Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

Goal Oriented Action Planning mit Genetischem Algorithmus

Goal Oriented Action Planning with Genetic Algorithm

Robert Eichner

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Dipl.-Inf. Christof Rezk-Salama

Trier, 01.08.2021

Kurzfassung

Das Testen und Balancieren von Computerspielen ist ein wesentlicher Bestandteil der Entwicklung. Durch komplexere Spielwelten und längere Spielzeiten steigt jedoch der damit verbundene Aufwand. Als Folge dessen stellt die Arbeit einen Ansatz vor, wie Teile der Entscheidungsfindung für Charaktere in einem Spiel von einem Programm balanciert werden können. Diesbezüglich wurde die Implementierung eines Genetischen Algorithmus vorgestellt, der die von der GOAP Architektur erzeugten Aktionssequenzen in unterschiedlichen Spielsituationen bewertet. Dabei wurde gezeigt, dass die Verteilung von Aktionen, die durch den GOAP Planer bestimmt werden verändert werden können, damit sie sich einem Mittelwert annähern. Weiterhin wurde durch unterschiedliche Experimente mit der Implementierung gezeigt, dass diese für einfache Situationen, schon nach den ersten Generationen des Genetischen Algorithmus, optimale Lösungen findet. Des Weiteren wurde untersucht, dass Aktionen nach einer Rangliste und nach der Kombination aus einer Rangliste und dem Mittelwert balanciert werden können. Für diese Versuche wurde festgestellt, das die Bewertung nach einer Rangliste zur Bildung von Extremen in der Häufigkeitserteilung der Aktionen führt. Außerdem wurde dargestellt das die Kombination aus der Rangliste und dem Mittelwert in dem meisten Fällen ähnliche oder gleiche Verteilungen ermittelt wie die Balancierung nach nur dem Mittelwert.
Anhand dieser Erkenntnisse zeigt die Arbeit das ein Genetischer Algorithmus zur Balancierung der GOAP Architektur genutzt werden kann. In Verbindung damit wurde jedoch auch dargestellt das diese Umsetzung die Balancierung eines Designers nicht ersetzen kann, sondern wahrscheinlicher genutzt wird, um diesen zu unterstützen.

Inhaltsverzeichnis

1

Einleitung

Tester sind ein wichtiger Bestandteil in der Produktion von Computerspielen. Indem sie Softwarefehler finden und Feedback zum Spielerlebnis geben können, tragen sie dazu bei, die Qualität des Endproduktes zu steigern. Durch die wachsende Komplexität von Computerspielen wird das Testen dieser jedoch erschwert. Spielwelten werden größer und enthalten mehr Inhalt, wodurch auch eine längere Zeit beansprucht wird diese zu testen. *Open-Worlds*, Welten, in denen der Spieler von einem vorbestimmten Spielverlauf zum größten Teil losgelöst ist, bieten so fast unendliche viele Handlungsmöglichkeiten. Folglich ist auch das Testen und Balancieren dieser Spiele erschwert.
In Verbindung damit verlagern viele Spieleentwickler das Testen ihrer Spiele an die Nutzer. Durch digitale Updates können auch, nachdem das Spiel bereits herausgebracht wurde, noch Änderungen vorgenommen werden. Auf Vertriebsplattformen für Computerspiele wie zum Beispiel Steam finden sich so Titel, die schon gespielt werden können, aber sich noch in einer Alpha oder Beta-Version befinden. Insbesondere Indie Entwickler, die nicht ein großes Budget für Tester zu Verfügung haben, nutzen diese Strategie des Early-Access. So sind auf Steam 88% der Spiele die als Early Access gekennzeichnet sind von Indie Entwicklern [LBH18]. Zudem ist zu beobachten, dass 34% der Spiele den Early Access verlassen und die durchschnittliche Bewertung dieser nach der Veröffentlichung abfällt [LBH18].
Nutzer dieser Produkte können zwar Feedback zu einem Spiel geben und grobe Fehler erkennen, aber sie sind nicht vergleichbar mit ausgebildeten Spieletestern. Diese spielen ein Computerspiel zum größten Teil nicht zur Unterhaltung, sondern um Fehler zu finden. Resultierend darauf unterscheidet sich auch die Spielweise eines Spieletesters zu der eines normalen Nutzers. So kann ein Spieletester ein unvoreingenommenes Urteil geben, weil er weniger emotional an ein Spiel gebunden ist, als der Käufer.

Mit den Fortschritten im Bereich der Künstlichen Intelligenz (KI) und dem Cloudcomputing in den letzten Jahren entwickelt sich eine weitere Möglichkeit, wie Entwickler ihre Produkte objektiv testen können. So bietet die Engine Unity3D Entwicklern an ihre Spiele mit *Unity Game Simulation* automatisch zu testen [Uni21]. Dabei wird das Spiel in eine Cloud hochgeladen und dann von einer KI getestet. Weiterhin ist es möglich durch die Cloudanwendung mehrere Testläufe

gleichzeitig durchzuführen. Nach Unity3D konnten die Unternehmen, die diese Software nutzen, ihre Zeit für Tests stark verringern und Geld für personal einsparen [Uni21]. Auch Amazon bietet mit den Amazon Web Service (AWS) ein ähnliches Angebot, bei dem Entwickler eine KI erstellen können, die ihr Spiel durchspielt [Ama21]. Diese Anwendungen bieten jedoch keine Einblicke in Ihre Künstliche Intelligenz, die die Spiele testet. Als Folge dessen ist es für Entwickler schwieriger die Ergebnisse der Testläufe nachzuvollziehen und Änderungen anhand dieser vorzunehmen.
Damit verbunden soll diese Arbeit eine Alternative vorschlagen mit einem Genetischen Algorithmus ein Teil eines Computerspiels zu testen und zu balancieren. Für die Arbeit wurde dazu die Goal Oriented Action Planning (GOAP) Architektur zum Testen ausgewählt. Diese kann zur Entscheidungsfindung von Agenten in Spielwelten eingesetzt werden. GOAP wählt dazu aus einer Liste von Aktionen eine passende Sequenz dieser aus, um ein vordefiniertes Problem zu lösen. Die Arbeit soll nun zeigen, dass der Genetische Algorithmus in Kombination mit Goal Oriented Action Planning die Sequenz der erhaltenen Aktion bezüglich verschiedener Kriterien balancieren kann. Dabei soll primär das Balancing eines Computerspiels imitiert werden, dass sonst auf Feedback von Testern basiert. Abgrenzend dazu wird der Genetische Algorithmus in dieser Arbeit nicht eingesetzt, um Bugs in einem fertigen Spiel zu finden. Damit verbunden wird der Algorithmus nur in einer experimentellen Umgebung untersucht und nicht in einem realen Spiel.

2

Verwandte Arbeiten

Zur Zeit der Ausarbeitung konnten keine wissenschaftlichen Arbeiten gefunden werden, die einen Genetischen Algorithmus in Verbindung mit Goal Oriented Action Planning thematisieren. Trotz dessen gibt es mehre Arbeiten über einen Genetischen Algorithmus in Verbindung mit einem Neuronalen Netz. Diese basieren zum größten Teil auf dem NEAT-Paper. Dieses veranschaulicht, dass Neuronale Netze mit einem Genetischen Algorithmus effizient entwickelt werden können [SM02].
Darauf aufbauend gibt es viele Arbeiten, die NEAT implementieren, um eine KI zu trainieren, die ein Computerspiel durchspielt. So kann ein Neuronales Netz Flappy Bird spielen, nach bereits 20 Generationen eines Genetischen Algorithmus [CSN+19]. Weiterhin kann eine KI auch so trainiert werden, dass sie das Spiel 2048 spielen kann, obwohl dieses viele zufällige Variablen besitzt, in Form der Position der einzelnen Spielelemente [BG16]. Bei diesen Ausarbeitungen ist jedoch anzumerken, dass das Neuronale Netz so trainiert wird, dass ein Spiel optimal durchgespielt wird. Das heißt durch die Evolution des Genetischen Algorithmus wird der Lösungsansatz verbessert und somit ist der Spielablauf nahe zu fehlerfrei. Jedoch spiegelt dies nicht das typische Spielerverhalten wider. Resultierend daraus können die Ansätze kaum genutzt werden, um ein Spiel zu balancieren. Sie dienen eher als eine Bestätigung, dass zum Beispiel ein Level überhaupt durchgespielt werden kann.
Durch das Trainieren einer KI ein Spiel zu spielen, ist es möglich das ein für den Designer unerwartetes Verhalten erzeugt werden könnte[BKM+20]. So kann zum Beispiel die Künstliche Intelligenz unentdeckte Bugs im Spiel nutzen oder Spielmechaniken anders einsetzten als es vom Designer beabsichtigt ist. Diese Erkenntnisse könne auch zum Balancing beitragen, aber solche Fehler zu finden ist nicht effizient, weil eine Künstliche Intelligenz erst trainiert werden müsste und dann die Wahrscheinlichkeit auch sehr gering ist, das ein solches Verhalten erzeugt wird.
Dem gegenüber existieren auch Ansätze, die Künstliche Intelligenz nutzen, um Gegner so zu balancieren, dass sie sich an das Level des Spielers anpassen. So ist es möglich mit NEAT, Einheiten in einem RTS Spiel dementsprechend anzupassen [OYH08]. Dieser Lösungsansatz scheitert jedoch bei komplexeren Problemstellungen. Dadurch das ein Lernprozess während des Spielablaufes stattfindet, ist dieser Ansatz weiterhin limitiert, weil es so zu Verzögerungen im Verhalten der Einheiten

