Zur Verwendung von Reinforcement Learning nutzenden neuronalen Netzwerken zum Mapping für Pathfinding auf verschiedenen 3D Objekten - Untersuchung unter besonderer Berücksichtigung der Deep Q-Learning-Strategie und des A*-Algorithmus

Modulnummer:	CMN6302
Modulname:	Major Projekt
Abgabedatum:	30.08.2019
Abschluss:	Bachelor of Science
Semester:	März 2019
Name:	Yannick Denker
Campus:	Hamburg
Land:	Deutschland
Wortanzahl:	8770

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen, als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Bargteheide, 24.08.2019
Ort, Datum

Yannick Denker
Unterschrift Student

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1 Thema

Die Idee ist, ein neuronales Netzwerk mit Deep Q-Learning-Strategie so zu programmieren, dass dieses beim Erstellen von runden dreidimensionalen Objekten (zum Beispiel einer Kugel) in der Unity3D-Engine die benötigten Knotenpunkte für ein Pathfinding mit dem A*-Algorithmus sinnvoll bewertet.

1.2 Zielsetzung

Ergebnis des Praxisprojektes soll eine visuell ansprechende und selbsterklärende Demo für die Anwendung des A*-Algorithmus auf einer verzerrten Kugel sein, bei der die potentiellen Wegpunkte von einer Deep Q-Learning anwendenden künstliche Intelligenz (KI) bewertet werden. Die Umsetzung soll in Unity mit Hilfe des Machine-Learning-Pakets erfolgen. Unity soll außerdem für die Visualisierung in der Demo verwendet werden.

Für das Erstellen der Demo muss eine KI entwickelt und trainiert werden, die eine Kugel mit sinnvoller Verteilung der Scheitelpunkte bewertet und diese potentiellen Wegpunkte an den A*-Algorithmus weitergibt. Dementsprechend muss der A*-Algorithmus so angepasst und implementiert werden, dass es möglich ist, die potentiellen Wegpunkte von der KI anzunehmen, diese zum Pathfinding zu verwenden und einen Weg aus den Punkten zu erstellen. Damit die Demo visuell ansprechend und verständlich ist, muss in Unity aus den Informationen der KI ein Kugel-Mesh erstellt werden und der gefundene Weg aus dem A*-Algorithmus eingezeichnet werden.

Um die KI zu trainieren, muss eine Art von Learning-Algorithmus verwendet werden. Der ausschlaggebende Algorithmus wird ein Deep Q-Learning Algorithmus sein. Dieser basiert auf Lernen durch Belohnungen und ist ein Reinforcement-Learning-Algorithmus. Das Machine-Learning-Paket von Unity soll die Möglichkeit bieten das Projekt, welches in C# geschrieben wird, mit der TensorFlow software library zu verknüpfen. Die neuronalen Netzwerke innerhalb von TensorFlow sollen über eine Python Schnittstelle mit Hilfe von Anaconda, einer Open-Source-Distribution für Python, geschult und verwaltet werden.

Optional zu den genannten Inhalten der Demo kann die Anwendung des A*-Algorithmus und die Analyse durch die KI auf Objekte jeglicher Formen ausgeweitet werden und die Möglichkeit bestehen, eine Auswahl an Pathfinding-Algorithmen zu implementieren, aus der in der Demo ausgewählt werden kann. Um in der Demo den Entstehungsprozess darzustellen, besteht die Möglichkeit die KI in unterschiedlichsten Lernzuständen zu zeigen. Die KI wird nach dem “Trial and Error Prinzip” über viele tausend Iterationen geschult. Die Lernzustände können aus unterschiedlichen Iterationen entnommen und für die Demo verwendet werden. Zum Beispiel könnten die Lernstände nach 100, nach 1000 und nach 10000 Iterationen demonstriert werden.

1.3 Motivation

In der Vergangenheit hatte ich bereits mit Pathfinding-Algorithmen oder auch Searching-Algorithmen zu tun. Gegen Ende meiner Schulzeit habe ich mit einem kleinen Team aus mehreren Programmierern an der “Software Challenge Germany” teilgenommen. Wir haben einen Algorithmus programmiert, der gegen andere Teams Brettspiele gespielt hat. Dort hatte ich viel mit dem Dijkstra-Algorithmus und mit der Performance von Algorithmen zu tun, weil diese ausschlaggebend für das Gewinnen gegen die anderen Teams waren.
Diese Challenge war einer der Hauptgründe, weshalb ich mich für dieses Studium entschieden habe. Es liegt in meinem Interesse, Experte für das Thema Pathfinding zu werden und mein Wissen zu erweitern.

Mit dem Thema künstliche Intelligenz hingegen habe ich mich im Studium das erste Mal beschäftigt und ich hatte weniger Wissen über diesen Themenbereich. Die aktuelle Relevanz dieses Themas ist für mich ein Ansporn, meine Kenntnisse auf diesem Gebiet zu erweitern, insbesondere da künstliche Intelligenz immer bedeutsamer in Videospielen wird.

Durch mein Basiswissen von Pathfinding-Algorithmen und mein Engagement, neue relevante Dinge zu lernen und auszuprobieren, bin ich sehr zuversichtlich, dass ich es schaffen werde, eine Demo zu erstellen, in der sowohl künstliche Intelligenz, als auch meine Expertise in Pathfinding zur Geltung kommen.

Künstliche Intelligenz ist schon in vielen unterschiedlichen Arten in Videospielen zu finden. Sei es in Rennspielen bei denen Autos die Fahrstile der Spieler analysieren und den Eigenen entsprechend anpassen oder in Shootern Bot-Gegner, die mit der Zeit besser werden, weil sie vom Spieler lernen. Fast überall findet man künstliche Intelligenz in Spielen. Nicht nur beim Bedienen wichtiger Elemente innerhalb eines Spiels, sondern auch als als autarke Spielfigur welche sich vollumfänglich durch künstliche Intelligenz steuert und alle mögliche Arten von Aufgabenstellungen meistert.

Das Thema Pathfinding ist sehr komplex und hat ebenfalls fast immer in irgendeiner Form Anwendung in Videospielen. Vor Allem in Spielen des Real-Time-Strategy-Genres wird eine Art des Pathfindings verwendet, welche zum Steuern und kommandieren der Einheiten benutzt wird. Aber nicht nur dort, sondern auch in Spielen des MOBA-Genres und anderen Top-Down-View-Spielen finden man Pathfinding in geringem Maß.

2. Kontext

2.1 Pathfinding

2.1.1 Allgemein

2.1.1.1 Definition

Beim Pathfinding werden Algorithmen verwendet, dessen Aufgabe es ist, in einer Umgebung mit einem Start- und einem Zielpunkt einen möglichst kostengünstigen oder den kürzesten Weg zwischen diesen beiden Punkten zu finden (vgl. Definitions.net, STANDS4 LLC 2019). Diese Algorithmen verwenden Bewertungsfunktionen und Heuristiken, um den optimalen Weg zu ermitteln. Am häufigsten werden Pathfinding-Algorithmen bei Netzwerk-Flussanalysen, bei Routenplanungen und in Computerspielen verwendet.

2.1.1.2 Verwendung

In den meisten Fällen ist die optimale Route zwischen zwei Punkten auch der kürzeste oder kostengünstigste Weg. Ohne Hindernisse und andere Faktoren welche die Algorithmen in ihrer Bewertungsfunktion mit einberechnen, wäre in der Praxis der optimale Weg die Luftlinie zwischen dem Start- und Zielpunkt.

Störende Faktoren bestimmen jedoch die Suche und verändern den Begriff des Optimums und lassen dadurch andere Routen sinnvoller erscheinen. Folgende Störfaktoren sind die am häufigsten Auftretenden:

1. Hindernisse, die den Weg an bestimmten Stellen nicht oder nur bedingt passierbar machen
2. variable Kosten in der Fortbewegung, wie zum Beispiel ein erhöhter Treibstoffverbrauch entgegen einer Strömung
3. mehrdimensionale Kostenmodelle in der Fortbewegung, wie zum Beispiel Zeit, Treibstoff und Unfallgefahren
4. eine Rasterung oder Diskretisierung der Umwelt in Schachbrett oder Spielfeld ähnliche Konzepte
5. bestimmte Zwischenziele, welche während der Route erreicht oder passiert werden müssen
6. nicht kartesische Koordinaten

Um den Anforderungen an Pathfinding je nach Kontext zu entsprechen, wurde in der Vergangenheit eine Vielzahl von Algorithmen entwickelt, von denen fast alle ihre speziellen Vor- und Nachteile haben.

Pathfinding im Bereich der Computerspielindustrie gibt es schon seit sehr langer Zeit. Einer der ersten Benutzer von Pathfinding-Algorithmen ist der bekannte Spieleklassiker Pac-Man,

bei dem einfache Pathfinding-Algorithmen dafür sorgen, dass auf dem Spielfeld die Gespenster als Computergegner durch ein Labyrinth finden. Später wurden sie vor allem in Echtzeit-Strategiespielen, wie Warcraft 3 oder der Anno-Reihe zur Routenplanung der Einheiten verwendet, aber auch in Ego-Shootern, wo sie den computergesteuerten Bot-Gegner zur Orientierung dienten. Abbildung 1 zeigt ein Beispiel von Pathfinding innerhalb des Videospiels Banished.

Abbildung 1: Beispiel für Pathfinding im Videospiel Banished

Die verschiedenen Spielgenres unterscheiden sich nicht nur im Gesamtkonzept, sondern benutzen auch unterschiedliche Arten von Pathfinding Algorithmen. Echtzeit-Strategiespiele basieren meist auf zweidimensionalen Karten mit Kachelmuster, auf denen sich die Truppen bewegen können. Hierbei besteht die größte Schwierigkeit darin, dass sich potentielle Wege dynamisch verändern können. Ein Beispiel für dynamische Wege entsteht, wenn Passagen durch Truppen versperrt werden und keine Möglichkeit zum Passieren mehr existiert oder Veränderungen innerhalb der Spielumgebung eintreten. Ganz andere Schwierigkeiten gibt es bei Ego-Shootern. Bei diesen ist die Karte nicht zweidimensional, sondern sie hat eine dritte Dimension und kann deshalb nicht in Kacheln aufgeteilt werde. Häufig wird in diesen Fällen Pathfinding mit künstlicher Intelligenz verknüpft, weil sich diese an gesetzten Wegpunkten orientieren kann.

