Fachhochschule Salzburg University of Applied Sciences

Umsetzung eines KI Spielers mit adaptiver Spielstärke anhand eines praktischen Beispiels

BACHELORARBEIT 2

Studierende/Studierender Alexander Cerny, 0910601006

BetreuerIn DI Robert Praxmarer

Salzburg, am 7. Mai 2012

Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst, und keine anderen als die angegeben Quellen und Hilfsmittel benutzt habe. Weiters versichere ich hiermit, dass ich die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission weder im In- noch im Ausland vorgelegt und auch nicht veröffentlicht.

.....	.....
Datum	Unterschrift

Zusammenfassung

Vor- und Zuname:	*Alexander CERNY*
Institution:	*FH Salzburg*
Studiengang:	*Bachelor MultiMediaTechnology*
Titel der Bachelorarbeit:	*Umsetzung eines KI Spielers mit adaptiver Spielstärke anhand eines praktischen Beispiels*
BegutachterIn:	*DI Robert Praxmarer*

Diese Arbeit beschäftigt sich mit den gängigen KI Methoden Finite State Machine und Fuzzy Logic. Diese werden theoretisch vorgestellt und im Laufe der Arbeit in ein praktisches Beispiel implementiert. Es werden verschiedene Schwierigkeitsstufen von einem KI Gegner modelliert. Für die Spielstärke der KI wird eine adaptive Methode entwickelt, die sich dem Spielniveau des/der menschlichen Gegners/Gegnerin anpasst. Sobald das Niveau erreicht ist, sollen nur mehr minimale Veränderungen an der Stärke vorgenommen werden. Dadurch soll ein knapper Ausgang in jedem Spiel erreicht werden. Die entwickelte Methode zur adaptiven Spielstärke wird in einem praktischen Beispiel umgesetzt. In einem abschließenden Test tritt die KI gegen sich selbst an. Beim Test wird die KI einmal mit dem adaptiven Algorithmus und einmal ohne ihn spielen. Dadurch wird festgestellt, wie effizient die erstellte adaptive Methode ist, im Vergleich zu einer fixen Spielstärke.

Schlüsselwörter: KI, Fuzzy Logic, Finite State Machine, adaptive Spielstärke, adaptive Schwierigkeit, verschiedene Schwierigkeitsstufen

Abstract

This paper introduces the common AI methods Finite State Machine and Fuzzy Logic. First the theoretical basics of the methods are explained and later an example shows how it is possible to implement these theories in a mobile game. The implemented concept of the AI opponent also covers how to model different skill levels for the AI. In addition there are adaptive methods described how the skill level can be changed during a game. Based on these methods an alternative method is created in which the AI skill searches and adapts to the skill of the human player. This adaptive algorithm should achieve that the challenge level of the AI matches the skill level of the human player. In the end the implemented method is tested. The different AI skill levels play numerous times against each other. Half of the matches are played without the adaptive skill method and the other half is played with the method. This test shows the efficiency of the created method.

Key words: AI, Fuzzy Logic, Finite State Machine, adaptive skill, adaptive difficult scaling, different skill levels

Inhaltsverzeichnis

Abkürzungsverzeichnis

C#	Programmiersprache
FSM	Finite State Machine
KI	Künstliche Intelligenz
Unity3D	Game Engine

1. Einleitung

Im Laufe dieser Arbeit soll ein KI Konzept entwickelt werden, das in ein konkretes Spiel eingebunden wird. Bei dem Spiel handelt es sich um ein Mobile Game, das übers Netzwerk gegen seine Freunde gespielt werden kann. Ein Match findet im eins gegen eins Modus statt. Damit ein solches Netzwerkspiel erfolgreich wird, müssen laufend SpielerInnen online sein. Dafür braucht es sehr viele Personen, die das Mobile Game besitzen und verwenden. Bis sich eine solche Fangemeinde aufgebaut hat, dauert es längere Zeit. Um die Zeit zu überbrücken, bis genügend SpielerInnen vorhanden sind, bietet sich ein Einzelspieler Modus an. Dafür wird ein KI Gegenspieler benötigt. Ein maschineller Gegenspieler bietet unter anderem auch die Möglichkeit, dass man offline trainieren kann, bevor man einen richtigen Menschen herausfordert. Zusätzlich kann mit der KI ein Tutorial erstellt werden, das das Spielprinzip intuitiv vermittelt. Aus diesen Gründen wird ein KI Gegenspieler entwickelt.

Für die praktische Implementierung wird eine KI erschaffen, welche das Spielverhalten eines Menschen imitiert. Neben der Aufgabe, als voll funktionierender Kontrahent das Mobile Game zu bedienen, soll die KI auch eine adaptive Spielstärke besitzen. Diese soll gewährleisten, dass sich das Niveau der KI an das des/der menschlichen Kontrahenten/Kontrahentin anpasst. Es sollen allerdings nicht über mehrere Spiele hinweg Informationen gesammelt werden, welche die Spielstärke bestimmen. Sondern die Adaption des Spielniveaus soll innerhalb eines Matches stattfinden. Der Schwierigkeitsgrad des Computers wird variabel gehalten, sodass der Ausgang eines Spiels bis zum Schluss offen und spannend bleibt. Dadurch wird gewährleistet, dass die KI für den/die menschlichen GegnerIn sowohl herausfordernd als auch schlagbar ist. Diese Eigenschaften machen die KI und damit auch das Spiel unterhaltsam für den/die AnwenderIn. (Scott 2002a, 18)

Im Zuge dieser Arbeit soll die folgende Forschungsfrage beantwortet werden:

Wie kann ein KI Gegner mit adaptiver Spielstärke für ein konkretes Spiel entwickelt und umgesetzt werden?

Um diese Frage zu beantworten, werden zuerst Ansätze vorgestellt, wie menschliches Verhalten mittels einer KI simuliert werden kann. Danach werden gängige KI Methoden erklärt, mit denen es möglich ist, ein Konzept für die Implementierung zu erstellen. Zusätzlich werden Ansätze vorgestellt, mittels denen man verschiedene Spielstärken modellieren kann. Diese Ansätze werden mit Methoden erweitern, die das Spielniveau der KI auf den/die menschlichen SpielerIn adaptieren. Nachdem die theoretischen Grundlagen erläutert wur-

den, wird im Implementierungsteil ein KI Konzept für ein Spiel erstellt und umgesetzt. Die Theorie wird bewusst allgemein gehalten, dass man diese Methoden für jedes Genre und jeden Spiel Typ verwenden kann. Die Implementierung soll zeigen, wie man diese Methoden modifizieren und in ein reales Spiel einbauen kann. Abschließend wird das Ergebnis der entwickelten Methode zur adaptiven Spielstärke getestet, indem die verschiedenen Schwierigkeitsstufen der KI gegeneinander antreten. Der Test soll zeigen, wie eng der Ausgang der einzelnen Spiele ist, wenn die adaptive Spielstärke der KI zum Einsatz kommt.

2. Illusion von Intelligenz

Man kann die künstliche Intelligenz (KI) in zwei Kategorien unterteilen, KI mit einem Denkprozess und daraus resultierenden Schlussfolgerungen und KI, die menschliches Verhalten simuliert. Bei der ersten Kategorie handelt es sich um die starke KI und bei der zweiten um die schwache KI. (Norvig and Russell 2003, 4) Bei der starken KI wird versucht, das menschliche Denken auf eine Maschine zu übertragen. Um das zu schaffen, muss die Maschine verstehen, lernen, Wissen speichern und richtig verknüpfen und daraus Schlussfolgerungen ziehen können. Mit dem Turing Test kann überprüft werden, ob diese Eigenschaften erfüllt werden. (Norvig and Russell 2003, 6) Searle hingegen behauptet, dass der Turing Test kein Beweis für starke KI ist, da die KI nichts verstehen muss. Außerdem behauptet er, dass menschliches Verstehen auf Maschinen nicht nachgebildet werden kann. (Norvig and Russell 2003, 231) Kritische Stimmen behaupten, dass starke KI überhaupt nicht möglich ist. Diese stützen sich zum Beispiel auf den gödelschen Unvollständigkeitssatz, der besagt, dass es in einem System Aussagen gibt, die weder beweisbar noch zu widerlegen sind. (Norvig and Russell 2003, 11) Mit anderen Worten, man wird für ein KI System immer eine Aussage finden, die sie nicht beantworten kann. Bei der starken KI handelt es sich um ein Forschungsgebiet. In der Spieleentwicklung hingegen kommt die schwache KI zum Einsatz, in der versucht wird, menschliches Vorgehen zu imitieren und menschliches Verhalten zu simulieren.

Das KI Konzept, das in dieser Bachelorarbeit ausgearbeitet wird, wird für ein Mobilegame entwickelt, das über das Netzwerk gegen einen menschlichen Kontrahenten gespielt werden kann. Der KI Gegner für dieses Spiel soll das Verhalten eines/einer menschlichen Spielers/Spielerin imitieren. Um die Illusion von menschlichem Verhalten glaubhaft auf eine Maschine zu übertragen, müssen einige menschliche Eigenheiten berücksichtigt werden.

2.1. Überraschende Aktionen

Ein Mensch kann für seinen/seine GegnerIn in einem Spiel sowohl durchschaubar als auch zur selben Zeit undurchschaubar sein. Zum Beispiel wird beim Schachspielen meist mit demselben Zug eröffnet oder beim virtuellen Fußballspielen der Angriff mit einem Auswurf des Torwarts forciert. Diese Gewohnheiten können den/die SpielerIn durchschaubar machen. Ein/eine SpielerIn geht meistens nach einer gewissen Strategie vor und führt dadurch vorhersehbare Aktionen aus. Auf der anderen Seite starten Menschen unvorhergesehene Aktionen, die den Gegner überraschen. Das kann zum Beispiel eine Verzweiflungstat oder ein Ablenkungsangriff sein. Wenn die KI einen Angriff aus einem Hinterhalt startet, kann das spielentscheidend sein und der/die KontrahentIn dadurch besiegt werden. (Scott 2002a, 16-17)

Eine andere Art der Überraschung kann auftreten, wenn der/die GegenspielerIn einen Fehler macht. Immerhin kann es während eines Spieles immer zu einem menschlichen Versagen kommen. Während es die leichtere Aufgabe ist für ein simples Spielprinzip eine KI zu konzeptionieren, die keine Fehler begeht, kann es zu einer Herausforderung werden, wenn der Computer glaubhafte Fehler begehen soll. Die Schwierigkeit bei der Fehleranfälligkeit ist, dass der/die menschliche/menschliche SpielerIn das Gefühl haben soll, dass diese Fehler nicht unrealistisch und erzwungen wirken. Zusätzlich sollte das Versagen der KI nur sehr selten auftreten. Solche Fehler können zum Beispiel sein, dass eine Einheit an einen falschen Ort geschickt wird oder dass in einem Rollenspiel ein mächtiger Zauberspruch angewandt wird, der dem Spielcharakter selbst Schaden zufügt und er dabei stirbt. (Scott 2002a, 17-18)

2.2. Konstanz

Eine weitere Eigenschaft, die den Menschen von der Maschine unterscheidet, ist die Konstanz. Für einen Computer ist es kein Problem, die exakte Stelle unzählige Male hintereinander zu treffen, wobei diese Präzision für einen Menschen sehr wohl eine Hürde darstellt. Um dieses Verhalten auf die KI zu übertragen, wird ein Wertebereich bestimmt, in welchem durch Zufall ein Wert ausgewählt wird. Dadurch variiert die Konstanz der Maschine. (Scott 2002b, 287)

3. Gängige KI Methoden

Um die Spielweise eines/einer menschlichen Gegenspielers/Gegenspielerin zu imitieren gibt es einige gängige KI Methoden, die sich in der Vergangenheit durchgesetzt haben und immer wieder verwendet wurden. Die folgenden Methoden wurden ausgewählt in Berücksichtigung, dass im Laufe dieser Arbeit ein KI Konzept für ein praktisches Beispiel erstellt wird. Bei diesem KI Konzept soll ein Computergegner entwickelt werden, der in der Lage ist, gegen einen/eine menschlichen/menschliche SpielerIn anzutreten. Dazu wird zuerst ein Entscheidungssystem für einen autonomen KI Gegner entwickelt. In diesem System werden Entscheidungen durch die Fuzzy Logic Methode getroffen, die danach zu Aktionen führen, die in den einzelnen States einer Finite State Machine ausgeführt werden.

3.1. Entscheidungssystem für einen autonomen KI Gegner

Wie man in der unteren Abbildung erkennen kann, werden die Entscheidungen in diesem System über einen Timer gesteuert. Es ist nicht nötig, dass in jedem Frame die KI ausgeführt wird. Der Computergegner soll einen Menschen imitieren und dieser besitzt beim Nachdenken eine Reaktionszeit. Ein flüssig laufendes Videospiel kann eine Aktualisierungsrate von 70 Aktualisierungen pro Sekunde haben. Würde die KI in jedem Frame ausgeführt werden, hätte der virtuelle Kontrahent eine Reaktionszeit von $\frac{1}{70}$ Sekunde. Da die menschliche Reaktionszeit weit darüber liegt, wird der Timer benötigt um das Ausführen der KI Algorithmen zu steuern. Das heißt mit anderen Worten, die Timerfrequenz simuliert gleichzeitig die Reaktionszeit der KI. Der Einsatz eines Timers steigert auch die Performance, da die Entscheidungsalgorithmen nicht 70 mal pro Sekunde ausgeführt werden. (McLean 2002, 292)

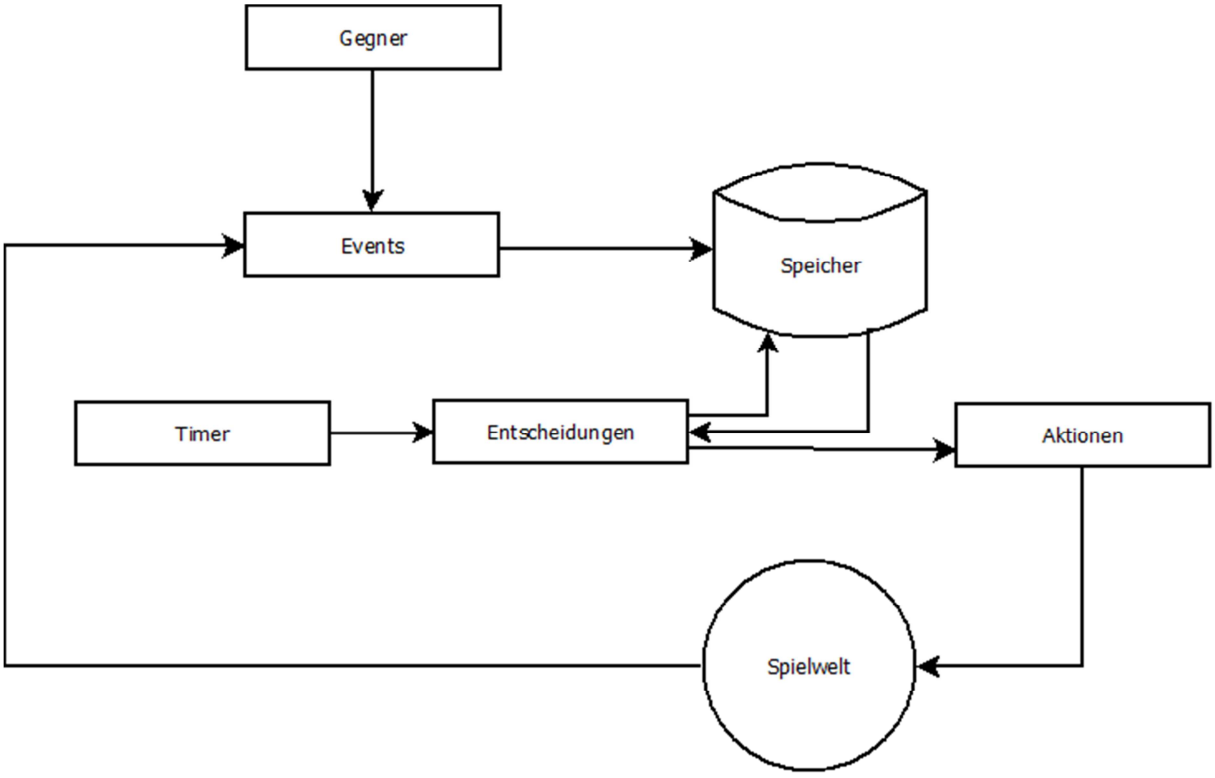


Abb. 1: Entscheidungssystem für autonomen KI Gegner (vgl. Thor 2002, 368)

Damit der Computer eine logische Entscheidung treffen kann, muss er einige Informationen aus der Spielwelt kennen. Dieses Wissen wird in einem Speicher festgehalten, der in der KI Architektur übergeordnet gespeichert wird. Dadurch kann von der KI jederzeit darauf zugegriffen werden, ohne dass der Speicher redundant in tieferen Hierarchien festgehalten werden muss. (Thor 2002, 368) Der Speicher kann durch zwei Möglichkeiten aktualisiert werden. Aktionen, die in der Spielwelt, von dem/der GegnerIn oder von der KI selbst ausgelöst werden, beeinflussen den Speicher automatisch durch Events. Für spezielle Informationen, kann der Algorithmus, der die Entscheidungen der KI trifft, explizit den Speicher anfordern, diese zu aktualisieren. (Thor 2002, 367)

Dadurch, dass der Speicher durch Events verändert wird und die Entscheidungen über einen Timer gesteuert werden, ist die Reaktionszeit des Maschinengegners variabel. Wenn ein Event kurz vor dem Timerupdate den Speicher verändert, reagiert die KI nahezu ohne Verzögerung. Falls ein Event einen Frame nach dem Update auftritt, wird die volle Reaktionszeit ausgenutzt. Dadurch simuliert die KI ein menschlicheres Verhalten, da auch die Reaktionszeit des Menschen variiert. (McLean 2002, 292)

Aufgrund des Wissens, das der KI über den Speicher mitgeteilt wird, werden Entscheidungsmöglichkeiten ausgearbeitet. Durch eine Reihung nach der Wichtigkeit wird eine Entscheidung getroffen. Diese Wahl führt zu einer Aktion, die der virtuelle Kontrahent ausführt. Diese Handlung beeinflusst die Spielwelt und führt wieder zu einem Event, das den Speicher aktualisiert. Um die verschiedenen Aktionen der KI übersichtlich zu verwalten, kann eine Finite State Machine verwendet werden.