kommen kann, wenn der Algorithmus nicht schnell genug lernt. In Verbindung damit ist es möglich, die KI offline lernen zu lassen. Im Spielverlauf passt diese sich nur an und entwickelt sich anhand der Situation weiter, wie die Arbeit *Automatic Computer Game Balancing: A Reinforcement Learning Approach* mit Reinforcement Learning demonstriert [ARSC05].
Reinforcement Learning kann auch eingesetzt werden, um Agenten zu trainieren, damit sie möglich viel von einer Spielwelt erkunden [GBTG]. Dadurch können auch größere Spielwelten getestet werden, bezüglich ihrer Zugänglichkeit für den Spieler. Somit ist es möglich zum Beispiel Datenstrukturen für die Pfadplanung anzupassen wie ein Navigation Mesh.
Außerdem können auch Teile eines Spieles nur mit einem Genetischen Algorithmus balancieren werden, wie der GDC Talk: *Balancing Nightmares: an AI Approach to Balance Games with Overwhelming Amounts of Data* verdeutlicht [Kaz19]. So nutzt Square Enix für das Spiel *Grimms Notes Repage* einen Genetischen Algorithmus, um die Daten einer Gruppe von Spielcharakteren zu balancieren. Dabei stellt nach Ihnen die hohe Anzahl an verschiedenen Kombinationen innerhalb der Gruppe eine große Herausforderung dar. Diese wird mit dem Genetischen Algorithmus gelöst, indem dieser eine Gruppe bewertet, wie erfolgreich sie in einem Kampf ist. Dazu wird unter anderem die durchschnittliche Siegesrate und die Zeit, die in einem Kampf verbracht wird, genutzt.
Zusammenfassend ist festzustellen, dass dieses Balancing mit nur dem Genetischen Algorithmus eine hohe Ähnlichkeit zu dem angestrebten Ziel dieser Arbeit aufweist, auch wenn in diesem Ansatz nicht der GOAP Algorithmus vorkommt. Die Formen der Künstlichen Intelligenz, die ein Level eines Computerspieles durchspielen, bieten jedoch nur Ansätze für mögliche Entwürfe eines Algorithmus, der Spiele balanciert. Ihre niedrige Relevanz für die Arbeit ist damit verbunden, das sie zwar in einer Form ein Spiel testen, aber daraus keine Veränderungen vornehmen können, sodass ein Spiel fairer ist. Damit zu erwähnen sind auch die Algorithmen die Spielcharaktere so anpassen, dass sie eines bestimmten Schwierigkeitsgrades entsprechen. Sie balancieren so zwar einen Teil des Spieles, aber das Prinzip von unterschiedlichen Schwierigkeiten lässt sich nicht direkt auf die GOAP Architektur übertragen, weil diese versucht eine optimale Entscheidung zu treffen. Indirekt wäre es jedoch möglich die Ziele für ein Charakter für unterschiedliche Schwierigkeitsgrade zu unterscheiden und somit auch eine diverse Entscheidungsfindung zu realisieren.

3

Goal Oriented Action Planning

Goal Oriented Action Planning wird zur Entscheidungsfindung in Computerspielen eingesetzt. Im Gegensatz zu geläufigen Methoden, die in der KI in Spielen verwendet werden, soll GOAP es den Entwickler vereinfachen neue Zustände in ein bestehendes System hinzuzufügen [Ork04].

3.1 Definition

Goal Oriented Action Planning wurde von Jeff Orkins im Jahr 2004 für das Spiel F.E.A.R. entworfen. Es basiert auf den Standford Research Institut Problem Solver (STRIPS) und dient dazu Entscheidungen für einen Spielcharackter zu treffen. Wie im STRIPS Modell versucht auch GOAP mithilfe von einer Liste an Aktionen von einem gegebenen Anfangszustand zu einem angestrebten Endzustand zu gelangen. Umgesetzt wird dies, indem eine Sequenz an Aktionen gesucht wird. Diese Aktionen besitzen jeweils Bedingungen und Effekte. Im Zusammenhang mit einem Computerspiel sind dies Veränderungen von Zustände in der Spielwelt, wie die Position von anderen Charakteren oder die Verfügbarkeit von Ressourcen. GOAP versucht dann mithilfe der Informationen, die ein Charakter über die Spielwelt hat, den Endzustand zu erreichen, indem die Aktionen den Anfangszustand so weit verändern bis er den Endzustand entspricht. Die dafür benötigten Aktionen werden von einem Planer ausgewählt. Dieser bildet daraus die verschiedene Aktionensequenzen, die durch ihre Bedingungen und Effekten aneinander gekettet sind. Eine Aktion kann mehrere Bedingungen und Effekte besitzen. Im Gegensatz zum STRIPS Modell besitzen die Aktionen bei GOAP einen Wert, der ihre Kosten repräsentiert. Anhand dieser kann der Planer verschiedene Aktionssequenzen miteinander vergleichen und eine optimale Lösung auswählen. Weiterhin ermöglicht dieser Ansatz, dass Algorithmen zur Pfadplanung eingesetzt werden können. So können die einzelnen Aktionen als Transitionen dargestellt werden, die die unterschiedlichen Zustände der Spielwelt, die Knoten verbinden. Auf dem gebildeten Graphen ist es dann möglich den kürzesten Pfad zwischen Anfangszustand und Endzustand zu finden. Abbildung 3.1 zeigt dies.

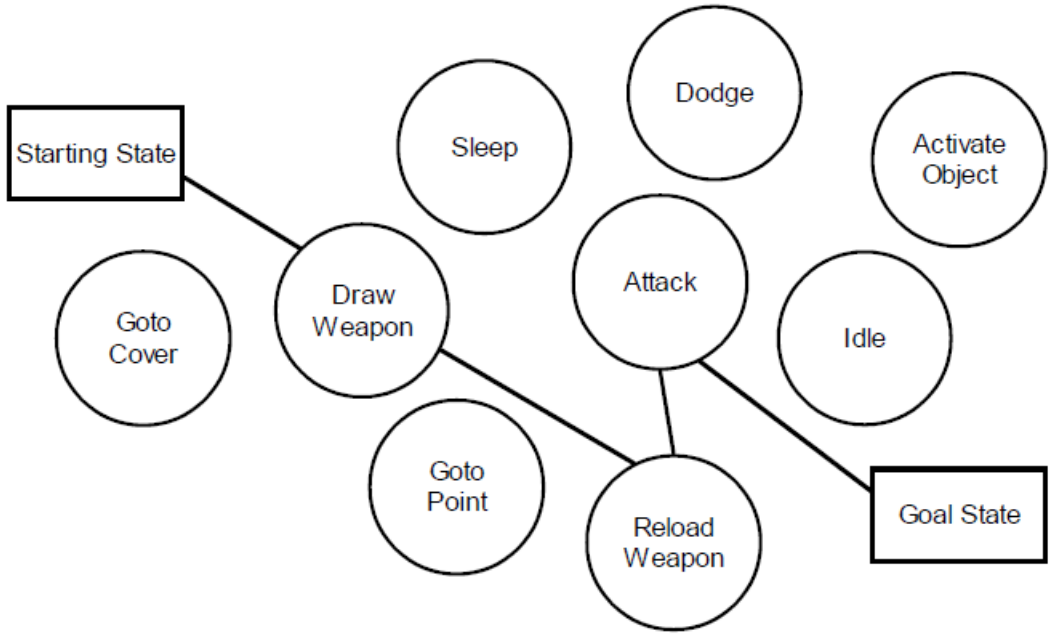

Abb. 3.1. Veranschaulichung eines Graphen mit ausgewählten Plan aus [Ork08]

Des Weiteren kann eine GOAP Architektur auch mehrere Endzustände in Form von unterschiedlichen Zielen für einen Charakter definieren. Diese befinden sich in einer Liste und sind ihre Priorität nach sortiert. Der Planer versucht dann für jede dieser Ziele eine passende Aktionssequenz zu finden. Wird eine Lösung gefunden, werden die Aktionen ausgeführt und der Zustand der Welt verändert sich dem entsprechend. Wenn die Aktionen abgeschlossen sind oder eine Lösung gefunden wurde, versucht der Planer für das nächste Ziel in der Liste wieder ein Ergebnis zu finden. Außerdem ist es möglich, dass der Planer den Planungsprozess abbricht, wenn die Spielwelt sich so verändert, dass ein neuer Plan erstellt werden muss, weil zum Beispiel der Anfangszustand, in dem sich der Charakter befindet, nicht mehr gültig ist [Ork06]
GOAP kann als eine Alternative zu einem Zustandsautomaten für eine KI in Spielen eingesetzt werden, weil es durch seine Architektur mehrere Vorteile bietet. So können Aktionen in das Spiel schneller eingefügt werden, die unterschiedliches Verhalten der Charaktere erzeugen [Ork04]. Weiterhin ermöglicht der Planer, dass auch Entscheidungen getroffen werden können in Randbedingungen, die der Programmierer nicht zuvor abgedeckt hat. Trotz dessen ist auch zu beachten, dass GOAP eine höhere Laufzeit ausweist im Vergleich zu einem Zustandsautomaten oder einem Behavior Tree, die auch für Künstliche Intelligenz in Computerspielen eingesetzt werden.
Zusammenfassend wählt der GOAP Planer aus einer Liste von Aktionen eine passende Sequenz aus, die die Spielwelt so verändert, dass sie einen angestrebten Zustand entspricht. Damit dies realisiert wird, wählt der GOAP Planer aus einem Graphen, aus diesen Aktionen, den kürzesten Weg von dem Anfangszustand, indem sich der Charakter befindet, zu dem Zielzustand.

3.2 Implementierung

Für die Implementierung des GOAP Algorithmus wird ein System benötigt, dass den Zustand der Welt verfolgt und verändern kann. Dafür existiert die Klasse *WorldState*. Diese hält ein Dictonary von *WorldVariables* und ihren Werten in der Spielwelt. Die *WorldVariables* sind Wrapper für C# Datentypen und erben von der Oberklasse ScriptableObject, die es ermöglicht diese Variablen, als Assets zu erstellen. Dadurch können diese einfacherer wiederverwendet werden und von einem Designer ohne Code zu schreiben erstellt werden.
Des Weiteren existieren die Klassen *Action* und *Goal*, die die Logik der Aktion und des Ziels der GOAP Architektur repräsentieren. So besitzt eine *Action* Bedingungen und Effekte, die von dem Planer genutzt werden, um die Aktionen aneinander zu ketten. Die Klasse *Goal* enthält den angestrebten Zustand der Spielwelt. Für die Implementierung wurden einfache Aktionen und Ziele erstellt, die mit Boolean Werten arbeiten. Die Implementierung des Weltzustandes ermöglicht es zwar Objekte zu benutzen und somit nicht nur Boolean, dies wurde aber nicht für die späteren Versuche genutzt. Ein Grund dafür ist, dass die Implementierung keine Umsetzung der GOAP Aktionen vorsieht, weil diese für die Tests redundant ist. Somit reichen die einfachen Boolean Werte aus, weil keine komplexere Logik für die Weltvariablen benötigt wird, die mit zum Beispiel Objekten sonst umgesetzt werden würde.
In der Klasse *Agent* werden die verfügbaren Aktionen und die angestrebten Ziele für den GOAP Planer hinterlegt. Auch hier wird keine Logik benötigt, um die Aktionen umzusetzen. Wenn die vollständige GOAP Architektur implementiert werden würde, wäre in dieser Klasse auch die Logik für einen Zustandsautomaten, der die Handlung des Agenten in der Spielwelt, anhand der vom Planer bestimmten Aktionen, lenkt.
Für den Planer wurde die Implementierung von [Owe15] referenziert. Diese unterscheidet sich von der originalen Jeff Orkins Implementierung, indem sie keinen Graphen mit A Stern Algorithmus nutzt, sondern eine Baumstruktur erstellt wird. Dabei ist die Wurzel des Baumes der Zielzustand des GOAP Planers und die davon ausgehenden Knoten sind jeweils die ausführbaren Aktionen. Diese werden auf den Zielzustand invertiert angewendet, mit dem Ziel, das die Blätter dem Ausgangszustand entsprechen und somit eine gültige Aktionssequenz gefunden wird. In der Implementierung wird während der Erstellung des Baumes auch ein Verweis auf das günstigste Blatt, also die Gesamtkosten der genutzten Aktionen, mit geführt. Dadurch entfällt der Aufwand für eine Suche nach der optimalen Lösung.