“Gutes” Pathfinding ist fast immer mit hoher Komplexität verbunden und muss auf jedes neue Szenario angepasst werden. Häufig verbraucht das Pathfinding vor allem in älteren Spielen, wie dem Echtzeit-Strategiespiel Age of Empires einen Großteil der gesamten CPU-Leistung während des Spielens.

Aus diesem Grund werden große Anstrengungen unternommen, um Pathfinding-Algorithmen zu optimieren. Bis jetzt erarbeitete Lösungskonzepte sind

beispielsweise Vorberechnungen und vorteilhafte Annahmen, die den einzigen Weg über eine Schlucht als zwangsläufige Route frühzeitig erkennen können.

Gerade jetzt, wo Künstliche Intelligenz immer mehr Einzug in Videospielen erhält, lässt es sich auch einfacher mit Pathfinding verknüpfen, um das Navigieren oder die Orientierung von Einheiten zu optimieren und die Leistung zu verbessern.

2.1.2 Algorithmen

2.1.2.1 A*-Algorithmus

Einer der am häufigsten verwendeten Algorithmen zum Pathfinding ist der A*-Algorithmus. Dieser Algorithmus interpretiert die Umgebung oder die Karte als Graph und verwendet zur Berechnung innerhalb der Bewertungsfunktion Heuristiken (vgl. Swift, N. 2017). Als Grundlage müssen dem Algorithmus Start- und Zielpunkt bekannt sein. Wenn dies der Fall ist, erhält jedes Feld vom Startpunkt ausgehend einen Wert, der proportional zur zurückgelegten Entfernung steigt (vgl. Lester, P. 2005). Der optimale Weg ist derjenige, bei dem das Zielfeld mit dem geringsten Wert erreicht wird (vgl. Swift, N. 2017). So können außerdem Zeitverluste oder überwindbare Hindernisse leicht eingerechnet werden.

Der A*-Algorithmus wird häufig in Videospielen oder bei der Navigation von Routenplanern verwendet. Der Luftlinienabstand von jedem Punkt zum Ziel wird standardmäßig als Heuristik verwendet. Bei Aufgabenstellungen, bei denen es keinen Abstand oder eine Luftlinie zum Ziel gibt, wie zum Beispiel beim 15er Puzzle (Abbildung 2) oder beim Damenproblem (Abbildung 3), dessen Ziel es ist, einen finalen Zustand zu erreichen, kann als Heuristik die Anzahl der falsch platzierten Steine verwendet werden.

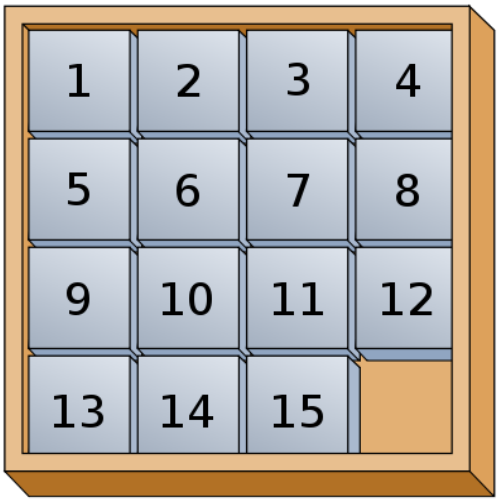


Abbildung 2 - Gelöstes 15er Puzzle

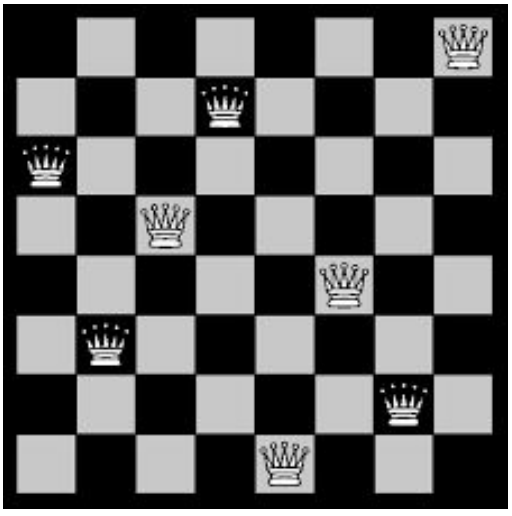
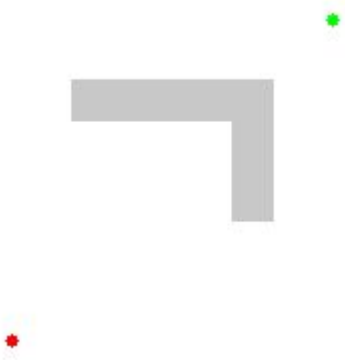


Abbildung 3 - Gelöstes Damenproblem

2.1.2.2 Dijkstra-Algorithmus

Fehlt einem die Heuristik komplett, um den Bewegungsaufwand zwischen zwei Punkten abzuschätzen, kann man statt des A*-Algorithmus den Algorithmus von Dijkstra verwenden.

In den Abbildungen 4 und 5 kann man den Unterschied zwischen den Operationsgeschwindigkeiten der beiden Algorithmen deutlich erkennen. Durch die Heuristik werden Punkte näher am Ziel präferiert und dadurch eine schnellere Lösung ermöglicht.



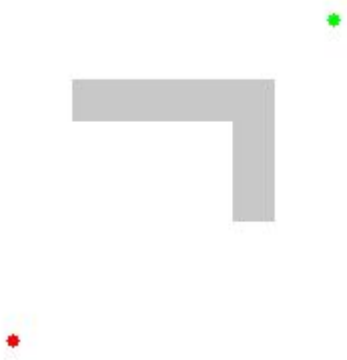


Abbildung 4 - Progression des Dijkstra Algorithmus | *Abbildung 5 - Progression des A* Algorithmus*

Beim Dijkstra-Algorithmus ist das Prinzip, immer derjenigen Kante zu folgen, die den kürzesten Weg zurück zum Startpunkt enthält (vgl. Vaidehi, J. 2017) . Die anderen Kanten werden erst nach Betrachtung aller kürzesten Streckenabschnitte beachtet, was gewährleistet, dass kein kürzerer Weg zum aktuellen Punkt existieren kann. Die berechneten Distanzen zwischen den aktuellen Punkten und dem Start ändern sich nicht mehr. Berechnet man die Distanz des Endpunktes hat man somit den optimalen Weg gefunden (vgl. Abiy, T., Khim, J., Pang, H., Ross, E., Williams, C. 2018). Man unterscheidet zwischen den Vorgehen “single-pair shortest path”, bei dem so lange berechnet wird, bis die Distanz des Endpunktes gefunden wurde und “single-source shortest path”, bei dem die Distanz aller Punkte zum Start bekannt sind, bevor aufgehört wird zu berechnen (vgl. Kevin Drumm 2016).

Ein sehr prominentes Beispiel sind Routenplaner, bei denen dieser Algorithmus angewendet werden kann. Das Verkehrswegenetz bildet hier den Graphen, indem es in Knotenpunkte aufgeteilt wird.

2.1.2.3 Heuristiken beim A*-Algorithmus

Heuristiken werden in Algorithmen verwendet, um sie schneller und effizienter zu gestalten, indem dabei auf Genauigkeit für erhöhte Geschwindigkeit verzichtet wird (vgl. Kenny, V., Nathal, M. & Saldana, S. 2014).

Heuristiken werden im A*-Algorithmus standardmäßig dafür benutzt, die geschätzte minimale Entfernung von einem Punkt n zum Ziel zu berechnen, um die Performance zu erhöhen. Es ist wichtig eine gute Heuristikfunktion zu wählen, um den Algorithmus zu verbessern und nicht zu verschlechtern.
Mit Heuristiken lässt sich die Schnelligkeit und Genauigkeit des Algorithmus einstellen. Dabei sind beide voneinander abhängig. Je genauer der Algorithmus arbeiten soll, desto langsamer wird er und umgekehrt (vgl. Amit 2019).

Oft gibt es Situationen in Videospielen, bei denen nicht immer der optimale Weg gefunden werden muss, sondern nur einer, welcher nahe genug dran ist. Von einem schnelleren Algorithmus wird immer profitiert, also muss ein Kompromiss gefunden werden, welcher ausreichend Genauigkeit und dadurch die schnellstmögliche Variante bietet.

Die Entscheidung zwischen Genauigkeit und Geschwindigkeit muss nicht statisch sein, sondern kann dynamisch im Verlauf des Spiels oder der Anwendung geändert werden.

Die Heuristik $h(n)$ in jedem Punkt n beeinflusst zusammen mit der Distanz zum Startpunkt $g(n)$, die gesamten Kosten $f(n)$ des Punktes. Die Formel für die Kosten jedes Punktes n ist $f(n) = g(n) + h(n)$ (vgl. Amit 2019).

Je nach Heuristik verhält sich der A*-Algorithmus anders. Je kleiner die Heuristik, desto langsamer, aber genauer wird der A*-Algorithmus. Bei einem Wert von $h(n) = 0$ spielt nur $g(n)$ eine Rolle und es entsteht ein exaktes Replikat des Algorithmus von Dijkstra. Wird die Heuristik größer so wird der Algorithmus schneller, aber dafür umso ungenauer. Haben wir das andere Extremum und $h(n)$ spielt ausschließlich eine Rolle wird der A*-Algorithmus zu “Greedy Best-First-Search” (vgl. Amit 2019).

Möchte man nun innerhalb des Spiels dynamisch den Algorithmus verändern, muss man lediglich die Heuristik verändern. Wichtig dabei bleibt nur, dass die Skalierung von der Heuristik gleichbleibend mit der Einheit der Distanz zum Startpunkt bleibt. Wenn beispielsweise die Distanz zum Start in Metern gemessen wird, aber die Heuristik in Zeit, dann wird der Algorithmus ungenaue Ergebnisse erzielen oder langsamer agieren als er eigentlich könnte, weil der Algorithmus beides gleich gewichtet, obwohl die Einheiten sich voneinander unterscheiden und anders betrachtet werden müssten (vgl. Amit 2019).

