C:\Users\haller\Desktop\Logo_HFU.tif

Ersetzen Sie alle [ ]-Klammern durch korrekte Daten.

PS: Diese Gedankenblasen sollten vor der Abgabe natürlich alle entfernt werden!

[Art der Arbeit]

in

[Studiengang]

[Titel der Arbeit] Neuroevolution benutzerdefinierter mehrbeiniger Kreaturen

[Untertitel der Arbeit]

**Achtung**: Diese Vorlage kann Fehler enthalten! Bitte prüfen Sie die Einhaltung der Richtlinie nochmal selbst nach. Für Hinweise auf Fehler in der Vorlage bin ich Dankbar!

|  |  |
| --- | --- |
| Referent: | [Name] |
| Korreferent: | [Name] |
| Vorgelegt am: | [Datum] |
| Vorgelegt von: | [Name] |
|  | [Matrikelnummer] |
|  | [Straße und Hausnummer]  [PLZ und Stadt] |
|  | [HFU-Emailadresse] |

# Vorwort

Das Vorwort ist optional. Wenn Sie kein Vorwort in Ihrer Arbeit verwenden wollen können Sie dieses Kapitel entfernen!

# Abstract

[Englisch, 100 -120 Worte]

[Deutsch, 100 – 120 Worte]

# Inhaltsverzeichnis

[Vorwort I](#_Toc331872775)

[Abstract III](#_Toc331872776)

[Inhaltsverzeichnis V](#_Toc331872777)

[Abbildungsverzeichnis VII](#_Toc331872778)

[Tabellenverzeichnis IX](#_Toc331872779)

[Abkürzungsverzeichnis XI](#_Toc331872780)

[1. Einleitung 1](#_Toc331872781)

[2. Grundlagen 3](#_Toc331872782)

[3. [Eigene Kapitel] 5](#_Toc331872783)

[4. Ausblick 7](#_Toc331872784)

[5. Fazit 9](#_Toc331872785)

[Literaturverzeichnis 11](#_Toc331872786)

[Stichwortverzeichnis 13](#_Toc331872787)

[Eidesstattliche Erklärung 15](#_Toc331872788)

[A. [Anhang] 17](#_Toc331872789)

# Abbildungsverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

Das Abbildungsverzeichnis muss vor der Finalen Abgabe entfernt werden, wenn keine Abbildungen in die Arbeit eingefügt wurden.

# Tabellenverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

Das Tabellenverzeichnis muss vor der Finalen Abgabe entfernt werden, wenn keine Tabellen in die Arbeit eingefügt wurden.

# Abkürzungsverzeichnis

|  |  |
| --- | --- |
| KGS  NEAT | Knochen-Gelenk-System  Neuroevolution of Augumented Topologies |
| ZMAX | höchste Z-Position, die von der Kreatur am Ende der Simulation erreicht wurde |
| ANN | Künstliches Neuronales Netzwerk (Artificial Neural Network) |
|  |  |
| DNN | Tiefes künstliches Neuronales Netzwerk (Deep Artificial Neural Network) |
|  |  |
|  |  |
|  |  |

# Einleitung

## Ausgangssituation

In dieser Thesis sollen mehrbeinige Kreaturen in einer Simulation selbstständig lernen Laufen, ohne dass ihnen Laufmuster gezeigt werden. Die Kreaturen laufen in Intervallen von 20 Sekunden, bevor sie auf ihre Startposition und Startpose zurückgesetzt werden. Entscheidend für die Qualität des Lernens einer Kreatur ist die höchste Z-Position, die von der Kreatur am Ende der Simulation erreicht wurde (im weiteren bezeichnet als).

Im Rahmen dieser Arbeit lernen die Kreaturen durch den genetischen Algorithmus [NEAT](#_Abkürzungsverzeichnis) Laufen. Es werden die Auswirkung auf das Lernverhalten verschiedener Konfigurationen und Parameter des [NEAT](#_Abkürzungsverzeichnis)-Algorithmus‘, die Inputs/Sensoren, (Outputs), Entlinearisierung, Fitness-Funktion und Hyperparameter betreffen, untersucht.

Des Weiteren wird ein Editor implementiert, in dem Kreaturen benutzerdefiniert gebaut und anschließend mit ihnen die Simulation gestartet werden können. Die Kreaturen werden aus Knochen, Gelenken und Rotationslimits gebaut und dadurch definiert. Es können von jedem Glied aus in 25 Richtungen neue Glieder gebaut werden, sowie deren Länge festgelegt werden ebenso können die Rotationslimits auf den jeweiligen X- und Y-Achsen eingestellt oder gänzlich blockiert werden. Es ist möglich sie zu serialisieren und deserialisieren, da sie in einem eigenen Format, dem KGS, gespeichert werden.

### 1.2 Zielsetzung

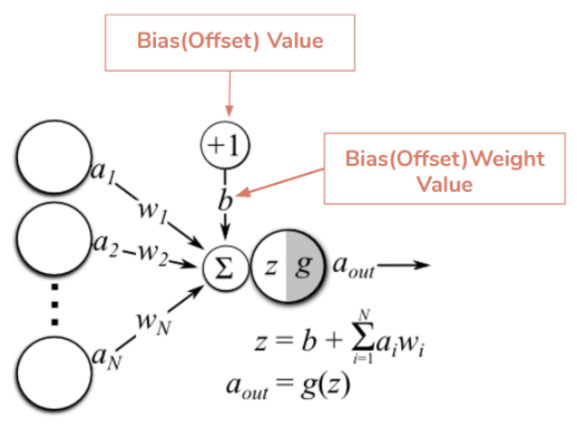
Ziel dieser Thesis ist es, herauszufinden, welche Konfigurationen bei mehreren sich signifikant voneinander unterscheidenden Kreaturen im KGS zu dem besten Lernerfolg führt. Dieser wird an dem Durchschnitt der [ZMAX](#_Abkürzungsverzeichnis)‘ über die verschiedenen Kreaturen bestimmt.