4

Genetischer Algorithmus

Genetische Algorithmen werden unter anderem eingesetzt, wenn eine große Menge an Möglichkeiten durchsucht werden muss, um ein Problem zu lösen, weil sie eine parallele Suche ermöglichen[Whi95]. So werden sie in Verbindung mit maschinellen Lernen implementiert, um Neuronale Netze weiterzuentwickeln oder Vorhersagen anhand von Daten zu treffen[Whi95].

4.1 Definition

Ein Genetischer Algorithmus findet zu einem Problem eine Lösung, indem er die Evolutionstheorie von Charles Darwin imitiert [Mit96]. Das bedeutet, dass der Algorithmus versucht ein Problem zu lösen, indem er aus einer Menge von Lösungsstrategien, die optimale Lösung wählt und diese durch Vererbung und Mutation verbessert. Die Menge an Lösungsstrategien wird Population genannt. Jede Lösung in dieser Menge hat nach dem Vorbild aus der Biologie Gene mit verschiedenen Parametern. Die Parameter geben den Ansatz wie ein Problem mit dem Genetischen Algorithmus gelöst werden soll. So könnten sie zum Beispiel Koordinaten enthalten, wenn der Algorithmus eingesetzt wird, um einen Pfad zu einem vorbestimmten Ziel zu finden.
Jedes Individuum aus der Population kann anhand seiner Gene und einer Fitnessfunktion bewertet werden, wie gut es eine Aufgabe löst. Mithilfe des damit entstandenen Wertes können aus der Population die besten Individuen bestimmt werden, die ihre Gene an eine neue Generation weitergeben. Die Wahl der fittesten Individuen, auch Eltern genannt, heißt Selektion.
In der nächsten Phase, der Vererbung, werden die ausgewählten Gene miteinander gekreuzt. Das bedeutet innerhalb der Parameter wird eine Schnittstelle ausgewählt, an der ein Teil des Gens mit den Parametern eines anderen Gens ausgetauscht wird. Dadurch entsteht eine neue Population, bei der die Individuen Kinder genannt werden. Weiterhin kann bei den Kindern auch ein Teil der Parameter mutieren. Das bedeutet, dass mit einer zuvor festgelegten Chance, zufällige Parameter in den Genen einen neuen Wert zugewiesen bekommen.
Mit der neu entstandenen Population wird der Prozess der Selektion, Kreuzung und Mutation so lange fortgeführt bis die Lösungen, die Gene der fittesten Individuen, kaum noch Unterschiede zueinander aufweisen. In diesem Fall ist der

Algorithmus konvergiert und es ist unwahrscheinlich, dass eine bessere Lösung gefunden wird.
Dabei ist jedoch zu beachten, dass der Algorithmus nicht Informationen über mehrere Generationen weitergeben kann. Als Folge dessen kann dieser an einem lokalen Optimum konvergieren. Dieses sind Lösungen, bei denen Individuen mit einem schlechteren Fitnesswert zu der angestrebten Lösung führen könnten, aber durch die fittesten Individuen verdrängt werden.
Zusammenfassend ist ein Genetischer Algorithmus ein Suchverfahren, das sich an der Evolutionstheorie orientiert. Durch unterschiedliche Gene in einer Population können Lösungsstrategien parallel evaluiert werden. Dabei werden diese durch Vererbung und Mutation verbessert, bis der Algorithmus ein Optimum gefunden hat.

4.2 Implementierung

Für die Implementierung des Genetischen Algorithmus wurden die Klassen *Genes* und *Populationmanager* geschrieben.
Die Klasse *Genes* enthält eine Liste mit Werten, die die Kosten der Aktionen für den GOAP Algorithmus repräsentieren. Die Anzahl der Werte wird durch die Anzahl der Aktion bestimmt, die dem GOAP Planer zur Verfügung stehen. Bei der Initialisierung der Individuen werden diese Werte zufällig bestimmt. Um Rundungsfehler in späteren Berechnungen des GOAP Planers zu vermeiden, wurden nur ganzzahlige Datentypen verwendet.
Die Klasse *PopulationManger* initialisiert die Population und ist für den Ablauf des Genetischen Algorithmus zuständig. Das bedeutet, dass die Klasse die Werte jedes Individuums einer Generation auf die GOAP Aktionen überträgt, mit dem GOAP Planer einen Plan erstellt und die Gene damit bewertet. Anschließend wird wie zuvor beschrieben, die besten zwei Individuen werden bestimmt, an einem zufälligen Punkt werden ihre Gene gekreuzt und Teile der Parameter werden zufällig mutiert. Die Zufallswerte für die Mutation und Initialisierung der Aktionskosten werden durch einen Maximalwert eingeschränkt, damit der Algorithmus schneller konvergiert.
Die Fitnessfunktion nutzt ein Histogramm, das für jede Generation neu angelegt wird. In diesem werden die Aktionen gespeichert, und hinterlegt wie häufig diese in den Aktionssequenzen vorkommen, die von dem GOAP Planer ermittelt werden. Dies ermöglicht es Aussagen über die Häufigkeitsverteilung von Aktionen zu treffen.
Der Genetische Algorithmus wird mithilfe einer Coroutine gestartet. Diese ermöglicht es, dass der Main- Thread der Anwendung durch den Algorithmus nicht blockiert wird. Dadurch kann der Nutzer während der Laufzeit des Genetischen Algorithmus mit dem User-Interface interagieren.

5

Versuchsaufbau

Das Ziel des Programms ist zu zeigen, dass eine GOAP Architektur mit einem Genetischen Algorithmus balanciert werden kann. Um dies zu realisieren sind hauptsächlich zwei Ansätze möglich. Zu einem können die Ergebnisse, die durch die Umsetzung der ausgewählten Aktionen des GOAP Planers entstehen, betrachtet werden. So könnten Charaktere beurteilt werden, wie gut sie eine Aufgabe lösen, mit den Aktionen die ausgewählt wurden. Dadurch entstehen jedoch Aktionen die anderen Aktionen gegenüber bevorzugt verwendet werden, wenn diese häufig in den Aktionssequenzen, die von dem GOAP Planer bestimmt werden, vorkommen. Somit entsteht eine dominante Lösungsstrategie, die der Charakter voraussichtlich in einer Mehrheit der auftretenden Situationen anwendet, um ein Problem zu lösen. Dies widerspricht jedoch dem Prinzip der GOAP Architektur. Diese versucht für jede Situation eine optimale Aktionssequenz zu finden. Indem jedoch bestimmte Aktionsketten bevorzugt verwendet werden, wird der GOAP Planer zum Teil überflüssig. So könnten diese dominanten Aktionssequenzen aus der GOAP Logik entfernt werden. Sie würden dann in eine andere Form der Spiellogik übertragen werden, zum Beispiel als ein Zustand in einem Zustandsautomaten, der den Charakter lenkt. Der GOAP Planer würde dann nur noch Aktionen finden in Situationen, die von der dominanten Aktionssequenz nicht gelöst werden können. Wenn man diese Form des Balancing weiterführt würden so immer mehr Aktionssequenzen ausgegliedert werden, wodurch die Nutzung der GOAP Architektur überflüssig wird.
Die Entscheidung diese bevorzugten Aktionssequenzen in der GOAP Architektur zu lassen ist auch ein Ansatz, wirft jedoch weitere Kritikpunkte auf. Ein großer Kritikpunkt der GOAP Architektur ist die polynomielle Laufzeit, die unter anderem durch die Planungsphase entsteht [Mil19]. Diesbezüglich müsste also abgewogen werden, wie effizient es ist, wenn in einer bestimmten Menge an Situationen, der Planer, die gleiche Aktionssequenz bestimmt. Diese könnte wie zuvor erwähnt dem gegenüber ausgelagert werden ,in zum Beispiel einen Zustandsautomaten, der eine lineare Laufzeit aufweist [Mil19].
Weiterhin können dominante Lösungsstrategien dem Spielerlebnis des Spielers schaden. So würde ein Charakter langweilig wirken, wenn er in vielen unterschiedlichen Situationen oft die gleiche Entscheidung trifft. So wäre ein Charakter der in Kampfsituationen hauptsächlich den Kampf sucht oder meistens wegläuft für

einen Spieler voraussichtlich uninteressant.
Des Weiteren ist es denkbar, dass Situationen entstehen, die von der dominanten Aktionssequenz optimal gelöst werden können, aber der Spieler die daraus resultierende Handlung nicht nachvollziehen kann. So lässt sich folgendes Situation konstruieren. Der Genetische Algorithmus bewertet wie gut ein Charakter Holz sammelt. Dafür besitzt dieser mehrere Aktionen, bei dem eine Holz zu zaubern ist und eine andere bei der Holz gesammelt wird. Für Fitnessfunktionen die eine maximale Erfolgschance oder eine minimal benötigte Zeit anstreben, könnte Holz zaubern eine dominante Aktion oder Aktionssequenz bilden, wenn der Charakter ohne Verzögerung mit einer hohen Wahrscheinlichkeit ein Stück Holz erhält. Folglich würden für die meisten Situationen, in denen Holz gesammelt wird, diese Aktionssequenz benutzt. Darunter könnten sich jedoch auch Situationen befinden, indem es für einen Spieler nicht logisch erscheint, wenn ein Charakter Holz herbeizaubert, zum Beispiel in einem Wald oder einem Baumarkt.
Unter Berücksichtigung dieser Aspekte wurde für den Versuchsaufbau der Genetische Algorithmus nicht genutzt, um die Ergebnisse des GOAP Algorithmus hinsichtlich verschiedener Erfolgskriterien zu optimieren. Stattdessen wird ein Ansatz genutzt, indem der Genetische Algorithmus die Kosten der Aktionen so verändert, dass keine dominanten Aktionssequenzen entstehen. Das bedeutet, dass der GOAP Planer für verschiedene Zustände in der Welt Pläne erstellt. Die dabei entstandenen Aktionssequenzen werden genutzt, um für jede Aktion zu bestimmen, wie häufig sie in den Plänen verwendet wurde. Damit keine dominante Aktion einsteht, ist das Ziel des Genetischen Algorithmus, das die Häufigkeit dieser Aktionen sich einem Mittelwert annähern. Dieser Mittelwert ist der Quotient aus der Anzahl der verfügbaren Aktionen und der Menge der verschiedenen Zustände der Spielwelt.
Der GOAP Planer erstellt im Normalfall immer den gleichen Plan für einen bestimmten Zustand der Spielwelt. Es ist zwar möglich, dass der Planer mehrere valide Lösungen findet, bei denen die Aktionssequenzen die gleiche Gesamtzahl an Aktionskosten aufweisen, aber in diesem Fall würde festgelegt sein welche Aktionssequenz er auswählt. Zum Beispiel immer die zuerst oder die zuletzt gefunden Kette würde ausgewählt werden. Die zufällige Bestimmung einer dieser Aktionssequenzen wird in der von Jeff Orkins vorgestellten Implementierung nicht thematisiert [Ork05]. Als Folge dessen kann jedem Zustand der Spielwelt genau eine Aktionssequenz zugewiesen werden, die der Planer bestimmt. Somit entspricht die Summe der verschiedenen Zustände der Spielwelt, die Summe der vom GOAP Planer bestimmten Aktionssequenzen, die genutzt werden kann, um einen Mittelwert zu bestimmen.
Damit jeder Zustand der Spielwelt getestet werden kann, würde eine Liste an Zustände erstellt werden, die jede mögliche Kombination der einzelnen Variablen enthält, die in der Spielwelt von der GOAP Architektur beobachtet werden. Um diese Liste zu bestimmen wäre ein Aufwand von $\mathcal{O}(n!)$ nötig, wobei n die Anzahl der zu beobachtenden Variablen in der Spielwelt ist. Da Spielwelten in den meisten Fällen komplexe Systeme sind, in denen eine Vielzahl von Variablen auftauchen, wie zum Beispiel Koordinaten von Objekten oder Verfügbarkeiten von Ressourcen, ist n voraussichtlich ein größerer Wert. In Verbindung mit dem fakto-