2.2 3D-Objekte

2.2.1 Meshes in 3D Engines

In Videospielen sind alle Objekte in einer dreidimensionalen Welt durch ihre Meshes definiert. Meshes bestehen aus vielen sogenannten Triangles oder Dreiecken, welche durch einen Shader aufgenommen werden. Dieser erzeugt aus den aufgenommenen Informationen jeden Frame das angezeigte Bild (Technologies, U. 2019).

Dadurch, dass alles aus Dreiecken besteht, gibt es keine Objekte mit Rundungen, sondern alles was einen Kreis, eine Kugel oder andere Rundungen beinhaltet, besteht im Mesh eigentlich aus vielen Ecken. Genau, wie ein Kreis in Wirklichkeit ein Unendlicheck ist, werden runde Objekte in Spielengines aus vielen Ecken erstellt, die eine Rundung vortäuschen.

Ein Mesh beinhaltet jede dieser Ecken als Vertex in einer Liste (Unity Technologies 2019). Für mein Projekt ist dies von Vorteil, weil ein Pathfinding-Algorithmus ein Netz aus Punkten und deren Verbindungen zueinander braucht, um einen optimalen Weg zu finden. Es kann

einfach das Mesh als Netz genutzt werden, sodass auf den “runden” Objekten keinerlei Punkte zusätzlich generiert werden müssen.

Es soll als Testobjekt eine Kugel benutzt werden, deren Vertices durch Zufall verzerrt werden. Hierbei gibt mehrere Möglichkeiten, die Kugeln je nach Verwendungszweck zu generieren.

Die meist verwendete Kugel ist die sogenannte UV-Sphere, weil sie leicht zu texturieren ist. Der Nachteil der UV-Sphere ist, dass ihre Vertices, im Vergleich zu den anderen Varianten am wenigsten genau verteilt sind.
Neben der UV-Sphere gibt es noch die Spherified-Cube- und Normalized-Cube-Variante. Beide entstehen aus einem Würfel, der immer wieder unterteilt wird, bis das Objekt rund erscheint (vgl. Cajaraville, O. S. 2015).

Das Icosahedron besteht, anders als die anderen Kugeln, aus gleichseitigen Dreiecken und bildet am ehesten den Körper einer Kugel ab. Durch die Gleichseitigkeit ist sie aber auch am wenigsten flexibel, was die Anzahl der Vertices und Triangles betrifft (vgl. Kahler, A. 2009). Diese Variante ist dennoch am effektivsten beim Pathfinding und Verzerren. Hierbei ist der gleiche Abstand zwischen den Vertices und die gleiche Anzahl an Verbindungen und somit Nachbarn eines Vertex besonders hilfreich.

Das Icosahedron oder Icosahedral, je nach Anzahl der Vertices, ist nicht das Einzige Polyhedral mit der Eigenschaft der Gleichseitigkeit, aber es ist das am einfachsten durch mathematische Formeln in einer Schleife erstellbare Polyhedral und wird deshalb als einziges in Videospielen verwendet. Theoretisch könnte auch jeder andere kugelförmige Körper aus der Liste der geodesischen oder Golberg Polyhedralen funktionieren, die Implementierung aber deutlich verkomplizieren.

2.2.2 Mapping

Als Mapping bezeichnet man die Analyse von mehrdimensionalen statischen Ereignissen und das Skizzieren einer Map mit den Ergebnissen (Thrun, S. 2000). Innerhalb des Projektes bezieht sich Mapping auf die Analyse der Vertices eines dreidimensionalen Objektes, je nach Lage und Situation und das Erstellen einer Bewertungs-Map als Ergebnis. Ein neuronales Netzwerk übernimmt hier die Aufgabe des Mappings, um die Heuristik des Pathfinding-Algorithmus in jedem Punkt zu generieren. Dabei ergibt sich aus der Bewertung des Punktes und der optimalen Entfernung zum Ziel bzw. der direkten Distanz zum Ziel die Heuristik.

2.2.3 Knotenkonstrukte

Wenn von Knotenkonstrukten innerhalb des Projektes die Rede ist, dann handelt es sich um eine Datenstruktur. Bei dieser wird ein Netz aus räumlich angeordneten Knotenpunkten gespannt, die über ihre Positionsvektoren einzigartig definiert sind und über die Verbindungen zu ihren Nachbarknoten abgespeichert werden. Eine vergleichbare Datenstruktur ist die Queue. Der Unterschied ist, dass jeder Knotenpunkt nicht zwangsweise zwei Nachbarn besitzt, sondern mehrere haben kann. Zusätzlich hat ein Knotenkonstrukt die

Eigenschaft, dass es durch ihre Positionsvektoren innerhalb eines Koordinatensystems dargestellt werden kann. Ein umgewandeltes Mesh in ein Knotenkonstrukt innerhalb einer Engine würde wie das wireframe des Meshes aussehen.

2.3 Neuronale Netzwerke

2.3.1 Reinforcement Learning

Reinforcement Learning ist eine Art der Schulung für Neuronale Netzwerke, bei denen ein Agent selbstständig eine Strategie erlernt (vgl. Mayer, D., Melegari, A. , Varone, M. 2017). Das Prinzip, mit dem der Agent lernt, ist ein Belohnungsprinzip. Er versucht, seine Strategie nach maximalen Erhalten von Belohnungen zu richten. Dabei wird dem Agenten nicht bei jeder Aktion die beste Alternative gezeigt, sondern es werden zu bestimmten Zeitpunkten Belohnungen, die auch negative sein können ausgeteilt. Jedem Zustand oder jeder Aktion wird so eine Nutzenfunktion anhand der Belohnungen approximiert, die deren Wert beschreibt (Simonini, T. 2018).

Es gibt zum Erlernen der Strategie des Agenten verschiedene Algorithmen, wie zum Beispiel die sehr erfolgreichen Monte-Carlo-Methoden oder Temporal Difference Learning (Simonini, T. 2018).

Wenn dem Agenten nur wenig Aktionsraum oder eine geringe Variabilität der Zustände zur Verfügung steht, reichen Tabellen, deren Felder anhand der erhaltenen Belohnungen aktualisiert werden. Gibt es aber einen großen Aktionsraum, muss die Funktion approximiert werden, wozu sich die Fourierreihe oder ein neuronales Netz eignen (Simonini, T. 2018).

Wenn Agenten an komplexen dreidimensionalen Objekten lernen sollen, ist zwar der Aktionsraum gering, weil pro Vertex immer nur eine einzige Bewertung stattfindet, aber die Variabilität der Zustände sehr groß. Es empfiehlt sich daher, neuronale Netzwerke statt Tabellen zu verwenden.

2.3.2 Deep Q-Learning

Die Deep Q-Learning-Strategie ist eine Art von Reinforcement Learning, welche es möglich macht, die neuronalen Netzwerke von Agenten, die in Umgebungen mit diskreten Aktionsbereichen arbeiten, zu trainieren (vgl. ADL 2018).
Ein diskreter Aktionsraum bezieht sich auf Aktionen, die gut definiert sind, z.B. Bewegungen nach links oder rechts, nach oben oder unten.

Das Ziel von Q-Learning ist es, die sogenannte Action-Value-Funktion $Q(s,a)$ zu lösen (vgl. Oppermann, A. 2018). Die Funktion Q gibt die Belohnung zurück, welche als “Qualität” der ausgeführten Aktion in einem bestimmten Zustand steht. Wenn der Agent das Ergebnis von $Q(s,a)$ ermittelt hat, kann das gegebene Ziel als gelöst betrachtete werden. Der Grund dafür ist, dass der Agent jetzt in der Lage ist, sich immer entsprechend zu verhalten, weil er die Qualität für jede mögliche Aktion in jedem Zustand ermitteln kann (vgl. ADL 2018).

Um die Action-Value-Funktion lösen zu können, muss der Agent trainiert werden bis die erhaltenen Belohnungen nahe genug am Optimum sind. Der Unterschied zu reinem Q-Learning ist, dass Deep Q-Learning ein Mischung aus zwei neuronalen Netzwerken benutzt, welche die Möglichkeit bieten, Aktionen mit unterschiedlichen Wahrscheinlichkeiten zu versehen statt sich auf eine zu fixieren (vgl. Oppermann, A 2018). Die unterschiedlichen Aktionen befinden sich auf unterschiedlichen “Ebenen”, welche mit unterschiedlichen Wahrscheinlichkeiten versehen werden. Am Ende wird die Ebene mit der höchsten Wahrscheinlichkeit als endgültig ausgewählt und die Aktion der Ebene als Ergebnis präsentiert (vgl. Oppermann, A 2018).

3. Methodik

3.1 Herangehensweise

Um das Projekt erfolgreich durchführen zu können, ist ein Konzept bestehend aus mehreren Meilensteinen für die Planung und Durchführung des Praxisprojektes erstellt worden.
Insgesamt besteht der lineare Arbeitsplan aus den folgenden sechs Meilensteinen:

1. Preparation des Projektes
2. Pathfinding Algorithmus erstellen und anpassen
3. Neuronales Netzwerk erstellen/outsourcen und anpassen
4. Deep Q-Learning für Agenten erstellen
5. Training der Agenten
6. Erstellen der Demo

Diese lineare Struktur macht es möglich, systematisch die benötigten Arbeitsschritte abzuarbeiten ohne dabei die Ordnung im Projekt zu verlieren. Die Meilensteine sind so angeordnet, dass nur das Endergebnis aus dem vorherigen Arbeitsschritt in den Nächsten einfließt. Dadurch ist die Konzentration auf genau ein Thema bzw. einen Arbeitsschritt sichergestellt.

3.2 Versions-Management

Im Projekt kommt ein auf dem Git-Protokoll basierendes Versionskontrollsystem zum Einsatz. Innerhalb dieses Systems lassen sich während der Arbeit Veränderungen durch sogenannte Commits speichern. Zu jeder Zeit kann der Arbeitsfortschritt zu einem beliebigen Commit zurückgesetzt oder damit verglichen werden. Voraussetzung für die Nutzung eines solchen Systems ist, regelmäßiges Registrieren der Veränderungen über besagte Commits mit sinnvollen Betitelungen. Zusätzlich bietet dieses System das Arbeiten an unterschiedlichen Verzweigungen sogenannten Branches. Durch eine Strukturierung von Arbeitsschritten oder Code-Abschnitten auf unterschiedlichen Branches, ist die Zusammensetzung des Projektes leichter zu überschauen und zu verstehen.

