Technische Hochschule Nürnberg Georg Simon Ohm
Fakultät Elektrotechnik Feinwerktechnik Informationstechnik

Studiengang Media Engineering

Bachelor-Arbeit von
Elena Moumoulidis
Matrikelnummer: 3535998

„Simulation eines Ökosystems in Unity“

SS 2023

Technische Hochschule Nürnberg Georg Simon Ohm
Fakultät Elektrotechnik Feinwerktechnik Informationstechnik

Studiengang Media Engineering

Bachelor-Arbeit von
Elena Moumoulidis
Matrikelnummer: 3535998

„Simulation eines Ökosystems in Unity“

SS 2023
Abgabedatum: 28.07.2023

Betreuer:
Prof. Dr. rer. nat. Matthias Hopf

ERKLÄRUNG

Hinweis: Diese Erklärung ist in alle Exemplare der Abschlussarbeit fest einzubinden. (Keine Spiralbindung)

TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Prüfungsrechtliche Erklärung der/des Studierenden

Angaben des bzw. der Studierenden:

Name: Elena | Vorname: Moumoulidis | Matrikel-Nr.: 3535998

Fakultät: EFI | Studiengang: Media Engineering

Semester: SS 2023

Titel der Abschlussarbeit:

Simulation eines Ökosystems in Unity

Ich versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Neustadt/Aisch, 28.07.2023 Elli Moum
Ort, Datum, Unterschrift Studierende/Studierender

Erklärung der/des Studierenden zur Veröffentlichung der vorstehend bezeichneten Abschlussarbeit

Die Entscheidung über die vollständige oder auszugsweise Veröffentlichung der Abschlussarbeit liegt grundsätzlich erst einmal allein in der Zuständigkeit der/des studentischen Verfasserin/Verfassers. Nach dem Urheberrechtsgesetz (UrhG) erwirbt die Verfasserin/der Verfasser einer Abschlussarbeit mit Anfertigung ihrer/seiner Arbeit das alleinige Urheberrecht und grundsätzlich auch die hieraus resultierenden Nutzungsrechte wie z.B. Erstveröffentlichung (§ 12 UrhG), Verbreitung (§ 17 UrhG), Vervielfältigung (§ 16 UrhG), Online-Nutzung usw., also alle Rechte, die die nicht-kommerzielle oder kommerzielle Verwertung betreffen.

Die Hochschule und deren Beschäftigte werden Abschlussarbeiten oder Teile davon nicht ohne Zustimmung der/des studentischen Verfasserin/Verfassers veröffentlichen, insbesondere nicht öffentlich zugänglich in die Bibliothek der Hochschule einstellen.

Hiermit ☑ genehmige ich, wenn und soweit keine entgegenstehenden Vereinbarungen mit Dritten getroffen worden sind,

☐ genehmige ich nicht,

dass die oben genannte Abschlussarbeit durch die Technische Hochschule Nürnberg Georg Simon Ohm, ggf. nach Ablauf einer mittels eines auf der Abschlussarbeit aufgebrachten Sperrvermerks kenntlich gemachten Sperrfrist

von _____ Jahren (0 - 5 Jahren ab Datum der Abgabe der Arbeit),

der Öffentlichkeit zugänglich gemacht wird. Im Falle der Genehmigung erfolgt diese unwiderruflich; hierzu wird der Abschlussarbeit ein Exemplar im digitalisierten PDF-Format auf einem Datenträger beigefügt. Bestimmungen der jeweils geltenden Studien- und Prüfungsordnung über Art und Umfang der im Rahmen der Arbeit abzugebenden Exemplare und Materialien werden hierdurch nicht berührt.

Neustadt/Aisch, 28.07.2023 Elli Moum
Ort, Datum, Unterschrift Studierende/Studierender

Datenschutz: Die Antragstellung ist regelmäßig mit der Speicherung und Verarbeitung der von Ihnen mitgeteilten Daten durch die Technische Hochschule Nürnberg Georg Simon Ohm verbunden. Weitere Informationen zum Umgang der Technischen Hochschule Nürnberg mit Ihren personenbezogenen Daten sind unter nachfolgendem Link abrufbar: **https://www.th-nuernberg.de/datenschutz/**

DokID: SB_0050_FO_Erklärung_zur_Veröffentlichung_der_Abschlussarbeit_prüfungsrechtliche Erklärung_public Vers. 2, 22.09.2017, A. Härtel (4.SB)

ABSTRACT

In dieser Bachelorarbeit mit dem Titel „Simulation eines Ökosystems in Unity“ soll ein Bewusstsein für die Problematik der Ausbreitung der einst eingeführten Possums in Neuseeland und der damit einhergehenden starken Bedrohung für die einheimischen Kiwis durch die Erstellung eines entsprechenden Ökosystems in Unity geschaffen werden. Nach einem theoretischen Einblick in die Thematik und der Betrachtung relevanter Aspekte eines Ökosystems wird genauer auf die Entwicklung der Simulation, welche sowohl die Programmierung und Umsetzung in Unity als auch das Design, 3D-Modelling, die Animation und das Erstellen der Musik beinhaltet, eingegangen. Ziel war es hierbei nicht, die Simulation realitätsgetreu darzustellen, sondern die Problematik auf eine anschauliche Art und Weise aufzuzeigen. Durch die erstellte Simulation ist die Beobachtung des Geschehens möglich und die Überlegenheit der Possums aufgrund fehlender Fressfeinde sichtbar. Die starke Bedrohung für den Kiwi wird dadurch deutlich gemacht.

ABSTRACT

In this bachelor thesis entitled "Simulation of an ecosystem in Unity", an awareness of the problem of the spread of the once introduced possums in New Zealand and the associated severe threat to the native kiwis should be raised, by creating an appropriate ecosystem in Unity. After a theoretical insight into the topic and a closer look at relevant aspects of an ecosystem, the development of the simulation, which includes programming and the implementation in Unity as well as the design, 3D modeling, the animation and creating the music, is presented. The aim was not to create the simulation in a realistic way, but to illustrate the issue. Based on the created simulation, it is possible to observe the process and point out the superiority of the possums due to the lack of predators. The serious threat to the kiwi is thus made clear.

DANKSAGUNG

An dieser Stelle möchte ich meinen herzlichen Dank an alle Personen aussprechen, die mich bei der Erstellung dieser Bachelorarbeit unterstützt haben. Besonders bei den hier genannten Personen:

Zuerst möchte ich mich bei meinem Betreuer Prof. Dr. Matthias Hopf, der sich stets Zeit für meine Fragen nahm, für die hilfreichen Hinweise und Tipps, sowie das gemeinsame Bugfixing bedanken.

Vielen lieben Dank auch an meinen Kommilitonen Luca Geiger, der eine große Hilfe und Unterstützung bei Ideenfindung, Ausarbeitung der Logik und dem Bugfixing war.

Im gleichen Zuge möchte ich mich herzlich bei meinem Kommilitonen Erik Schneider bedanken, der ebenfalls einen beachtlichen Beitrag bei der Fehlerbehebung und Umsetzung der Logik leistete.

Ein herzliches Dankeschön geht auch an meine Kommilitonin Paulina Stengel, welche bei der Ideenfindung unterstützte.

INHALTSVERZEICHNIS

1 EINLEITUNG

Seit geraumer Zeit werden Ökosysteme aus dem Gleichgewicht gebracht – oft durch den Eingriff des Menschen in die Natur. Vor allem der Import von Tierarten in ein Gebiet, in dem die eingeführte Tierart vorher nicht existierte, verursachte des Öfteren fatale Folgen für das Ökosystem des Landes und bedeutete auch nicht selten eine große Bedrohung für manche heimischen Tierarten. Zahlreiche Beispiele, wie das Einführen der Aga-Kröte nach Australien, die gegen Schädlinge wirksam werden sollte [1] oder der Import von Bibern ins Feuerland, welche sich massiv ausbreiteten und der Natur ungemeinen Schaden zufügten [2], demonstrieren diese Problematik.

In dieser Bachelorarbeit wurde sich mit der enormen Ausbreitung der Possums in Neuseeland beschäftigt, welche einst aus Gründen des Pelzhandels [3, S. 20] ins Land gebracht wurden und sich aufgrund des Mangels an Feinden dort explosiv vermehrten [4]. Besonders zu Schaden kommt dabei die heimische Vogelbevölkerung Neuseelands [5], die zur Beute des Possums wurde. Unter ihnen ist auch Neuseelands Nationaltier, der Kiwi [5], betroffen, dessen Populationsdichte von einer Anzahl von rund 12 Millionen Kiwis bereits auf nur noch ca. 65.000 Kiwis geschwunden ist [6].

Aufgrund dieser misslichen Situation, wurde als Thema dieser Bachelorarbeit eine Simulation in der Game Engine Unity ausgearbeitet, die das Ökosystem Neuseelands darstellen und das genannte Dilemma demonstrieren soll. Ziel ist es dabei, ein Bewusstsein für die enorme Problematik, mit der Neuseeland nach wie vor zu kämpfen hat, zu schaffen, indem die massive Ausbreitung des Possums und die damit verbundene starke Bedrohung für den Kiwi [5] anhand der Simulation sichtbar gemacht wird. Der Nutzer hat dabei die Möglichkeit, bestimmte Anzahlen beider Tierarten auf die Karte zu setzen und zu beobachten, was passiert. Es geht nicht um die Darstellung realistischer Werte und Gegebenheiten, sondern lediglich um das Aufzeigen der Problematik auf eine spielerische Art und Weise. In der folgenden Ausarbeitung wird in einem theoretischen Teil zuerst Hintergrundwissen über Ökosysteme und biologische Invasionen vermittelt und anschließend ein tieferer Einblick zu Neuseelands Possum-Problematik gegeben. Auch ein Vergleich mit bereits bestehenden Simulationen von Ökosystemen in Unity, wird gezogen. Im Anschluss wird Genaueres über die Game Engine Unity erklärt, um einen Eindruck über die verwendeten Mittel zur Ausarbeitung der Simulation zu erhalten, bevor die eigentliche Umsetzung der Simulation des Ökosystems erklärt wird. Abschließend werden diverse Diagrammtests aufgeführt und die Ergebnisse eines Nutzertests offengelegt. Im Schlussteil ist eine Zusammenfassung der Arbeit zu finden und mögliche Weiterentwicklungen werden im Abschnitt „Ausblick“ genannt.

2 THEORIE

Um einen intensiveren Einblick in die Thematik dieser Bachelorarbeit zu erhalten, wurde genauer recherchiert und die Ergebnisse in einem theoretischen Teil niedergeschrieben. Im Folgenden werden informationstechnisch relevante Aspekte zum Thema Ökosystem und invasive Tierarten genauer beleuchtet. Zunächst wird über die Problematik der massiven Verbreitung des Possums in Neuseeland informiert und anschließend ein paar bereits erstellte Simulationen von Ökosystemen in Unity zum Vergleich dargelegt.

2.1 THEORETISCHER HINTERGRUND

In diesem Teil der Arbeit werden Hintergrundinformationen zum Thema vermittelt, indem erklärt wird, was man unter einem Ökosystem versteht, was ein Schweinezyklus ist und es wird Wissenswertes über das ökologische Gleichgewicht herausgearbeitet. Zunächst werden biologische Invasionen beschrieben und einige Beispiele hierzu angeführt.

2.1.1 DEFINITION ÖKOSYSTEM

Bei einem Ökosystem handelt es sich um „ein Funktionsgefüge aus den Populationen von Organismen verschiedener Arten sowie den von ihnen genutzten Bedingungen und Gegenständen ihrer Umwelt“ [7, S. 715]. Ein Ökosystem besteht aus „mindestens […] [einer] Population von primären Produzenten (z. B. Pflanzen) und Reduzenten (z. B. Bakterien), dazwischengeschaltet sind meist Populationen von Konsumenten (z. B. Tiere)“ [7, S. 715]. Zudem kann ein Ökosystem auch als Zusammenspiel von Biotop (Lebensraum) und Biozönose (Lebewesen, die das Biotop bewohnen) bezeichnet werden [8].
Die Aufgabe der Produzenten ist die Erzeugung einer hohen Biomasse [8], während die Reduzenten [7, S. 715], welche sich auch Destruenten [8] nennen, den „Abbau und […] [die] Remineralisierung von abgestorbenem, organischem Material“ [8] zu ihrer Aufgabe machen. Die Konsumenten können in verschiedene Typen unterteilt werden. Es gibt z. B. Herbivoren (Pflanzenfresser), Carnivoren (Fleischfresser) und Omnivoren (Allesfresser). [8], [9]
Zudem wird unterschieden zwischen terrestrischen Ökosystemen, zu welchen Ökosysteme an Land zählen, die nicht generell aus Wasser bestehen und aquatischen Ökosystemen, die Ökosysteme von Gewässern umfassen. Ein wesentliches Merkmal von Ökosystemen ist, dass sie dynamisch sind, was bedeutet, dass sie „sich durch Einflüsse von Innen und Außen verändern“[8]. [8]
Relevante Dynamiken werden im folgenden Abschnitt genauer erklärt.

2.1.2 DER SCHWEINEZYKLUS

Als Schweinezyklus wird „eine periodische Schwankung der Angebotsmenge und des Marktpreises“ [10] bezeichnet. Es wird sich an Angebot- und Nachfrage des Marktes orientiert, allerdings ist das Anpassen der Produktion an die Nachfrage, aufgrund von längerer Vorlaufzeit nicht sofort möglich

[10]. Der Schweinezyklus erhielt seinen Namen durch den Vergleich dieses Szenarios anhand von Schweinemärkten, bei denen nach Feststellen des Preisanstiegs für Schweinefleisch, neue Ferkel gekauft oder gezüchtet wurden, um Profit aus der gestiegenen Nachfrage zu erlangen. Bis die Ferkel ausgewachsen sind und ein Verkauf möglich ist, dauert es jedoch. Da es nun ein höheres Angebot an Schweinefleisch gibt, geht der Preis wieder zurück. Demzufolge werden nun weniger Schweine gezüchtet und der Schweinezyklus beginnt von vorne. [11]

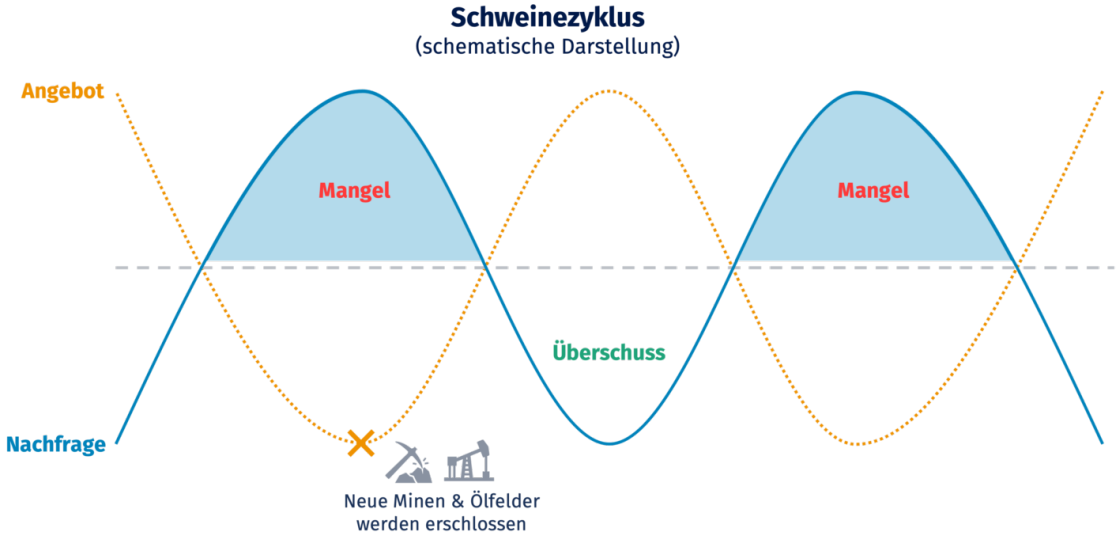

Abbildung 1: Schweinezyklus aus [11]

So ergibt sich ein Wechselspiel aus Angebot und Nachfrage, welches anhand von Abbildung 1, die aus folgender Quelle: [11] stammt, veranschaulicht wird.
Bezogen auf Ökosysteme spiegelt sich eine Angebots- und Nachfragekurve anhand von verfügbaren Nahrungsressourcen und der damit verbundenen Ab- bzw. Zunahme von Populationen (Konsumenten) wider. Ein zu erwartender Schweinezyklus wäre hier z. B. der Anstieg der Population bei einer ausreichenden Menge an Nahrungsressourcen, da es bei der Deckung überlebenswichtiger Bedürfnisse häufiger zur Fortpflanzung kommen kann und zu einer besseren Überlebenschance für den Nachwuchs beiträgt. Infolgedessen kann bei mangelnder Nahrung ein Rückgang der Population erwartet werden.

2.1.3 DAS ÖKOLOGISCHE GLEICHGEWICHT

Von einem ökologischen Gleichgewicht kann gesprochen werden, wenn die Anzahl von Tieren- und Pflanzenarten annähernd konstant um einen bestimmten Mittelwert schwankt [12]. Kommt es innerhalb eines längeren Zeitraums zu einseitigen Vorgängen, so beginnt eine Sukzession [13], welche „den schrittweisen Wiederaufbau der Lebewesen eines geschädigten Ökosystems“ [14] meint.
Mögliche Störungen des ökologischen Gleichgewichts können z. B. Wettereinflüsse, wie Hitzeperioden sein, die beispielsweise die Überbevölkerung des Borkenkäfers begünstigen [12] oder, wie in der Thematik dieser Bachelorarbeit, die starke Zunahme der Possum-Population in

Neuseeland. Laut der Meinung einiger Forscher:innen sei der Begriff „Ökologisches Gleichgewicht“ jedoch kritisch zu betrachten, da „Vorgänge in der Natur nicht immer sicher voraussagbar sind“ [12] und aus einer Störung auch ein neues, funktionierendes Gleichgewicht entstehen kann [12]. Der Import von fremden Tierarten in ein bestehendes Ökosystem kann jedoch das Gleichgewicht des Ökosystems aus dem Konzept bringen, wie an den im folgenden aufgeführten Beispielen zu sehen ist.

2.1.4 BIOLOGISCHE INVASIONEN

Biologische Invasionen stellen mitunter eine der massivsten Gefährdungen für die biologische Vielfalt dar [15, S. 394]. Durch die enorme Ausbreitung der ins Land gelangten Tierarten können diese die heimischen Arten des Gebiets unterdrücken und „Ökosysteme tiefgreifend [sic] umgestalten“ [15, S. 394]. [15, S. 394] Welche Auswirkungen das Importieren von fremden Tierarten in ein bestehendes Ökosystem haben kann, wird anhand folgender problematischer Ereignisse deutlich gemacht.

DIE AUSBREITUNG DER AGA-KRÖTE IN AUSTRALIEN

Um Schädlinge auf den Zuckerrohrfeldern Australiens zu bekämpfen, wurden im Jahre 1935 ca. 100 Aga-Kröten, die ihren natürlichen Lebensraum in Süd- und Mittelamerika haben, eingeführt. Durch die massive Fortpflanzung der Kröten und ihre rasche Bewegungsgeschwindigkeit aufgrund ihrer langen Beine, haben sich diese jedoch explosiv innerhalb Australiens ausgebreitet und besiedeln mittlerweile fast den gesamten Bereich der Ostküste Australiens. Zu Schaden kamen dabei einige Tierarten des Kontinents, die z. B. versuchten die Aga-Kröte zu fressen und an ihrem Gift verendeten. Andere Tiere wurden zur Beute der Kröte oder ein Nahrungskonkurrent. Diese Problematiken führten zu einer enormen Veränderung der Ökosysteme, in denen sich die Kröten befanden. [1]

DIE KANINCHENPLAGE AUSTRALIENS

Die im Jahre 1859 aus England [16] nach Australien eingeführten Kaninchen „fanden in Australien ideale Lebensbedingungen für ihre Art und wurden schon nach wenigen Jahrzehnten zu einer Plage“ [17, S. 20]. „In nur 50 Jahren hatten die Tiere ein Gebiet kolonisiert, das etwa 13-mal größer ist als ihr ursprüngliches europäisches Verbreitungsgebiet“ [16]. Sie richten großen Schaden, welcher „jährlich 200 Millionen Dollar an landwirtschaftlichen Schäden“ [16] umfasst, an den Pflanzen und Feldern Australiens an. Die Landschaft Australiens leidet noch immer unter den negativen Folgen der Kaninchenausbreitung. Eine mögliche Eindämmung der Plage durch Einzäunung oder Viren, die spezifisch zur Ausrottung des Kaninchens entwickelt wurden, blieben erfolglos. [16]

DIE BIBERPLAGE IM FEUERLAND

Für eine Pelzzucht wurden im Jahre 1946 25 Biberpaare aus Kanada nach Argentinien importiert. Durch den Mangel an Feinden in diesem Gebiet vermehrten sich die Biber bis zu einer Anzahl von ca. 100.000 Bibern. Da sie Bäume fällen und Staudämme bauen, beeinflussen sie die Landschaft und das Ökosystem enorm. „Sie haben auf diese Weise schon Flussläufe umgeleitet und regionale Ökosysteme verändert“ [2]. [2] Eine Folge ist „das Absterben der Südbuchenwälder in überstauten

Bereichen der Insel Navarino“ [15, S. 395]. Infolgedessen ist die Lebensgrundlage einiger Tierarten, die abhängig sind von schnell fließenden Binnengewässern, gestört [15, S. 395].

DIE AUSBREITUNG DES POSSUMS IN NEUSEELAND

Des Weiteren ist die Überbevölkerung des Possums in Neuseeland zu nennen, die mit einer Anzahl von ca. 70 Millionen Tieren [3, S. 20], eine große Bedrohung für die Vögel Neuseelands, unter anderem den Kiwi, darstellt [5]. Da sich diese Bachelorarbeit auf jene Thematik bezieht, wird dieser Fall im folgenden Untergliederungspunkt genauer behandelt.

2.2 DIE POSSUMPROBLEMATIK NEUSEELANDS

Eine große Problematik in Neuseeland stellen unter anderem die Possums dar, welche immensen Schaden an der Natur Neuseelands anrichten und den Vogelarten erheblich zusetzen [5]. Sie zählen „zu den schlimmsten Ökoproblemen“ [3, S. 20] Neuseelands.

2.2.1 ENTSTEHUNG DER ENORMEN POSSUMVERMEHRUNG

Im Jahre 1837 wurden Possums aus Australien nach Neuseeland importiert, um sie für den Pelzhandel zu nutzen. Solange sich die Felle des Possums gut verkaufen ließen, kam es zu keinem ökologischen Ungleichgewicht. Durch Anti-Pelz-Kampagnen schwand die Nachfrage jedoch und die Population der Possums stieg massiv an. [3, S. 20] Im Jahre 1980 bevölkerten die Possums rund 91% Neuseelands. Durch ihre vielseitigen Nahrungspräferenzen und die Möglichkeit mit einer geringeren Menge an Futter auszukommen, ist es ihnen auch möglich, in Gebieten mit wenig Nahrungsquellen zu überleben. [18, S. 1 f.] In ihrem Heimatland Australien, in dem sie sogar eine geschützte Tierart sind, haben sie natürliche Feinde, wie die Dingos, die eine Überbevölkerung des Possums verhindern. In Neuseeland gibt es jedoch kaum Feinde für das Possum und jede Menge Unterschlupf. Die zusätzlich ausreichende Nahrung gibt dem Possum eine gute Möglichkeit in Neuseeland zu überleben. [19]

2.2.2 SCHÄDEN DURCH DAS POSSUM

Durch das immense Wachstum der Possum-Population kam es zu Schäden an der Flora und Fauna Neuseelands. Eine große Problematik stellt die hohe Menge an Pflanzen dar, welche durch die Possums als Futterquelle vernichtet werden [3, S. 20]. Sie „haben dadurch ganze Landstriche verändert.“ [3, S. 20]. Infolgedessen werden sie zu Nahrungskonkurrenten mancher Vogelarten Neuseelands [18, S. 3]. Zum anderen setzen sie den Bäumen Neuseelands zu, indem sie die Baumkronen beschädigen. „Durch die Possums besonders gefährdet sind breitblättrige Bäume wie Rata“ [4]. [4] Auch für die Farmer stellt das Possum ein Problem dar, da Possums der Träger der Krankheit Tuberkulose sein können und somit der Auslöser einer Infektion bei Kühen und Hirschen sein können [19]. Das in dieser Bachelorarbeit relevanteste Problem, stellt jedoch die Bedrohung der

Possums für die Vogelarten Neuseelands, wie den Kiwi dar, welche im folgenden Abschnitt genauer beleuchtet wird.

2.2.3 PROBLEMATIK DER POSSUMAUSBREITUNG FÜR DEN KIWI

Dadurch, dass Vogelarten wie der Kiwi zuvor kaum in Kontakt mit Säugetieren wie dem Possum kamen, war ihnen nicht bewusst, wie sie sich am besten vor ihnen schützen konnten. Der Schutzmechanismus vieler neuseeländischer Tierarten bestand im Stillstehen und zeigte bisher bei einheimischen Feinden gute Erfolge. Beim Possum allerdings, war diese Strategie eine schlechte Wahl. [5] Hinzu kommt, dass Kiwis sowohl bezüglich des Lebensraums als auch der Nahrungsquellen in Konkurrenz mit dem Possum stehen. Eine ebenfalls große Problematik ist das Verzehren von Kiwi-Eiern durch die Possums. [3, S. 20] Das Fressen von Eiern und Jungen neuseeländischer Vögel konnte mithilfe einer Infrarotkamera beobachtet werden [18, S. 3]. Infolge genannter Gründe stellt das Possum eine große Gefahr für den Kiwi dar und begründet mitunter den Rückgang der Kiwi-Population [4].

2.2.4 NEUSEELAND VORHER

Vor der Ankunft des Menschen in Neuseeland galt das Land als Vogelland. „Es gab nur einige wenige Arten von Fledermäusen und Meeressäugetieren und alles andere waren Vögel, Reptilien und Insekten“ [6]. Schätzungen zu Folge soll es damals rund 12 Millionen Kiwis gegeben haben. Die Anzahl der Kiwis heute hat sich jedoch auf 65.000 reduziert. [6] Schuld an dieser Problematik tragen die Menschen, die Säugetiere mit nach Neuseeland brachten [20].

2.2.5 VERSUCHE ZUR POSSUMBEKÄMPFUNG

Durch die vorherigen Darlegungen ist es möglich, ein Verständnis zu entwickeln, warum die Reduktion der Possums in Neuseeland von großer Bedeutung ist. Versuche zur Bekämpfung der Possums wurden z. B. vom DOC getätigt, welches mit Fallen und Giftködern versucht, das Possum zu beseitigen [3, S. 20]. Mit dem DOC, ausgeschrieben „Department of Conservation“ ist „die zentrale staatliche Organisation Neuseelands, die sich im Auftrag von und zum Wohle der jetzigen und zukünftigen Einwohner mit dem Schutz der Natur und dem historischen Erbe des Landes beschäftigt“ [21] gemeint. Eine weitere Methode, die vollzogen wird, ist das Überfahren und Erschießen der Possums [3, S. 20]. Das DOC hat sich zum Ziel gesetzt, jegliche Beuteltiere in Neuseeland auszurotten und dafür einen Plan entwickelt, der die Beuteltiere bis 2050 vernichtet haben soll. In Zukunft sollen Fallen mit künstlicher Intelligenz, die in der Lage sind, ein sich in der Nähe befindendes Possum zu detektieren, vom DOC eingesetzt werden. Außerdem gäbe es die Möglichkeit, in das Erbgut der Tiere einzugreifen, um die Unfruchtbarkeit neugeborener Männchen hervorzurufen. Obwohl diese Methode eine gute Wirksamkeit zeigen würde, wird der Ansatz nicht verfolgt, da er als umstritten angesehen wird. Der verantwortliche für den Aktionsplan, Brent Beaven, ist sicher, dass die

eingesetzten Methoden ihre Wirkung zeigen werden. „Selbst wenn die Ausrottung der eingeschleppten Raubtiere nicht komplett gelinge, werde die Natur massiv profitieren“ [5]. [5]

2.2.6 DAS POSSUM

Beim Possum ist wichtig zu beachten, dass es sich nicht um das Opossum handelt, mit welchem es aufgrund seiner englischen Bezeichnung „Possum“ oft verwechselt wird [3, S. 20]. Tatsächlich ist mit dem Possum der Fuchskusu (Trichosurus vulpecula) gemeint (s. Abbildung 2). Beim Fuchskusu handelt es sich um einen Beutelsäuger, welcher der Familie der Kletterbeutler angehört. Er stellt die größte Art des Kusus dar. Sie sind vorwiegend Baumbewohner und nachtaktiv. [4] Die Kommunikation von Possums funktioniert sowohl über den Geruch als auch über Geräusche, die z. B. zur Paarung rufen oder Alarm schlagen bedeuten können [22].

Abbildung 2: Der Fuchskusu ("Possum") aus [4]

DIE ERNÄHRUNG

Possums sind Omnivoren [22], was bedeutet, dass sie sowohl Pflanzen, als auch Fleisch fressen. Zu ihrer Hauptnahrungsquelle gehören Blätter [4]. Zusätzlich zählen aber auch Früchte, Blüten, Knospen [4] und Insekten [22], sowie Vögel und die Eier dieser [19], zu ihrem Nahrungsspektrum.

DIE PAARUNG

Die Geschlechtsreife erreichen die Possums in einem Alter von 12 Monaten [23]. In der Paarungszeit bewegen sich Weibchen und Männchen aus ihrem bekannten Gebiet heraus, um nach einem Partner zu suchen [4]. Die Männchen locken die Weibchen mit ihren Lauten zur Paarung an [22]. Nachdem das Weibchen trächtig wurde, dauert es ca. 16 - 18 Tage, bis es ein Junges bekommt, das nachher vorerst für 4 - 5 Monate im Beutel des Weibchens bleibt [24]. Im Regelfall wirft das Weibchen nur ein Junges pro Jahr [4], allerdings kann es in selteneren Fällen auch zwei Junge bekommen [18, S. 2].

2.2.7 DER KIWI

Der Kiwi (Apteryx [25]) (s. Abbildung 3) gilt als Nationalzeichen Neuseelands, woher auch die Namensgebung für die einheimischen Neuseeländer kommt, die sich ebenfalls „Kiwis“ nennen und auch der neuseeländischen Währung seinen Namen erteilt haben [26].

Abbildung 3: Der Kiwi aus [26]

Der Kiwi stellt einen der einzigartigsten und auch ältesten noch lebenden Vogel der Welt dar, da er schon vor rund 30 Millionen Jahren auf der Erde weilte [25]. Da der Kiwi ein Laufvogel ist und nicht fliegen kann, hat er sehr kräftige Beine und im Vergleich zu flugfähigen Vögeln, sehr schwere Knochen. Eine Besonderheit des Kiwis ist sein ausgeprägter Geruchs- und Gehörsinn. Im Gegensatz dazu sind seine Augen allerdings unterentwickelt, was seine Nachtaktivität erklären kann, da seine anderen Sinne besser ausgeprägt sind als das Sehen. [26]

DIE ERNÄHRUNG

Genau wie das Possum, zählt auch der Kiwi zu den Omnivoren [25]. Nachts geht er auf Nahrungssuche, indem er mit seinem langen Schnabel in der Erde wühlt [26]. Dort sucht er nach Würmern, Maden, Samen und Beeren [25], wobei seine Lieblingsnahrung die einheimischen Würmer sind [27]. Auch manche Blätter, wie die „Totara-Blätter“, gehören zu den Nahrungsquellen der Kiwis [27].

DIE PAARUNG

Jedes Jahr erfolgt ein Paarungsritual in den Monaten März bis Juni [25], wenn eine Menge Nahrung vorhanden ist [28]. Männliche Kiwis werden mit ca. 18 Monaten geschlechtsreif, während die Weibchen erst im Alter von ca. 3 Jahren die Geschlechtsreife erreichen [28]. Meist bleiben die Kiwis für ein Leben lang Partner, wenn sie sich einmal gefunden haben. Sie bauen anschließend eine Höhle, in der das Ei gelegt werden kann. Dieses ist, proportional gesehen, „das größte Ei in der Vogelwelt“ [26], da das gelegte Ei 25% der Körpergröße des Kiwis ausmacht. Die nächsten 10 Wochen brütet das Männchen das Ei aus, während das Küken den im Ei vorhandenen Dotter frisst, um nach dem Schlüpfen nicht auf das Füttern der Eltern angewiesen zu sein. [26] Pro Jahr kann ein Kiwi ca. 6 Eier legen [29].

2.3 VERGLEICH VON BESTEHENDEN SIMULATIONEN

Um eine Vorstellung zu bekommen, wie man eine Simulation eines Ökosystems in Unity am besten kreieren kann und welche Elemente dafür notwendig sind, wurde vorerst nach bereits bestehenden Ökosystemen, die ebenfalls mit der Game Engine Unity erstellt wurden, gesucht. Hierfür bot sowohl Google als auch die Plattform YouTube geeignete Anregungen und Beispiele. Auf zwei dieser Beispiele, zum einen auf das erstellte Ökosystem von Sebastian Lague [30] und zum anderen auf die Simulation von Alexandre Sajus [31], wird in den folgenden Abschnitten genauer eingegangen.

