Künstliche Intelligenz in Videospielen

Inhaltsverzeichnis

Einleitung

Videospiele haben schon seit Langem die Technologie gefördert. Die Interaktion zwischen Mensch und Maschine ist die Grundlage für ein Spiel, wobei die Entwickler dabei ständig die Grenzen der verfügbaren Hardware herausfordern. Physik- und Grafik-Simulationen können somit immer weiter verbessert werden. Spiele können dadurch auch in Bereichen wie Wirtschaft und Medizin hilfreich sein. Doch welche Rolle haben Künstliche Intelligenzen in Videospielen? KI‘s haben eine lange Tradition in der Videospiel-Industrie. Die KI‘s von denen man hierbei spricht sind jedoch streng genommen nicht wirklich KI‘s. Zu den Techniken, die bei der KI-Spieleprogrammierung verwendet werden, gehören vor allem Behaviour-Trees und Pathfinding. Richtige KI-Ansätze finden sich nur vereinzelt in Videospielen wieder. Die folgenden Themen erläutern den Begriff der KI in Videospielen und stellen mögliche Szenarien dar, die für eine Nutzung von KI‘s in Bereich von Videospielen sprechen. Außerdem werden im Folgenden KI‘s vorgestellt, die schon heute die Videospiel-Industrie verändern könnten.

1. KI als Non-Player-Character

In Videospielen kommt – anders als die meisten Spieler denken – keine KI zum Einsatz, wie sie in autonom fahrenden Fahrzeugen oder in anderen intelligenten Systemen verwendet wird. In fahrerlosen Autos oder z.B. in Googles Bildbearbeitungs-KI „Google Creatism“ wird eine selbstlernende Software verwendet, die sich immer weiter entwickelt und selbstständig neue Vorgehensweisen erlernt [1]. Die „wissenschaftliche“ künstliche Intelligenz ist ein System, das seine Umgebung analysiert und an Hand dieser Wahrnehmung gezielt Entscheidungen trifft, um bestimmte Ziele zu erreichen. Ziele wie z.B. das menschliche Gehirn nachzubilden oder das Sprachverstehen. Bei Gegner-KI‘s wie „Deep Blue“ oder „AlphaGo“ gilt genau diese Definition (s. 2.1. & 2.2.). Würde man eine solche künstliche Intelligenz in Videospielen verwenden, würde diese das Spiel wahrscheinlich unspielbar machen. KI im Sinne des Videospiel-Designs bezieht sich lediglich auf das geregelte und vordefinierte Verhalten von Non-Player-Characters (kurz: NPC) [2][3][4].

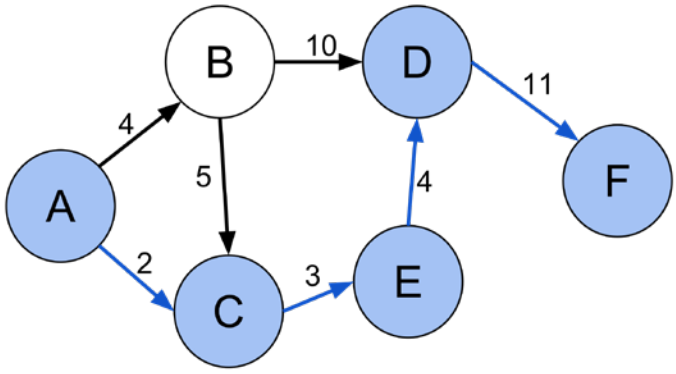
In Computerspielen kommen verschiedene Werkzeuge zum Einsatz, die eine künstliche Intelligenz der NPCs vortäuschen und dem Spieler die Illusion vermitteln, als würde er mit intelligenten Charakteren, Kreaturen oder Tieren interagieren. Als Vorreiter hierfür gilt das Computerspiel „Space Invaders“, das 1978 erschienen ist. Durch im Vorfeld erstellte, zufällige Bewegungsmuster werden dem Spieler intelligente Aliens simuliert. Auch das weltweit bekannte und 1980 erschienene Spiel „Pacman“ täuscht dem Spieler eine künstliche Intelligenz vor, in der die vier Geister wirken, als hätten sie einzelne Persönlichkeitsmerkmale [5][6][7].

Für Videospiele gibt es eine Reihe an Möglichkeiten, KI-ähnliche Muster und Algorithmen einzubauen. Im Folgenden werden der „Pathfinding-Algorithmus“, die „Finite State Machine“ sowie „Entscheidungs- und Verhaltensbäume (Behaviour Trees)“ genauer betrachtet.

1.1. Pathfinding

Beim Pathfinding, zu deutsch „Wegfindung“, handelt es sich um eine algorithmusgestützte Suche nach dem bzw. den optimalen Wegen. Dabei wird die Strecke zwischen einem gegebenen Startpunkt und einem oder mehreren Zielorten berechnet – wie bei Google Maps oder bei einem Satellitennavigationssystem im Auto. In der Praxis ist die schnellste Route nicht immer die Luftlinie. Hindernisse wie z.B. Wälder oder Seen, die den direkten Weg versperren, werden bei der Berechnung mit berücksichtigt. In der Historie der Computerspiele reicht das Prinzip des Pathfindings bis zum Spieleklassiker *Pacman* zurück, bei dem einfache Wegfindungs-Algorithmen dafür zuständig waren, den Weg der Gespenstergegner durch das Labyrinth zu berechnen. In Ego-Shootern dient das Pathfinding der Orientierung von NPCs im dreidimensionalen Raum, wobei hierbei oft mehrere Wegpunkte verwendet werden, die der NPC nacheinander abarbeitet. In Echzeit-Strategiespielen, die meist zweidimensionale Karten verwenden, wird die Spielfläche in einzelne Kacheln unterteilt. Eine Schwierigkeit stellt oft das Durchqueren von engen Passagen wie etwa einer kleinen Brücke für eine große Anzahl von Einheiten dar. Gutes Pathfinding ist meist mit einer hohen Komplexität verbunden. Beim 1999 erschienenen *Age of Empires 2* wurden etwa 60 bis 70 % der CPU-Leistung alleine für das Berechnen der Wege der eigenen und gegnerischen Einheiten verbraucht [8].

Eine häufig verwendete Methode ist der A*-Algorithmus. Bei diesem Algorithmus müssen Start- und Zielpunkt vorher definiert sein. Anschließend wird jedem Feld – ausgehend vom Startpunkt – ein Wert zugeteilt, der proportional zur Entfernung ansteigt. Der idealste Weg ist nun der mit dem geringsten Gesamtwert.



