsmöglichkeiten für Ausgabeaspekte, beispielsweise die grafische Komplexität sowie Steuerungsparameter, wie das Ändern von für Aktionen genutzten Tasten, wären angebracht.

Zum Erreichen einer besseren Nutzungserfahrung wäre zudem eine Überarbeitung respektive Erweiterung der visuellen Programmelemente denkbar. So würden sich etwa Effekte, beispielsweise erweiterte Animationen, für Interaktionen und das generelle Navigieren der Anwendung anbieten. Für die momentan rein visuelle Darstellung des Programms böte sich zudem eine auditive Ergänzung bei Interaktionen an. Um auftretende Stille zu vermeiden und Nutzenden eine höhere Immersion in das Programm zu ermöglichen, könnte zudem Musik und/oder eine das generelle Simulationsgeschehen abbildende Kulisse an Hintergrundgeräuschen implementiert werden.

Insgesamt bieten sich somit für eine Weiterentwicklung des erstellten Prototyps eine Vielzahl an unterschiedlichen Ansatzpunkten, auch über die hier erwähnten hinaus.

VI Literaturverzeichnis

[1] N. Llopis, „Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP) – Games from Within“, 04. Dezenmber 2009, [Online]. Verfügbar unter: https://gamesfromwithin.com/data-oriented-design. [Zugriff am 19. Januar 2023].

[2] Unity Software Inc., „Common game development terms and definitions | Game design vocabulary | Unity“, Unity Software Inc., [Online]. Verfügbar unter: https://unity.com/how-to/beginner/game-development-terms. [Zugriff am 21. Januar 2023].

[3] L. Marques und J. Rodriques, „Tech Bytes: Instructions Per Clock (IPC)“, Intel Corporation, [Online]. Verfügbar unter: https://www.intel.com/content/www/us/en/partner-alliance/sales-enablement/topic/resource/video/tech-bytes-instructions-per-clock. [Zugriff am 09. Mai 2023].

[4] S. Harding, „What Is a CPU Core? A Basic Definition | Tom's Hardware“, Future plc, 17. Juni 2022, [Online]. Verfügbar unter: https://www.tomshardware.com/news/cpu-core-definition,37658. [Zugriff am 09. Mai 2023].

[5] Intel Corporation, „CPU-Geschwindigkeit: Was versteht man unter CPU-Taktfrequenz? | Intel“, Intel Corporation, [Online]. Verfügbar unter: https://www.intel.de/content/www/de/de/gaming/resources/cpu-clock-speed. [Zugriff am 09. Mai 2023].

[6] Unity Software Inc., „Best practices for profiling game performance | Unity“, Unity Software Inc., [Online]. Verfügbar unter: https://unity.com/how-to/best-practices-for-profiling-game-performance. [Zugriff am 14. Mai 2023].

[7] J. Naughton, „We’re approaching the limits of computer power - we need new programmers now | John Naughton | The Guardian“, Guardian News & Media Ltd., 11. Januar 2020, [Online]. Verfügbar unter: https://www.theguardian.com/commentisfree/2020/jan/11/we-are-approaching-the-limits-of-computer-power-we-need-new-programmers-n-ow. [Zugriff am 16. Januar 2023].

[8] Intel Corporation, „Intel® Core™ Prozessoren der 12. Generation_ Anpassungsfähige...“, Intel Corporation, [Online]. Verfügbar unter: https://www.intel.de/content/www/de/de/products/docs/processors/core/12th-gen-processors. [Zugriff am 16. Januar 2023].

[9] Advanced Micro Devices, Inc., „AMD Infinity Architecture Technology | AMD“, Advanced Micro Devices, Inc., [Online]. Verfügbar unter: https://www.amd.com/de/technologies/infinity-architecture. [Zugriff am 21. Januar 2023].

[10] Advanced Micro Devices, Inc., „AMD 3D V-Cache™ Technology | AMD“, Advanced Micro Devices, Inc., [Online]. Verfügbar unter: https://www.amd.com/en/technologies/3d-v-cache. [Zugriff am 21. Januar 2023].