Auch soll die Frage beantwortet werden, ob es möglich ist, einen Editor zu realisieren, in dem dreidimensionale Kreaturen benutzerdefiniert erstellt werden können, die ohne weitere Eingriffe des Nutzers lernen zu Laufen.

# Grundlagen

## Künstliche Neuronale Netzwerke

Künstliche Neuronale Netzwerke (im Folgenden als ANN bezeichnet) sind Systeme, die von der Art der Problemlösung des menschlichen Gehirns inspiriert sind. Wie in seinem künstlichen Replikat, befinden sich im menschlichen Gehirn eine Vielzahl miteinander verbundener Neuronen [1]. Künstliche, digitale Neuronen stellen biologische Neuronen, auch Nervenzellen genannt [2], in der Funktionsweise ihrer Dendriten und Axone nach, indem sie mehrere Eingänge (von nun an als „Inputs“ bezeichnet) mit jeweiligen Gewichten verrechnen und zu einem Ausgang (von nun an als „Output“ bezeichnet) zusammenfassen.

Die Gewichte eines Neurons sind Skalare, die jeweils mit den skalaren Inputs multipliziert und deren Ergebnis summiert wird, um als Ergebnis einen Skalar für weitere Rechnungen zu erhalten. Werden die skalaren Inputs und Gewichte als Vektor betrachtet, stellt das Ergebnis das Skalarprodukt da. Zu diesem wird oft, wie in Abb. 2.1 zu erkennen, eine gewichtete Konstante als Bias addiert.

*Abbildung 2.1: Auszug aus [3]*

Die Summe des Skalarproduktes und eventuellem Bias wird meistens durch eine nichtlineare Aktivierungsfunktion, wie der Sigmoidfunktion, Logistischen Funktion, oder dem Tangens hyperbolicus, zwischen 0 und 1 oder -1 und 1 limitiert [1]. Das Ergebnis ist der Output des Neurons und in Abb. 2.2 als aout bezeichnet.

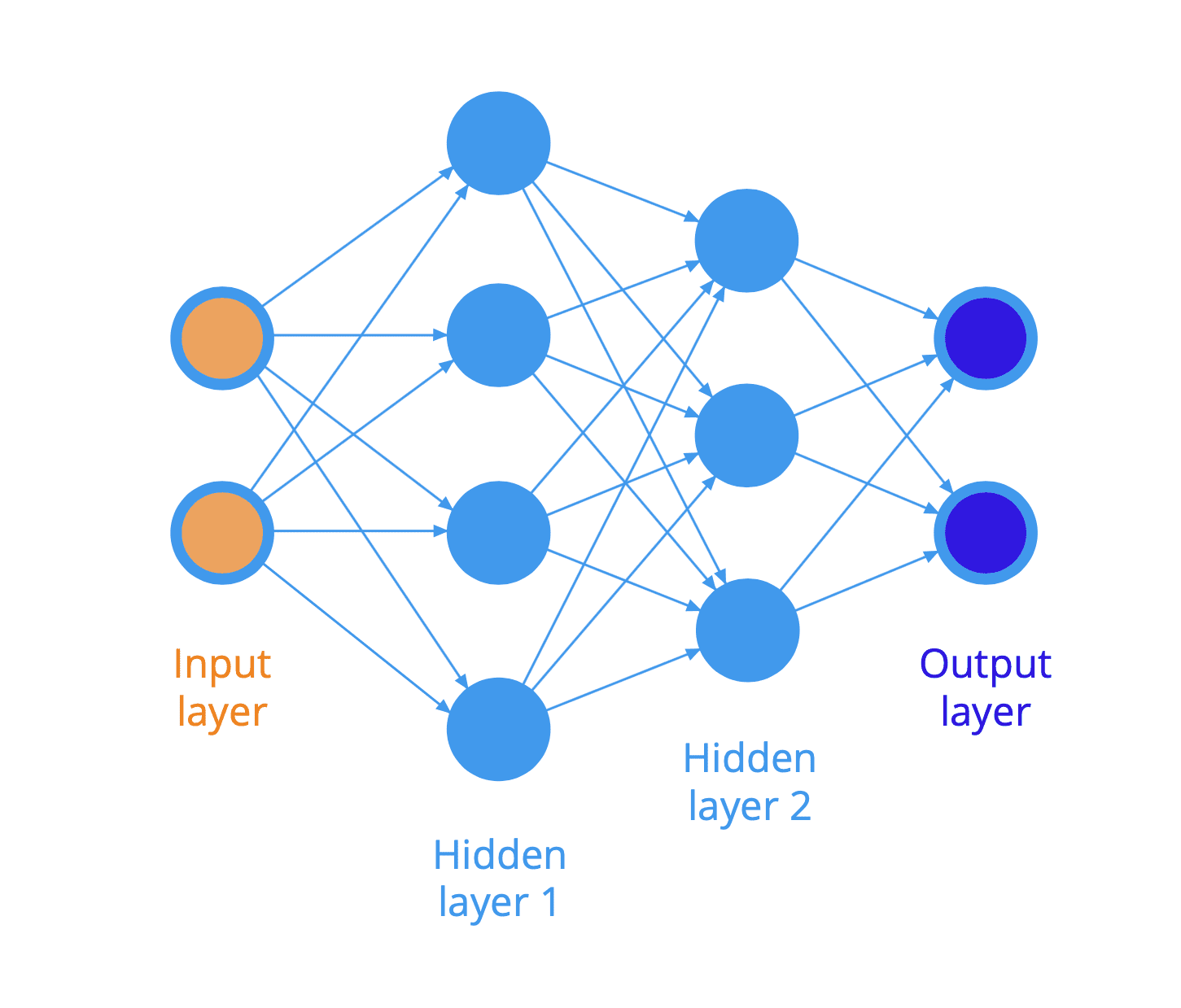
Der Output eines Neurons kann wiederrum einer der Inputs von einem oder mehreren anderen Neuronen sein, wodurch sich ein Neuronales Netzwerk bildet. Dieses hat, wie in Abb. 2.2 erkennbar, Inputs, welche nicht Outputs anderer Neuronen sind, sondern die Eingaben der bekannten Werte des Problems darstellen. Zu diesem Problem berechnet das Neuronale Netzwerk durch die Vernetzung der Neuronen, die oft auf „Hidden Layers“ stattfindet, und den Gewichten dieser wiederrum Outputs, welche nicht Inputs anderer Neuronen sind, sondern die Lösung des Problems quantifiziert darstellen.