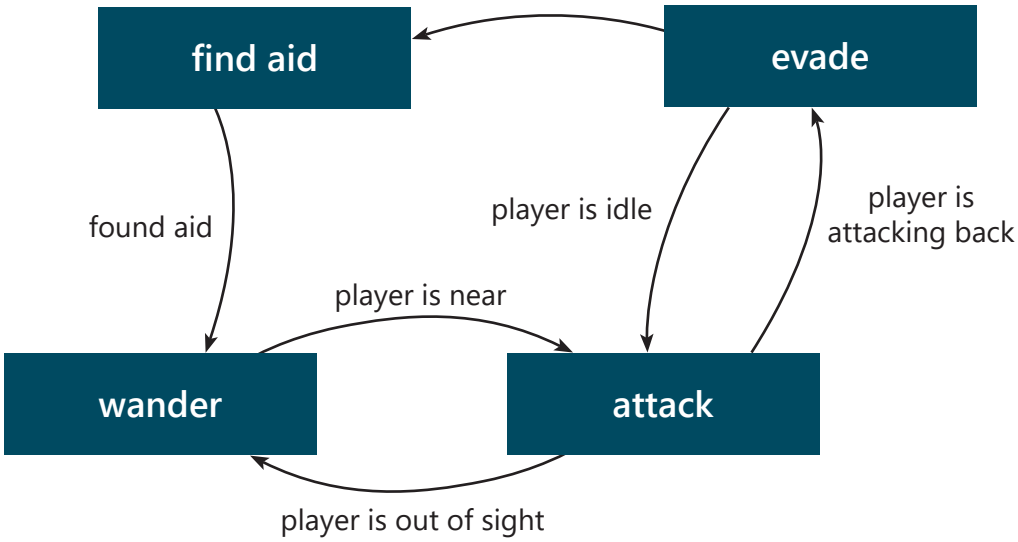
Quelle: https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867 (Nach Vorlage)

Pathfinding ist lediglich ein Abruf eines Algorithmus und hat nur in Ansätzen etwas mit künstlicher Intelligenz zu tun, weswegen auf diesen Abschnitt nicht weiter eingegangen wird [9].

1.2. Finite State Machine

Das 1992 veröffentlichte *Wolfenstein 3D* enthielt bereits eine rudimentäre Form von KI – den *Finite State Machine Algorithmus* (kurz: FSM). Die Spieleentwickler erstellen für jeden NPC eine Liste an Ereignissen, auf die er reagieren kann. Die Liste enthält verschiedene, aber eine endliche Anzahl an Situationen, anhand dessen der NPC handelt [10]. Doch wie genau funktioniert diese endliche Maschine?

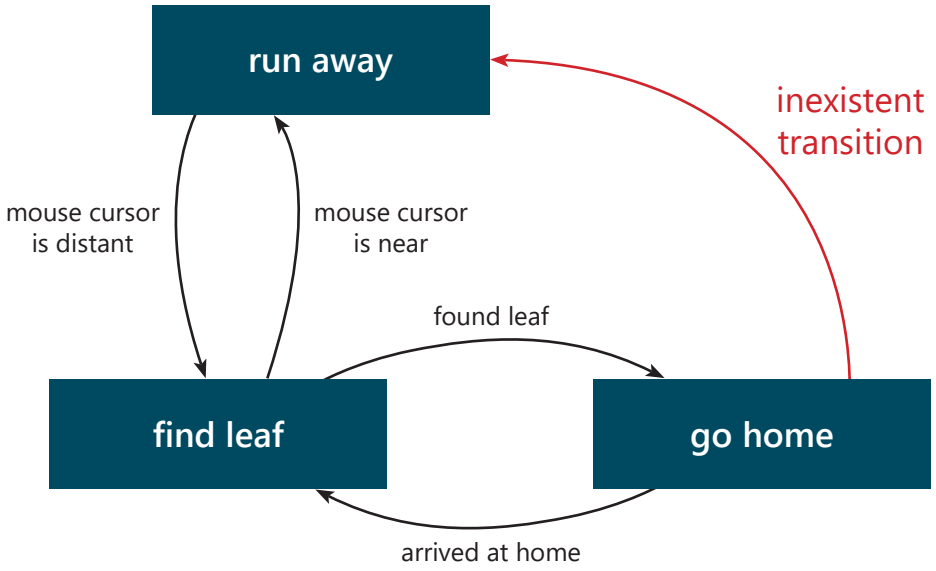
Die FMS wird für die Darstellung und die Steuerung des Ausführungsflusses von NPCs verwendet. Es handelt sich hierbei um ein Berechnungsmodell, bei dem immer nur ein Status aktiv sein kann. Durch die Darstellung mit einem Graphen wird das System verständlich:



Quelle: https://cdn.tutsplus.com/gamedev/uploads/2013/10/fsm_enemy_brain.png (nach Vorlage)

Die Knoten (z.B. „attack“) beschreiben die verschiedenen Zustände, die Pfeile die Übergänge bzw. Übergangsvoraussetzungen. Hierbei besitzt der NPC immer einen Anfangs- bzw. Grundzustand – hier „wander“. Der Computergegner bleibt so lange in diesem Zustand, bis eine der Bedingungen – meist externer Natur – erfüllt ist, um einen anderen Zustand einzunehmen.

Für Wolfenstein 3D bedeutet dies: Der NPC „bewacht“ solange den Raum, in dem er sich befindet, bis der Spieler diesen Raum betritt und in Sichtweite des NPCs gelangt. Nun ist der sog. *Treshhold* überschritten und der Zustand wird in „attack“ gewechselt. Hierbei gibt es auch nicht-existente Übergänge (*inexistent transitions*). Das sind Übergänge, die nicht durch die FSM definiert sind. Im folgenden Beispiel wird das Gehirn einer Ameise simuliert. Der Grundzustand ist „find leaf“. Wurde dies erledigt, ändert sich der Zustand in „go home“. Wenn während dem Grundzustand der Mauszeiger in die Nähe der Ameise kommt, läuft sie von diesem weg. Nähert sich der Mauszeiger jedoch während dem Nachhauseweg, wird die Ameise nicht darauf reagieren, da diese Bedingung nicht in der FSM definiert ist [11].



Quelle: https://cdn.tutsplus.com/gamedev/uploads/2013/10/fsm_ant_brain_no_transition.png (nach Vorlage)

Dies sind zwei vereinfachte Beispiele einer FSM. Je mehr Details ein NPC erhält, desto komplexer wird das Diagramm und die Programmierung. Darüber hinaus kann ein FSM nicht in jedem Spiel verwendet werden. Grundsätzlich wurde es für (Ego-)Shooter entwickelt, um das Gegnerverhalten realitätsnah darstellen und simulieren zu können. In einem Strategiespiel ist dieses System jedoch nahezu unbrauchbar. Wenn die Reaktion eines NPCs auf einen bestimmten Input immer die selbe wäre, würde der Spieler sehr schnell lernen, wie er den Gegner überlisten kann und das würde das Spiel auf längere Sicht sehr eintönig und langweilig machen. Auf Grund dessen wurden Verhaltensbäume entwickelt, mit denen die Wiederholbarkeit des FSM verhindert wird [12].