riellen Aufwand würde eine Berechnung dieser Kombinationen eine zu große Laufzeit in Anspruch nehmen. Folglich ist es nicht möglich für alle unterschiedlichen Zustände der Spielwelt einen Plan und somit eine Aktionssequenz zu bestimmen. Daher können in der Implementierung diese Zustände manuell eingefügt werden. So kann zum Beispiel ein Designer, die Zustände der Spielwelt ergänzen, die für ihn von Interesse sind, weil sie zum Beispiel häufiger vorkommen oder von besondere Wichtigkeit für den Spielverlauf sind.
Weiterhin enthält der Versuchsaufbau die Möglichkeit das Balancing nach einem Mittelwert mit einer Rangliste zu ergänzen. Das bedeutet, dass die Verteilung der Aktionen beeinflusst werden kann. Die dabei entstehende Rangliste zielt nicht darauf ab eine dominante Aktionssequenz durch eine Reihenfolge zu erstellen. So soll die Häufigkeitsverteilung der einzelnen Aktionen immer noch nach einem Mittelwert ausgelegt sein, aber das Auftreten der einzelnen Aktionen soll, nach der vorgebenden Rangliste bestimmt werden. Diese Funktion soll es ermöglichen, das ein Designer mehr Einfluss auf die getroffenen Entscheidung der GOAP Architektur hat. Dadurch kann er zum Beispiel für Charaktere unterschiedliche Verhaltensmuster erstellen, indem sie bestimmte Aktionen anderen Aktionen bevorzugen. In der normalen GOAP Implementation würde dies durch eine Variation in den Kosten der einzelnen Aktionen erzeugt werden. Da aber der Versuchsaufbau diese Kosten verändert, damit keine dominanten Aktionen entstehen, entfällt dies. Folglich wird durch die Rangliste versucht, diesen Sachverhalt wieder zu ermöglichen.
Zusammenfassend soll gezeigt werden, dass ein Genetischer Algorithmus die verwendeten Aktionen einer GOAP Architektur nach einem Mittelwert und nach einer Rangliste balancieren kann. Um dies zu überprüfen, werden die folgenden Kriterien untersucht.
Am Anfang wird geprüft, dass der genetische Algorithmus funktioniert. Dafür wird gezeigt, dass der Algorithmus konvergiert und in gestellten Situation die erwartenden Ergebnisse erzeugt. Damit die Resultate möglichst unbefangen sind, werden verschiedene Kombinationen von Aktionen getestet. So werden diese in ihrer Anzahl und Komplexität variiert. Letzteres bezieht sich darauf, das verschiedene Aktionssequenzen sich zum Beispiel Aktionen untereinander teilen können. Oder Aktionen für das Balancing ausgewählt werden, die sich auf verschiedenen Ebenen in den Aktionssequenzen befinden. So würden nicht nur die Aktionen betrachtet werden, die ein Ziel direkt erfüllen, sondern auch solche Aktionen, die zur Bildung der Aktionskette beitragen, indem sie die Bedingungen erfüllen für die finale Aktion.
Damit das Balancing nach dem Mittelwert und einer Rangliste bewertet werden kann, werden sowohl Experimente durchgeführt, in dem diese separat betrachtet werden, als auch in Kombination.
Letztlich werden auch Tests ausgeführt, für Ausnahmefälle.

6

Testergebnisse

In den folgenden Abschnitten werden die Testergebnisse, die mit dem zuvor vorgestellten Versuchsaufbau bestimmt wurden ausgewertet. Damit die daraus resultierenden Ergebnisse untereinander vergleichbar sind, wurden für alle Versuche die gleichen Rahmenbedingungen festgelegt. Pro Versuch finden fünf Durchgänge des Genetischen Algorithmus statt. Jeder Durchgang durchläuft 100 Generationen mit jeweils 50 Individuen. Die Mutationschance wurde für alle Durchgänge auf 0.125 gesetzt.

6.1 Konvergenz

Zu Zeigen: Der Algorithmus konvergiert mit zwei voneinander unabhängigen Aktionen.
Aufbau: Es wurden zwei Aktionen konstruiert, aus denen zwei Aktionssequenzen gebildet werden können. Die Aktionssequenzen erfüllen das gleiche Ziel, haben aber jeweils unterschiedliche Voraussetzungen. Für diese wurden vier Zustände konstruiert, die durch die Aktionssequenz gelöst werden können und die Aktionen gleich häufig verwenden sollen.
Annahme: Beide Aktionen werden jeweils gleich oft genutzt. Da diese auf vier unterschiedliche Zustände angewendet werden, kommen beide Aktionen jeweils zweimal vor.

Ergebnisse des Versuches			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	4	2:2	1
1	4	2:2	1
1	4	2:2	1
1	4	2:2	1
1	4	2:2	1

Auswertung: Die Ergebnisse zeigen, dass der Algorithmus konvergiert und den vorhergesagten Mittelwert anstrebt. Jedoch ist auch festzustellen, dass diese Resultate schon in der ersten Generation auftreten. Dies ist vermutlich dadurch be-

gründet, dass für diesen Versuch einfach eine Lösung durch den Genetischen Algorithmus gefunden werden kann. So müsste in der ersten Generation nur ein Individuum auftauchen, das für die Aktionen ohne Bedingung, geringere Kosten festlegt, im Vergleich zu der anderen Aktion. Dadurch würde die allgemeine Aktion vom Planer nur ausgewählt werden, wenn die andere Aktion nicht erfüllt werden kann. Dieser Vorgang erfüllt das Kriterium nach einem Mittelwert, weil die vorgebenden Weltzustände so konstruiert wurden, dass die Aktionen jeweils zweimal ausgewählt werden durch den Planer.
Anhand dieser Erkenntnisse werden für die kommenden Versuche sowohl die Anzahl der Aktionen, als auch die Anzahl der Zustände erhöht.

6.2 Drei Aktionen gleicher Effekt

Diese Versuche sollen zeigen wie sich der Algorithmus verhält, wenn mehrere zufällige Weltzustände zum Testen erstellt werden. Dabei werden zunächst drei Aktionssequenzen untersucht.
Zu Zeigen: Der Algorithmus nähert sich einem Mittelwert an, es bildet sich eine Rangliste und beides ist kombiniert möglich.
Aufbau: Es werden drei Aktionen mit 15 verschiedenen Weltzuständen getestet. Dabei werden drei unterschiedliche Versuche ausgeführt, für nur den Mittelwert, nur nach Rangliste und beides in Kombination.
Annahme: Durch die erhöhte Anzahl an Zuständen und Aktionen konvergiert der Algorithmus später. Ein Mittelwert kann nicht genau getroffen werden, weil die Zustände vermutlich nicht ausgeglichen sind, sodass die Aktionssequenzen gleichmäßig auf diese verteilt werden können. Ein Sortieren nur nach Rangliste wird sich an Extremen orientieren. Im Gegensatz dazu wird erwartet, dass die Kombination aus beiden Werten die Häufigkeitsverteilung der Aktionen so ändert, das diese sich nach einem Mittelwert ausrichtetet, aber diese Werte nach der Rangliste sortiert sind.

Ergebnisse des Versuches mit Mittelwert			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
5	3	5:5:5	19
3	3	5:5:5	37
20	3	5:5:5	20
5	3	5:5:5	5
2	3	5:5:5	24

Ergebnisse des Versuches mit Rangliste			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	3	0:0:15	1
1	3	2:6:7	nicht feststellbar
1	3	0:0:15	1
1	3	0:6:9	nicht feststellbar
1	3	4:5:5	nicht feststellbar

Ergebnisse des Versuches Zusammen			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
8	6	5:5:5	8
5	6	5:5:5	42
2	6	5:5:5	2
4	6	5:5:5	4
2	6	5:5:5	15

Auswertung: Wenn der Genetische Algorithmus nur nach einem Mittelwert entscheidet, wurden die Aktionen auch entsprechen nach diesem Mittelwert verteilt. Abweichungen sind nicht wie vermutet aufgetreten. Ein Grund dafür könnte sein, dass die Aktionen alle den gleichen Zielzustand erzeugen konnten und so der Genetische Algorithmus die Häufigkeiten in Verbindung mit den getesteten Weltzuständen gleichmäßig aufteilen kann.
Die Balancierung nach nur einer Rangliste hat gezeigt, dass dieser Wert allein nicht ausreicht, damit der Algorithmus konvergiert. So wurden mehrere Wertepaare ermittelt, die die Aktionen nach einer Rangliste sortieren. Jedoch können diese nur mit dem Kriterium einer Staffelung nicht bewertet werden. Als Folge dessen sind für alle gültigen Kombinationen die Fitnesswerte gleich. Trotz dessen konnte auch gezeigt werden, das eine extreme Verteilung durchaus häufiger von dem Genetischen Algorithmus gefunden und wiederholt wird. Ein Grund dafür könnte es sein, das diese Anordnung am einfachsten zu finden ist.
Wenn beide Parameter kombiniert werden, zeigen die Ergebnisse die gleichen Resultate auf wie die Balancierung nach einem Mittelwert. Der Grund dafür ist, dass eine Verteilung von 5:5:5 besser bewertet wird als eine von 6:5:4. Letzteres wird schlechtere bewertet, weil diese Werte weiter vom Mittelwert entfernt sind. Weiterhin erzeugen beide Verhältnisse den gleichen Wert für den Ranglistenparameter, weil die Verteilung 5:5:5 durch die Implementierung in der darunter liegenden Datenstruktur und den verwendeten Sortieralgorithmus nach der gegebenen Rangliste angeordnet sein kann.