3.2. Finite State Machine

Die Finite State Machine (FSM) ist die meist verwendete Datenstruktur in der KI Programmierung. (Schwab 2004) Die FSM bietet eine einfache Organisationsmöglichkeit der KI Logik. Es wird dabei die Gesamtaufgabe der KI in kleinere Teilaufgaben unterteilt. Durch das Aufteilen der Aufgaben wird die KI Struktur übersichtlicher. Das KI Konzept wird dabei in mehrere States aufgeteilt. Jeder State kümmert sich um eine Teilaufgabe. Dabei wird immer nur ein State ausgeführt. Die FSM ist dafür verantwortlich, die States zu wechseln.

Ein bekanntes Beispiel für die Verwendung einer FSM in einem Spiel ist der rote Geist Blinky in Pac Man. Die Aufgabe von ihm ist es Pac Man zu verfolgen. In der unteren Abbildung kann man sehen, wie das State-Diagramm für Blinky aussieht. Im Diagramm steht ein Rechteck für einen State. Die Verbindungen zwischen den States beschreiben die Regeln, die erfüllt werden müssen, damit die Finite State Machine in einen anderen State wechselt.

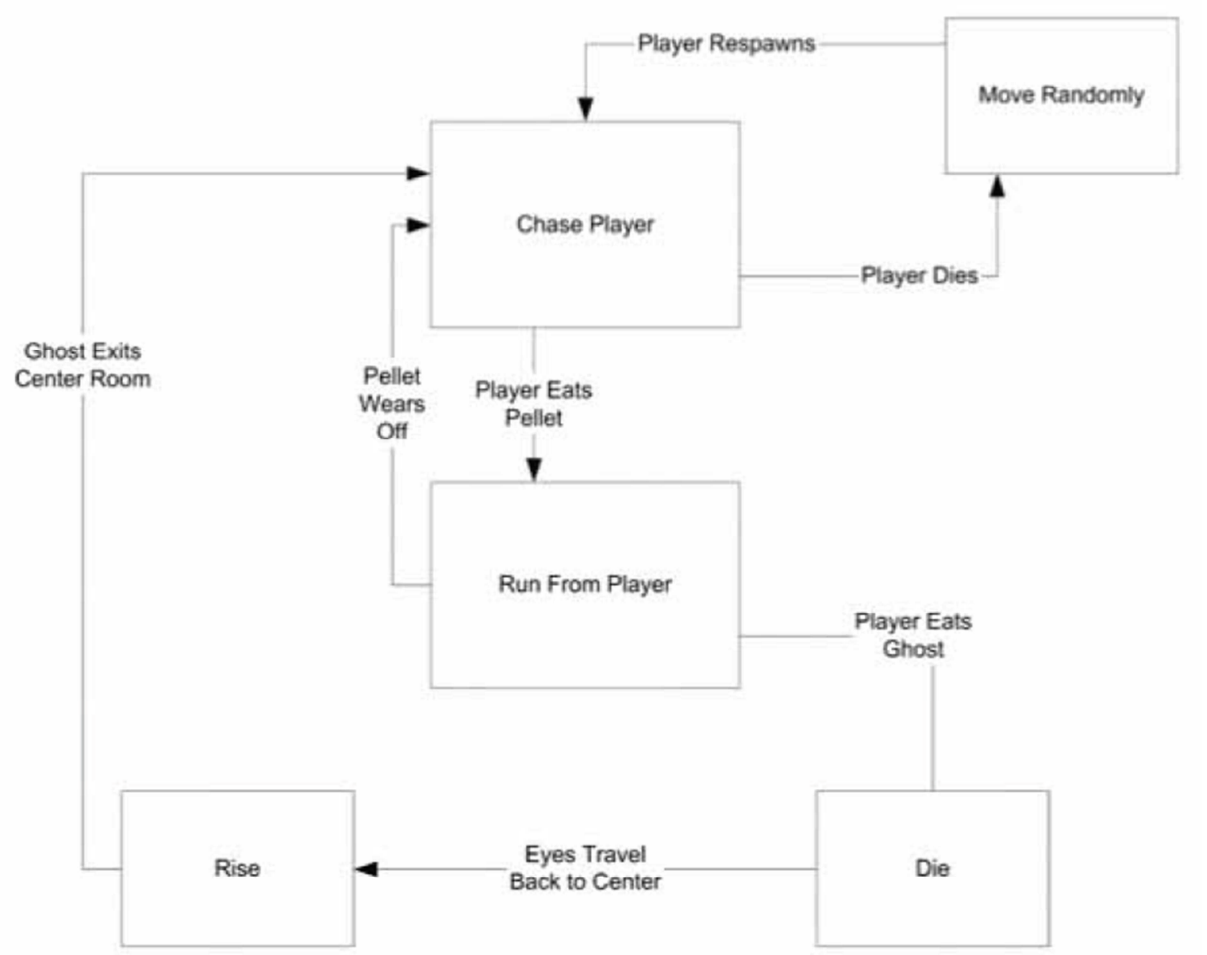


Abb. 2: State-Diagramm roter Geist von Pac Man (Schwab 2004, 243)

Der Normalzustand von dem Geist ist der Chase-Player-State, indem er den/die SpielerIn verfolgt. Sollte Pac Man sterben, wechselt die FSM in den Move-Randomly-State. Dieser ist so lange aktiv, bis der gelbe Nimmersatt wieder erscheint und die FSM wieder in den Chase-Player-State zurückwechselt. Wenn eine große Pille gegessen wurde, wird der Run-From-Player-State aktiv. In diesem State kann es passieren, dass der Geist stirbt. Ist das der Fall, wird der Die-State gestartet. In diesem State erscheinen die Augen des Geistes und fliegen in den mittleren Bereich des Bildschirmes. Ist dieser erreicht, erscheint der Geist wieder. Sobald dieser wieder am Leben ist und den mittleren Bereich verlassen hat, ist wieder der Chase-Player-State an der Reihe und die Verfolgung des/der Spielers/Spielerin wird wieder aufgenommen. Falls der Geist im Run-From-Player-State nicht getötet wird, wird wieder in den Chase-Player-State gewechselt, nachdem der Power-Up-Timer abgelaufen ist. (Schwab 2004, 243-244)

Die Finite State Machine Datenstruktur besteht aus drei Komponenten. Eine davon ist die State Komponente, in der die KI Logik ausgeführt wird. Die States werden alle in der Finite State Machine gespeichert. In der AIControl Klasse sind die FSM und die relevanten Informationen zum Entscheidungstreffen gespeichert. Dabei handelt es sich um den Speicher der Informationen aus der Spielwelt, der bereits im Kapitel *Entscheidungssystem für einen autonomen KI Gegner* vorgestellt wurde.

3.2.1. State

In der `State` Klasse wird die `StateID` gespeichert. Dabei handelt es sich um ein Enum, das den State bezeichnet. Über diese IDs ist es der FSM möglich, die States auszuwählen. Außerdem wird eine Referenz der AIControl Klasse gespeichert, da diese den Speicher von relevanten Informationen zur Spielwelt beinhaltet.

Das Grundgerüst einer `State` Klasse besteht zusätzlich aus einer `Enter()` und einer `Exit()` Funktion. Diese Methoden werden ausgeführt, wenn der State aktiviert beziehungsweise verlassen wird. Eine weitere Funktion ist `Update()`, in welcher sich die KI Logik befindet und überprüft wird, ob der aktuelle State verändert werden soll. Ist das der Fall, wird die ID vom gewünschten State zurückgegeben. Soll allerdings der aktuelle State beibehalten werden, ist die eigene `StateID` der Rückgabewert. Die letzte Funktion ist `Init()`, mittels welcher der State initialisiert wird. Normalerweise wird das vom Konstruktor übernommen, da die Implementierung dieser Arbeit in der Unity3D Game Engine umgesetzt wird und diese Game Engine keine Konstruktoren zulässt, übernimmt die `Init()` Funktion diese Aufgabe.

```
public class State {
    protected AIControl aiControl;
    protected StateType stateID;

    public virtual void Init(AIControl _aiControl)
    {
        aiControl = _aiControl;
    }
    public virtual void Enter() { }
    public virtual void Exit() { }
    public virtual StateType Update()
    {
        return stateID;
    }
}
```

3.2.2. FSM

In der Finite State Machine Klasse werden alle States gespeichert und verwaltet. Die States befinden sich dabei in einer Generic List namens `states`. Der aktuelle State und der standard

State werden in den Variablen `currentState` und `defaultState` festgehalten. Die FSM ist außerdem dafür verantwortlich, die States zu initialisieren. Durch die Funktionen `AddState()` und `RemoveState()` können States hinzugefügt und entfernt werden.

```
public class FSM : State {
    protected State defaultState;
    protected State currentState;

    protected List<State> states = new List<State>();


    public void AddState(State state)
    {
        states.Add(state);
    }
    public void RemoveState(State state)
    {
        states.Remove(state);
    }
    public virtual StateType UpdateMachine()
    {
        if (currentState == null)
            currentState = defaultState;

        if (currentState == null)
            return stateID;

        StateType newStateID = currentState.Update();

        if (newStateID != currentState.GetID())
        {
            currentState.Exit();
            currentState = GetStateFromID(newStateID);
            currentState.Enter();
        }
        return stateID;
    }
}
```

In der `UpdateMachine()` Funktion wird zuallererst überprüft, ob ein aktueller State gesetzt ist. Ist das nicht der Fall, wird der standard State zum aktuellen State. Wenn dieser ebenfalls nicht festgelegt wurde, wird die Funktion beendet. Sollte ein aktueller State vorhanden sein, wird dessen `Update()` Methode ausgeführt. Der Rückgabewert der `Update()` Funktion bestimmt, ob der aktuelle State gewechselt werden muss. Die Logik, ob der aktuelle State ausgetauscht werden soll, befindet sich in der State Klasse selbst. Wenn es eine Änderung geben soll, wird der `StateType` vom gewünschten State zurückgegeben. Soll der aktuelle State aktiv bleiben, liefert die `Update()` Funktion den `StateType` des aktuellen States.

Eine sehr wichtige Eigenschaft der FSM Klasse ist, dass diese von der State Klasse erbt. Einer der Vorteile der FSM Datenstruktur ist, dass die States einfach und übersichtlich organisiert werden können. Diese Übersicht kann jedoch sehr schnell verloren gehen, wenn es

zu viele States gibt. Beim vorherigen Beispiel mit dem roten Geist von Pac Man war das State Diagramm klar verständlich. Bei einem komplizierteren Spiel wie einem Echtzeit-Strategie-Spiel, gibt es hunderte verschiedene States. Um diesem Problem entgegen zu wirken, kann man die States hierarchisch verwalten. Das bedeutet, dass die States zuerst grob aufgeteilt werden. Diese States sind dann Finite State Machines, die selbst wieder untergeordnete States verwalten. Dadurch bleibt das KI System auch bei größeren Spielen übersichtlich und verständlich. (Orkin 2002, 32)

3.2.3. AIControl

Die `AIControl` Klasse hat drei Hauptkomponenten. Die erste ist die FSM, die in `AIControl` erstellt und gespeichert wird. Damit die KI in den States Entscheidungen treffen kann, werden Informationen zur Spielwelt benötigt. Der Speicher dieser Informationen wird ebenfalls in der `AIControl` Klasse gespeichert. Dafür sollte eine eigene `MemoryManager` Klasse erstellt werden, die alle nötigen Informationen gesammelt speichert. Zum Schluss werden noch die Eigenschaften zur Spielstärke der KI in der `AIControl` Klasse gespeichert. Welche Werte dafür gebraucht werden, wird später im Kapitel *Modellierung verschiedener Spielstärken des Computergegners* erklärt.

3.2.4. Probleme von FSM

Das Problem von Finite State Machines, dass zu viele States die Übersicht und Verständlichkeit des KI Konzepts zerstören, wurde bereits erwähnt. Auch die Lösung, States hierarchisch zu verwalten, wurde vorgestellt. Ein weiteres Problem sind die Regeln, nach denen die States gewechselt werden. Diese Bedingungen können als if then statements programmiert werden und treffen entweder zu oder nicht.

Eine Regel könnte zum Beispiel sein, dass ein/eine GegnerIn dem Spieler nicht näher als zehn Meter kommen darf. Ist das der Fall, beginnt der KI Spielcharakter zu fliehen. Dabei wechselt die Animation in eine Laufbewegung und er bewegt sich schneller. Wenn dann der Abstand von 10 Metern überschritten wird, bleibt die Figur wieder stehen. Sobald sich der/die KontrahentIn wieder nähert, setzt erneut die Fliehanimation ein. Das kann in einem Spiel sehr merkwürdig wirken, wenn sich der Charakter an den Grenzwerten zwischen zwei States bewegt. Um dieses Problem zu unterbinden, bietet es sich an, die FSM mit der Fuzzy Logic Methode zu kombinieren. (Schwab 2004, 267)

3.3. Fuzzy Logic

In der Fuzzy Logic geht es darum Entscheidungsmöglichkeiten zu finden. Dabei ist es möglich, dass mehrere Bedingungen für verschiedene Entscheidungen zur selben Zeit wahr sind. Diese Bedingungen sind allerdings zu verschiedenen Graden wahr. In der Fuzzy Logic entfernt man sich von der booleschen Logik, bei der eine Bedingung nur wahr oder falsch sein kann. In Zahlen ausgedrückt werden bei der booleschen Logik nur die Werte eins oder null genutzt. Bei der Fuzzy Logic kann eine Bedingung absolut wahr sein mit einem Wert von 1, absolut falsch mit dem Wert 0 oder teilweise wahr mit einem Wert zwischen 0 und 1.

Ein Vorteil von der Fuzzy Logic ist, dass reale Werte in Fuzzy Kategorien umgewandelt werden. Dabei wird eine Denkweise simuliert, die der menschlichen sehr ähnlich ist. Zum Beispiel ordnet der Mensch andere Personen nach deren Körpergröße in Kategorien wie klein, normal, groß und sehr groß ein. Dasselbe Prinzip verfolgt die Fuzzy Logic. Dadurch können in den Fuzzy Rules Wörter zur Beschreibung der Bedingungen genutzt werden. Mit dieser Methode wird das Regelwerk leichter lesbar und verständlicher. Außerdem ist es einfacher mit nicht technisch versierten Personen das KI Regelwerk auszuarbeiten. (Bourg and Seemann 2004, 257)

Bei den Fuzzy Kategorien ist es möglich, dass derselbe Wert in zwei verschiedene Kategorien fällt. Zum Beispiel kann eine Körpergröße von 1,90 Meter die Grenze zwischen den Kategorien groß und sehr groß sein. Für das menschliche Auge wäre ein Wert von 1,89 Meter nicht klar identifizierbar, in welche Kategorie diese Person eingeordnet werden sollte. Dasselbe Prinzip verfolgt die Fuzzy Logic, bei der es Grauzonen für die Zuordnung der Kategorien gibt. Eine Person mit der Größe von 1,89 Metern wäre zum Beispiel zu einem Grad von 0,55 in der Kategorie groß und zu einem Grad von 0,45 in der Kategorie sehr groß. Durch diese Grauzonen können die Übergänge zwischen zwei States viel weicher und abgestimmter wirken. Bei der wahr/falsch Logik kann es hingegen zu sehr abrupten Wechseln kommen. Für den/die SpielerIn wird die KI schwerer zu durchschauen, wenn der Wechsel nicht durch einen fixen Wert stattfindet, sondern variiert. (Bourg and Seemann 2004, 258)

Bei der Fuzzy Logic muss die KI nichts lernen. Die Regeln und Bedingungen müssen alle per Hand vor der Veröffentlichung des Spiels festgelegt und programmiert werden. Diese Regeln müssen allerdings nur ein einziges Mal gesetzt werden. Die Vorgangsweise der Fuzzy Logic kann in drei Schritte aufgeteilt werden. Der erste ist die Fuzzification, in welchem reale Zahlen aus der Spielwelt wie die Körpergröße in den Fuzzy Input umgewandelt werden. Dieser Fuzzy Input wird in die Fuzzy Rules eingesetzt, welche ein Fuzzy Output für jede einzelne

Regel erstellen. Der Fuzzy Output wird letztendlich durch die sogenannte Defuzzification in reale Zahlen zurückverwandelt, mittels denen wirkliche Aktionen in der Spielwelt ausgelöst und beeinflusst werden. (Bourg and Seemann 2004, 260)

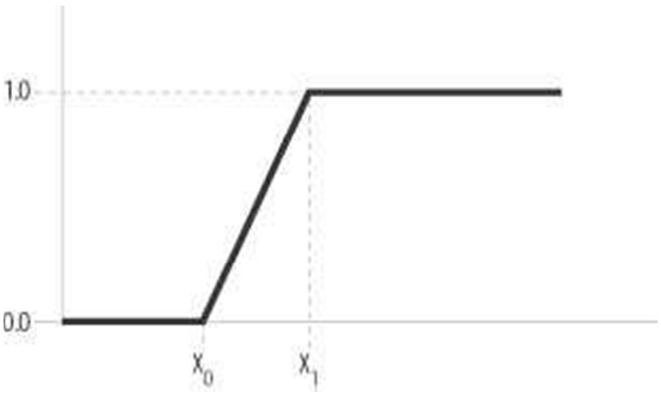
3.3.1. Fuzzification

Wie bereits erklärt wurde, werden reale Zahlen aus der Spielewelt in der Fuzzification Phase in Kategorien eingeordnet. Ein Beispiel dafür wäre, die Stärke eines Gegners in die Kategorien schwach, normal und stark zu unterteilen. Um zu überprüfen, in welche Kategorie der reale Wert passt, wird mittels einer Zugehörigkeitsfunktion ermittelt. Jede Fuzzy Kategorie besitzt eine eigene Zugehörigkeitsfunktion. Ein Rückgabewert von 0 bedeutet, dass der Wert nicht in diese Kategorie gehört, ein Wert von 1 heißt, dass er absolut in die Kategorie passt, und ein Wert zwischen 0 und 1 sagt aus, dass der Wert zu einem gewissen Grad in diese Kategorie gehört.