1.3. Behaviour-Trees

Ein Behaviour-Tree (kurz: BT) besteht aus einem hierarchischen, verzweigten System von Knoten mit einem gemeinsamen Elternteil, das als „Wurzel“ bezeichnet wird. Anders als bei State Machines, die mit einer Abfolge „linearer“ Zustände arbeiten und auf Übergangsregeln angewiesen sind, um durchlaufen werden zu können, wird der Ablauf von BTs streng durch die Reihenfolge der einzelnen Knoten in der größeren Hierarchie bestimmt, um den Entscheidungsprozess eines NPCs zu steuern [13].

Prinzipiell sind BTs eine Weiterentwicklung der Finite Machines. Innerhalb der Verhaltensbäume sind die „Blätter“ die Befehle, welche die KI steuern. Die Verzweigungen sind verschiedene Arten von Hilfsknoten, die den Weg der KI entlang der Bäume navigieren. So werden bestimmte Befehlsfolgen erreicht, die für die jeweilige Situation am geeignetsten sind.

Verhaltensbäume können sehr tief und umfangreich werden, wobei Knotenpunkte weitere Teilbäume aufrufen können. Diese Teilbäume führen wiederum weitere Funktionen und Aktionen aus. So können aus Sicht der Entwickler große Bibliotheken mit verschiedensten Verhaltensweisen erstellt werden, die so miteinander verbunden sind, dass ein überzeugendes KI-Verhalten erreicht werden kann.

Grundsätzlich startet ein NPC mit einem Basisverhalten, mit weiteren Zweigen kann das Erreichen eines bestimmten Ziels angestrebt werden. Scheitert das Ziel, gibt es zusätzlich Rückfalltaktiken, um wieder auf einen Ausgangspunkt zurück zu gelangen.

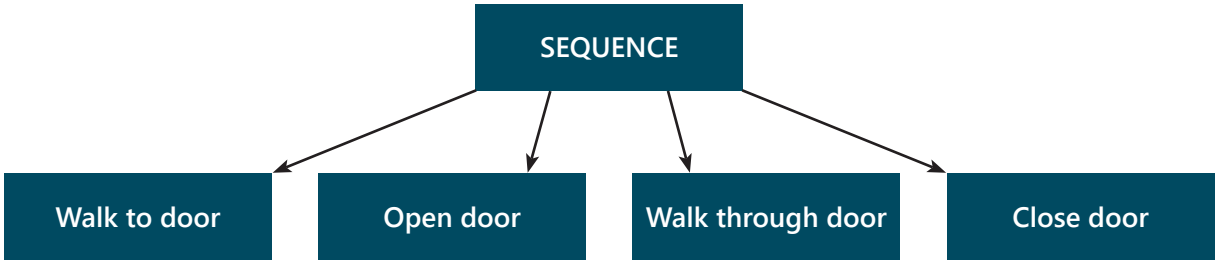
Bei der Implementierung eines BTs durchläuft das System iterativ jeden einzelnen Frame von der Wurzel bis zum aktiven Knotenpunkt. Hierbei muss bei jedem Knoten überprüft werden, ob es sich um den aktuell aktiven handelt. Erst dann kann wiederholt kontrolliert werden, ob er erneut aktiviert werden soll. Diese Art der Überprüfung ist sehr ineffizient, vor allem dann, wenn es sich um tiefreichende Verhaltensbäume handelt. Bei der Implementierung muss darauf geachtet werden, dass alle Knoten, die derzeit verarbeitet, auch gespeichert werden.

Generell kann jeder Knoten eines BT einen der folgenden drei Zustände einnehmen:

- Erfolg *(Success)*
- Misserfolg *(Failure)*
- Im Gange *(Running)*

Success und Failure teilen ihren Elternknoten mit, ob die aktuelle Ausführung ein Erfolg oder Misserfolg war. Running beudetet, dass aktuell noch nicht über Erfolg oder Scheitern entschieden werden kann und der Knoten noch aktiv bleibt. Beim nächsten Durchlauf des Baumes wird erneut überprüft, ob bereits ein Erfolg oder Misserfolg feststeht. Soll ein NPC beispielsweise von A nach B laufen, so bleibt der „Walk“-Knoten so lange auf dem Running-Status wie die Pathfinding-Berechnung andauert und so lange es dauert, bis der Charakter den Zielort erreicht. Stimmt die aktuelle Position des Charakters mit der Position des Ziels überein, so wird *Success* an den Elternknoten zurückgegeben. Sollte das Berechnen des Pfades oder der Gang zum gewünschten Ort unterbrochen werden bzw. fehlschlagen, wird *Failure* vom aktuellen „Walk“-Knoten zurückgegeben.

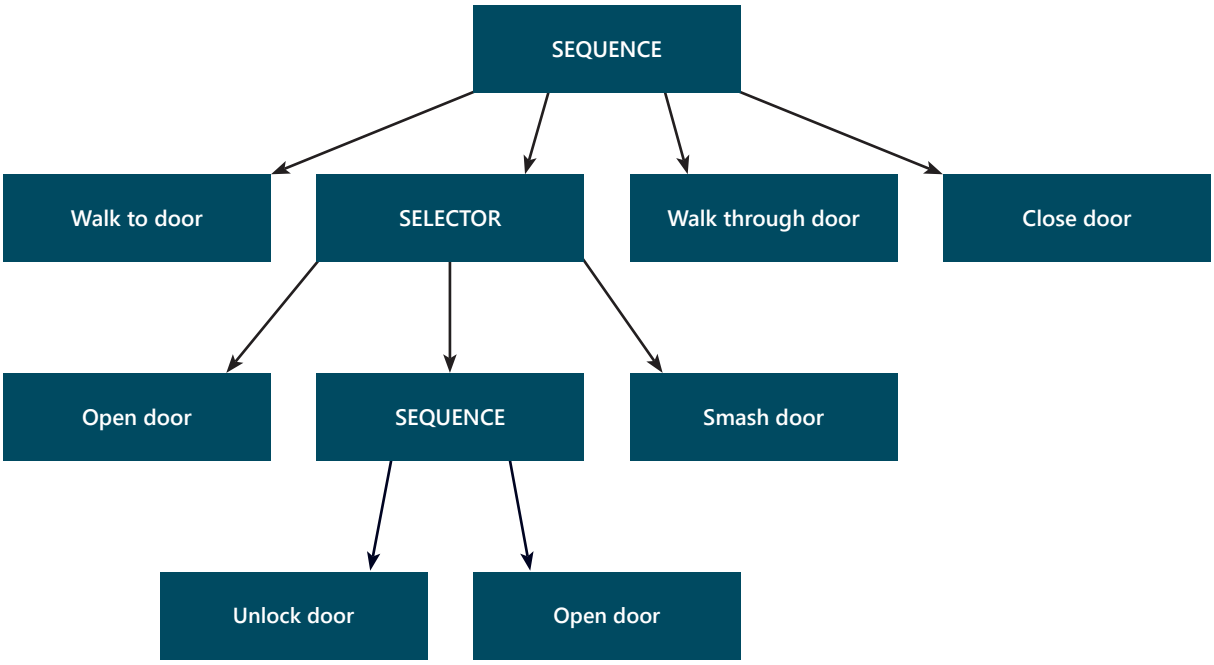
Zusammengesetzte Knoten werden häufig als Sequenz verwendet. Eine Sequenz überprüft alle Kindknoten in einer festgelegten Reihenfolge. Nur wenn das erste Kind erfolgreich ist,