[11] A. Frumusanu, „Apple Announces The Apple Silicon M1: Ditching x86 - What to Expect, Based on A14“, Future plc, 10. November 2020, [Online]. Verfügbar unter: https://www.anandtech.com/show/16226/apple-silicon-m1-a14-deep-dive. [Zugriff am 16. Januar 2023].

[12] RISC-V International, „History - RISC-V International“, RISC-V International, [Online]. Verfügbar unter: https://riscv.org/about/history/. [Zugriff am 21. Januar 2023].

[13] RISC-V International, „RISC-V International and Intel team up to accelerate RISC-V adoption: Introducing Intel Pathfinder for RISC-V | Intel Corporation - RISC-V International“, RISC-V International, 30. August 2022, [Online]. Verfügbar unter: https://riscv.org/blog/2022/08/risc-v-international-and-intel-team-up-to-accelerate-risc-v-adoption-introducing-intel-pathfinder-for-risc-v-intel-corporation/. [Zugriff am 21. Januar 2023].

[14] Intel Corporation, „Intel Pathfinder for RISC-V | Intel® Pathfinder for RISC-V* Delivers New Capabilities for Pre-Silicon Development“, Intel Corporation, 30. August 2022, [Online]. Verfügbar unter: https://pathfinder.intel.com/news/intel-pathfinder-for-risc-v-delivers-new-capabilities-for-pre-silicon-development/. [Zugriff am 21. Januar 2023].

[15] RISC-V International, „About RISC-V - RISC-V International“, RISC-V International, [Online]. Verfügbar unter: https://riscv.org/about/. [Zugriff am 21. Januar 2023].

[16] A. Blake-Davis, „AMD FidelityFX Super Resolution is Here - AMD Community“, Advanced Micro Devices, Inc., 22. Juni 2021, [Online]. Verfügbar unter: https://community.amd.com/t5/gaming/amd-fidelityfx-super-resolution-is-here/ba-p/477919. [Zugriff am 16. Januar 2023].

[17] E. Kilgariff, H. Moreton, N. Stam und B. Bell, „NVIDIA Turing Architecture In-Depth | NVIDIA Technical Blog“, Nvidia Corporation, 14. September 2018, [Online]. Verfügbar unter: https://developer.nvidia.com/blog/nvidia-turing-architecture-in-depth/. [Zugriff am 16. Januar 2023].

[18] Intel Corporation, „Intel® Arc™ - X^e Super Sampling“, Intel Corporation, [Online]. Verfügbar unter: https://www.intel.de/content/www/de/de/products/docs/arc-discrete-graphics/xess. [Zugriff am 16. Januar 2023].

[19] A. Burnes, „GeForce RTX 30 Series Performance Accelerates With Resizable BAR Support | GeForce News | NVIDIA“, Nvidia Corporation, 30. März 2021, [Online]. Verfügbar unter: https://www.nvidia.com/en-us/geforce/news/geforce-rtx-30-series-resizable-bar-support/. [Zugriff am 21. Januar 2023].

[20] Advanced Micro Devices, Inc., „AMD Smart Access Memory | AMD“, Advanced Micro Devices, Inc., [Online]. Verfügbar unter: https://www.amd.com/de/technologies/smart-access-memory. [Zugriff am 21. Januar 2023].

[21] Intel Corporation, „Was ist BAR (Resizable BAR) und wie aktiviere ich es?“, Intel Corporation, 08. August 2022, [Online]. Verfügbar unter: https://www.intel.de/content/www/de/de/support/articles/000090831/graphics. [Zugriff am 21. Januar 2023].

[22] K.-H. Waldmann und W. E. Helm, Simulation stochastischer Systeme: Eine anwendungsorientierte Einführung, Berlin/Heidelberg: Springer Gabler, 2016.

[23] U. Hedtstück, Simulation diskreter Prozesse: Methoden und Anwendungen, Berlin/Heidelberg: Springer Vieweg, 2013.

