Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Sebastian Zagaria

Emotional adäquat reagierende KI-Agenten zur Erhöhung von Immersion in Computerspielen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sebastian Zagaria

Emotional adäquat reagierende KI-Agenten zur Erhöhung von Immersion in Computerspielen

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Philipp Jenke

Eingereicht am: 16. Juni 2017

Sebastian Zagaria

Thema der Arbeit

Emotional adäquat reagierende KI-Agenten zur Erhöhung von Immersion in Computerspielen

Stichworte

Affective Computing, Affective Gaming, AI, Game AI, Behavior Trees, Companion Technologie, Emotion Bike, Affectiva, Unity3d, Emotion recognition

Kurzzusammenfassung

Affective Gaming beschäftigt sich mit der Fragestellung wie Technologien zur Emotionserkennung verwendet werden können, um bisherige Spielmechaniken mit einer emotionalen Komponente zu ergänzen. So kann das emotionale Verhalten von Spieler analysiert und emotional adaptive Spiele entwickelt werden, sodass die Immersion des Spielers erhöht wird. Eine der attraktivsten Lösungen zur Emotionserkennung ist das Erfassen von Emotionen über Kameras. Diese sind kostengünstig und in den meisten Spielplattformen bereits vorhanden. Ziel dieser Arbeit ist die Konzipierung und Entwicklung einer Plattform zum Testen von KI-Agenten für Computerspiele, die adäquat auf die Emotionen eines Spielers reagieren. Für die Umsetzung werden Techniken aus den Bereichen künstliche Intelligenz, Affective Computing und Affective Gaming hinzugezogen.

Sebastian Zagaria

Title of the paper

Emotional appropriate reacting AI-Agents to increase immersion in videogames

Keywords

Affective Computing, Affective Gaming, AI, Game AI, Behavior Trees, Companion Technologie, Emotion Bike, Affectiva, Unity3d, Emotion recognition

Abstract

Affective Gaming is concerned with the question of how emotion recognition software can be utilized in Games to extend classic game mechanics with an emotional aspect. By analyzing the emotional behavior of players and developing emotional adaptive games to increase the immersion of players. One of the most promising technologies is emotion recognition with cameras. With a low budget entry cost and already available in most households. The goal of this work is to conceptualize and develop a platform to test AI agents for games, which can react appropriately to the emotions of a player. The realization will utilize technologies and techniques used in Artificial Intelligent, Affective Computing and Affective Gaming.

Inhaltsverzeichnis

Listings

1. Einleitung

In den letzten Jahren sind im Bereich der Human Computer Interaktion eine Vielzahl von Technologien entstanden, die dem Menschen die Bedienung mit dem Computer erleichtern sollen. Ein erheblicher Teil der Interaktion zwischen Menschen oder auch Mensch und Tier findet bewusst, aber auch unbewusst über einen emotionalen Dialog statt. So sind in diesem Zusammenhang die Forschungsbereiche Affective Computing und Companion Technologie entstanden.

Die Fragestellung dieser Forschungsbereiche ist es, wie durch das Einbeziehen von Emotionen die Mensch-Maschinen-Interaktion verbessert werden kann. Zur Emotionserkennung wird eine Vielzahl verschiedenster Sensorik eingesetzt. So werden Biosensoren zum Messen von Puls, Herzrate oder Hautwiderstand verwendet, um auf einen affektiven Zustand zu schließen.

Eine sehr attraktive Lösung bietet das visuelle Erfassen von Emotionen über Kameras, da es menschlicher Emotionswahrnehmung am nächsten kommt. Sie bieten zudem den Vorteil, dass sie nicht an den Körper des Benutzers angebracht werden müssen. Der allgemein geringe Anschaffungspreis und eine weite Verbreitung von Webcams durch Laptops verringern die Einstiegshürden, welche durch andere Sensorik gegeben ist.

In jüngster Zeit führte ein rasanter Anstieg an Start-up-Unternehmen, die Emotionserkennungssoftware anbieten, zu einer allgemeinen Verfügbarkeit dieser Technologie. Auf Basis dieser Software sind sowohl in der Forschung als auch in der Wirtschaft erfolgreiche Projekte und Produkte entstanden. Steigende Erkennungsraten und Verfügbarkeit der Technologien erhöhen das Interesse in der Marktforschung, Qualitätskontrolle, Unterhaltung, interaktiven Medien und Überwachung.