Abbildung 2.2: Auszug aus [4]

Für das Lernen Neuronaler Netzwerke gibt es verschiedene Ansätze, worunter „Supervised Learning“ das häufigste ist. Es stellt das Vergleichen von erwartetem Output eines Inputs mit dem berechnetem Output eines Inputs und anschließendem manipulieren der Gewichte, um den erwarteten Output näher zu kommen, dar. Wenn es sich allerdings nicht um ein Klassifizierungs- oder Regressionsproblem handelt, da es beispielsweise keine Testdaten gibt, aus denen zu bestimmten Inputs erwartete Outputs hervorgehen, spricht man von sogenannten „Reinforcement Learning“-Aufgaben. Hier eignen sich andere Ansätze besser.

Da bei „Deep Q-Learning“ die Topologie des Netzes, insbesondere der Hidden Layer, sowohl eine große Rolle für den Lern- oder Trainingsprozess spielt, als auch vor Trainingsbeginn festgelegt werden muss und starr bleibt, wird nun in 2.2 der NEAT-Algorithmus beschrieben. Sein flexibles evolvieren der gesamten Topologie während des Trainings kommt hypothetisch der vollkommenen Unbekanntheit der Anzahl an Inputs und Outputs, sowie optimalen Anzahl von Hidden Layers oder Hidden Neuronen, zugute.

## Neuroevolution

Laut Kenneth O. Stanley und seinen Kollegen werden DNNs typischerweise mittels Gradienten basierten Lernalgorithmen namens Backpropagation trainiert. Evolutionäre Strategien könnten mit Backpropagation-basierten Algorithmen, wie Q-Learning und Policy Gradients, um das Bestreiten von RL-Problemen rivalisieren[5].

Evolutionäre System, auch Evolutionäre Algorithmen genannt, orientieren sich bei der Lösung eines Problems an der Evolution realer biologischer Organismen in Populationen, indem mehrere Individuen als mögliche Lösungen für das Problem über mehrere Generationen hinweg evolvieren. Dabei beschreibt die Fitness jedes Individuums quantitativ, wie gut es das Problem löst. Eine höhere Fitness eines Individuums führt zu einer höheren Wahrscheinlichkeit seiner Reproduktion in den nächsten Generationen. Die Evolution wird durch genetische Operatoren wie Selektion, bei der eine kleine Anzahl der Individuen mit der höchsten Fitness zur Reproduktion selektiert werden; Crossover, bei dem selektierte Individuen miteinander gekreuzt werden und Mutation, die zufällige Änderungen an Individuen vornimmt. Die Evolution der Individuen wird meistens wiederholt, bis eine maximale Anzahl an Generationen erreicht wurde oder eine andere Bedingung, wie das Vorliegen einer ausreichend guten Lösung, eintritt [6].

## NEAT

Neuroevolution of Augumenting Topologies ist ein von Kenneth O. Stanley und Risto Miikkulainen entwickelter evolutionärer Algorithmus. In NEAT werden Genome als lineare Repräsentation des Aufbaus eines ANNs, definiert durch die Verbindungen von Neuronen, hier Knoten (Nodes) genannt, codiert. Jedes Genom enthält eine Liste von Verbindungsgenen (Connection Genes) und eine Liste von Knotengenen (Node Genes). [7]

Jedes Verbindungsgen bestimmt den Eingangsknoten (In-Node); Ausgangsknoten (Out-Node); das Gewicht, mit dem der Eingang multipliziert wird, einen Status über die Aktivierung (Enabled / Disabled) und eine Innovations-Kennzahl (Innovation Number). [7]