3.3.1.1. Gängige Zugehörigkeitsfunktionen

Theoretisch kann jeder Funktionsgraph verwendet werden. Die gängigsten Zugehörigkeitsfunktionen werden kurz vorgestellt. Bei den folgenden Funktionen handelt es sich um lineare Funktionen. Es ist auch möglich nicht lineare Funktionen zu verwenden. Damit man für eine Kategorie eine Zugehörigkeitsfunktion verwendet werden kann, muss ein Funktionstyp gewählt werden und für jeden x_i Punkt ein Wert bestimmt werden. Über diese Werte wird die Zugehörigkeitsfunktion konfiguriert. Die x-Achse zeigt die realen Zahlen an und die y-Achse den Grad, zu welchem der reale Wert in die Kategorie passt.

Stufenartige Funktion



Bei dieser Funktion sind die Werte der Kategorie immer 0 oder 1, es sei denn, der Wert befindet sich zwischen x_0 und x_1. Ist das der Fall, steigt der Wahrheitswert linear mit einem steigenden realen Wert an. Die mathematischen Bedingungen werden wie folgt beschrieben.

Abb. 3: Stufenartige Funktion (Bourg and Seemann 2004, 264)

$$f(x) = \begin{cases} 0; x \leq x_0 \\ \frac{x}{x_1 - x_0} - \frac{x_0}{x_1 - x_0}; x_0 < x < x_1 \\ 1; x \geq x_1 \end{cases}$$

Abb. 4: Stufenartige Funktion mathematische Bedingung (Bourg and Seemann 2004, 264)

Umgekehrte stufenartige Funktion

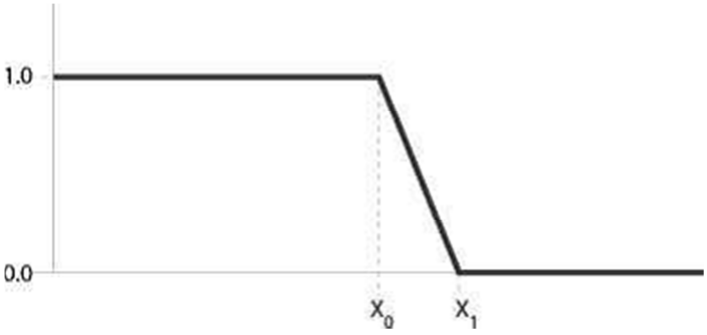


Abb. 5: Umgekehrte stufenartige Funktion (Bourg and Seemann 2004, 266)

Hierbei handelt es sich um die umgekehrte stufenartige Funktion. Der Wahrheitswert der Kategorie fällt mit einem ansteigendem realen Wert, wenn dieser zwischen x_0 und x_1 liegt. Bei der mathematischen Bedingung müssen nur kleine Änderungen durchgeführt werden.

$$f(x) = \begin{cases} 1; x \leq x_0 \\ \frac{-x}{x_1 - x_0} + \frac{x_1}{x_1 - x_0}; x_0 < x < x_1 \\ 0; x \geq x_1 \end{cases}$$

Abb. 6: Umgekehrte stufenartige Funktion mathematische Bedingung (Bourg and Seemann 2004, 266)

Dreieckige Funktion

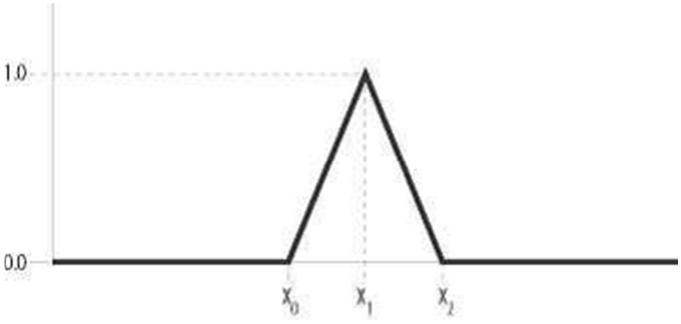


Abb. 7: Dreieckige Funktion (Bourg and Seemann 2004, 266)

Diese Funktion steigt zwischen x_0 und x_1 an. Bei einem Wert von x_1 wird der absolute Wahrheitswert von 1 erreicht. Dieser Wert wird bei einer solchen Funktion nur selten erreicht. Befindet sich der reale Wert zwischen x_1 und x_2 , fällt der Wahrheitswert mit einem steigenden realen Wert. Bei der mathematischen Funktion handelt es sich um eine Kombination aus den beiden vorherigen.

$$f(x) = \begin{cases} 0; x \leq x_0 \\ \dfrac{x}{x_1 - x_0} - \dfrac{x_0}{x_1 - x_0}; x_0 < x < x_1 \\ 1; x = x_1 \\ \dfrac{-x}{x_2 - x_1} + \dfrac{x_2}{x_2 - x_1}; x_1 < x < x_2 \end{cases}$$

Abb. 8: Dreieckige Funktion mathematische Bedingungen (Bourg and Seemann 2004, 265)

Trapezartige Funktion

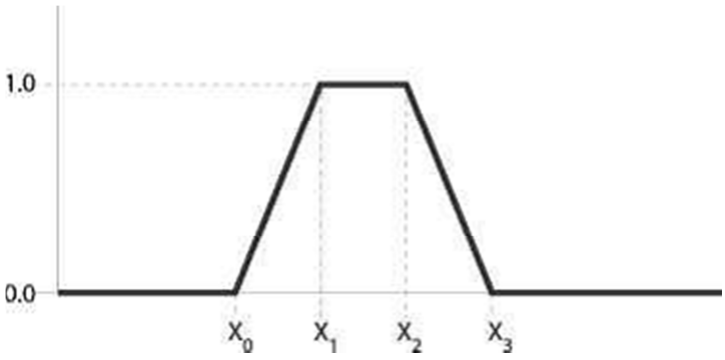


Abb. 9: Trapezartige Funktion (Bourg and Seemann 2004, 266)

Bei dieser Funktion ist es um einiges einfacher einen Wahrheitswert von 1 zu erreichen als bei einer dreieckigen Funktion. Befindet sich der reale Wert im Bereich von x_1 bis x_2 , wird ein Wahrheitswert von 1 erreicht. Liegt der Wert zwischen x_0 und x_1 beziehungsweise x_2 und x_3 steigt oder fällt der Wahrheitswert. Über x_3 oder unter x_0 passt der Wert nicht in diese Kategorie. Die mathematische Bedingung für diese Funktion sieht folgendermaßen aus:

$$f(x) = \begin{cases} 0; x \leq x_0 \\ \frac{x}{x_1 - x_0} - \frac{x_0}{x_1 - x_0}; x_0 < x < x_1 \\ 1; x_1 \leq x \leq x_2 \\ \frac{-x}{x_3 - x_2} + \frac{x_3}{x_3 - x_2}; x_2 < x < x_3 \end{cases}$$

Abb. 10: Trapezartige Funktion mathematische Bedingung (Bourg and Seemann 2004, 267)

3.3.1.2. Fuzzy Kategorie bestimmen

Wenn für jede mögliche Kategorie eine Funktion bestimmt wurde, kann man zur Visualisierung der Kategorien die Funktionen in einem Graphen vereinen.

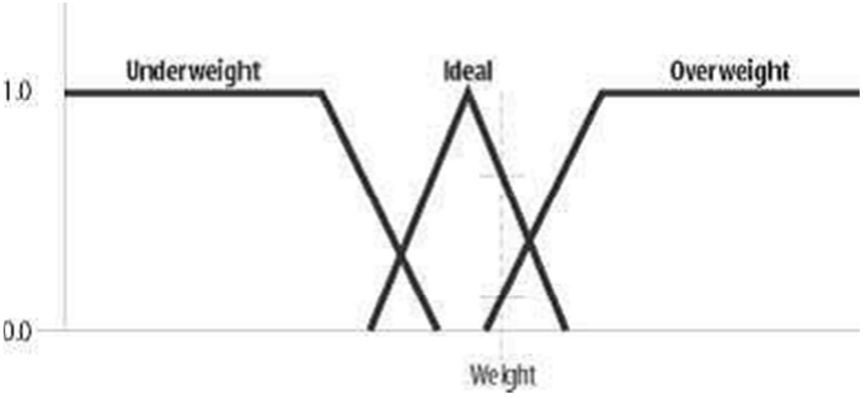


Abb. 11: Alle Zugehörigkeitsfunktionen in einem Graphen (Bourg and Seemann 2004, 265)

Wie man im Graphen erkennen kann, überschneiden sich die Wertebereiche der einzelnen Kategorien. Diese Überschneidungen sind die Grauzonen. Gibt es mehr Überschneidungen, ist die Fuzzy Logic verschwommener. Wenn keine Überschneidungen existieren, ergibt sich eine schärfere Logik. Eine allgemein akzeptierte Regel besagt, dass jede Kategorie den Wertebereich seines Nachbars um circa 25 % überschneiden soll. (Bourg and Seemann 2004, 268) Durch das Einsetzen des realen Wertes in die einzelnen Funktionen bekommt man den Wahrheitsgrad, zu welchem der reale Wert in die jeweilige Kategorie passt. Dieser Wahrheitswert wird auch Fuzzy Input genannt, welcher den Input für die Fuzzy Rules darstellt.

3.3.2. Fuzzy Rules

Wenn die realen Zahlen in Fuzzy Input Werte umgewandelt wurden, können diese Zahlen in die aufgestellten Fuzzy Rules eingesetzt werden. Dabei wird bestimmt, welche Aktion die KI auslösen soll. Wie es bereits bei den Fuzzy Kategorien der Fall war, können auch bei den Fuzzy Rules mehrere zur gleichen Zeit wahr sein. Jede Regel ist allerdings wieder nur zu einem bestimmten Grad wahr.

Um Bedingungen für die Regeln aufstellen zu können, brauchen wir logische Operatoren, mit denen wir den Fuzzy Input logisch verknüpfen können. Die logischen Operatoren liefern dabei je nach Bedingung für den Fuzzy Input einen bestimmten Fuzzy Output. Für die Operatoren werden AND, OR und NOT festgelegt. (Bourg and Seemann 2004, 268)

Beim AND Operator werden zwei Fuzzy Inputs verglichen und der kleinere der beiden Werte genommen. Der OR Operator liefert das Maximum von zwei Fuzzy Input Werten. Der invertierte Fuzzy Wert wird vom NOT Operator zurückgegeben, indem der Fuzzy Input von 1 subtrahiert wird. (Bourg and Seemann 2004, 269)

Bei der booleschen Logik werden meistens Bedingungen aufgestellt wie zum Beispiel, wenn A und B wahr sind, dann führe C aus. Mit der Fuzzy Logic ist das nicht so einfach, da wir keine absoluten wahr oder falsch Ergebnisse erhalten. Daher müssen wir jede mögliche Aktion berechnen. Das Ergebnis drückt für alle Regeln einen Wahrheitswert aus. Zum Beispiel können wir in einem Spiel die möglichen Aktionen angreifen, fliehen oder nichts machen, ausführen. Diese Aktionen werden von zwei Variablen beeinflusst, der Distanz zum Gegner und die Stärke des Gegners. Diese zwei Variablen sind reale Werte aus der Spielwelt. Diese müssen zuerst mittels Zugehörigkeitsfunktionen in Fuzzy Input umgewandelt werden. (Bourg and Seemann 2004, 270)

Für die Distanz gibt es die Fuzzy Kategorien nahe, weit und sehr weit und für die Stärke schwach, normal und stark. Für jede dieser sechs Fuzzy Kategorien gibt es einen Fuzzy Input Wert zwischen 0 und 1. Für jede Aktion werden nun Regeln mittels der logischen Operatoren aufgestellt. Diese könnten zum Beispiel folgendermaßen aussehen.

```
angreifen = if(nahe AND NOT stark);
nichtsMachen = if( (NOT nahe) AND stark);
fliehen = if( (NOT sehr weit) AND stark);
```

Diese Regeln liefern als Ergebnis den Fuzzy Output. Dieser könnte folgende Werte haben. (Bourg and Seemann 2004, 271)

```
angreifen = 0.2;
nichtsMachen = 0.4;
fliehen = 0.7;
```

Um eine Aktion auszuwählen, kann die mit dem höchsten Wert genommen werden. In manchen Fällen reicht es nicht, sich nur für eine Aktion zu entscheiden. Der Fuzzy Output soll die Aktion beeinflussen können. Zum Beispiel soll der Fuzzy Output beim Fliehen bestimmen, ob der KI Charakter davon geht oder läuft. Das kann durch die Defuzzification realisiert werden.

3.3.3. Defuzzification

Für alle möglichen Aktionen sind Werte festgelegt, die als Fuzzy Output bezeichnet werden. Die ausgewählte Aktion soll durch den Fuzzy Output beeinflusst werden. Deshalb müssen wir diesen Wert in eine reale Zahl umwandeln. Im vorherigen Beispiel wurde die Aktion fliehen mit einem Wert von 0,7 gewählt, angreifen mit 0,2 und nichts machen mit 0,4.

Eine Methode um den Fuzzy Output in eine reale Zahl zu transformieren ist, dass wieder Zugehörigkeitsfunktionen verwendet werden. Jede möglichen Aktionen besitzt eine eigene Zugehörigkeitsfunktion.

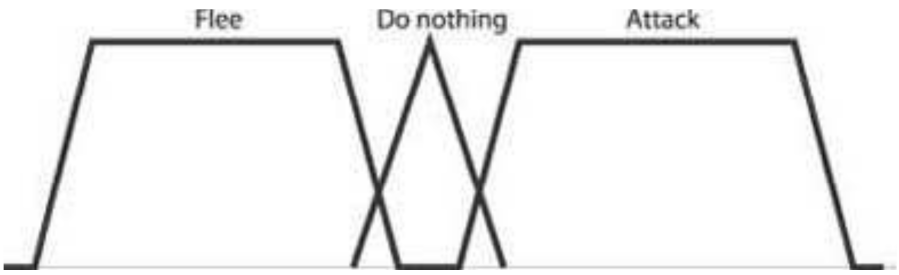


Abb. 12: Alle Zugehörigkeitsfunktionen in einem Graphen (Bourg and Seemann 2004, 271)

Diese werden danach auf die Maximalwerte vom Fuzzy Output gekürzt und von der gekürzten Fläche wird der Schwerpunkt berechnet.

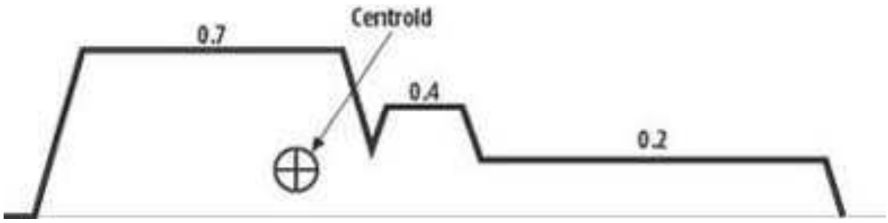


Abb. 13: Gekürzte Zugehörigkeitsfunktionen (Bourg and Seemann 2004, 272)

Da diese Methode sehr rechenaufwändig ist und die Algorithmen zum Entscheidungstreffen sehr oft ausgeführt werden, wird eine vereinfachte Methode genutzt. Diese ist die Singleton Output Zugehörigkeitsfunktion. Für jede Aktion wird eine reale Maximalzahl festgelegt. Im aktuellen Beispiel könnte man für die Geschwindigkeit des Flüchtens einen Maximalwert von -10, fürs Angreifen einen Wert von 10 und fürs nichts machen einen Wert von 1 festlegen. Das würde bedeuten, der Charakter kann maximal mit 10 Zentimeter pro Sekunde flüchten. Unter Berücksichtig der Maximalwerte und dem Fuzzy Output der Aktionen wird ein gewichteter Mittelwert errechnet. Der Mittelwert ist daher gewichtet, da bei dieser Kalkulation nicht nur der Fuzzy Output der gewählten Aktion ausschlaggebend ist, sondern die reale Zahl wird in Relation zu allen Werten des Fuzzy Outputs berechnet. (Bourg and Seemann 2004, 272)

$$output = \frac{\sum_{i=1}^{n} \mu_i x_i}{\sum_{i=1}^{n} \mu_i}$$

Abb. 14: Formel des Singleton Outputs (Bourg and Seemann 2004, 272)

Dabei werden alle Fuzzy Outputs, μ_i, mit den Maximalwerten der jeweiligen Aktion, x_i, multipliziert und anschließend die Ergebnisse aller Multiplikationen addiert. Diese Summe davon wird durch die Summe aller Fuzzy Outputs, μ_i, dividiert. Im aktuellen Beispiel würde die Rechnung einen Wert von -2,5 ergeben. (Bourg and Seemann 2004, 272)

$$Output = \frac{[(0{,}7)(-10) + (0{,}4)(1) + (0{,}3)(10)]}{0{,}7 + 0{,}4 + 0{,}3} = -2{,}5$$

Abb. 15: Beispiel Singleton Output (Bourg and Seemann 2004, 272)

Dieser Wert würde festlegen, dass die Spielfigur mit 2,5 cm pro Sekunde flüchtet.