2.3.1 DAS ÖKOSYSTEM VON SEBASTIAN LAGUE [30]

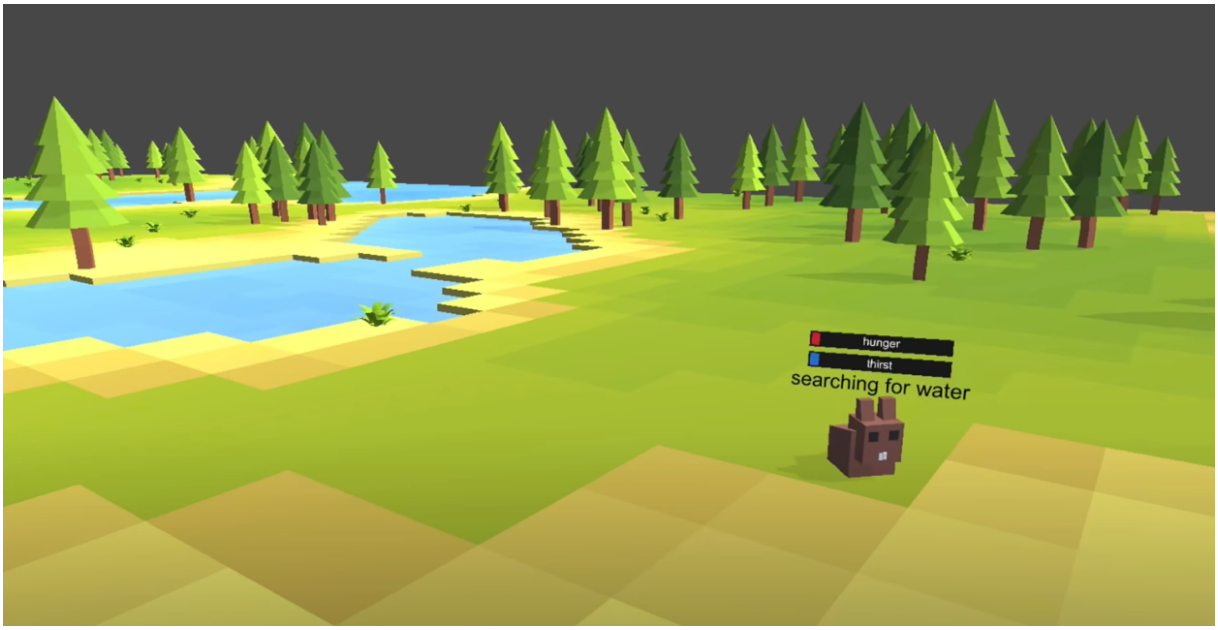

Abbildung 4: Ausschnitt der Simulation von Sebastian Lague nach [30]

Die Simulation, an der sich am meisten für die Erstellung des in dieser Bachelorarbeit beschriebenen Ökosystems orientiert wurde, ist die Umsetzung von Sebastian Lague, mit dem Titel „Coding Adventure: Simulating an Ecosystem“, die er auf YouTube [30] kurz präsentiert. Dieses Video war hilfreich zur Ideenfindung und Konzeptionierung sowie zur Herausarbeitung wichtiger Komponenten, die in einem Ökosystem enthalten sein müssen. Ein Ausschnitt des Programms ist der Abbildung 4 zu entnehmen. Im Folgenden wird auf die verschiedenen Themenpunkte der erstellten Simulation von Sebastian Lague genauer eingegangen.

IDEENFINDUNG UND ERSTE SCHRITTE

Auch Sebastian Lague hat sich, wie er auf YouTube erzählt, an einer bereits bestehenden Simulation [32] orientiert, welche sich „Natürliche Selektion simulieren“ nennt und ebenfalls auf YouTube zu finden ist. Zunächst wurde eine Umgebung für die Tiere in Form von einer Karte mit Wasser- und Naturelementen erstellt. Als Futterquelle wurden kleine Pflanzen eingefügt. Die Tierarten umfassen Hasen und Füchse, wobei zuerst mit der Hasenpopulation gestartet wurde und später der Fuchs als Feind dazukam, vor dem die Hasen fliehen können.

BEDÜRFNISSE

Jedes Tier hat Bedürfnisse, welche Hunger, Durst und den Fortpflanzungstrieb umfassen. Als Futterquellen werden für die Hasen Pflanzen und für die Füchse der Hase als Beute genutzt. Ihren Durst können die Tiere beim Trinken aus den Wasserbereichen auf der Karte stillen. Die Bedürfnisse können direkt über dem Tier in Balkenform eingesehen werden. Ist der Hunger oder Durst stark genug, so begibt sich das Tier auf die Suche nach Nahrung oder einer Wasserquelle. Gegensätzlich zur in dieser Bachelorarbeit beschriebenen Simulation, teilt Sebastian Lague dem Fortpflanzungstrieb einen höheren Stellenwert zu und lässt das Tier allem voran nach einem Partner suchen.

DETEKTION

Um jedes Tier befindet sich ein vordefinierter Radius, in welchem das Individuum Objekte erkennen kann. Außerhalb des Radius ist es dem Tier nicht möglich, Objekte zu detektieren. Sobald ein Objekt innerhalb des Radius erkannt wird, macht sich das Tier, bei bestehendem Bedürfnis, auf den Weg dorthin.

FORTPFLANZUNG

Über den Radius können sich die Tiere untereinander erkennen und unter bestimmten Voraussetzungen (z. B. gegensätzliche Geschlechter) den Paarungsakt vollziehen. Hierfür hüpfen beispielsweise die Hasen voreinander auf und ab und gehen zunächst wieder ihrer Wege. Das Weibchen ist schwanger und kurz darauf spawnen nacheinander mehrere kleinere Häschen um die Mutter herum.

WACHSTUM DER BABYS

Der Nachwuchs der Hasen wächst allmählich heran und entwickelt seine Fähigkeiten (Geschwindigkeit, Sichtfeld) erst mit der Zeit. Aus diesem Grund ist er vorerst verwundbarer als ein erwachsenes Tier. Die Intensität der Verwundbarkeit ist abhängig davon, wie lange das Baby im Bauch der Mutter heranwuchs. Je länger die Mutter trächtig ist, desto stabiler ist das Kind, desto weniger oft kann sie allerdings Nachwuchs produzieren.

GENE

Jedes Tier hat verschiedene Gene, die das Individuum beeinflussen. Dazu gehört die Trächtigkeitszeit, die Größe des Sichtfelds (Radius um das Tier), die Zeit, um nach einem möglichen Partner zu suchen, die Geschwindigkeit und spezifisch bei den Männchen ein Attraktivitätslevel (hier „Desirability“ genannt). Gene werden bei der Geburt eines Babys mitgegeben. Dabei wird zufällig entschieden, welche Gene von der Mutter oder vom Vater vererbt werden. Zudem besteht bei jedem Gen die Möglichkeit einer Mutation, sodass die Werte der Gene variieren.

ATTRAKTIVITÄT

Ein Attraktivitätsgen besitzen nur die männlichen Hasen. Attraktive Männchen wurden durch eine rote Fellfarbe gekennzeichnet. Diese haben eine höhere Chance, vom Weibchen als Paarungspartner akzeptiert zu werden.

DESIGN

Das Design des Ökosystems von Sebastian Lague ist sehr reduziert ausgearbeitet worden und kann am ehesten als Low-Poly-Stil bezeichnet werden. Die Wahl des Designs gibt der Simulation einen niedlichen Charakter und stellt Elemente und Objekte gut erkennbar dar. Das Design wurde auch für die Ausarbeitung der eigenen Simulation als Beispiel hergenommen. Die Tiere wurden kaum animiert. Lediglich der Hase, welcher blockartig gestaltet wurde, bewegt sich in Form einer Hüpf-Animation fort.

2.3.2 DAS ÖKOSYSTEM VON ALEXANDRE SAJUS [31]

Abbildung 5: Ausschnitt der Simulation von Alexandre Sajus nach [31]

Das Ökosystem von Alexandre Sajus [31] (s. Abbildung 5), welches ebenfalls auf der Plattform YouTube zu finden ist, soll, laut dem Ersteller, keinen besonderen Nutzen haben, sondern lediglich zum entspannten Ansehen dienen. Auch diese Simulation wurde mit Unity erstellt.

IDEENFINDUNG UND ERSTE SCHRITTE

In dieser Umsetzung wurde mit Hühnern und Löwen als Tierarten gearbeitet. Das Verhalten der Tiere wird durch eine sogenannte „State Machine“, bzw. eine Zustandsmaschine gesteuert. Hier wird festgelegt, wann und unter welchen Voraussetzungen ein Tier in Aktion tritt. Genaueres zur Zustandsmaschine ist im entsprechenden Abschnitt nachlesbar.

BEDÜRFNISSE & VERHALTEN

Wie sich ein Tier verhält, ist hier von der Menge der Energie abhängig. Ist die Energie unter dem Wert 50, so sucht das Tier nach etwas Essbarem, bzw. es beginnt nach einer Beute zu jagen. Ist das Energielevel höher als 50, so versucht es sich fortzupflanzen und hält Ausschau nach einem möglichen Partner. Die Energie entspricht hier also dem Sättigungswert der Tiere. Nach der Nahrungsaufnahme erhöht sich der Energiewert wieder. Entspricht die Energie dem Wert 0, so stirbt das Tier. Hühner können vor Löwen fliehen, indem sie sich in die entgegengesetzte Richtung bewegen.

DETEKTION

Um Objekte zu detektieren, wird sich hier der Raycast-Funktion bedient (s. Abbildung 6). Hierbei werden Linien in eine Richtung gezeichnet. Die Reichweite dieser Linien entspricht hier dem Sichtfeld des Tieres. Kollidiert eine Linie mit einem Objekt (z. B. der Beute oder einer essbaren Pflanze), so wird dies als Nahrung erkannt und kann vom entsprechenden Tier gefressen werden.

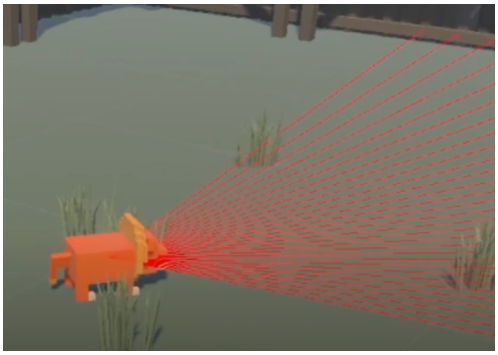

Abbildung 6: Detektion über Raycast nach [31]

FORTPFLANZUNG

Die Partnersuche funktioniert hier simpel, da sich mit jedem Tier unabhängig von Geschlecht oder sonstigen Bedingungen gepaart werden kann, solange der Energiewert hoch genug ist. Die Tiere laufen ineinander und direkt im Anschluss entstehen neue, bereits ausgewachsene Tiere um die sich paarenden Individuen herum. Auch die Wachstumsphase wird demzufolge übersprungen.

GENE

Als vorgegebene und vererbbare Gene ist hier die Geschwindigkeit zu nennen. Das Baby erbt den Durchschnitt der Geschwindigkeit beider Eltern. Schnellere Tiere verlieren als Nachteil mehr Energie und können sich daher nicht so oft fortpflanzen wie langsamere Tiere, dafür aber schneller die Flucht vor Feinden ergreifen.

DESIGN

Auch das Design der Simulation von Alexandre Sajus ist sehr minimalistisch und in einer Art Low-Poly-Stil umgesetzt worden. Die Tiere wurden in einem blockartigen 3D-Design gestaltet, wie in Abbildung 5 und 6 zu sehen ist. Die Szene spielt sich nicht, wie bei dem vorherigen Beispiel, auf einem offenen Stück Land (Karte) mit einem Wechsel von Wasser- und Grasflächen ab, sondern in einem eingezäunten Terrain, auf dem sich nur Grasland und Pflanzen befinden. Um die Zäune herum wurden Bäume im Low-Poly-Stil beigefügt. Auf Animationen wurde hier gänzlich verzichtet.

2.3.3 UNTERSCHIEDE & GEMEINSAMKEITEN ZUR EIGENEN SIMULATION

Für die eigene Umsetzung einer Simulation eines Ökosystems war es hilfreich, sich diverse bestehende Beispiele anzusehen und nach Präferenz und Notwendigkeit zu entscheiden, welche Elemente und Funktionen enthalten sein sollen. Hierbei wurden einige Inhalte der in den vorherigen beschriebenen Beispielen ähnlich umgesetzt. Allerdings gibt es auch diverse Unterschiede, da nach eigener Präferenz entschieden wurde, welche Funktionsweisen vorhanden sein sollen. In den folgenden Abschnitten werden ein paar der Gemeinsamkeiten und Unterschiede der beschriebenen Beispiel-Ökosysteme im Vergleich zur eigenen Umsetzung herausgearbeitet und erklärt.

IDEENFINDUNG UND ERSTE SCHRITTE

Wie bereits erwähnt, wurde sich bei der Erstellung vorwiegend am Beispiel des Ökosystems von Sebastian Lague [30] orientiert. Die ersten Schritte und die Erstellung des Konzepts wurden nach mehrmaligem Ansehen des Videos erstellt und immer wieder ergänzt. So sollte ähnlich wie in Sebastian Lagues Simulation, eine Karte vorhanden sein, die Wasser- und Grasland enthält, auf der sich die Tiere bewegen können. Genau wie im Beispiel gibt es ein Beutetier und einen Fressfeind, der in der eigenen Umsetzung allerdings dem Possum und als Beutetier dem Kiwi entspricht. Die Tiere werden, wie prinzipiell in beiden Beispielen, über eine Zustandsmaschine gesteuert.

BEDÜRFNISSE

In der eigenen Simulation unterscheiden sich die Bedürfnisse im Vergleich zu den beiden Beispielen wie folgt: Es gibt zusätzlich zum Sättigungs-, Durst- und Paarungssuche-Wert, welcher im selbst erstellten Ökosystem dem Sozialbedürfnis entspricht, auch noch einen spezifischen Gesundheitswert, der nur bei Possums relevant ist, die sich beim Fressen eines Pilzes infizieren können. Die Reihenfolge der Bedürfniserfüllungspräferenzen variiert ebenfalls im Vergleich zu Sebastian Lagues Simulation, denn im eigenen Ökosystem steht das Erfüllen der lebenserhaltenden Bedürfnisse des Individuums, wie das Stillen des Hungers, an erster Stelle. Dies wurde auf diese Weise ausgearbeitet, da es natürlicher erscheint, dass ein Tier nur bei gestillten lebenswichtigen Bedürfnissen mit der Partnersuche beginnt. Zudem gibt es nicht nur eine Nahrungsquelle für jedes Tier, sondern, wie in der Realität, mehrere Nahrungsarten, welche Possums und Kiwis aufnehmen können. Diese werden nach vorgegebenen Präferenzen aufgesucht. Genaueres zur Nahrungsaufnahme ist im Kapitel „Bewegungsziele und Zustände“ und „Nahrungsquellen“ nachlesbar. Als Gemeinsamkeit zu beiden vorgestellten Simulationen, ist die Fluchtmöglichkeit des Beutetiers zu nennen. Das Beutetier bewegt sich mit erhöhter Geschwindigkeit vom Feind weg. Wie auch in der Simulation von Sebastian Lague [30] zu sehen ist, sollte eine Anzeige der Bedürfnisse jedes einzelnen Tieres vorhanden sein. Diese befindet sich jedoch links oben, in Form des „AnimalMenus“, das sich öffnet, sobald man das gewünschte Tier mit der Maus anwählt. Dies sollte einer besseren Übersicht dienen, sodass nicht jedes Tier einige Bedürfnis-Balken über sich hat, die eventuell bei engem Beieinanderstehen mehrerer Tiere zu Verwirrung führen.

DETEKTION

Bei der Detektion wurde sich nicht wie bei Alexandre Sajus' Simulation [31] für die Umsetzung mittels Raycast entschieden, sondern vorerst, wie bei Sebastian Lagues Ökosystem [30], mit einem Radius gearbeitet. Nach genauerer Überlegung wurde sich dafür entschieden, den Radius für das Geruchsfeld zu nutzen und zusätzlich ein Sichtfeld einzubauen, das man sich eher wie eine Halbkugel vor dem Tier vorstellen kann. Insofern gibt es hier, im Vergleich zu den erklärten Beispielen, zwei verschiedene Möglichkeiten Objekte zu detektieren. Für die Gestaltung des Sichtfelds wurde sich hier am menschlichen Sichtfeld orientiert [33], wie in Abbildung 7 zu sehen ist. Genaueres zur Detektion ist dem Abschnitt „Objektdetektion (Sight & Smell)" zu entnehmen.

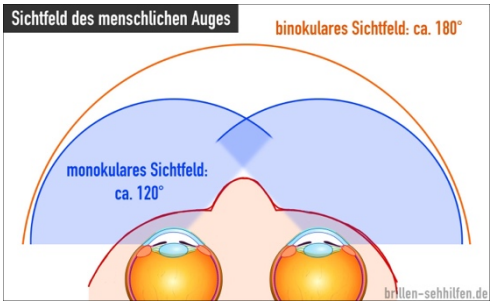

Abbildung 7: Menschliches Sichtfeld aus [33]

FORTPFLANZUNG

Die Fortpflanzung sollte, wie in Sebastian Lagues Simulation [30], nur unter bestimmten Bedingungen durchführbar sein. Um den Paarungsakt noch einmal mehr hervorzuheben, wurde mit Herzchen-Symbolen über den Tieren gearbeitet. Nach der Paarung entstehen nicht direkt neue Tiere, wie in Alexandre Sajus' Umsetzung [31], sondern es beginnt eine Trächtigkeitsphase, bevor der Nachwuchs zur Welt kommt. Bei den Kiwis entstehen zusätzlich vorerst Eier, aus denen die Kleinen nach einer bestimmten Zeit schlüpfen. Die Trächtigkeitsdauer ist im Gegensatz zu Sebastian Lagues Simulation [30] vordefiniert und hat keinen Einfluss auf die Überlebensfähigkeit der Babys.

GENE

Die jeweiligen Attribute jedes Tieres werden bei der Instanziierung festgelegt. Um den Rahmen des Projekts nicht zu sprengen, wurde, im Gegensatz zu den beispielhaften Simulationen, sowohl auf Vererbung als auch auf Mutationen in diesem simulierten Ökosystem verzichtet. Damit trotzdem eine gewisse Varianz vorhanden ist, die auch in der Natur besteht, werden die Werte jedes Tieres bei der Instanziierung zufällig innerhalb eines festgelegten Wertebereichs generiert.

ATTRAKTIVITÄT

Die Idee eine Attraktivität für männliche Tiere einzubauen, entstammt der Umsetzung von Sebastian Lague [30] und wurde auch in die eigene Simulation eingebaut. Genau wie in seinem erstellten Ökosystem beeinflusst die Attraktivität des Männchens die Zustimmung des Weibchens zum Paarungsakt. Genaueres ist dem Kapitel „Paarung" zu entnehmen.

DESIGN

Beim Design wurde sich anfangs stark am Beispiel der Simulation von Sebastian Lague [30] orientiert. Vor allem die Naturfläche, auf der sich die Tiere bewegen, ist gestalterisch an seinem Beispiel angelehnt. Für das Design wurde sich jedoch gänzlich auf den Low-Poly-Stil festgelegt, sodass keine blockartigen 3D-Tiere, sondern ebenfalls Low-Poly-Modelle erstellt wurden. Genaueres zum Design ist im entsprechenden Kapitel nachlesbar. Außerdem sollten die Tiere animiert sein, sodass sie lebendiger und anschaulicher wirken, was in den vorgesellten Simulationen nicht oder kaum der Fall ist.

BESONDERHEITEN UND ABHEBUNGEN

Als Besonderheiten, die nicht in den vorher beschriebenen Ökosystemen vorhanden sind, ist z. B. der Kostümmodus zu nennen, das Bewegen der Kamera über die Karte, um Tiere genauer beobachten zu können und die Möglichkeit der Kiwis die Wasserflächen zum Schwimmen zu nutzen. Es wurde also nicht nur Wert auf das Funktionelle gelegt, sondern ebenfalls auf die Bedienung für den Nutzer, was der Simulation ein wenig Interaktion hinzugibt. Weitere Besonderheiten sind z. B. im Abschnitt „Spezialeffekte“ zu finden.

3 UMSETZUNG DER SIMULATION

3.1 IDEE UND KONZEPT

Nachdem sich genauer mit sich verändernden Ökosystemen durch den Import gebietsfremder Tierarten befasst wurde, fiel die Entscheidung auf die Darstellung einer solchen Problematik in Form von einer Simulation eines derartigen Ökosystems. Dabei wurde aus den im Theorieteil beschriebenen Fällen, die Problematik der Ausbreitung des Possums und die enorme Bedrohung für die heimischen Kiwis Neuseelands als Thema ausgewählt.

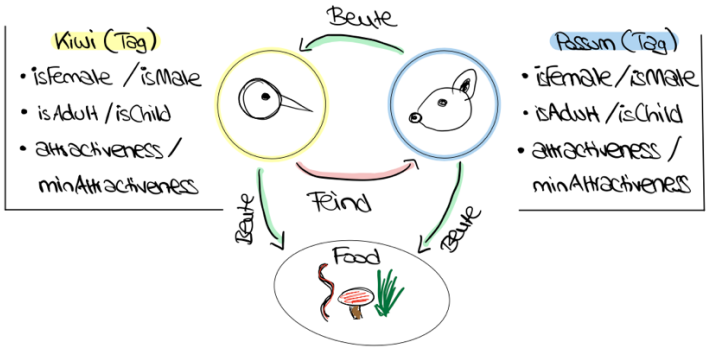

Abbildung 8: Skizze der Planung

ERSTE ÜBERLEGUNGEN

Die ersten Überlegungen zur Umsetzung entstanden auf dem iPad in der Notiz-App „Notability“ und enthielten vorerst bestimmte Zielsetzungen und Skizzen, welche vor allem die beiden Tierarten und deren Verhalten beschrieben (s. Abbildung 8). Auch weitere Funktionen und Features, die enthalten sein sollten, wurden erstmals auf dem iPad notiert. Ein Aufbereiten dieser Vorüberlegungen war sehr hilfreich für die strukturierte Umsetzung des Programms.

LASTEN- UND PFLICHTENHEFT

PROGRAMMIERUNG		DESIGN		RECHERCHE & SCHREIBEN	
BASICS		**MUSS**		Allgemeine Recherche Kiwis & Possums	✓
Bewegung der Tiere	✓	Kiwi & Possum Pack	✓	Recherche Ökosysteme	✓
Nahrung & Jagen	✓	Ground bauen mit Blender	✓	Schreiben	✓
Beutetier flieht vor Feind	✓	Natur gestalten (Nature Pack)	✓	**NACHBEREITUNG**	
HUNGER-SYSTEM		**SOLL**		Usertests & Analyse	✓
Hunger-System überlegen & einbauen	✓	Kiwi in Blender	✓	Arbeit Korrektur lesen lassen	✓
Objektdetektion (Smell- & Sight-Radius)	✓	Kiwi animieren	✓	Arbeit binden lassen	✓
Hungerwerte Random bei Initialisierung	✓	Possum in Blender	✓	Präsentation vorbereiten	✓
FORTPFLANZUNG		Possum animieren	✓		
Geschlechter bei Initialisierung mitgeben	✓	Naturelemente selbst bauen	✓		
Paarung & Babies	✓	Musik selber machen	✓		
Kiwis legen Eier -> Babies schlüpfen	✓	Passende Sounds einfügen	✓		
FORTPFLANZUNG					
Plants / Trees spawnen nach gewisser Zeit	✓				
Spawning nicht in Wasser	✓				
Spawning nach gewisser Menge stoppen	✓				
NATUR					
Wassercollider als Trinkquelle	✓				
Wasser nur für Kiwis „Walkable“ machen	✓				
MENÜ					
Startmenü bauen	✓				
Feste Anzahl Tiere / Pflanzen in Startmenü setzen	✓				
SPECIALS					
Diagramme einbauen	✓				
Outline bei angeklicktem Animal	✓				
Status anzeigen -> Symbole über Animal (!, <3)	✓				
Kostümmodus	✓				
UI verbessern (Sound an/aus, zoom in, zoom out)	✓				
Steuerung	✓				
Diverse Mutationen					

Abbildung 9: Lasten- und Pflichtenheft

Zunächst wurde ein Lasten- und Pflichtenheft (s. Abbildung 9) mit dem Programm „Numbers“ erstellt, in dem einige Ziele in den Bereichen Programmierung, Design, Recherche und Nachbereitung definiert wurden. Diese wurden während der Umsetzung, bei der Entstehung neuer Ideen oder aufgrund von abweichender Zeiteinschätzung regelmäßig angepasst.

3.2 EINARBEITUNG UND ERSTE SCHRITTE

AUSWAHL DES ENTWICKLUNGSPROGRAMMS

Durch die Empfehlung von Prof. Dr. Hopf die angestrebte Simulation mit Unity zu entwickeln, wurde gezielt nach Beispielsimulationen von Ökosystemen in YouTube gesucht, welche mit der Game Engine Unity erstellt wurden. Unity bietet die Möglichkeit, in den Programmiersprachen C++ oder C# zu programmieren. Da die Webseite von Unity nennenswerte Vorteile von C# beschreibt, wie z. B. die Abnahme der Speicherverwaltung und die einfachere Einarbeitung [34], wurde sich für das Programmieren der Simulation in C# entschieden. Die für diese Simulation genutzte Unity-Version nennt sich „Unity 2021.3.10f1“. Genaueres über Unity wird im Kapitel „Die Game Engine Unity“ berichtet.

WAHL DER DIMENSION

Des Weiteren kann man sowohl ein 2D- als auch ein 3D-Programm in Unity erstellen. Die entsprechende Einstellung lässt sich beim Anlegen eines neuen Projekts konfigurieren. Nach einiger Überlegung wurde sich aus designtechnischen Gründen und zur besseren Übersicht für eine Simulation im 3D-Stil entschieden. Der 3D-Stil trägt dazu bei, Individuen anschaulicher und lebendiger wahrzunehmen und ermöglicht dem Nutzer, mehr in die Simulation einzutauchen.

WAHL DER IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

Für die Erstellung der Skripte wurde die IDE „Visual Studio 2022 for Mac“ ausgewählt, da diese weitaus mehr Optionen und Vereinfachungen beim Coding bietet als der bisher genutzte Code Editor „Visual Studio Code“.

EINARBEITUNG IN UNITY

Um sich mit dem Programm Unity vertraut zu machen, wurden vorerst kleinere Projekte, wie z. B. rudimentäre Spiele in Unity mithilfe von YouTube-Tutorials erstellt. Dies schaffte einen besseren Einblick über den Funktionsumfang des Programms.

ERSTE SCHRITTE

Nach einer kurzen Einarbeitungszeit wurde ein erstes „Ökosystem-Projekt“ angelegt. Es wurde simpel begonnen, indem aus einem langgezogenen Würfel eine Bodenfläche gemacht wurde, die als Karte dienen sollte. Die Tiere bestanden ebenfalls aus kleinen Würfeln, die sich vorerst nur auf der Karte hin- und herbewegten. Es stellte sich heraus, dass die Nutzung eines „NavMeshs“, auf welches später genauer eingegangen wird, für Bewegungsabläufe gut geeignet ist. Zur korrekten Verwendung des „NavMeshs“ war folgendes YouTube-Video hilfreich: [35]. Wie auch in diesem Video, wurden nun als Dummys nicht mehr Würfel, sondern Kapseln („Capsules“) als Bewegungsobjekte verwendet.
Große Teile der Programmlogik wurden zuerst eingebaut und programmiert, bevor den Tieren ihr eigentliches Erscheinungsbild zugeordnet wurde. Dies vereinfachte den anfänglichen Arbeitsprozess,

da nicht sofort zwischen den verschiedenen Eigenschaften der Tiere (Größenunterschiede, Formunterschiede…) unterschieden werden musste. Bei Verwendung der Kapsel-Modelle konnte sich vorerst nur auf die programmatische Logik konzentriert werden, wie z. B. die Bewegungsziele der Tiere.

3.3 DIE GAME ENGINE UNITY

Um einen genaueren Einblick zur Game Engine Unity zu erhalten, wird in diesem Kapitel verstärkt über den Aufbau und den Funktionsumfang von Unity informiert. Da Unity bereits vorgefertigte Implementierungen, wie beispielsweise adäquate Komponenten zur Ersterstellung von 3D-Programmen bereitstellt, kann der Arbeitsprozess durch die Nutzung von Unity vereinfacht werden.

3.3.1 ALLGEMEINER AUFBAU

Generell ist es möglich, in Unity den Aufbau der Fenster nach eigenen Vorlieben anzupassen, indem man Fenster an beliebiger Stelle hinzufügt, entfernt oder verschiebt. Im Weiteren wird demzufolge verstärkt auf die Standard-Anordnung der Fenster von Unity eingegangen und die Funktionen der jeweiligen Bereiche verdeutlicht.

Abbildung 10: Aufbau Unity

DIE PROGRAMMANSICHT

Im mittleren zu sehenden Fenster kann man das Programm in seinem Ist-Zustand sehen. Über die Auswahl der Tabs ist es möglich, zwischen der „Scene“- und der „Game-View“ zu wechseln (s. Abbildung 10). Die „Scene-View“ stellt hierbei eine Art Bearbeitungsmodus dar, in welchem man dem Programm z. B. Objekte hinzufügen kann oder Veränderungen vornehmen kann. In der „Game-View“ hingegen kann man das erstellte Programm so wahrnehmen, wie es im Ausgabezustand aussehen würde. Hierfür ist die Position der Kamera ausschlaggebend, denn die Sicht der Kamera auf das Programm, bestimmt den Bildausschnitt, in welchem das Programm bei Ausgabe präsentiert wird. Beim Betätigen des „Play-Buttons“ kann man den Ist-Zustand des Programms beurteilen. Möchte man Änderungen vornehmen, so ist bei gerade laufendem Programm zu beachten, dass diese, sobald der „Stop-Button“ betätigt wird, wieder zurückgesetzt werden. Die Scene-View eignet sich trotzdem hervorragend, um Beobachtungen am Objekt während der Laufzeit vorzunehmen, wie z. B. die Funktion des Detektions-Radius eines Objekts, während es sich bewegt.

DIE HIERARCHY

Auf der linken Seite befindet sich die „Hierarchy“ des Programms, welche enthaltene Szenen und Objekte aufzeigt. Diese lassen sich nach eigener Präferenz ordnen oder auch gruppieren. Eine Gruppierung ist mithilfe der Erstellung eines „Empty Parents“, welches ein leeres „Game-Object“ darstellt und die gruppierten „Game-Objects“ umfasst, möglich. Game-Objekte bezeichnen in Unity jedes in der Hierarchy aufgeführte Objekt. Dabei kann es sich um Spielfiguren handeln, aber auch um eine Kamera oder ein leeres Objekt. Jedes Game-Objekt hat gewisse Komponenten, welche im Inspector einsehbar und bearbeitbar sind. Neue Objekte können per Rechtsklick in der „Hierarchy“ erstellt werden. Beim Hinzufügen hat man mehrere Arten von Objekten zur Auswahl, wie auf Abbildung 11 zu sehen ist.

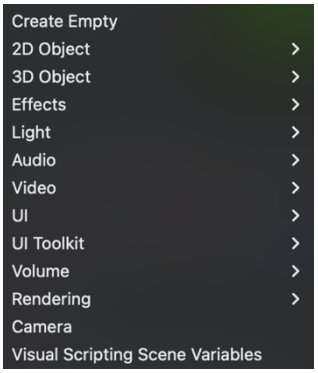

Abbildung 11: Auswahl der Objekterstellung

DER INSPECTOR

Auf der rechten Seite befindet sich nach der Standard-Einstellung der „Inspector“, in welchem man je nachdem, welches Objekt (z. B. Game-Objekt, Material, Shader…) ausgewählt ist, die Eigenschaften des Objekts einsehen und anpassen kann. Außerdem ist es hier möglich über die Schaltfläche „Add Component“ dem Objekt bestimmte Komponenten zuzuweisen. Komponenten können z. B. Skripte sein oder auch „Physics“, wie eine Rigidbody-Komponente, mit der sich Masse und Gravitation eines Objekts beeinflussen lassen. Der Inspector ist demzufolge ein sehr relevantes und häufig genutztes Tool bei der Erstellung von Programmen in Unity.

DIE PROJECT-ÜBERSICHT

Im unteren Abschnitt ist die Ordnerstruktur des Projekts zu sehen, in der sämtliche Elemente zu finden sind, die das Projekt enthält. Hier ist es möglich, neue Ordner anzulegen oder zu löschen und zusammenhängende Elemente in einen jeweiligen Ordner einzupflegen. Dies gewährleistet eine übersichtliche Gestaltung des Projekts. Unterteilungsmöglichkeiten können hier z. B. das Einbetten von Skripten in einen Ordner „Scripts“ sein oder die Zuordnung sämtlicher Materialien zu einem „Material“-Ordner.

DIE KONSOLE („CONSOLE“)

Klickt man, ebenfalls im unteren Bereich, auf den Tab „Console“, so ist es möglich auf die Konsole zuzugreifen und diverse „Debug-Logs“ einzusehen. Ein „Debug-Log“ ermöglicht es, Nachrichten in der Konsole auszugeben, um bestimmte Teile des Codes testen oder das programmatisch erzeugte Verhalten genauer beobachten zu können. Mit dem Code-Befehl

```
Debug.Log(„zu printende Nachricht“);
```

lässt sich ein „Debug-Log“ erstellen.

3.3.2 BASIC-ELEMENTE

SZENEN

Eine Szene in Unity stellt eine Art Setting eines bestimmten Szenarios dar. Bei der Entwicklung eines Spiels lässt sich eine Szene am besten mit einem Level vergleichen, allerdings ist ein Szenenwechsel auch notwendig, um z. B. ein Startmenü erstellen zu können. Der Wechsel einer Szene lässt sich programmatisch, z. B. über das Betätigen eines Buttons, einleiten.

DIE KAMERA

Ein Objekt, das ebenfalls einer gesonderten Beschreibung bedarf, ist das Kamera-Objekt. Die Kamera ist für den Ausschnitt und zu einem großen Teil auch für das sonstige Erscheinungsbild des Programms verantwortlich. Ihre Position und Rotation lässt sich mithilfe des Inspectors anpassen. Bei Verwendung einer Skybox für den Hintergrund, ist es erforderlich, dies in der Einstellung des Kamera-Objekts anzugeben. Hierfür ist es nötig, bei dem Punkt „Clear Flags“ (s. Abbildung 12) die „Skybox“ auszuwählen. Unter das Kamera-Objekt lassen sich über die „Hierarchy“ auch Objekte der UI anhängen, um diese in korrekter Anordnung sichtbar zu machen.