[24] Water Education Foundation, „San Francisco Bay Model - Water Education Foundation“, Water Education Foundation, [Online]. Verfügbar unter: https://www.watereducation.org/aquapedia/san-francisco-bay-model. [Zugriff am 22. Januar 2023].

[25] T. Scott, „This giant model stopped a terrible plan - YouTube“, Pad 26 Limited, 01. Dezember 2016, [Online]. Verfügbar unter: https://www.youtube.com/watch?v=i70wkxmumAw. [Zugriff am 22. Januar 2023].

[26] Stiftung Medien in der Bildung, „Simulation - e-teaching.org“, Stiftung Medien in der Bildung, 21. März 2016, [Online]. Verfügbar unter: https://www.e-teaching.org/didaktik/gestaltung/visualisierung/simulation. [Zugriff am 25. Januar 2023].

[27] S. Elter, Zum Einfluss von Simulationen auf das funktionale Denken: Am Beispiel von Mathematisierungssituationen im MATHEMATIK-Labor, Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2020.

[28] game – Verband der deutschen Games-Branche e. V., „Was sind digitale Spiele? | game“, game – Verband der deutschen Games-Branche e. V., [Online]. Verfügbar unter: https://www.game.de/topics-archive/digitalespiele/. [Zugriff am 27. Januar 2023].

[29] D. Behnke, „Lernen mit digitalen Spielen im Unterricht | bpb.de“, Bundeszentrale für politische Bildung, 28. März 2022, [Online]. Verfügbar unter: https://www.bpb.de/themen/kultur/digitale-spiele/504550/lernen-mit-digitalen-spielen-im-unterricht/. [Zugriff am 27. Januar 2023].

[30] Klei Entertainment Inc., „Oxygen Not Included | Klei Entertainment“, Klei Entertainment Inc., [Online]. Verfügbar unter: https://www.klei.com/games/oxygen-not-included. [Zugriff am 29. Januar 2023].

[31] Klei Entertainment Inc., „Oxygen Not Included bei Steam“, Klei Entertainment Inc., [Online]. Verfügbar unter: https://store.steampowered.com/app/457140/Oxygen_Not_Included/. [Zugriff am 29. Januar 2023].

[32] J. Couture, „Layering challenges in Klei's survival sim Oxygen Not Included“, Informa plc, 07. Juni 2017, [Online]. Verfügbar unter: https://www.gamedeveloper.com/design/layering-challenges-in-klei-s-survival-sim-i-oxygen-not-included-i-. [Zugriff am 29. Januar 2023].

[33] B. Francis, „Behind the design of hit sim game Oxygen Not Included“, Informa plc, 07. Juni 2017, [Online]. Verfügbar unter: https://www.gamedeveloper.com/design/behind-the-design-of-hit-sim-game-i-oxygen-not-included-i-. [Zugriff am 29. Januar 2023].

[34] Squad, „Kerbal Space Program - Create and Manage Your Own Space Program“, Squad, [Online]. Verfügbar unter: https://www.kerbalspaceprogram.com/games-kerbal-space-program. [Zugriff am 28. Januar 2023].

[35] Squad, „Kerbal Space Program bei Steam“, Squad, [Online]. Verfügbar unter: https://store.steampowered.com/app/220200/Kerbal_Space_Program/. [Zugriff am 28. Januar 2023].

[36] S. White, „Minecraft in space: why Nasa is embracing Kerbal Space Program | Games | The Guardian“, Guardian News & Media Ltd., 22. Mai 2014, [Online]. Verfügbar unter: https://www.theguardian.com/technology/2014/may/22/kerbal-space-program-why-nasa-minecraft. [Zugriff am 28. Januar 2023].

[37] CodeParade, „Hyperbolica bei Steam“, CodeParade, [Online]. Verfügbar unter: https://store.steampowered.com/app/1256230/Hyperbolica/. [Zugriff am 22. Januar 2023].

[38] CodeParade, „4D Golf bei Steam“, CodeParade, [Online]. Verfügbar unter: https://store.steampowered.com/app/2147950/4D_Golf/. [Zugriff am 22. Januar 2023].