Der Forschungsbereich Affective Gaming beschäftigt sich mit der Fragestellung, wie diese Technologien verwendet werden können, um die bisherigen Spielmechaniken mit einer emotionalen Komponente zu ergänzen. So werden emotionale Verhalten von Spielern analysiert, emotional adaptive Spiele entwickelt und die Wirkung auf Spieler untersucht. Meist wird diese Technologie jedoch nur verwendet, um einzelne Spielmechaniken oder die Spielumgebung zu manipulieren. So passt sich beispielsweise die Beleuchtung an den affektiven Zustand des

Spielers an oder Postprocessing Effekte werden benutzt, um die Emotionen des Spielers zurück zu koppeln.

Es erscheint jedoch unnatürlich Spielelemente wie das Wetter im Spiel oder die Effektivität einer Waffe anhand der Emotion des Spielers anzupassen. Die Emotionserkennung sollte auf natürliche und glaubwürdige Art in Spiele integriert werden.

In dieser Arbeit wird untersucht, wie Emotionen in einem Dialog zwischen dem Spieler und Nicht-Spieler-Charakteren einbezogen werden können. Diese Charaktere in Computerspielen hätten somit das Potenzial, lebensechter zu wirken und für den Menschen nachvollziehbarer Reaktionen zu zeigen. Dadurch können überzeugendere Spielewelten erschaffen werden, welche den Spieler noch tiefer in virtuelle Welten eintauchen lassen.

Ziel ist die Konzipierung und Entwicklung einer Plattform zum Testen von KI-Agenten für Computerspiele, die adäquat auf die Emotionen eines Spielers reagieren. Für die Umsetzung werden Techniken aus den Bereichen künstliche Intelligenz, Affective Computing und Affective Gaming hinzugezogen.

Dafür werden im Kapitel 2 die Themenbereiche des Affective Computing, Affective Gaming und Companion Technologie analysiert, sowie die Funktionsweise und die verwendete Sensorik dargestellt und die Anwendungsbereiche für Affective Computing beschrieben.

Anhand ausgewählter Arbeiten wird beschrieben, wie Technologien aus dem Affective Computing in Spielen eingesetzt werden. Anschließend wird der Forschungsbereich Companion Technologie untersucht, um die Anforderungen, die an einen virtuellen Begleiter gestellt werden könnten, zu bestimmen.

Im Kapitel 3 wird das Emotion Bike Projekt beschrieben. Während des Projekts wurde ein Spiel entwickelt und mittels Emotionserkennung der Spielverlauf und der emotionale Einfluss auf den Spieler untersucht. Des Weiterem werden in dem Projekt entstandene Veröffentlichungen beschrieben und deren Ergebnisse diskutiert.

Anschließend wird im Kapitel 4 der Begriff der künstlichen Intelligenz und Agenten definiert. Danach werden verschiedene Techniken untersucht, die in der Spieleindustrie und Forschung angewendet werden.

Im Kapitel 5 wird die verwendete Architektur für den emotional adäquat reagierenden KI-Agenten beschrieben. Basierend auf der Agentenarchitektur wird im Kapitel Game Design 6 ein Prototyp für ein Spiel und dessen Implementierung beschrieben. Im Anschluss wird die Implementation für die Untersuchungen mit Probanden verwendet. Das letzte Kapitel 7 beschreibt den Versuchsaufbau, die Versuchsdurchführung und die Auswertung der Ergebnisse.

2. Analyse

Zuerst wird der Forschungsbereich des Affective Computing und dessen Teilbereich Affective Gaming erläutert. Affective Computing konzentriert sich auf die Erkennung und Analyse von Emotionen sowie deren Anwendungsbereiche. Der Teilbereich Affetive Gaming untersucht die Fragestellung, wie Emotionserkennung in Computerspielen sinnvoll eingesetzt werden kann. Darauffolgend wird der Forschungsbereich Companion Technology beschrieben, welcher sich mit den Anforderungen und Umsetzung von künstlichen Begleitern beschäftigt, deren Hauptaufgabe die Interaktion mit einem menschlichen Benutzer ist. Anschließend wird in einem Fazit der Bezug der einzelnen Forschungsbereiche zu dieser Arbeit diskutiert.