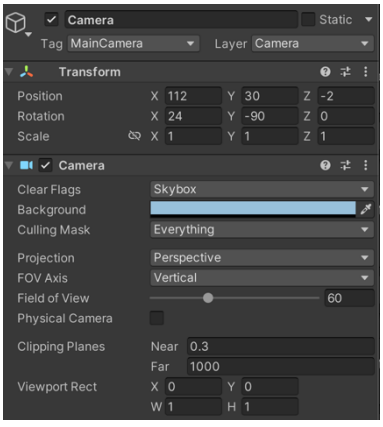

Abbildung 12: Kamera-Einstellungen

DAS LICHT

So wie auch die Kamera, ist bereits beim Erstellen eines neuen Unity-Programms ein Licht-Objekt „Directional Light“ beigefügt, um bestimmte Lichtverhältnisse im Programm erzeugen zu können. Bei Änderung der Rotation des Licht-Objekts, ändert sich auch der Lichteinfall auf die Szene. So kann man dunklere oder hellere Lichtverhältnisse schaffen. Außerdem ist es möglich, die Farbe des Lichts anzupassen und auch eine Farbtemperatur miteinzubeziehen (s. Abbildung 13), um beispielsweise eine Szene in der Abenddämmerung erschaffen zu können.

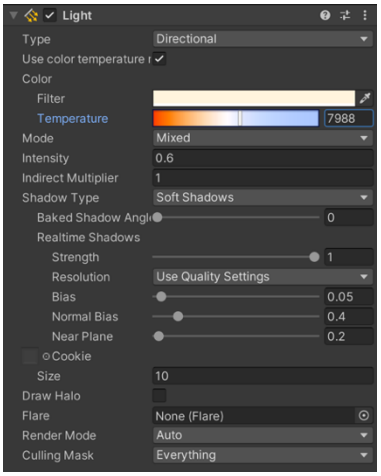

Abbildung 13: Licht-Einstellungen

PREFABS

Sogenannte „Prefabs“ sind Game-Objekte in Unity, welche bestimmte vorkonfigurierte Komponenten und Einstellungen enthalten und / oder ein Hauptobjekt für mehrere Objekte des gleichen Typs darstellen. Derartige Game-Objekte ermöglichen eine kontinuierliche Nutzung und Änderung eines bestimmten Objekts. Bei der Nutzung eines Prefabs ist es möglich, mehrere Instanzen des gleichen Typs zu erstellen und deren Eigenschaften für alle erzeugten Instanzen gleichzeitig über die Einstellung des Prefabs zu verändern. Dies beschleunigt und vereinfacht das Verändern und Anpassen eines bestimmten Objekttyps erheblich.

In Bezug auf die erstellte Simulation ist es dadurch z. B. möglich, ein Objekt „Kiwi“ zu erstellen und diesem die zugehörigen Komponenten und Einstellungen mitzugeben, wie z. B. eine bestimmte Größe. Beim Instanziieren mehrerer Kiwis kann nun auf das Kiwi-Prefab verwiesen werden und es werden mehrere Kiwis mit den gleichen Eigenschaften erstellt. Möchte man im Nachhinein beispielsweise die Größe aller Kiwis ändern, so wäre dies über die Scale-Einstellung im Kiwi-Prefab möglich. Ändert man allerdings ein über das Prefab erzeugte Objekt in der Szene, so ändert sich nur dieses ausgewählte Objekt.

3.3.3 RELEVANTE KOMPONENTEN & TOOLS

TRANSFORM

Die Transform-Komponente ist bei jedem Game-Objekt zu finden, da hier sowohl die Größe als auch die Position und Rotation des Objekts konfiguriert werden kann. Veränderungen von Eigenschaften der Transform-Komponente wurden bei der Ausarbeitung dieser Arbeit auch des Öfteren im Code durchgeführt. Über „`transform.position`“ lässt sich beispielsweise die Position des Objekts verändern.

COLLIDER

Eine ebenfalls sehr nennenswerte Komponente ist die Collider-Komponente (s. Abbildung 14), welche, je nach Bedarf, verschiedene Typen von Collidern zur Auswahl bereithält. Darunter können Collider im 2D- oder 3D-Bereich fallen, wie z. B. ein Box-Collider, welcher eine rechteckige Form aufweist, ein Sphere-Collider, der sich mit seiner Kreisform z. B. gut als Radius eignet oder ein Mesh-Collider, unter dem man eine exakte Anpassung der Collider-Maske auf das Objekt versteht. Mit Collidern kann man ein Kollidieren von Objekten miteinander auslösen. Es gibt zwei zu unterscheidende Funktionen bei der Konfiguration von Collidern. Zum einen kann man eine Kollision mit „`OnTriggerEnter`“ auslösen, wobei der Haken bei „Trigger“ gesetzt werden muss. Diese Art der Kollision wird vor allem bei Objekten angewendet, die nicht physisch miteinander interagieren müssen, im Gegensatz zu „`OnCollisonEnter`“, was für die physische Interaktion von Objekten geeignet ist [36].

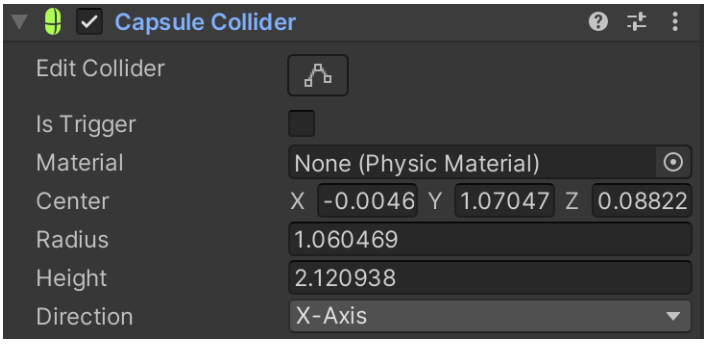

Abbildung 14: Capsule-Collider

RIGIDBODY

Mit der Rigidbody-Komponente (s. Abbildung 15) können die physischen Eigenschaften des Objekts, wie Masse und Gravitation beeinflusst werden. Je höher der Wert der Masse eingestellt wurde, desto schwerer ist das Objekt. Dies wirkt sich in bestimmten Szenarien, wie z. B. bei Kollisionen entsprechend aus, indem die Objekte mehr voneinander abprallen. Um dies zu vermeiden sind hier Einstellungen wie „Use Gravity“ oder „Is Kinematic“, wie in Abbildung 15 zu sehen ist, entscheidend. Ist „Use Gravity“ ausgewählt, so wirken physikalische Kräfte auf das Game-Objekt ein. Wählt man allerdings „Is Kinematic“ aus, so lassen sich Objekte nicht von physikalischen Kräften oder Kollisionen beeinflussen. Eine Steuerung ist hier lediglich über das Skript oder Animationen realisierbar [37]. Wenn man z. B. bei einer Kollision eine Rotation oder ein

Abbildung 15: Rigidbody-Komponente

„Herumschieben“ des Objekts vermeiden möchte, so sollte man „Freeze Position“ und „Freeze Rotation“ aktivieren.

SKRIPTE

Skripte, welche man in einem Coding-Programm, wie z. B. Visual Studio bearbeiten kann, lassen sich einem Game-Objekt als Komponente hinzufügen. Stellt man den Schutzgrad bestimmter Variablen auf „public“, so sind diese in der Skript-Komponente im Inspector-Fenster sichtbar. Eine sicherere Option bieten „Serialize Fields“, welche z. B. wie folgt im Code eingepflegt werden können:

```
[Serialize Field] private float hunger;
```

Sie bieten die Möglichkeit, Variablen im Inspector-Fenster einzusehen und zu verändern, ohne den Schutz durch die Verwendung von „public“ zu gefährden und stellen aus diesem Grund eine adäquatere Alternative dar. Beide Möglichkeiten erlauben, Variablen im Inspector anzupassen, was das Vornehmen von Veränderungen vereinfachen kann. Der Wert einer Geschwindigkeit beispielsweise lässt sich über den Inspector deutlich schneller anpassen, als eine Änderung über die entsprechende Variable im Code herbeizuführen.
Ein in Unity angelegtes Skript ist von Beginn an als „MonoBehaviour“ eingestellt. Die Funktionen Start() und Update() sind bereits vorkonfiguriert in einem Skript enthalten. Während die Start()-Methode nur einmal aufgerufen wird, wird die Update()-Funktion jeweils einmal per Frame aufgerufen. Auch zu nennen ist die Methode Awake(). Sie ist für Initialisierungen zuständig und wird noch vor der Start()-Methode, bzw. bereits beim Laden der Skript-Instanz aufgerufen. Sie eignet sich z. B. zur Initialisierung von Zuständen oder Variablen [38].

TAGS & LAYER

Unity bietet die Option, Game-Objekten Tags und / oder Layer zuzuweisen. Über gesetzte Tags kann ein Game-Objekt im Skript einfacher angesprochen werden. Dies funktioniert z. B. über den Befehl „`gameObject.CompareTag(„Tag“)`“ oder auch über „`GameObject.FindGameObjectWithTag(„Tag“)`“.

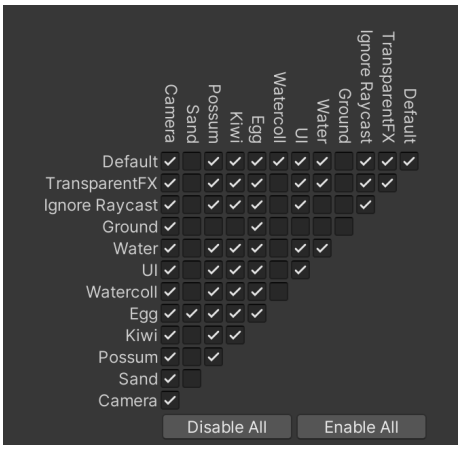

Abbildung 16: Layer-Collision-Einstellungen

Layer sind beispielsweise nützlich, um festzulegen, welche Objekte miteinander kollidieren dürfen. Die Einstellung hierfür lässt sich über die Project-Settings von Unity festlegen. Der Navigationspfad der Unity Version für Mac lautet: Edit -> Project-Settings -> Physics. Nach ein wenig Herunterscrollen findet sich die entsprechende Einstellungsmöglichkeit (s. Abbildung 16). Ein gesetzter Haken bedeutet hier, dass die Ebenen miteinander kollidieren können. Möchte man dies vermeiden, so ist es erforderlich den Haken zu entfernen.

NAVMESH

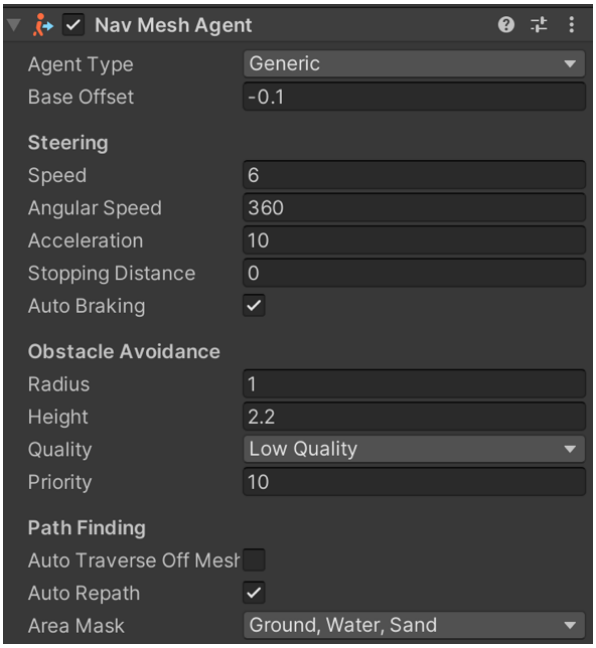

Abbildung 17: NavMeshAgent-Komponente

Die Verwendung eines sogenannten „Navmeshs“ ist vor allem bei Bewegungsabläufen ein sinnvolles Tool. Über „AI -> Navigation“ lässt sich das Navigations-Fenster öffnen, in welchem sich Einstellungen rund um mögliche Bewegungen machen lassen. Öffnet man das Tab „Areas“, so kann man dort die verschiedenen Bereiche sehen, die als Fläche festgelegt wurden. Jedes Game-Objekt, das sich auf einer Fläche bewegt, die als „Navmesh-Area“ definiert wurde, muss eine „NavMeshAgent-Komponente“ besitzen (s. Abbildung 17). In dieser kann im Bereich „Path Finding“ bei „Area Mask“ festgelegt werden, welche Bereiche für das Game-Objekt begehbar sind und welche nicht. Alle Objekte, die als Hindernisse fungieren und durch die nicht hindurchgegangen werden soll, müssen als „Obstacles“ markiert werden. Dies lässt sich realisieren, indem den jeweiligen Objekten eine „NavMeshObstacle-Komponente“ beigefügt wird. Sich bewegende Objekte werden beim Aufeinandertreffen mit einem „Obstacle“ einen neuen Weg suchen. Diese bereits eingebauten Möglichkeiten bieten einen entscheidenden Vorteil bei der Konfiguration von Bewegungsabläufen in Programmen.

3.4 ENTHALTENE SZENEN

Wie bereits im Vorherigen beschrieben, stellt eine Szene in Unity ein definiertes Setting oder eine Art Level dar. In der hier vorgestellten Arbeit gibt es zwei Szenen. Die erste besteht aus dem Startmenü, in welchem die Anzahl der Tiere festgelegt und der Start der Simulation ausgelöst werden kann, während die zweite Szene die Hauptszene darstellt, hier „SampleScene“ genannt. In der „SampleScene“ wurde die eigentliche Simulation erstellt. Genauere Informationen zum Startmenü sind im entsprechenden Kapitel zu finden.

3.5 KONFIGURATION DER TIERE

Da es in dieser Simulation um ein Ökosystem in Neuseeland geht, in dem sich auf die Ausbreitung der Possums und die damit verbundene Bedrohung für die Kiwis fokussiert wurde, sind für diese Arbeit nur der Kiwi und das Possum als Tierarten zum Einsatz gekommen. Der Kiwi stellt in diesem Kontext die Beute des Possums dar und das Possum nimmt die Rolle des Raubtiers ein. Nach umfassender Recherche wurden Unterschiede zwischen Kiwis und Possums herausgearbeitet und diese versucht

sinnvoll durch verschiedene Variablen und Verhaltensweisen in die Arbeit zu integrieren. Die diversen Unterschiede werden im Folgenden bei den einzelnen Punkten genauer betrachtet.

3.5.1 STEUERUNG UND BEWEGUNG

BEWEGUNG DURCH DEN EINSATZ EINES NAVMESHS

Die Bewegung der Tiere wird durch die Verwendung des Pathfinding-Tools „Navmesh“ möglich, dessen korrekte Anwendung und Einpflegung mithilfe eines YouTube-Tutorials zum Thema „NavMesh“ [35] verständlich wurde.

FESTLEGUNG BEGEHBARER OBJEKTE

Zunächst wurde die Karte („Ground“), auf der sich die Tiere fortbewegen sollen „gebaked“, sodass sie als begehbare Fläche fungieren kann. Wichtig dabei war es, den Haken bei „Navigation Static“ zu setzen, um festzulegen, dass das ausgewählte Objekt sich nicht selbst bewegen kann (s. Abbildung 18). Das gleiche Vorgehen wurde auch bei den einzeln hinzugefügten Objekten „Sand“ und „Water“ angewendet, um eine Unterscheidung dieser Bereiche zu gewährleisten, sodass die Möglichkeit des Laufens über bestimmte Flächen nach Bedarf deaktiviert werden kann.

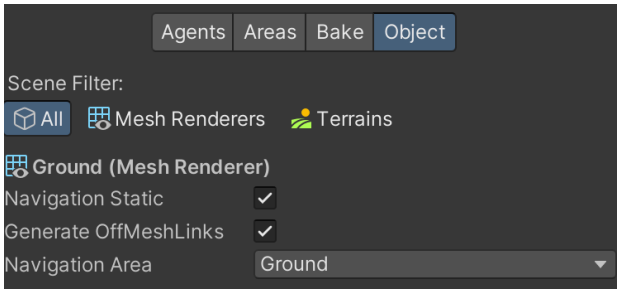

Abbildung 18: NavMesh-Optionen

KONFIGURATION DER SICH ZU BEWEGENDEN OBJEKTE

Das Objekt, welches sich bewegen soll, was in diesem Fall der Kiwi oder das Possum ist, erhält eine „NavMeshAgent-Komponente“, die per Skript angesprochen werden kann. Um ein „Navmesh“ im Skript nutzen zu können, muss die Codezeile „`using UnityEngine.AI;`“ am Anfang des Skripts hinzugefügt werden. Auf diese Weise kann z. B. ein Ziel definiert werden, indem man die „NavMeshAgent-Komponente“ in der Awake()-Methode einbettet und dann die „Destination“, also das Bewegungsziel des Objekts vorgibt. Die Geschwindigkeit oder auch die Beschleunigung der Tiere lassen sich ebenfalls in der „NavMeshAgent-Komponente“ im Inspector einstellen.

KONFIGURATION VON HINDERNISSEN („OBSTACLES“)

In der Simulation wurden außerdem alle Bäume als „Obstacles“ definiert, damit die Tiere diese umgehen und nicht einfach hineinlaufen. Eine Ausnahme bilden die Bäume in der Nähe des Wassers, welche schon vorher gestalterisch gesetzt wurden. Dort ist der Abstand an einigen Stellen zu gering, sodass die Tiere manchmal an den entsprechenden Punkten „hängenblieben“.

3.5.2 BEWEGUNGSZIELE UND ZUSTÄNDE

ZUSTANDSMASCHINE

Das Verhalten, das die Tiere ausführen wird über eine sogenannte Zustandsmaschine gesteuert. Tiere können sich in verschiedenen Zuständen befinden, welche über die Zustandsmaschine unter bestimmten festgelegten Voraussetzungen getriggert werden. Es handelt sich hierbei also um heuristische Abläufe. Die Inspiration zur Gestaltung des Themas „Zustandsmaschine“ stammt aus der Bachelorarbeit „Simulating an Ecosystem“ der „University of Gothenburg“ [39].

KONFIGURATION DER ZUSTÄNDE IM CODE

Grundsätzlich wurde eine Variable „needsThreshold“ erstellt, welche ab dem Wert 80 ein bestimmtes Bedürfnis, wie z. B. Hunger, auslöst. Sie legt den Wert, ab dem ein Bedürfnis entsteht, für jegliche Bedürfnisse fest. Welche Bedürfnisse als wichtiger angesehen werden, wird nach verschiedenen Präferenzen unterschieden. In der Methode „CheckNeeds()“, wird das Verhalten bei einem aufkommenden Bedürfnis entsprechend einer Rangfolge bestimmt.

Beim Festlegen des Zieles wird ein String-Wert mitgegeben, der die nächste Aktion des Tieres beschreibt, z. B. „Walk to nest“. Im jeweiligen Navigations-Skript des Tieres, wird überprüft, ob es sich bei der nächsten Aktion um das genannte „Walk to nest“ handelt und ob das Tier bereits an der gewünschten Position angekommen ist. Bei Übereinstimmung wird die Aktion über eine entsprechende Methode, z. B. beim Eierlegen die Methode „LayEgg“, ausgeführt.
Im Code sieht die Umsetzung beispielsweise folgendermaßen aus:

```
if (kiwi.NextAction == ("Walk to nest: " + nextTarget.name))
{
      StartCoroutine(kiwi.LayEgg(nextTarget));
}
```

ZUSTÄNDE

In der erstellten Simulation gibt es grundsätzlich folgende Zustände, die der untenstehenden Tabelle zu entnehmen sind:

	Kiwi	Possum
Walk to prefered food	•	•
Walk to normal food	•	•
Walk to water	•	•
Walk to mate	•	•
Randomwalk	•	•
Walk to nest	•	
Run (isFleeing)	•	
Swim	•	
Fight		•

Da beide Tiere ähnliche Bedürfnisse haben, sind die Möglichkeiten der Zustände, in denen sie sich befinden können, sehr ähnlich. Dennoch gibt es ein paar Variationen zwischen den möglichen Zuständen der beiden Tierarten, die in der obenstehenden Tabelle deutlich gemacht werden. Im Folgenden werden deshalb gemeinsame und tierartenspezifische Zustände getrennt behandelt.

GEMEINSAME ZUSTÄNDE

WALK TO PREFERED FOOD & WALK TO NORMAL FOOD

Die Zustände “Walk to prefered food” und “Walk to normal food” können zusammengefasst erklärt werden, da sie dieselben Voraussetzungen haben und lediglich in ihrer Rangordnung variieren. Ist ein Tier hungrig (der Sättigungswert entspricht der Zahl 80 oder kleiner) und das präferierte Futter befindet sich im Detektionsradius des Tieres, so wird es sich auf den Weg zu seiner Lieblingsnahrung machen und diese verzehren. Befindet sich allerdings nur normales Futter im Radius, so begibt sich das Tier zu dieser Art von Nahrung.

WALK TO WATER

Ist ein Tier hingegen durstig, so wird es sich auf den Weg zu einer Wasserquelle machen, insofern das Tier eine solche entdeckt hat. Das Trinken steht dem Aufsuchen von normalem Futter übergeordnet in der Rangordnung.

WALK TO MATE

Sind alle Bedürfnisse erfüllt oder es ist momentan trotz Hunger oder Durst nur ein potenzieller Partner in Sicht aber keine Nahrung oder Wasser und das Sozialbedürfnis ist hoch genug, so wird sich das Tier unter getroffenen Bedingungen mit dem detektierten Partner fortpflanzen.

RANDOMWALK

Sind alle Bedürfnisse des Tieres gestillt, so läuft das Tier in zufällige Richtungen, bis sich ein neues Bedürfnis entwickelt und der Vorgang von vorne beginnt.

SPEZIFISCHE ZUSTÄNDE DES KIWIS

Da der Kiwi das Beutetier ist, verfügt er zusätzlich über den Zustand „Run“ (Fliehen) und „Swim“. Die Möglichkeit, sich auf Wasserflächen aufzuhalten gibt dem Kiwi eine Art Schutzgebiet, auf dem er sicher vor dem Possum ist. Des Weiteren legen Kiwis Eier und haben aus diesem Grund noch den Zustand „Walk to nest“. Nachfolgend werden die Zustände erklärt, die lediglich den Kiwi betreffen.

WALK TO NEST

Ist ein Kiwi trächtig und bereit das Ei zu legen, so wird er sich, sobald er ein Nest entdeckt hat, auf den Weg dorthin machen, um das Ei abzulegen, unabhängig von seinen lebenserhaltenden Bedürfnissen.

RUN (FLIEHEN)

Die einzige Ausnahme ist der Zustand „Run“, der allem voran ausgeführt wird. Die höchste Priorität ist beim Kiwi demnach, sich vor dem Possum in Sicherheit zu bringen.

SWIM

Der Zustand „Swim“ wird ausgeführt, wenn der Kiwi unter sich Wasser detektiert. Hierfür wechselt er in die dafür vorgesehene Animation „Swim“, welche den Kiwi von seiner Position her ein wenig heruntersetzt, sodass es aussieht, als ob der Kiwi im Wasser schwimmt. Dieser Zustand wird immer getriggert, wenn sich eine Wasserfläche unter dem Tier befindet. Die Wasserfläche wird mithilfe eines Raycasts und dem Einsatz von „Layern“ ermittelt.

SPEZIFISCHE ZUSTÄNDE DES POSSUMS

FIGHT

Das Possum hat als alleinigen Zustand lediglich den Zustand „Fight“, der ausgeführt wird, sobald das Possum einen Kiwi, anstatt einer Pflanze frisst.

KONKRETER ABLAUF DES KIWIS

Am Beispiel des Kiwis sieht der Ablauf der Zustände folgendermaßen aus: Allem Voran bewegt sich der Kiwi nur zu einem bestimmten Ziel, wenn er nicht vor einem Possum fliehen muss. Ansonsten ist die erste Priorität des Kiwis, sollte er trächtig sein und bereit das Ei zu legen, ein Nest zu finden, an

dem er das Ei ablegen kann. An nächster Stelle steht das Suchen nach Nahrung, wenn der Sättigungswert kleiner oder gleich 80 ist. Hierbei wird sich der Kiwi zuerst zu seinem Lieblingsfutter bewegen. Zunächst wird überprüft, ob der Kiwi Durst hat. Ist dies der Fall, wird er sich, bei Entdecken einer Wasserquelle, zu dieser bewegen, um zu trinken. Erst danach werden weniger gewünschte Futterarten (Pflanzen und Pilze) als Nahrungsquelle angesteuert. Sind alle lebenserhaltenden Bedürfnisse erfüllt und das Sozialbedürfnis ist kleiner oder gleich dem Wert 80, ist der Kiwi empfänglich für einen möglichen Partner.

KONKRETER ABLAUF DES POSSUMS

Beim Possum verhält sich der Ablauf sehr ähnlich, mit dem Unterschied, dass das Possum keine Eier legt und auch keine Feinde hat, vor denen es fliehen muss. Deshalb beginnt das Possum, wenn es hungrig ist, direkt seine Lieblingsnahrung zu suchen. Possums begeben sich zuerst auf die Suche nach Kiwi-Babys oder Kiwi-Eiern, da diese noch verletzlicher sind. In dieser Simulation wurde diese Präferenz berücksichtigt.

Eine Visualisierung dieser Abläufe, die auch die Präferenzen veranschaulicht, ist dem folgenden Zustandsdiagramm (s. Abbildung 19), welches mit „draw.io“ erstellt wurde, zu entnehmen:

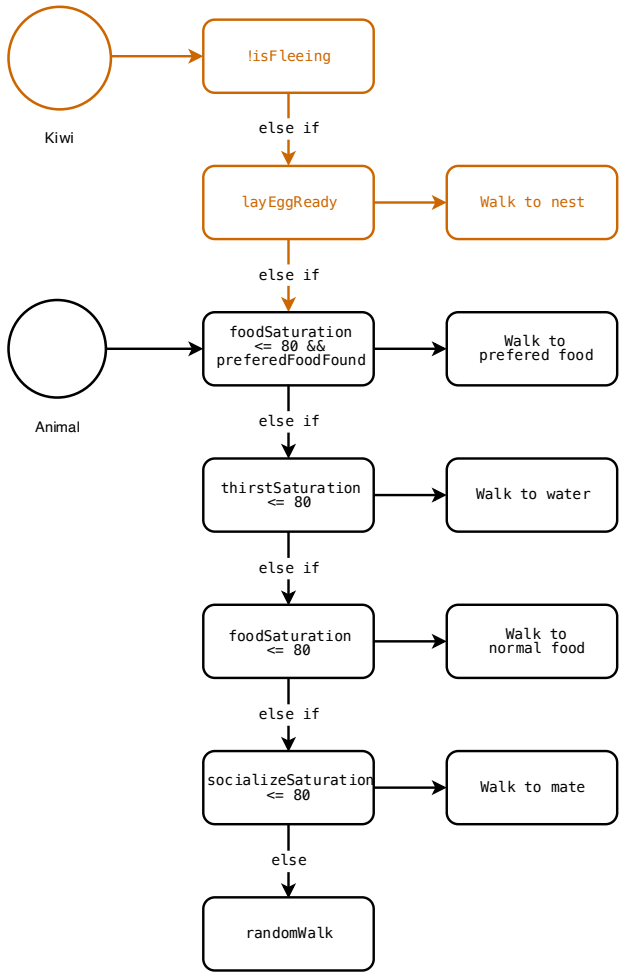

Abbildung 19 : Zustandsdiagramm

3.5.3 OBJEKTDETEKTION (SIGHT & SMELL)

Zunächst mussten Überlegungen angestellt werden, auf welche Weise Objekte von den Tieren erkannt werden. Hier wurde sich am Prinzip der Natur orientiert, in der Nahrung, Feinde oder potenzielle Partner meist über das Sehen oder Riechen aufzufinden sind. Um passende Einstellungen vornehmen zu können war eine Recherche nötig, die Aufschluss darüber gab, wie ausgeprägt der Seh- und Geruchssinn der in dieser Simulation relevanten Tiere ausfällt. Es wurden entsprechend zwei Arten von Collidern am Tier-Prefab angebracht, die zum einen das Sichtfeld und zum anderen das Geruchsfeld darstellen sollen.

SICHTFELD (SIGHT)

Ein in Blender erstellter Halbkreis, welcher sich direkt vor dem Tier befindet und als Collider fungiert, stellt die Sicht („Sight“) des Tieres dar (s. Abbildung 20). Objekte, welche über die Sicht aufgenommen werden sollen, wurden vorher, mithilfe der Methode „OnSight()“ definiert. Es wurde eine Liste von Game-Objekten, namens „objectsInSightRange“ erstellt, zu der jegliches Objekt, dessen Tag mit der if-Abfrage in der Funktion „OnSight()“ übereinstimmte, hinzugefügt wurde. Das Erkennen der Objekte funktioniert an dieser Stelle über „OnTriggerEnter“, wenn das Objekt auf den Collider trifft. Die zur „Sight“ gehörende Methode „OnTriggerEnter“ und „OnTriggerExit“ sind jeweils für jedes Tier in einem zusätzlichen Skript definiert („KiwiSight.cs“, „PossumSight.cs“), welches der Collider zugeordnet bekommt. Befindet sich das Objekt nicht mehr im Radius des Sichtfelds, so wird es wieder aus der Liste gelöscht.

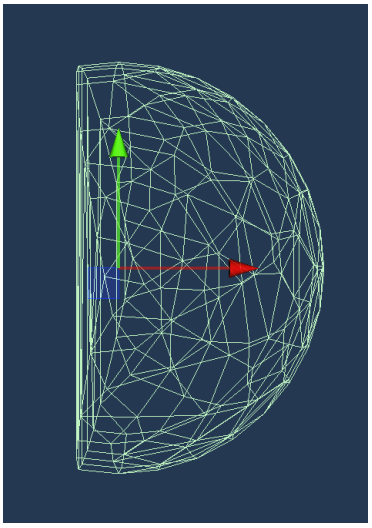

Abbildung 20: Sight-Collider

GERUCHSFELD (SMELL)

Der „Smell-Collider“ hat die Form eines Kreises (s. Abbildung 21), der das Tier umschließt und dadurch einen Radius um das Individuum herum bildet. Die Funktionsweise ist die gleiche, wie die des „Sight-Colliders“, mit dem Unterschied, dass genannte Methoden und Listen hier „OnSmell()“ und „objectsInSmellRange“ heißen. Die „Smell-Range“ des Possums ist ein wenig größer, als seine „Sight-Range“.

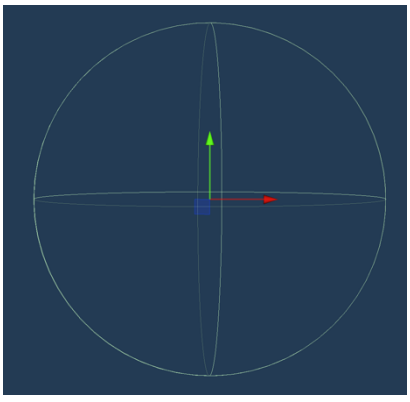

Abbildung 21: Smell-Collider

SMELL UND SIGHT IM VERGLEICH

SMELL UND SIGHT BEIM KIWI

Dem Kiwi wurde aufgrund seiner schlechten Sehfähigkeiten [26] ein sehr kleines Sichtfeld zugeordnet (s. Abbildung 22). Er nimmt jegliche Nahrung, Wasserquellen und Nester über sein Sichtfeld auf.

Gravierendere Größenunterschiede gibt es jedoch beim Geruchsfeld des Kiwis im Vergleich zum Possum. Aufgrund seines besonders guten Geruchssinns [26] wurde dem Kiwi eine sehr große „Smell-Range“ zugeteilt. Er nimmt über den Geruchssinn mögliche Partner und Feinde wahr. Welche Objekte genau über welchen Collider aufgenommen werden, ist in Abbildung 23 zu sehen. Durch die größere „Smell-Range“ erkennt er Feinde demzufolge schnell über den Geruchssinn. Um einen natürlichen Faktor einzufügen, wurde allerdings programmatisch festgelegt, dass der Kiwi zu 20% nicht vor seinem erkannten Feind flieht. Dies sollte den Eindruck erwecken, dass der Kiwi den Feind nicht immer zuverlässig „erschnüffeln“ kann. Hierfür wurde die von Unity bereitgestellte Methode „Random.Range()“ genutzt, in der man an erster Stelle in den Klammern den Minimumwert und an zweiter Stelle den Maximalwert einfügen kann. Es wird dann eine Zufallszahl zwischen diesen beiden Werten generiert. Im Code sieht die Abfrage, die ein Fliehen bei Wahrheit ermöglicht, folgendermaßen aus:

```
if (Random.Range(0, 10) >= 2 && other.CompareTag("Possum")) {…}
```

Es wird ein Random-Wert ermittelt. Wenn dieser größer oder gleich der Zahl 2 ist und das entdeckte Objekt den Tag „Possum“ hat, dann wird der Kiwi fliehen. Entspricht der Random-Wert der Zahl 0 oder 1, was bei einer Random-Range von 0-10, die die Zahl 10 aber ausklammert, 20% ausmacht, dann wird der Kiwi nicht fliehen.

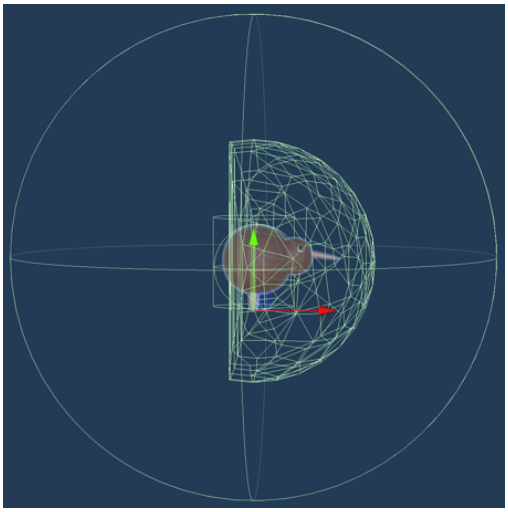

Abbildung 22: Smell-/ Sight-Range des Kiwis

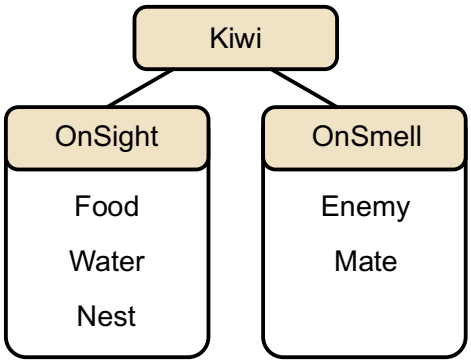

Abbildung 23: Detektion der Collider des Kiwis

SMELL UND SIGHT BEIM POSSUM

Das Possum, welches keine Beeinträchtigung beim Sehen aufweist, hat hingegen ein größeres Sichtfeld erhalten (s. Abbildung 24). Auch das Possum speichert jegliche Wasserquellen und Nahrungs-Objekte, zu denen auch die Kiwis und deren Eier gehören, in die Liste des Sichtfelds. Partner werden über den Smell-Collider entdeckt. Eine Gegenüberstellung der zu detektierenden Objekte über den jeweiligen Collider ist der Abbildung 25 zu entnehmen.