[39] CodeParade, „CodeParade - YouTube“, [Online]. Verfügbar unter: https://www.youtube.com/@CodeParade/videos. [Zugriff am 22. Januar 2023].

[40] J. Helps, „Primer - YouTube“, [Online]. Verfügbar unter: https://www.youtube.com/@PrimerBlobs/videos. [Zugriff am 22. Januar 2023].

[41] S. Lague, „Coding Adventure: Simulating an Ecosystem - YouTube“, 10. Juni 2019, [Online]. Verfügbar unter: https://www.youtube.com/watch?v=r_It_X7v-1E. [Zugriff am 23. Januar 2023].

[42] Bay 12 Games, „Bay 12 Games: Dwarf Fortress“, Bay 12 Games, 29. Januar 2020, [Online]. Verfügbar unter: http://bay12games.com/dwarves/dev. [Zugriff am 23. Januar 2023].

[43] Museum of Modern Art, „Tarn Adams, Zach Adams. Dwarf Fortress. 2006 | MoMA“, Museum of Modern Art, [Online]. Verfügbar unter: https://www.moma.org/collection/works/164920. [Zugriff am 23. Januar 2023].

[44] J. Parkin, „What is Dwarf Fortress? It’s indescribable - but let me try - Polygon“, Vox Media, Inc., 05. Dezember 2022, [Online]. Verfügbar unter: https://www.polygon.com/guides/23486243/dwarf-fortress-steam-release-explained-simulator-losing-is-fun. [Zugriff am 22. Januar 2023].

[45] Bay 12 Games, „Bay 12 Games: Dwarf Fortress“, Bay 12 Games, [Online]. Verfügbar unter: http://bay12games.com/dwarves/features. [Zugriff am 23. Januar 2023].

[46] Bay 12 Games, „Dwarf Fortress bei Steam“, Bay 12 Games, [Online]. Verfügbar unter: https://store.steampowered.com/app/975370/Dwarf_Fortress/. [Zugriff am 23. Januar 2023].

[47] P. Antonelli und P. Galloway, „When Video Games Came to the Museum | Magazine | MoMA“, Museum of Modern Art, 03. November 2022, [Online]. Verfügbar unter: https://www.moma.org/magazine/articles/798. [Zugriff am 23. Januar 2023].

[48] Ludeon Studios, „RimWorld - Sci-Fi Kolonie-Simulation“, Ludeon Studios, [Online]. Verfügbar unter: https://rimworldgame.com/?lang=de. [Zugriff am 23. januar 2023].

[49] Ludeon Studios, „RimWorld bei Steam“, Ludeon Studios, [Online]. Verfügbar unter: https://store.steampowered.com/app/294100/RimWorld/. [Zugriff am 23. Januar 2023].

[50] C. Marshall, „RimWorld turns disaster and loss into a chaotic but hilarious party - Polygon“, Vox Media, Inc., 11. Juli 2022, [Online]. Verfügbar unter: https://www.polygon.com/23199105/rimworld-colony-sim-sci-fi-adventure. [Zugriff am 22. Januar 2023].

[51] Informa plc, „How RimWorld fleshes out the Dwarf Fortress formula“, Informa plc, 11. August 2016, [Online]. Verfügbar unter: https://www.gamedeveloper.com/design/how-i-rimworld-i-fleshes-out-the-i-dwarf-fortress-i-formula. [Zugriff am 22. Januar 2023].

[52] J. Couture, „Q&A: The appeal of building a natural world in Equilinox“, Informa plc, 26. Dezember 2018, [Online]. Verfügbar unter: https://www.gamedeveloper.com/design/q-a-the-appeal-of-building-a-natural-world-in-i-equilinox-i-. [Zugriff am 23. Januar 2023].

[53] ThinMatrix, „Equilinox“, ThinMatrix, [Online]. Verfügbar unter: https://equilinox.com/presskit/. [Zugriff am 23. Januar 2023].

[54] ThinMatrix, „Equilinox bei Steam“, ThinMatrix, [Online]. Verfügbar unter: https://store.steampowered.com/app/853550/Equilinox/. [Zugriff am 23. januar 2023].