Quelle: https://outforafight.files.wordpress.com/2014/07/image03.png (nach Vorlage)

wird das zweite aufgerufen. Sobald ein Kind einen Misserfolg mitteilt, wird die gesamte Sequenz abgebrochen. Ist der letzte Unterknoten ein Erfolg, dann wird der Gesamterfolg an das Elternelement gegeben.
Wenn in dieser Sequenz beispielsweise der Weg zur Tür versperrt ist und somit bereits die erste Aktion nicht ausgeführt werden kann, dann ist es nicht mehr relevant zu prüfen, ob sich die Tür öffnen lässt. Die gesamte Sequenz wird schon im ersten Schritt abgebrochen und vom Elternknoten anderweitig behandelt. Mit Sequenzen können jedoch nicht nur einzelne Aktionen überprüft und ausgeführt werden. Es besteht auch die Möglichkeit, jeden Kindknoten als einen eigenen Test mit weiteren Unterobjekten zu definieren, bei dem verschiedene Bedingungen erst nacheinander überprüft werden, bevor bei Erfolg aller Tests eine bestimmte Aktion ausgeführt wird.

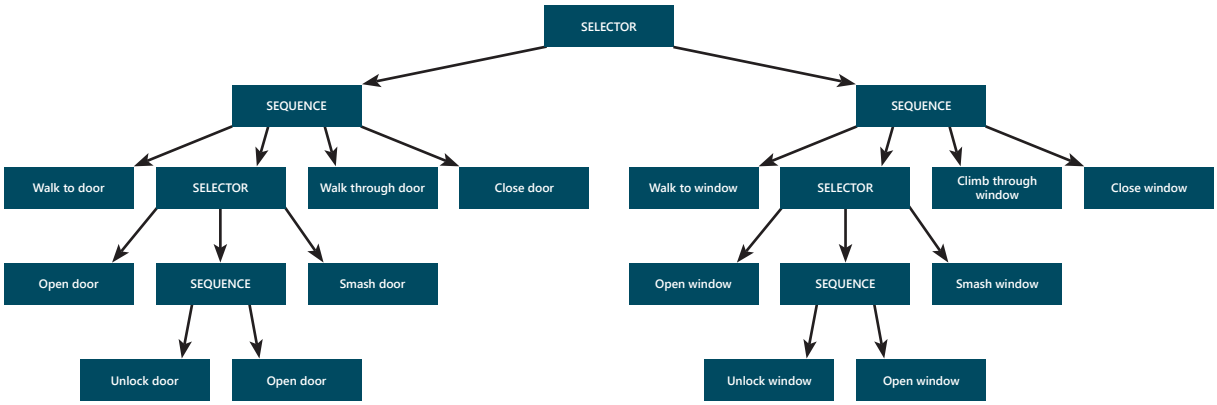
Ist die Sequenz das analoge „AND“, bei dem alle Kindknoten erfolgreich sein müssen, so handelt es sich beim *Selector* um das „ODER“. Ein Selektor gibt einen Erfolg zurück, sobald nur eines der Unterelemente einen Erfolg aufweist. Antwortet das erste Kindelement mit einem Misserfolg, dann wird das zweite Kindelement überprüft. War der Selektor bereits beim ersten Kindknoten erfolgreich, beachtet er die weiteren Möglichkeiten nicht und gibt direkt einen Erfolg zurück. Im folgenden Beispiel bedeutet das:



Quelle: https://outforafight.files.wordpress.com/2014/07/selector1.png (Nach Vorlage)

Die einfachste Variante durch die Tür zu gelangen ist es, die Tür einfach zu öffnen. Gelingt dies, besteht keine Notwendigkeit die anderen Unterknoten des Selectors zu prüfen und es kann in der Sequenz direkt der nächste Schritt bearbeitet werden. Schlägt auch das letzte Element „Smash Door“ fehl, gibt es in diesem Beispiel keine weiteren Möglichkeiten, mit denen der Charakter in den anderen Raum gelangen kann. Um das zu kompensieren, verschafft mögli-

cherweise ein weiterer Selektor über dem obersten Elternteil Abhilfe. Erst wird der linke, bevorzugte Teilbaum bearbeitet, schlägt „Smash Door“ fehl, wird der rechte Teilbaum überprüft.



Quelle: https://outforafight.files.wordpress.com/2014/07/selector3.png (nach Vorlage)

Behaviour Trees im Bereich der Spieleentwicklung wurden erstmals beim 2005 erschienenen Ego-Shooter *Halo 2* verwendet [14]. Bei modernen Spielen können die Bäume sehr komplex und tief werden. *Red Dead Redemption 2* vom Spieleentwickler Rockstar Games ist eines der realistischsten Spiele auf dem derzeitigen Videospielmarkt. Jedes einzelne der über 200 geschaffenen Tierarten zeigt einzigartige Verhaltensweisen und ist in der Lage mit dem Spieler oder mit Artgenossen zu interagieren. Feinde des Hauptcharakters Arthur Morgan besitzen ebenfalls verschiedenste Verhaltensweisen. In einem Feuergefecht suchen sie beispielsweise nach Deckung, um nachzuladen oder flankieren ihre Gegner. Arthur und seine Kollegen reagieren auch auf das Sperrfeuer, indem sie zusammenzucken oder sich ducken, wenn eine Kugel nah an ihnen vorbeifliegt [15][16].

3.4. Ausblick

Wie eingangs erwähnt werden die von NPCs getroffenen Entscheidungen nicht durch eine „richtige“ künstliche Intelligenz gesteuert. Es ist vielmehr ein Abruf vieler vordefinierter Verhaltensmuster, die für die jeweilige Situation angemessen sind. Auch eine Konversation mit einem NPC erfolgt einzig durch das Klicken durch Dialogbäume, wodurch eine Gesprächsdynamik lediglich angedeutet wird. Wie würde jedoch eine solche „richtige“ KI in Videospielen aussehen und welche Folgen hätte eine Implementierung dieser für die zukünftige Spielerfahrung?