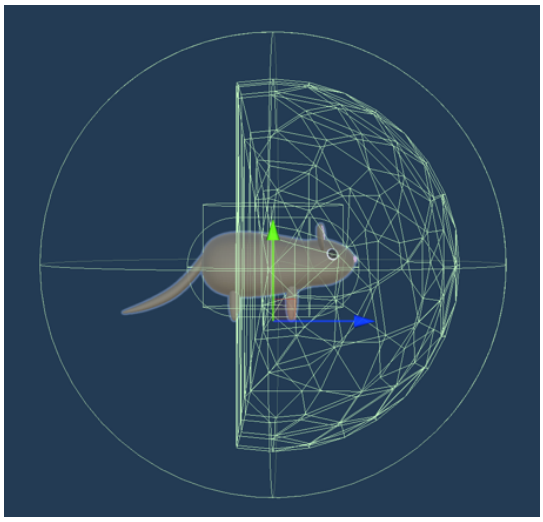

Abbildung 24: Smell-/Sight-Range des Possums

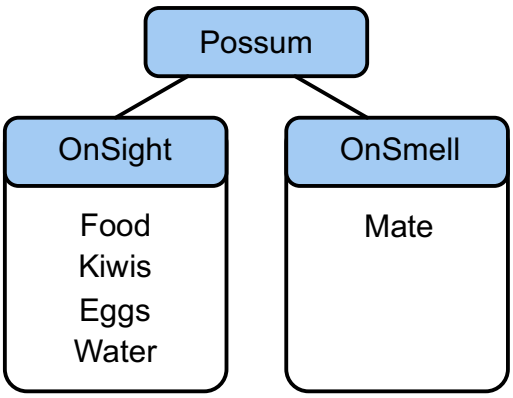

Abbildung 25: Detektion der Collider des Possums

3.5.4 GENE

Bei der Instanziierung von Kiwis oder Possums, werden jedem einzelnen Tier bestimmte Werte und Eigenschaften zugeordnet. Darunter fallen z. B. das Geschlecht, der Name, die Attraktivität und zufällig generierte Bedürfniswerte. Diese Eigenschaften, die jedes Tier individuell für sich bei jeder Geburt oder Erschaffung mitbekommt, kann man als Gene bezeichnen. Die verschiedenen „Gene“ werden über das Skript „Spawner.cs“, in welchem das Instanziieren von statten geht, mitgegeben.

NAMEN

Um die Tiere im Programm besser unterscheidbar zu machen und als kleiner „Fun Fact“ wurden dem Spawner-Skript in Form eines Arrays zwei Namenslisten hinzugefügt, die eine Unterscheidung zwischen männlichen und weiblichen neuseeländischen Vornamen macht. Die Namensliste stammt aus folgender Quelle: [40]. Je nach Geschlechtstyp wird einem Tier entweder ein zufälliger weiblicher oder männlicher Name zugeteilt. Die Zufallsgenerierung lässt sich erneut über „Random.Range()“ erzeugen, allerdings wird als Maximalwert diesmal die Länge der jeweiligen Namensliste mitgegeben. Der zufällig erzeugte Name wird beim Klick auf ein Tier im „AnimalMenu“, dem Menü, das die Eigenschaften eines einzelnen Tieres offenlegt, angezeigt.

BEDÜRFNISSE

Damit die Tiere verschiedene Aktionen ausführen, ist die Mitgabe von Bedürfnissen nötig. Diese steuern das Verhalten der Tiere, je nach aktuellem Bedürfnis und Präferenz. In dieser Simulation hat jedes Individuum die Bedürfnisse Hunger, Durst und das Bedürfnis nach sozialem Kontakt. Eine weitere Variable „health“ wurde für den Gesundheitszustand des Tieres definiert. Diese ist hier allerdings nur für Possums relevant, die beim Verzehr eines giftigen Pilzes (in Unity „toxic plant“ genannt) über die Zeit an Gesundheit verlieren. Der „needsThreshold-Wert“ legt fest, ab wann ein Tier ein Bedürfnis nach etwas hat. Voreingestellt liegt dieser Wert bei 80, allerdings kann er im Inspector beliebig angepasst werden. Um sich ein wenig dem Realistischen anzunähern, wurden wie in der Natur, in der jedes Individuum unterschiedlich ist, verschiedene Zufallswerte für die Bedürfnisse der Tiere erschaffen. Diese werden bei der Initialisierung der Tiere erstellt. So hat z. B. ein Tier etwas schneller Hunger oder Durst, als das andere. Um dies zu realisieren, wurden jedem Tier die nötigen Variablen (z. B. „foodSaturationLoss“, „thirstSaturationLoss“…) im jeweiligen Skript des Tieres mitgegeben. Dem „Spawner“-Skript, in welchem die Erstellung eines Tieres gesteuert wird, wurden festgelegte Minimum- und Maximalwerte für diese Variablen als „Serialize Fields“ zugeteilt. Da man „Serialize Fields“ im Inspector jederzeit bearbeiten kann, ist so die Steuerung der jeweiligen Werte einfach anpassbar. Durch die Methode „Random.Range()“ werden die Zufallswerte zunächst generiert.

ATTRAKTIVITÄT

Bei der Erstellung eines Tieres werden bestimmte Attraktivitätswerte mitgegeben. Darunter fällt die „attractiveness“, welche den Attraktivitätswert des Tieres beschreibt und die „minAttractiveness“, welche die Mindestanforderung an Attraktivität, die für die Zulassung der Paarung nötig ist, darstellt. Die „minAttractiveness“ ist nur bei weiblichen Tieren interessant, da diese Partner ablehnen, wenn dessen Attraktivitätswert kleiner ist, als der gewünschte. Die Werte werden, genau wie die Bedürfnisse, mit „Random.Range()“ ermittelt. Dabei beträgt die Spanne 0-10. Da die 10 nicht mit eingeht, ist der Wert 9 der Maximalwert an Attraktivität. Besitzt ein Männchen den Wert 9, so ändert sich sein Erscheinungsbild in der Simulation. Ein Vergleich zwischen einem normalen und einem attraktiven Kiwi (s. Abbildung 26) und einem normalen und attraktiven Possum (s. Abbildung 27) wird in den nachfolgenden Abbildungen deutlich gemacht.

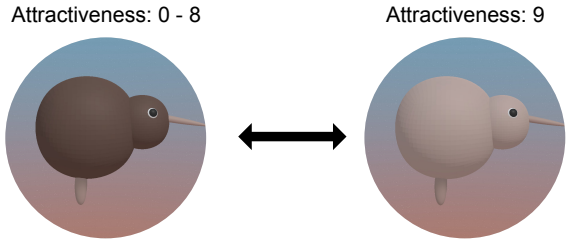

Abbildung 26: Vgl. normaler und attraktiver Kiwi

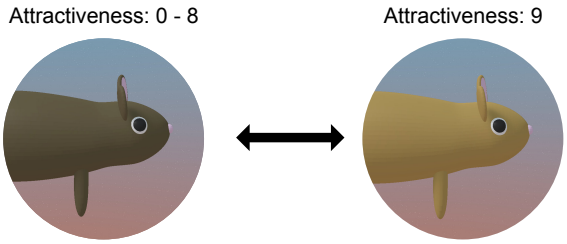

Abbildung 27: Vgl. normales und attraktives Possum

Die Änderung des Aussehens wird über das Zuweisen eines anderen Materials erreicht. Dies sieht im Code folgendermaßen aus:

```
if (gender.Equals("male") && attractiveness == 9)
{
	this.gameObject.GetComponentInChildren<SkinnedMeshRenderer>().materials =
	kiwiAttSkin;
}
```

Attraktive Männchen haben den Vorteil, nie von einem Weibchen abgelehnt zu werden, da sie ausnahmslos den Attraktivitätswert 9 aufweisen, welcher immer größer oder gleich dem Wert der „minAttractiveness“ ist.

3.5.5 PAARUNG

POTENZIELLEN PARTNER FINDEN

Wenn sich männliche und weibliche Tiere der gleichen Art begegnen, ist eine potenzielle Paarung möglich. Dafür ausschlaggebend ist, dass die Tiere sich gegenseitig über den „Smell-Collider“ erkannt haben. Ist dieser Schritt erfolgt, so nimmt jedes Tier für sich eine Prüfung vor, ob das erkannte potenzielle Paarungsobjekt auch wirklich zur Fortpflanzung geeignet ist.

Dies erfolgt im Code über mehrere Schritte. Zuerst werden die wichtigsten Bedingungen für eine Paarung über eine boolesche Methode überprüft, die sich „private bool isPotentialPartner()“ nennt. Hier wird gecheckt, ob es sich beim erkannten Objekt um dieselbe Tierart handelt, das Geschlecht das Gegenstück bildet, das andere Tier sich nicht gerade schon paart, sich vom Paarungsvorgang ausruht oder bereits schwanger ist und ob es sich um ein erwachsenes Tier handelt. Kiwis überprüfen noch, ob das andere Tier gerade vor einem Feind flieht, da dies Priorität hätte. Eine Übersicht dieser Überprüfungen für jede Tierart ist der nachfolgenden Tabelle zu entnehmen:

Kiwi	Possum	Bedeutung
`other.CompareTag(„Kiwi“)`	`other.CompareTag(„Possum“)`	Gleiche Tierart
`gender != otherKiwi.Gender`	`gender != otherPossum.Gender`	Untersch. Geschlecht
`!otherKiwi.isMating`	`!otherKiwi.isMating`	Paart sich nicht bereits
`!otherKiwi.isPregnant`	`!otherKiwi.isPregnant`	Ist nicht schwanger
`!otherKiwi.onMatingCoolDown`	`!otherKiwi.onMatingCoolDown`	Ist nicht am Ausruhen
`otherKiwi.isAdult;`	`otherKiwi.isAdult;`	Ist erwachsen
`!otherKiwi.isFleeing`		Flieht nicht gerade

Sind all diese Bedingungen erfüllt, erfolgt noch die gesonderte Prüfung von Männchen und Weibchen. Weibchen überprüfen, ob das Männchen den angeforderten Attraktivitätswert besitzt und Männchen checken, ob das erkannte Objekt sie schon einmal abgelehnt hat. Bei jeder Ablehnung speichert das männliche Tier das Weibchen, das es abgelehnt hat, in die Liste „rejectedFromFemale“. Das Männchen vergewissert sich demzufolge, ob das erkannte Weibchen sich nicht bereits in der Liste „rejectedFromFemale“ befindet, um einen weiteren vergeblichen Paarungsversuch zu vermeiden. Beim ersten Versuch, sich mit einem Weibchen zu paaren, das höhere Attraktivitätsanforderungen an das Männchen hat, als dieses besitzt, verliebt sich das Männchen kurz und startet einmalig einen Versuch sich zu paaren. Das Weibchen zieht allerdings von dannen und es kommt keine Paarung zustande.
Schließlich werden bei Erfüllung aller genannter Kriterien, die detektierten Tiere zur Liste „potentialPartners“ hinzugefügt und über die Methode „GetPotentialPartners()“ zurückgegeben.

PAAURNGSAKT

Haben sich zwei Tiere als Partner zur Fortpflanzung gefunden, laufen sie kurz zueinander und vollziehen den Paarungsakt. Sichtbar wird dies an einem Herz-Symbol über den beiden sich paarenden Tieren (s. Abbildung 28). Das Symbol wurde in Blender erstellt und wird je nachdem, ob die Variable „isMating“ „true“ oder „false“ ist auf aktiv oder inaktiv gesetzt. Nachdem der Paarungsakt abgeschlossen ist, wird sich das Männchen für eine bestimmte Zeit erholen, indem die Variable „onMatingCoolDown“ auf „true“ gesetzt wird. Das Weibchen verlässt den Paarungsakt mit dem Status „isPregnant“. Wie die Dauer der bestimmten Stadien erschaffen wurde, wird im Folgenden genauer erklärt.

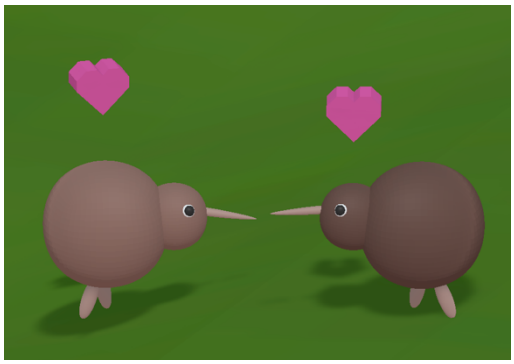

Abbildung 28: Paarungsakt

BERECHNUNG VON WERTEN RUND UM DIE PAARUNG

Den Fortpflanzungsakt betreffend, gibt es verschiedene Wartezeiten, die im Folgenden genauer erläutert werden. Um Fakten, die recherchiert wurden, einfließen zu lassen, wurden für die unterschiedlichen Wartezeiten Berechnungen angestellt, die die recherchierten Werte im Verhältnis widerspiegeln. Zeiten mussten für die folgenden Variablen festgelegt werden:

`onMatingCoolDown`	Die Zeit, bis zur nächsten Paarungswilligkeit (Ausruhzeit)
`pregnancyTime`	Die Zeit, bis ein weibliches Tier ein Ei legt oder Nachwuchs wirft
`layEggTime`	Die Zeit, in der der Kiwi ein Nest suchen kann. Erreicht der Kiwi das Nest nicht in dieser Zeit, so legt er das Ei an Ort und Stelle ab.
`hatchingTime`	Die Zeit, bis ein junger Kiwi aus dem Ei schlüpft
`growUpTime`	Die Zeit, bis ein Junges erwachsen wird und die volle Größe erreicht

Um diese Zeiten an realistischen Gegebenheiten auszurichten, wurden die in der untenstehenden Tabelle aufgeführten Unterschiede bezüglich Possums und Kiwis recherchiert und abgewandelt, bzw. angepasst (erkennbar an Blaugeschriebenem) in die Simulation integriert. Eine Gegenüberstellung dieser Werte, sind folgender Tabelle zu entnehmen:

	Kiwi	Possum
Geschlechtsreif Für Simulation: Erwachsen = geschlechtsreif	Weibchen: ca. 3 Jahre, Männchen: 18 Monate [28] Für Simulation: Verhältnis 3/2 zu Possums (22,5 Sekunden, entspricht 1,5 Monaten)	12 Monate [24] Für Simulation: Verhältnis 2/3 zu Kiwis (15 Sekunden, entspricht 1 Monat)
Trächtigkeitsdauer	1 Monat (30 Tage) [28] Für Simulation: (15 Sekunden, entspricht einem Monat)	16 - 18 Tage [24] Für Simulation: Auf 2 Wochen gerundet = 7,5
Wurf / Eier	1-2 Eier [25] Brutzeit: 10 Wochen [26] / 70 - 80 Tage [28] Eier pro Jahr: ca. 6 [29] Für Simulation: 6 Eier pro Jahr	1 oder seltener 2 [18, S. 2] (ca. 1-2 pro Jahr) Für Simulation: 1-2

Die Informationen stammen aus verschiedenen Quellen, deren Verweise in der Tabelle hinter der jeweiligen Information zu finden sind.

Um die Werte nun sinnvoll in die Simulation einzubauen, wurde eine Zeit, die einem Jahr entspricht („SECONDSPERYEAR“) festgelegt. Diese Variable befindet sich im Skript „GameVariables“, welches Variablen beinhaltet, die skriptübergreifend zugänglich sind. Zunächst wurden 360 Sekunden als Dauer für ein Simulationsjahr festgelegt. Dieses wurde zur Beschleunigung nochmal durch 2 geteilt, sodass ein Jahr in der Simulation 180 Sekunden entspricht.

ERKLÄRUNG DER WERTE DES KIWIS

Die Variablen für den Kiwi wurden wie folgt berechnet:

```
pregnancyTime = GameVariables.SECONDSPERYEAR / 6f * (2f/4f); -> 15
layEggTime = GameVariables.SECONDSPERYEAR / 6f * (1f/4f); -> 7,5
onMatingCoolDown = GameVariables.SECONDSPERYEAR / 6f * (1f/4f); -> 7,5

hatchingTime = GameVariables.SECONDSPERYEAR * (2.5f / 12f) / 2 -> 18,75
growUpTime = GameVariables.SECONDSPERYEAR / 8f; -> 22,5
```

Zunächst sind drei Werte gegeben, die sinnvoll aufgeteilt werden mussten und die die Zeit bis zur nächsten möglichen Paarung beeinflussen: Die „pregnancyTime“, welche 2/4 entspricht, die „layEggTime“, welche 1/4 entspricht und der „onMatingCoolDown“, welcher ebenfalls 1/4 ausmacht.

pregnancyTime

Um die Zeit der Trächtigkeit zu ermitteln, wurde die Zeit eines Jahres (180 Sekunden) durch die Anzahl der Eier, die ca. pro Jahr gelegt werden (6) geteilt und mit den oben beschriebenen 2/4 multipliziert. Als Ergebnis bekommt man die Zahl 15, welche in dieser Simulation einem Monat, also auch der laut Recherche [28] genannten Trächtigkeitszeit des Kiwis entspricht.

layEggTime & onMatingCoolDown

Bei der „layEggTime“ und beim „onMatingCoolDown“ wurde genauso vorgegangen, mit dem Unterschied, dass mit 1/4 multipliziert wurde. Das Ergebnis beträgt 7,5, was zwei Simulationswochen entspricht.

hatchingTime

Die Zeit, bis ein Kiwi schlüpft liegt laut Recherche [28] zwischen 70 und 80 Tagen. Es wurde aufgrund dessen der Mittelwert 75 gewählt, was ca. 2,5 Monaten entspricht. Darum wurde die Zeit des Jahres multipliziert mit (2,5 / 12), was 37,5 zurückliefert. Da dieser Wert in der Simulation zu lange wäre, um dem Nachwuchs des Kiwis auf der begrenzten Karte eine Chance zu gewähren zu schlüpfen, bevor die Eier gefressen werden, wurde der Wert noch einmal durch zwei geteilt. Das Endergebnis beträgt 18,75 Sekunden.

growUpTime

Bei der „growUpTime“ wurde sich auf die männliche Geschlechtsreifezeit fokussiert, die ca. 18 Monate betragen würde. Da dies in der Simulation zu lange wäre, wurden eineinhalb Monate daraus gemacht. Durch Teilen der Zahl eines Jahres durch die Zahl 8 ergibt sich genau diese Zeit (22,5). Da es drei Wachstumsphasen gibt, werden diese Werte in der Methode „GrowUp()“ noch einmal durch drei geteilt.

ERKLÄRUNG DER WERTE DES POSSUMS

Die Variablen für das Possum ergaben sich auf folgende Weise:

```
pregnancyTime = GameVariables.SECONDSPERYEAR / 12f / 2f; -> 7,5
onMatingCoolDown = GameVariables.SECONDSPERYEAR / 1,36f – 7,5f; -> 124,85..

growUpTime = GameVariables.SECONDSPERYEAR / 8f * (2f/3f); -> 15
```

pregnancyTime

Die Trächtigkeitsdauer sollte beim Possum ca. 2 Wochen betragen, da dies ähnlich in der Realität aussieht. Aus diesem Grund wurde die Zahl eines Simulationsjahres durch 12 und dann nochmal durch 2 geteilt um den Wert 7,5 (entsprechend 2 Simulationswochen) zu erhalten.

onMatingCoolDown

Damit das Possum, wie recherchiert, nur 1-2 Babys im Jahr bekommt [18, S. 2], fiel der „onMatingCoolDown“-Wert besonders groß aus. Er ergibt sich aus der Differenz der Würfe pro Jahr und der Trächtigkeitszeit. Um die Würfe pro Jahr zu ermitteln, wurde wie folgt vorgegangen: Es wurden die durchschnittlichen Kinder pro Jahr (1,5) durch die durchschnittliche Anzahl an Kindern pro Wurf mit Einbezug der Wahrscheinlichkeiten, die im Abschnitt „Nachwuchs“ beschrieben sind (1,10) geteilt.

growUpTime

Beim Possum wurde die „growUpTime“ genauso berechnet wie beim Kiwi, nur noch zusätzlich mit 2/3 multipliziert, sodass das Possum um 2/3 schneller heranwächst, also in 15 Tagen erwachsen wird, was hier einem Monat entspricht. Auch hier werden diese Werte in der Methode „GrowUp()“ noch einmal durch drei geteilt.

Durch diese Art der Zeitberechnung ist es möglich, die Simulation zu beschleunigen, um lange Wartezeiten zu vermeiden und schneller Ergebnisse zu erhalten. Dies lässt sich umsetzen, indem man im Skript „GameVariables“ die Zeit eines Jahres durch einen gewünschten Wert teilt. So würden sich alle Werte verkleinern und dadurch würden die Ereignisse in der Simulation schneller von statten gehen, wobei die Werte trotzdem im Verhältnis zueinander gleichbleiben.

EINBAUEN VON WARTEZEITEN

Wartezeiten lassen sich über eine „IEnumerator“-Methode, welche eine Coroutine ermöglicht, realisieren. In dieser lässt sich der Befehl „yield return new WaitForSeconds()“, bei der man die Wartezeit in der Klammer einträgt, ausführen.

TRÄCHTIGKEIT

Kam es zu einem Paarungsakt, so verlässt das Weibchen diesen mit dem Zustand „pregnant“. Ist ein Tier erfolgreich schwanger geworden, so wird dies an einem im Audiomanager konfigurierten Sound erkennbar. Zunächst läuft die Zeit der Trächtigkeit ab. Nach Ablauf dieser Phase, unterscheiden sich die Vorgänge zwischen Possum und Kiwi wie folgt: Das Possum ruft die Methode „GenerateBabies()“ auf, in der Nachwuchs erzeugt wird, während der Kiwi die Methode „LayEgg()“ aufruft, in der er beginnt nach einem Nest zu suchen. Nester werden über die Sicht des Kiwis erkannt und speichern sich beim Auffinden eines solchen in die „objectsInSightRange“-Liste. Hat ein Kiwi einmal ein Nest entdeckt, so bleibt dieses in seiner Liste gespeichert und er wird nach Ablauf der Trächtigkeitszeit

dieses Nest sofort finden. Nach dem das Ei abgelegt wurde und Nachwuchs bei den Possums entstand, erholen sich auch die Weibchen in einer „onMatingCoolDown“-Phase.

NACHWUCHS

Der Nachwuchs entsteht bei den Possums sofort nach der Trächtigkeitsphase, während beim Kiwi erstmal ein Ei gelegt wird. Das Kiwi-Ei ist nun aus programmatischer Sicht für sich selbst zuständig, denn es wird über ein Ei-Skript „Egg.cs“ gesteuert. Sobald das Ei-Objekt in der Simulation auftaucht, beginnt mit dem Aufruf der Methode „StartHatching()“ der Ablauf der Zeit, bis der Kiwi aus dem Ei schlüpft. In der Funktion selbst, wird zunächst die Position des schlüpfenden Kiwis („childSpawnPos“) auf die Position des Eies gesetzt. Anschließend werden, nach Ablauf der „hatchingTime“, neue Kiwis mit folgendem Befehl instanziiert:

```
InitiateKiwiSpawn(childSpawnPos, 0.45f, false);
```

Der oben stehenden Methode wird die Position des Babys, die Größe des Babys, welche der Hälfte eines erwachsenen Tieres entspricht und der Status „Kind“ mitgegeben. Ist der Kiwi geschlüpft, so wird das Ei zerstört.

Beim Possum verhält sich dieser Vorgang sehr ähnlich. Mit der Methode „GenerateBabies()“ wird nach Ablauf der Trächtigkeitszeit der Possum-Nachwuchs auf die gleiche Art und Weise erschaffen. Das zu generierende Baby bekommt hier allerdings die Position seiner Mutter. Als Unterschied ist hier jedoch zu nennen, dass die Anzahl des Nachwuchses, in Abhängigkeit von verschiedenen Wahrscheinlichkeiten, variiert. Da ein Possum laut Recherche meist ein Baby oder selten zwei Babys zur Welt bringt [18, S. 2], wurden die Wahrscheinlichkeiten entsprechend angepasst. Dies wurde mit der Methode „RandomValues()“, die sich im „Spawner.cs“-Skript befindet, realisiert. Im Code sieht die Ermittlung der Anzahl zu gebärender Kinder folgendermaßen aus:

```
int children = sceneManager.GetComponent<Spawner>().RandomValues(1, 1, 2, 0);
```

Die in der Klammer stehenden Werte bedeuten, dass zu 40% und zu 30% ein Kind geboren wird, was in der Summe eine Wahrscheinlichkeit von 70% ausmacht. Zu 20% bekommt das Possum 2 Kinder und zu 10% hat das Possum eine Fehlgeburt. Die Wahrscheinlichkeiten sind in der Funktion „RandomValues()“ festgelegt, auf die später noch einmal genauer eingegangen wird.

WACHSTUM

Nachdem die Babys auf der Welt sind, erscheinen sie in Miniaturform eines erwachsenen Tieres. Um das Wachstum natürlicher wirken zu lassen, wurden hier die Wachstumsphasen in drei Schritte untergliedert. Für das Wachstum ist die IEnumerator-Funktion „GrowUp()“ zuständig. In dieser Methode werden durch eine for-Schleife drei Durchläufe erzeugt, in denen das Kind wächst. Die „growUpTime“ legt fest, wie viele Sekunden jede der drei Wachstumsphasen dauert. Da ein Baby die

Hälfte der Größe eines erwachsenen Tieres hat, muss es um das Doppelte heranwachsen. Dieser Wert wird durch 3 geteilt, um die Größe, um die das Tier pro Wachstumsphase wächst, zu ermitteln. Am Beispiel eines Possum-Babys sieht das wie folgt aus:
Das Possum-Baby hat eine Größe von 0.3. Das Kind wächst demzufolge 3x alle 10 Sekunden (definiert durch die „growUpTime“) um 0.15. Die Größe eines Objekts kann man in Unity über den Befehl „`transform.localScale`“ verändern. Nachdem die Wachstumsphase abgeschlossen ist, ändert sich der Status des Kindes zu „Adult“, also erwachsen.

3.5.6 NAHRUNGSQUELLEN

Es gibt verschiedene Nahrungsquellen in der Simulation, die unterschiedlich von den beiden Tierarten behandelt werden. Als generierbares Futter gibt es Pflanzen, Pilze und Würmer. Würmer stellen das Lieblingsfutter des Kiwis dar [27] und werden nur von Kiwis gefressen. Pflanzen können beide Tierarten als normales Futter aufnehmen. Das Possum hat, wie schon oben beschrieben, noch zusätzlich den Kiwi und die Eier des Kiwis als Nahrungsquelle. Eine Übersicht über die Futterarten, die jedes einzelne Tier aufnehmen kann, sieht man in der nebenstehenden Abbildung (s. Abbildung 29).

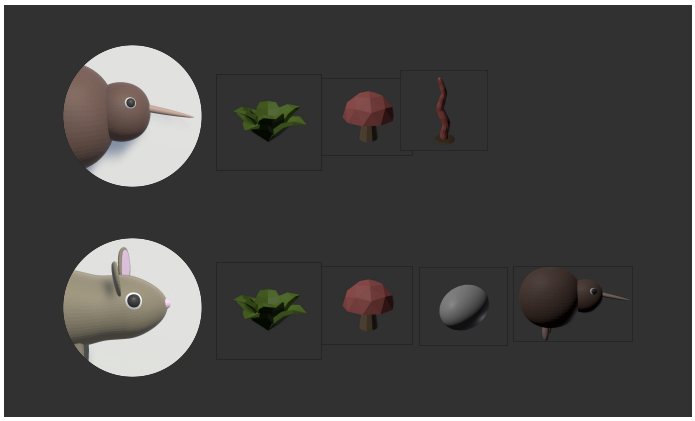

Abbildung 29: Unterschiede der Nahrungsquellen

GIFTIGE PILZE

Auch Pilze können von beiden Tieren gefressen werden, unterscheiden sich jedoch in den Auswirkungen. Frisst das Possum einen Pilz, so wird dieses vergiftet oder infiziert und mit der Zeit an der Vergiftung sterben, da der Gesundheitswert ab der Vergiftung nach und nach sinkt. Ein vergiftetes Possum ist erkennbar durch das lila Virus-Symbol im „AnimalMenu“, das aufpoppt, wenn man auf das Possum klickt. Diese Art die Possums zu vergiften, sollte es den Possums in der Simulation etwas erschweren zu überleben und auch eine Art Possum-Bekämpfung, die in der Realität durch z. B. Giftköder [3, S. 20] eingeleitet wird, darstellen. Der Kiwi kann den giftigen Pilz fressen, ohne zu erkranken, damit er eine weitere Nahrungsquelle hat und einen kleinen Vorteil gegenüber dem Possum gewinnt. Der Einsatz von Giftpilzen entspricht in dieser Form nicht der Realität, aber sollte der Possum-Bekämpfung ähneln und dem Possum eine Erschwernis beifügen.

WASSER

Wasserquellen lassen sich über eingebaute Wasser-Collider von den Tieren detektieren, wenn sich das Tier in der Nähe der Collider befindet. Diese wurden mithilfe von Würfeln („Cubes“), welche kein Material aufweisen, damit sie im fertigen Programm unsichtbar sind, um die Seen herum platziert, wie in nebenstehender Abbildung (s. Abbildung 30) zu erkennen ist.

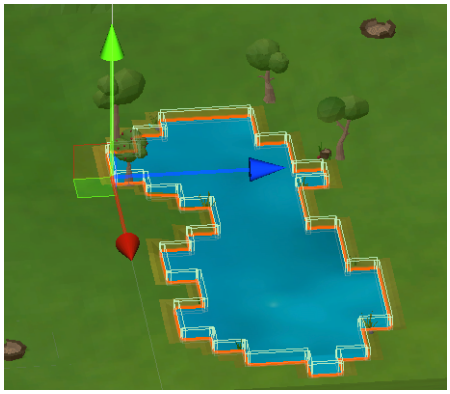

Abbildung 30: Wasser-Collider

3.5.7 TOD

UNTERSCHIEDE VON POSSUM UND KIWI

Das Sterben von Tieren ist ebenfalls je nach Tierart unterschiedlich zu betrachten. Während der Kiwi zusätzlich zu einem natürlichen Tod auch vom Possum gefressen werden kann, kann das Possum im Gegensatz zum Kiwi, an einem giftigen Pilz erkranken und daran sterben. Eine andere Möglichkeit ist der Tod aufgrund von mangelnder Nahrung oder Durst.

BEDINGUNGEN FÜR DEN EINTRITT DES TODES

Die genauen Bedingungen, die nötig sind, damit die Methode, die das Sterben auslöst, getriggert wird, sind in der Funktion „IsDeathConditionMet()“ definiert. Dort wird festgelegt, dass das Sterben eintritt, wenn die Sättigung, die Wassersättigung oder die Gesundheit dem Wert 0 entsprechen.

AUSWIRKUNGEN DES TODES

Wenn ein Tier stirbt, wird dies über einen entsprechenden Sound (eine Art „Stampf-Sound“) erkenntlich gemacht, solange das Abspielen von Sounds nicht ausgeschaltet wurde. Zunächst wird das „AnimalMenu“ des entsprechenden Tieres, falls dieses geöffnet war, wieder geschlossen. Des Weiteren werden bestimmte Counter-Zählungen durchgeführt. All das muss passieren, bevor das Objekt letztendlich zerstört wird, da sonst eine „NullReferenceException“ ausgelöst wird. Dieser Bug tritt auf, wenn das Programm versucht, Befehle an einem Objekt auszuführen, das nicht mehr existiert. Um eine „NullReferenceExcepiton“ zu vermeiden, ist meist ein „null-Check“ die Lösung oder das Einbauen eines „try/catch-blocks“ [41]. Ist ein Tier gestorben, so verschwindet es aus dem Programm. Tote Tiere bleiben also nicht liegen.

3.5.8 GENERIERUNG VON OBJEKTEN

Es gibt zwei verschiedene Arten, über die Objekte generiert werden können. Entweder sie entstehen nach einer bestimmten Zeit von selbst oder man kann sie manuell über entsprechende Input-Felder setzen.

GENERIERUNG ÜBER DEN „AMOUNTSETTER“

Das manuelle Erstellen von Objekten ist über den "AmountSetter“, welcher über das „AmountSetter“-Skript gesteuert wird, möglich. Eine Abbildung des „AmountSetters“ zeigt das folgende Bild (s. Abbildung 31).

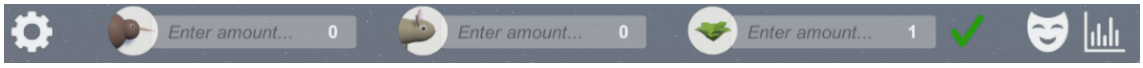

Abbildung 31: AmountSetter

Klickt man in das Feld, in welchem „Enter amount…“ steht, so kann man die gewünschte Zahl eingeben, zu der Objekte spawnen sollen. Zunächst lässt sich auf das jeweilige Symbol klicken (Kiwi, Possum oder Nahrung) um einen einzelnen Wert zu spawnen. Schreibt man in alle drei Felder bestimmte Zahlen, so kann man die gleichzeitige Generierung über Betätigen des grünen Hakens auslösen. Sobald man die Objekte generiert hat, wird der Wert der weiß abgebildeten Zahl mit dem gesetzten Wert überschrieben. Dies geschieht über bestimmte Zähler („Counter“), die die Anzahl des jeweiligen Objekts bei jeder Instanziierung hochzählen. Das Zahnrad ermöglicht mithilfe eines Klicks darauf das Einklappen des „AmountSetters“. Auf diese Weise kann man den „AmountSetter“ ausblenden, falls er während der Simulation als störend empfunden wird.