2.1. Affective Computing

Der Forschungsbereich des Affective Computing ist ein Teilbereich der Human-Computer-Interaktion, siehe Picard (1999). Der Ursprung des Forschungsbereiches wird auf Dr. Rosalind Picard vom Massachusetts Institute of Technology (MIT) zurückgeführt. Die 1995 einen Fachbericht mit dem Titel Affective Computing veröffentlichte (vgl. Picard (1995)). Affative Computing vereint Sensorik und Software Technologien zur Erkennung des affektiven Zustandes einer Person. Es handelt sich um einen sehr aktiven Forschungsbereich, der mittlerweile andere Forschungsbereiche wie das Affective Gaming hervorgebracht hat (vgl. Gilleade u. a. (2005b)). Grundlegend gibt es zwei Teilbereiche innerhalb des Affective Computing:

1. Erkennen von Emotionen mittels Sensorik, wie beispielsweise Kameras, Eyetracker, Blutdruckmesser, den Hautwiderstand oder die Körpertemperatur.

2. Die Simulation von Emotionen, z. B. zur Erhöhung der Glaubwürdigkeit von virtuellen Charakteren.

Der Prozess des Affective Computing kann in drei Aufgabenbereiche aufgeteilt werden (Abbildung 2.1). Erstens, das Erfassen der affektiven Daten über Sensoren mittels Kameras, Eyetracker, Mikrofone und physiologische Sensoren aus dem medizinischen Bereich. Zweitens, das Analysieren der erfassten Daten durch das Übertragen auf Emotionsmodelle (diskrete und

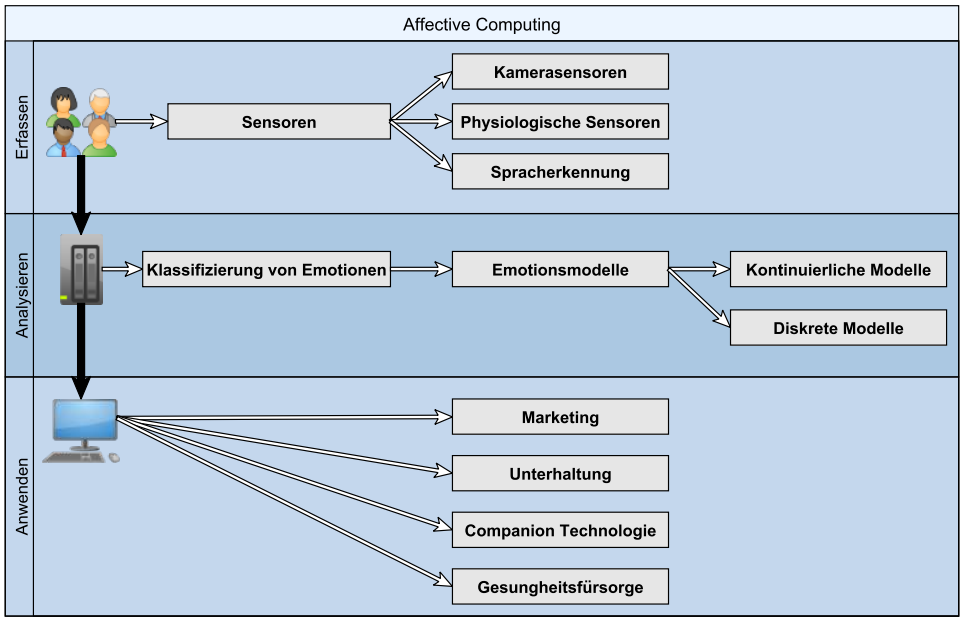

Abbildung 2.1.: Übersicht Affective Computing

kontinuierliche) und die Klassifizierung durch maschinelles Lernen. Drittens, Anwendungen basierend auf den erfassten und analysierten Daten für beispielsweise Marketing, Unterhaltung oder Gesundheitsvorsorge.

2.1.1. Erfassen von affektiven Daten

Gesichtserkennung: Im Bereich des Affective Computing haben sich fünf Basisemotionen größtenteils durchgesetzt. Diese können durch Merkmale im menschlichen Gesicht identifiziert und beispielsweise über Kameras erfasst werden. Begründet wurden diese durch Paul Ekman (vgl. Ekman (1992)), welcher sich mit nonverbaler Kommunikation beschäftigt. Er beschreibt die Emotionen Wut, Angst, Vergnügen, Traurigkeit, Überraschung und Ekel (Anger, Fear, Enjoyment, Sadness, Surprise, und Disgust). Oft wird die Emotion Enjoyment auch mit Joy oder Happy abgekürzt. Diese Emotionen gelten als Universalemotionen, die unabhängig von der ethnischen Herkunft, sozialem Umfeld und Kultur entstehen. So konzentrieren sich die meisten kamerabasierten Emotionserkennungssysteme oft nur auf diese sechs Basisemotionen.