4. Modellierung verschiedener Spielstärken des Computergegners

Ein/eine SpielerIn soll gegen die KI antreten können. Nicht alle ComputerspielerInnen beherrschen ein Videospiel gleichermaßen gut. Daher ist es wichtig, dass die Stärke der KI sowohl für einen Neuling als auch für einen Profi eine Herausforderung darstellt. Der ein-

fachste Schritt ist, verschiedene Spielstärken der KI zu erstellen wie schwach, fortgeschritten und experte.

Um verschiedene Spielstärken gewährleisten zu können, muss die KI über verschiedene, variable Eigenschaften gesteuert werden. Diese Eigenschaften können entweder feste Werte sein oder ein Wertebereich, aus dem per Zufall ein Wert gewählt wird. Ein Wertebereich ist für die KI besser geeignet, weil dabei die Ergebnisse variieren können und die Aktionen dadurch abwechslungsreicher sind. (Scott 2002b, 287) Wenn die einzelnen Eigenschaften der KI variabel gehalten werden, können für die jeweilige Spielstärke andere Wertebereiche festgelegt werden.

Zum Beispiel kann die Reaktionszeit der KI eine solche Eigenschaft sein. Eine kürzere Reaktionszeit kann den KI Gegner stärken und eine längere das Gegenteil bewirken. Andere Eigenschaften, die als Stellschrauben für die Stärke der Maschine dienen können, sind die Zielgenauigkeit oder die Geschwindigkeit einer Einheit.

4.1. Wertebereiche für Eigenschaften

Indem wir nicht einen linearen Wertebereich für unser Zufallsprinzip verwenden, sondern eine bestimmte Funktion wählen, können wir die Genauigkeit beziehungsweise die Fehlerquote und damit die Spielstärke des Computerspielers beeinflussen.

Nehmen wir als Beispiel einen First-Person-Shooter, in dem die KI immer auf die Brustmitte des Gegners zielt. Die Abweichung des Zielens wird bestimmt, indem ein maximaler Abweichungswert `maxDeviation` mit dem Zufallsrückgabewert `randomOutput` multipliziert wird. Das Ergebnis wird anschließend zu den x, y und z Koordinaten der Brustmitte addiert. Der Zufallsrückgabewert `randomOutput` wird errechnet, indem eine Zufallszahl zwischen 0 und 1 der Funktion `float getDeviationFactor()` übergeben wird. Je nachdem ob diese Methode mit einer linearen Funktion oder mit einer quadratischen Funktion den `randomOutput` Wert bestimmt wird, variiert die Wahrscheinlichkeit, dass der/die GegnerIn getroffen wird.

Wir gehen in diesem Beispiel davon aus, dass eine Abweichung von 0 bis 20 Prozent ein kritischer Treffer, eine Abweichung von 21 bis 40 Prozent ein schlechter Treffer und eine Abweichung von über 40 Prozent ein Fehlschuss ist. Mit diesen Werten können wir überprüfen, wie wahrscheinlich die einzelnen Treffermöglichkeiten bei einer linearen beziehungsweisen quadratischen Funktion sind.

Bei einer linearen Funktion trifft die KI zu einer Wahrscheinlichkeit von 20 Prozent einen kritischen Treffer. Auch ein schlechter Treffer wird von 20 Prozent der Schüsse getroffen. Daraus folgt, dass der Computer in 60 Prozent der Fälle den gegnerischen Körper verfehlt.

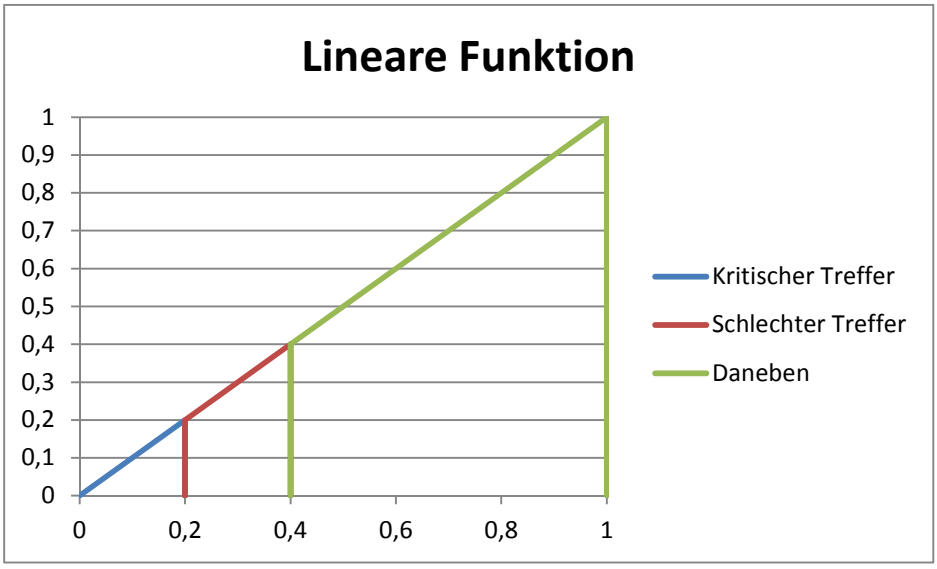


Abb. 16: Wahrscheinlichkeit der Treffermöglichkeiten bei einer linearen Funktion

Um einen akkurateren und stärkeren Gegner zu erschaffen, reicht es zum Beispiel schon, die lineare gegen eine quadratische Funktion auszutauschen. Dadurch würde die KI ca. 45 Prozent der geschossenen Kugeln als kritischen Treffer verwerten. Immer noch 18 Prozent der Schüsse würden einen schlechten Treffer verursachen und es würde nur mehr in 37 Prozent der Fälle danebengeschossen. Um die Genauigkeit der Maschine zu verändern, können natürlich alle möglichen Funktionen verwendet werden.

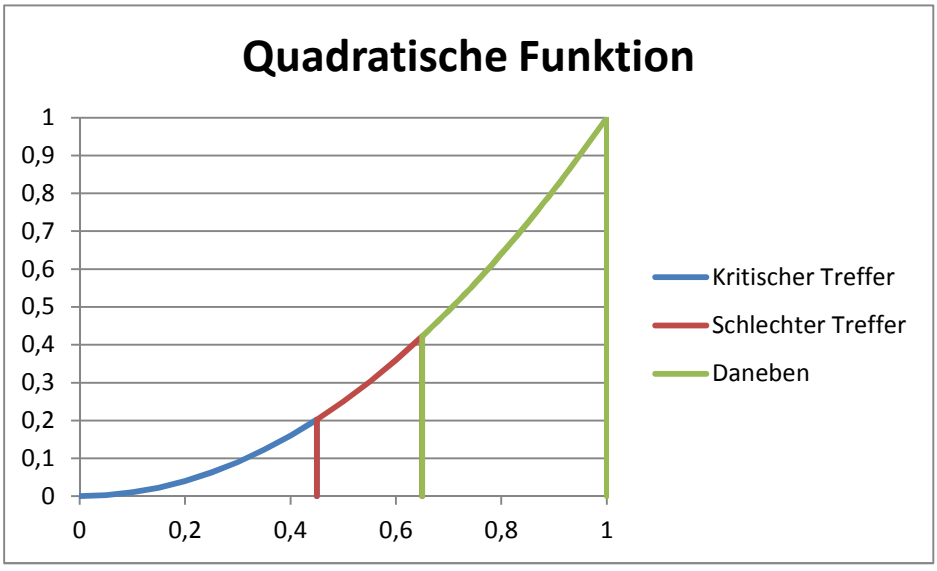


Abb. 17: Wahrscheinlichkeit der Treffermöglichkeiten bei einer quadratischen Funktion

4.2. Adaptive KI Stärke

Bei einem simplen Spielprinzip ist es sehr einfach einen Computergegner zu modellieren, der immer gewinnt oder immer verliert. Dabei werden die Eigenschaften der KI entweder extrem schlecht oder hervorragend festgelegt. Um die passenden Werte für die jeweilige Stärke zu finden, muss sehr viel getestet und probiert werden.

Die richtige Herausforderung beim Modellieren der Stärke ist eine KI zu entwickeln, die bei jedem Spiel einen knappen Ausgang gewährleistet. Für einen/eine SpielerIn ist es viel spannender bis zum Ende kämpfen zu müssen um das Spiel zu gewinnen, als gegen einen weit unterlegenen Kontrahenten zu gewinnen. Um diese Anforderung erfüllen zu können, müssen sich die Werte für die Eigenschaften der KI während des Spiels dynamisch verändern können. Wenn der/die SpielerIn zum Beispiel am Verlieren ist, sollte die Stärke des virtuellen Spielers abbauen, beziehungsweise falls der/die menschliche KontrahentIn am Gewinnen ist, stärker werden. Diese Veränderungen an den Werten müssen allerdings für den/die GegnerIn glaubhaft sein. Das heißt der Computer darf sich nur minimal steigern. Er soll nicht von einem Moment auf den anderen von einem schlechten Schützen zu einer fehlerfreien Killermaschine werden. Obwohl dadurch gewährleistet werden könnte, dass der Ausgang des Matches knapp ist, würde es für den/die Beteiligten/Beteiligte keinen Spaß machen. Daher ist es wichtig, dass sich die Werte minimal über einen längeren Zeitraum verändern. Außerdem muss es Begrenzungen für die Wertebereiche der einzelnen Eigenschaften geben.

4.2.1. Betrügen

Eine Möglichkeit um das Spiel ausgeglichen zu halten ist, dass der Computer betrügen kann. Unter Betrügen versteht man das Gebrauchen von unerlaubten Hilfsmitteln wie Informationen, die ein/e menschlicher/menschliche Spieler/In nicht wissen kann. Diese Informationen können zum Beispiel die Position von dem/der GegenspielerIn sein, ohne dass er/sie gesehen oder gehört wurde.

Wie beim Verändern der Stärke soll der/die BenutzerIn vom Betrügen nichts merken. Das Betrügen ist unter ComputerspielerInnen sehr verpönt, allerdings kommt es auf die Definition von Betrügen an. SpielerInnen haben oft eine andere Ansicht als die Entwickler, was genau Betrügen ist. So wird zum Beispiel toleriert, dass die KI das Geländer der Spielwelt vor dem Spielstart scannt. Obwohl die Maschine dadurch einen Vorteil erhält, wird es akzeptiert, da dadurch zum Beispiel Performanceprobleme behoben werden können. (Scott 2002a, 18) Es

wird eine betrügende KI einer extrem schlecht spielenden KI bevorzugt, solange das Betrügen für den/die BenutzerIn nicht offensichtlich ist. (Scott 2002a, 19)

4.2.2. Methoden zu adaptiver Spielstärke

Für den Implementierungsteil dieser Bachelorarbeit wird eine Methode zur Veränderung der Spielstärke der KI entwickelt, die sich auf ein Spielniveau des/der menschlichen GegnerIn einpendelt. Die Eigenschaften der KI sollen sich dabei nicht laufend steigern oder verschlechtern, sondern es soll ein ähnliches Niveau gefunden werden, auf welchem die KI konstant für das restliche Match bleibt. Für diese Methode wird angenommen, dass ein/eine SpielerIn während einer Partie mit einer ungefähr gleichbleibenden Stärke spielt.

In den ersten paar Sekunden nach dem Spielstart wird versucht, schnell die richtigen Eigenschaften für die KI zu finden. Nachdem diese Einstellungen gefunden wurden, soll nur noch durch minimale Änderungen für einen knappen Ausgang des Spiels gesorgt werden.

Diese Methode baut auf zwei bestehende Konzepte der dynamischen Adaption der KI Stärke auf. Das erste Konzept heißt Rapidly Adaptive Game AI. (Bakkes, Spronck, and van den Herik 2010) Beim zweiten Konzept handelt es sich um Dynamic Scripting Technik. (Postman, Sprinkhuizen-Kuyper, and Spronck 2004)

4.2.2.1. Rapidly Adaptive Game AI

Bei der Rapidly Adaptive Game AI wird die Spielwelt durchgehend beobachtet. Sobald sich etwas verändert, werden die neuen Informationen an den Adaptionsmechanismus weitergegeben. Dieser überprüft, ob sich aufgrund der neuen Informationen die KI Stärke ändern soll. Ist das der Fall, werden die Eigenschaften der KI angepasst. (Bakkes, Spronck, and van den Herik 2010)

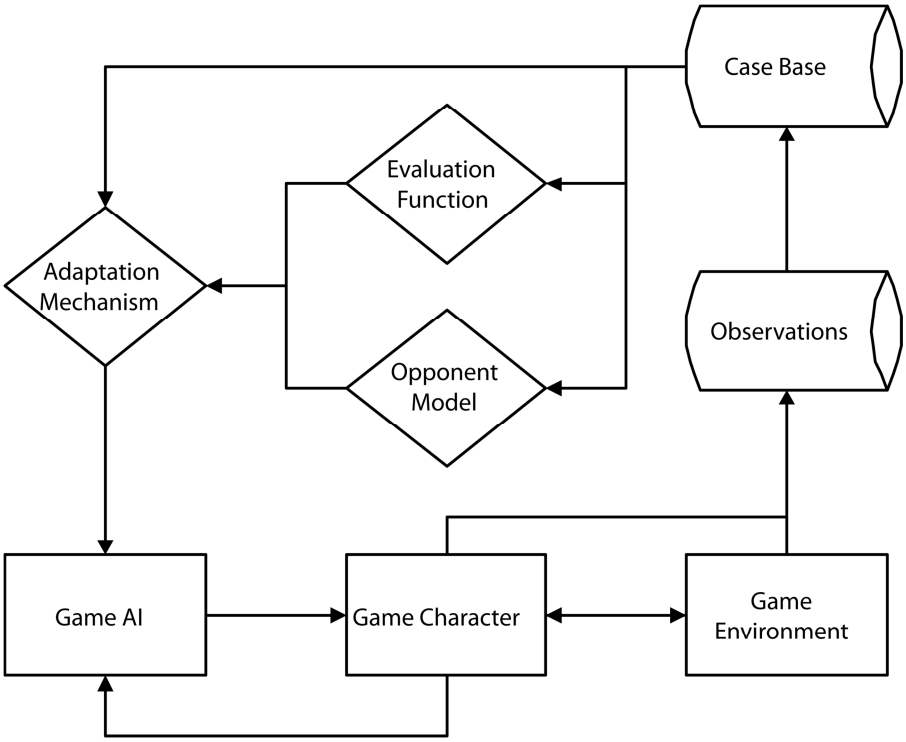


Abb. 18: Rapidly Adaptive Game AI (Bakkes, Spronck, and van den Herik 2010, 2)

4.2.2.2. Dynamic Scripting

Die Dynamic Scripting Technik wird verwendet, um die Spielstärke der KI im Laufe von mehreren Matches zu verändern. Die KI hat bei diesem Ansatz vordefinierte Taktiken, die alle eine dynamische Gewichtung besitzen. Die Gewichtung ändert sich immer nach einem Spiel. Die KI lernt nach einem Sieg, dass die verwendeten Taktiken erfolgreich waren oder nach einer Niederlage, dass diese Taktiken schlechter gewichtet werden müssen. Die neue Gewichtung der Taktik bestimmt, ob sie eingesetzt wird. Um sie zu berechnen gibt es drei verschiedene Methoden. Diese sind High-Fitness Penalising, Weight Clipping und Top Culling. Für die adaptive Technik, die die KI Stärke auf ein konstantes Niveau einpendelt, wird auf die Weight Clipping Methode aufgebaut. Bei dieser Methode werden ein Anfangswert und ein Minimal- beziehungsweise Maximalwert festgelegt. Zusätzlich wird bestimmt, um wie viel sich die Gewichtung nach einem Spiel verändern darf. Die aktuelle Gewichtung kann das Maximum oder Minimum nicht überschreiten beziehungsweise unterschreiten. (Postman, Sprinkhuizen-Kuyper, and Spronck 2004)

4.2.2.3. Methode zur einpendelnden adaptiven KI Stärke

Für diese Methode, wird das Verwaltungssystem des Rapidly Adaptive Game AI Konzepts verwendet. Das System wird minimal geändert, indem die Veränderungen nicht sofort an den

Adaptionsmechanismus weitergegeben werden. Der Adaptionsmechanismus holt sich diese laufend in regelmäßigen Abständen. Ein Timer steuert die Intervalle, in denen die Stärke angepasst wird.

Wie bereits erwähnt wurde, wird für die Methode, bei der sich die Stärke der KI auf das Spielerniveau anpasst, auf das Weight Clipping Konzept aufgebaut. Dieses Konzept wird abgeändert, damit sich die Spielstärke der KI während einem Spiel auf den/die GegenspielerIn einpendeln kann. Dazu wird ein Wertebereich für die Eigenschaften festgelegt, die die Stärke der KI Steuern. Die Werte dieser Eigenschaften können sich während eines Matches dynamisch im Rahmen des festgelegten Wertebereichs verändern.

Ein Beispiel für eine solche Eigenschaft ist die Zielabweichung bei einem Shooter. Diese Eigenschaft bestimmt, wie zielsicher die Maschine schießt. Durch einen Maximal- und Minimalwert wird ein Wertebereich definiert. Der Startwert ist der Mittelwert der beiden Beschränkungen.