3.3 Feedback-Methode

Neben der Strukturierung des Projektes wird die Feedback-Methode verwendet, um Fehler frühzeitig zu erkennen. Bei dieser Methode finden entlang des Bearbeitungsfortschritts regelmäßig kollegiale Beratungen statt. Diese dienen dazu, zusätzlich Expertise oder Ratschläge einzuholen und damit den Arbeitsaufwand möglichst gering und fehlerfrei zu halten. Die Frequenz, in der der Fortschritt vorgestellt wird, ist abhängig von dem Meilensteinkonzept und davon in welcher Geschwindigkeit im Projekt Fortschritte erzielt werden.

3.4 Evaluation der Algorithmen

Für die Evaluation der Algorithmen werden die beiden Methoden Analyse und Vergleich verwendet. Am Ende dieser Evaluation soll jeweils ein Algorithmus im Bereich Pathfinding und einmal ein Learning-Algorithmus für das Projekt ausgesucht werden.

Die Algorithmen werden durch Vergleiche miteinander in bestimmten Kriterien evaluiert. Der Algorithmus, der alle Kriterien am besten erfüllt, wird für das Projekt ausgewählt. Damit die Algorithmen miteinander verglichen werden können, müssen sie auf die überlegten Vergleichskriterien analysiert werden.

3.4.1 Analyse

3.4.1.1 Pathfinding-Algorithmen

Es gibt eine sehr große Anzahl an Pathfinding-Algorithmen, welche in bestimmten Situationen jeweils andere Vor- und Nachteile haben. Um für das Projekt den richtigen Algorithmus auszuwählen, muss zuerst klar sein, was genau mit dem Algorithmus überhaupt erreicht werden soll.
Aus dieser Überlegung sind zwei Arten von Analysekriterien entstanden. Ausschlusskriterien, die erfüllt werden müssen, damit der Algorithmus in Frage kommt und Vergleichskriterien, in denen die Algorithmen verglichen werden können.
Die wichtigsten Kriterien, mit denen die Pathfinding-Algorithmen analysiert werden sollen, lassen sich in folgende vier Punkte aufteilen:

1. Funktionalität mit Knotenkonstrukten
2. Beeinflussbarkeit durch Heuristiken
3. Effektivität
4. Komplexität

Der erste Punkt ist ein Ausschlusskriterium. Das Projekt kann nur funktionieren, wenn der Pathfinding-Algorithmus umgewandelt werden kann, sodass er auf einem Knotenkonstrukt funktioniert. Ist dies nicht gegeben, ist keine weitere Analyse notwendig, denn der Algorithmus ist aus der Evaluation ausgeschieden.

Das neuronale Netzwerk soll im Projekt die Knotenpunkte eines Knotenkonstruktes bewerten. Damit ein Pathfinding-Algorithmus mit diesen Bewertungen arbeiten kann, muss eine Verwendung von zusätzlichen Daten in jedem Punkt in Form von Heuristiken gegeben sein. Der zweite Punkt der Analysekriterien ist aus diesem Grund ebenfalls ein Ausschlusskriterium, auf das die Algorithmen analysiert werden.

Sind die beiden ersten Punkte erfüllt, kann der Algorithmus für das Projekt verwendet werden. Gibt es eine große Auswahl an Algorithmen für das Projekt, wird durch den Vergleich in den letzten beiden Kriterien diese Auswahl auf einen Algorithmus reduziert.

Die Effektivität eines Algorithmus wird durch eine Messung von Geschwindigkeit und Genauigkeit ermittelt. Das Verhältnis beider Werte beurteilt wie effektiv ein Algorithmus ist.
Die Geschwindigkeit ergibt sich daraus, für wie viele Knotenpunkte ein potentieller Weg berechnet werden muss, bevor das Ziel erreicht ist. Je näher die Anzahl dieser Knotenpunkte an der Gesamtlänge der berechneten Strecke vom Start- zum Zielpunkt liegt, desto höher ist die Geschwindigkeit des Algorithmus.
Die Genauigkeit des Algorithmus wird dadurch beschrieben, wie kurz die berechnete Strecke vom Start- zum Zielpunkt ist. In diesem Projekt gilt: Je näher diese Strecke an der Länge einer Luftlinie zwischen Start- und Zielpunkt liegt, desto größer ist die Genauigkeit. Diese Berechnung der Genauigkeit funktioniert nur in Situationen, wo die Knotenpunkte in einem mehrdimensionalen Raster zu jedem anderen Punkt eine Luftlinie besitzen, wie es bei einem Knotenkonstrukt der Fall ist.

Die Analyse zur Komplexität erfolgt, indem alle Funktionen und Einflussmöglichkeiten des Algorithmus betrachtet werden. Je mehr Einflussmöglichkeiten es gibt, desto komplexer ist der Algorithmus. Die geringste Komplexität hat ein Algorithmus, wenn er nur die Grundvoraussetzung erfüllt und keine weiteren Funktionen und Einflussmöglichkeiten hat.

3.4.1.2 Learning-Algorithmen

Auch bei den Algorithmen, welche für das Training der Agenten zuständig sind, gibt es viele Optionen aus denen eine Auswahl getroffen werden muss. Um die Algorithmen evaluieren zu können, müssen sie genau, wie die Pathfinding-Algorithmen mithilfe von Ausschlusskriterien und Vergleichskriterien analysiert werden. Dafür wurden folgende Kriterien als Faktoren verwendet:

1. Ergebnisorientiertes Lernen
2. Selbständiges Training
3. Lernen ohne Ergebnisvergleich
4. Komplexität

Das Prinzip des maschinellen Lernens ist es, aus einer Situation Wissen aus Erfahrung anzueignen. Um den ersten Punkt zu erfüllen, muss der Algorithmus aus diesem Wissen für jede ihm gegebene Situation ein Ergebnis liefern können. Ohne ein solches Ergebnis könnte keine der notwendigen Bewertungen stattfinden.

Auch stellt selbstständiges Lernen ein Schlüsselkriterium da. Bei mehreren tausend Trainingsläufen für ein ausreichendes Ergebnis müssen alle Algorithmen, die nur mithilfe von menschlicher Evaluation oder anderen Inputs funktionieren, ausgeschlossen werden. Bei einem Algorithmus ohne menschliche Hilfe ist die Lerngeschwindigkeit um ein Vielfaches höher. Damit die gesetzten Meilensteine eingehalten werden können, ist selbstständiges Lernen daher eine Notwendigkeit.

Learning-Algorithmen, die zum Lernen eine Methodik verwenden, die einen Ergebnisvergleich als Grundlage haben, werden durch den dritten Punkt ausgeschlossen. Da bei diesem Projekt kein perfektes Ergebnis für jede Situation vorliegt, kann auch kein Vergleich von Ergebnissen stattfinden.

Für das Projekt können nach Anwendung der ersten drei Ausschlusskriterien somit nur sehr spezielle Learning-Algorithmen eingesetzt werden. Eine abschließende Auswahl muss anhand eines Vergleiches der Komplexität getroffen werden.

Komplexität lässt sich in diesem Fall nicht, wie bei den Pathfinding-Algorithmen, durch zusätzliche Funktionen oder andere feste Werte festlegen. Das Kriterium ist weniger eine Analyse als eine Abschätzung, wie zeitaufwändig und kompliziert es ist, den Algorithmus zu implementieren. Ein wichtiger nicht messbarer Faktor hierbei ist die Verständlichkeit für den Anwender, die sich von Person zu Person unterscheidet. Man kann die Komplexität auch als persönliche Präferenz bezeichnen.

3.4.2 Vergleich

3.4.2.1 Pathfinding-Algorithmen

Die Punkte, in denen die Pathfinding Algorithmen verglichen werden, sind die beiden Vergleichskriterien Effektivität und Komplexität, in denen sie vorher analysiert wurden.

Im Punkt Effektivität schneidet ein Algorithmus im Vergleich besser ab, wenn die Geschwindigkeit höher und die Genauigkeit größer ist. Bei dem Vergleich liegt außerdem auf der Genauigkeit eine höhere Gewichtung, als auf der Geschwindigkeit. Grund dafür ist das Ziel des Projektes, die Geschwindigkeit zu erhöhen. Die Genauigkeit kann nicht erhöht werden und bleibt bestenfalls gleich.

Für die Komplexität der Algorithmen im Vergleich gilt: Je komplexer desto schlechter. Dieses Kriterium ist dann ausschlaggebend, wenn die Effektivität von zwei oder mehreren Algorithmen vergleichbar ist oder wenn sich der ausgewählte Algorithmus als so komplex erweist, dass sich Probleme für die Implementierung oder die Einhaltung des Meilensteinkonzeptes ergeben.

3.4.2.2 Learning-Algorithmen

Der Vergleich bei den Learning-Algorithmen lässt sich nur im Punkt Komplexität durchführen. Dieser ergibt durch die Abschätzung eher eine Präferenzliste als ein Ergebnis basierend auf einem Kriterium. Trotzdem gilt für den Vergleich der Learning-Algorithmen ebenfalls: Je komplexer desto schlechter. Dieser Vergleich ist notwendig damit Algorithmen, die zwar in der Theorie im Projekt funktionieren würden, aber in der Praxis zu viel Zeit beim Implementieren in Anspruch nehmen würden, nicht ausgewählt werden.

4. Durchführung

4.1 Preparation

4.1.1 Framework

Für die Durchführung des Projektes wurde eine Kombination aus der Unity Engine und TensorFlow benutzt, um das Rendering, sowie die Ablaufsteuerung innerhalb der Demo und das neuronale Netzwerk outzusourcen. Außerdem wurde Conda zum Erstellen und Verwalten eines Environments, zum Trainieren der Agenten, sowie das Unity ml-Agents Package, welches die Schnittstelle zwischen Unity und TensorFlow übernimmt, benutzt. Diese Frameworks ermöglichten es, die Arbeit auf das Schreiben der benötigten Logik für die Agenten, sowie die Abwandlung des A*-Algorithmus zu konzentrieren.