Wird ein NPC mit einer KI ausgestattet, wären diese Mit- oder Gegenspieler „virtuelle Wesen“, die verstehen, was der Spieler sagt oder tut und je nachdem darauf reagieren. Um dies umsetzbar zu machen, benötigen alle NPCs eine Spracherkennung, etwa wie Googles KI „Google Duplex“, mit der die phonetischen Laute des Spielers entschlüsselt werden. Dadurch könnte

beispielsweise mit einem Schmied, der verschiedene Waffen und Rüstungen anbietet, interagiert werden. Das immersive Erlebnis dabei wäre wesentlich größer, als wenn sich ein Dialogfeld öffnet und man über ein User Interface alle verfügbaren Waren angezeigt bekommt. Des Weiteren würden alle NPCs mit einer Persönlichkeit ausgestattet sein. Der Schmied-NPC hilft dem Spieler dann beispielsweise nicht weiter, wenn er ihn unfreundlich findet oder einfach einen schlechten Tag hat.

Die Geschichte, die ein Spieler beim Erleben seines virtuellen Abenteuers schreibt, ist von den Entwicklern meist strikt vorgegeben. Die Entwickler entscheiden im Großen und Ganzen im Vornherein darüber, welche Erlebnisse der Nutzer machen wird. Würde ein neuronales Netz in jeden NPC eingebaut werden und würde jeder Computergegner dadurch selbstständig seinen Behaviour Tree entwickeln, hätte das weitreichende Folgen. Die Geschichte, die der Spieler erlebt, kann nun nicht mehr vom Entwickler vorherbestimmt werden. NPCs entwickeln sich in eine nicht vorher definierbare Richtung. Möglicherweise wird der Schmied mehrmals beleidigt oder angegriffen und beschließt deshalb, seinen Job aufzugeben und sich stattdessen bei allen zu rächen, die ihn misshandelt haben. Dann gäbe es ab diesem Moment keinen Schmied mehr, bei dem sich der Spieler neu ausstatten kann. Der Schmied, der einst ein NPC war – und das streng genommen natürlich auch bleibt –, beginnt nun mit seiner eigenen Geschichte [17].

2. KI gegen Mensch

Seit Jahrzehnten werden Spiele herangenommen, um die Leistung von künstlicher Intelligenz zu testen und zu bewerten. Mit zunehmender Leistungsfähigkeit wurde nach komplexeren Spielen gesucht, die die KI meistern soll. Meilensteine dieser Forschungen waren z.B. der *Deep-Blue-Schachcomputer* von IBM oder Googles DeepMind-Entwicklung *AlphaGo* [18] [19][20].

2.1. DeepBlue – Schach

Entwickler: IBM

Dem von IBM entwickelte und 1996 erschienene Schachcomputer DeepBlue gelang es als ersten Computer, gegen den damaligen amtierenden Schachweltmeister Garri Kasparow in einer Partie zu gewinnen. Selbigen Gegner schlug die KI 1997 erneut – diesmal in einem Wettkampf aus sechs Partien unter Turnierbedingungen. Die Spielstärke wurde hauptsächlich aus der für damalige Verhältnisse enormen Rechenleistung bezogen. 1997 bestand der Computer aus 30 Knoten mit 480 Chips. Dadurch konnten im Durchschnitt 126 Millionen Stellungen pro Sekunde berechnet werden [19].

2.2. AlphaGo / MuZero – Go

Entwickler: Googles Deep Mind

AlphaGo ist ein von Googles DeepMind entwickeltes Computerprogramm, das 2015 den mehrfachen Europameister Fan Hui im Brettspiel Go besiegen konnte. Unter Turnierbedingungen wurde der menschliche Gegner auf einem 19×19-Brett geschlagen. Dies konnte erreicht werden indem AlphaGo das Modell und Prinzip des Spieles erlernt hat. Was wenn die Regeln und das Modell des Spieles nicht klar sind? Kann eine KI die Spielregeln selbstständig lernen und sich eigenständig ein Modell eines Spiels für die Planung erstellen? Basierend auf AlphaGO konnte MuZero entwickelt werden. MuZero erlernt nicht nur Brettspielmodelle sondern auch Videospielmodelle und kann diese verstehen. Dazu versucht sie die Umgebung zu erfassen und leitet daraus ihr Verhalten ab.

MuZero kann Modell-basierten KI´s helfen, indem es Aufgaben löst, bei denen es für Menschen zu aufwändig bzw. unmöglich wäre ein Modell im Vorfeld zu definieren [20][21][22].

2.3. OpenAI Five – Dota 2

Entwickler: OpenAI

Die aus fünf Neuronalen Netzen bestände Künstliche Intelligenz hat großes Aufmerksamkeit in Bereich KI´s in Videospielen gewonnen. Die KI zielt darauf Teams in Dota 2, einem komplexen Multiplayer Online Battle Arena Videospiel, zu besiegen. Dazu benötigt es keine menschlichen Daten. Durch Verwendung von LSTM werden erkennbaren Strategien erlernt. In Gegensatz zu Schach oder Go haben Videospiele noch mehr Möglichkeiten. Beispielsweise werden 20.000 Bewegungen analysiert und verarbeitet. In Schach wären es 40, in Go 150 mögliche Bewegungen. Zudem sind Areale durch Hindernisse blockiert, wobei Schach und Go volle Informationen liefern beim Spielen. Open AI fängt praktisch von allein an zu lernen. Dabei spielt die KI das Spiel selbstständig mehrere Male. Pro Tag lernt die KI 250 Jahre an Spielzeit dazu. Ziel von solchen Projekten ist das Sammlen von Erfahrungen, um die Kapazität künstlicher Intelligenz in praktischen Szenarios voranzutreiben. OpenAI Five ist ein gutes Beispiel um die Kommunikation zwischen zwei KI´s zu trainieren [23].

2.4. AlphaStar – Starcraft

Entwickler: Googles Deep Mind

StarCraft vom Entwickler *Blizzard* ist eines der derzeit anspruchsvollsten Echtzeit-Strategie-Spielen. Spiele wie *Mario* oder *Dota 2* wurden bereits von einer KI gemeistert. Bis 2019 hatten die KI-Techniken jedoch Schwierigkeiten, mit der Komplexität von *StarCraft* umzugehen. Nur durch erhebliche Einschränkung der Spielregeln, übermenschlichen Fähigkeiten oder lediglich auf stark vereinfachten Spielkarten wurden bislang die besten Ergebnisse erzielt. Jedoch konnte die KI – auch mit diesen Modifikationen – noch nicht an menschliche Spieler herankommen. *AlphaStar* spielt im Gegensatz zu bisher getesteten künstlichen Intelligenzen mit einem tiefen neuronalen Netz, das direkt aus Rohdaten des Spiels trainiert wird. In zwei Sätzen zu je fünf Spielen gelang er der KI im Jahr 2019 gegen die professionellen Spieler Dario „TLO“ Wünsch und Grzegorz „MaNa“ Komincz zu gewinnen. Dabei konnte die KI alle Spiele für sich entscheiden. Im Gegensatz zu menschlichen Spielern hat *AlphaStar* ständig einen Überblick über alle sichtbaren Bereiche der Karte, fokussiert sich dabei jedoch immer nur auf einzelne Kartenbereiche. Die Anzahl der Aktionen pro Minute wurde auf menschliches Maß angepasst. AlphaStar zeigte dabei besondere Stärken im Micromanagement und im Multitasking [24][25].