6.3 Drei Aktionen mit unterschiedlichen Effekten

In diesem Versuch werden Aktionen ausgewählt, die nicht den gleichen Zustand verfolgen. Damit die Ergebnisse miteinander verglichen werden, nutzt dieser Ver-

such die gleichen Rahmenbedingungen wie das Experiment zuvor mit den drei Aktionen zu den gleichen Weltzuständen.
Zu Zeigen: Der Algorithmus nähert sich einem Mittelwert an, es bildet sich eine Rangliste und beides ist kombiniert möglich.
Aufbau: Es werden drei Aktionen mit 15 verschiedenen Weltzuständen getestet. Dabei werden drei unterschiedliche Versuche ausgeführt, für nur den Mittelwert, nur nach Rangliste und beides in Kombination.
Annahme: Die Erkenntnisse aus dem Versuch zuvor sollen ähnlich wieder erzeugt werden. Da diesmal die Aktionen unterschiedliche Effekte besitzen, ist davon auszugehen, das sich an einem Mittelwert nur angenähert werden kann und dieser nicht wie im Versuch zuvor genau getroffen wird. In Verbindung soll so auch gezeigt werden, dass die Kombination aus Mittelwert und Rangliste ein anderes Ergebnis erzeugt, als nur der Versuch mit den Mittelwerten.

Ergebnisse des Versuches mit Mittelwert			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
80	9	3:5:7	nicht feststellbar
19	9	3:5:7	nicht feststellbar
24	9	3:5:7	nicht feststellbar
75	9	3:5:7	nicht feststellbar
37	9	3:5:7	nicht feststellbar

Ergebnisse des Versuches mit Rangliste			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	3	0:0:6	nicht feststellbar
1	3	0:0:0	nicht feststellbar
1	3	0:0:8	nicht feststellbar
1	3	0:0:0	nicht feststellbar
1	3	0:0:6	nicht feststellbar

Ergebnisse des Versuches Zusammen			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
95	12	3:5:7	nicht feststellbar
57	12	3:5:7	nicht feststellbar
14	17	3:4:8	nicht feststellbar
36	12	3:5:7	nicht feststellbar
67	17	3:4:8	nicht feststellbar

Auswertung: Die Ergebnisse für den Mittelwert zeigen, dass die ausgewählten Aktionen nicht gleichmäßig auf die getesteten Weltzustände aufgeteilt werden können. Als Folge dessen nähern sie sich einem Mittelwert nur an.
Wenn die Aktionen nur nach einer Rangliste bewertet werden, neigt die Verteilung

dazu wieder Randwerte zu priorisieren. Weiterhin ist auch zu erkennen, dass in diesem Fall Aktionen die balanciert werden auch nicht ausgewählt werden können durch den Planer. Als Folge dessen entspricht das Auftreten der ausgewählten Aktionen nicht mehr der Anzahl der zu testenden Weltzuständen.
Die Kombination aus beiden Parametern zeigt erneut, dass diese den Ergebnissen der Verteilung nach dem Mittelwert entsprechen. Diesmal treten zwar auch davon abgewandelte Resultate auf, die jedoch schlechtere Fitnesswerte aufweisen. Folglich repräsentieren diese ein lokales Optimum, weil die Verteilung die den Mittelwerten entspricht, die globale Lösung darstellt, durch ihre geringeren Kosten.
In diesem Versuch ist es auch nicht möglich eine Konvergenz des Algorithmus in den durchgeführten 100 Generation zu erkennen. So wiederholen sich die Gene mit den besten Werten kaum oder gar nicht.

6.4 Komplette Aktionssequenzen

Bisher wurden einzelne Aktionen von Sequenzen balanciert. In diesem Abschnitt soll herausgearbeitet werden, welche Ergebnisse die Implementierung erzeugt, wenn alle verfügbaren Aktionen balanciert werden sollen.
Zu Zeigen: Der Algorithmus balanciert die Aktionen in ihrer Häufigkeit nach einem Mittelwert. Wie der Versuch zuvor schon zeigen sollte, wird wieder versucht, dass die Kombination aus einer Rangliste und dem Mittelwert ein anderes Ergebnis erzeugt. Auf die Werte für nur eine Rangliste wird verzichtet, weil es keine Indikatoren gibt, dass sich das Verhalten in diesem Versuch im Vergleich zu den anderen ändern wird.
Aufbau: Alle sieben verfügbaren Aktionen werden für das Balancing genutzt. Damit die Ergebnisse vergleichbar sind, handelt es sich um die gleichen, wie in den Experimenten mit den drei Aktionen. Diese bilden drei verschiedene Aktionssequenzen. Auch die verschiedenen Zustände der Spielwelt wurden beibehalten.
Annahme: Durch die unterschiedlichen Aktionen, die zum Teil auch auf sich aufbauen, wird sich ein Teil einem Mittelwert annähern, während der Rest höhere Werte aufweisen wird. Letzteres könnten so zum Beispiel Aktionen sein, die in vielen Aktionssequenzen gebraucht werden und somit ihre Häufigkeit nicht an den Mittelwert angepasst werden kann.
Außerdem werden sich die Werte für die Kombination aus Rangliste und dem Mittelwert von nur dem Mittelwert unterscheiden, weil durch die hohe Anzahl an Aktionen der Wert nach der Rangliste mehr in Gewicht fällt, als der Mittelwert.

Ergebnisse des Versuches Mittelwert			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	33.88	5:2:1:5:0:0:5	5
5	33.88	5:2:1:5:0:0:5	22
2	33.88	5:2:1:5:0:0:5	14
19	33.88	5:2:1:5:0:0:5	19
6	33.88	5:2:1:5:0:0:5	34

Ergebnisse des Versuches Zusammen			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	85.81	4:1:0:5:0:0:6	1
1	85.81	4:1:0:5:0:0:6	7
2	85.81	4:1:0:5:0:0:6	2
3	85.81	4:1:0:5:0:0:6	9
2	85.81	4:1:0:5:0:0:6	2

Auswertung: Die Balancierung nach einem Mittelwert erzeugt wie angenommen für einen Teil der Aktionen eine Annäherung an diesem. Jedoch anders als vermutet, kommen die restlichen Aktionen kaum oder gar nicht vor. Ein Grund dafür könnte sein, das die Verwendung dieser die anderen Aktionen, die sich nach dem Mittelwert orientieren, beeinflussen könnten. Folglich würden diese Aktionen häufiger oder weniger auftreten, wodurch ein höherer Abstand zum Mittelwert entsteht. Wenn eine Aktion immer nur eine Aktion beeinflussen würde, wären diese Interaktionen wahrscheinlich kaum ersichtlich. So ist vorstellbar das, wenn eine Aktion häufiger vorkommt, die andere dementsprechend reduziert wird. Dieses Verhältnis ist jedoch nicht nachweislich erkennbar.
Die Kombination aus Mittelwert und Rangliste erzeugt in diesem Versuch eindeutig unterschiedliche Werte im Vergleich zu nur dem Mittelwert. So ist erkennbar, dass die Häufigkeit von Aktionen angepasst wurden, damit diese der Reihenfolge in der Rangliste entsprechen.

6.5 Verschachtelte Aktionssequenzen

In diesem Abschnitt werden sieben neue Aktionen konstruiert, die Aktionssequenzen bilden die Aktionen untereinander teilen. Dadurch soll geprüft werden, wie der Aufbau sich verhält, wenn Aktionssequenzen verschachtelt sind.
Zu Zeigen: Untereinander verschachtelte Aktionen können nach einem Mittelwert, einer Rangliste und einer Kombination aus diesen nach ihrer Häufigkeit balanciert werden.
Aufbau: Anhand der sieben Aktionen können wieder drei Aktionssequenzen gebildet werden. Jedoch teilen diese sich Aktionen untereinander. Die zu testenden Zustände der Spielwelt werden von den zuvor durchgeführten Versuchen übernommen.
Annahme: Die Ausrichtung nach einem Mittelwert ist erschwert, weil Aktionen schwieriger entfallen können wie im Versuch zuvor (Komplette Aktionssequenzen). Für die Sortierung nach einer Rangliste wird ein ähnliches Verhalten erwartet, wie im Versuch Drei Aktionen mit unterschiedlichen Effekten.

Ergebnisse des Versuches Mittelwert			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	51.6	7:3:0:4:5:0:2	1
1	51.6	7:3:0:4:5:0:2	1
1	51.6	7:3:0:4:5:0:2	1
1	51.6	7:3:0:4:5:0:2	1
1	51.6	7:3:0:4:5:0:2	1

Ergebnisse des Versuches Rangliste			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
4	12	0:0:0:6:15:7:11	25
2	12	0:0:0:6:15:7:11	36
5	12	0:0:0:6:15:7:11	5
6	12	0:0:0:6:15:7:11	10
6	12	0:0:0:6:15:7:11	12

Ergebnisse des Versuches Zusammen			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	117.11	5:3:0:4:7:0:4	1
1	117.11	5:3:0:4:7:0:4	1
1	117.11	5:3:0:4:7:0:4	1
1	117.11	5:3:0:4:7:0:4	1
1	117.11	5:3:0:4:7:0:4	1

Auswertung: Die Annäherung an dem Mittelwert liefert in diesem Versuch nur eine Lösungsstrategie, die bereits in der ersten Generation gefunden wird. Durch die Verschachtelung der Aktionen ist davon auszugehen, das die Anzahl der Aktionen um die zu testenden Weltzustände zu erreichen kaum variieren kann. Als Folge dessen tritt das gleiche Phänomen auf wie im ersten Versuch.
Die Ausrichtung nach der Rangliste erzeugt in diesem Versuch ein unerwartetes Verhalten. So können die Aktionen die Rangliste nicht mehr einhalten. Dies ist wahrscheinlich damit begründet, dass teilweise Aktionen von der Mehrheit der gültigen Aktionssequenzen benötigt werden und somit kaum in ihrer Häufigkeit verringert werden können.
Auch die Kombination aus beiden Parametern kann nicht die Rangliste einhalten. Trotz dessen nähert sich ein Großteil der Aktionen dem Mittelwert an. Es ist zu erkennen, dass die Häufigkeit von Aktionen angepasst wurde, damit sie der Rangliste entsprechen.