4.1.2 Datenkontrolle

Durch das Versionsmanagement des Projektes basierend auf dem Git-Protokoll eignete sich ein online Git-Repository zur Verwaltung und Datenkontrolle. Vor der Programmierung des Praxisteils wurde ein Repository beim Online-Anbieter GitHub erstellt und die benötigte Gitignore-Datei passend zu einem Unity Projekt hinzugefügt. Dieses war dafür zuständig, dass beim Hochladen der Projektdaten nur die notwendigen Dateien auf dem Server gespeichert wurden. Damit das Projekt auch lokal existierte, wurde die Client-Software GitHub-Desktop verwendet, um das Repository zu klonen. Neben den Commits, die über das Git-Protokoll funktionieren, konnten auch Pushes durchgeführt werden, um den Fortschritt ins Internet hochzuladen. Aus diesem Grund war das Projekt beim Einhalten von regelmäßigen Pushes gegen Datenverlust durch Hardwarefehler geschützt.

Nach den vorbereitenden Maßnahmen, konnte ein Unity Projekt innerhalb des geklonten Repositories erstellt und ins Internet gepusht werden.

4.2 Evaluation der Algorithmen

4.2.1 Pathfinding Algorithmus

Die Evaluation nach beschriebener Methode wurde zuerst am bereits bekannten Dijkstra Algorithmus durchgeführt.
Der Algorithmus konnte jegliche Art von Datenstruktur als Map verwenden und erfüllte somit das erste Auswahlkriterium.
Die zweite Bedingung (Beeinflussbarkeit durch Heuristiken) wurde vom Dijkstra-Algorithmus nicht erfüllt, dieser konnte aber durch leichte Modifizierungen ertüchtigt werden. Durch die Weiterentwicklung wurde der Dijkstra-Algorithmus zu einem A*-Algorithmus, welcher Heuristiken verwendete und somit auch das zweite Auswahlkriterium erfüllte.

Der A*-Algorithmus war durch seine Heuristiken variable in Effektivität, konnte eine höchstmögliche Genauigkeit erreichen und war deshalb gut für das Projekt geeignet. Durch die Heuristik war das Worst-Case-Szenario des Algorithmus immer noch gleich des Dijkstra-Algorithmus, hatte aber im Durchschnitt eine stark verbesserte Geschwindigkeit.
Durch weitere Recherche wurde erkannt, dass der A*-Algorithmus neben der hohen Effektivität sowohl eine hohe als auch eine sehr niedrige Komplexität, je nach Modifizierung haben konnte.
Weil der Algorithmus die beiden Ausschlusskriterien erfüllte und sehr gut bei den Vergleichskriterien abschnitt, wurde er für das Projekt ausgewählt.

4.2.2 Learning Algorithmus

Algorithmen zum maschinellen Lernen lassen sich in überwachte und unüberwachte Algorithmen aufteilen. Neben dieser Aufteilung gibt es noch die semi-überwachte und die reinforcement oder auch bestärkt maschinelle Lernmethode. Bevor ein spezifischer Algorithmus ausgewählt werden konnte, wurden die Gruppierungen mit der genannten Methodik evaluiert.
Dadurch, dass die Algorithmen laut des ersten Ausschlusskriteriums ergebnisorientiert lernen sollten, kamen unüberwachte Algorithmen, sowie halb überwachte für das Projekt nicht in Frage. Diese werden nur zum Entdecken versteckter Strukturen aus Daten benutzt, anstatt eindeutige Ergebnisse zu erzeugen.
Das zweite Ausschlusskriterium negiert alle Algorithmen, die nicht selbstständig lernen können. Imitation-Learning ist aus diesem Grund für dieses Projekt ausgeschlossen.
Die Entscheidung zwischen bestärktem Lernen und überwachtem Lernen wird durch das dritte Kriterium gefällt. Das Prinzip des überwachten Lernens ist es, dessen Ergebnis mit einem beabsichtigten Ergebnis zu vergleichen und daraus zu lernen.
Unter Beachtung aller Faktoren musste ein deep Reinforcement-Learning-Algorithmus für das Projekt ausgewählt werden.
Die Entscheidung für den deep Q-Learning-Algorithmus wurde durch das machine learning agents package von Unity getroffen und in diesem Projekt benutzt, weil dort die Agents mit diesem Algorithmus arbeiten. Dadurch war die Implementierung nicht schwierig, also die Komplexität mit der der Algorithmus verglichen werden musste, so gering wie möglich. Ein weiterer Vergleich war aus diesem Grund nicht mehr nötig.

4.3 Abwandlung des A*-Algorithmus

Als erstes wurde der Pathfinding-Algorithmus programmiert und auf die Anwendung so angepasst, dass er Vertices als Map annehmen und darauf Pathfinding betreiben konnte.

Im ersten Schritt musste eine Methode entwickelt werden, die ein Mesh in ein Knotennetz umwandelt und bei jedem Vertex alle Verbindungen zu den benachbarten Vertices beibehält und als potentiellen Weg definiert. Dadurch, dass mit einem Icosahedral gearbeitet wurde, hatte jeder Knoten exakt sechs Nachbarn und somit auch sechs Verbindungen.

Danach wurde ein einfacher A*-Algorithmus erstellt, der in einem zweidimensionalen Array funktioniert und sich auch über Diagonalen bewegen kann. Rechnet man alle Verbindungen

eines Knotens zusammen, kommt man bei dieser Variante auf acht. Um den Algorithmus auf das Projekt anwendbar zu machen, musste die Verbindungszahl lediglich um zwei reduziert und die Randknoten, welche im zweidimensionalen Array weniger Verbindungen haben, komplett ignoriert werden, weil es bei einer Kugel keinen Rand gibt.

Als letzten Schritt musste jedem Knoten im Knotennetz eine Variable, die für die Bewertung zuständig ist, hinzugefügt werden. Mit diesen Variablen konnte der Algorithmus in jedem Knotenpunkt die Heuristik für den Vertex berechnen und bildete gleichzeitig die Schnittstelle zum neuronalen Netzwerk.

In diesem Zustand war das Projekt in der Lage, Meshes mit bewerteten Knotenpunkten anzunehmen und mit einem A*-Algorithmus Pathfinding darauf zu betreiben.

4.4 Neuronales Netzwerk mit Deep Q-Learning Agenten

Nachdem ein auf die Anwendung angepasster, funktionierender A*-Algorithmus entwickelt worden war, wurde das Programmieren der Agenten, sowie die Schnittstelle zu einem neuronalen Netzwerk realisiert.

Für die neuronalen Netzwerke wurde das TensorFlow Framework, welches einen großen Teil der Arbeit übernehmen sollte, verwendet. Aufgrund fehlender Expertise in Python stellte sich TensorFlow, welches nur von Anwendungen aus Python benutzt werden konnte, als Hindernis heraus.

Um diese Hürde zu überbrücken, wurde das Unity ML-Agents Toolkit verwendet, welches sich zu diesem Zeitpunkt noch in der Beta-Phase, aber bereits zur freien Verwendung befand. Das Toolkit bot zusammen mit der Python-Distribution Anaconda die Möglichkeit, TensorFlow nun doch in diesem Projekt zu verwenden. Außerdem enthielt es bereits funktionierende deep Reinforcement-Learning nutzende Agenten. Das bedeutete, dass alle nötigen Funktionen vorlagen, um sich ausschließlich auf das Schreiben der Logik für die Agenten zu konzentrieren.

Damit die Agenten lernen konnten, die Vertices von Meshes richtig zu bewerten, musste dem neuronalen Netzwerk an jedem Vertex eine fixe Anzahl an Informationen als Observation bereitgestellt und eine Aktion als Bewertung entgegengenommen werden.

4.4.1 Agent Mk1

Bei ersten Versuchen, die Logik der Agenten zu programmieren, sollten die verzerrten Kugeln nicht nur einfach observiert und bewertet werden, sondern dieses sollte parallel zur Erstellung der Kugel erfolgen. Dabei musste das neuronale Netzwerk zusätzlich die Aufgabe übernehmen, die Kugeln zu erstellen und zu verzerren. Die Idee von Mk1 war es, zufällig Koordinaten im gewünschten Radius um einen Mittelpunkt zu erzeugen und sie nach ihrem Abstand zu den benachbarten Koordinaten zu bewerten. Auf Abbildung 6 lässt sich erkennen, dass nach 500.000 Trainingsdurchläufen zwar ein kugelförmiges Objekt entstanden war, aber den Agenten nicht abtrainiert werden konnte, Koordinaten

überwiegend in einer einzigen Richtung zu platzieren. Die abgebildete Kugel war “top-heavy”. Sie funktionierte durch die ungleichmäßige Punkteverteilung nicht als Map. Die Abstände der Koordinaten in der oberen Richtung waren zu gering, um die Punkte voneinander unterscheiden zu können.