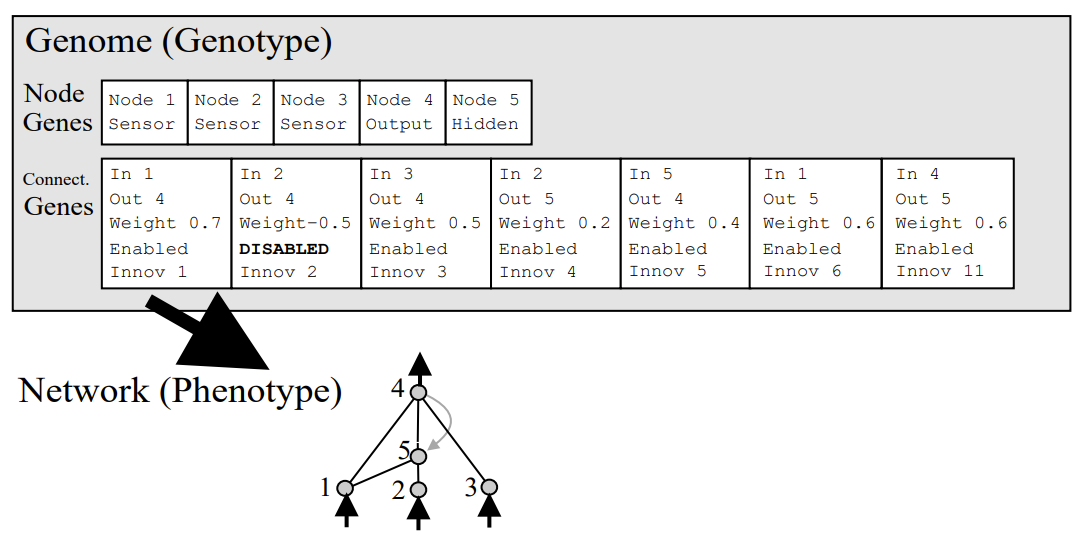
Jedes Knotengen ist eindeutig durch eine Kennzahl bestimmt und gibt Auskunft über die die Rolle im Netzwerk als Sensor, Hidden oder Output Node. [7]

Abbildung 2.4: Auszug aus [7]

### Mutation

In NEAT werden durch Mutation sowohl die Gewichte von Verbindungen, als auch die Netzwerkstruktur mit bestimmten Wahrscheinlichkeiten verändert, oder bleiben unberührt. Diese zwei Arten der Mutation, „Mutate Add Connection“ und „Mutate Add Node“ sind jeweils mit ihren Veränderungen in der Liste von Verbindungsgenen in Abb. 2.5 zu sehen [7].

Das Hinzufügen einer Verbindung erzeugt ein weiteres Verbindungsgen mit zufälligem Gewicht, hier die Verbindung von Knoten 3 zu Knoten 5 [7].

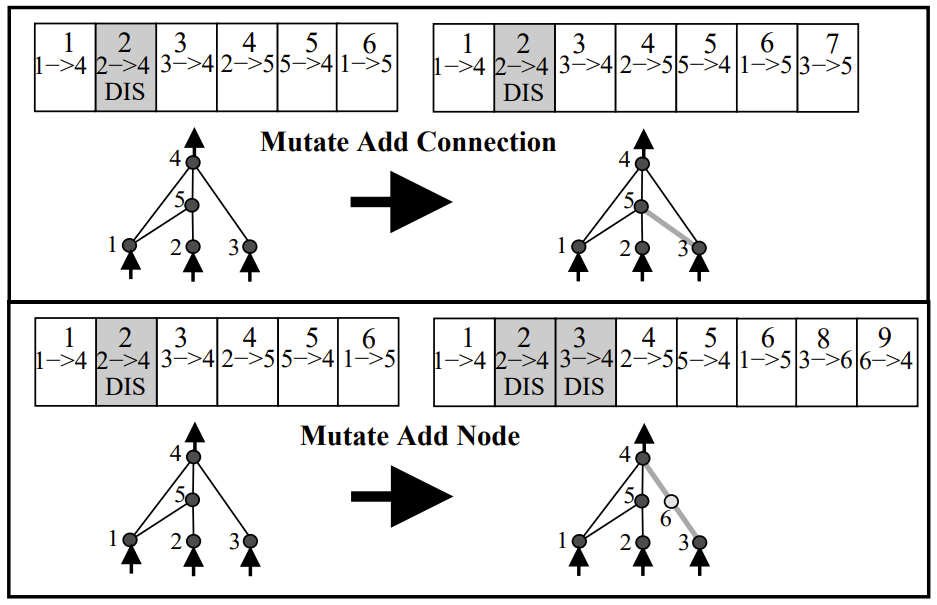
Das Hinzufügen eines neuen Knotens auf einer Verbindung verursacht das Deaktivieren des alten Verbindungsgens, sowie das Hinzufügen zwei neuer Verbindungsgene, hier das von Knoten 3 zu dem neuen Knoten 6, sowie das von diesem zu dem Knoten 4. In diesem Fall erhält das Verbindungsgen, welches als Output den neuen Knoten hat, ein Gewicht von 1, das Verbindungsgen zwischen dem neuen Knoten und dem Ursprünglichen Output das Gewicht des ursprünglichen Verbindungsgens, um den initialen Effekt der Mutation auf das Verhalten und die Fitness des Individuums zu verringern [7].

Abbildung 2.5: Auszug aus [7]

### Crossing

Durch die Mutationen werden die Genome allmählich größer und unterscheiden sich immer mehr in ihrem Aufbau, was die Kreuzung erschwert. Kenneth O. Stanley und Risto Miikkulainen lösen dieses Problem, indem sie historische Verläufe von Genen implementieren. Bei jeder Mutation wird eine globale Innovations-Kennzahl erhöht und dem entstandenen Gen zugeordnet [7].