GENERIERUNG ÜBER ZEIT

Nahrung und Bäume entstehen automatisch nach einer gewissen Zeit. Diese Funktion ist in das „Spawner-Skript“ eingebaut und wird durch den Aufruf der Methoden „InitiateFoodSpawn()“ und „InitiateTreeSpawn()“ ermöglicht. In diesen Funktionen werden die Objekte so lange generiert, bis die maximale Anzahl erreicht ist. Der Maximalwert für Nahrung liegt bei 500 Objekten und der von Bäumen bei 200, um ein Überfüllen der Karte zu vermeiden und einer Verlangsamung des Programms entgegenzuwirken. Die Methode „InitiateFoodSpawn()“ wird zum genaueren Verständnis hier abgebildet:

```
private IEnumerator InitiateFoodSpawn()
{
   while (foodCounter < MAXFOOD)
   {
      yield return new WaitForSeconds(CalculateSpawnFoodDelay());
      RandomFood();
   }
}
```

Wie man im obenstehenden Code sehen kann, entstehen Objekte immer wieder nach einer gewissen Zeit (hier „CalculateSpawnFoodDelay()“) und werden bei der Nahrung über die Methode „RandomFood()“ generiert. Um genauer zu verstehen, was sich dahinter verbirgt, wird nun das Ermitteln der Zeit, in der Objekte entstehen genauer erklärt und die Funktionen, die für komplexere Zufallswerte verantwortlich sind.

ERMITTLUNG VON SPAWNZEITEN

Um der Simulation einen realistischeren Effekt zu geben und die Auswirkungen bei einer Dynamik im Ökosystem zu zeigen, wurden die „Spawnzeiten“ für Bäume und Nahrung in Abhängigkeit von verschiedenen Bedingungen erstellt. Bei der Spawnzeit von Nahrung wurde sich an bekannten Gegebenheiten der Natur orientiert, die meist bei einer hohen Anzahl von Würmern, die die Struktur des Bodens verbessern und organische Abfälle zerlegen [42], einfacher neue Pflanzen produzieren kann. Aufgrund dessen gibt es in diesem Programm einen „Wurmbonus“, der besagt, dass bei einer hohen Anzahl von Würmern die Zeit für das Spawnen von Nahrung kleiner wird. Nahrung entsteht demzufolge in kleineren Abständen, was im Schlussendlichen in einer höheren Anzahl von Nahrung resultiert. Im Konkreten sieht das Ganze so aus:

```
private float CalculateSpawnFoodDelay()
{
	float wormBonus = wormCounter * 0.1f;
	return Mathf.Clamp(maxSpawnFoodDelay – wormBonus,minSpawnFoodDelay,
	maxSpawnFoodDelay);
}
```

Der Wurmbonus entsteht, indem die Anzahl der gezählten Würmer mit 0,1 multipliziert wird. Bei 100 Würmern beispielsweise, würde der Wurmbonus 10 betragen. Die Maximal- und Minimalzeit wurde im Vorhinein festgelegt und beträgt bei der Nahrung minimal 4 und maximal 8. Durch „Mathf.Clamp“ wird die resultierende Zahl auf das Minimum oder Maximum gesetzt. Am Beispiel würde das bedeuten, dass die Spawnzeit bei 100 Würmern 4 ergibt, da

die maximale Zeit (8) – den Wurmbonus (10) = -2

den Wert -2 ergeben würde und dieser zunächst auf den kleinstmöglichen Wert (4) gesetzt wird. Dadurch wird Nahrung in kürzeren Abständen entstehen.

Bei den Bäumen verhält es sich ebenso, nur dass die Minimal- und Maximalwerte variieren und bei den Bäumen die Anzahl der Kiwis und Possums ausschlaggebend für eine Veränderung der Spawnzeit sind. Bei mehr Kiwis sollen auch mehr Bäume entstehen und bei mehr Possums soll seltener ein Baum spawnen. Dieses Ereignis entspricht nicht der Realität, aber soll ungefähr den Schaden widerspiegeln, den Possums auf die Bäume ausüben [4].

3.5.9 ZUFALLSWERTE

Um Zufälliges zu generieren, wurden drei grundlegende Methoden in das „Spawner.cs“-Skript integriert, die sich in ihrer Funktion und Umsetzung unterscheiden. Genaueres wird in den nächsten Abschnitten erklärt.

„RANDOMOBJECTS“

In der Methode „`RandomObjects(GameObject obj1, GameObject obj2, GameObject obj3)`“ werden drei Game-Objekte mitgegeben, die beim Aufruf der Methode benannt werden können, sodass klar wird, welche Objekte angesprochen werden sollen. Diese Objekte werden mit bestimmten Wahrscheinlichkeiten generiert. Objekt 1 und Objekt 2 werden beide mit 40% Wahrscheinlichkeit erstellt, während Objekt 3 nur zu 20% entsteht. Diese allgemeine Methode kommt z. B. bei der Generierung der Bäume zum Einsatz, bei der zwischen drei verschiedenen Baumarten unterschieden wird.

„RANDOMVALUES“

Die nächste zu nennende Methode ist die Methode „`RandomValues(int value1, int value2, int value3, int value4)`“, die schon bei der Generierung der Anzahl des Nachwuchses kurz erklärt wurde. Hierbei wird, anstatt eines zufälligen Objekts, ein zufälliger Wert als Ergebnis geliefert. Die Wahrscheinlichkeitsverteilung ist hier wie folgt: value1 entsteht zu 40%, value2 zu 30%, value3 zu 20% und value4 zu 10%. Gibt man beim Ausführen der Methode bestimmte Werte mit, wie z. B. beim Festlegen der Kinderanzahl die Werte 1, 1, 2, 0, dann entsteht zu 40% 1 Kind, zu 30% ebenfalls 1 Kind, zu 20% 2 Kinder und zu 10% kein Kind. Bei der Erstellung dieser Logik wurde zuerst mithilfe von „Random.Range()“ eine Zufallszahl („randomInt“) zwischen 0 und 10 erstellt, wobei die 10 ausgeschlossen ist. Zum besseren Verständnis wird der folgende Codeausschnitt aus dem Skript beigefügt:

```
int randomInt = Random.Range(0, 10);

if (randomInt <= 3) // 0,1,2,3 -> 40%
{
	return value1;
}
else if (randomInt <= 6) // 4,5,6 -> 30%
{
	return value2;
}
```

Angenommen die generierte Zufallszahl ist 3, dann würde der Wert 3 mit „randomInt“ verglichen werden und das Ergebnis wäre, dass zu 40% „value1“ zurückgegeben wird.

„RANDOMFOOD“

Beim Generieren von Nahrung musste eine Extra-Methode „RandomFood()“ geschrieben werden, da es hier, aufgrund der Dynamik der Spawnzeiten, die sich in Abhängigkeit von der Anzahl der Würmer verändern, von großer Bedeutung war, die generierten Objekte zu zählen. Aus diesem Grund musste das Objekt selbst in der Methode genannt werden, um eine Zählung über den „Counter“ möglich zu machen.

3.6 STARTMENÜ

Bevor die Simulation startet, befindet man sich im Startmenü, in welchem man Programmeinstellungen vor dem Start der Simulation vornehmen kann. Das Startmenü stellt zugleich die erste Szene des Programms dar. Einen Überblick über das Startmenü ist mithilfe der untenstehenden Abbildung (s. Abbildung 32) zu erhalten.

Abbildung 32: Startmenü

Auf Die Erstellung des Startmenüs und dessen Funktionen wird im Weiteren genauer eingegangen.

3.6.1 FUNKTIONEN

Wie Abbildung 32 zu entnehmen, wurde das Startmenü mit diversen Buttons versehen, die beim Betätigen ihre jeweilige Funktion auslösen.

START-BUTTON UND SZENENWECHSEL

Beim Klick auf „Start“ beginnt die Simulation. Dies geschieht über den Szenenwechsel, der über das Skript „MainMenu.cs“ gesteuert wird. Mit dem Codebefehl

```
SceneManager.LoadScene("SampleScene");
```

wird der Szenen-Manager angesprochen, der zwischen den jeweiligen Szenen unterscheidet. Szenen können in den „Build-Settings“ in eine bestimmte Reihenfolge gebracht werden. Sie erhalten dort jeweils eine Nummerierung, die mit der Zahl 0 beginnt. Es gibt dann die Möglichkeit, die Szene per Nummer anzusprechen, oder bei „LoadScene()“ den Namen der Szene in die Klammern einzutragen. In diesem Programm hat das Startmenü die Nummer 0, da es zuerst ausgeführt werden soll und die „SampleScene“, also die Szene, in der sich die Simulation abspielt, die darauffolgende Nummer 1.

OPTIONS-BUTTON

Bei Betätigen des „Options“-Buttons, erscheint das Fenster des „OptionsMenu“, in welchem man, wie beim „AmountSetter“, die Anzahl der Kiwis, Possums und Nahrung einstellen kann. Dies kann an dieser Stelle allerdings vor dem Simulationsstart konfiguriert werden, sodass man schon mit der gewünschten Anzahl der Tiere in die Simulation starten kann. Damit die im Startmenü gesetzten Werte in der „SampleScene“ erkannt werden, müssen sie gespeichert und der Anzahl, der zu generierenden Objekte, im „AmountSetter.cs“ gleichgesetzt werden. Dies geschieht über das „GameVariables-Skript“, das Variablen zwischen verschiedenen Szenen nutzbar macht.

CONTROLS-BUTTON

Der „Controls“-Button zeigt bei Betätigen eine Grafik, in der die Steuerung innerhalb der Simulation erkennbar wird. Diese Funktion basiert daher auf einer Art Bedienungshilfe. Die abgebildete Grafik (s. Abbildung 33) wurde selbst mit dem Programm „Affinity Designer“ erstellt und soll anschaulich vermitteln, wie die Steuerung innerhalb des Programms funktioniert. Als Anregung wurde in Google-Bilder [43] für die Erstellung der Grafik gesucht.

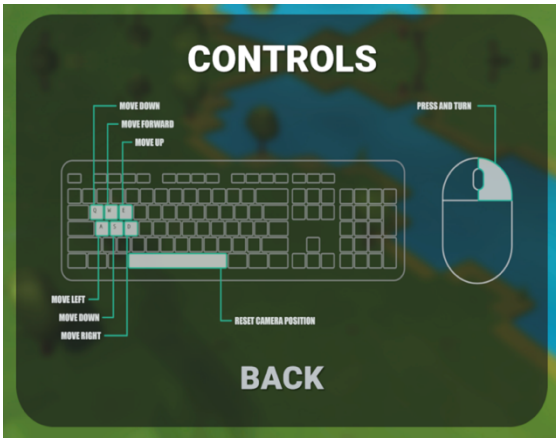

Abbildung 33: Controls

QUIT-BUTTON

Der Button „Quit“ ist selbsterklärend. Er beendet bei Betätigen das Programm. Dies funktioniert über den Befehl:

```
Application.Quit();
```

3.6.2 DESIGN DES STARTMENÜS

Damit das Startmenü passend zur Thematik der Simulation ist, wurden die im Folgenden genannten designtechnischen Komponenten eingebaut.

LOGO

Über den Buttons befindet sich das Logo der Simulation, welche sich „Ecosys“ nennt. Dieses ist in Abbildung 32 zu sehen. Das „O“ stellt eine Weltkugel dar, die die Karte des Programms symbolisieren soll. Um die Weltkugel herum läuft ein Kiwi, um das Thema der Simulation aufzugreifen. Das Logo wurde selbst entwickelt und im Programm „Affinity Designer“ in Zusammenarbeit mit dem „Affinity Publisher“ erstellt. Die verwendete Schriftart nennt sich „Skia“.

HINTERGRUND

Beim Hintergrund wurde zuerst überlegt ein statisches Hintergrundbild in Affinity Designer zu bauen, bis auf jenes YouTube-Video: [44] gestoßen wurde. Zunächst fiel die Entscheidung auf einen beweglichen Background, der langsam beim Start des Programmes in eine Richtung scrollt. Auf diese Art und Weise wirkt das Startmenü lebendig und modern. Als zu scrollender Hintergrund wurde die Karte der Simulation als Screenshot in das Programm Adobe Photoshop eingefügt und dort mit dem Tool „Gaußscher Weichzeichner“ verschwommener gemacht. Dies ist hilfreich, um die Funktionen des Menüs in den Vordergrund rücken zu lassen und so den Fokus mehr auf das eigentliche Menü zu legen als auf den Hintergrund. Um das Scrollen so fließend wie möglich zu machen, war es wichtig in den Einstellungen des Images unter dem Reiter „Advanced“ den „Wrap Mode“ auf „Mirror“ umzustellen. Dadurch wiederholt sich das Bild ständig in einer gespiegelten Form. Der Background wird mit dem Skript „BGScroller.cs“ gesteuert. Im Inspector kann man über die Skript-Komponente das gewünschte Background-Image einfügen und die Geschwindigkeit in X- und Y-Richtung entsprechend anpassen. Hier ist ein Wert von 0.02 bei X und Y ausreichend, um eine adäquate Scroll-Geschwindigkeit erzeugen zu können.

3.7 UI

In der Simulation selbst können bestimmte Einstellungen während der Programmlaufzeit mithilfe von UI-Elementen gemacht werden. Diese liegen in Form eines Canvas-Objects auf der Kamera, sodass es zu einer passenden Anordnung kommt. In der nachfolgenden Abbildung (s. Abbildung 34) werden alle Elemente, mit denen Einstellungen oder Sachverhalte verändert werden können, gezeigt und im Weiteren genauer erklärt.

Abbildung 34: UI

Links oben befindet sich der „AmountSetter“, mit dem man die Anzahl der Tiere und Nahrung selbst setzen kann. Genaueres zum „AmountSetter“ wurde im vorherigen Kapitel „Generierung über den AmountSetter“ vertiefter dargelegt. Neben der Möglichkeit Objekte zu erstellen, sind noch zwei weitere Buttons zu sehen. Das Masken-Symbol beschreibt den Kostümmodus und das Statistik-Symbol gewährt Einblick in die Diagramme. Genaueres hierzu, ist im Kapitel „Spezialeffekte“ zu finden.

3.7.1 WEITERE EINSTELLUNGEN IN DER SIMULATION

Das kleinere Menü rechts daneben, ermöglicht weitere Einstellungen innerhalb der Simulation vorzunehmen, auf die nun genauer eingegangen wird. Das Symbol mit dem Notenschlüssel bedeutet, dass sich die Musik während der Simulation ausschalten lässt. Dies ist nützlich, falls man die Sounds besser hören möchte oder sich von der Musik gestört fühlt. Beim Klick auf das Lautsprecher-Symbol verhält es sich genauso, nur dass sich hierbei die Sounds deaktivieren lassen. Die beiden Lupen-Symbole ermöglichen ein Hinein- und Hinauszoomen aus der Simulation per Klick. Hierfür sind für beide Einstellungen die gezeigten Ausschnitte (s. Abbildung 35) vorkonfiguriert, indem der Kamera feste Positionen für die jeweilige Einstellung zugeteilt wurden.

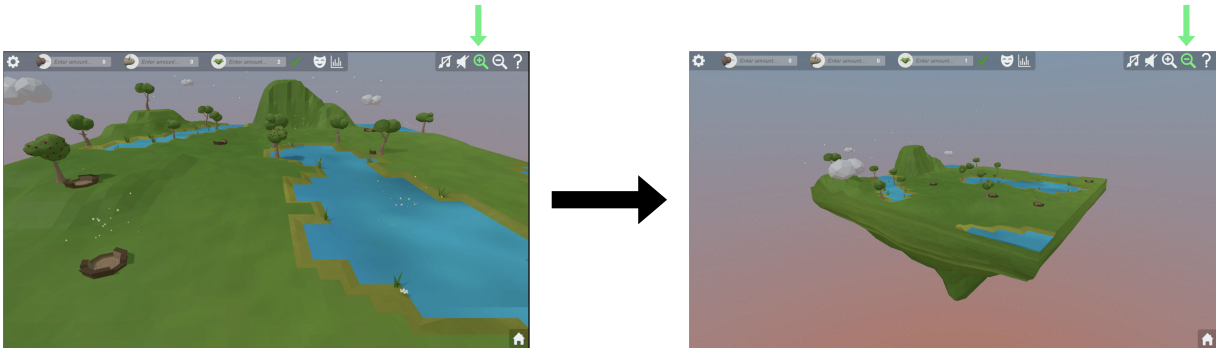

Abbildung 35: Zoom im Vergleich

Das letzte gezeigte Symbol ist ein Fragezeichen. Bei Betätigen dieses Buttons öffnet sich ein Menü direkt darunter, wie in der nebenstehenden Abbildung (s. Abbildung 36) zu sehen ist, in dem kurz die Steuerung für das Programm erklärt ist. Auch dieses wurde mit dem „Affinity Designer“ selbst erstellt. Dieses Menü soll bei Unsicherheit über die Steuerung Abhilfe schaffen. Welcher Modus gerade ausgewählt ist, ist an einer grünen Markierung erkennbar.

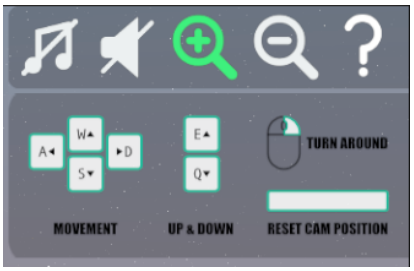

Abbildung 36: Kleines Steuerungsmenü

3.7.2 DAS „ANIMALMENU“

Da jedes einzelne Tier unterschiedliche Merkmale hat und sich in verschiedenen Stadien befinden kann, wurde zur genaueren Betrachtung und Informationsgewinnung über ein Tier ein „AnimalMenu“ eingebaut. Dieses öffnet sich beim Mausklick auf das jeweilige Tier. Funktionsfähig wird das „AnimalMenu“, indem dem „AnimalMenu.cs“-Skript die Daten der Tiere übergeben werden, sodass sie z. B. in den Textfeldern oder als Symbole erscheinen. Anhand der Abbildung (s. Abbildung 37) werden einige Komponenten des „AnimalMenus“ nun genauer beschrieben.

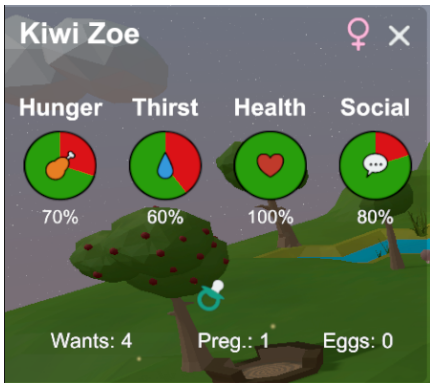

Abbildung 37: Das „AnimalMenu“

BEDEUTUNG DER KOMPONENTEN

Links oben befindet sich der Name des Tieres und rechts daneben das jeweilige Geschlecht. Beim Klick auf den „x“-Button, kann man das Menü wieder schließen. Zunächst sieht man den Bedürfnisstand der Tiere in Prozent. Der untere Abschnitt ist für das Thema Fortpflanzung zuständig. Ein Schnuller-Symbol zeigt, dass das Weibchen gerade trächtig ist. Ein gerader Strich stellt dar, dass sich das Tier im Status „onMatingCoolDown“ befindet. Beim Weibchen bedeutet „Wants“ die Mindestanforderung an Attraktivität, die sie an den Mann hat. Beim Mann steht an dieser Stelle „Att.“ für „Attractiveness“, die wiederum seinen Attraktivitätswert anzeigt. „Preg.:“ stellt die Anzahl der

Schwangerschaften, die das Weibchen bereits hatte, dar und „Eggs“ oder beim Possum „Babies“ die Anzahl an gelegten Eiern oder geborenen Babys. Diese Werte können interessant für die Beobachtung der Fortpflanzung und der Möglichkeit einer Spezies seine Population zu vergrößern sein. Es gibt noch zwei weitere Symbole, die nur unter bestimmten Bedingungen erscheinen. Diese werden in folgender Abbildung (s. Abbildung 38) dargestellt.

Abbildung 38: Weitere Status-Symbole

Das lila Symbol, zeigt auf, dass eine Infektion oder Vergiftung vorliegt. Da nur Possums erkranken können, gibt es nur beim Possum die Möglichkeit, dieses Symbol zu sehen. Das nebenstehende Kind-Symbol, gibt Aufschluss darüber, dass es sich beim angeklickten Tier noch um ein Baby handelt. Sobald das Baby erwachsen wird, verschwindet das Kind-Symbol wieder. Wichtig einzubauen war hier die Methode „PossumDied()“ und „KiwiDied()“, die im jeweiligen Tier aufgerufen wird, sobald dieses stirbt und dafür sorgt, dass dabei auch das „AnimalMenu“ geschlossen wird. In diesen Methoden findet ein ID-Abgleich des ausgewählten Tieres statt, sodass klar ist, welches „AnimalMenu“ geschlossen werden muss.

3.8 STEUERUNG

Um einen lebendigeren Faktor einzubauen und dem Nutzer der Simulation die Möglichkeit zu geben, vollkommen in das Ökosystem einzutauchen sowie spezifische Tiere genauer zu beobachten, wurde eine Steuerung eingebaut. Diese macht es möglich, mithilfe der Tastatur und der Maus durch die Karte zu navigieren. Zur genaueren Identifikation der relevanten Tasten dient Abbildung 39.

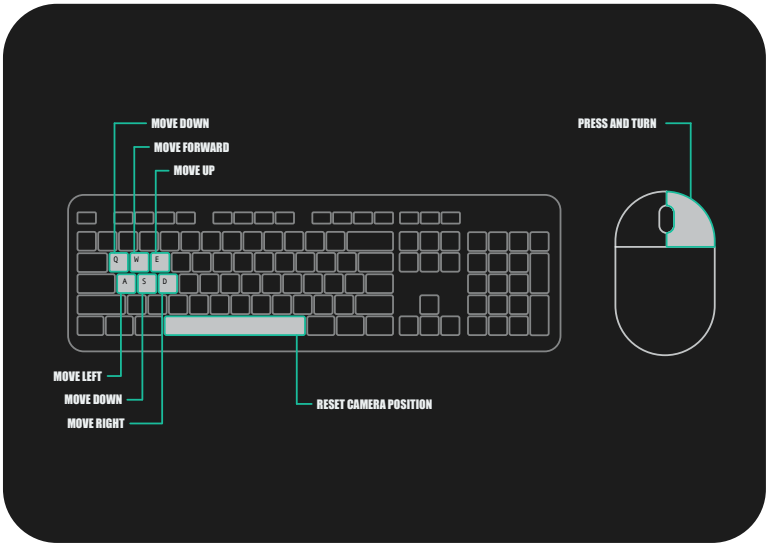

Abbildung 39: Grafik zur Steuerung

ERKLÄRUNG DER TASTENFUNKTIONEN

Das Hin-und Herbewegen funktioniert per Tastatursteuerung mit den Tasten „W“, „A“, „S“ und „D“. Möchte man nach oben oder nach unten navigieren, so kann man die Taste „E“ für eine Aufwärtsbewegung und die Taste „Q“ für die Bewegung nach unten verwenden. Bei Klicken und Halten auf die rechte Maustaste, während man die Maus hin- und herbewegt, kann man die Ansicht drehen.

Mit der Leertaste kann man die Position der Kamera zurücksetzen, falls man wieder vom Startpunkt aus navigieren möchte oder zu dieser Ansicht zurückmöchte. Als Startpunkt definiert ist die „Zoom“-Ansicht, die man auch erhält, wenn man im Menü auf die Lupe mit dem „+“ klickt.

KONFIGURATION DER STEUERUNG

Die Steuerung wurde konfiguriert, indem der Standpunkt der Kamera bei Betätigen von entsprechenden Tasten oder der Maus verändert wird. Dafür befinden sich die Methoden „MoveCamera()“ für die Tastatursteuerung und „TurnCamera()“ für das Drehen mit der Maus im „GameController.cs“-Skript, welches generelle Settings und Funktionen für die Simulation enthält.

Um ein Hindurchwandern durch den Boden zu vermeiden, wurde in der Methode „MoveCamera()“ eine Limitierung für den erreichbaren Y-Wert hinzugefügt. Befindet sich dieser unter dem Wert 10, so wird er zurückgesetzt auf den Wert 10, sodass sich die Kamera nicht weiter als festgelegt hinunterbewegen kann. Die Zuweisung von Funktionen auf Tasten gelingt bei Unity z. B. über den Befehl „`Input.GetKey(Keycode.A))`“, wobei man für A jede beliebige Taste einfügen könnte. Auf diese Weise wurden die Tasten und die Maus belegt.

3.9 NIEDERLAGE & SIEG

Um eine überfüllte Karte und das Verlangsamen des Programms zu vermeiden, wurde ein Simulationsende in Form von einer Niederlage oder einem Sieg festgelegt. Da es sich hierbei um grundlegende Programmfunktionen handelt, ist die Implementierung erneut im „GameController“-Skript zu finden. Hierfür wurde für Possums und Kiwis jeweils eine Maximalzahl von 200 angegeben. Um ein Simulationsende herbeizuführen, muss die Maximalzahl einer Tierart erreicht werden oder die andere Tierart ausgestorben (0) sein. Außerdem müssen beide Tierarten zu Beginn gespawnt worden sein, um überhaupt einen „Sieg“ oder eine „Niederlage“ triggern zu können. Dies ermöglicht, dass die Simulation bei einem Wert von 0 nicht direkt endet, was z. B. passieren kann, wenn man nur mit einer Tierart startet. Zudem wurde ein Zeitcounter eingebaut,

Abbildung 40: Information über eine Niederlage

der erst nach 10 Sekunden ein Gewinnen oder Verlieren ermöglicht. Als Sieg wird in dieser Simulation, passend zum Thema, eine Maximalerreichung der Kiwi-Population oder das Aussterben der Possums gewertet. Eine Niederlage entsteht, wenn es keine Kiwis mehr gibt oder die Possums die Anzahl 200 erreicht haben. Tritt eine der beiden Szenarien ein, so wird ein Sound abgespielt, der Sieg oder Niederlage symbolisiert und das Programm wird über den Befehl „`Time.timeScale = 0f;`“ angehalten. Zunächst wird man via Text über das Erreichen eines entsprechenden Zustands informiert (s. Abbildung 40) und hat nun die Möglichkeit sich die Statistik der Simulation, auf welche im folgenden Kapitel genauer eingegangen wird, anzusehen oder zum Startmenü zurückzukehren.

3.10 STATISTIK

Abbildung 41: Statistik

Da diese Simulation darauf ausgelegt ist, das Verhalten von Possums und Kiwis untereinander und die Dynamik zwischen den beiden Tierarten zu erkennen und zu visualisieren, sind bestimmte Werte, wie die Anzahl der Kiwis, der Possums oder der Tode interessant zu beobachten (s. Abbildung 41).

EINBETTUNG DER STATISTIK

Um dies möglich zu machen, wurde vorerst das Skript „Statistics.cs“ erschaffen, in welchem mehrere Listen für verschiedene Gegebenheiten erstellt wurden. In die vordefinierten Listen werden alle fünf Sekunden die Werte bestimmter Zähler („Counter“) eingespeichert. Um möglichst viele interessante Vergleiche ziehen zu können, wurden Werte für die Anzahl der Possums und Kiwis, für gestorbene Possums und Kiwis und die Ursache des Todes sowie für die Anzahl der Possum-Babys, der Kiwi-Eier

und der von den Possums gefressenen Eiern in die entsprechenden Listen gespeichert. Außerdem wird die Anzahl der Nahrung und Bäume festgehalten.
Um die Werte sinnvoll zu veranschaulichen und die Geschehnisse auswerten zu können, wurde ein Diagram-Pack aus dem Unity Assets Store gekauft, welches sich „Graph and Chart – Lite Edition“ [45] nennt. Zunächst wurde ein Game-Objekt „Graph“ definiert, welches das Elternobjekt für alle Diagramm-Objekte darstellt. Auf dem „DataInitializer“ liegt das erstellte „Graph“-Skript, in welchem festgelegt wird, welche Werte welcher Sparte zugeordnet werden sollen. Alle Graph-Objekte, die in der Simulation sichtbar sind, müssen dem „DataInitializer“ zugeordnet werden, um die Funktionsfähigkeit des jeweiligen Graphens zu gewährleisten.

FUNKTIONEN DER STATISTIK

Bei Klick auf das „Statistics“-Symbol, kann man die Diagramme auch während der Programmlaufzeit einsehen und mit einem weiteren Klick auf den „Refresh“-Button, welcher auf der Abbildung 41 oben links zu sehen ist, die Werte aktualisieren. Klickt man oben rechts in einem Diagramm auf das Zoom-Symbol, so kann man das ausgewählte Diagramm in großer Ansicht sehen.

EINGEBAUTE GRAPHEN

In der Simulation kann man vier verschiedene Graphen einsehen. Zuerst wird die Population der jeweiligen Tierart angezeigt. Daneben kann man die Werte für gelegte Eier und geborene Possum-Babys betrachten. Außerdem werden hier auch gefressene Eier gelistet. Die beiden unteren Graphen untersuchen die Tode und Todesursachen der beiden Tiere. Ein besonderes Untersuchungsmerkmal beim Kiwi ist hier, wie viele Kiwis gefressen wurden und beim Possum, die Tode, die durch den giftigen Pilz entstanden sind. Die x-Achse stellt die Zeit dar und die y-Achse die Anzahl. An jedem Punkt, der ausgegeben wurde, kann man durch Labels sehen, welche Anzahl zu welcher Zeit genau vorhanden war. Leider führt dies nach einer längeren Laufzeit zu einem Mangel an Übersicht. Um die Graphen zu konfigurieren war die Einarbeitung in das Tool über das vom Entwickler bereitgestellte „Documentation Portal“ [46] notwendig.

3.11 SPEZIALEFFEKTE

Um die Simulation anschaulich zu gestalten oder manche Gegebenheiten besser erkennbar zu machen, wurden diverse „Special Effects“ eingefügt, die im Folgenden genauer dargelegt werden.

3.11.1 STATUSANZEIGE ÜBER SYMBOLE

Um bestimmte Ereignisse darzustellen, die eventuell nicht sofort auf den ersten Blick erkennbar sind, wurden für die Szenarien „Paarung“ und „Fliehen“ Symbole über den Tieren erstellt, die die derzeitige Tätigkeit besser veranschaulichen und untermalen sollen. Hierfür wurden vorerst die nötigen Symbole (Herz und Ausrufezeichen) in Blender erstellt und diese dann als Kind-Objekt dem

Tier-Prefab untergeordnet. Für die Visualisierung des Fliehen-Status wurde sich für ein rotes Ausrufezeichen entschieden und bei dem Vorkommnis einer Paarung sollte ein Herz über beiden Tieren erscheinen. Wenn sich nur das Männchen verliebt, so zeigt sich das Herz-Symbol nur über diesem. Ein Beispiel für das Flieh-Symbol (s. Abbildung 42) und eines für das Herz-Symbol (s. Abbildung 43) zeigen die folgenden Abbildungen.

Abbildung 42: Flieh-Symbol

Abbildung 43: Herz-Symbol

3.11.2 DER KOSTÜMMODUS

Um ein witziges Feature einzubauen und zur leichteren Unterscheidung der Geschlechter, wurde ein Kostümmodus (s. Abbildung 44) implementiert, der bei Klick auf das Masken-Symbol ausgelöst werden kann. Bei erneutem Betätigen des Masken-Buttons lässt sich der Kostümmodus wieder deaktivieren. Für den Kostümmodus wurden ebenfalls in Blender zwei Objekte kreiert. Zum einen war dies ein Zylinder, den männliche Objekte tragen und für die Weibchen eine Schleife, die ebenfalls auf dem Kopf getragen wird. Das An- und Ausschalten der Kostüme funktioniert über das Aktiv- und Inaktivsetzen des Schleifen- und Zylinder-Objekts. Die Objekte wurden, genau wie bei den Status-Symbolen über den Tieren, dem Prefab als Kind-Objekt mitgegeben. Damit sich die Kopfbedeckungen allerdings mitbewegen, wenn ein Tier z. B. frisst, war es notwendig, das Objekt an dem Knochen anzuordnen, der für die Bewegung zuständig ist. Ansonsten würden Zylinder oder Schleifen, sobald eine Bewegung ausgeführt wird, in der Luft schweben.

Abbildung 44: Tiere bei aktiviertem Kostümmodus

3.11.3 DIE UMRANDUNG („OUTLINE“)

Abbildung 45: Outline

Da bei der Entwicklung der Simulation aufgefallen ist, dass nicht immer genau erkennbar ist, welches Tier gerade angeklickt wurde, sollte das Hinzufügen einer gelben Umrandung, bei Klick auf das Tier, Abhilfe

schaffen (s. Abbildung 45). Hierfür wurde ein Tool aus dem Unity Asset Store geladen, das sich „QuickOutline“ [47] nennt. Dem Tier-Prefab wurde zunächst das im Package enthaltene „Outline.cs“-Skript als Komponente hinzugefügt. In dieser kann man sowohl die Farbe der Umrandung als auch die Dicke der Linie anpassen. Damit die Umrandung nur erscheint, wenn man auf das Tier klickt, wird das Outline-Skript standardmäßig deaktiviert und bei Klick auf das Tier wieder aktiviert. Über einen „RayCastHit“ wird in der Methode „MainMouseListener()“ gecheckt, auf welches Objekt geklickt wurde und zunächst die Aktivierung des Outline-Skripts ausgeführt. Klickt man auf ein anderes Tier, so wird die Outline bei diesem Tier aktiviert und beim Vorherigen wieder deaktiviert.