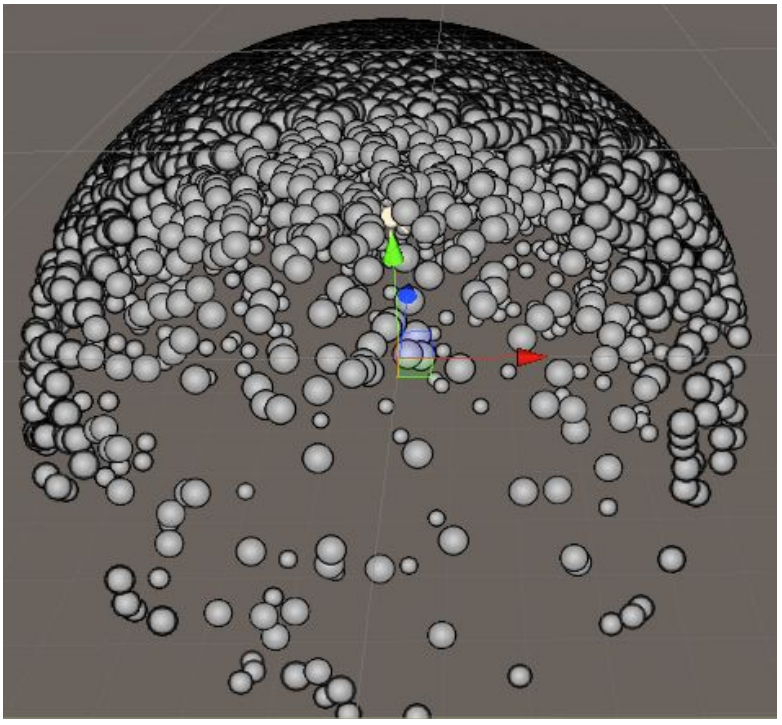
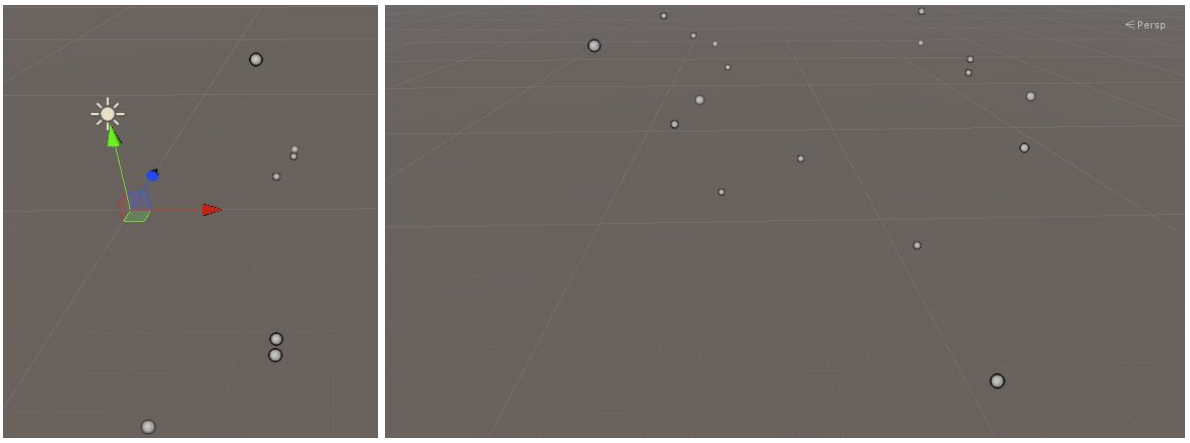


Abbildung 6 - Fehlschlag beim Generieren einer Kugel im 1. Versuch

4.4.2 Agent Mk2

Beim zweiten Ansatz wurde an der Idee, die Agenten die Kugel generieren zu lassen, festgehalten, aber die Logik verändert. Im Agenten Mk2 sollten die Koordinaten anders gesetzt werden. Statt die Koordinaten zufällig auszuwählen, wurden von einem Startpunkt aus Koordinaten in einem bestimmten Abstand in einer zufällig ausgewählten Richtung gesetzt, bis die gesamte Kugel von Punkten mit gleichmäßiger Verteilung geformt wurde. Wie in den Abbildungen 7 und 8 zusehen ist, konnte erneut nach 500.000 Testläufen keine für das Projekt nützliche Kugel erstellt werden. Bei diesem Konzept wurde mit Bestrafungen gearbeitet, wenn Koordinaten zu nah an Anderen gesetzt wurden. Das Erstellen der Kugel wurde abgebrochen, sobald die Bewertungen zu niedrig ausfielen. Die Agenten schafften es selten mehr als 15 Koordinaten zu setzen.



Abbildungen 7 & 8 - Fehlschlag beim Generieren einer Kugel im 2. Versuch

4.4.3 Agent Mk3

Bei der dritten Logik wurde immer noch davon ausgegangen, dass die Kugel für die Bewertung durch die Agenten erstellt werden musste. Der Ansatz in diesem Versuch hatte große Ähnlichkeit mit dem vorherigen Agenten Mk2. Der Unterschied war nur, dass die Koordinaten von einem Startpunkt aus nicht durch Richtungsvektoren sondern durch Winkel gesetzt wurden. Nach den ersten Versuchen wurde schnell klar, dass das Problem, welches bei der zweiten Logik auftrat, durch diese Veränderung nicht gelöst werden konnte. Die Abbildung 9 zeigt einen erneuten Fehlschlag beim Erstellen der Kugel.

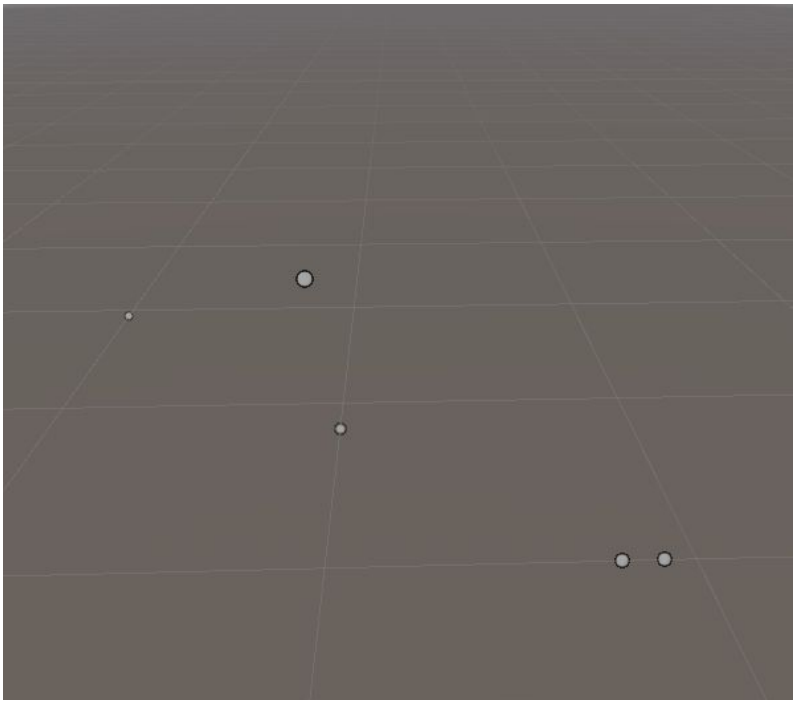


Abbildung 9 - Fehlschlag beim Generieren einer Kugel im 3. Versuch

Nach dem dritten Fehlschlag des Projektes wurde auf die Feedback-Methode zurückgegriffen. Durch das Darlegen meiner Probleme wurde das Projekt modifiziert und darauf verzichtet, dass die Agenten in der Lage sein mussten, eine Kugel selbst zu erstellen.

4.4.4 Agent Mk4

Diese Version des Agenten konzentrierte sich nur noch auf das Bewerten. Das Erstellen der Kugel wurde von einem statischen Algorithmus übernommen und mit Zufallswerten verzerrt. In den Abbildungen 10 und 11 ist das Ergebnis dieses Algorithmus in Form eines Meshes dargestellt. Als Observation erhielt das neuronale Netzwerk die Position des Start- und Zielpunktes, sowie die Position des zu bewertenden Vertex zusammen mit allen sechs anliegenden Vertices.

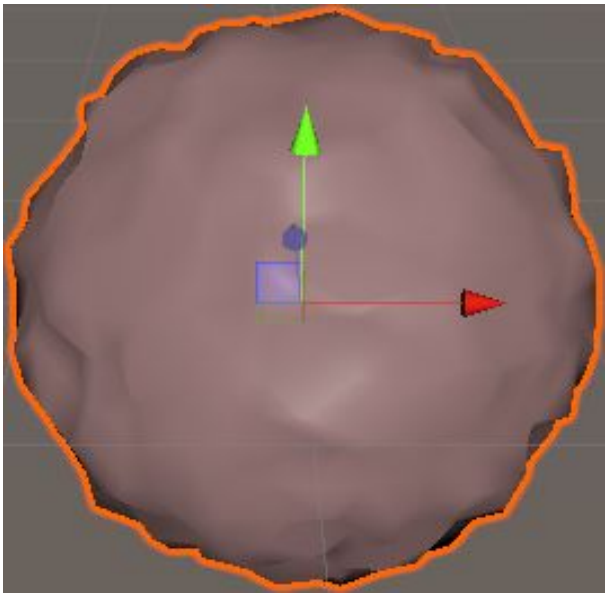


Abbildung 10 - Shaded-Kugel-Mesh

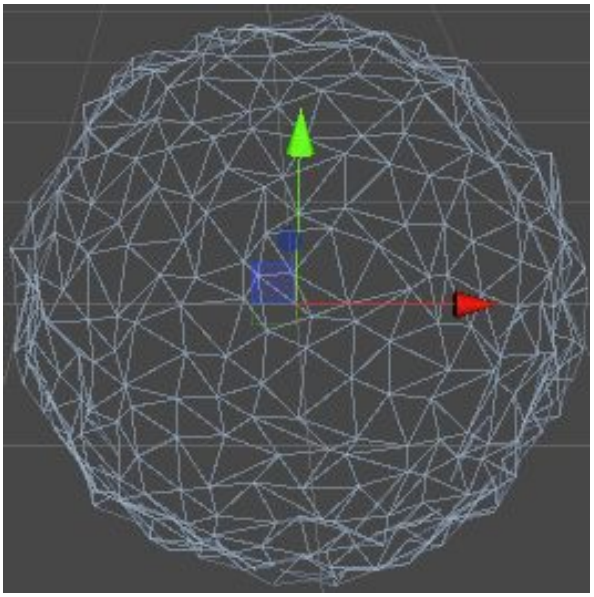


Abbildung 11 - Wireframe-Kugel-Mesh

Die Bewertungsfunktion für diesen Agenten gab nach jeder vollständig bewerteten Kugel eine Belohnung aus der Länge der Strecke durch den angepassten A*-Algorithmus und der Luftlinie vom Start- zum Zielpunkt zurück.
Mit der fertigen Logik waren die Agenten bereit zum Lernen und mussten nur noch über viele tausend Testläufe trainiert werden.

4.5 Training des neuronalen Netzwerkes

Um die Agenten zu trainieren, musste ein Environment in der Konsole mithilfe von Conda, welches zusammen mit Anaconda heruntergeladen und installiert wurde, erstellt werden. Innerhalb des neuen Environments mussten die machine-learning-Pakete aus dem Unity ML-Agents Toolkit, sowie alle benötigten Pakete für TensorFlow installiert werden.