Bei der einpendelnden Methode ist vor allem wichtig, dass so schnell wie möglich nach Matchbeginn eine ähnliche Stärke, wie der/die menschliche KontrahentIn besitzt, gefunden wird. Deshalb wird die Weight Clipping Methode mit einer Art binären Suche kombiniert. Um das richtige Niveau zu finden, wird die Zielabweichung entweder erhöht, wenn die KI besser ist oder verringert, falls der/die GegnerIn mehr Können zeigt.

Die Suche nach der richtigen Stärke soll der binären Suche ähnlich sein, indem zu Beginn der Suche mit großen Sprüngen der Eigenschaftswert verändert wird. Es kann nur eine ähnliche Form der binären Suche verwendet werden, da diese Sprünge nicht zu groß sein dürfen. Das Geschick der KI darf sich nämlich nicht in kurzer Zeit extrem verändern. Sollte der/die BenutzerIn die Veränderung der Fähigkeit bemerken und nicht nachvollziehen können, wird er/sie sich betrogen fühlen. (Scott 2002a, 18) Zu Beginn wird ein Betrag bestimmt, um den sich die Eigenschaft verändern kann. Dieser Wert definiert, um wie viel sich die Fähigkeit maximal steigern beziehungsweise verschlechtern kann. Um extreme Sprünge zu verhindern, muss dieser Wert dementsprechend gewählt werden.

Vorgangsweise

Bei dieser Methode wird angenommen, dass die Stärke des/der menschlichen Spielers/Spielerin konstant für die Dauer eines Matches ist. Diese Stärke ist in der Abbildung darunter durch die rote Linie dargestellt. Der grüne Strich stellt die Stärke der KI dar, die sie zu Beginn des Spiels hat. Beim ersten Mal messen der Differenz der beiden Spieler wird fest-

gestellt, dass die KI schlechter als der/die menschliche KontrahentIn ist. Daher wird der festgelegte Veränderungswert zur Eigenschaft addiert. Bei der zweiten Überprüfung ist die Maschine immer noch schlechter und der Fähigkeitswert wird noch einmal erhöht. Nun hat das Spielniveau des Computers den Menschen überholt. Daher entscheidet sich nach dem dritten Mal messen, dass nicht mehr addiert sondern subtrahiert werden muss.

Da die Richtung der Veränderung gewechselt hat, beginnt eine neue Phase. Bei einem Phasenwechsel wird der Veränderungswert halbiert. Das kann man in der Abbildung bei Phase 2 erkennen, da der dunkelrote Pfeil deutlich kleiner ist als der blaue Pfeil, der die Steigerung der Fähigkeit ausgedrückt hat.

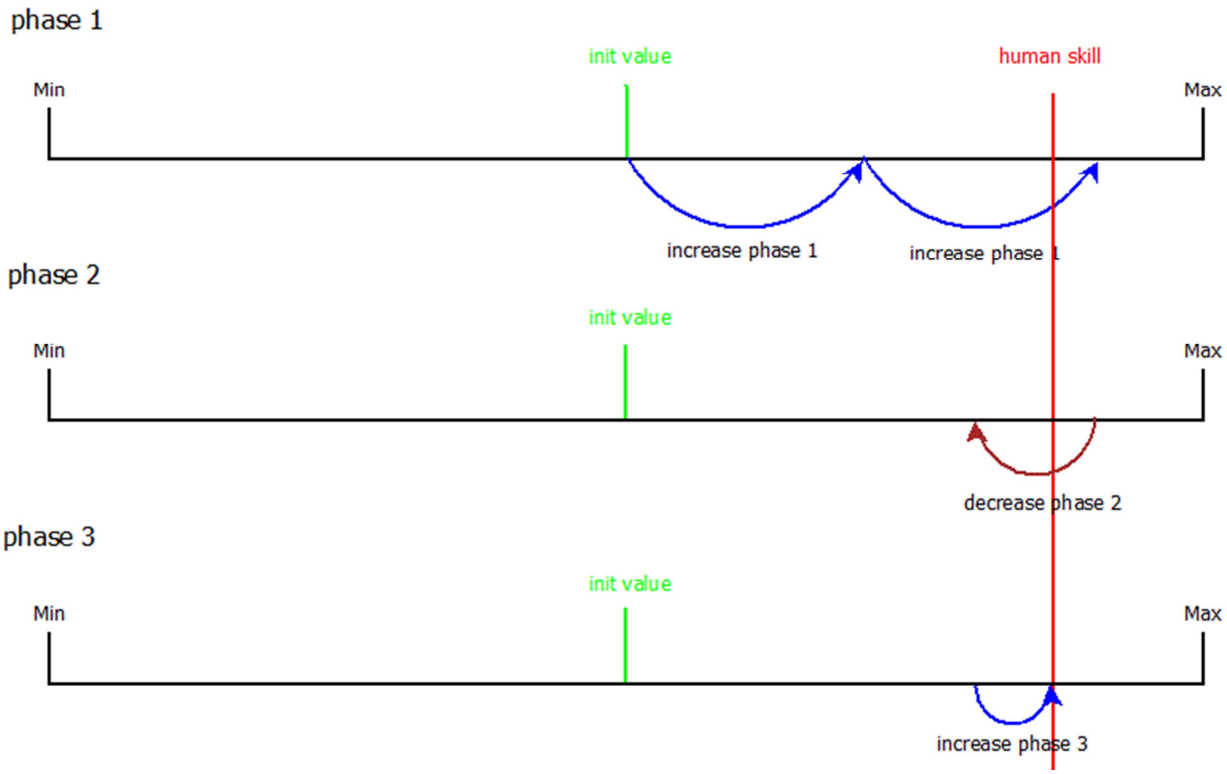


Abb. 19: Vorgangsweise bei der adaptiven Methode

Nach dem Subtrahieren wird erkannt, dass der/die SpielerIn wieder ein höheres Spielniveau hat. Deshalb muss wieder die Richtung geändert und in eine neue Phase gewechselt werden. Dabei wird auch der Veränderungswert wieder halbiert. Nach dem letzten Schritt hat die Stärke der KI eine ähnliche wie die des/der lebenden KontrahentenIn erreicht.

Der Veränderungswert soll nicht beliebig vermindert werden können, deshalb wird ein Mindestwert festgelegt, den er nicht unterschreiten darf. Sollte er nämlich zu klein werden, kann

im Verlaufe des Matches nicht mehr auf Veränderungen der menschlichen Spielstärke reagiert werden. Obwohl angenommen wird, dass die Stärke konstant bleibt, kann es dennoch passieren, dass sich der Spieler minimal steigert oder verschlechtert. Um für diese Fälle gerüstet zu sein, gibt es den festgelegten Minimalwert. Dieser gewährleistet, dass die KI auch später im Verlaufe eines Matches noch schnell genug auf Veränderungen reagieren kann.

5. Implementierung

In der Implementierung sollen die vorgestellten KI Methoden genutzt werden, um für ein konkretes Spiel ein Konzept eines KI Gegners zu erstellen. Das Konzept soll dabei für das bestehende Spiel Piratenkampf ausgearbeitet werden und mit der Game Engine Unity3D und der Programmiersprache C# Script umgesetzt werden. Um zu verstehen, welche Aktionen ausgeführt werden können und welche Entscheidungen die KI treffen kann, wird das Spiel Piratenkampf kurz vorgestellt.

5.1. Piratenkampf

Bei Piratenkampf handelt es sich um ein Mobilegame, das man über das Netzwerk gegen seine Freunde/Freundinnen spielen kann. Im Spiel schlüpft man in die Rolle eines Piraten, der auf einer einsamen Insel gestrandet ist. Die einzigen Dinge, die er retten konnte, sind ein kleines Boot, eine Kanone und einige Holzfässer. Zu allem Überfluss ist der Erzfeind auf einer gegenüberliegenden einsamen Insel gestrandet. Das Ziel ist nun, die gegnerische Insel zu versenken, bevor die eigene Insel zerstört wird.

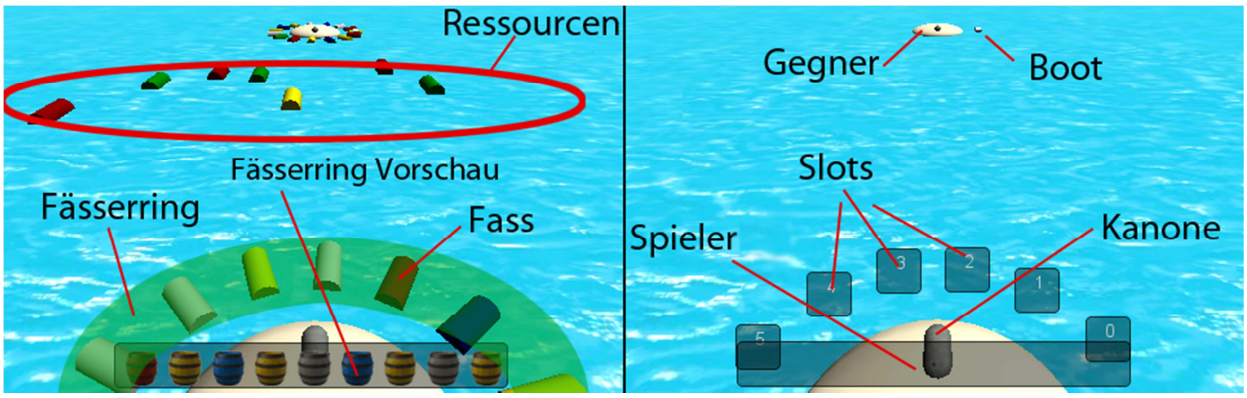


Abb. 20: Piratenkampf

Der/die SpielerIn hat nun zwei Hauptaufgaben. Zum einen muss er/sie versuchen mit Hilfe von den Fässern, die vor der eigenen Insel liegen, den/die GegnerIn anzugreifen. Zum anderen hat das Boot die Aufgabe weitere Fässer einzusammeln, die von den versenkten Piratenschiffen zwischen den beiden Inseln auftauchen.

5.1.1. Angriff

Es existieren mehrere Möglichkeiten um anzugreifen. Es gibt fünf verschiedene Fasstypen. Platziert man drei Fässer vom selben Typen nebeneinander im Ring um die Insel, wird je nach Fasstyp eine andere positive Kombination ausgelöst.

Eine andere Möglichkeit wie man dem/der FeindIn schaden kann ist, wenn man ein Fass in die Kanone lädt und versucht, bei der gegnerischen Insel eine negative, dreier Kombination auszulösen. Das gelingt, indem man das geladene Fass zwischen zwei Fässer mit dem gleichen Typ schießt. Fasstyp abhängig werden dabei verschiedene Aktionen ausgelöst.

5.1.2. Fasstypen

Wie bereits erwähnt, gibt es für jede Dreierkombination sowohl einen positiven als auch einen negativen Effekt. Der positive wird gestartet, wenn man eine Kombination bei seiner eigenen Insel auslöst. Ein negativen Effekt wird tritt auf, wenn man bei der gegnerischen Insel drei gleiche Fässer kombiniert.

5.1.2.1. Offensivfass

Schafft man es, drei offensiv Fässer vor der eigenen Insel nebeneinander zu platzieren, wird eine positive Kombination ausgelöst und eine Kanonenkugel mit dem Inhalt der Fässer gefüllt. Diese Kugel wird automatisch in die Kanone geladen. Durch einen platzierten Schuss ist es nun möglich entweder Fässer von dem/der GegnerIn zu zerstören oder seine Insel direkt anzugreifen. Durch einen direkten Angriff werden der Insel Lebenspunkte abgezogen.

Ist es dem Angreifer allerdings gelungen, dem/der ErzfeindIn ein offensives Fass bei der gegnerischen Insel zwischen zwei Offensivfässer zu schießen, wird eine negative Kombination ausgelöst. Dabei explodiert diese Kombination und fügt der Insel Schaden zu.

5.1.2.2. Defensivfass

Wird eine positive Kombination von Defensivfässern bei der eigenen Insel ausgelöst, wird die Verteidigung durch Materialien aus den Fässern verstärkt und die Angriffe richten weniger Schaden an. Bei einer negativen Kombination aus Defensivfässern bei der gegnerischen Insel wird ein Defensivfass mit Attrappen rüber geschossen. Durch die Attrappen wird die Defensive des Feindes geschwächt und Attacken richten größeren Schaden an.

5.1.2.3. Ressourcenfass

Durch eine Kombination aus drei Ressourcenfässern kann das Schiff repariert werden, das die Fässer einsammelt. Durch die Reparaturen und Verbesserungen des Bootes wird es schneller. Platziert man ein Ressourcenfass bei dem/der GegnerIn zwischen zwei Ressourcenfässer, wird das gegnerische Boot sabotiert. Dadurch wird es für kurze Zeiten langsamer.

5.1.2.4. Rumfass

Sammeln sich drei Rumfässer nebeneinander vor der eigenen Insel, wird der Pirat in Rage versetzt. Dadurch fällt er in einen Angriffsrausch und richtet mehr Schaden bei den Angriffen an. Wenn es der/die SpielerIn schafft, ein gepanschtes Rumfass bei dem/der GegenspielerIn zu kombinieren, wird das Sichtfeld von dem/der Kontrahenten/Kontrahentin beeinträchtigt. Dabei verschwimmt das Bild und die Kamera beginnt zu schwanken.

5.1.2.5. Jokerfass

Das Jokerfass beeinflusst das Kombinieren von Fässern. Kommt es zu einer positiven Kombination, wird aus den Inhalten der drei Jokerfässer ein Universalfass gebaut. Dieses Fass kann mit jedem Fasstyp kombiniert werden. Schafft man eine negative Kombination aus drei Jokerfässern, wird der/die GegnerIn von der Explosion geblendet und ist für eine kurze Zeit farbenblind. Dadurch wird das Platzieren und Kombinieren von Fässern erschwert.

5.1.3. Fässer einsammeln

Da die Fässer laufend verbraucht werden und im Spiel als Ressourcen verwendet werden, muss das Schiff neue Fässer aus dem Meer fischen. Im Spiel tauchen laufend Fässer zwischen den beiden Inseln der Gestrandeten auf. Diese versinken nach kurzer Zeit wieder. Diese Fässer stammen von den schiffsbrüchigen Piratenschiffen, die den beiden Piraten gehört haben. Der/die SpielerIn hat nun die Aufgabe, die Fässer schneller einzusammeln als sein Gegenüber. Sobald ein Fass auftaucht, muss schnell darauf geklickt werden, um dem Boot den Startauftrag zu erteilen. Erreicht das Boot vor dem/der GegnerIn das Fass, wird es

zur Insel zurückgebracht und steht nun am linken Bildschirmrand zur Auswahl zur Verfügung. Per Drag and Drop kann das Fass nun vor der Insel an einer gewünschten Position platziert werden. Durch das Positionieren der Fässer kann eine positive Kombination an der eigenen Insel ausgelöst werden.

5.2. States der Implementierung

Es wurde bereits die Finite State Machine Methode vorgestellt, die es möglich macht, die Abläufe der KI in States aufzuteilen und diese übersichtlich zu verwalten. Diese Methode wird für das KI Konzept von Piratenkampf verwendet. Im folgenden State-Diagramm sieht man die einzelnen States, die für Piratenkampf verwendet werden. Die Rechtecke symbolisieren die States und die Pfeile beschreiben, wann zwischen den States gewechselt wird. Die Pfeilrichtung bestimmt, von welchem State wohin gewechselt werden kann.

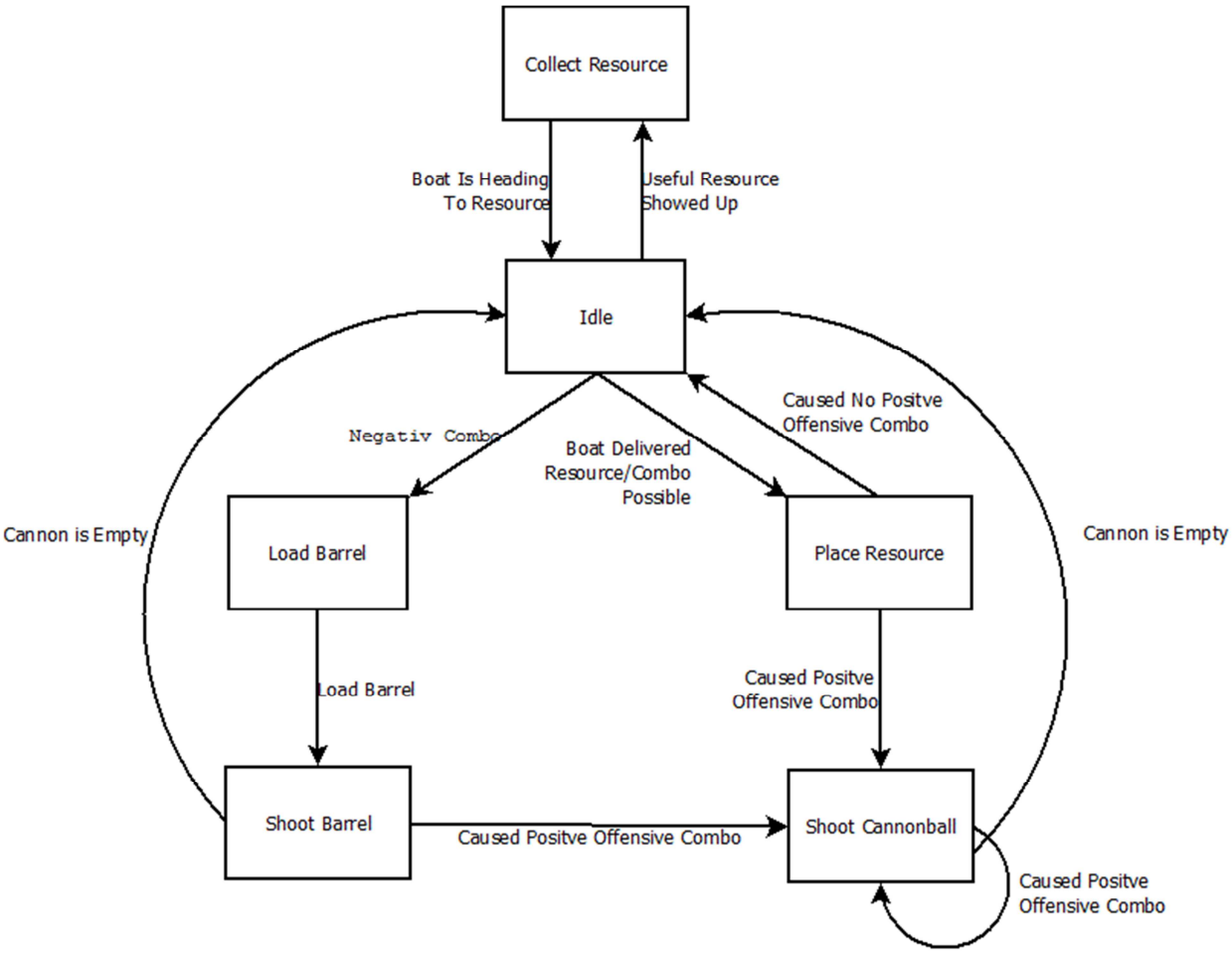


Abb. 21: State-Diagramm von Piratenkampf

5.2.1. Idle-State

Der Idle-State ist der standard State, der durchgehend ausgeführt wird, wenn der Computergegner nichts unternimmt. Es wird laufend überprüft, ob in den Collect-Resource-State, den Place-Resource-State oder den Load-Barrel-State gewechselt werden soll.