3.11.4 DIE STERNENHIMMEL-SYKBOX

Da Kiwis und Possums beide nachtaktiv sind, war es am sinnvollsten die Simulation in einer Umgebung bei Nacht spielen zu lassen. Um das nächtliche Erscheinungsbild am besten erkennbar zu machen, wurden zum einen Wolken, die über Blender erstellt wurden, im Himmel platziert und zum anderen eine „Sternenhimmel-Skybox“ mit einem YouTube-Tutorial [48] erstellt, über die in diesem Kapitel genauer berichtet wird.

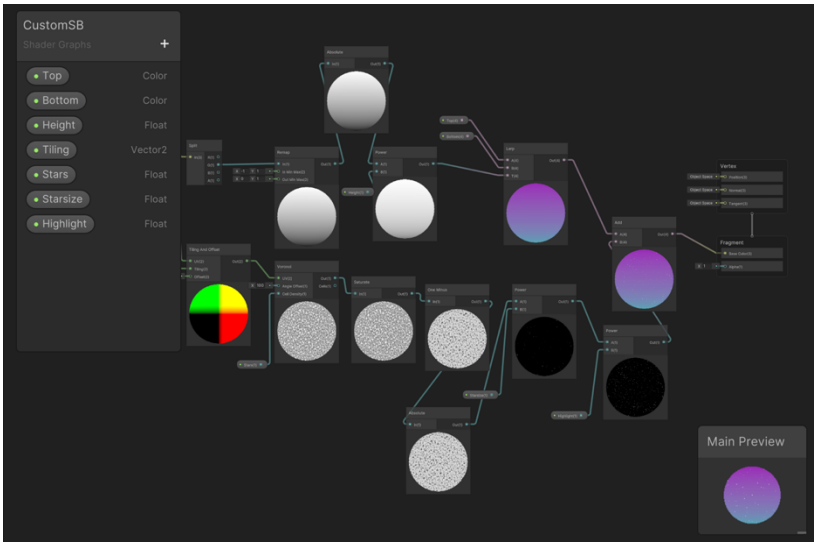

Abbildung 46: Shader Graphs

ERSTELLUNG DER SKYBOX

Um die Skybox zu erstellen, war die Arbeit mit „Shader Graphs“ erforderlich. Ein Ausschnitt hiervon ist in Abbildung 46 zu sehen. Die Variablen, welche auf der linken Seite zu sehen sind, lassen sich hier konfigurieren und später im Inspector anpassen. Die Felder „Top“ und „Bottom“ beschreiben die beiden Farben, die zur Erstellung des Verlaufs genutzt werden. Des Weiteren kann man z. B. auch die Menge, Größe und Helligkeit der Sterne im Inspector anpassen.

Zunächst wurde so vorgegangen, dass im oberen Bereich ein Verlauf kreiert wurde. Im unteren Teil wurde ein Muster erstellt, welches an Sterne erinnert. Dies geschah zuerst, in dem die Komponente „Voronoi“ genutzt und die Farben anschließend mit „One Minus“ umgekehrt wurden. Daraus ergab sich ein schwarzer Hintergrund mit weißen Punkten. Zum Schluss musste die Komponente des Farbverlaufs mit der Komponente der „Sterne“ verbunden werden, sodass das Zielbild, welches einem Sternenhimmel gleichkommen sollte, erreicht wird. In der „Main Preview“ zeigt sich das Resultat. Zunächst wurde ein neues Material namens „MySkybox“ erstellt, welches als Shader-

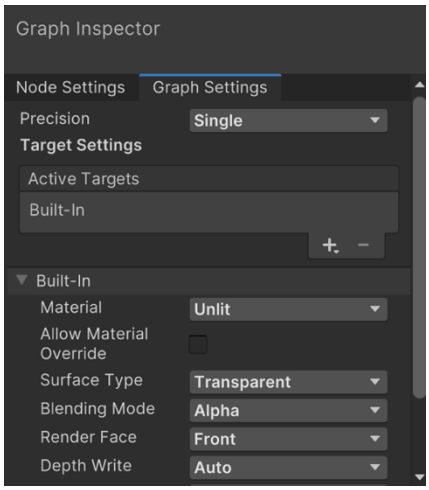

Abbildung 47: Graph Inspector

Einstellung den in Abbildung 46 zu sehenden Shader Graph „CustomSB“ nutzt. Um die erstellte Skybox nun auch im Programm sehen zu können, muss das „Lighting“-Fenster von Unity geöffnet werden und unter dem Reiter „Environment“ dem „SkyboxMaterial“-Feld das neu erstellte Material „MySkybox“ zugewiesen werden. Damit der Sternenhimmel auch tatsächlich zu sehen ist, ist es essenziell im „Shader Graph“ den „Graph Inspector“ (s. Abbildung 47) zu öffnen und den „Surface Type“ auf „Transparent“ zu stellen.

3.11.5 DAS PARTIKELSYSTEM

Um den Nachtmodus zusätzlich zu unterstreichen wurde ein Partikelsystem, welches in Unity als Game-Objekt hinzugefügt werden kann, genutzt. Mithilfe des Partikelsystems sollten Glühwürmchen dargestellt werden. Hierfür wurden mehrere kleine Häufchen mit Glühwürmchen-Schwärmen über die Karte verteilt. Bei einem Partikelsystem kann man mehrere Einstellungen im Inspector-Fenster vornehmen, die z. B. die Geschwindigkeit, die Richtung, die Farbe oder die Art der Partikel bestimmen. Um die Glühwürmchen möglichst real erscheinen zu lassen, wurde das Partikelsystem mithilfe eines YouTube-Tutorials [49] angepasst. Ein Beispiel eines Glühwürmchen-Häufchens ist der Abbildung 48 zu entnehmen.

Abbildung 48: Glühwürmchen-Häufchen

3.12 DESIGN

Der Design-Teil der Simulation umfasst das Anfertigen jeglicher in der Simulation enthaltenen 3D-Modelle, sowie das Erstellen der verschiedenen Icons für Symbole und die UI. Hierfür wurden die Programme Affinity Designer, Affinity Publisher und Adobe Photoshop genutzt. Die 3D-Modelle wurden im Programm Blender umgesetzt. Im Folgenden wird näher auf das Ausarbeiten des Designs eingegangen.

3.12.1 ICON-ERSTELLUNG

Da für einige Komponenten bestimmte Icons zur besseren Erkennung oder Visualisierung nötig waren, wurden diese mit den oben genannten Programmen selbst erstellt. Der Affinity Designer, welcher ähnliche Funktionen, wie Adobe Illustrator aufweist, wurde vorwiegend für das Anfertigen der Icons genutzt. Das Herausexportieren erfolgte dann über den Affinity Publisher, da es dort einfacher war, das Icon ohne Hintergrund zu exportieren.

DIE ICONS

Bei den zu erstellenden Icons kann man unterscheiden zwischen den Icons, die notwendig für die Statusanzeige der Tiere sind und jenen, die in Menüs, bzw. für die UI benötigt werden. Eine Auflistung der für die UI erstellten Icons sind der Abbildung 49 zu entnehmen und werden nachfolgend genauer beschrieben.

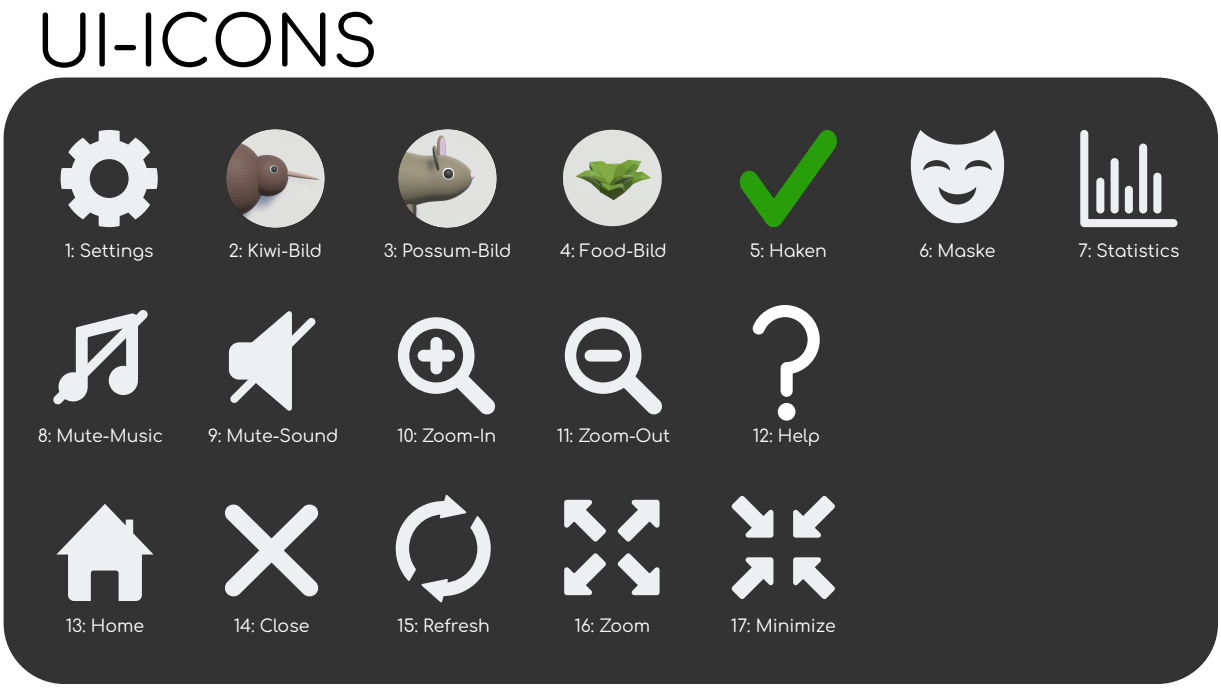

Abbildung 49: UI-Icons

Um das Zahnrad zu erstellen, welches die Möglichkeit bietet, den „AmountSetter“ zu öffnen oder zu schließen, wurde das vorgefertigte Zahnrad von Affinity-Designer genutzt und entsprechende Einstellungen, wie die Lochgröße und Zackenmenge, angepasst.

Um Icon 2, 3 und 4 zu erstellen, wurde ein Screenshot des jeweiligen Objekts gemacht und dieses zunächst in Adobe Photoshop freigestellt. Anschließend wurde ein weißer Hintergrund hinzugefügt und das Gesamtergebnis in eine Kreisform eingepflegt.

Für das „Haken“-Icon wurde das Werkzeug „Formkonstrukteur“ des Affinity Designers genutzt, welches erlaubt Formen zusammenzufügen oder auszuschneiden. Hier wurden zwei Rechtecke abgerundet und miteinander verbunden. Auch die restlichen Icons wurden auf diese Art und Weise erstellt. Zur Ideenfindung für die Symbole wurden sich Icons aus bekannten Programmen, wie z. B. das Lupen-Symbol von Adobe Photoshop oder das Zoom- und Minimieren-Symbol von YouTube, wenn ein Video größer oder kleiner gemacht werden soll, angesehen. Für das Statistics-Icon wurden Inspirationen über Google eingeholt [50], genau wie für das Icon „Maske“ [51]. Im Folgenden werden die Icons für das „AnimalMenu“ (s. Abbildung 50) dargestellt.

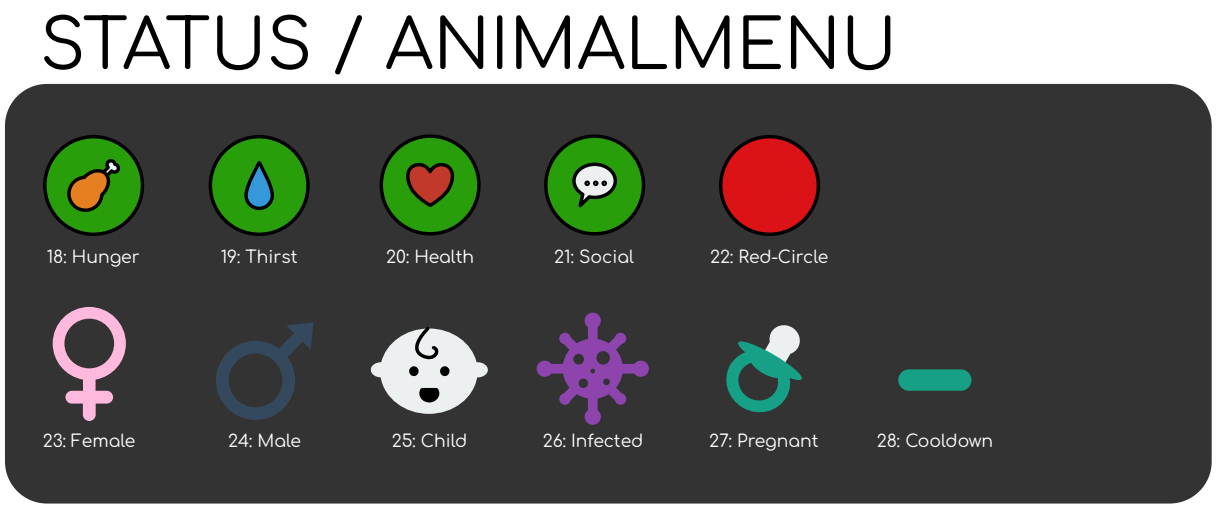

Abbildung 50: Status- / „AnimalMenu“-Icons

Für die Darstellung der Bedürfnisse wurde sich für ein Kreisdesign entschieden, da dieses ermöglicht, die Bedürfnisse nebeneinander anzuzeigen und so eine gute Übersicht über den aktuellen Stand gewährleistet ist. Bei Icon 19, 20 und 21 wurde sich der Formsammlung von Affinity Designer bedient, die Träne, Herz und Sprechblase bereits in der Formsammlung integriert hat. Für das Hunger-Icon wurde eine Fleischkeule als adäquat empfunden, da dieses Symbol oft in Spielen zur Hungeranzeige verwendet wird.

Die weiteren Symbole zeigen den Status des Tieres an. Für die Geschlechter-Icons wurde sich wieder dem „Formkonstrukteur“-Tool bedient, genau wie für das „Infected“-Symbol, bei welchem Google zur Ideenfindung zu Rate gezogen wurde [52]. Für das Icon „Pregnancy“ wurde ein Schnuller als passend empfunden, welcher aus mehreren Formen kreiert wurde. Für den Zustand „onMatingCoolDown“ (Icon 28) wurde eine Art Gedankenstrich genutzt.

3.12.2 3D-MODELL-ERSTELLUNG

BLENDER

Um 3D-Elemente selbst erstellen zu können, wurde das kostenlose 3D-Erstellungs-Programm „Blender“ genutzt. In dieses musste sich vorerst ausgiebig mithilfe von YouTube Tutorials [53] eingearbeitet werden, da noch keine Vorerfahrung mit dem Programm Blender vorhanden war.

BODEN

Zuerst musste ein „Ground“ erstellt werden, auf dem sich die Tiere bewegen sollten. Hierbei wurde nach Naturbildern von Neuseeland gegoogelt. Es wurde versucht, die Eigenschaften dieser Natur mit in die Erstellung der Karte einzubeziehen. Aus diesem Grund ist die Beschaffenheit der Bodenfläche ein wenig hügelig gestaltet worden. Für das Design der Simulation wurde der Low-Poly-Stil gewählt, um die Darstellung spielerisch zu untermalen und einen reduzierten Stil zu erhalten. Im bereits genannten Tutorial [53] wurde schon ein kleines Inselstück im Low-Poly-Stil kreiert. Auf dieselbe Weise, entstand zunächst der im Programm verwendete „Ground“. Als die „Zoom-Out“-Funktion hinzugefügt wurde, wurde, um eine Art Inseloptik zu erzeugen, noch eine tropfenartige Fläche nahtlos unter den eigentlichen Boden gesetzt (s. Abbildung 51).

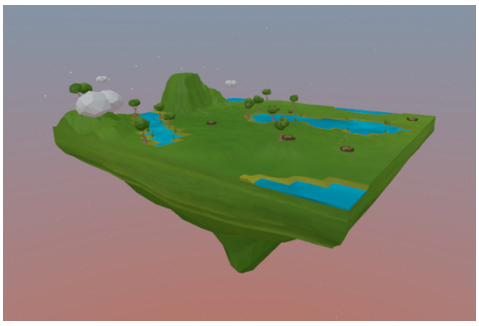

Abbildung 51: Insel

NATURELEMENTE & SYMBOLE

Zunächst war es nötig, die Karte mit in der Natur vorkommenden Elementen auszustatten. Hierzu gehören z. B. Bäume, Pflanzen, Steine und Blümchen. Für die Erstellung dieser Objekte wurden weitere YouTube-Tutorials angesehen, die zunächst aufgeführt werden. Für die Anfertigung jeglicher Naturelemente im Low-Poly-Stil wurde das Tutorial „How to make Low Poly Nature (Blender-Tutorial)“ [54] angesehen. Ein weiteres Tutorial [55] erklärt genauer, wie sich Low-Poly-Bäume in Blender erstellen lassen. Nachdem alle nötigen Elemente erstellt wurden, war die Einarbeitung in das Programm Blender vollbracht und es konnten problemlos weitere Elemente hinzugefügt werden. Eine Auflistung der erstellten Elemente sind in Abbildung 52 zu sehen. Steine, Blumen und Gras wurden passend auf der Karte verteilt, um die Umgebung anschaulicher zu machen. Die Wolken wurden aus den Baumkronen der Bäume erstellt, weswegen sie nicht gesondert in der Abbildung aufgelistet

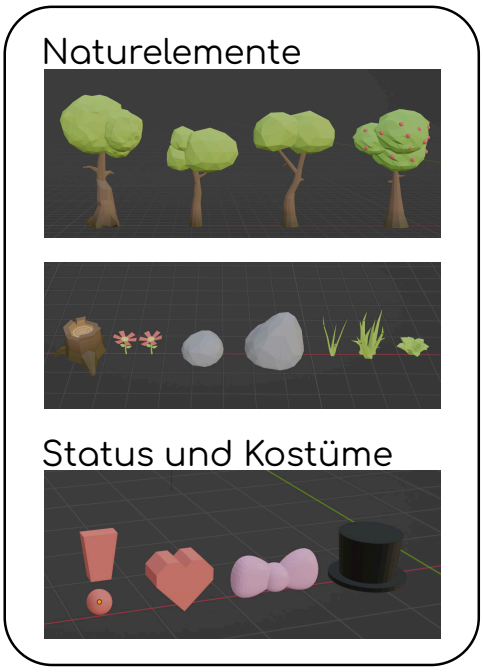

Abbildung 52: Naturelemente & Symbole Blender

wurden. Der Pilzstamm fungiert in überdimensionierter Größe als Nest für den Kiwi. Nester ragen weiter in den Boden hinein, sodass man den Pilzstamm nicht mehr erkennen kann, da die Wurzeln im Boden versteckt sind.

KIWI & POSSUM

Als das Programm aufgesetzt wurde, wurden vorerst als Tiere einfach Kapseln eingesetzt, die sich durch die Karte bewegten, da dies während der Entwicklungsphase der Basis-Funktionen ausreichend war. Später wurde sich jedoch dafür entschieden, die Kiwis und Possums selbst in Blender zu bauen, da nun schon ein wenig Erfahrung mit dem Programm Blender gesammelt wurde und der eigene Stil beibehalten werden sollte. Um dem Low-Poly-Stil treu zu bleiben, wurden die meisten Elemente der Tiere vorerst aus einem simplen Würfel kreiert und zunächst mit dem Modifier „Subdivision Surface“ abgerundet. Die Intensität der Rundung lässt sich über die Einstellung des Modifiers nach Belieben anpassen. Es wurden ein paar Possum und Kiwi-Modelle erstellt, bis das gewünschte Ergebnis erreicht wurde. Die verschiedenen Modelle sind in den folgenden Abbildungen (s. Abbildung 53 und 54) zu sehen.

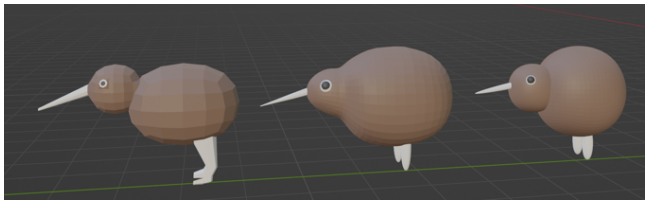

Abbildung 53: Kiwi-Modelle

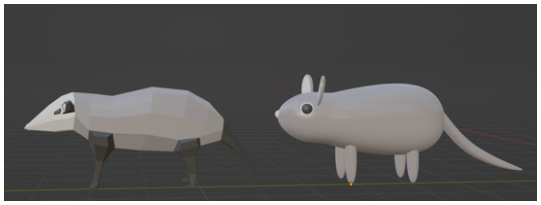

Abbildung 54: Possum-Modelle

Bei beiden Tieren wurde sich jeweils für das letzte abgebildete Modell entschieden, da bei diesen Modellen der Stil am ehesten zum restlichen Design der Simulation passt und mehr Niedlichkeitsfaktor vorhanden ist.

3.13 ANIMATION

3.13.1 RIGGING

Zunächst mussten die Tiere animiert werden, um nicht statisch über die Karte zu wandern. Hierfür war es erforderlich jedem Tier ein Skelett hinzuzufügen (s. Abbildung 55). Diesen Vorgang bezeichnet man als „Rigging“. Da das „Rigging“ bei fehlender Erfahrung eine Herausforderung sein kann, wurde ein entsprechendes YouTube-Tutorial [56] zu Rate gezogen, welches Grundlagen zum Thema „Rigging“ und dem Animieren vermittelte. Um das spätere Animieren zu

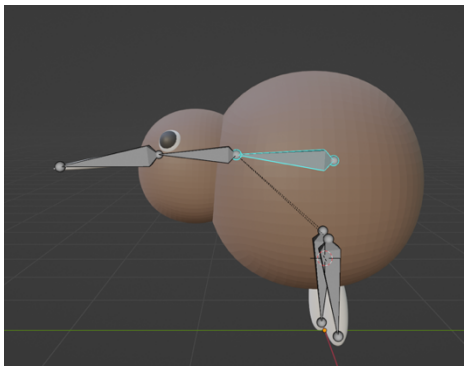

Abbildung 55: Skelett des Kiwis

vereinfachen, wurden jedem Tier möglichst wenig Knochen zugeteilt. Eine komplexe Bewegung wäre aufgrund des Low-Poly-Stils, welcher nicht durch realistische Eigenschaften charakterisiert ist, nicht nötig.

3.13.2 ANIMATION

Zunächst musste überlegt werden, welche Animationen jedes Tier ausführen können muss. Dazu gehört beim Possum das Laufen, das Essen bzw. Trinken und das Angreifen eines Kiwis. Außerdem braucht jedes Tier einen „Idle“-Zustand. Dabei steht das Tier lediglich und atmet. Dieser wurde zwar erstellt, wird allerdings in der Simulation nicht benötigt, da die Animationen nicht über einen Animation-Controller gesteuert werden. Aus diesem Grund startet das Tier schon vordefiniert mit der Lauf-Animation. Der Kiwi benötigt zusätzlich zu den bereits genannten Animationen noch eine Schwimm-Animation, die ausgeführt wird, wenn der Kiwi im Wasser ist. Animationen lassen sich in Blender kreieren, indem man die Knochen im „Pose-Mode“ an die gewünschte Stelle bewegt und dort „Key-Frames“ setzt. Vorher müssen die Knochen jedoch mit dem Modell verknüpft werden, damit sich das Modell auch mitbewegt. Dies ist möglich, indem das Skelett und das Modell gleichzeitig markiert, gruppiert („Parent“) und anschließend „With automatic weights“ aktiviert wird. Jegliche Animationen wurden in Blender erstellt und zunächst zusammen mit dem Modell in Unity eingefügt.

3.13.3 EXPORTIEREN IN UNITY

BERICHTIGEN DER NORMALEN

Beim Exportieren von Blender-Objekten war es wichtig, bestimmte Gegebenheiten vor dem Export zu überprüfen. Dazu gehört z. B. die korrekte Ausrichtung der Normalen. Diese lassen sich mithilfe des Anhakens der „Face Orientation“ sichtbar machen. Sollten die Normalen verkehrtherum sein, so werden diese bei aktivierter „Face Orientation“ rot markiert. Alle blauen Objekte sind richtig herum. Um die Normalen zu berichtigen, ist es nötig auf „Mesh“ und dann „Normals“ zu klicken. Anschließend besteht die Wahl zwischen „Flip“, „Recalculate Outside“ oder „Recalculate Inside“. Überspringt man das Berichtigen der Normalen, so werden die 3D-Modelle in Unity teilweise in einer umgestülpten Optik dargestellt.

Zunächst ist es wichtig vor dem Exportieren die Größe und Rotation des Objekts wieder auf die Standard-Einstellung zurückzusetzen. Dies ist mit der Tastenkombination „Ctrl+A“ und Klick auf das Objekt möglich. Anschließend ist es erforderlich „Rotation & Scale“ auszuwählen, um die Größe und die Rotation zurückzusetzen. Bei der Auswahl von „Location“ zeigt sich das gleiche Verhalten, bezogen auf die Position des Objekts. Damit das Objekt auch richtig herum in Unity erscheint, ist es außerdem nötig, beim Exportieren „Apply Transform“ anzuhaken.

3.13.4 ANIMATION IN UNITY

Nachdem die 3D-Modelle mit sämtlichen Animationen in Unity importiert wurden, war das Konfigurieren der Animationen im Modell selbst nötig. Dies ließ sich umsetzen, indem die Animation benannt wurde und ihr die entsprechende Zeit von Anfang bis Ende zugeteilt wurde (s. Abbildung 56). Üblicherweise wäre der nächste Schritt einen Animation-Controller für das Objekt zu erstellen und dort die Einstellungen und den Ablauf für die Animationen festzulegen. Durch die Nutzung des NavMeshs kam es hierbei allerdings zu Fehlermeldungen. Das Problem wurde gelöst, indem jedem Tier eine „Animation“-Komponente mitgegeben wurde, die am Kind-Objekt des Prefabs anhängt und alle Animationen enthält. Diese können per Code an der entsprechenden Stelle getriggert werden. Das geht z. B. mit dem Befehl „`animator.Play("Swim");`“. Ein Ablauf, der die Animationen beider Tierarten darstellt, ist dem folgenden Aktivitäts-Diagramm (s. Abbildung 57) zu entnehmen, welches mithilfe der Webseite „draw.io“ erstellt wurde.

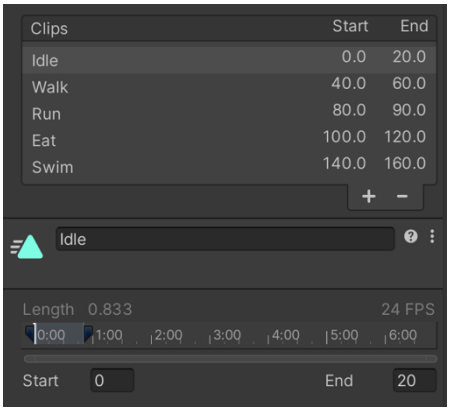

Abbildung 56: Animationen einstellen

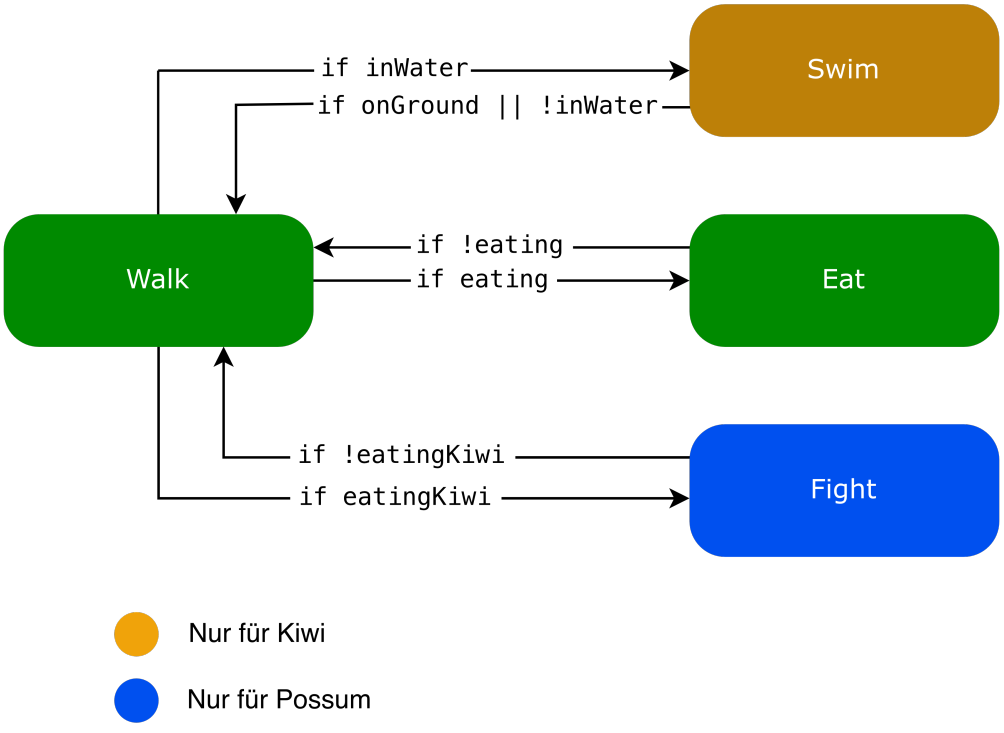

Abbildung 57: Animationsabläufe

3.14 MUSIK & AUDIO

Damit die Simulation lebendiger und nicht zu statisch wirkt, wurde sowohl Sound als auch Musik eingefügt. Für den Sound wurde teilweise Eigenes kreiert, als auch Sounds eines Packs des Unity Asset Stores [57] oder der Pixabay-Sound-Sammlung [58] gewählt.

3.14.1 VERWENDUNG VON SOUNDS

Die Verwendung von Sounds kam z. B. bei Aktionen der Tiere zum Einsatz oder auch bei der Verwendung der UI, sowohl in der Simulation selbst als auch im Startmenü. Für die Aktionen der Tiere, wurden beispielsweise Sounds aus dem oben genannten Pack [57] verwendet, z. B. wenn ein Tier stirbt, wenn es frisst oder wenn ein Possum einen Kiwi frisst. Der Sound, der beim Trinken abgespielt wird, stammt aus der Sound-Bibliothek von „Pixabay“ [58]. Er wurde mithilfe des Programms „Quick Time Player“ zugeschnitten, sodass nur der relevante Teil zu hören ist. Für den „Trächtigkeits-Sound“ wurde eine nach oben gehende Tonfolge mit der App „GarageBand“ kreiert. Weitere eigene Kompositionen werden im Folgenden genauer dargelegt.

3.14.2 EIGENE KOMPOSITIONEN

SOUNDS FÜR NIEDERLAGE & SIEG

Da es in der Simulation zu einer Niederlage oder einem Erfolg kommen kann, wenn eine bestimmte Anzahl von Tieren erreicht ist oder eine Tierart ausgestorben ist, wurden für beide Szenarien Sounds kreiert.

Erhält man eine Niederlage, so wird eine bedrückende Tonabfolge abgespielt, die den Nutzer des Programms vermitteln soll, dass das Ergebnis ungünstig ausfällt. Der Sound trägt den Titel „Lose“ und wurde in „Logic Pro X“ erstellt. Hierfür wurde das Instrument Trompete verwendet, da dieses gut zur Art des Sounds passte.

Bei einem Sieg wird der Sound „Victory“ abgespielt, welcher ebenfalls in „Logic Pro X“ in Zusammenarbeit mit dem Programm „GarageBand“ erstellt wurde. Die Instrumente, die hier verwendet wurden, bilden ein Streicherquartett. Für die Erstellung des Sounds wurden schon fertige Akkorde des Streicherquartetts genutzt. Das Gesamtbild des Sounds ergibt eine fröhliche Tonabfolge, die auf diese Weise adäquat einen Sieg symbolisiert.

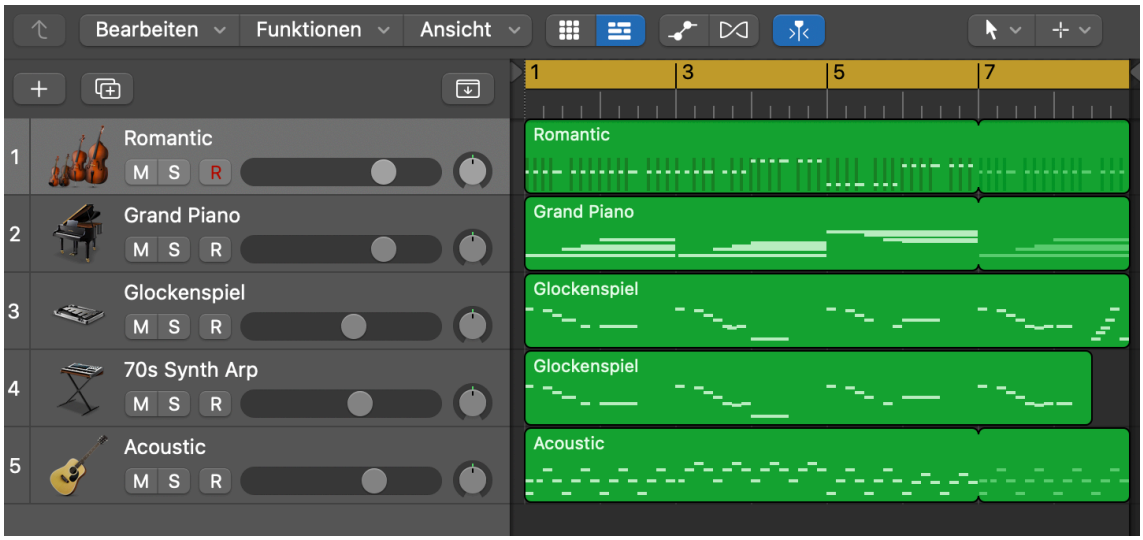

Abbildung 58: GameSong

Als musikalische Untermalung, liegen zwei Musikstücke vor, die mit den Musikerstellungsprogrammen „GarageBand“ und „Logic Pro X“ komponiert wurden. Das Stück „GameSong“ wird während der Laufzeit der Simulation abgespielt, während der Song „StartSong“ im Startmenü zu hören ist. Die beiden Musikstücke haben zwar andere Melodien, jedoch wurden die gleichen Instrumente genutzt, welche als Plug-In in den jeweiligen Programmen vorhanden sind, sodass die Songs zueinander passen. Beide Musikstücke sollen eine entspannte Atmosphäre schaffen und sind aus diesem Grund von den Tonabfolgen und Wechseln eher simpel gehalten. Bei den Musikstücken wurde jeder Ton per Hand an die richtige Stelle der Tonleiter geschoben und die Länge durch Ziehen des Tons angepasst. Die Idee für die beiden Songs ergab sich während des Erstellungsprozesses. Es wurde sich nicht an bereits bestehenden Songs orientiert, sondern nach Gehör gearbeitet. Für den „GameSong“ beispielsweise, wurde zuerst mit dem Instrument „Acoustic-Gitarre“ eine Melodie erstellt und die nachfolgenden Instrumente aus dem Gehör heraus angepasst. Es entstand also immer wieder eine neue Begleitmelodie für die bereits bestehenden Melodien und Instrumente, sodass sich am Schluss ein zusammenpassendes und sich ergänzendes Musikstück ergab. Den Aufbau des „GameSongs“ zeigt Abbildung 58.