Eyetracker: Eyetracker werden zum Erfassen der Blickrichtung und Pupillenerweiterung verwendet. Die Pupillenerweiterung wird mit den Emotionen Stress, Angst und Aufre-

gung assoziiert. Über diesen Sensor kann die Aufmerksamkeit und Fokus eines Benutzer gemessen werden.

Physiologischen Sensoren: Es existiert eine Menge unterschiedlicher physiologischer Sensoren, die es erlauben, den affektiven Zustand einer Person zu ermitteln. Die Geräte variieren zwischen kabellosen Sensoren, welche beispielsweise über Bluetooth ihre Daten übermitteln, bis hin zu kabelgebundener Sensorik. Generell werden alle diese Sensoren am Körper der zu messenden Person angebracht. In der Regel werden Muster in Signalen ausgemacht, welche anschließend in einen Zusammenhang mit dem affektiven Zustand der Person gebracht werden. Die am häufigsten verwendeten physiologischen Sensoren im Bereich des Affective Computing sind:

- Elektrodermale Aktivität (EDA), Galvanic Skin Response (GSR), Skin Conductance (SC) die, die Hautleitfähigkeit messen. Eine Zunahme der Hautleitfähigkeit durch Schweißproduktion wird mit Erregung des Benutzers assoziiert.
- Elektrokardiogramm (EKG) und Puls (PPG) messen die Herzaktivität. Diese kann mit Aufregung, Stress oder körperlicher Anstrengung in Verbindung gebracht werden.
- Atmung (RESP) misst die Atmungsrate und das Volumen. Diese kann als Aufregung, Stress oder körperlicher Anstrengung interpretiert werden.
- Temperatur Sensor, der an der Haut angebracht wird. Dieser kann mit Aufregung, Stress oder körperlicher Anstrengung in Zusammenhang gebracht werden.
- Elektromyografie (EMG) misst die Muskelaktivität z. B. zum Erfassen von Gesichtsausdrücken wie Wut.

Spracherkennung: Emotionen können auf zwei verschiedene Arten durch Sprache erkannt werden (vgl. Kanjo u. a. (2015)). Zum einen, die explizite Analyse der Sprache. Hier wird untersucht, was gesprochen wird, Ausschlaggebend bei dieser Methode ist der Inhalt des Gesprochenen. Zum anderen, die implizite Analyse der Sprache. Bei dieser Methode ist der Inhalt des Gesprochenen unerheblich, vielmehr wird analysiert, wie es gesprochen wurde. So kann beispielsweise über die Tonlage ein positiver oder negativer affektiver Zustand bestimmt werden z.B. ob eine Person gestresst ist. Die Technologie ist jedoch abhängig von der Zielsprache bzw. der unterstützten Sprachen des Systems und ist dadurch nicht universell einsetzbar. Die Einsatzgebiete beschränken sich auf Szenarien, in denen viel gesprochen wird, beispielsweise ein Callcenter, bei der Kundenakquise und Video- bzw. Telefonkonferenzen.

2.1.2. Analysieren der erfassten Daten

Um mit Emotionen arbeiten zu können, brauchen wir eine generelle Definition, was Emotionen sind und wie sie entstehen, sowie den Unterschied zwischen Emotion, Stimmung und Affekt. Wir verwenden die Begriffe, wie sie durch die Arbeiten von Hume (2012) und Becker-Asano (2008) definiert wurden. Affekt oder affektiver Zustand ist eine weitgehende Klassifizierung von menschlichen Gefühlen und umfasst sowohl die Emotionen als auch die Stimmung einer Person. Emotionen können negativ und positiv sein, sie sind spezifisch und vielfältig. Emotionen werden durch spezifische Ereignisse ausgelöst und haben in der Regel eine zeitlich begrenzte Dauer von Sekunden oder Minuten. Meist werden sie von klar erkennbaren Gesichtsausdrücken oder körperlichen Ausdrücken begleitet. Emotionen sind meist handlungs- oder objektorientiert. Demgegenüber ist die Ursache von Stimmungen oft allgemein und unklar. Stimmungen sind länger anhaltend als Emotionen und bewegen sich im Bereich von Stunden und Tagen. Sie sind das Resultat vieler spezifischer negativer und positiver Emotionen und werden im Allgemeinen nicht durch ausgeprägte körperliche Ausdrücke gezeigt.

Es existiert eine Vielzahl verschiedenster Emotionstheorien, die über die Jahrhunderte und Jahrzehnte entstanden sind und bis zum heutigen Zeitpunkt diskutiert werden. Emotionen werden in der Regel in zwei Klassen unterschieden, basierend auf den Theorien von Antonio R. Damasio (zitiert durch Becker-Asano (2008) und Huang u. a. (2015)).