6.6 Mehrere Ziele

In diesem Abschnitt soll überprüft werden, ob die Implementierung auch genutzt werden kann, wenn mehrere Ziele verfolgt werden und die dafür genutzte Menge

an Aktionen gleich bleibt.
Zu Zeigen: Der Algorithmus konvergiert, wenn das Balancing um mehrere unterschiedliche für den GOAP Planer zu erreichende Ziele erweitert wird.
Aufbau: Es werden sieben Aktionen erstellt, die drei Aktionssequenzen bilden. Dabei werden drei Aktionen ausgewählt. Der Versuch wird dabei sowohl Aktionen auswählen, die die gleichen Zielzustände erzeugen, als auch Aktionen die sich diesbezüglich unterscheiden. Es wird nur die Kombination aus Mittelwert und Rangliste untersucht.
Annahme: Der Algorithmus konvergiert in beiden Fällen. Es findet eine Annäherung an dem Mittelwert statt.

Ergebnisse des Versuches für gleiche Zielzustände			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	6	5:5:5	20
3	6	5:5:5	17
1	6	5:5:5	4
5	6	5:5:5	19
2	6	5:5:5	8

Ergebnisse des Versuches für unterschiedliche Zielzustände			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	18	5:5:2	7
1	18	5:5:2	1
1	18	5:5:2	6
1	18	5:5:2	1
1	18	5:5:2	3

Auswertung: Der Algorithmus konvergiert für mehrere Ziele. Gleichzeitig wird die Häufigkeitsverteilung der Aktionen sowohl nach einem Mittelwert, als auch nach einer Rangliste balanciert.

6.7 Ausnahmefälle

In diesem Versuch sollen einige Ausnahmefälle überprüft werden, damit Aussagen zu treffen sind, wann die Implementierung Fehler erzeugt.
Zu Zeigen: Das Verhalten der Implementierung wird geprüft, wenn Aktionen balanciert werden sollen, die die gleichen Bedingungen und Effekte besitzen.
Aufbau: Es werden zwei Aktionen mit den gleichen Effekten und ohne Bedingungen konstruiert. Dem GOAP Planer stehen auch nur diese zwei Aktionen zur Verfügung.
Annahme: Der Algorithmus konvergiert nicht, wenn nur der Mittelwert betrachtet wird. Es werden zufällige Werte ausgegeben. Die Kombination aus Mittelwert und Rangliste bildet ein Extrem, bei dem die letzte Aktion nicht vorkommt und die Erste in allen restlichen Fällen.

Ergebnisse des Versuches für gleiche Effekte und Bedingungen Mittelwert			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	362.03	0:15	nicht feststellbar
1	362.03	0:15	nicht feststellbar
1	362.03	15:0	nicht feststellbar
1	362.03	0:15	nicht feststellbar
1	362.03	15:0	nicht feststellbar
Ergebnisse des Versuches für gleiche Effekte und Bedingunen Kombination			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	364.03	0:15	1
1	364.03	0:15	1
1	364.03	0:15	1
1	364.03	0:15	1
1	364.03	0:15	1

Auswertung: Der Versuch zeigt, dass die Implementierung in diesem Fall für die Bestimmung des Mittelwerts nicht konvergiert, weil die Werte sich zufällig abwechseln. Da in der GOAP Implementierung ein Standartverhalten eingebaut ist, für gleiche valide Aktionssequenzen, treten keine unterschiedlichen zufälligen Werte auf. Stattdessen wird eine Aktion von den zwei verfügbaren zufällig durch die geringeren Kosten ausgewählt und löst somit alle zu testenden Zustände. Im Zusammenhang mit dem Parameter für die Ausrichtung nach einer Rangliste kann nach einer Aktion entschieden werden. Folglich findet kein zufälliger Wechsel der Aktionen mehr statt.

Zu Zeigen: Das Verhalten der Implementierung wird gezeigt, wenn Aktionen balanciert werden sollen, die sich in einer Aktionssequenz befinden die in Aktionen enden, die die gleichen Effekte besitzen.
Aufbau: Es werden zwei Aktionen mit den gleichen Effekten konstruiert. Weiterhin werden zwei weitere Aktionen erstellt, die zusammen eine Kette bilden und an die zwei Aktionen anschließen. Letztere werden für die Balancierung ausgewählt.
Annahme: Die Aktionen können nicht balanciert werden. Ihre Häufigkeit repräsentiert ihr Vorkommen, als unabhängige Lösungen zu einigen der getesteten Weltzustände.

Ergebnisse des Versuches für Mittelwert			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	50.08	2:9	1
1	50.08	2:9	1
1	50.08	2:9	1
1	50.08	2:9	1
1	50.08	2:9	1
Ergebnisse des Versuches für Rangliste			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	4	2:9	1
1	4	2:9	1
1	4	2:9	1
1	4	2:9	1
1	4	2:9	1
Ergebnisse des Versuches für Kombination			
Beste Generation	Fitnesswert	Verteilung	Konvergenz ab Generation
1	52.08	2:9	1
1	52.08	2:9	1
1	50.08	2:9	1
1	52.08	2:9	1
1	52.08	2:9	1

Auswertung: Der Versuch zeigt, dass die Verteilung für die drei Ansätze sich nicht ändert. Weiterhin stimmt die Reihenfolge der Häufigkeiten nicht mit der Rangliste überein, weil sonst bei nur zwei Aktionen ein Fitnesswert von zwei entstehen müsste für die Balancierung nach nur der Rangliste.
Aus diesen Erkenntnissen ist zu schlussfolgern, dass diese Verteilung unabhängig von der Verteilung nach Mittelwert und Rangliste ist. Die Implementierung kann für diesen Ausnahmefall kein balanciertes Ergebnis erzeugen.

7

Bewertung der Methodik

Dieser Abschnitt der Arbeit dient dazu die Implementierung und die damit verbundenen Entscheidungen kritisch zu betrachten, damit die erzeugten Ergebnisse besser bewertet werden können. Gleichzeitig soll dadurch auch die Nützlichkeit der Balancierung einer GOAP Architektur mit einem Genetischen Algorithmus herausgearbeitet werden.

7.1 Vereinfachung der GOAP Architektur

Für die Implementierung der GOAP Architektur wurde nur der GOAP Planer und die damit verbundenen Systeme für Aktionen, Ziele und Weltzustände realisiert. Dadurch ist diese Umsetzung im Vergleich zu der von Jeff Orkins präsentierten GOAP Architektur unvollständig, weil ein System fehlt, dass die Aktionen des Planers umsetzt und somit auch die Weltzustände aktiv verändert. Durch diese Entscheidung kann die GOAP Architektur nur in einer „experimentellen“ Umgebung getestet werden, weil sie nicht mit einer Spielwelt direkt interagiert und somit auch den Spielfluss nicht beeinflusst. Folglich entfällt durch diese Vereinfachung die Chance die Implementierung in Zusammenhang mit einem fertigen Spiel zu testen. Dadurch kann die GOAP Architektur nur isoliert durch den Genetischen Algorithmus balanciert werden. Dies widerspricht der traditionellen Rolle von Testern und Designern, die versuchen die Handlungen eines Spielers nachzuahmen, um damit das Spiel zu optimieren. Indem die Aktionssequenzen so in einer vom Spiel abgrenzten Umgebung getestet werden, ist nicht ausgeschlossen, dass die Balancierung durch den Genetischen Algorithmus zu Fehlern führt, wenn die Änderungen in ein komplettes Spiel integriert werden würden.
Trotz dessen ist es dadurch effizienter möglich einzelne Szenarien zu überprüfen. Zum einen wird die Testzeit verkürzt, weil keine Aktionen in der Spielwelt ausgeführt werden müssen. Weiterhin steht dem Programm so mehr Rechenleistung zur Verfügung, weil Systeme für Animationen oder die Physik der Spielwelt nicht aktiv sind. Folglich wird so auch Testzeit eingespart.
Weiterhin können auch Situationen in der vereinfachten Betrachtung einfacher nachgebildet und nachvollzogen werden. In einer Spielwelt könnten je nach Programmierung viele verschiedene Umstände indirekt auf den Weltzustand Einwirkung haben. So könnte zum Beispiel andere System die Zustände verändern,

während ein Test stattfindet und somit ein ungenaues Ergebnis erzeugen. Diese Einflüsse in einer Spielwelt zu minimieren, würde viel Zeit in Anspruch nehmen, um sie zum einen zu identifizieren und zum anderen während der Tests zu verhindern. Durch die Entkopplung der Spielwelt in der vereinfachten Implementierung entfällt dieser Aufwand.
Letztlich ermöglicht dieser Ansatz auch potenziell ein Teil des Spieles früher zu balancieren. In der Entwicklung von Spielen ist die Phase des Testen und des Balancieren erst am Ende des Entwicklungszyklus [RW13]. Unter anderem, da Spiele während der Produktion nicht spielbar sind, weil noch nicht alle Systeme implementiert sind. Die Implementierung ist durch die Vereinfachung nur von einer GOAP Architektur abhängig und könnte somit auch genutzt werden, während sich andere Teile des Spieles in Entwicklung befinden.

7.2 GOAP Planer mit einer Baumstruktur

In der originalen GOAP Architektur von Jeff Orkins wird für den GOAP Planer ein A Stern Algorithmus benutzt, der auf einem Graphen den kürzesten Pfad zwischen dem Zustand der Welt in der, der Planer anfängt zu planen und einem gegebenen Zielzustand findet. Dabei stellen die Transitionen im Graphen, die Aktionen dar und die daraus resultierenden Zustände sind die Knoten. Die Laufzeit des Algorithmus ist abhängig von der verwendeten Heuristik und kann so eine Laufzeit von $\mathcal{O}(n * \log n)$ erreichen [Mil19]. Somit setzt sich der Aufwand dieser Variante aus der einmaligen Erstellung des Graphen und der wiederholten Suche des A Stern Algorithmus zusammen.
Im Gegensatz dazu sucht die implementierte Baumstruktur während der Baum erstellt wird auch nach der optimalen Lösung. Der Aufbau des Baumes hat ungefähr eine Laufzeit von $\mathcal{O}(n^n)$, dabei ist jedoch zu beachten, dass die Anzahl der verfügbaren Aktionen nicht konstant ist, sondern abnimmt, weil Aktionen nicht doppelt in einer Aktionssequenz vorkommen können. Weiterhin ist das n nur maximal die Anzahl der verfügbaren Aktionen und meistens sogar geringer, weil bei dem Aufbau des Baumes nicht alle Aktionen genutzt werden, sondern nur solche die sich durch die Bedingungen und Effekte mit ihren Elternknoten verketten lassen.
Ob die Baumstruktur besser ist, als ein Graph mit A Stern hängt davon ab, wie aufwendig es ist den Graphen zu erstellen. Bei gleichem Aufwand, wie die Erstellung des Baumes, würde die Suche mit dem A Stern Algorithmus eine schlechtere Laufzeit darstellen. Im Graphen repräsentiert jeder Knoten einen Weltzustand. Folglich existieren auch keine Duplikate dieser Zustände, weil zu einem Knoten mehrere Aktionssequenzen führen können, die den gleichen Weltzustand erzeugen. Im Baum hingegen ist dies nicht gewährleistet, weil zum Beispiel auf mehreren Blättern der Zielzustand erreicht werden kann und sich somit in unterschiedlichen Teilbäumen Knoten mit denselben Zuständen befinden können. Die einzelnen Knoten im Graphen miteinander zu verbinden hätte auch eine Laufzeit von $\mathcal{O}(n^n)$, weil für jeden neu erstellten Weltzustand n neue Zustände durch die n Aktionen ent-

stehen können.
In Anbetracht dessen ist zu schlussfolgern, dass die implementierte Baumstruktur besser ist als ein Graph. Weiterhin kann auch die Baumstruktur wie der Graph einmalig erstellt werden und der angestrebte Weltzustand würde als Knoten im Baum gesucht werden. Die Suche im Baum könnte durch eine lineare Suche über alle Knoten im Baum realisiert werden und wäre somit schneller als der A Stern Algorithmus.