Wenn zwischen den beiden Inseln eine neue Ressource auftaucht, wird überprüft, ob diese benötigt wird. Ob das zutrifft, wird mittels den Fuzzy Rules entschieden. Welche das sind, wird später erklärt. Kriterien für die Entscheidung sind zum Beispiel, wie viele Fässer bereits um die Insel der KI schwimmen oder ob das gegnerische Boot bereits auf dem Weg zur Ressource ist. Wird entschieden, dass die Ressource eingesammelt werden soll, wechselt die FSM in den Collect-Resource-State.

Bringt das Boot eine neue Ressource zurück, wird überprüft, ob diese im Fässerring platziert werden soll. Das hängt vom Typen der Ressource ab und welche Fässer sich im Fässerring befinden.

Falls sich die Chance bietet, bei dem/der GegnerIn eine negative Kombination auszulösen, muss in den Load-Barrel-State gesprungen werden. Auch hier ist der Fasstyp entscheidend, ob der State-Wechsel ausgeführt wird.

5.2.2. Collect-Resource-State

Ist im Idle-State entschieden worden, dass man eine Ressource einsammeln soll, startet der Collect-Resource-State. Dann wird der Startbefehl an das Boot erteilt, das automatisch Richtung Ressource losfährt. Sobald die Befehle an das Boot gegeben wurden, wird wieder in den Idle-State umgeschaltet.

5.2.3. Place-Resource-State

Um die Übersicht im KI Konzept zu bewahren, ist der Place-Resource-State eine Finite State Machine, die selbst drei Unter-States besitzt. Wie bereits beim Vorstellen der Finite State Machine Methode erklärt wurde, erbt die FSM Klasse von der State Klasse und wird daher als normaler State behandelt. Im State-Diagramm in der unteren Abbildung sind die drei Unter-States zu sehen.

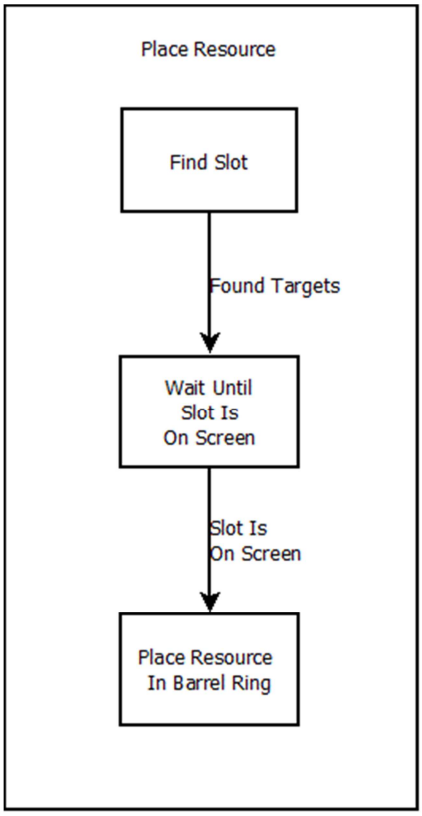


Abb. 22: Place-Resource-State-Diagramm

Sobald in den Place-Resource-State gewechselt wurde, wird der Find-Slot-State gestartet. Die einzelnen Fässer, die um die Insel schwimmen, sind immer einem Slot im Fässerring zugeteilt. Jeder Fässerring besteht aus 15 Slots. Wenn sich keine Fässer im Ring befinden, sind die Slots dennoch vorhanden.

Im Find-Slot-State wird der Slot gesucht, auf welchen die Ressource platziert werden soll. Welcher das ist, entscheidet sich in den Fuzzy Rules. Die Slots rotieren durchgehend um die Inseln herum. Sobald der Slot gewählt ist, muss die KI darauf warten, dass sich der ausgewählte Slot in den Bildschirm verschiebt. Der/die menschliche SpielerIn kann Ressourcen nur dann auf einem Slot platzieren, wenn dieser am Bildschirm zu sehen ist. Sobald das zutrifft, kann in den Place-Resource-In-Barrel-Ring-State gewechselt werden. Dort wird nur mehr die Aktion ausgeführt, die auch bei dem/der SpielerIn nach der Drag and Drop Aktion ausgeführt wird.

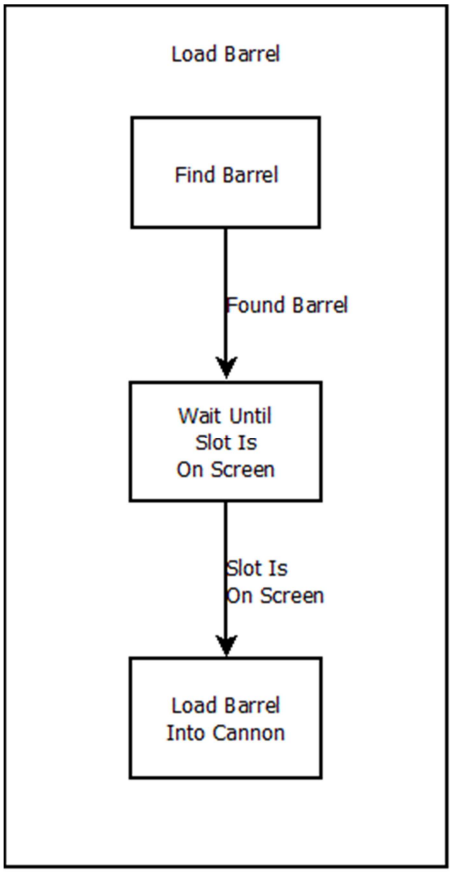
Falls nach dem Platzieren der Resource eine positive Offensivkombination ausgelöst wird, wechselt die FSM in den Shoot-Cannonball-State. Wenn allerdings keine solche Kombination gestartet wurde, wechselt die FSM wieder zurück in den Idle-State.

5.2.4. Shoot-Cannonball-State

Sobald sich drei Offensivfässer im Fässerring nebeneinander befinden, wird eine Kanonenkugel in die Kanone geladen und die FSM wechselt in den Shoot-Cannonball-State. Das Ziel ist es die gegnerische Insel zu treffen. Nach einigen Berechnungen, die von der Spielstärke der KI abhängen, wird die Kugel abgefeuert. Wenn eine Kombination ausgelöst wird, verschwinden die drei Fässer aus dem Ring. Falls durch das Verschwinden der drei Offensivfässer eine weitere Offensivkombination ausgelöst wurde, bleibt der Shoot-Cannonball-State der aktuelle State. Ansonsten wird der Idle-State aufgerufen.

5.2.5. Load-Barrel-State

Sobald sich die Chance bietet, dass man bei dem/der GegnerIn eine negative Kombination auslösen kann, wird vom Idle-State in den Load-Barrel-State gewechselt. Beim Load-Barrel-State handelt es sich um eine Finite State Machine, die in weitere States unterteilt ist.



Im Find-Barrel-State wird das Fass ausgewählt, das in die Kanone geladen werden soll. Sobald entschieden ist, welches Fass ausgewählt wird, wird gewartet bis der Slot, auf dem sich das Fass befindet, im Bildschirmbereich auftaucht. Beim Wait-Until-Slot-Is-On-Screen-State handelt es sich um den gleichen Unter-State wie vom Place-Resource-State. Der Grund, warum die KI warten muss, ist auch wieder derselbe. Die KI hätte sonst einen Vorteil gegenüber dem/der menschlichen SpielerIn. Er/sie kann nämlich per Drag and Drop nur die Fässer in die Kanone laden, die am Bildschirm angezeigt werden. Sobald dieser Slot am Bildschirm auftaucht, wird der Load-Barrel-Into-Cannon-State aufgerufen, indem die Aktion in der Spielwelt ausgeführt wird und anschließend in den Shoot-Barrel-State gewechselt wird.

Abb. 23: Load-Barrel-State-Diagramm

5.2.1. Shoot-Barrel-State

Der Shoot-Barrel-State ist ebenfalls eine FSM. Beim Start wird der Find-Possible-Targets-State aufgerufen.

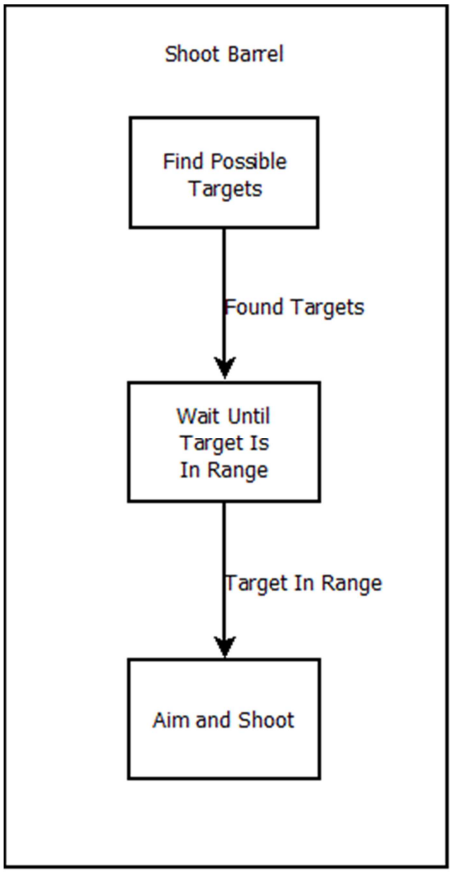


Abb. 24: Shoot-Barrel-State-Diagramm

In diesem State wird festgestellt, wie viele mögliche zweier Gruppierungen vom selben Fasstypen es gibt. Gesucht wird nach dem Typ, den auch das Fass in der Kanone hat. Sind alle möglichen Ziele ausgemacht worden, wechselt die FSM in den Wait-Until-Target-Is-In-Range-State. Die Kanone kann nicht auf den gesamten Fässerring des/der Gegners/GegnerIn schießen, sondern nur auf bestimmte Teile. Sobald eines der möglichen Ziele in Schussweite treibt, wird der Aim-And-Shoot-State ausgeführt. Dabei werden die Richtung und die Schusskraft unter Berücksichtigung der Spielstärke der KI festgelegt. Danach steht einem Abschuss des Fasses nichts mehr im Weg.

Falls eine Offensivkombination ausgelöst wurde, als das Fass aus dem Ring entfernt wurde, ruft die FSM den Shoot-Cannonball-State auf. Andernfalls wird der Idle-State gestartet.

5.3. Fuzzy Rules

Nachdem das FSM Konzept für Piratenkampf kreiert wurde, müssen die Regeln erstellt werden, durch welche die States gewechselt werden. Bei den Regeln handelt es sich um die Fuzzy Rules, welche in den einzelnen States ausgeführt werden.

5.3.1. Idle-State

Vom Idle-State ausgehend ist es möglich, in drei verschiedene States zu wechseln. Diese sind der Collect-Resource-State, der Place-Resource-State und der Load-Barrel-State.

5.3.2. Collect-Resource-State

Sobald eine neue Ressource zwischen den Inseln auftaucht, wird diese in einer Liste gespeichert. Wenn die `Update()` Funktion des Idle-States aufgerufen wird, werden die Fuzzy Rules überprüft, ob eine Ressource aus der Liste eingesammelt werden soll. Die Entscheidung hängt von drei Parametern ab. Der erste ist, wie voll der Fässerring ist. Dazu gibt es

eine Zugehörigkeitsfunktion, die die Anzahl der leeren Slots in Fuzzy Kategorien einordnet. Diese sind leer, wenige, viele und voll. Die vier Zugehörigkeitsfunktionen sind in der Abbildung darunter zu sehen.

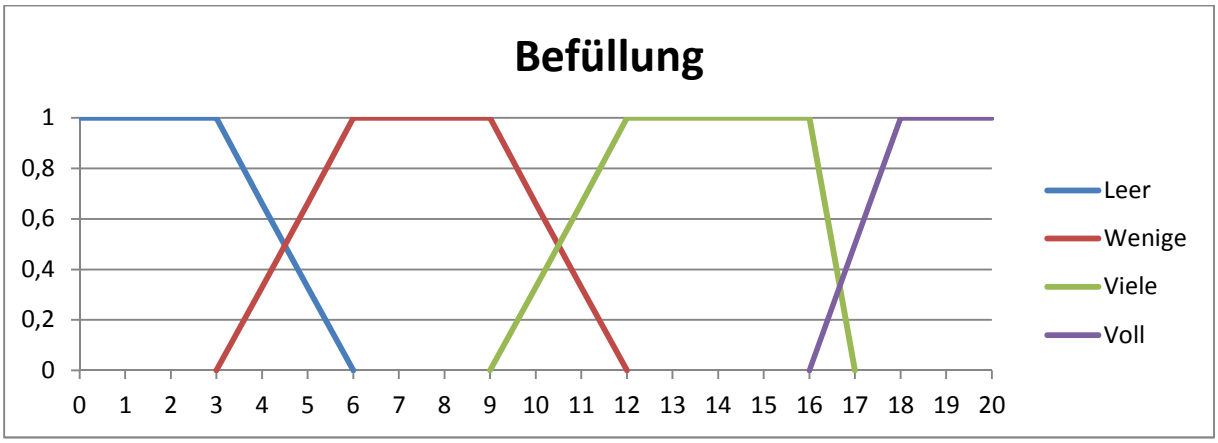


Abb. 25: Zugehörigkeitsfunktionen Fässerring Befüllung

Das zweite Kriterium für die Entscheidung ist, ob das eigene Boot vor dem gegnerischen ankommen kann. Diese Chance wird durch den Distanzunterschied der beiden Boote zur Ressource berechnet. Der Betrag des Unterschieds wird den Fuzzy Kategorien unmöglich, möglich und sicher zugeordnet.

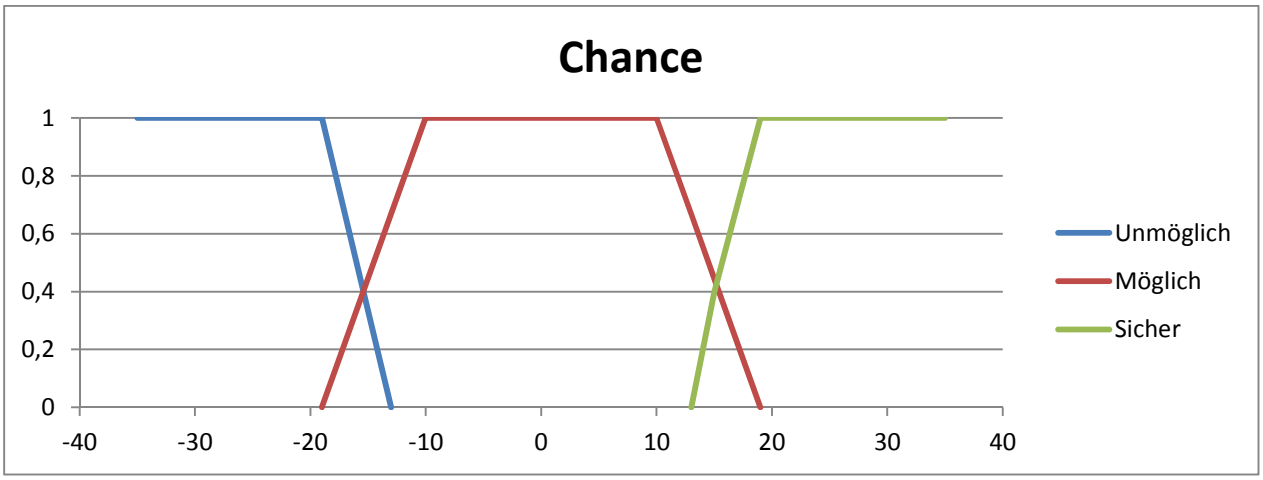


Abb. 26: Zugehörigkeitsfunktionen Chance zum Ressource einsammeln

Beim Berechnen von der Distanzdifferenz zwischen den Booten und der Ressource wird zusätzlich eine zufällige Fehleinschätzung hinzugefügt. Die Fehleinschätzung ist von der Stärke der KI abhängig. Der letzte Parameter der Fuzzy Rule ist der Typ von der Ressource. Für jeden Typ wird eine eigene Fuzzy Rule erstellt.