[55] ThinMatrix, „Equilinox“, ThinMatrix, [Online]. Verfügbar unter: https://equilinox.com/. [Zugriff am 23. Januar 2023].

[56] S. Eckstein, Informationsmanagement in der Systembiologie: Datenbanken, Integration, Modellierung, Berlin/Heidelberg: Springer-Verlag, 2011.

[57] A. Kremling, Kompendium Systembiologie: Mathematische Modellierung und Modellanalyse, Wiesbaden: Vieweg+Teubner Verlag/Springer Fachmedien Wiesbaden GmbH, 2012.

[58] A. Reinhart, Statistics Done Wrong: The Woefully Complete Guide, San Francisco: No Starch Press, Inc., 2015.

[59] S. Axon, „Unity at 10: For better -or worse- game development has never been easier | Ars Technica.pdf“, Condé Nast Digital, 27. September 2016, [Online]. Verfügbar unter: https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/. [Zugriff am 17. Januar 2023].

[60] M. Dealessandri, „What is the best game engine: is Unity right for you? | GamesIndustry.biz“, Gamer Network Limited, 16. Januar 2020, [Online]. Verfügbar unter: https://www.gamesindustry.biz/what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you. [Zugriff am 17. Januar 2023].

[61] Unity Software Inc., „What platforms are supported by Unity? - Unity“, Unity Software Inc., 27. Juni 2022, [Online]. Verfügbar unter: https://support.unity.com/hc/en-us/articles/206336795-What-platforms-are-supported-by-Unity-. [Zugriff am 17. Januar 2023].

[62] Unity Software Inc., „Unity - Manual: System requirements for Unity 2021 LTS“, Unity Software Inc., [Online]. Verfügbar unter: https://docs.unity3d.com/Manual/system-requirements. [Zugriff am 17. Januar 2023].

[63] Unity Software Inc., „Sie fragen sich was Unity ist? Entdecken Sie, wer wir sind, wo wir angefangen haben und wohin wir uns entwickeln | Unity“, Unity Software Inc., [Online]. Verfügbar unter: https://unity.com/de/our-company. [Zugriff am 28. Januar 2023].

[64] Unity Software Inc., „Unity Gaming Report 2022“, Unity Software Inc., [Online]. Verfügbar unter: https://create.unity.com/gaming-report-2022. [Zugriff am 28. Januar 2023].

[65] Unity Software Inc., „Made with Unity | Unity“, Unity Software Inc., [Online]. Verfügbar unter: https://unity.com/de/madewith. [Zugriff am 28. Januar 2023].

[66] S. Edelstein, „Unity Automotive Uses Tech From Gaming to Aid Automakers | Digital Trends“, Digital Trends Media Group, 17. Mai 2018, [Online]. Verfügbar unter: https://www.digitaltrends.com/cars/unity-automotive-virtual-reality-and-hmi/. [Zugriff am 17. Januar 2023].

[67] N. Bonfiglio, „DeepMind and Unity Will Collaborate on Artificial Intelligence Research“, The Daily Dot, 01. Oktober 2018, [Online]. Verfügbar unter: https://www.dailydot.com/debug/unity-deempind-ai/. [Zugriff am 17. Januar 2023].

[68] A. Liptak, „How Neill Blomkamp and Unity are shaping the future of filmmaking - The Verge“, Vox Media, 30. November 2017, [Online]. Verfügbar unter: https://www.theverge.com/2017/10/4/16409734/unity-neill-blomkamp-oats-studios-mirror-cinemachine-short-film. [Zugriff am 17. Januar 2023].

[69] Unity Software Inc., „2021 LTS Version mit Langzeit-Support im Überblick | Unity“, Unity Software Inc., [Online]. Verfügbar unter: https://unity.com/de/releases/2021-lts. [Zugriff am 04. Februar 2023].

[70] D. Tutino-Galletti, „Project Management is evolving! Unity Package Manager Overview | Unity Blog“, Unity Software Inc., 04. Mai 2018, [Online]. Verfügbar unter: https://blog.unity.com/technology/project-management-is-evolving-unity-package-manager-overview. [Zugriff am 17. Januar 2023].