3. KI als Spieleentwickler

Künstliche Intelligenz soll dem Menschen in verschiedenen Bereichen helfen. Doch wie weit kann Künstliche Intelligenz einem bei kreativen Aufgaben wie das Erstellen von Videospielen unter die Arme greifen? Dieser Abschnitt soll eine Vorstellung dessen liefern, in welche Richtung Künstliche Intelligenz in der Gaming Branche sich entwickeln könnte.

3.1. Level Generierung mit Hilfe von KI

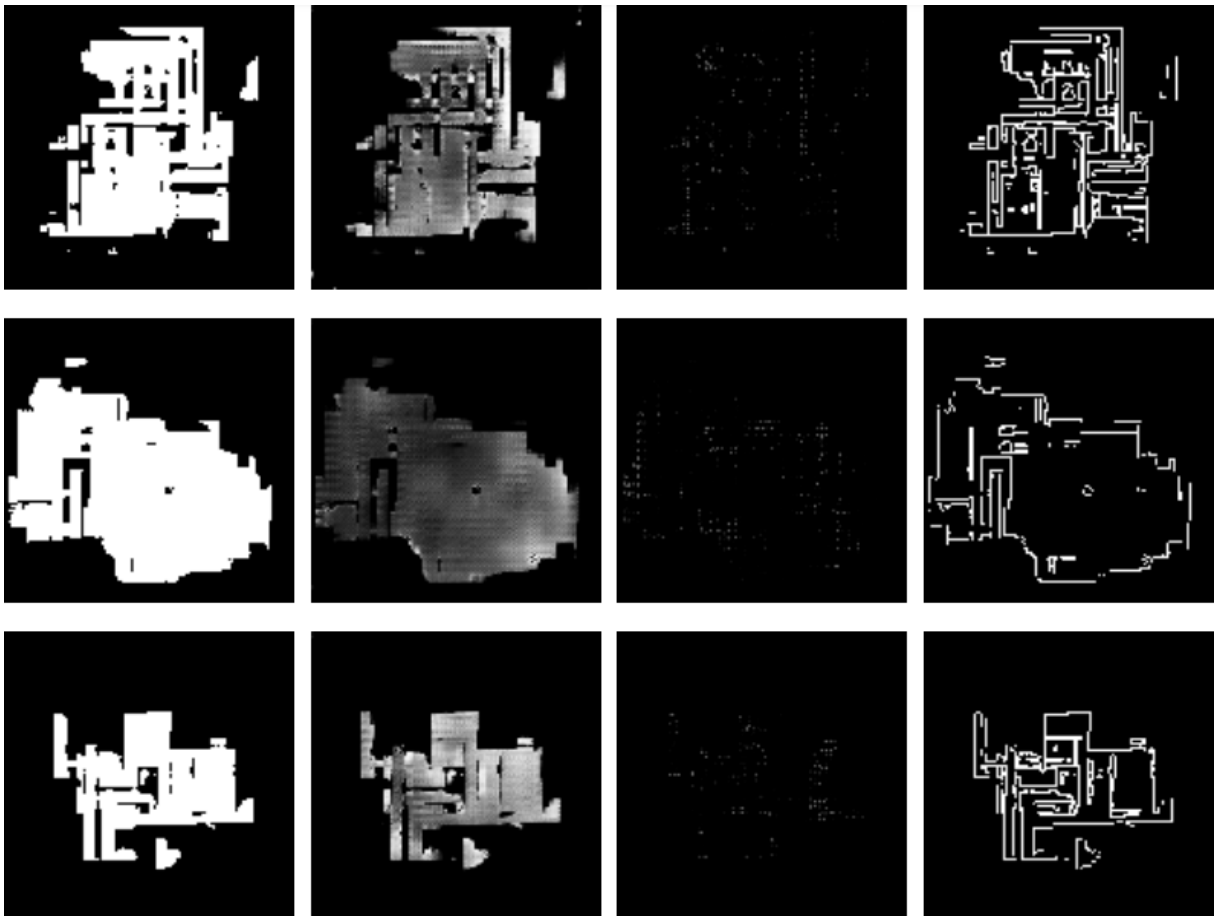
Spieleentwickler erstellen viele verschiedene Abschnitte, die ein Spieler während dem Spielen durchlaufen muss, um ein gewisses Ziel zu erreichen. Dabei sind Balance, Performance und das Aussehen wichtige Faktoren, die den Spieler an ein Spiel fesseln sollen.

Wie soll Künstliche Intelligenz dabei helfen? Game Development ist sehr kostspielig und zeitaufwändig. Künstliche Intelligenz soll den Spieleentwickler dabei helfen, schneller und effizienter zu arbeiten. Beim maschinellen Lernen können (dem Entwicklungsprozess unabhängige) Datenströme übermittelt werden, damit Assoziationen hergestellt oder Muster erkannt werden können. Diese Muster und Assoziationen werden benutzt um Level zu generieren.

Beispielsweise wurden neuronale Netze erfolgreich darauf trainiert Level-Maps für Doom, einen Ego-Shooter aus dem Jahre 1993, zu generieren. Dabei werden GANs(Generative Adversarial Networks) verwendet. GANs bestehen aus zwei künstlichen neuronalen Netzwerken, die sich gegenüberstehen. Einer davon generiert Ergebnisse, auch Generator genannt, der andere bewertet die Ergebnisse und wird als Diskriminator bezeichnet [26].

Der Generator erhält mehrere tausende Doom Level-Maps als Trainingsdaten. Dabei werden wichtige Merkmale als Referenzbilder dargestellt, wie beispielsweise begehbare Flächen, Wände oder Objekte. Zudem werden Vektorgrößen für Form, Flächengröße und Raumanzahl standardisiert. Referenzbilder und Vektorgrößen können dann in numerischer Form im GANs verwendet werden. Basierend auf diesen Daten versucht der Generator neue Levels zu erstellen. Der Diskriminator gibt sich zufrieden wenn er glaubt, dass das Level von einem Menschen erschaffen wurde [27][28].

Der schwierigste Teil hierbei ist die komplexe Struktur einer Level-Map. Die Trainingsdaten besitzen eine große Anzahl verschiedener Werte die ein Level beschreiben. Die Auswahl derjenigen, die besser mit dem neuronalen Netzwerk funktionieren, ist eine der größten Herausforderungen. Die automatisch erzeugten Level-Maps können im Nachhinein von Spieleentwicklern - unter Zuhilfenahme ihres Fachwissens - erweitert werden. Die Möglichkeit, den Prozess zu automatisieren, erleichtert das Gestalten neuer Level.