3.14.3 AUDIOMANAGER

Um sich beim Verwenden von Sounds und Musik in Unity eine Erleichterung zu schaffen, wurde mithilfe eines YouTube-Videos [59] ein Audio-Manager erstellt, für den das Skript „AudioManager.cs“ und „Sounds.cs“ nötig war. Im „Sound.cs“-Skript wurden alle Variablen definiert, die der Audio-Manager, enthalten soll. Diese können nachher im Inspector angepasst werden. Verknüpft wurde das Ganze dann im „AudioManager.cs“-Skript. Das Einbetten des Audio-Managers, bringt den Vorteil mit sich, dass Sounds im Inspector direkt hinzugefügt werden können. Per Name wird erkannt, welche AudioSource-Komponente benötigt wird, um den Sound abspielen zu können. Diese wird dem Audio-Manager automatisch hinzugefügt, sobald der „Play“-Button gedrückt wird. Hierfür wurde ein leeres Game-Objekt „AudioManager“ erstellt, auf dem das „AudioManager.cs“-Skript liegt (s. Abbildung 59). Ein Abspielen der Sounds ist auch per Skript möglich, indem man z. B.

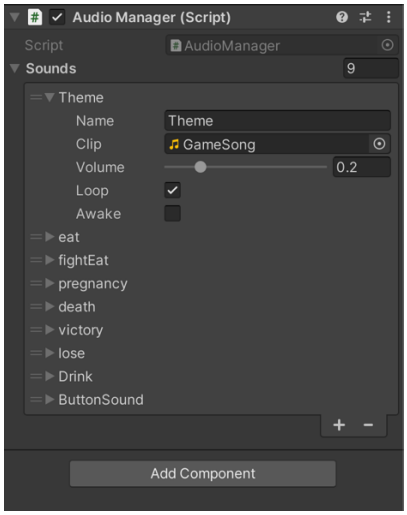

Abbildung 59: Der „AudioManager“

```
FindObjectOfType<AudioManager>().Play("eat");
```

als Codebefehl eingibt. Der Audio-Manager steuert die Sounds der „SampleScene“.

In der „StartScene“, in welcher Sounds z. B. beim Klicken eines Buttons verwendet werden oder zum Abspielen des „StartSongs“, wurde eine andere Art Audio-Manager verwendet, der sich hier „Menu-Sounds“ nennt und mit einem anderen YouTube-Tutorial [60] erstellt wurde. Hierbei wird für jeden Sound ein leeres Game-Objekt erstellt, auf den die relevante Audio-Source gelegt wird. Um z. B. beim Klick auf einen Button einen Sound abzuspielen, muss in den Einstellungen des Buttons das Game-Objekt, auf dem sich der Sound befindet, bei „OnClick“ hinzugefügt werden und „AudioSource.Play“ eingestellt sein.

3.15 BUGS & HERAUSFORDERUNGEN

Bei der Ausarbeitung der Simulation des Ökosystems kam es des Öfteren zu mehr oder weniger problematischen Fehlern und Herausforderungen, die den Arbeitsprozess oft verlangsamt haben und viel Geduld und akribisches Nachdenken erforderten, um funktionierende Lösungsansätze entwickeln zu können. Nachfolgend werden ein paar dieser Bugs herausgegriffen und die entsprechenden Lösungsstrategien, insofern diese gefunden wurden, beschrieben.

3.15.1 ZÄHLER („COUNTER“) MIT PUBLIC STATIC

Um die Anzahlen dokumentieren zu können und das Auslösen bestimmter Funktionen und Ereignisse möglich zu machen, wie z. B. das Anzeigen der Tieranzahl im „AmountSetter“, mussten einige Zähler eingefügt werden, die sich auch „Counter“ nennen. Die Initialisierung dieser „Counter“ befindet sich im „Spawner.cs“-Skript. Die Verwendung der „Counter“ ist in mehreren Skripten nötig, weshalb eine skriptübergreifende Zugänglichkeit erforderlich ist. Um den Schutzgrad beizubehalten, wurde zuerst versucht dies mit „Gettern und Settern“ umzusetzen, was allerdings zur Ausgabe falscher Zahlen führte. Aus diesem Grund wurde sich schließlich doch für die Verwendung vom Schutzgrad „public static“ entschieden, welches den Zugriff für alle Klassen erlaubt.

3.15.2 CAMERA-MOVEMENT: LOKALE- VS. WELTKOORDINATEN

Eine weitere Herausforderung stellte die Achse der Kamera beim Bewegen dar, da sich die Weltkoordinaten und die lokalen Koordinaten eines Objekts unterscheiden, wenn das Objekt, welches in die Welt eingefügt wurde, gedreht wurde. Hier mussten die Werte entsprechend angepasst werden, da sonst die Kamera beispielsweise in die entgegengesetzte Richtung drehte.

3.15.3 NULL-REFERENCE-EXCEPTION: GEFRESSENE NAHRUNG

Ein schwerwiegenderes Problem ergab sich bei folgendem Szenario: Wenn ein Tier ein anderes Tier oder Nahrung gefressen hat, welche in den Detektionslisten „objectsInSightRange“ oder „objectsInSmellRange“ anderer Tiere eingespeichert war, so wurde dieses nicht mehr gefunden und es ergab sich eine Null-Reference-Exception. Dies passiert, wenn Objekte genutzt werden sollen oder angesprochen werden, die nicht mehr vorhanden sind. Die Lösung für dieses Problem zu finden war etwas komplex, da alle Tiere, die das entfernte Objekt in der Liste haben, informiert werden mussten, dass das Objekt verschwunden ist. Um den Rechenaufwand nicht zu sehr auszulasten, wird nicht jedes Tier informiert, sondern nur jedes Tier, welches sich in Reichweite des entfernten Objekts befindet. Zunächst wurden die Methoden „NotifyPossums“ und „NotifyKiwis“ erstellt, welche jedes Possum und jeden Kiwi, der in der Nähe steht, informieren, wenn das gefundene Nahrungsobjekt gefressen und somit entfernt wird. Dies passiert schon bevor das Objekt zerstört wird, um eine weitere Null-Reference-Exception zu vermeiden. In den „Notify“-Methoden wird zunächst die Funktion „GetNotification“ ausgeführt, in der die nicht mehr verfügbaren Objekte nun auch aus den

Listen der Tiere gelöscht werden. Auf diese Weise konnte der Fehler nach einigem Herumprobieren schließlich behoben werden.

3.15.4 FLIEGENDE TIERE

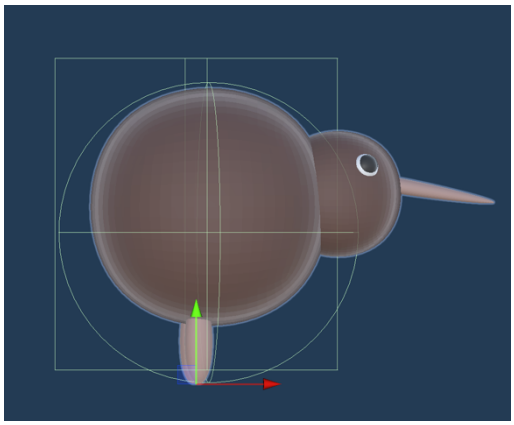

Abbildung 60: Base Offset

Ein kleiner designtechnischer Bug, welcher sich leider nicht ganz beheben ließ, ist das nicht punktgenaue Aufkommen der Tiere auf dem Boden, was sie manchmal leicht fliegend erscheinen lässt. Die Ursache hierfür liegt zum einen in den physikalischen Eigenschaften, da die Gravitation der Tiere ausgeschaltet werden musste, weil sie sich sonst gegenseitig “hin- und herschießen“ würden. Wäre diese aktiviert, so würden die Tiere auf dem Boden landen und es wäre ein Abstand vermieden. Zum anderen ist die Verwendung des „NavMehs“ teilverantwortlich. Da der Einblick in die Mechanik des „NavMeshs“ nicht ganz trivial ist, konnte das Problem nicht gänzlich behoben werden. Um allerdings eine Verbesserung hervorzurufen, wurde in den Optionen der „NavMesh Agent“-Komponente der „Base Offset“-Wert ein wenig heruntergesetzt, sodass die Tiere nicht allzu weit oben sind. Der „Base Offset“ ist die rechteckige Form, die in Abbildung 60 zu sehen ist. Hier ist erkennbar, dass die Füße des Kiwis sogar leicht herausragen, sodass dieser eher im Boden versinken müsste als darüber zu sein. Diese Einstellung hat das „Fliegen“ zwar etwas reduziert, allerdings nicht gänzlich verhindert.

4 DIAGRAMMTESTS

Um die Auswirkungen variierender Werte zu testen, wurden mehrere Simulationsdurchläufe unter unterschiedlichen Bedingungen (verschiedene Startzahlen der Objekte oder verschiedene Werte) vollzogen. Die Ergebnisse, die nun präsentiert werden, wurden anschließend in diesem Kapitel festgehalten. Für die Darstellung der Diagramme wurde das Programm Excel genutzt, da sich die dort erstellten Diagramme besser auf Papier veranschaulichen lassen als die der eingebauten „Statistics-Funktion“. Mit der Funktion „PrintValues()“des „Statistics“-Skripts werden alle Werte im 5 Sekunden-Abstand in der Konsole ausgegeben. Für die Art der Diagramm-Darstellung und dem Aufbau wurde sich an einer bereits bestehenden Bachelorarbeit namens „Simulating an Ecosystem“ der „University of Gothenburg“ [39] inspiriert.

Die Tests fanden mit zwei unterschiedlichen Einstellungen der „Fortpflanzungswerte“ statt. Zum einen wurde mit den in der Ausgabe des Programms verwendeten Werte getestet, welche im Kapitel „Berechnung von Werten rund um die Paarung“ genau erklärt wurden und zum anderen mit erfundenen Werten. Die erfundenen Werte sollten ein schnelleres und prägnanteres Ergebnis liefern.

Die berechneten Werte werden zum einfacheren Vergleich hier noch einmal aufgelistet, wobei die Einheit Sekunden entspricht:

	Trächtigkeitszeit	Erholungszeit	Eilegzeit	Schlüpfzeit	Wachstumszeit
Kiwi	15	7.5	7.5	18.75	22.5
Possum	7.5	124.85	/	/	15

Die erfundenen Werte sehen wie folgt aus:

	Trächtigkeitszeit	Erholungszeit	Eilegzeit	Schlüpfzeit	Wachstumszeit
Kiwi	10	10	10	10	24
Possum	5	10	/	/	18

Wie auffallend sichtbar ist, variieren die Werte nicht stark. Hervorstechend ist aber die Erholungszeit („onMatingCoolDown“) des Possums, die bei den berechneten Werten sehr lange ist, sodass sich das Possum nicht zu oft fortpflanzen kann. Dies sollte gewährleisten, dass das Possum nur 1-2 Babys pro Jahr [18, S. 2] zur Welt bringt. Da nicht nur Weibchen eine solch lange Erholungszeit haben, ist dies nicht ganz realitätsgetreu umgesetzt. Mit der Verwendung einer kürzeren Erholungszeit ist es möglich, die Population des Possums schneller ansteigen zu lassen.

Es ist zu beachten, dass jegliche Ergebnisse nicht mit der Realität gleichzusetzen sind und auch nicht realitätsgetreu getestet werden konnte. Hierfür gibt es zahlreiche Ursachen, wie z. B. das Fehlen von Tag- und Nachtzyklen oder unterschiedlichen Wetterverhältnissen. Auch der begrenzte Lebensraum der Tiere in der Simulation spielt eine große Rolle. Die Resultate können demzufolge lediglich zur Veranschaulichung der Problematik und Testung des Programms verwendet werden. Die verschiedenen getesteten Szenarien werden nachfolgend aufgelistet.

4.1 DIAGRAMMERGEBNISSE

4.1.1 KIWIS VOR EINFÜHRUNG DER POSSUMS

Um ungefähr die Situation vor der Einführung der Possums nachzustellen, wurde hier ein Simulationsdurchlauf mit lediglich der Tierart Kiwis durchgeführt. Es wurde ausreichend Futter (1000) zu Beginn beigefügt und mit einer Kiwi-Population von 30 gestartet. Diese Anzahl der Kiwis sollte zudem die Vermehrung der Kiwis zeigen, die ohne Possums möglich ist. Der Verlauf wurde 360 Sekunden beobachtet, was in der Simulation zwei Jahren entspricht. Die beiden Durchläufe mit berechneten und erfundenen Werten variierten kaum, weshalb hier nur der Durchlauf mit den berechneten Werten dargestellt wird. Es ergaben sich folgende Resultate (s. Abbildung 61: Population und Abbildung 62: Todesursachen).

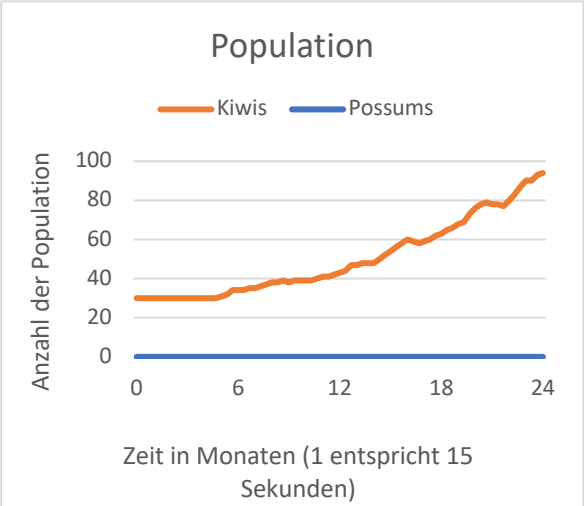

Abbildung 61: Population bei Durchlauf mit 30 Kiwis, 0 Possums und 1000 Nahrung für 360 Sekunden (berechnete Werte)

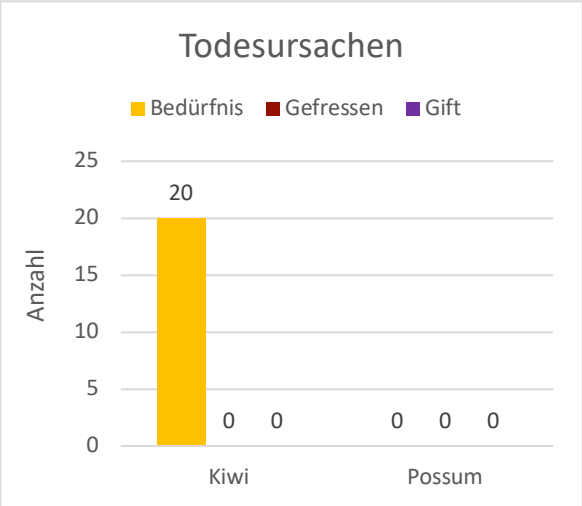

Abbildung 62: Todesursachen bei Durchlauf mit 30 Kiwis, 0 Possums und 1000 Nahrung für 360 Sekunden (berechnete Werte)

Die Kiwis haben innerhalb der Zeitspanne von 2 Simulationsjahren ihre Population von 30 auf 94 erhöhen können, da sie zum einen genügend Nahrung für sich selbst zur Verfügung hatten und zum anderen ungestört viele Eier legen konnten, um sich zu vermehren. Vor allem aber hatten sie keine Feinde, wie das Possum, dass für viele Tode des Kiwis verantwortlich ist. In diesem Durchlauf sind lediglich 20 Kiwis an Hunger oder Durst verstorben. Der Trend der Kiwi-Population geht nach oben, sodass erkenntlich ist, dass sich die Kiwis weiterhin fortpflanzen.

4.1.2 EINFÜHRUNG DER POSSUMS

Zunächst wurde versucht, die Situation darzustellen, in der Possums in das Land Neuseeland eingeführt wurden. Hier wurde mit dem Verhältnis 1:4 gearbeitet, sodass die Kiwis vorerst einen „Vorsprung“ haben. Dies sind keine realistischen Verhältnisse, sondern ein Versuch, die Situation der Kiwis, die anfangs deutlich mehr vorhanden waren als die Possums, darzustellen.

Gestartet wurde der Durchlauf mit 100 Kiwis, 25 Possums und 1000 Nahrung. Hier ist der Unterschied zwischen den berechneten Werten und den erfundenen Werten nennenswert.

Beim Durchlauf mit den berechneten Werten haben die Possums die Kiwis nach 315 Sekunden, entsprechend ca. 2 Simulationsjahren, ausgerottet, jedoch stieg die Population der Possums nur langsam an und hielt sich eher stetig. Dies liegt an der langsameren Fortfplanzung der Possums bei den berechneten Werten. Trotz allem wird die enorme Gefahr der Possums für die Kiwis ersichtlich, da das Fressen der Kiwis den massiven Rückgang ihrer Population begründete (s. Abbildung 63).

Bei den erfundenen Werten sieht man im Gegensatz einen deutlichen Anstieg der Possumpopulation (s. Abbildung 65), was auch die Anzahl der Babys welche hier 70 betragen (s. Abbildung 66) und im Durchlauf mit den berechneten Werten nur 26 (s. Abbildung 64), bekräftigt. Zudem wurden die Kiwis bereits nach 285 Sekunden eliminiert (s. Abbildung 65). Auf den Einsatz von giftigen Pilzen wurde in diesen Durchläufen verzichtet, da die Situation den Beginn der Possumausbreitung darstellen soll, in der noch nicht gegen die Plage in Form von Possumbekämpfung durch Gift vorgegangen werden sollte.

Durchlauf mit berechneten Werten:

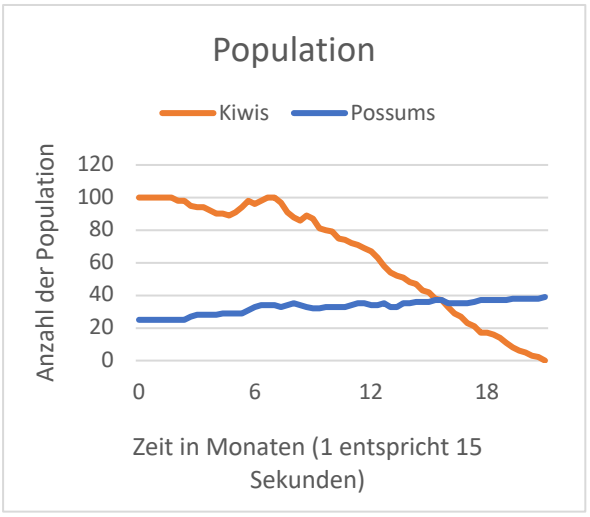

Abbildung 63: Population bei Durchlauf mit 100 Kiwis, 25 Possums und 1000 Nahrung für 315 Sekunden (berechnete Werte)

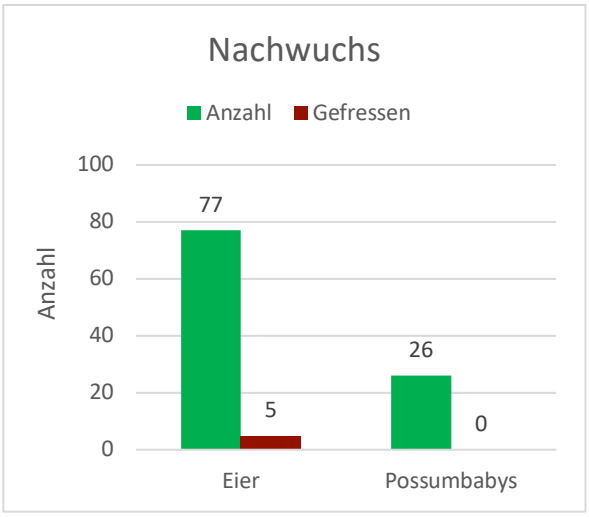

Abbildung 64: Nachwuchs bei Durchlauf mit 100 Kiwis, 25 Possums und 1000 Nahrung für 315 Sekunden (berechnete Werte)

Durchlauf mit erfundenen Werten:

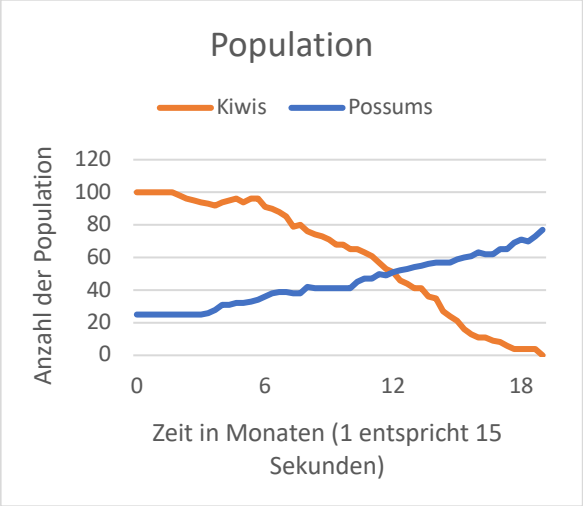

Abbildung 65: Population bei Durchlauf mit 100 Kiwis, 25 Possums und 1000 Nahrung für 285 Sekunden (erfundene Werte)

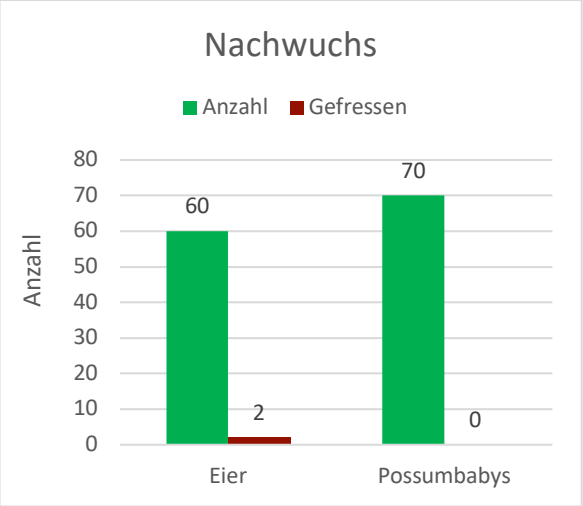

Abbildung 66: Nachwuchs bei Durchlauf mit 100 Kiwis, 25 Possums und 1000 Nahrung für 285 Sekunden (erfundene Werte)

Den Diagrammen ist zu entnehmen, dass die Possums in ihrem momentanen Umfeld gut überleben können, da genug Nahrung für sie bereitsteht und sie keine Fressfeinde haben. Ihre Population steigt stetig an, während gleichzeitig die Population des Kiwis enorm abfällt, da sie von den Possums gefressen werden. Bei beiden Versionen war die Hauptursache des rasanten Rückgangs der Kiwi-Population das Auffressen durch das Possum.

4.1.3 AUSBREITUNG DER POSSUMS

Das nächste Szenario soll eine Situation darstellen, in der sich die Possums bereits ausgebreitet haben und es schon weniger Kiwis gibt. Auch die Nahrungsmenge ist zurückgegangen, da sich die Possums ebenfalls an der verfügbaren Nahrung bedient haben.

Aus diesem Grund wurde mit 80 Kiwis, 50 Possums und 500 Nahrung gestartet. Da die Problematik hier erst richtig zum Vorschein kommt, wurde nur die verstärkte Verbreitung der Possums verzeichnet und noch kein Gifteinsatz vorgenommen. Die Ergebnisse der Durchläufe mit berechneten und erfundenen Werten ähneln sich stark, weswegen auch hier nur der Durchlauf mit den berechneten Werten aufgezeigt wird. Wie in Abbildung 67 zu sehen ist, ist die Population der Kiwis stark abgefallen, während die Population der Possums rasant anstieg. Die Ursache hierfür ist zum einen die enorme Vermehrung der Possums (s. Abbildung 67) und zum anderen das Auffressen des Kiwis durch das Possum (s. Abbildung 68).

Im Vergleich zum vorherigen Durchlauf, bei dem die Possums eingeführt wurden, ist hier ein deutlich schnellerer Rückgang der Kiwis und ein rascherer Anstieg der Possum-Population zu verzeichnen.

Außerdem fällt auf, dass die Kiwis durch die vielen Feinde kaum Gelegenheit haben, sich fortzupflanzen. Sie haben nur 9 Eier gelegt, von denen 4 gefressen wurden (s. Abbildung 69).

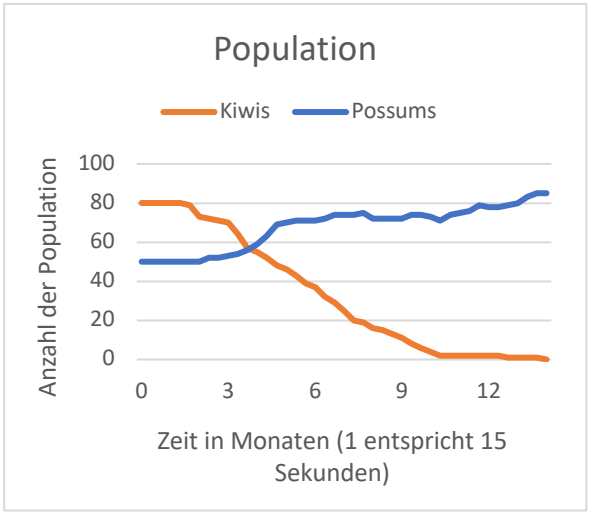

Abbildung 67: Population bei Durchlauf mit 80 Kiwis, 50 Possums und 500 Nahrung für 210 Sekunden (berechnete Werte)

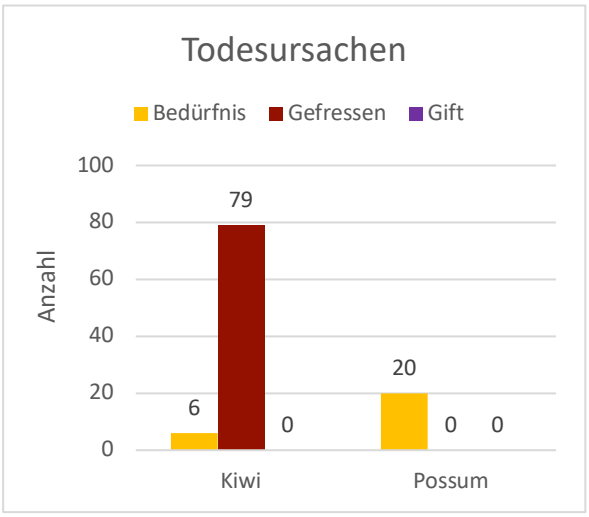

Abbildung 68: Todesursachen bei Durchlauf mit 80 Kiwis, 50 Possums und 500 Nahrung für 210 Sekunden (berechnete Werte)

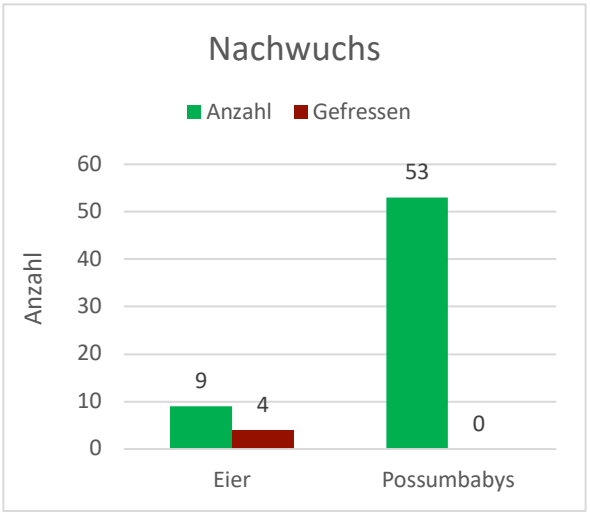

Abbildung 69: Nachwuchs bei Durchlauf mit 80 Kiwis, 50 Possums und 500 Nahrung für 210 Sekunden (berechnete Werte)

4.1.4 BEKÄMPFUNGSVERSUCH DER POSSUMS (MODERAT)

Um die Problematik der Possum-Ausbreitung ein wenig einzudämmen, wurden, unter gleichen Bedingungen zum vorherigen Test (mit Ausnahme der Nahrungsmenge, die mehr sein musste, damit genügend Giftpilze spawnen) bei diesem Durchlauf giftige Pilze eingefügt. Zunächst wurde ein Durchlauf mit moderatem Einsatz von giftigen Pilzen (giftige Pilze spawnen zu 20%) gestartet. Auch

hier unterscheiden sich die Ergebnisse unter Verwendung verschiedener Fortpflanzungswerte kaum. Zu erwähnen ist jedoch, dass mit den erfundenen Werten die Kiwi-Population wieder begann zu wachsen (s. Abbildung 72), während sie bei den berechneten Werten, genau wie die Anzahl der Possums, sank (s. Abbildung 70). Bei Betrachtung der Todesursachen war klar, dass die Ursache hierfür, das vermehrte Fressen von giftigen Pilzen war (s. Abbildung 71). Im Durchlauf mit den erfundenen Werten haben Possums sehr viele giftige Pilze gefressen (s. Abbildung 73). Bei beiden Durchläufen ist jedoch zu erkennen, dass sich die Kiwi-Population trotz des Bekämpfungsversuchs stark reduziert hat.

Durchlauf mit berechneten Werten:

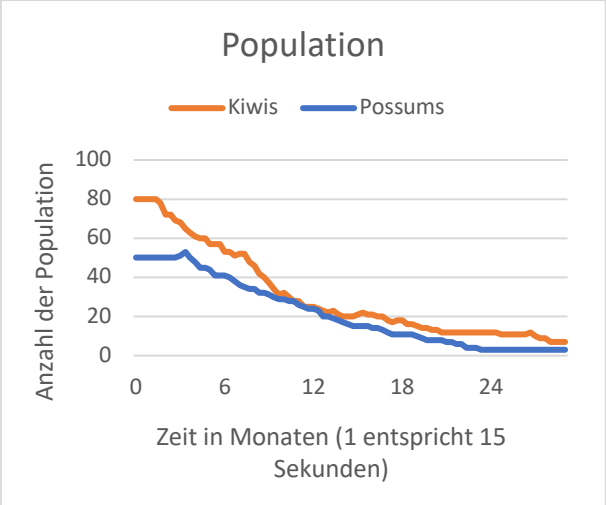

Abbildung 70: Population bei Durchlauf mit 80 Kiwis, 50 Possums und 1000 Nahrung für 435 Sekunden (berechnete Werte)

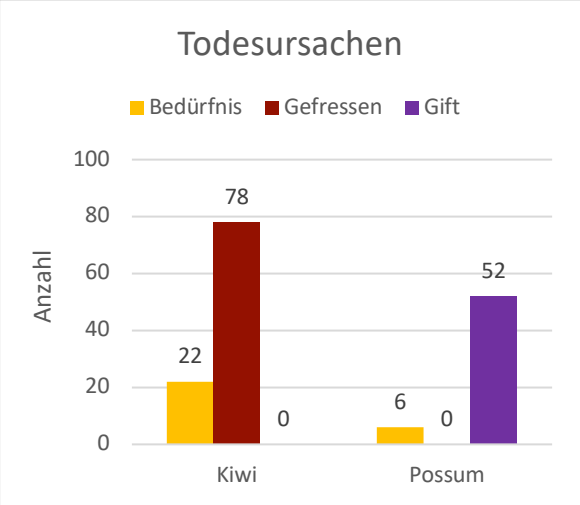

Abbildung 71: Population bei Durchlauf mit 80 Kiwis, 50 Possums und 1000 Nahrung für 435 Sekunden (berechnete Werte)

Durchlauf mit erfundenen Werten:

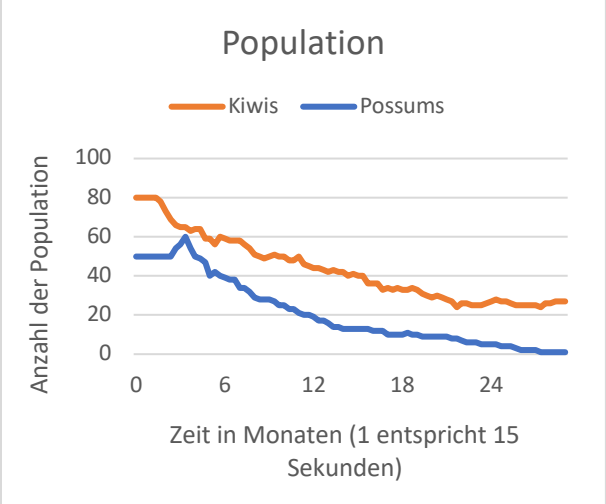

Abbildung 72: Population bei Durchlauf mit 80 Kiwis, 50 Possums und 1000 Nahrung für 435 Sekunden (erfundene Werte)

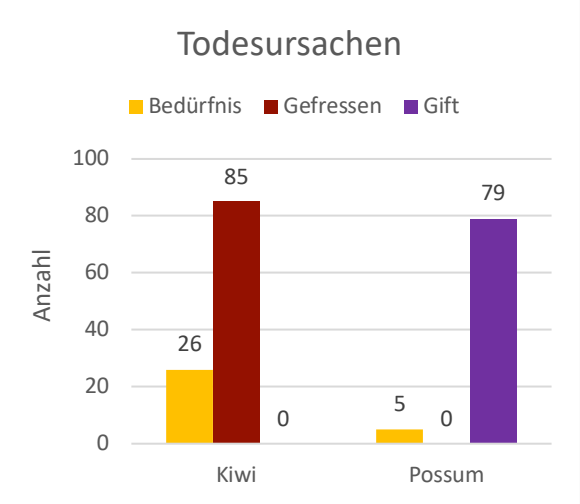

Abbildung 73: Population bei Durchlauf mit 80 Kiwis, 50 Possums und 1000 Nahrung für 435 Sekunden (erfundene Werte)

4.1.5 BEKÄMPFUNGSVERSUCH DER POSSUMS (STARK)

Da der vorherig gezeigte Versuch der Possumbekämpfung der Kiwi-Bevölkerung nur mäßig zugutekam, wird hier ein stärkerer Ansatz der Bekämpfung genutzt (giftige Pilze spawnen zu 40%). Die Startanzahlen bleiben gleich, um den Unterschied der größeren Giftmenge besser erkennbar zu machen. Unter Verwendung der berechneten Werte fällt auf, dass die Possum-Population rasch stark abfällt, während die Kiwi-Population vorerst um minimale Werte fluktuiert, sich aber stabil hält und später wieder ansteigt (s. Abbildung 74). Mit den erfundenen Werten fällt die Kiwi-Population erstmal mit den Possums ab und steigt erst gegen Ende wieder an. Beide Simulationen liefen für 260 Sekunden. Es ist zu erkennen, dass bei einem stärkeren Einsatz von Giftpflanzen schneller ein Rückgang der Possum-Population erreichbar ist (s. Abbildung 74 und 75), weswegen die Kiwis eher eine Chance haben, sich zu vermehren und ihre Population stabil zu halten (s. Abbildung 74 und 76). Zudem werden sie weniger gefressen. Da die Grundaussage für beide Durchläufe getroffen werden kann, wird hier nur der Durchlauf mit den berechneten Werten gezeigt.