Ab diesem Zeitpunkt konnten innerhalb des Environments die Trainingsphasen gestartet und neuronale Netzwerke mithilfe von TensorFlow erstellt werden.

Damit die Testphasen erfolgreiche Ergebnisse lieferten, musste in Unity alles vorbereitet und aufs Training anpasst werden. Es musste ein neues Learning Brain angelegt werden. Außerdem musste eingestellt werden, wie viele Informationen bei jeder Bewertung an das

neuronale Netzwerk gegeben und wie viele Aktionen getätigt werden sollten. Zusätzlich mussten sich die Agenten mit der programmierten Logik in der Szene befinden, sowie eine sogenannte Academy, welche als Schnittstelle zwischen Agent und TensorFlow agierte. Als weitere Besonderheit wurde ein selbstgeschriebener Mesh-Generator in der Szene benötigt, der in jedem neuen Trainingsdurchlauf ein neues Mesh erstellte.

Sobald alle nötigen Referenzen in der Szene erzeugt waren, konnte über die Konsole innerhalb des neu erstellten Environments mit Referenz zu einer Konfigurationsdatei, in der die Rahmenbedingungen der Tests festgelegt waren, die Trainingsphase gestartet werden.
Die ersten positiven Ergebnisse aus den Testphasen traten nach 10.000 Durchläufen auf. Nach 100.000 wurde die Trainingsphase gestoppt und das neuronale Netzwerk aufgrund der Ergebnisevaluation für trainiert befunden.

4.6 Erstellen der Demo

4.6.1 Visualisierung

Um das Ergebnis des Projektes darstellen zu können, wurde das Knotenkonstrukt als Mesh sichtbar gemacht. Außerdem sollte der Start- und Zielpunkt, sowie die verbindende Strecke durch eindeutige Markierungen auf dem Mesh visualisiert werden. Diese Markierungen sollten über einen Open-GL-Renderer in die Szene gezeichnet werden. Durch auftretende Fehler beim Anhalten der Bewertungsprozesse konnte keine Pause für das Visualisieren der Strecke eingelegt werden. Stattdessen wurde eine Anzeige mit einem Vergleich erstellt, die die Länge mit und ohne Bewertung des Algorithmus auf der vorherigen Kugel beschrieb.

4.6.2 Interface

Eine Veränderung der Blickrichtung auf das Mesh wurde durch ein Drücken und Bewegen der rechten Maustaste realisiert, die Kontrolle über einige Rahmenparameter durch ein Userinterface. Geplant war, die Größe und die Anzahl der Vertices vom Mesh über einen Reglerelement zu steuern. Außerdem sollte über ein Knopfelement die Start- und Zielposition, sowie das Mesh neu generiert werden. Dieses konnte ebenfalls aufgrund des “Nicht-Anhalten-Problems” nicht umgesetzt werden. Das Ergebnis des Projektes wurde in Zahlen gefasst im Interface angezeigt. Dazu gehören die Länge der berechneten Strecke als Genauigkeit, die Anzahl der benutzten Punkte für das Pathfinding als Geschwindigkeit und der Vergleich dieser Werte.

4.6.3 Zusammensetzung

Nachdem alle Teile des Projektes praktisch funktionsfähig waren, wurde das trainierte neuronale Netzwerk in die Szene fest eingebaut. Zusätzlich wurde das Projekt mit dem beschriebenen Interface ausgestattet. Zum Schluss wurde das Projekt zu einer ausführbaren Exe-Datei konvertiert, damit es ohne weitere Expertisen ausgeführt und verwendet werden kann.

5. Ergebnis

Als Hauptuntersuchungsschwerpunkt des Praxisprojektes wurde eine Arbeitsreihenfolge von drei Algorithmustypen, die abwechselnd ein Mesh erstellen und verzerren, Knotenpunkte bewerten und Pathfinding mit zufälligen Start- und Endpunkten betreiben, realisiert.
Durch ein Interface wurde ein Vergleich gezogen, welcher den Beweis erbrachte, dass der Einsatz von künstlicher Intelligenz in Kombination mit Pathfinding auf dreidimensionalen Objekten eine positive Auswirkung auf die Geschwindigkeit ohne Genauigkeitsverlust bei der Wegfindung haben kann.
Das Ziel, eine künstliche Intelligenz zu entwickeln, die Pathfinding-Algorithmen unterstützt, wurde erreicht.

5.1 Funktionalität

Auch wenn nicht alle, aus dem ursprünglich sehr ambitioniert angelegten Projektplan, hinterlegten Funktionalitäten vollumfänglich realisiert werden konnten, wurde, unter Einbeziehung von Modifikationen, ein funktionsfähiges Projekt erstellt, welches eine fundierte Aussage zum Untersuchungsauftrag ermöglicht hat.

5.1.1 Pathfinding

Es wurde erfolgreich ein A*-Algorithmus implementiert der alle Vorgaben erfüllt. Aus der Zielsetzung wurden folgende Punkte umgesetzt.

Ziel	Umsetzung
Der A*-Algorithmus sollte in der Lage sein, Wegpunkte von der KI anzunehmen.	Das Pathfinding wurde im Projekt anhand einer eigenen Datenstruktur umgesetzt. Der A*-Algorithmus konnte auf Knotenkonstrukten, die er vom Agenten des neuronalen Netzwerkes bekam, fungieren.
Der Algorithmus sollte auf unterschiedlichen Formen funktionieren.	Durch das Konzept, mit Knotenkonstrukten zu arbeiten, konnte der A*-Algorithmus auf jedem Mesh, egal welcher Form, einen Weg errechnen.
Eine Auswahl an Pathfinding Algorithmen sollte angeboten werden.	In dem Projekt wurde ein Vergleich mit dem Dijkstra Algorithmus gezogen, auch um den Unterschied in den Geschwindigkeiten mit und ohne Bewertung zu zeigen.

5.1.2 Neuronales Netzwerk mit Deep Q-Learning Agenten

Das Machine-Learning-Paket von Unity übernahm alle Aufgaben des neuronalen Netzwerkes und fungierte als Bindeglied im gesamten Projekt. TensorFlow hatte das Trainieren gänzlich übernommen und wurde erfolgreich angewendet.
Die Logik der Agenten durchlief einige Veränderungen und wurde, abweichend zu den Zielen, die zu Beginn des Projektes gestellt wurden, modifiziert.

Ziel	**Umsetzung**
Aus den Informationen des neuronalen Netzwerkes sollte ein Kugel-Mesh erstellt werden.	Aufgrund einer Empfehlung im Rahmen der Feedback-Methode wurde der Agent auf das Bewerten der Knotenpunkte reduziert. Der vierte und finale Agent übernahm nicht mehr das Erstellen der Kugel. Es wurde ein neuer statischer Algorithmus mit der Aufgabe, eine Kugel zu generieren, programmiert. Dieser Algorithmus wurde nicht von der KI beeinflusst.
Die Agenten sollten das erstellte Kugel-Mesh verzerren.	Das Verzerren wurde ebenfalls auf den neuen Algorithmus, der auch für das Erstellen zuständig war, abgegeben.
Jeder Knotenpunkt sollte von einem Agenten vor dem Pathfinding bewertet werden.	Die Agenten waren alle in der Lage, jedem Punkt aus folgenden Observationen eine Bewertung zu geben: 1. Lage von Start- und Zielpunkt 2. Lage des zu bewertenden Punktes 3. Lage aller Nachbarpunkte
Optional konnte das Projekt mit einer Auswahl von unterschiedlichen Lernständen des neuronalen Netzwerkes ausgestattet werden.	Dieses Kann-Ziel wurde nicht erreicht. Es ist nicht gelungen, eine Auswahl der Lernzustände in das Projekt einzubauen. Aufgrund die Rückschläge beim Erstellen der Agentenlogik, war nicht mehr ausreichend Zeit, um dieses Ziel realisieren.

5.1.3 Visualisierung und Interface

Die Zielsetzungen für das Interface und die Visualisierung wurden komplett verfehlt somit musste eine Alternativlösung für beide Punkte gefunden werden. Durch auftretende Fehler in der Academy, welche für das Management der Agenten zuständig war, konnte der Bewertungsprozess nicht pausiert werden. Es war also nicht genügend Zeit für das Einzeichnen der Wege auf der bearbeiteten Kugel vorhanden, bevor die Nächste neu erstellt wurde.
Das einzige Ziel, welches erfolgreich umgesetzt werden konnte, war die Visualisierung der verzerrten Kugeln. Das generierte Knotenkonstrukt wurde in der Szene als Mesh dargestellt. Für alle anderen Ziele wurden folgende Gegenmaßnahmen getroffen:

Ziel	**Gegenmaßnahme / Umsetzung**
Der berechnete Weg sollte auf dem generierten Mesh sichtbar gemacht werden.	Statt den Weg auf der Kugel einzuzeichnen, wurden die Positionen von Start- und Zielpunkt in einem Interface angegeben. Außerdem wurde die Länge des Weges mit den jeweiligen Pathfinding Varianten (A* und Dijkstra) und die Luftlinie angezeigt.
In einem Userinterface sollten die Radien und die Anzahl der Vertices der Kugel vom Benutzer frei definiert werden.	Es wurde auf integrierbare Elemente im Interface gänzlich verzichtet. Die Größe und Komplexität der Kugel hatte keinen Einfluss auf die Aussage des Projektes und wäre durch das “Nicht-Anhalten-Problem”, unnötig schwierig zu realisieren gewesen
Die Ausrichtung des angezeigten Kugel-Meshes sollte mit Input über die Maus verändert werden können.	Als Alternative zur Veränderung des Meshes in der Szene, wurde eine Veränderung des Blickwinkels auf die Kugel programmiert. Probleme, die entstanden, wenn sich während der Bewertung die Positionen der Vertices im Mesh verändern, konnten so umgangen werden.

5.2 Kritische Betrachtung der Arbeit

Im Durchschnitt ist die Bearbeitung positiv verlaufen. Für die Fehlschläge beim Programmieren der Agentenlogik und das daraus entstandene Zeitproblem kurz vor Ende konnten gute Alternativen gefunden werden.