[71] Unity Software Inc., „Unity - Manual: Unity Services“, Unity Software Inc., [Online]. Verfügbar unter: https://docs.unity3d.com/Manual/UnityServices. [Zugriff am 28. Januar 2023].

[72] T. Krogh-Jacobsen und S. McGreal, „Profiling in Unity 2021 LTS: What, when, and how | Unity Blog“, Unity Software Inc., 01. Juni 2022, [Online]. Verfügbar unter: https://blog.unity.com/technology/profiling-in-unity-2021-lts-what-when-and-how. [Zugriff am 17. Januar 2023].

[73] M. Nakamura, „Games Focus: Profiling and performance optimization | Unity Blog“, Unity Software Inc., 28. September 2022, [Online]. Verfügbar unter: https://blog.unity.com/technology/games-focus-profiling-and-performance-optimization. [Zugriff am 19. Januar 2023].

[74] Unity Software Inc., „DOTS - Unity’s Data-Oriented Technology Stack“, Unity Software Inc., [Online]. Verfügbar unter: https://unity.com/dots. [Zugriff am 18. Januar 2023].

[75] I. Seah, „Games Focus: Expanded scale for ambitious games | Unity Blog“, Unity Software Inc., 23. November 2022, [Online]. Verfügbar unter: https://blog.unity.com/technology/games-focus-expanded-scale-for-ambitious-games. [Zugriff am 18. Januer 2023].

[76] L. Gibert, „Official - DOTS Development Status And Next Milestones - March 2022 - Unity Forum“, Unity Software Inc., 15. März 2022, [Online]. Verfügbar unter: https://forum.unity.com/threads/dots-development-status-and-next-milestones-march-2022.1253355/. [Zugriff am 06. Februar 2023].

[77] T. Johansson, „What is a Job System? | Unity Blog“, Unity Software Inc., 22. Oktober 2018, [Online]. Verfügbar unter: https://blog.unity.com/technology/what-is-a-job-system. [Zugriff am 18. Januar 2023].

[78] L. Meijer, „On DOTS: C++ & C# | Unity Blog“, Unity Software Inc., 26. Februar 2019, [Online]. Verfügbar unter: https://blog.unity.com/technology/on-dots-c-c. [Zugriff am 18. Januar 2023].

[79] N. Henning, „In parameters in Burst | Unity Blog“, Unity Software Inc., 25. November 2020, [Online]. Verfügbar unter: https://blog.unity.com/technology/in-parameters-in-burst. [Zugriff am 18. Januar 2023].

[80] J. A und Y. Habets, „Enhancing mobile performance with the Burst compiler | Unity Blog“, Unity Software Inc., 17. August 2020, [Online]. Verfügbar unter: https://blog.unity.com/technology/enhancing-mobile-performance-with-the-burst-compiler. [Zugriff am 18. Januar 2023].

[81] N. Henning, „Enhanced Aliasing with Burst | Unity Blog“, Unity Software Inc., 07. September 2020, [Online]. Verfügbar unter: https://blog.unity.com/technology/enhanced-aliasing-with-burst. [Zugriff am 18. Januar 2023].

[82] T. Jones, „Raising your game with Burst 1.7 | Unity Blog“, Unity Software Inc., 14. März 2022, [Online]. Verfügbar unter: https://blog.unity.com/technology/raising-your-game-with-burst-17. [Zugriff am 18. Januar 2023].

[83] L. Meijer, „On DOTS: Entity Component System | Unity Blog“, Unity Software Inc., 08. März 2019, [Online]. Verfügbar unter: https://blog.unity.com/technology/on-dots-entity-component-system. [Zugriff am 18. Januar 2023].

[84] Unity Software Inc., „ECS for Unity“, Unity Software Inc., [Online]. Verfügbar unter: https://unity.com/ecs. [Zugriff am 18. Januar 2023].