7.3 Mehrere valide Aktionssequenzen zu einem Zustand

Der GOAP Planer kann mehrere Aktionssequenzen finden mit den gleichen Kosten, die einen angestrebten Zustand in der Spielwelt erzeugen können. Da ein Charakter für ein Ziel nicht mehre Aktionssequenzen ausführen kann, gibt der GOAP Planer nur einer dieser Sequenzen weiter. Folglich wird für diesen Fall ein Standardverhalten für den Planer festgelegt. So kann zufällig eine der Aktionssequenzen ausgewählt werden oder es wird immer eine bestimmte ausgegeben. Letzteres wurde auch in der Implementierung gewählt, weil ohne eine Zufallskomponente die Ergebnisse wiederholt erzeugt werden können.
In Bezug auf ein Balancing nach einem Mittelwert schränken sowohl ein festgelegte als auch eine zufällige Auswahl der Aktionen die Annäherung an einen Mittelwert ein. Bei einer zufälligen Aktionssequenz könnte der Genetische Algorithmus fehlerhaft werden, indem er nicht konvergiert, weil Ergebnisse entstehen, die nicht mehr nachvollziehbar sind, da sie auf Zufall basieren.
Für eine festgelegte Auswahl, zum Beispiel immer die erste gefundene Aktionssequenz, ist es vorstellbar das dadurch eine Aktionssequenz häufiger vorkommt, als andere. Folglich würde sich die Häufigkeitsverteilung der Aktionen schlechter an einem Mittelwert annähern, weil auch in diesem Fall der Genetische Algorithmus keinen Einfluss nehmen kann, welche Sequenzen in einem solchen Fall ausgewählt wird.
Für ein Balancing mit dem Genetischen Algorithmus wäre somit ideal, wenn der GOAP Planer anhand der Häufigkeit der Aktionen, sich für eine Aktionssequenz entscheidet, wenn mehrere diese validen Lösungen zur Verfügung stehen. Dafür müsste jedoch jede genutzte Aktionssequenz gespeichert werden, dafür ein Mittelwert bestimmt werden und anhand dessen wird die Entscheidung getroffen. Dadurch würde mehr Rechenzeit für einen Ausnahmezustand benötigt werden. Resultierend daraus wäre eine solche Implementierung nicht geeignet, weil der Aufwand für jede Ausführung des GOAP Planers für diese Bedingung erhöht wird. Weiterhin müsste diesbezüglich auch bestimmt werden, ob es für den Spieler einen merkbaren Unterschied gibt, im Vergleich zu zum Beispiel einer zufälligen Auswahl.

7.4 Fitnessfunktion des Genetischen Algorithmus

Die Fitnessfunktion wird von dem Genetischen Algorithmus genutzt, um die Individuen einer Generation zu bestimmen, die ihre Gene an die neue Generation

vererben. Jedes Individuum ist mit einer Lösungsstrategie zu einem bestimmtem Problem, das der Genetische Algorithmus zu lösen versucht, gleichzusetzen. Somit ist die Fitnessfunktion auch eine Bewertung der Lösungsansätze. Folglich wird diese in der Implementierung genutzt, um zu bestimmen, ob die Wahl der Aktionssequenz sich einem Mittelwert oder einer vorbestimmten Rangliste annähert. Je weiter die Werte entfernt sind von diesen Zielen, desto schlechter wird ein Individuum von der Fitnessfunktion bewertet.
Der Abstand zwischen den Werten wird in der Implementierung als Exponent für eine Zweierpotenz genutzt, weil somit größere Abstände durch die Fitnessfunktion schlechter bewertet werden. Es werden sowohl der Mittelwert, als auch der Wert für die Rangliste in der Fitnessfunktion als ein Ergebnis kombiniert. Als Folge dessen wurde auch für Bewertung nach einem Mittelwert auch nur der einfache Abstand zu diesem gewählt. Im Vergleich dazu könnte sonst die mittleren quadratischen Abweichung genutzt werden.

7.5 Vererbung der Gene

Für die Vererbung werden in der Implementierung nur die zwei besten Individuen ausgewählt. Indem nur eine geringe Anzahl an Individuen ihre Gene weitergibt, konvergiert der genetische Algorithmus schneller, es wird jedoch die Chance erhöht, das dabei ein lokales Optimum getroffen wird [Jeb13]. Also eine Lösung, die von dem Genetischen Algorithmus nicht weiterentwickeln werden kann, aber nicht der besten Lösung entspricht.

7.6 Laufzeit

Die Laufzeit des GOAP Planer ist abhängig davon wie viele Aktionen in einer Aktionssequenz enthalten sind [Ork05]. Je mehr Aktionen durchsucht werden müssten, desto höher ist Laufzeit. Die Anzahl der Aktionen in der Implementierung kann frei gewählt werden. Falls diese eingesetzt werden sollte, wird diese wahrscheinlich für eine hohe Anzahl an Aktionen genutzt. Begründet ist dies dadurch, das durch mehr Aktionen, der Aufwand für einen Designer steigt, diese manuell untereinander zu balancieren.
Weiterhin erhöht sich die Laufzeit, je mehr Zustände der Welt getestet werden, weil für jeden diese Zustände eine Aktionssequenz von dem GOAP Planer bestimmt wird. Resultierend daraus ist die Implementierung am effektivsten, wenn sie für speziell ausgewählte Zustände genutzt wird. Eine potenzielles Balancierung nach vielen Weltzuständen würde durch den faktoriellen Aufwand stark an zeitlichen Aufwand zunehmen.

7.7 Multithreading

Der vorgestellte Ansatz wurde nicht für Multithreading optimiert. So könnten die Individuen einer Generation parallel bewertet werden oder auch die Pläne für ver-

schiedene Weltzustände könnten gleichzeitig erstellt werden. Durch die verwendete GOAP Architektur mit ScriptableObjects ist dies jedoch mit dem Stand zur Erstellung des Projektes und den verfügbaren Lösungen von Unity nur schwer möglich. So können ScriptableObjects nicht von dem Unity Jobssystem, die Multithreading Lösung von Unity, benutzt werden. Folglich müsste eine Anbindung geschaffen werden die ScriptableObjects in Klassen umwandelt, die von dem Jobssystem genutzt werden können.

7.8 Bedienung

Die Implementierung wurde in Unity3D umgesetzt und nutzt somit auch die dadurch bereitgestellten Funktion um Werte außerhalb der Laufzeit eines Programmes zu bearbeiten. Dadurch können wie Abbildung 7.1 zeigt im Unity Inspektor die Werte eingeben werden und während er Laufzeit verändert werden. Das Programm an sich dient nur der initialen Ausführung und der visuellen Darstellung der Ergebnisse und ist somit nicht einzeln zu betrachten. Die dadurch entstandene Abhängigkeit, die Anwendung immer in der Unity Engine auszuführen wurde bewusst getroffen, weil es sich bei Implementierung nicht um ein eigenständiges Programm wie ein Spiel handelt, sondern eher um ein Tool, das während der Entwicklung genutzt wird.

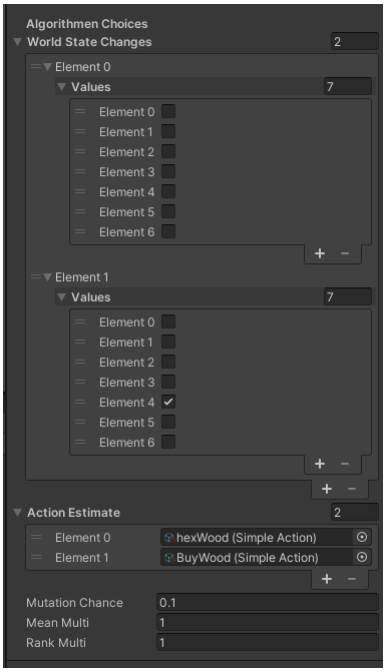

Abb. 7.1. Ausschnitt des Inspektors aus der Implementierung

8

Alternative Lösungsansätze

Mit dem Ziel, die Ergebnisse des GOAP Algorithmus zu balancieren, wäre es möglich den Genetischen Algorithmus durch ein anderes System zu ersetzten. So könnte ein Neuronales Netz aufgebaut werden, das über Reinforcement Learning es schafft, die Kosten der Aktionen für die GOAP Architektur so zu bestimmen, dass diese ausgewogen benutzt werden. Verfügbare Aktionen können sich jedoch ändern und Aktionssequenzen variieren. Als Folge dessen müsste ein solcher Ansatz wahrscheinlich für jede Änderung an den verfügbaren Aktionen oder den zu prüfenden Weltzuständen, neu gelernt werden. Dies betrifft in gewissen Teilen auch die Implementierung mit einem genetischen Algorithmus. Dieser weist jedoch im Gegensatz zum Reinforcement Learning eine geringe Lernzeit auf [Xia19].
Weiterhin ist es möglich die Balancierung der Aktionen an einen Designer abzugeben und die Implementierung so zu ändern, das sie nur potenzielle dominante Aktionssequenzen erkennt. Das bedeutet, der Designer würde für jede Aktionen selber Kosten festlegen. Anschließend könnte das Programm Probleme in seiner Umsetzung finden, in Form von Aktionssequenzen die vorgegebene Situation zum Beispiel zu häufig genutzt werden. Diese Idee unterscheidet sich zu dem in der Arbeit vorgestellten Ansatz, indem das die Kosten der Aktionen nicht verändert werden. Folglich müssten nur diese dominanten Aktionen erkannt werden. Dafür sollte jedoch diese Methode der Erkennung herausgearbeitet werden. Insbesondere, wenn das Programm eine bestmögliche Lösung bereitstellen soll. So wäre es zum Beispiel naiv möglich die Kosten aller Aktionssequenzen zu vergleichen und damit Rückschlüsse zu treffen wie häufig eine Sequenz für eine bestimmte Menge an Weltzuständen genutzt wird. Sollte so ein Ergebnis gefunden werden, müsste außerdem auch noch bestimmt werden, welche Aktionen und ihre Werte geändert werden sollten. Ansonsten müsste ein Designer raten oder nach jeder seiner Änderungen das Programm neu ausführen. Insbesondere in komplexeren Situationen, bei denen Aktionssequenzen mehre Aktionen untereinander teilen, sollte dieser Ansatz noch weiter betrachtet werden. Denn in Verbindung damit ist es erschwert Rückschlüsse zu treffen auf Änderungen anhand der festgestellten dominanten Aktionen, weil durch die erhöhte Komplexität einzelnen Änderungen der Kosten wieder neue bevorzugte Sequenzen erzeugen könnten.