Die Methoden für das Projekt wurden vorbereitend durchdacht, gut ausgewählt und vollumfänglich im Projekt erfolgreich angewendet. Das Versionsmanagement war bei den Rückschlägen eine große Hilfe und hat das schnelle Einleiten von Gegenmaßnahmen möglich gemacht. Durch die Feedback-Methode konnten viele Fehler umgangen und Veränderungen im Konzept vorgenommen werden. Die Analyse- und Vergleichsmethoden bildeten die Grundlage dafür, dass ein Basiskonzept im Projekt entstehen konnte.
Die Strukturierung des Projektes war ebenfalls sinnvoll und hat sich als gute Arbeitsunterstützung herausgestellt. Insbesondere war hilfreich, dass aus den beiden Themenbereichen, die in dem Projekt miteinander verknüpft wurden, in einem bereits ausreichend Expertise vorhanden war, sodass nur ein neuer Themenschwerpunkt erlernt werden musste.

Um die Fehlschläge im Projekt zu vermeiden, hätten einzelne unterstützende Methoden konsequenter angewendet werden müssen. Besonders die Feedback-Methode wurde sehr spät in Betracht gezogen und hätte regelmäßiger angewandt werden sollen.
Die Einschätzung des zeitlichen Arbeitsaufwandes der im Meilensteinplan fixierten einzelnen Arbeitsschritte, war beim Erstellen des Zeitplans zu optimistisch gewählt. Aus dieser Fehleinschätzung sind Probleme durch Zeitdruck und auftretende Stresssituationen beim Abschließen des Projektes entstanden.
Mit der Nutzung des, noch im Beta-Zustand befindlichen, Unity-Machine-Learning-Paketes, welches eine essentielle Aufgabe im Projekt übernommen hat, wurde ein gewisses Risiko eingegangen. Diese Nutzung kombiniert mit noch fehlender Expertise, führte am Ende zu weiteren Hindernissen. Als Folge konnten beim Arbeitsschritt Visualisierung schlussendlich die gesetzten Ziele nicht erreicht werden, so dass weitere kompensierende Gegenmaßnahmen ergriffen werden mussten.
Letztendlich konnte aber trotz aller Hindernisse eine fundierte Aussage zum Untersuchungsauftrag anhand des Praxisprojektes getroffen und die Arbeit erfolgreich beendet werden.

6. Quellenverzeichnis

6.1 Künstliche Intelligenz

ADL (2018) *An introduction to Q-Learning: reinforcement learning*. [Online]. 3 September 2018. freeCodeCamp.org. Available from: https://medium.freecodecamp.org/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc [Accessed: 22 February 2019].

Anon (n.d.) *neuronalenetze-en-zeta2-2col-dkrieselcom.pdf*. [Online]. Available from: http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf [Accessed: 1 March 2019].

com.prehensible (2013) *unity - Algorithmus zum Erstellen von Kugeln?* [Online]. 4 May 2013. Switch-Case. Available from: https://de.switch-case.com/53911949 [Accessed: 27 February 2019].

LeCun, Y., Bengio, Y. & Hinton, G. (2015) Deep learning. *Nature*. [Online] 521 (7553), 436–444. Available from: https://doi.org/10.1038/nature14539 [Accessed: 1 March 2019].

Mayer, D., Melegari, A. , Varone, M. (2017) What is Machine Learning? A definition - Expert System. [Online] 7 March 2017. Available from: https://www.expertsystem.com/machine-learning-definition/ [Accessed: 11 August 2019].

Oppermann, A. (2018) *Self Learning AI-Agents Part II: Deep Q-Learning*. [Online]. 28 October 2018. Towards Data Science. Available from: https://towardsdatascience.com/self-learning-ai-agents-part-ii-deep-q-learning-b5ac60c3f47 [Accessed: 1 March 2019].

Simonini, T. (2018) *An introduction to Reinforcement Learning*. [Online]. 31 March 2018. freeCodeCamp.org. Available from: https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419 [Accessed: 27 February 2019].

Thrun, S., Burgard, W. & Fox, D. (2000) A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. [Online]. 2000 San Francisco, CA, USA, IEEE. pp. 321–328. Available from: doi:10.1109/ROBOT.2000.844077 [Accessed: 1 March 2019].

6.2 Pathfinding

Abiy, T., Khim, J., Pang, H., Ross, E., Williams, C. (2018) Dijkstra's Shortest Path Algorithm | Brilliant Math & Science Wiki. [Online]. Available from: https://brilliant.org/wiki/dijkstras-short-path-finder/ [Accessed: 20 December 2018].

Amit (2019), Heuristics [Online]. 22 September 2019. Available from: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html [Accessed: 6 August 2019].

Definitions.net, STANDS4 LLC (2019) “pathfinding”. [Online]. Available from: https://www.definitions.net/definition/pathfinding [Accessed: 5 March 2019].

Ferguson, D., Likhachev, M. & Stentz, A. (n.d.) *A Guide to Heuristic-based Path Planning*. 10.

Ghallab, M. & Allard, D.G. (n.d.) *Ae — AN EFFICIENT NEAR ADMISSIBLE HEURISTIC SEARCH ALGORITHM*. 3.

Goldberg, A.V., Harrelson, C., Kaplan, H. & Werneck, R.F. (n.d.) *Efficient Point-to-Point Shortest Path Algorithms*. 12.

Harabor, D. (2009) *Clearance-based Pathfinding and Hierarchical Annotated A* Search | AiGameDev.com*. [Online]. 5 May 2009. AiGameDev.com. Available from: http://aigamedev.com/open/tutorials/clearance-based-pathfinding/ [Accessed: 20 December 2018].

Kevin Drumm (2016) Graph Data Structure 4. Dijkstra’s Shortest Path Algorithm [Online]. Available from: https://www.youtube.com/watch?v=pVfj6mxhdMw [Accessed: 20 December 2019].

Lester, P. (2005) A* Pathfinding for Beginners. 11.

Likhachev, M., Gordon, G. & Thrun, S. (2013) ARA*: Anytime A* with Provable Bounds on Sub-Optimality. *2013*. 8.

Qiao (n.d.) *PathFinding.js*. [Online]. Available from: https://qiao.github.io/PathFinding.js/visual/ [Accessed: 20 December 2018].

Swift, Nicolas (2017) Easy A* (star) Pathfinding. [Online]. 1 March 2017. Available from: https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2 [Accessed: 6 August 2019].

Technologies, U. (2018) *Unity - Manual: Navigation and Pathfinding*. [Online]. 2018. Available from: https://docs.unity3d.com/Manual/Navigation.html [Accessed: 20 December 2018].

Vaidehi, J. (2017) Finding The Shortest Path, With A Little Help From Dijkstra. [Online]. Available from: https://medium.com/basecs/finding-the-shortest-path-with-a-little-help-from-dijkstra-613149fbdc8e [Accessed: 20 December 2018].

6.3 Sonstige

Abdelbar, A.M. (2012) Alpha-Beta Pruning and Althöfer's Pathology-Free Negamax Algorithm. *Algorithms*. [Online] 5 (4), 521–528. Available from: doi:10.3390/a5040521.

Alan Mackworth (2013) *Lecture 9 | Search 6: Iterative Deepening (IDS) and IDA**. [Online]. Available from: https://www.youtube.com/watch?v=5LMXQ1NGHwU [Accessed: 20 December 2018].

Anon (n.d.) *A New Hashing Method With Application For Game Playing.pdf*. [Online]. Available from: https://minds.wisconsin.edu/bitstream/handle/1793/57624/TR88.pdf?sequence=1 [Accessed: 20 December 2018].

Baeldung (2017) *Introduction to Minimax Algorithm*. [Online]. 11 July 2017. Baeldung. Available from: https://www.baeldung.com/java-minimax-algorithm [Accessed: 20 December 2018].

Cajaraville, O. S. (2015) Four Ways to Create a Mesh for a Sphere - Oscar Sebio Cajaraville. [Online]. 7 December 2015. Available from: https://medium.com/game-dev-daily/four-ways-to-create-a-mesh-for-a-sphere-d7956b825db4 [Accessed: 11 July 2019].

Jain, R. (2017) Minimax Algorithm with Alpha-beta pruning. *HackerEarth Blog*. [Online]. Available from: https://www.hackerearth.com/blog/artificial-intelligence/minimax-algorithm-alpha-beta-pruning/ [Accessed: 20 December 2018].

Kahler, Andreas (2009) Creating an icosphere mesh in code. [Online]. Available from: http://blog.andreaskahler.com/2009/06/creating-icosphere-mesh-in-code.html [Accessed: 18 July 2019].

Kenny, V., Nathal, M. & Saldana, S. (2014) *Heuristic algorithms - optimization*. [Online]. 24 May 2014. Available from: https://optimization.mccormick.northwestern.edu/index.php/Heuristic_algorithms [Accessed: 20 December 2018].

Moura, L. (2009) *Heuristic Search*.

Technologies, U. (2019) Unity - Scripting API: Mesh [Online]. 2018. Available from: https://docs.unity3d.com/ScriptReference/Mesh.html [Accessed: 20 February 2018].

6.4 Bildquellen

Alle Bilder ohne angegebene Quelle sind Abbildungen aus dem Praxisprojekt und wurden selber erstellt.

Shining Rock Software (2013) [Online]. Available from: http://www.shiningrocksoftware.com/wp-content/uploads/2013/11/LongPath-1024x576.jpg [Accessed: 20 August 2019].

Schildberger, G. (2016) 15_puzzle. [Online] Available from: https://rosettacode.org/mw/images/7/79/15_puzzle.png [Accessed: 20 August 2019].

(n.a) (n.d.) Damenproblem. [Online]. Available from: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSdThHEwesd4ZBnDOA-lqt0ag2lOw8wc1waE_vYorjxAt0R7X5y [Accessed: 20 August 2019].

Subh83 (2011) Dijkstras progress animation [Online]. Available from: https://commons.wikimedia.org/w/index.php?curid=14916903 [Accessed: 20 August 2019].

Subh83 (2011) Astar progress animation [Online]. Available from: https://commons.wikimedia.org/w/index.php?curid=14916867 [Accessed: 20 August 2019].