[85] K. Hougaard, „Creating a third-person zombie shooter with DOTS | Unity Blog“, Unity Software Inc., 27. November 2019, [Online]. Verfügbar unter: https://blog.unity.com/technology/creating-a-third-person-zombie-shooter-with-dots. [Zugriff am 18. Januar 2023].

[86] L. Gibert, „Official - DOTS Development Status And Next Milestones - December 2021 - Unity Forum“, Unity Software Inc., 09. Dezember 2021, [Online]. Verfügbar unter: https://forum.unity.com/threads/dots-development-status-and-next-milestones-december-2021.1209727/. [Zugriff am 06. Februar 2023].

[87] Intel Corporation, „Was führt bei meinem PC zu Performance-Engpässen? - Intel“, Intel Corporation, [Online]. Verfügbar unter: https://www.intel.de/content/www/de/de/gaming/resources/what-is-bottlenecking-my-pc. [Zugriff am 21. Januar 2023].

[88] E. McClure, „Multithreading Problems In Game Design“, 23. Mai 2012, [Online]. Verfügbar unter: https://erikmcclure.com/blog/multithreading-problems-in-game-design/. [Zugriff am 21. Januar 2023].

[89] L. Gibert, „Official - DOTS development status and next milestones – September 2022 - Unity Forum“, Unity Software Inc., 26. September 2022, [Online]. Verfügbar unter: https://forum.unity.com/threads/dots-development-status-and-next-milestones-september-2022.1341077/. [Zugriff am 07. Februar 2023].

[90] mfuad, „Official - Experimental Entities 0.50 is Available - Unity Forum“, Unity Software Inc., 16. März 2022, [Online]. Verfügbar unter: https://forum.unity.com/threads/experimental-entities-0-50-is-available.1253394/. [Zugriff am 06. Februar 2023].

[91] mfuad, „Official - Experimental Entities 1.0 is available - Unity Forum“, Unity Software Inc., 26. September 2022, [Online]. Verfügbar unter: https://forum.unity.com/threads/experimental-entities-1-0-is-available.1341065/. [Zugriff am 07. Februar 2023].

[92] raymondyunity, „DOTS Editor 0.1.0-preview Release - Unity Forum“, Unity Software Inc., 28. November 2019, [Online]. Verfügbar unter: https://forum.unity.com/threads/dots-editor-0-1-0-preview-release.785084/. [Zugriff am 06. Februar 2023].

[93] B. Will, „Official - Entities 0.17 changelog - Unity Forum“, Unity Software Inc., 10. Dezember 2020, [Online]. Verfügbar unter: https://forum.unity.com/threads/entities-0-17-changelog.1020202/. [Zugriff am 06. Februar 2023].

[94] J. Valenzuela, „Official - Exciting developments in upcoming Entities 1.0 releases - Unity Forum“, Unity Software Inc., 26. September 2022, [Online]. Verfügbar unter: https://forum.unity.com/threads/exciting-developments-in-upcoming-entities-1-0-releases.1341071/. [Zugriff am 07. Februar 2023].

[95] Unity Software Inc., „DOTS roadmap | Unity“, Unity Software Inc., [Online]. Verfügbar unter: https://unity.com/roadmap/unity-platform/dots. [Zugriff am 19. Januar 2023].

[96] Unity Software Inc., „Entities overview | Entities | 1.0.0-pre.15“, Unity Software Inc., 23. November 2022, [Online]. Verfügbar unter: https://docs.unity3d.com/Packages/com.unity.entities@1.0/manual/index.html. [Zugriff am 04. Februar 2023].

[97] Unity Software Inc., „Was ist Versionskontrolle und wie funktioniert sie? | Unity“, Unity Software Inc., [Online]. Verfügbar unter: https://unity.com/de/solutions/what-is-version-control. [Zugriff am 04. Februar 2023].

[98] Atlassian Corporation, „Was ist Versionskontrolle? | Atlassian Git Tutorial“, Atlassian Corporation, [Online]. Verfügbar unter: https://www.atlassian.com/de/git/tutorials/what-is-version-control. [Zugriff am 04. Februar 2023].