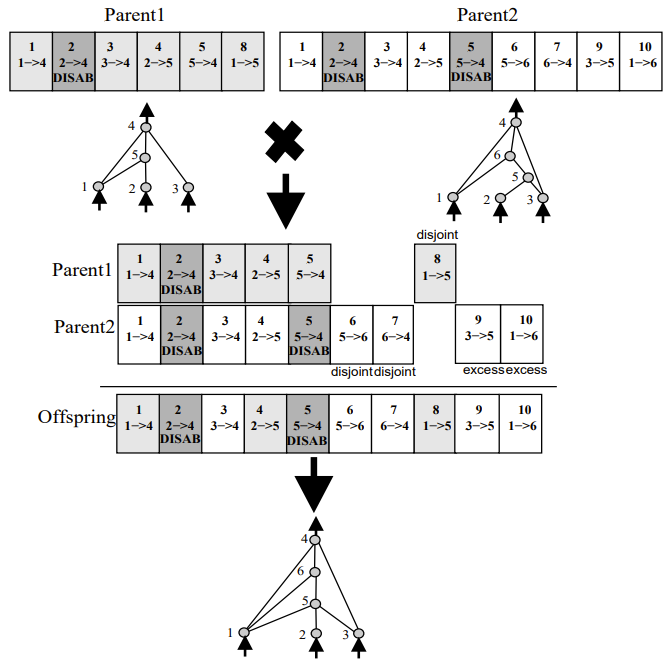
Wie in Abb. 2.6 erkennbar können mit diesem chronologischen Bezug der Gene im System zwei Genome, die nacheinander mutiert wurden und eine unterschiedliche Anzahl an Hidden Nodes aufweisen, gekreuzt werden. Beispielsweise folgt das Verbindungsgen 8 in der Liste von „Parent1“ auf das Verbindungsgen 5, weshalb bei der Kreuzung die Verbindungsgene 6 und 7 von „Parent2“ in „Parent1“ eingefügt werden können, wodurch 5 deaktiviert, ein neuer Knoten eingefügt und verbunden wird [7].

Abbildung 2.6: Auszug aus [7]

# Implementierung

Für die Implementierung wurde der Editor und die Game-Engine Unity gewählt, da sie bereits eine stabile Physics Engine enthält, die für diese Arbeit notwendig ist. Programmiert wurde ausschließlich in C#.

## Editor

Im Editor kann der User mit dem Drücken des Mausrades die Kamera um einen Fixpunkt rotieren. Drückt er gleichzeitig die Shift-Taste, verschiebt er die Kamera, sowie den Fixpunkt um die sie sich dreht. Diese Navigationssteuerung ist an die der Modelling-Software Blender angelehnt, in der die Geometrie für die Glieder erstellt wurde.

Der Editor soll es Benutzern ermöglichen, schnell Kreaturen im KGS zu bauen. Das Andocken neuer Glieder an andere soll mit möglichst wenigen Klicks und Komplikationen möglich sein. Schwebt die Maus über einem Glied, wird die Möglichkeit, ein Glied anzudocken, sowie die voraussichtliche Richtung, durch eine transparente Vorschau signalisiert.

Abbildung 3.1: Editor BauVorschau

Ein Bild, das Licht enthält.

Automatisch generierte BeschreibungWird nun die rechte Maustaste betätigt, wird das Glied angedockt und bleibt in einem Modus der Längeneinstellung. Mit der Position des Mauscursors kann nun die Länge des Glieds in festen Schritten eingestellt werden. Das Glied wird final gebaut, wenn die linke Maustaste erneut betätigt wird. Mit der rechten Maustaste kann das Bauen des Glieds abgebrochen werden. Es wurde sich dafür entschieden, in maximal 25 Richtungen pro Glied zu Bauen, anstatt in beliebig viele, um das Bauen symetrischer Kreaturen zu erleichtern. Dies erhöht die initiale Balance der Kreaturen, und beschleunigt somit das Lernen.

Abbildung 3.2: Editor, Längeneinstellung eines Glieds

Ein Bild, das Text, Segelflugzeug enthält.

Automatisch generierte BeschreibungDas Einstellen der Rotationslimits soll intuitiv erfolgen, und getrennt von dem Bauen erfolgen, weshalb sich für getrennte Modi entschieden wurde.

Sobald mindestens ein Glied an das Startglied angedockt wurde, kann mit der Taste „R“ in den Rotationsmodus, und wieder zurück gewechselt werden.

Abbildung 3.3: Handles für Einstellung der Rotationslimits

Die einzelnen Limits (Abb. 3.3) können im Rotationsmodus durch „Greifen“ der roten und grünen Handles und anschließendes Bewegen der Maus eingestellt werden.

Durch Drücken des „Save“ Buttons wird die vorliegende Kreatur auf einem neuen Slot gespeichert. Durch Drücken des „Load“ Buttons wird die Kreatur des nächsten Slots geladen. Ein überschreiben der Kreaturen ist momentan nicht implementiert.  
  
Durch Drücken des „Start“ Buttons wird die physikalische Simulation und das Training gestartet.