Das Offensivfass wird immer eingesammelt, wenn genügen Platz im Ring ist und wenn die Möglichkeit besteht, dass man vor de gegnerischen Boot die Ressource erreicht. Da das Rumfass und das Jokerfass beide sehr selten erscheinen, gilt dieselbe Regel. Für die Fuzzy Rules werden die Operatoren verwendet, die im Theorieteil vorgestellt wurden.

```
offensive = rum = joker = FuzzyAND(
                                FuzzyOR(möglich, sicher),
                                FuzzyNOT(voll)
                            );
```

Ein Defensivfass soll immer dann geholt, wenn weniger als sechs Defensivfässer im Fässerring sind. Ansonsten treffen dieselben Regeln wie beim Offensivfass zu.

```
defensive = FuzzyAND(
                    FuzzyAND(
                            FuzzyOR(möglich, sicher),
                            FuzzyNOT(voll)
                    ),
                    defensiveBarrelCount < 6
                );
```

Falls ein Ressourcenfass auftaucht, sollte der Ring entweder leer sein oder nur wenige Fässer beherbergen. Außerdem soll der Ring nicht mehr als drei Ressourcenfässer beherbergen.

```
resource = FuzzyAND(
                    FuzzyAND(
                                FuzzyOR(leer, wenig),
                                FuzzyNOT(unmöglich)
                            ),
                    resourceBarrelCount < 3
                );
```

Jede Fuzzy Rule liefert einen Wert zwischen 0 und 1. Es wird dabei jede Ressource überprüft, die aktuell zwischen den Inseln schwimmt. Es wird immer das Fass mit dem höchsten Fuzzy Output eingesammelt.

5.3.3. Place-Resource-State

Falls das Boot eine neue Ressource liefert und diese im Fässerring platziert werden soll, wird der Place-Resource-State gestartet, der gleichzeitig den Find-Slot-State ausführt. In diesem State müssen alle passenden Slots gefunden werden, auf die die Ressource eingefügt werden kann. Dabei gibt es von Fasstyp zu Fasstyp Unterschiede.

Beim Ressourcenfass und dem Jokerfass wird immer versucht, eine dreier Kombination auszulösen. Deshalb wird zuerst nach Gruppierungen von zwei Fässern vom gleichen Typ gesucht. Gibt es keine Gruppierung von zwei Fässern, wird nach einem einzelnen Fass gesucht. Ist kein einziges Fass vom selben Typen im Ring, wird nach einem Slot gesucht, der zwischen zwei verschiedenen Fasstypen liegt.

Falls es nicht nur einen möglichen Slot gibt, werden alle in einer Liste gespeichert und dem nächsten State als Ziel übergeben. Der Wait-Until-Slot-Is-On-Screen-State wartet solange, bis einer der möglichen Slots aus der Zielliste am Bildschirm auftaucht. Sobald das geschieht, wird dieser Slot dem nächsten State übergeben und die Ressource wird auf diesen Slot platziert.

Das Offensivfass, das Defensivfass und das Rumfass haben einen ähnlichen Ablauf mit ein paar besonderen Regeln. Bei einem Rumfass wird darauf gewartet, dass eine Offensivkombination möglich ist. Deshalb wird das dritte Rumfass solange nicht platziert, bis ein drittes Offensivfass auftaucht. Beim Offensivfass wird immer versucht eine Kombination auszulösen. Zuvor wird allerdings überprüft, ob ein drittes Rumfass darauf wartet, platziert zu werden. Sollte das der Fall sein, wird die Rumkombination ausgelöst und danach wird das Offensivfass im Ring platziert.

Beim Defensivfass wird die Kombination nur ausgelöst, wenn der/die GegnerIn die Kanone bereits geladen hat. Sollte allerdings ein viertes Defensivfass vom Boot gebracht werden, wird eine neue Kombination zwischen zwei verschiedenen Fasstypen angelegt.

5.3.4. Load-Barrel-State

Ob in den Load-Barrel-State gewechselt werden soll, wird mittels Fuzzy Rules entschieden. Zuallererst wird kontrolliert, ob eine negative Kombination überhaupt möglich ist. Dazu überprüft die KI, ob bei dem/der Kontrahenten/Kontrahentin zwei gleiche Fasstypen nebeneinander sind, und ob die KI selbst ein Fass von diesem Typ besitzt. Für die Fasstypen, auf die das zutrifft, werden die einzelnen Fuzzy Rules ausgeführt. Ausschlaggebend sind dafür die Lebensenergie der beiden Spieler und wie viele Fässer um beide Inseln schwimmen. Für die Anzahl der Fässer wird dieselbe Zugehörigkeitsfunktion wie beim Collect-Resource-State verwendet.

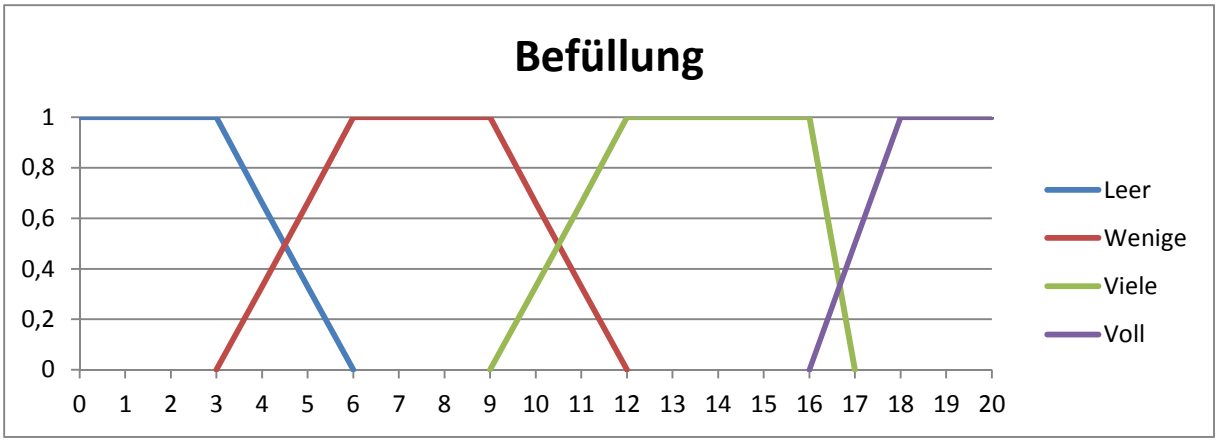


Abb. 27: Zugehörigkeitsfunktionen Fässerring Befüllung

Für die Lebensenergie gibt es vier verschiedene Fuzzy Kategorien. Diese sind voll, mittel, wenig und sehr wenig. Die jeweiligen Zugehörigkeitsfunktionen sind in der Abbildung darunter zu finden.

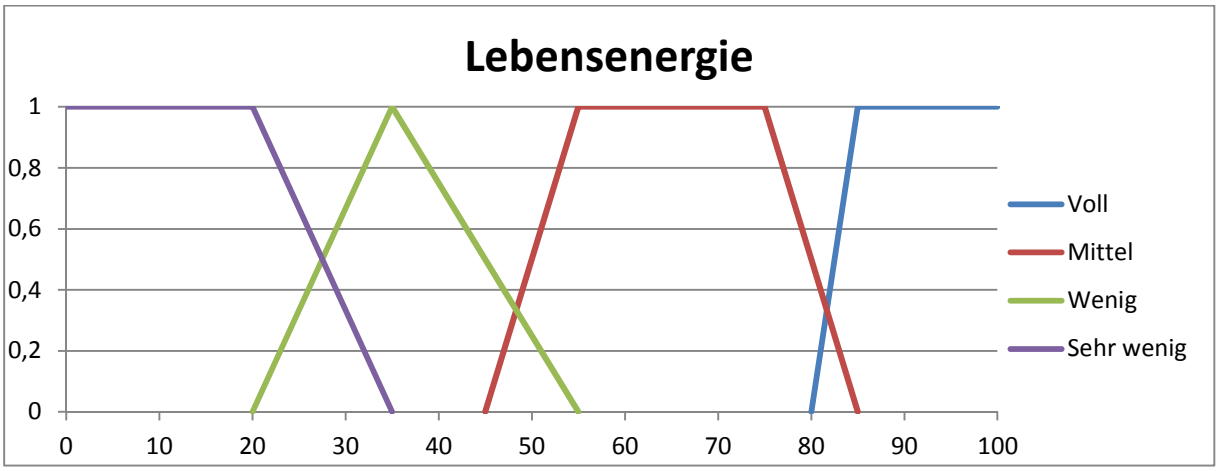


Abb. 28: Zugehörigkeitsfunktionen Lebensenergie

Die Fuzzification von der Fassanzahl und der Lebensenergie muss sowohl mit den Daten der KI als auch mit den Werten von dem/der GegenspielerIn durchgeführt werden. Danach wird mit den logischen Fuzzy Operatoren für jeden Fasstyp eine Fuzzy Rule erstellt.

Ein Offensivfass wird in die Kanone geladen, wenn der/die GegnerIn eine volle Lebensenergie besitzt oder wenn der/die FeindIn sehr wenig und die KI selbst auch wenig oder sehr wenig Energie übrig hat. Wenn die KI viele Energiepunkte besitzt, ist es klüger auf eine positive Offensivkombination zu warten, da diese mehr Schaden verursacht. Ist der Maschinengeg-

ner allerdings selbst kurz vor der Niederlage, sollte die KI nicht warten, sondern versuchen negative Offensivkombinationen auszulösen.

```
offensiv = FuzzyAND(
                    gegnerVolleEnergie,
                    FuzzyAND(
                           gegnerSehrWenigEnergie,
                           FuzzyOR(
                                  KIWenigEnergie,
                                  KISehrWenigEnergie
                           )
                    )
             );
```

Die KI soll eine negative Defensivkombination auslösen, wenn sie selbst mindestens drei Offensivfässer besitzt. Eine negative Defensivkombination verstärkt nämlich die Angriffe und mit drei Offensivfässern ist es möglich einen Angriff zu starten.

```
defensiv = KIOffensivFassAnzahl >= 3;
```

Falls der/die SpielerIn nahezu keine Fässer hat, kann die KI eine negative Ressourcekombination auslösen, damit der/die GegnerIn für kurze Zeit schlechter Ressourcen einsammeln kann. Ein anderer Grund für eine negative Ressourcenkombination kann sein, dass die KI selbst sehr viele Fässer besitzt und daher mit einer positiven Ressourcenkombination nichts anfangen kann.

```
resource = FuzzyAND(
                    FuzzyOR(
                           gegnerLeer,
                           gegnerWenige
                    ),
                    FuzzyOR(
                           KIViele,
                           KIVoll
                    )
             );
```

Um ein Rumfass in die Kanone zu laden, muss die KI wenig oder sehr wenig Lebensenergie haben und der/die GegenspielerIn mindestens drei Offensivfässer besitzen. Dadurch erschwert man dem/der Kontrahenten/Kontrahentin das Zielen.

```
rum = FuzzyAND(
             gegnerOffensivFassAnzahl >= 3,
             FuzzyOR(
                    KIWenigEnergie,
                    KISehrWenigEnergie
             )
       );
```

Eine negative Kombination aus Jokerfässern macht den/die GegnerIn farbenblind. Das soll ausgeführt werden, wenn beide SpielerInnen mittel oder wenig Lebensenergie besitzen.

```
Joker = FuzzyAND(
            FuzzyOR(
                  KIMittelEnergie,
                  KIWenigEnergie
            ),
            FuzzyOR(
                  gegnerMittelEnergie,
                  gegnerWenigEnergie
            )
      );
```

Beim Find-Barrel-State, der ein Unter-State vom Load-Barrel-State ist, werden alle möglichen Fässer gesucht, die man zu dem/der FeindIn schießen kann. Sollte es im Ring ein einzelnes Fass geben, wird dieses einer Zweiergruppe vorgezogen. Falls es mehrere Möglichkeiten gibt, werden alle passenden Fässer in einer Liste gespeichert. Diese Liste wird wieder dem Wait-Until-Slot-Is-On-Screen-State übergeben. Sobald eines der möglichen Ziele am Bildschirm auftaucht, wird der Load-Barrel-Into-Cannon-State aufgerufen.

5.3.5. Shoot-Barrel-State

Sobald das Fass in die Kanone geladen wurde, wird im Find-Possible-Targets-State nach passenden Zielen auf der gegnerischen Insel gesucht. Dabei wird nach Zweiergruppen vom selben Typ Ausschau gehalten. Die möglichen Ziele werden in einer Liste gespeichert. Im Wait-Until-Target-Is-In-Range-State wird darauf gewartet, dass eines der Ziele in Schussreichweite schwimmt. Sobald das der Fall ist, wird dieses Ziel dem Aim-And-Shoot-State übergeben. Dort wird unter Berücksichtigung der KI Stärke versucht auf das Ziel zu schießen.

5.3.6. Shoot-Cannonball-State

Wenn eine positive Offensivkombination bei dem KI Spieler ausgelöst wird, wird in den Shoot-Cannonball-State gewechselt. In diesem wird mittels der Stärkeeigenschaften der KI ein Ziel festgelegt, auf das mit einer Kanonenkugel geschossen wird.

5.4. Benötigte Informationen der KI

Damit die Fuzzy Rules richtig ausgeführt werden können, muss die KI einige Informationen über die Spielwelt wissen. Diese Informationen sollen gekapselt in der `MemoryManager` Klasse gespeichert werden. Zusätzlich gibt es noch die Hilfsklasse `PlayerInformation`, bei der In-

formationen zum Spieler gesammelt gespeichert werden, da viele Informationen sowohl von dem/der Kontrahenten/Kontrahentin als auch von der KI selbst gespeichert werden müssen.

In der `PlayerInformation` Klasse wird eine Referenz zur Insel, die Lebensenergie, die Slots und alle Fässer aus dem Fässerring gespeichert. Außerdem wird eine Liste der Ressourcen gebraucht, die das Boot eingesammelt hat, die aber noch nicht im Ring platziert wurden. Zudem wird noch eine Referenz vom Boot benötigt. In einer weiteren Liste werden die Fässer aus dem Fässerring festgehalten, die momentan am Bildschirm zu sehen sind. Zusätzlich werden die gegnerischen Fässer benötigt, die von der eigenen Kanone getroffen werden können. Abschließend wird gespeichert, ob die Kanone im Moment geladen ist.

In der `MemoryManager` Klasse werden zwei Variable vom Typ `PlayerInformation` angelegt, einmal für die KI selbst und einmal für den/die GegenspielerIn. Außerdem werden die Ressourcen gespeichert, die zwischen den Inseln auftauchen. Zuletzt wird noch festgehalten, ob sich im eigenen Fässerring etwas geändert hat und ob das Boot eine neue Ressource gebracht hat.

5.5. Modellierung der KI Spielstärken

Um verschiedene Spielstärken der KI modellieren zu können, werden einige variable Eigenschaften in der `DifficultManager` Klasse festgehalten.

5.5.1. Verwaltungsklasse der Stärkeeigenschaften

In der `DifficultManager` Klasse wird zum einen die Reaktionszeit gespeichert. Diese bestimmt automatisch die Aktualisierungsrate des KI Algorithmus, der in der `AIControl` Klasse durch einen Timer aufgerufen wird. Je kürzer die Reaktionszeit ist, umso besser ist der Computergegner.

Ein anderes Kriterium, das die Stärke der KI bestimmt, ist die Zielsicherheit beim Schießen mit der Kanone. Für diese Eigenschaft gibt es eine Variable, die die mögliche Abweichung speichert. Ein zufällig generierter Wert wird in eine Funktion eingesetzt, um eine Zahl zwischen 0 und 1 zu erhalten. Welche Funktion für die Abweichung gewählt wird, entscheidet die Variable `deviationSkill`. Der Wertebereich für diese Eigenschaft bewegt sich von 0 bis 4. Liegt der Wert unter 1,5, wird eine lineare Funktion verwendet. Zwischen 1,5 und 2,3 kommt eine quadratische Funktion zum Einsatz. Die nächste Stufe geht bis 2,8 und liefert eine kubische Funktion, um die Abweichung zu berechnen. Bei jedem Wert über 2,8 wird

eine biquadratische Funktion verwendet. Je nach Stärke der KI ist der Abweichungswert geringer oder höher. Die verwendete Funktion ist bei einem/einer schwachen GegnerIn eine lineare Funktion und bei einem/r sehr starken eine biquadratische Funktion. Die Zahl, die durch die Funktion berechnet wird, wird mit dem möglichen Abweichungswert multipliziert und zur Zielkoordinate hinzugefügt.

Damit entschieden wird, ob eine Ressource eingesammelt werden soll, berechnet die KI den Distanzunterschied der einzelnen Boote zur Ressource. Da dieser Vorgang bei einem Menschen geschätzt wird und deshalb nicht exakt stimmt, soll eine zufällige Fehleinschätzung berücksichtigt werden. Dasselbe gilt für die Fässer, die um die gegnerische Insel rotieren. Beim Zielen muss diese Rotation miteinberechnet werden, welche von einem Menschen wieder nur geschätzt werden kann. Deshalb werden für beide Aktionen dieselbe Logik und dieselben Eigenschaften wie beim Zielen verwendet.

Wie bereits erklärt wurde, kann es durchaus passieren, dass ein/eine SpielerIn einen Fehler begeht. Damit dieses Verhalten simuliert werden kann, sollen zu einer bestimmten Wahrscheinlichkeit Fehler auftreten können. Dafür gibt es eine Fehlerquote, bei der eine prozentuale Chance besteht, dass die KI eine falsche Entscheidung trifft. Ein Fehler kann zum Beispiel sein, dass eine Ressource an der falschen Position platziert wird, dass eine falsche Ressource eingesammelt wird oder dass beim Fässerschießen auf einen falschen Slot gezielt wird.