9

Zusammenfassung und Ausblick

In dieser Arbeit wurde herausgearbeitet, dass ein Genetischer Algorithmus die GOAP Architektur nach bestimmten Parametern balancieren kann. Dabei wurde eine Implementierung vorgestellt, die die Häufigkeitsverteilung von Aktionen versucht zu ändern. Dafür werden verschiedene Zustände der Spielwelt genutzt für die Aktionssequenzen von dem GOAP Planer anhand von verfügbaren Aktionen erstellt werden. Aus den daraus ermittelten Zahlen, ist es möglich eine Auswahl an Aktionen nach einem Mittelwert, einer Rangliste und einer Kombination aus diesen zwei zu balancieren.
Weiterhin wurde diese Implementierung mit verschiedenen Versuchen getestet. Es wurde gezeigt das die Verteilung der Aktionen einem Mittelwert entsprechen kann. Dabei konnte festgestellt werden, das bereits in der ersten Generation des Genetischen Algorithmus für einfache Konstrukte aus Aktionen und zu prüfenden Zuständen der Spielwelt, eine Lösung gefunden werden konnte. Für komplexere Situationen findet auch eine Annäherung an den Mittelwert statt.
Im Zusammenhang mit der Balancierung nach einer Rangliste wurde gezeigt, das dieser Parameter allein stehend nicht die erwarteten Ergebnisse erzeugt. So können zwar die meisten Aktionen nach der Rangliste in ihrer Häufigkeit sortiert werden, aber die Verteilung dieser Werte tendiert zu Extremen. So wurde gezeigt, das vermehrt Aktionen erzeugt werden, die für fast alle Zustände genutzt werden, und der Rest wird kaum oder gar nicht benutzt. Begründet ist dies durch die Umsetzung dieser Funktion. So werden die Aktionen nach ihrer Verteilung sortiert und dann der Abstand zu ihrer vorbestimmten Position in der Rangliste bestimmt. Als Folge dessen können gleiche Werte unterschiedliche Positionen in der Rangliste repräsentieren. Der Genetische Algorithmus nutzt dies und erzeugt somit die Extremen, weil diese durch die geringen unterschieden in den Aktionskosten einfach zu erreichen sind.
Die Kombination aus beiden Parametern hat in den meisten Fällen eine ähnliche Verteilung der Aktionen erzeugt, wie die Versuche, die nur den Mittelwert genutzt haben. Dies ist auch wieder auf die Implementierung der Bewertung nach der Rangliste zurückzuführen. Indem gleiche Werte unterschiedliche Ränge haben können, wird es begünstigt, dass nur der Mittelwert betrachtet wird. Trotz dessen konnte auch gezeigt werden, das in komplexeren Situationen vom Mittelwert abgewichen wird damit Werte an die Rangliste angepasst werden.

Anhand dieser Erkenntnisse lässt sich schlussfolgern, dass die Balancierung nur nach einem Mittelwert besser abschneidet als nach einer Rangliste oder einer Kombination aus beiden. Weiterhin ist auch festzustellen, dass die angestrebten Werte mit einer kleinen Auswahl an Aktionen und einfach strukturierten Sequenzen am besten nachgebildet wurden.
In Hinblick auf eine praktische Nutzung ist dies jedoch abzuwägen. So könnte auch ein Designer für eine geringe Anzahl an Aktionen manuell Tests in einem Spiel ausführen und diese anhand seiner Erkenntnisse balancieren. Dem gegenüber steht jedoch, das die Implementierung wahrscheinlich schneller und objektiver diese Ergebnisse erzeugen kann. Außerdem wurde festgestellt das so ein Spiel früher balanciert werden kann, weil der Versuchsaufbau nur von der GOAP Implementierung abhängig ist. Ein menschlicher Tester oder ein Designer können dem gegenüber das Spiel erst testen, wenn sie es auch spielen können. Dafür werden diverse weitere Systeme benötigt, die in den verschiedenen Phasen der Entwicklung noch nicht implementiert sein könnten.
Der gezeigte Aufbau ersetzt jedoch nicht einen Designer. In vielen Fällen ist ein Spiel komplexer und somit würde nur die Balancierung nach einem Mittelwert nicht ausreichen. So würde die Implementierung wahrscheinlich eher eingesetzt werden, um einen Designer zu unterstützen.
Zusammenfassend hat die Arbeit gezeigt, das ein Genetischer Algorithmus mit der GOAP Architektur verwendet werden kann und diese nach gewissen Kriterium, wie einer Verteilung nach einem Mittelwert, balanciert werden kann. Diese zwei Aspekte bilden auch die Ansatzpunkte für zukünftigen Arbeiten. So könnte weiter herausgearbeitet werden, mit welchen anderen Methoden die GOAP Architektur balanciert werden könnte. Daran anknüpfen könnte der Einfluss einer Balancierung nach bestimmten Kriterien auf die Funktionsweise der GOAP Architektur untersucht werden. Auch der Genetische Algorithmus als Form der Balancierung könnte in weiteren Arbeiten auf weitere Algorithmen zur Entscheidungsfindung angewendet werden. Ein Beispiel dafür könnte ein Genetischer Algorithmus mit einem Behavior Tree sein.

Literaturverzeichnis

Ama21. Amazon: *AUTOMATIC GAME LEVELING WITH REINFORCEMENT LEARNING*, 2021.
`https://automatic-game-leveling.workshop.aws/en`
Zuletzt geprüft am 29.07.2021.

ARSC05. Andrade, Gustavo, Geber Ramalho, Hugo Santana und Vincent Corruble: *Automatic computer game balancing.* In: Pechoucek, Michal, Donald Steiner und Simon Thompson (Herausgeber): *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems - AAMAS '05*, Seite 1111, New York, New York, USA, 2005. ACM Press.

BG16. Boris, Tuponja und Sukovic Goran: *Evolving neural network to play game 2048.* In: *2016 24th Telecommunications Forum (TELFOR)*, Seiten 1–3. IEEE, 22.11.2016 - 23.11.2016.

BKM+20. Baker, Bowen, I. Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew und Igor Mordatch: *Emergent Tool Use From Multi-Agent Autocurricula.* ArXiv, abs/1909.07528, 2020.

CSN+19. Cordeiro, Matheus G., Paulo Bruno S. Serafim, Yuri Lenon B. Nogueira, Creto A. Vidal und Joaquim B. Cavalcante Neto: *A Minimal Training Strategy to Play Flappy Bird Indefinitely with NEAT.* In: *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, Seiten 21–28. IEEE, 28.10.2019 - 31.10.2019.

GBTG. Gordillo, Camilo, Joakim Bergdahl, Konrad Tollmar und Linus Gisslén: *Improving Playtesting Coverage via Curiosity Driven Reinforcement Learning Agents.*

Jeb13. Jebari, Khalid: *Selection Methods for Genetic Algorithms.* International Journal of Emerging Sciences, 3:333–344, 2013.

Kaz19. Kazuko Manabe, Shigeru Awaji: *Balancing Nightmares: an AI Approach to Balance games with Overwhelming Amounts of Data: Grimms Notes Repage*, 21.03.2019.

LBH18. Lin, Dayi, Cor-Paul Bezemer und Ahmed E. Hassan: *An empirical study of early access games on the Steam platform.* Empirical

Software Engineering, 23(2):771–799, 2018.

Mil19. Millington, Ian: *AI for Games*. CRC Press, Third edition. — Boca Raton : Taylor & Francis, a CRC title, part of the Taylor & Francis imprint, a member of the Taylor & Francis Group, the academic division of T&F Informa, plc, [2019], 2019.

Mit96. Mitchell, Melanie: *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1996.

Ork04. Orkin, Jeff: *Symbolic Representation of Game World State: Toward Real-Time Planning in Games*. 2004.

Ork05. Orkin, Jeff: *Agent Architecture Considerations for Real-Time Planning in Games*. In: Young, R. Michael und John E. Laird (Herausgeber): *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA*, Seiten 105–110. AAAI Press, 2005.

Ork06. Orkin, Jeff: *Three States and a Plan: The A.I. of F.E.A.R.* 2006.

Ork08. Orkin, Jeff: *Applying Goal-Oriented Action Planning to Games*. 2008.

Owe15. Owens, Brent: *goap*. `https://github.com/sploreg/goap`, 2015.

OYH08. Olesen, Jacob Kaae, Georgios N. Yannakakis und John Hallam: *Real-time challenge balance in an RTS game using rtNEAT*. In: *2008 IEEE Symposium On Computational Intelligence and Games*, Seiten 87–94. IEEE, 15.12.2008 - 18.12.2008.

RW13. Ramadan, Rido und Yani Widyani: *Game development life cycle guidelines*. In: *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, Seiten 95–100. IEEE, 28.09.2013 - 29.09.2013.

SM02. Stanley, Kenneth O. und Risto Miikkulainen: *Evolving neural networks through augmenting topologies*. Evolutionary computation, 10(2):99–127, 2002.

Uni21. Unity Technologies: *Unity Game Simulation*, 2021.
`https://unity.com/de/products/game-simulation`
Zuletzt geprüft am 29.07.2021.

Whi95. Whitley, D.: *Genetic Algorithms and Neural Networks*. In: *Genetic Algorithms in Engineering and Computer Science*, Seiten 191–201. John Wiley, 1995.

Xia19. Xiang, Z.: *A comparison of genetic algorithm and reinforcement learning for autonomous driving*. 2019.

A

Erklärung der Kandidatin / des Kandidaten

☐ Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

☐ Die Arbeit wurde als Gruppenarbeit angefertigt. Meine eigene Leistung ist ...

Diesen Teil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser: ...

Datum	Unterschrift der Kandidatin / des Kandidaten