### State Machine

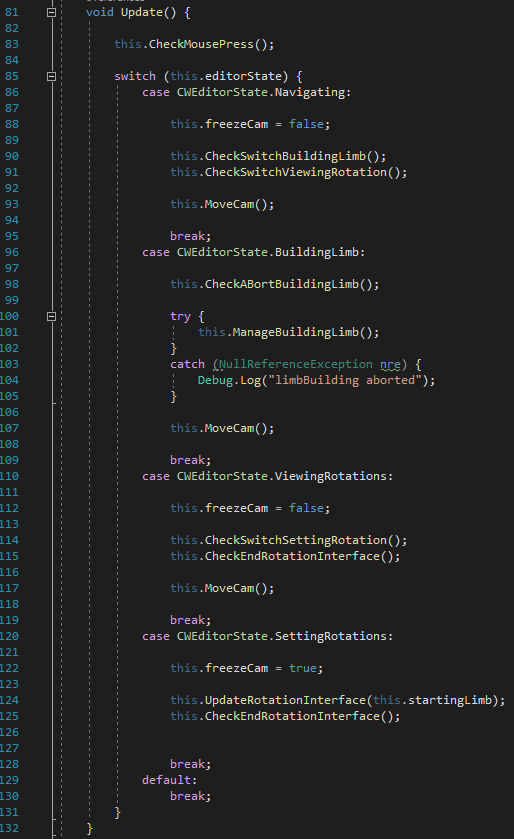


Abbildung 3.4

### In der Klasse CWEditorController wird in jedem Frame in der Update-Methode eine StateMachine durchlaufen, die die verschiedenen Editor-States (Navigating, BuildingLimb, ViewingRotations, SettingRotations), deren Wechsel und deren Logik steuert.

### Docking und Rotation

DockingPoints, Handles Addon

### Serialisierung

### Editor-Kreatur zu Trainings-Kreatur

-Massen, ArticulatedJoints (settings aus BP MLAgents, BodyParts)KGS System, DockingBalls, Problem Ausrichtung Docking Balls

Aktivitätsdiagramm, Klassendiagramm

## Training

Für die Implementierung des NEAT-Algorithmus‘ und Berechnungen von Neuronalen Netzwerken wurde auf das Package ANN&NEAT von VirtualStar zurückgegriffen [10].

ANN&NEAT registriert Instanzen der Klasse „CWCreatureBrain“ als „StudentChilds“, welche in 2.2 beschriebenen Individuen oder in 2.3 beschriebenen Genome verkörpern, und steuert anhand eingestellter Hyperparameter die in 2.2 und 2.3 beschriebenen genetischen Operationen.

CWCreatureBrain stellt, wie in Abb. 4.1 erkennbar, die Schnittstelle zwischen dem ANN&NEAT Package, und CWCreatureController dar. Hier wird in jedem Frame der Output des Netzwerks berechnet und dem CWCreatureController übergeben.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 4.1

Im CWCreatureController befindet sich der Teil der eigenen Implementierung, der die Sensoren der Kreaturen als Inputs berechnet, und die Outputs in Steuerbefehle der Gliedmaßen umsetzt, um die Kreatur zu bewegen. Die einzelnen Gliedmaßen werden mit BodyParts und JointDriveControllern realisiert, die Steuerbefehle von ConfigurableJoints, wie dem Einstellen eines JointDrives, zusammenfassen. BodyParts und JointDriveController, sowie wenige Methoden zum Aufsetzen dieser, wurden aus dem Unity-eigenen Package ML-Agents entnommen [11].

Bei Instanzierung eines CWCreatureController wird die Anzahl an Inputs und Outputs definiert. Diese hängt von der Anzahl beweglicher Rotationsachsen der Gliedmaßen ab, unterscheidet sich also bei verschiedenen im Editor erstellten Kreaturen stark.

Es wird versucht, die Anzahl an Inputs und Outputs minimal zu halten, da größere Netzwerke meistens wesentlich langsamer lernen als kleine. Um dies zu erreichen, werden zunächst nur bewegliche Gliedmaßen mit eigenen Sensoren, und somit eigenen Inputs ausgestattet (Abb.4.2), sowie Outputs nur je rotierbare Achse angelegt (Abb.4.3). Somit resultieren effizient aufgebaute Kreaturen in effizienteren Netzwerken.

Abbildung 4.2

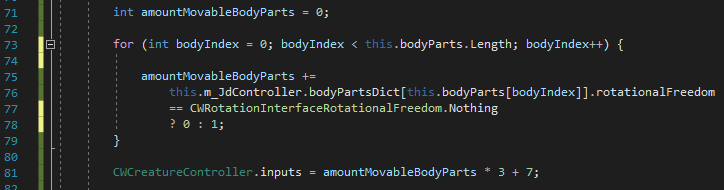
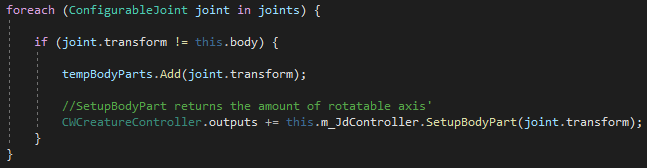


Abbildung 4.3

Unabhängig von dem Aufbau der Kreatur werden sieben Inputs pro Kreatur angelegt (Abb. 4.2), die als Sensoren des Startglieds, welches in ausnahmslos jeder Kreatur vorhanden ist, fungieren.

In der Funktion InputInputs() (Abb. 4.4) werden die Inputs auf Variablen der Kreatur, der Umgebung oder dem Bezug der Kreatur zur Umgebung, gesetzt. Die Wahl einiger dieser Werte resultiert biologisch inspirierten oder intuitiven Hypothesen, sie könnten im Training des NEAT Algorithmus‘ zu einer guten Wahrnehmung der momentanen Pose sowie daraus resultierendem Handlungsbedarf führen. Zuerst werden die sieben den Body betreffenden Sensoren verwertet, darunter die Geschwindigkeit, der Winkel der Geschwindigkeit zu der globalen Z-Achse, der Winkel des Bodys selbst zu der globalen Z-Achse sowie Skalarprodukte der lokalen Z-Achse des Bodys mit der globalen Z- und Ein Bild, das Text enthält.