Die letzten Eigenschaften ermöglichen es der KI falsch zu spielen. Die erste Eigenschaft ist, dass der Computer verschieden viel Schaden mit demselben Angriff anrichten kann. Der Wert, der als Schaden abgezogen wird, ist in der `DifficultManager` Klasse gespeichert. Die zweite Fähigkeit, die das Betrügen möglich macht, ist die Geschwindigkeit, mit dem sich das Boot fortbewegt.

5.5.2. Adaptive KI Stärke

Die adaptive Spielstärke der KI soll den Spielstil der Maschine schnell auf eine ähnliche Spielstufe wie des/der menschlichen Gegners/Gegnerin einstellen. Dafür werden die Eigenschaften der Spielstärke verändert.

Um dieses Konzept umsetzten zu können, müssen für jede Eigenschaft einige zusätzliche Werte definiert werden. Für jede Eigenschaft muss ein Startwert, ein Minimum und ein Maximum festgelegt werden. Zusätzlich gibt es einen Veränderungswert, der das Spielniveau

beeinflusst. Damit erkannt wird, ob ein Phasenwechsel nötig ist, muss die Richtung der Stärkenveränderung gespeichert werden. Dafür wird festgehalten, ob die KI gerade stärker oder schwächer gemacht wurde.

Alle Eigenschaften, die zuvor zur Regulierung der Spielstärke festgelegt wurden, werden während des Spiels auf das Spielniveau des/der Gegners/Gegnerin adaptiert. Um zu überprüfen, ob der/die menschliche SpielerIn besser oder schlechter als die KI ist, werden in 15 Sekunden Abständen einige Parameter überprüft.

Die Lebensenergie der beiden Kontrahenten/Kontrahentinnen ist ein solcher Parameter, der bestimmt, wie viel Schaden die KI bei einem Angriff anrichtet. Ein weiterer Indikator ist das Verhältnis zwischen der Anzahl der Schüsse und den erfolgreichen Treffern. Das beeinflusst die Zieleigenschaften für das Schießen. Der Vergleich zwischen der Anzahl der eingesammelten Ressourcen bestimmt, ob die Reaktionszeit verkürzt werden und die Geschwindigkeit des Bootes erhöht werden muss.

Zusätzlich zur adaptiven Spielstärke der KI kann der/die BenutzerIn vor dem Spiel eine von drei Schwierigkeitskategorien wählen. Diese sind Anfänger, Fortgeschrittener und Experte. Diese Stufen werden alle während des Spiels an den/die GegnerIn angepasst. Die Unterschiede zwischen den Schwierigkeitsstufen sind die Startwerte und der möglichen Wertebereiche der Eigenschaften. Die Spielstärke wird während des Spiels in jeder Kategorie an den/die KontrahentIn adaptiert. Die Unterschiede sind die Wertebereiche der adaptiven Eigenschaften, die bestimmen, wie gut oder wie schlecht die KI werden kann.

In den folgenden Tabellen stehen die Wertebereiche der Eigenschaften, die sich zwischen Minimum und Maximum befinden und dem Änderungswert zu Beginn eines Spiels. Außerdem gibt es ein Minimum für den Änderungswert, den er nicht unterschreiten kann.

Anfänger				
	Änder	**Min Änder**	**Min**	**Max**
Reaktionszeit (ms)	30	7	200	400
Mögliche Abweichung Schießen (°)	8	2	35	62
Mögliche Abweichung Boot Distanz (m)	3	0,5	10	25
Abweichungsfunktion	0.7	0.2	0	2,7
Fehlerquote (%)				7 %
Schaden bei Angriff (%)	3	0,5	50	70
Geschwindigkeit Boot (%)	4	0,7	70	90

Tab. 1: Werte für Schwierigkeitsstufe Anfänger

Fortgeschrittener				
	Änder	**Min Änder**	**Min**	**Max**
Reaktionszeit (ms)	20	5	150	240
Mögliche Abweichung Schießen (°)	7	2	24	48
Mögliche Abweichung Boot Distanz (m)	2	0,4	7	20
Abweichungsfunktion	0,5	0,1	1,6	3
Fehlerquote (%)				4 %
Schaden bei Angriff (%)	3	0,5	70	90
Geschwindigkeit Boot (m/s)	3	0,5	85	100

Tab. 2: Werte für Schwierigkeitsstufe Fortgeschrittener

Experte				
	Änder	**Min Änder**	**Min**	**Max**
Reaktionszeit (ms)	20	5	80	160
Mögliche Abweichung Schießen (°)	5	1	5	24
Mögliche Abweichung Boot Distanz (m)	2	0,4	5	15
Abweichungsfunktion	0,4	0,1	2,3	3,5
Fehlerquote (%)				2 %
Schaden bei Angriff (%)	3	0,5	100	120
Geschwindigkeit Boot (m/s)	4	0,7	100	120

Tab. 3: Werte für Schwierigkeitsstufe Experte

6. Ergebnisse

Das Ziel dieser Arbeit war es, für ein praktisches Beispiel ein KI Konzept zu entwickeln, das mittels adaptiver Spielstärke der KI für einen knappen Ausgang des Spiels sorgt. Um das zu erreichen wurde für das praktische Beispiel eine Methode entwickelt, die die Fähigkeiten der KI auf das Spielniveau des/der Gegners/Gegnerin einpendelt. Um zu überprüfen, ob die entwickelte Methode die Anforderungen erfüllt, wurde ein Test erstellt, in dem die KI der Stärkestufe Anfänger gegen alle drei Schwierigkeitsstufen 100 Mal angetreten ist. Von diesen 100 Spielen wurden jeweils 50 mit und 50 ohne der entwickelten Methode zur adaptiven Spielstärke gespielt. Das Ergebnis der Methode wird daran gemessen, wie viel Prozent der möglichen Spiele gewonnen wurden, und wie viel verbleibende Lebensenergie am Schluss von einem Spiel übrig geblieben ist. Die verbleibende Lebensenergie repräsentiert, wie knapp der Ausgang von einem Match ist.

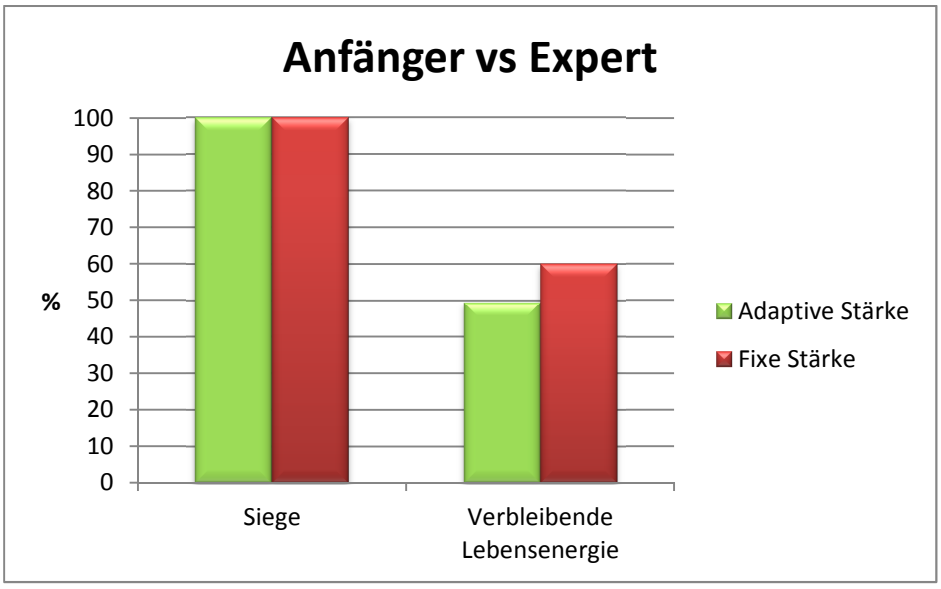


Abb. 29: Vergleich Anfänger gegen Experte

Das Ergebnis zeigt, dass mit der entwickelten Methode ein sehr knapper Ausgang des Spiels erzielt wird. Sogar zwischen dem Anfänger und dem Experten gibt es ein engeres Ergebnis, im Vergleich zu den Spielen ohne die adaptive Methode.

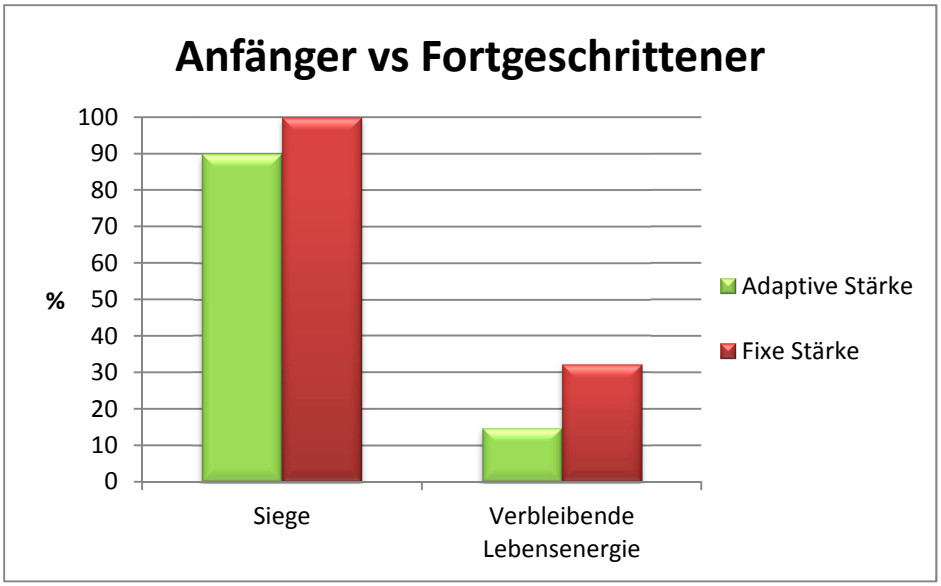


Abb. 30: Vergleich Anfänger gegen Fortgeschrittener

Dem Anfänger ist es außerdem gelungen, einige Matches gegen den Fortgeschritten zu gewinnen. Das ist ohne den Algorithmus zur Veränderung der Spielstärke nie aufgetreten. Bei der durchschnittlich verbliebenen Lebensenergie gewährleistet die KI mit der variablen Stärke ein viel knapperes Ergebnis.

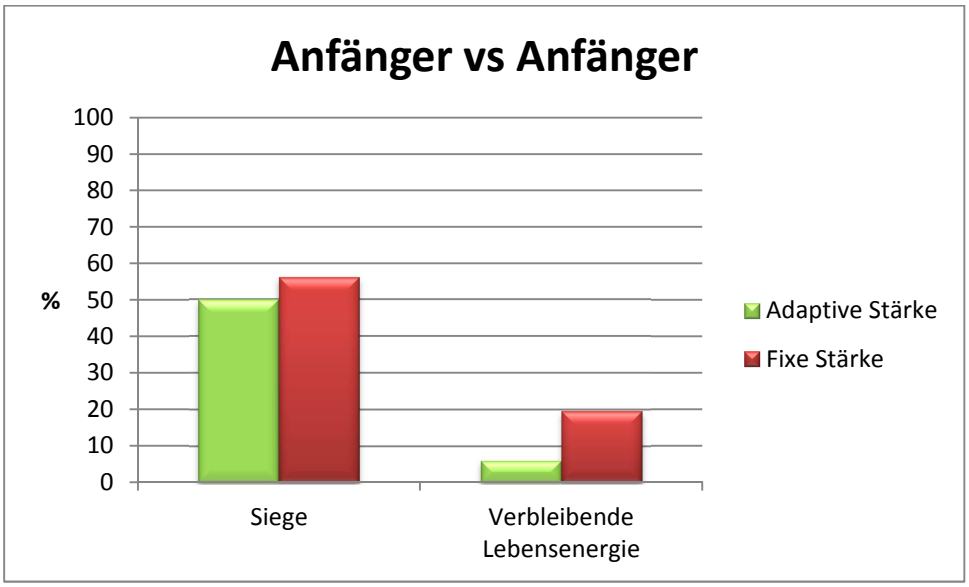


Abb. 31: Vergleich Anfänger gegen Anfänger

Selbst beim Ausgang zwischen zwei KI Spielern mit der Anfänger Stärke ist ein Erfolg zu erkennen. Obwohl man davon ausgehen könnte, dass es irrelevant, ob das Spielniveau angepasst wird, da die Gegner ohnehin dieselben Werte von Beginn an besitzen. Aber da eini-

ge Aktionen wie das Zielen und Schießen durch Zufallswerte beeinflusst werden, können die Werte der KI stark variieren. Bei der adaptiven Stärke ist es möglich, auf diese Zufallsschwankungen zu reagieren. Deshalb ist das Ergebnis, dass die durchschnittlich verbleibende Lebensenergie bei Spielen mit einem variablen Spielniveau um einiges geringer als bei Spielen mit einer fixen Stärke.

Eine Einschränkung zur Verwendung dieser Methode ist, dass das Spielniveau des Menschen einigermaßen konstant sein muss. Daher bietet sich diese Methode der Stärkenanpassung nur bei simplen Spielprinzipen mit kurzer Spieldauer an. Bei einem komplexen und länger dauernden Spiel ist es wahrscheinlich, dass der/die BenutzerIn unter dem Match einiges dazulernt. Das würde das Spielniveau erheblich beeinflussen. Es wäre unnütz, wenn die KI versucht, ein konstantes Niveau zu finden, dieses aber variabel ist. Dadurch dass der Veränderungswert bei dieser Methode immer kleiner wird, kann auf größere Veränderungen der Spielstärke des/der Gegners/Gegnerin nur langsam reagiert werden.

Damit die KI gegen einen menschlichen Spieler gute Ergebnisse erzielen kann, die Werte der Eigenschaften viel feiner eingestellt werden können, müsste viel mehr getestet werden. Ein weiteres Problem des bestehenden Konzeptes ist, dass die Fuzzy Rules sehr strikt sind und dadurch schnell von einem Menschen durchschaut werden können. Das macht die KI transparent. Wenn die Fuzzy Rules hingegen in sich verschwommener sind, ist es schwerer etwas vorherzusagen.

7. Zusammenfassung

Im Zuge dieser Arbeit wurden gängige KI Methoden vorgestellt, mit denen es möglich ist, ein Konzept für einen KI Gegenspieler zu entwickeln. Bei den Methoden handelt es sich um ein Entscheidungssystem für einen Gegenspieler, das mit der Finite State Machine Methode und der Fuzzy Logic kombiniert wird. Außerdem wurde beschrieben, wie es möglich ist, verschiedene Schwierigkeitsstufen für einen KI Gegenspieler zu modellieren. Zusätzlich wurden Methoden zur Adaption der Spielstärke vorgestellt. Aufbauend auf die adaptiven Methoden wurde eine eigene entwickelt. Wie die vorgestellten Methoden genutzt werden können, um ein KI Konzept zu erstellen, wird anhand eines praktischen Beispiels erklärt. Es wird gezeigt, wie States erstellt werden und mittels welchen Fuzzy Rules diese wechseln. Außerdem wird die Modellierung von verschiedenen Spielstärken gezeigt und wie die Methode für die adaptive Spielstärke praktisch umgesetzt wird.

Abschließend wird ein Vergleich zwischen einer fixen Spielstärke und einer adaptiven Spielstärke erstellt. Dieser hat gezeigt, dass die adaptive Methode gute Resultate bringt. Danach wird noch auf Verbesserungsmöglichkeiten und Probleme beziehungsweise Einschränkungen dieser adaptiven Methode und des KI Konzeptes eingegangen.

Eine interessante Weiterentwicklung der adaptiven Methode könnte sein, dass sobald das Spielniveau des/der KontrahentenIn gefunden wurde, die Spielstärke der KI ein wenig höher eingestellt wird. Danach sollte sich das Können des Computers nicht mehr verändern, bis das Spiel vorbei ist. Dadurch wird von dem/der SpielerIn verlangt, dass er/sie für einen Sieg über sich hinauswachsen muss. Dadurch muss er/sie sich bei jedem Match anstrengen und sich laufend steigern. Ein Problem bei diesem Ansatz ist, dass der/die BenutzerIn zu Beginn bewusst schlecht spielen könnte, bis die Stärke der KI fixiert ist. Danach wird wieder mit dem normalen Können weitergespielt und mit Leichtigkeit gewonnen.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

Bakkes, Sander, Pieter Spronck, and Jaap van den Herik. 2010. *Rapid adaptation of video game AI*.

Bourg, David M., and Glenn Seemann. 2004. *AI for game developers*. Sebastopol CA: O'Reilly.

McLean, Alex W.2002. "An Efficient Al Architecture Using Prioritized Task Categories." In *AI game programming wisdom*. Hingham, Mass.: Charles River Media. 290-297.

Norvig, Peter, and Stuart J. Russell. 2003. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs New Jersey: Alan Apt.

Orkin, Jeff. 2002. "Tips from the Tranches." In *AI game programming wisdom*. Hingham, Mass.: Charles River Media. 29-35

Schwab, Brian. 2004. *AI game engine programming*. Hingham, MA: Charles River Media.

Scott, Bob. 2002a "The Illusion of Intelligence." *In AI game programming wisdom*. Hingham, Mass.: Charles River Media. 16-20.

Scott, Bob. 2002b. "Architecting a Game AI." *In AI game programming wisdom*. Hingham, Mass.: Charles River Media. 285-289.

Postman, Eric, Ida Sprinkhuizen-Kuyper, and Pieter Spronck. 2004. Difficulty scaling of game AI. In *Proceedings of the 5th International Conference on Intelligent Games and Simulation* (GAME-ON 2004), pp. 33–37.

Thor, Alexander. 2002. "An Optimized Fuzzy Logic Architecture for Decision-Making." In *AI game programming wisdom*. Hingham, Mass.: Charles River Media. 367-374.