Quelle: https://hackaday.com/wp-content/uploads/2018/05/generated-doom-maps1.png

Das Ziel der künstlich generierten Level-Maps ist die verkürzte Entwicklungszeit. Dabei soll ein Teil der Entwicklung von der Künstlichen Intelligenz übernommen werden. Der Mensch soll hierbei nicht ersetzt werden, sondern sich eher auf höherer Ebene auf das Game Design fokussieren.

Geschichten in einem Videospiel zu erzählen ist nur begrenzt möglich. Der Spieleentwickler überlegt sich vorher eine Geschichte die er erzählen will und versucht diese zu implementieren indem er Antwortmöglichkeiten, Story-Verläufe und Ergebnisse händisch programmiert. Was wenn die Geschichte jedoch dynamisch sein soll? Jeder Spieler soll selbst seine Geschichte frei erforschen können, eigene Pfade erkunden und den Spielfluss beeiflussen können. Das ist heute schon möglich. Das Spiel AI Dungeon 2 ist beispielsweise ein klassisches textbasiertes Abenteuerspiel. AI Dungeon 2 befreit den Entwickler von den Grenzen der Programmierung. Die Künstliche Intelligenz berechnet die Antworten und setzt den Verlauf des Spieles um. Die Problematik liegt darin das viele Aspekte für eine Spielumsetzung dynamischer werden müssten. Da sich AI Dungeon 2 um ein textbasiertes Abenteuerspiel handelt, müssen Objekte, Landschaften und Personen nicht dynamisch modelliert werden. Könnte man jedoch auch die Welt von der Künstliche Intelligenz erschaffen, so könnten gesamte Videospiele über Künstliche Intelligenz erzeugt werden [29].

3.2. Level Testen durch Ki

Videospiele müssen getestet werden, bevor sie dem Markt übergeben werden können. Bugs und Glitches können den Spielspaß senken oder gar ein Spiel unspielbar machen. Deswegen gib es Quality Assurance Tester, die Videospiele testen und bis ins kleinste Detail auf Fehlersuche gehen. Je größer das Spiel desto mehr Zeit wird benötigt um einem Zero Bug Release gerecht zu werden [30].

Um die Entwicklungszeit weiter zu kürzen und effektiver zu nutzen könnte künstliche Intelligenz die Lösung sein. Ein Videogame durchzuspielen und nach Fehler zu suchen benötigt mehrere Durchläufe. Was wenn diese eine künstliche Intelligenz übernehmen würde? Eine künstliche Intelligenz baut im Trainingsprozess sowieso auf Iterationen auf. Wichtig ist hierbei das die künstliche Intelligenz versucht einen Menschen zu kopieren und dessen Ausführung nachahmt.

Künstliche Intelligenz automatisiert der Prozess vom Testen eines Videospiels. Beispielsweise kann mit Icarus, einer künstliche Intelligenz von Daedalic Entertainment, Point&Click Adventures durchlaufen werden [31].

Icarus wird verwendet um tägliche Builds auszuwerten, Hardware-Kompatibilitätstests durchzuführen oder teilüberwachte Durchläufe vom Videospiel durchlaufen zu lassen [32]. Künstliche Intelligenz in Videospielen sinnvoll zum Level-Testen einzubauen benötigt allgemeine Richtlinien. Künstliche Intelligenz wird sich sicherlich weiter in Richtung Testing weiterent-

wickeln. Die Möglichkeit hunderte oder tausende Test Szenarien aufeinmal zu durchlaufen ist ein guter Grund dafür. Künstliche Intelligenz ist sicherlich die Zukunft des Testens. Menschen können beim Testen von Videospiel dann mehr auf die kreative Seite achten und der künstlichen Intelligenz beispielsweise neue Testfälle geben [33].

Quelle: https://www.researchgate.net/profile/Johannes_Pfau/publication/320315039/figure/fig1/AS:711301957427203@1546599010635/Example-scene-of-the-game-Annas-Quest-containing-18-target-objects-indicated-by-blue.png

3.3. Neural State Machine

Videospiele simulieren viele Aspekte unserer Welt. Von Laufen bis hin zum Springen über Objekte.

In der 8/16 Bit Ära von Videospielen waren Charakteranimationen rudimentär, mit limitierten Interaktionsmöglichkeiten. Deswegen benötigte man nicht viele verschiedene Animationen. Nach dem Übergang zu 3D Videospielen wurden auch die Animationen komplizierter. Jetzt, da Spiele mit riesigen und offenen Welten, die erfroscht werden und mit denen man interagieren kann, erheblich komplexer geworden sind, erforderen die Charakteranimationen im Spiel Hunderte oder sogar Tausende verschiedener Bewegungensmuster.

All diese Bewegungen werden heutzutage mithilfe von Motion Capture Verfahren implementiert. Dazu werden beispielsweise Menschen mit Tracking-Verfahren erfasst und aufgezeichnet. Die Daten werden später in 3D-Systeme importiert und weiterverarbeitet. Daraus werden

sogenannte Rigs erstellt die zusammen mit der Figur den 3D Charakter bilden. Es ist jedoch praktisch unmöglich, alle möglichen Arten der Interaktion eines Spielers mit der Umgebung zu erfassen. Der Prozess ist ziemlich teuer und zeitaufwändig. Viele Animationen müssen aufgezeichnet werden. Das Animieren von Charakteren ist eine schwierige Aufgabe, wenn es um die Interaktion mit Objekten und der Umgebung geht.

Übergänge zwischen Animationen können verzerrt aussehen. Normalerweise werden Änderungen zwischen Bewegungen von wiederverwendeten Algorithmen verarbeitet. Ein Problem liegt dabei bei verschiedenen Größen von Objekten. Wird beispielsweise eine Kiste aufgehoben, muss die Animation der Arme auf das Objekt angepasst werden.

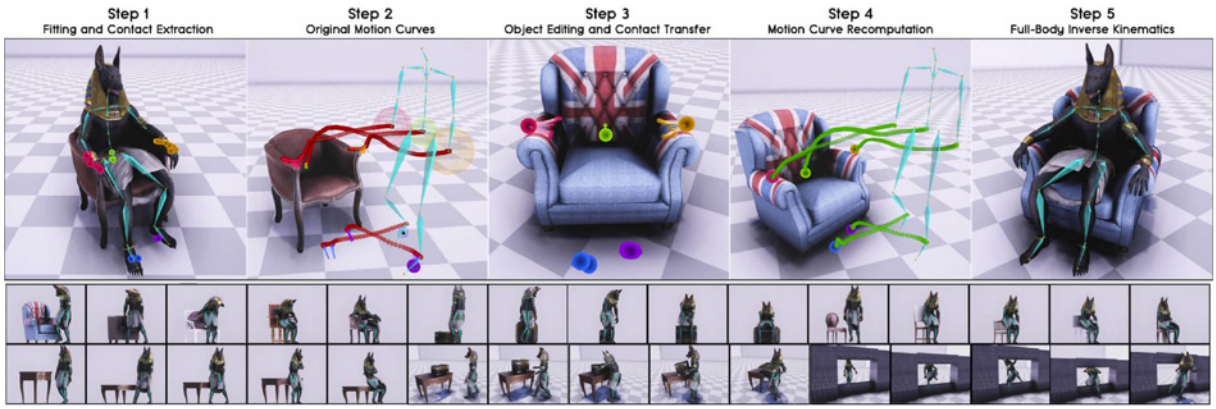


Fig. 6. The five steps of data augmentation for adapting to different shapes (top). Step 3-5 are applied per frame to prepare training data where the character can adapt to different shapes without increasing the data size (bottom).