Automatisch generierte BeschreibungY-Achse.

Abbildung 4.5

Abbildung 4.4

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Es folgt die Eingabe je dreier Werte pro bewegliches Glied durch die Methode CollectObservationBodyPartOptimized (Abb. 4.5). Zwei der Inputs werden auf Distanzen zu dem Masseschwerpunkt der gesamten Kreatur gesetzt, sowie mit der maximal geschätzten Distanz, die in der Kreatur vorkommen kann, skaliert. Die Ursache hierfür ist die Hypothese, eine Wahrnehmung des Bezugs eines jeden Glieds zu dem Masseschwerpunkt könne den Kreaturen helfen, Genome zu evolvieren, die Umkippen und Stolpern durch bessere Balance vermeiden. Ein weiterer Input erzeugt eine Abhängigkeit zu der Distanz zum Boden, der sich auf der Höhe Y=0 befindet.

Die Codezeilen 298-303 werden genauer in 4 erklärt.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungDie Methode CalculateFitness berechnet, wie in Abb. 4.6 erkennbar, im einfachsten Fall die Fitness als Z-Position der Kreatur. Dies geht aus der in 1.2 beschriebenen Zielsetzung hervor. Da Kreaturen mit negativer Fitness, wie im CWCreatureController codiert, sofort sterben, also aus der Simulation entfernt werden, wird auf diesen Wert eins addiert. Somit haben die Kreaturen in den ersten Frames einen Spielraum, für Sprünge oder Schritte auszuholen.

Abbildung 4.6

Für die Implementierung gibt es an mehreren Stellen verschiedene Konfigurationen. Die Standard-Konfiguration zeichnet den Ausgangszustand aus, die Variations-Konfiguration zeichnet den zu testenden Zustand aus.

* NEAT Genom pro Kreatur-Instanz
* Einige Kreaturen pro Welle
* NEAT Outputs
  + -1 bis 1
  + Standard-Config.:
    - Steuern Zielrotationen pro Glied
      * -1: kleinster möglicher Winkel
      * 1: größter möglicher Winkel
  + Variations-Config:
    - Steuern Zielrotationen pro Glied
      * -1: kleinster möglicher Winkel
      * 1: größter möglicher Winkel
    - Steuern maximale Kräfte pro Glied
* NEAT Inputs
  + Triviale und vorhersehbare Probleme
    - Negative Werte erreichen
    - Wertebereich -1 bis 1 ausschöpfen
    - Lernrelevante Einflüsse kodieren
  + Lösungen in dieser Thesis: Verschiedene Sensoren aus der Simulation
  + Standard-Konfig.:
    - Distanz zum CoM für quantifizierbaren Bezug zur Gewichtsverteilung des gesamten KGS pro Kreatur
  + Variations-Konfig:
    - Faktor der momentanen Rotation für Bezug zur Pose des gesamten KGS pro Kreatur

Bias?

* + Delinearisierung
    - Standard-Konfig.:
      * Tangens Hyperbolikus
    - Variations-Konfig:
      * Keine
* Wellen
  + Nach Dauer einer Welle -> nächste Generation, nächste Welle
  + UI für die MaxWaveTime
* Fitness
  + Standard-Konfig.:
    - Fitness = Z-Position
  + Variations-Konfig:
    - Kontinuierlich Z-Geschwindigkeit auf Fitness addieren
    - Kontinuierlich Z-Position auf Fitness addieren
    - Kontinuierlich X-Position von Fitness abziehen
    - Wenn Z-Pos zu negativ, abziehen
    - Kontinuierlich Winkel zwischen Body und forward von Fitness abziehen
* Hyperparameter
  + Standard-Konfig.:
    - Crossing aus
  + Variations-Konfig:
    - Crossing an

Aktivitätsdiagramm, Klassendiagramm

# Methode

## Konfigurationen

Konfigurationen werden im Rahmen dieser Arbeit definiert als Konstellation aus verschiedenen Berechnungstypen von Werten, die hypothetisch Einfluss auf den Lernfortschritt der Individuen nehmen. Die Berechnungstypen haben jeweils zwei Zustände und werden definiert als

1. Input Type (Abb. 5.1)

Berechnung der Inputs:

* 1. Com Distances – Distanz zum Masseschwerpunkt
  2. Ein Bild, das Text enthält.

     Automatisch generierte BeschreibungRotational Factor – Normalisierte momentanRotationen zwischen -1 als minimale, und 1 als maximale Rotation

1. Delinearization Type
2. Hyperparameter Type
3. Fitness Function Type
4. Output Type

* Delinearization Type
* Default



* RotationalFactor

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

* LinearInputs

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

* CrossingOn

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

* AlsoPunishX

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## Test-Kreaturen

Der Lernerfolg wird an fünf verschiedenen Kreaturen gelistet. Diese sind, mit ausnahme des Dreibeiners, in ihrer Form und den Rotationslimits an reale Lebewesen angelehnt, da sie in der Realität bereits Bewegungsmuster evolviert haben. Deshalb werden sie als sinnvoll für diese Untersuchung angenommen.

Um die Auswirkungen verschiedener

- Verschiedene Beinanzahl

- Angelehnt an die Tierwelt

Spinne

Ein Bild, das Outdoorobjekt enthält.