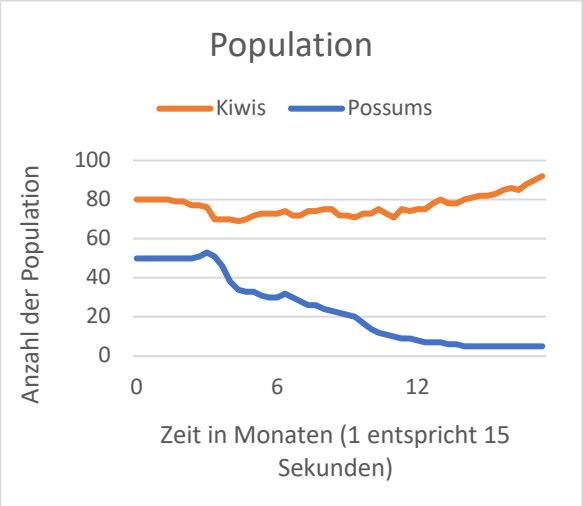

Abbildung 74: Population bei Durchlauf mit 80 Kiwis, 50 Possums und 1000 Nahrung für 260 Sekunden (berechnete Werte)

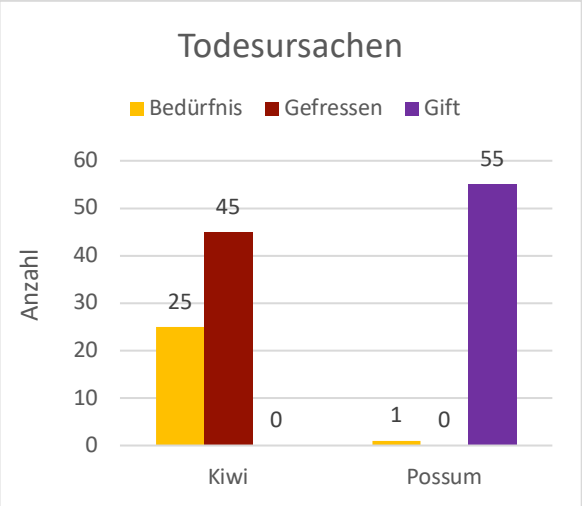

Abbildung 75: Population bei Durchlauf mit 80 Kiwis, 50 Possums und 1000 Nahrung für 260 Sekunden (berechnete Werte)

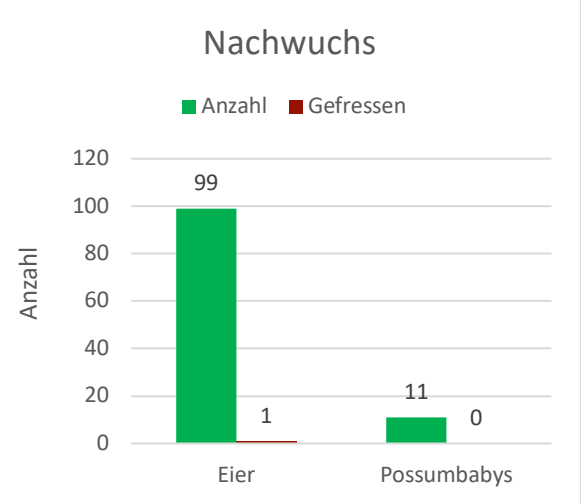

Abbildung 76: Population bei Durchlauf mit 80 Kiwis, 50 Possums und 1000 Nahrung für 260 Sekunden (berechnete Werte)

4.2 DISKUSSION & BEWERTUNG DER ERGEBNISSE

Anhand der Diagrammergebnisse lassen sich weder allgemeingültige Aussagen in Bezug auf die Realität noch Prognosen treffen. Hierfür fehlen einige Eigenschaften, die zur realitätsgetreuen Darstellung und Bewertung erforderlich sind.

Was man aber feststellen kann ist, dass die Simulation in der Lage ist, die Problematik der Possums in Bezug auf die Kiwis zu veranschaulichen, indem der Rückgang der Kiwis bei zu vielen Possums deutlich wird. Auch die Veranschaulichung einer Possumbekämpfung ist mit Einbauen der giftigen Pilze gelungen, da einige Possums, wie in den Statistiken sichtbar ist, an diesen verenden.

Um die Situation realitätsgetreuer nachzustellen wäre es z. B. erforderlich eine größere Karte einzupflegen. Auf begrenztem Raum haben die Possums ein leichtes Spiel, da sie schnell Kiwis detektieren können, die sich auch nicht wie in der Realität verstecken können, sondern lediglich davonlaufen. Im Allgemeinen bekräftigen die Diagrammergebnisse die in der Simulation dargestellte Problematik Neuseelands, können aber nicht für reale Forschungen oder Problemlösungen hergenommen werden.

5 NUTZERTEST & FEEDBACK

Um die umgesetzte Simulation in Funktion und Empfinden besser einschätzen zu können wurde ein Nutzertest durchgeführt. Hierfür wurden 8 Testpersonen im Alter von 23 – 64 Jahren gebeten, die Simulation, welche auf „itch.io“ unter folgendem Link zu finden ist: https://miafoni.itch.io/ecosys zu testen und anschließend den über Google Docs selbst erstellten Fragebogen, welcher bei Klick auf nachfolgenden Link zu sehen ist: https://forms.gle/aAhQ3p4RBzdE1Rub8, auszufüllen. Dieser enthält vor allem Fragen zum Verständnis und zur richtigen Erfassung des Themas, aber auch zur Umsetzung des Designs, der UI und der Musik. Unter den Testpersonen ist oder waren die Hälfte der Personen Media Engineering-Studenten. Bei den restlichen Testern handelt es sich nur teilweise um technikaffine Menschen. Es gab 50% männliche und 50% weibliche Teilnehmer.

5.1 THEMENBEZOGENE FRAGEN

VERSTÄNDNIS DES THEMAS

Bei der Frage, ob die Thematik erkennbar war, haben die meisten mit 1 (Ja, sehr) geantwortet. Den Anmerkungen war zu entnehmen, dass die Problematik „verbildlicht“ wurde. Auf Anhieb wurde das Thema nicht von allen Teilnehmern erkannt und z. B. dazu geraten, die Bilder der Possums und Kiwis noch mit der jeweiligen Bezeichnung zu beschriften, um genauer klarzumachen, um was es geht, falls die Beschreibung des Programms nicht gelesen wurde. Ein Teilnehmer merkte an, dass die Dringlichkeit der Problematik nur bedingt erkennbar ist und auf diese noch verstärkter hingewiesen werden müsste.

SINNHAFTIGKEIT DES THEMAS

Auf die Frage, ob das Thema sinnvoll aufzugreifen sei, wurde vorwiegend positiv reagiert. Es wurde als wichtig empfunden, auf die Problematik hinzuweisen, sowie für eine „coole Idee“ gehalten, die Thematik durch eine „spielerische Simulation“ darzustellen. Einige merkten an, vorher noch nichts von dem Problem gehört zu haben und durch die umgesetzte Simulation erst darauf aufmerksam wurden.

ERFOLGSMÖGLICHKEIT FÜR DEN KIWI

Bei der Frage, ob die Possums in einem der getesteten Durchläufe jemals verloren haben, ergab sich ein 50:50 Resultat. Ein Teilnehmer merkte an, dass auch bei den Startwerten „50 Kiwis und 2 Possums“ das letzte Possum nicht starb. Andere Teilnehmer haben es geschafft, dass die Kiwi-Population gewinnt. Dies war z. B. mit den Zahlen „10 Kiwis und 2 Possums“ oder „20 Kiwis und 4 Possums“ möglich. Sind also die Kiwis in 5-facher Anzahl vorhanden, so ist die Wahrscheinlichkeit hoch, dass sie gewinnen.

5.2 FRAGEN ZU UI & BEDIENUNG

FRAGEN ZUR STEUERUNG

Auf die Fragen, ob die Steuerung intuitiv erkennbar ist oder nach dem Durchlesen der Anleitung („Controls“) klar wird, wurde ebenfalls meist positives Feedback gegeben. Die Steuerung mit der Tastatur wurde als „gewohnt“ bezeichnet und das Rotieren mit der Maus als „super“ betitelt. Als Wunsch wurde das Herein- und Herauszoomen mit dem Mausrad genannt. Ein Teilnehmer empfand die Möglichkeit der Steuerung nicht als sehr wichtig, da er keinen Sinn darin sehen konnte, außer die Tiere aus der Nähe zu betrachten.

FRAGEN ZUR ERKENNBARKEIT DER FUNKTIONEN DER BUTTONS

Die Buttons, welche im „AnimalMenu“ zu sehen sind, wurden von den meisten Testern in ihrer Funktion erkannt. Eine Teilnehmerin beteuerte, dass nicht alle Buttons sichtbar waren, was an der Nutzung eines Laptops lag, für dessen Auflösung die Simulation nicht geeignet ist. Ein weiterer Teilnehmer hatte Schwierigkeiten, den Kostümmodus und den Nutzen der Lupen zu erkennen. 75% der Testpersonen erkannten die Funktion der Buttons sehr gut, während 2 Teilnehmer eine eher negative Bewertung abgaben (s. Abbildung 77).

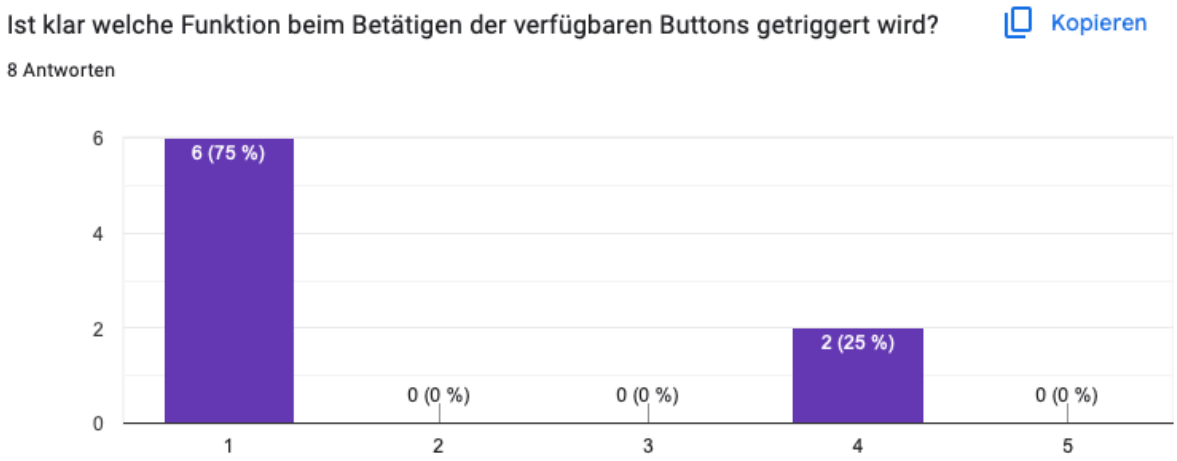

Abbildung 77: Google-Docs Fragebogen, Frage zur Erkennbarkeit der Funktion der Buttons

5.3 FRAGEN ZUR GESTALTUNG

FRAGEN ZUM DESIGN

Das Design wurde von nahezu allen Teilnehmern mit „sehr gut“ bewertet. Ein Tester beschrieb das Design als „Highlight der Simulation“ und hob das durchgezogene Design im Low-Poly-Stil und die „perfekte Abstimmung“ besonders hervor. Andere Testpersonen beschrieben die Gestaltung als „süß“, „schlicht“ und „übersichtlich.

FRAGEN ZU MUSIK & SOUND

Bei der Frage, ob die Musik während der Programmlaufzeit angenehm sei, unterschied sich die Meinung der Tester stark. Manche empfanden die Musik als „entspannend“, „angenehm“ oder „sanft“, während andere Worte wie „eintönig“ oder „störend“ als Bewertung abgaben. Einige hielten die Ausschaltfunktion für sehr sinnvoll. Bei der Erkennbarkeit der Sounds ist anhand der Anmerkungen der Tester zu erkennen, dass diese nicht direkt zuordenbar waren und auch nicht klar war, was sie bedeuten. Vor allem der „Sterbesound“, welcher mit einem Knall oder „Stampf-Geräusch“ einhergeht, wurde als „verwirrend“ empfunden.

5.4 VERBESSERUNGSVORSCHLÄGE

Als Verbesserungsvorschläge für die Simulation wurden verschiedene Dinge genannt. Im Folgenden wird auf ein paar der Vorschläge genauer eingegangen. Es wurde z. B. ein Vorspulmodus vorgeschlagen und eine begrenzte Eingabe der Tieranzahl, sodass das Programm nicht zu Beginn langsam ist oder Tiere „hängenbleiben“. Ein anderer Teilnehmer würde gerne mit realistischen Zahlen arbeiten und hätte sich eine Information darüber gewünscht, dass Tiere angeklickt werden können, um genaue Werte von ihnen zu sehen. Des Weiteren hätte sich eine Teilnehmerin die Simulation zweisprachig gewünscht, da sie kein Englisch spricht. Außerdem wurde die Möglichkeit des Wechselns zwischen verschiedenen Musikstücken vorgeschlagen.

5.5 FAZIT

STELLUNGNAHME

Im Folgenden wird zu einigen Aussagen der Tester Stellung genommen, um die Vorgehensweise bei der Umsetzung verständlich zu machen. Dass ein Possum in der Simulation ewig überlebt, liegt daran, dass es so lange lebt, wie es Futter findet, da es weder Feinde noch ein eingebautes Alter besitzt, welche den Tod für das Possum bedeuten könnten. Eine weitere Möglichkeit wäre, dass das Possum einen giftigen Pilz zu sich nimmt. Passiert das jedoch nie und es findet immer Nahrung, so bleibt es am Leben. Die Betitelung der Tiere beim Festlegen der Anzahl wurde aus Platzmangel und Designgründen vermieden. Beim Zuordnen der Sounds, die die Tiere auslösen, wäre es sinnvoller, nur Sounds bei dem angeklickten Tier abzuspielen, um diese besser zuordnen zu können. Die Umsetzung hierfür hätte allerdings den zeitlichen Rahmen gesprengt, sowie auch das Hinzufügen mehrerer Musikstücke, da die Musik der Simulation selbst erstellt wurde und weitere Stücke einen größeren Zeitaufwand bedeutet hätten. Im Bereich „Ausblick“ werden genannte Verbesserungsvorschläge noch einmal genauer aufgegriffen.

ZUSAMMENFASSUNG

Den Ergebnissen ist zu entnehmen, dass die Simulation im Gesamten gelungen ist aber noch ein paar Verbesserungen vorgenommen werden könnten. Besonders positiv wurde das Design bewertet. Ansonsten wurde die Steuerung und das Thema vorwiegend verstanden und positiv betrachtet.

6 ZUSAMMENFASSUNG, FAZIT UND AUSBLICK

6.1 ZUSAMMENFASSUNG

Nachdem sich dafür entschieden wurde, ein Ökosystem zu simulieren, wurden verschiedene Themen recherchiert, die interessant wären aufzugreifen. Als ein spannendes Themengebiet wurden hierbei invasive Arten empfunden, die ein Ökosystem aus dem Gleichgewicht bringen können. Aus diesem Grund wurden unterschiedliche Fälle recherchiert, in denen Tiere in andere Länder importiert wurden und nun dort das Ökosystem stören oder eine Bedrohung für die dort lebenden Tierarten darstellen. Es wurde sich schließlich dafür entschieden, eine Simulation zu erstellen, die auf die Problematik Neuseelands, die durch das Einführen des Possums entstand und insbesondere Vogelarten, wie den Kiwi bedroht [5], aufmerksam macht. Nach einem Theorieteil, der genaueren Aufschluss über Ökosysteme gibt und die genannte Problematik umfassender darlegt, folgt eine ausführliche Erklärung zur Erstellung des Programms. Nach Fertigstellen der Simulation wurden Tests mit verschiedenen Durchläufen ausgeführt, welche im Abschnitt „Diagrammtests“ einzusehen sind. Auch ein Nutzertest wurde erstellt, über den in dieser Arbeit nachzulesen ist.

6.2 FAZIT

Im Allgemeinen lässt sich sagen, dass die Erstellung der Simulation ein Erfolg war, da die Problematik durch den deutlich sichtbaren Rückgang der Kiwis meiner Meinung nach erkenntlich wird. Es ist ein Programm entstanden, das dem Nutzer die Möglichkeit zur Beobachtung des Tierverhaltens gibt und durch die eingebaute Steuerung, sowie andere Features, wie den Kostümmodus, den Nutzer auch ein wenig interaktiv werden lässt. Allgemeingültige Schlüsse auf die Realität lassen sich aufgrund vieler fehlender Einflüsse, wie Wetterverhältnisse, Mutationen, Krankheiten und einige andere Eigenschaften, allerdings nicht ziehen. Das Einbauen dieser Komponenten war aufgrund des zeitlichen Rahmens auch nicht möglich. Da die Intention dieser Arbeit jedoch lediglich ist, ein Bewusstsein für die Problematik zu schaffen und eine anschauliche Simulation zu erstellen, bin ich mit dem Ergebnis sehr zufrieden. Gleichwohl gibt es noch einige mögliche Optimierungen, die sich in die Simulation einbauen lassen würden. Diese werden im nachfolgenden Kapitel „Ausblick“ genauer geschildert.

6.3 AUSBLICK

Aufgrund des zeitlichen Rahmens war es nicht möglich alle Ideen einzubauen. Aus diesem Grund werden nun Verbesserungen und Weiterentwicklungsmöglichkeiten erläutert. Die Diagrammtests wurden zum einen mit berechneten Werten, die mithilfe von Recherche entstanden sind und mit erfundenen Werten durchgeführt. Damit der Nutzer selbst steuern kann, welche Werte verwendet werden, könnte z. B. ein Auswahlfeld hierfür in den „Options“ des Startmenüs angebracht werden, indem entweder zwischen beiden Optionen gewechselt werden kann oder jegliche Werte per Hand

gesetzt werden können. Zunächst könnte eine größere Beeinflussungsmöglichkeit für den Nutzer eingebaut werden, in dem eine gezielte Possumbekämpfung, durch z. B. das manuelle Streuen von Gift oder giftigen Pflanzen per Button verfügbar gemacht wird. Wie auch von einem Tester der Simulation vorgeschlagen, bestand bereits die Idee einen Vorspulmodus einzubauen, sodass schneller Ergebnisse sichtbar werden. Auch dies könnte bei einer Weiterentwicklung der Simulation hinzugefügt werden. Ebenso war es angedacht, einen „Pausieren"-Modus einzubauen, sodass bestimmte Situationen besser aus der Nähe beobachtet werden können, da manche Ereignisse schnell wieder vorbei waren. Um noch mehr Variation in die Simulation zu bringen, wäre auch das Implementieren von Mutationen eine gute Weiterentwicklungsmöglichkeit, die allerdings noch einige Zeit in Anspruch nehmen würde. Im Allgemeinen können Simulationen wie diese auf eine Problematik hinweisen und zum Nachdenken anregen.

7 LITERATURVERZEICHNIS

[1] M. Gehrig, „Australien: Die unerwünschte Kröte“, 5. Januar 2018. https://www.umweltnetz-schweiz.ch/themen/naturschutz/2755-kroetenplage-australien.html (zugegriffen 18. Juni 2023).

[2] Unbekannter Autor, „Biberplage“, *National Geographic*, 16. November 2011. https://www.nationalgeographic.de/tiere/biberplage (zugegriffen 18. Juni 2023).

[3] H. KlŸche, *Neuseeland*. Dumont Reiseverlag, 2013.

[4] Unbekannter Autor, „Fuchskusu“. https://www.biologie-seite.de/Biologie/Fuchskusu (zugegriffen 19. Juni 2023).

[5] M. Stadler, „Kiwi und andere Vögel in Neuseeland stark bedroht“, *Neue Zürcher Zeitung*, 1. Juni 2021. Zugegriffen: 18. Juni 2023. [Online]. Verfügbar unter: https://www.nzz.ch/panorama/kiwi-und-andere-voegel-in-neuseeland-stark-bedroht-ld.1625624

[6] J. Kaiser, „Artensterben in Neuseeland - Dem Kiwi droht das Ende“, *Deutschlandfunk Kultur*, 16. Juli 2020. https://www.deutschlandfunkkultur.de/artensterben-in-neuseeland-dem-kiwi-droht-das-ende-100.html (zugegriffen 4. April 2023).

[7] G. Toepfer, „Ökosystem“, in *Historisches Wörterbuch der Biologie: Geschichte und Theorie der biologischen Grundbegriffe. Band 2: Gefühl — Organismus*, G. Toepfer, Hrsg., Stuttgart: J.B. Metzler, 2011, S. 715–745. doi: 10.1007/978-3-476-00455-0_37.

[8] Unbekannter Autor, „Ökosystem“, *Biologie-Schule.de*. https://www.biologie-schule.de/oekosystem.php (zugegriffen 18. Juni 2023).

[9] Unbekannter Autor, „Ernährung - Kompaktlexikon der Biologie“, *Spektrum.de*. https://www.spektrum.de/lexikon/biologie-kompakt/ernaehrung/3834 (zugegriffen 18. Juni 2023).

[10] Unbekannter Autor, „BMEL-Statistik: Schweinezyklus“. https://www.bmel-statistik.de/preise/preise-fleisch/schweinezyklus# (zugegriffen 18. Juni 2023).

[11] Unbekannter Autor, „Schweinezyklus | Ginmon Finanz-Wiki“, *Ginmon*, 29. Mai 2021. https://www.ginmon.de/wiki/schweinezyklus/ (zugegriffen 18. Juni 2023).

[12] L. Rau, „Ökologisches Gleichgewicht: Das steckt dahinter“, *Utopia.de*, 5. Oktober 2021. https://utopia.de/ratgeber/oekologisches-gleichgewicht-das-steckt-dahinter/ (zugegriffen 18. Juni 2023).

[13] Unbekannter Autor, „ökologisches Gleichgewicht“, *Spektrum.de*. https://www.spektrum.de/lexikon/biologie/oekologisches-gleichgewicht/47478 (zugegriffen 18. Juni 2023).

[14] Unbekannter Autor, „Sukzession • Definition und Beispiel Wald und See“, *Studyflix*. https://studyflix.de/biologie/sukzession-2470 (zugegriffen 18. Juni 2023).

[15] H. J. Boehmer, „Biologische Invasionen – globale Herausforderung oder lokales Problem? (Biological Invasions – global challenge or local problem?)“, *Natur und Landschaft*, Bd. 83, S. 394–398, Sep. 2008.

[16] T. Jack, „Australiens Kaninchen: Wie ein Farmer die Plage verursachte“, *Süddeutsche.de*, 25. August 2022. https://www.sueddeutsche.de/wissen/australien-kaninchen-plage-1.5644992 (zugegriffen 18. Juni 2023).

[17] J. H. Voigt, *Australien*. C.H.Beck, 2000.

[18] Department of Conservation, „A pest of plague proportions“. https://www.doc.govt.nz/documents/science-and-technical/everybodyspossum.pdf (zugegriffen 18. Juni 2023).

[19] R. C., „Why New Zealand Hates Possums“, *NZ Pocket Guide*, 12. September 2015. https://nzpocketguide.com/new-zealand-hates-possums/ (zugegriffen 4. April 2023).

[20] mep/dpa, „Vögel Neuseelands in Gefahr: Natur bräuchte Millionen Jahre zur Regeneration“, *The Weather Channel*, 5. August 2019. https://weather.com/de-DE/wissen/tiere/news/2019-08-05-verheerende-aussterbewelle-in-neuseeland-schuld-ist-der-mensch (zugegriffen 6. März 2023).

[21] Department of Conservation, „Department Of Conservation in Neuseeland | Sehenswürdigkeiten & Aktivitäten in New Zealand“. https://www.newzealand.com/de/department-of-conservation/ (zugegriffen 19. Juni 2023).

[22] Unbekannter Autor, „Possum“, *AZ Animals*. https://a-z-animals.com/animals/possum/ (zugegriffen 4. April 2023).

[23] Unbekannter Autor, „Wissenswertes über den Fuchskusu“, 16. April 2019. https://donaufischer.com/zoo/fuchskusus.htm (zugegriffen 19. Juni 2023).

[24] Unbekannter Autor, „Fuchskusu - Fakten, Ernährung, Lebensraum & Bilder auf Animalia.bio“. https://animalia.bio/de/common-brushtail-possum (zugegriffen 19. Juni 2023).

[25] Unbekannter Autor, „Kiwi“, *AZ Animals*. https://a-z-animals.com/animals/kiwi/ (zugegriffen 4. April 2023).

[26] H. Lübben, „Der Kiwi – Nationalvogel Neuseelands“, *Vogel & Natur - Onlinemagazin für Vogelbeobachtung*, 15. April 2022. https://www.vogelundnatur.de/vogelarten-kiwi/ (zugegriffen 4. April 2023).

[27] Unbekannter Autor, „What kiwi eat“, *Save the Kiwi*. https://savethekiwi.nz/about-kiwi/kiwi-facts/what-kiwi-eat/ (zugegriffen 19. Juni 2023).

[28] Unbekannter Autor, „Kiwi life cycle“, *Save the Kiwi*. https://savethekiwi.nz/about-kiwi/kiwi-facts/kiwi-life-cycle/ (zugegriffen 11. Juni 2023).

[29] Unbekannter Autor, „Kiwi | San Diego Zoo Animals & Plants“, *San Diego Zoo Wildlife Alliance Animals & Plants*. https://animals.sandiegozoo.org/animals/kiwi# (zugegriffen 20. Juni 2023).

[30] S. Lague, „Coding Adventure: Simulating an Ecosystem“, 10. Juni 2019. https://www.youtube.com/watch?v=r_It_X7v-1E (zugegriffen 6. März 2023).

[31] A. Sajus, „Creating an Ecosystem in Unity (Unity 3D Prey Predator System) - YouTube“, 8. März 2023. https://www.youtube.com/watch?v=RyfW9GjyoxA&t=1s (zugegriffen 2. Juli 2023).

[32] Primer, „Natürliche Selektion simulieren - YouTube“, 15. November 2018. https://www.youtube.com/watch?v=0ZGbIKd0XrM&t=1s (zugegriffen 1. Juli 2023).

[33] M. Mißfeldt, „Was ist das Sichtfeld (Gesichtsfeld)?“, 2023. https://www.brillen-sehhilfen.de/auge/gesichtsfeld.php (zugegriffen 2. Juli 2023).

[34] U. Technologies, „Skripting in Unity für erfahrene C#- und C++-Programmierer | Unity“. https://unity.com/de/how-to/programming-unity (zugegriffen 8. Juni 2023).

[35] Code Monkey, „How to use Unity NavMesh Pathfinding! (Unity Tutorial)“, 14. Juni 2021. https://www.youtube.com/watch?v=atCOd4o7tG4 (zugegriffen 14. Mai 2023).

[36] Pixelbug Studio, „ontriggerenter vs oncollisionenter in Unity2d“, 11. Mai 2020. https://www.youtube.com/watch?v=SNbYpd3tiBI (zugegriffen 9. Juni 2023).

[37] U. Technologies, „Unity - Scripting API: Rigidbody.isKinematic“. https://docs.unity3d.com/ScriptReference/Rigidbody-isKinematic.html# (zugegriffen 9. Juni 2023).

[38] U. Technologies, „Unity - Scripting API: MonoBehaviour.Awake()“. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html (zugegriffen 9. Juni 2023).

[39] Mattias Heinl Sebastian Holm Edwin Holst Albin Johansson Oliver Karmetun und Karl Öqvist, „Bachelor’s Thesis 2021 - Simulating an Ecosystem“, 2021. https://odr.chalmers.se/server/api/core/bitstreams/43871a99-bfa3-4a59-a7e9-fb8dfdae6fac/content (zugegriffen 12. Juli 2023).

[40] C. Green, „Popular Names in New Zealand“, *Nameberry*, 2. Februar 2023. https://nameberry.com/popular-names/new-zealand (zugegriffen 13. April 2023).

[41] U. Technologies, „Unity - Manual: Null Reference Exceptions“. https://docs.unity3d.com/Manual/NullReferenceException.html (zugegriffen 13. Juni 2023).

[42] Nadine, „Welche Vorteile bringen Würmer im Garten?“, 4. März 2021. https://www.wurmwelten.de/welche-vorteile-bringen-wuermer-im-garten/ (zugegriffen 14. Juni 2023).

[43] „Devolverland Expo PC Keyboard \u0026 Gamepad“. https://qplays.top/ProductDetail.aspx?iid=184862545&pr=23.88 (zugegriffen 11. Juli 2023).

[44] Tarodev, „Scrolling Background in 90 seconds - Unity Tutorial - YouTube“, 26. Januar 2021. https://www.youtube.com/watch?v=-6H-uYh80vc (zugegriffen 8. März 2023).

[45] BitSplash Interactive, „Graph and Chart - Lite Edition | GUI Tools | Unity Asset Store“. https://assetstore.unity.com/packages/tools/gui/graph-and-chart-lite-edition-148497 (zugegriffen 16. Juni 2023).

[46] BitSplash, „Graph And Chart – BitSplash IO“. http://bitsplash.io/graph-and-chart (zugegriffen 16. Juni 2023).

[47] C. Nolet, „Quick Outline | Particles/Effects | Unity Asset Store“. https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488 (zugegriffen 15. April 2023).

[48] The Game Guy, „How to make a Gradient Skybox with Stars in Unity | Shader Graph Unity“, 3. Januar 2021. https://www.youtube.com/watch?v=yw2J9NWRdow (zugegriffen 24. Mai 2023).

[49] ParadoxCodee, „How to make Fireflies in Unity #games #gaming #unity #tutorial #unitytutorials“, 31. Juli 2022. https://www.youtube.com/watch?v=1fZf3R-SINY (zugegriffen 27. Mai 2023).

[50] „‚Statistics Icon‘ Bilder – Durchsuchen 757 Archivfotos, Vektorgrafiken und Videos“, *Adobe Stock*. https://stock.adobe.com/de/search/images?k=%22statistics+icon%22 (zugegriffen 16. Juni 2023).

[51] „Opera Mask Icon Set Vector Grafik Von sakmeniko666 · Creative Fabrica“, *Creative Fabrica*. https://www.creativefabrica.com/de/product/opera-mask-icon-set-vector/ (zugegriffen 16. Juni 2023).

[52] „Black vector icon of virus cell on white background. Coronavirus symbol. Dangerous infection. Isolated vector icon. 5568013 Vector Art at Vecteezy“. https://www.vecteezy.com/vector-art/5568013-black-vector-icon-of-virus-cell-on-white-background-coronavirus-symbol-dangerous-infection-isolated-vector-icon (zugegriffen 16. Juni 2023).

[53] CG Geek, „Blender 3.1 Beginner Tutorial | Low Poly Worlds“, 16. April 2022. https://www.youtube.com/watch?v=ELiqWceCk0Q (zugegriffen 6. März 2023).

[54] Ryan King Art, „How to Make Low Poly Nature (Blender Tutorial)“, 2. Januar 2021. https://www.youtube.com/watch?v=a2vhQ6GfXIw (zugegriffen 23. April 2023).

[55] Pelopoly, „LOW POLY Tree In 5 MINUTES - Blender 2.9 Tutorial - YouTube“, 5. November 2020. https://www.youtube.com/watch?v=v116NiMPUf0&t=4s (zugegriffen 20. April 2023).

[56] Grant Abbitt, „Create Any Animal in Blender 3 - Detailed Beginner Tutorial“, 15. Oktober 2022. https://www.youtube.com/watch?v=GHv42up5xA0 (zugegriffen 17. Juni 2023).

[57] Dustyroom, „FREE Casual Game SFX Pack | Audio Sound FX | Unity Asset Store“. https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116 (zugegriffen 6. März 2023).

[58] „Free Pour Shot Drink Slam Sound Effects Download - Pixabay“. https://pixabay.com/de/sound-effects/search/pour%20shot%20drink%20slam/ (zugegriffen 17. Juni 2023).

[59] Brackeys, „Introduction to AUDIO in Unity“, 31. Mai 2017. https://www.youtube.com/watch?v=6OT43pvUyfY (zugegriffen 31. März 2023).

[60] aodhán, „Adding Sound Effects to your Unity Menu System | Unity 2022 | Quick and Code Free! - YouTube“, 14. März 2022. https://www.youtube.com/watch?v=qnS9Y8eu35I (zugegriffen 8. März 2023).