Quelle: https://www.youtube.com/watch?v=7c6oQP1u2eQ (Screenshot)

Ein Framework, das das Animieren von Charakteren vereinfacht, ist die Neural State Machine. Neural State Machine ist ein datengesteuertes Deep-Learning-Framework, das mit Animationen umgeht. Es ist in der Lage Interaktionen zwischen Charakteranimationen und Szenen zu lernen und damit hochwertige Animationsübergänge zu erstellen. Als Beispiel kann das Aufheben einer Kiste genommen werden. Bevor die Kiste angehoben werden kann, müssen mehrere Bewegungen animiert werden. Der Charakter muss zum Objekt gehen, langsam werden und sich zur Kiste drehen. Bei der Neural State Machine lernt der Charakter die nötigen Bewegungen direkt aus der Szenengeometrie und einer bestimmten Zielaktion. Die Neural State Machine wird unter Verwendung von Motion-Capture-Daten geschult, um zu lernen, wie man auf natürliche Weise von einer Bewegungen zur nächsten übergeht. Dabei leitet es die nächste Pose von der vorherigen Pose und der Szenengeometrie ab [34][35][36].

Nimmt man die Neural State Machine als Werkzeug für die Erstellung Animation von Charakteren in Videospielen, so lassen sich viele Interaktionen wesentlich realistischer darstellen. Beispielsweise könnte sich der Charakter durch Menschenmengen flüssig hindurch bewegen, anstatt abgehackte Bewegungen zu erzeugen wie in Assassins Creed Unity.

Literaturverzeichnis

[1] https://winfuture.de/news,98576.html
(Aufgerufen am: 05.12.2019)

[2] https://tvtropes.org/pmwiki/pmwiki.php/Main/VideoGameAI
(Aufgerufen am: 05.12.2019)

[3] https://www.theverge.com/2019/3/6/18222203/video-game-ai-future-procedural-generation-deep-learning
(Aufgerufen am: 05.12.2019)

[4] https://www.hs-weingarten.de/c/document_library/get_file?uuid=816a49a0-0713-467a-9222-07af213679e8&groupId=34068
(Aufgerufen am: 05.12.2019)

[5] https://www.linkedin.com/pulse/evolution-artificial-intelligence-gaming-chris-pearson
(Aufgerufen am: 05.12.2019)

[6] https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games
(Aufgerufen am: 05.12.2019)

[7] https://towardsdatascience.com/artificial-intelligence-in-video-games-3e2566d59c22
(Aufgerufen am: 05.12.2019)

[8] https://de.wikipedia.org/wiki/Pathfinding
(Aufgerufen am: 06.12.2019)

[9] https://medium.com/swlh/pathfinding-algorithms-6c0d4febe8fd
(Aufgerufen am: 05.12.2019)

[10] https://towardsdatascience.com/artificial-intelligence-in-video-games-3e2566d59c22
(Aufgerufen am: 06.12.2019)

[11] https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867
(Aufgerufen am: 05.12.2019)

[12] https://towardsdatascience.com/artificial-intelligence-in-video-games-3e2566d59c22
(Aufgerufen am: 06.12.2019)

[13] https://hub.packtpub.com/building-your-own-basic-behavior-tree-tutorial/
(Aufgerufen am: 10.12.2019)

[14] https://de.wikipedia.org/wiki/Behavior_Tree
(Aufgerufen am: 10.12.2019)

[15] https://www.vg247.com/2018/12/12/red-dead-redemption-2-physics-ai-euphoria-phil-hooker-interview/
(Aufgerufen am: 10.12.2019)

[16] https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php
(Aufgerufen am: 10.12.2019)

[17] https://mixed.de/ki-gaming-wie-kuenstliche-intelligenz-spiele-revolutionieren-koennte/
(Aufgerufen am: 11.12.2019)

[18] https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii
(Aufgerufen am: 12.12.2019)

[19] https://de.wikipedia.org/wiki/Deep_Blue
(Aufgerufen am: 12.12.2019)

Literaturverzeichnis

[20] https://de.wikipedia.org/wiki/AlphaGo
(Aufgerufen am: 12.12.2019)

[21] https://mixed.de/deepmind-muzero-schlaegt-alphago/
(Aufgerufen am: 14.12.2019)

[22] https://venturebeat.com/2019/11/20/deepminds-muzero-teaches-itself-how-to-win-at-atari-chess-shogi-and-go/
(Aufgerufen am: 14.12.2019)

[23] https://openai.com/five/
(Aufgerufen am: 14.12.2019)

[24] https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii
(Aufgerufen am: 13.12.2019)

[25] https://de.wikipedia.org/wiki/DeepMind
(Aufgerufen am: 13.12.2019)

[26] https://gizmodo.com/an-ai-created-new-doom-levels-that-are-as-fun-as-the-ga-1826111529
(Aufgerufen am: 12.12.2019)

[27] https://www.vice.com/en_us/article/9k85mz/ai-gan-generate-new-doom-levels-automate-game-design
(Aufgerufen am: 12.12.2019)

[28] https://hackaday.com/2018/05/16/neural-networks-using-doom-level-creator-like-its-1993/
(Aufgerufen am: 12.12.2019)

[29] https://www.gamestar.de/artikel/ai-dungeon-2-ki-text-adventure,3352183.html
(Aufgerufen am: 14.12.2019)

[30] https://www.vice.com/en_us/article/gv58z3/gears-of-war-4-aaa-games-are-hard-so-why-make-them
(Aufgerufen am: 14.12.2019)

[31] https://www.researchgate.net/publication/320315039_Automated_Game_Testing_with_ICARUS_Intelligent_Completion_of_Adventure_Riddles_via_Unsupervised_Solving
(Aufgerufen am: 14.12.2019)

[32] https://www.smeddinck.com/publication/pfau2017/
(Aufgerufen am: 14.12.2019)

[33] https://www.vice.com/en_us/article/538we3/researchers-are-building-ai-to-replace-video-game-testers
(Aufgerufen am: 14.12.2019)

[34] https://arxiv.org/pdf/1907.03950.pdf
(Aufgerufen am: 13.12.2019)

[35] https://www.youtube.com/watch?v=7c6oQP1u2eQ
(Aufgerufen am: 13.12.2019)

[36] https://www.techspot.com/news/82566-using-deep-learning-make-video-game-characters-move.html
(Aufgerufen am: 13.12.2019)