Automatisch generierte Beschreibung

### Data Collecting

- Konfigurationen

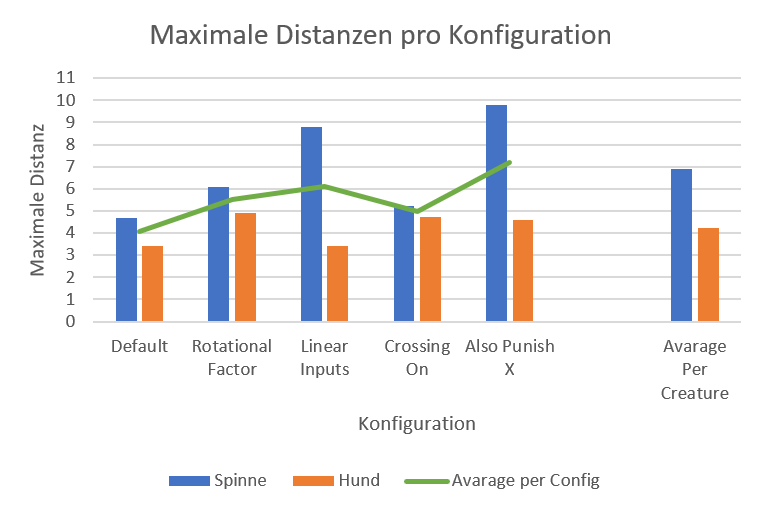
- inspirationen für konfigurationen

Test runs:

* Jede Konfiguration mit jeder Kreatur getestet
* Mehrere Batches pro Kreatur um Zufall zu verringern
* Pro Batch 5 Minuten Simulationszeit
* Pro Batch unterschiedlich viele Wellen
* Pro Welle wird die maximal-Distanz gespeichert
* Pro Batch wird ein ANN gespeichert

# [Eigene Kapitel] Resultate

Hier sind die „Maximale Distanzen“ Die Durchschnitte der Batch-Maxima. Die Batch-Maxima sind jeweils die Höchstwerte der Maximalen Distanzen pro Welle



Am besten Abgeschnitten hat „Also Punish X“ mit einer Durchschnittlichen Maximalen Distanz von 7,187. Darunter folgt die Konfiguration „Linear Inputs“ mit einer durchschnittlichen maximalen Distanz von 6,107 und „Rotational Factor“ mit einer durchschnittlichen maximalen Distanz von ….  
  
Der Unterschied zwischen der besten und schlechtesten Kreatur war bei Konfiguration …XY… am höchsten

* Fehlt in CSV:
  + Der Unterschied maximaler Distanzen bei verschiedenen Batches war bei Konfiguration …XY… am höchsten(Standardabweichung)

# [Eigene Kapitel] Auswertung und Diskussion

### -Warum war Konfiguration xy am besten?

# Ausblick

## Als Playground - Didaktisch, Forschend und Unterhaltend

## Was hätte man noch machen können

### Kreaturen mit Reibung

7.2.1 Besserer Editor

# Fazit

# Literaturverzeichnis

[1] R. E. Uhrig, "Introduction to artificial neural networks," Proceedings of IECON '95 - 21st Annual Conference on IEEE Industrial Electronics, Orlando, FL, USA, 1995, pp. 33-37 vol.1, doi: 10.1109/IECON.1995.483329.

[2] Larsen R. Nervensystem. Anästhesie und Intensivmedizin für die Fachpflege. 2016 Jun 14:13–25. German. doi: 10.1007/978-3-662-50444-4\_2. PMCID: PMC7531560.

[3] <https://machine-learning.paperspace.com/wiki/weights-and-biases>

[4] <https://machinelearningmastery.com/calculus-in-action-neural-networks/>

[5] „Deep artificial neural networks (DNNs) are typically trained via gradient-based learning algorithms, namely backpropagation. Evolution strategies (ES) can rival backprop-based algorithms such as Q-learning and policy gradients on challenging deep reinforcement learning (RL) problems.“ (Übersetzung durch Calvin Dell’Oro) Such, Felipe & Madhavan, Vashisht & Conti, Edoardo & Lehman, Joel & Stanley, Kenneth & Clune, Jeff. (2017). Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning.

[6] E. Galván and P. Mooney, "Neuroevolution in Deep Neural Networks: Current Trends and Future Challenges," in IEEE Transactions on Artificial Intelligence, vol. 2, no. 6, pp. 476-493, Dec. 2021, doi: 10.1109/TAI.2021.3067574.

[7] Kenneth O. Stanley, Risto Miikkulainen; Evolving Neural Networks through Augmenting Topologies. Evol Comput 2002; 10 (2): 99–127. doi: <https://doi.org/10.1162/106365602320169811>

[8] Blender https://www.blender.org/

[9] Unity Platform https://unity.com/products/unity-platform

[10] VirtualSTAR, ANN & NEAT Addon, Unity Asset Store <https://assetstore.unity.com/packages/tools/ai/ann-neat-138940>

[11] MLAgents, Unity https://docs.unity3d.com/Manual/com.unity.ml-agents.html

# Stichwortverzeichnis

#### **Keine Indexeinträge gefunden.**

Das Stichwortverzeichnis ist optional. Wenn Sie kein Stichwortverzeichnis in Ihrer Arbeit verwenden wollen können Sie dieses Kapitel entfernen!

# Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Thesis selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Alle verwendeten Quellen und Hilfsmittel sind angegeben. Der Einsatz von KI-Anwendungen ist dem betreffenden Thesisteil, der Art sowie dem Umfang nach detailliert benannt.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[Ort, Datum Name]

Auch hier müssen die Platzhalter mit den korrekten Daten ersetzt werden.

# [Anhang]

Der Anhang ist optional. Wenn Sie keinen Anhang in Ihrer Arbeit verwenden wollen können Sie dieses Kapitel entfernen!