Als Primäremotionen werden Emotionen bezeichnet, welche bereits Kindern im Alter von unter einem Jahr zugeschrieben werden können. Diese werden von deutlichen Gesichtsausdrücken begleitet, welche unabhängig von Kultur und ethnischer Herkunft identifizierbar sind, beispielsweise Angst, Wut, Ekel, Traurigkeit und Fröhlichkeit. Emotionsmodelle, welche auf Primäremotionen basieren sind Ekman (1992) oder Robert Plutchik (zitiert durch Becker-Asano (2008) und Huang u. a. (2015)).

Sekundäremotionen sind ein Produkt komplexer kognitiver Prozesse, basierend auf sozialem Umfeld, vorherigen Erfahrungen und Erwartungen. Beispiel hierfür sind Emotionen wie Stolz oder Scham. Oft werden diese von Primäremotionen und Gesichtsausdrücken begleitet.

Basierend auf dem Emotionsmodell von Ekman ist das Facial Action Coding System (FACS) entstanden (siehe Ekman und Friesen (1977)). In FACS werden die gezeigten Gesichtsausdrücke einer Person in Action Units (AU) aufgeteilt. Die AUs sind Gesichtsmerkmale anhand dessen Primäremotionen identifiziert werden können. Systeme wie Affectiva (siehe Affectiva (2017)) nutzen FACS. Mithilfe maschinellen Lernens und Tausenden von vorher gekennzeichneten Beispielvideos wird das System trainiert. Meist wird zusätzlich zu den bestehenden Primäremotionen ein neutraler Zustand hinzugefügt, um einen Grundzustand, bei dem keine Emotion erkannt werden kann zu definieren.

Zusätzlich zu den diskreten Modellen, wie sie durch Ekman oder Plutchik festgelegt wurden, gibt es die kontinuierlichen Modelle, welche Emotionen auf zwei- oder dreidimensionalen Räumen abbilden. Diese klassifizieren die Emotionen nicht in Kategorien, sondern bilden sie auf Punkte in einem mehrdimensionalen Raum ab. Nach der Theorie von Wundt (vgl. Becker-Asano (2008)), werden Emotionen auf einen kontinuierlichen dreidimensionalen Raum abgebildet. Die Achsen: Lust - Unlust, Erregung - Beruhigung und Spannung - Lösung, wobei keinem der Punkte eine Emotion zugeordnet wurde. Ein anderes Beispiel für einen dimensionalen Raum ist das PAD Modell (Pleasure Arousal Dominace) nach Russell und Mehrabian (vgl. Becker-Asano (2008)). Dieses Modell hat die Dimensionen Angenehm - Unangenehm, Erregt - Schläfrig, Dominant - Unterwürfig. Des Weiterem gibt es noch zweidimensionale Modelle mit den Dimensionen Pleasure und Arousal (Lust und Erregung). Die Koordinaten dieser Modelle können nachträglich in konkrete Emotionen überführt werden. Da diese Systeme jedoch kontinuierlich sind, ist keine eindeutige Zuordnung der Koordinaten zu einer konkreten Emotion mehr möglich, beispielsweise aufgrund gleicher Entfernungen mehrere Emotionen, die Infrage kommen könnten. Deswegen muss abhängig von der Entfernung zu den Emotionen die Wahrscheinlichkeit berechnet werden.

2.1.3. Anwenden von Affective Computing

Die erfassten Emotionen können die unterschiedlichsten Anwendungsbereiche bereichern. Im Marketing können erfasste Daten dazu verwendet werden, die Effektivität von Produktwerbung zu analysieren. In der Unterhaltung und Softwareentwicklung können Emotionen dazu verwendet werden, um sich an das aktuelle Befinden einer Person anzupassen oder Hilfestellung zu bieten. Insbesondere interaktive Medien (vgl. 2.2) und Forschungsbereiche wie Compainion Technology (vgl. 2.3) können von Emotionen profitieren. In dem Bereich der Gesundheitsfürsorge können Applikationen entwickelt werden welche beispielsweise zur Behandlung von Depressionen eingesetzt werden oder Personen mit Asperger Syndrom unterstützen können.

2.2. Affective Gaming

Der Teilbereich Affective Gaming verwendet Techniken und Forschungsergebnisse des Affective Computing um die Immersion in Computerspiele zu erhöhen und neue Spielerfahrungen zu erschaffen. Nach der Definition von Gilleade u. a. (2005a) ist Affective Gaming eine Form von Gameplay, in dem der aktuelle emotionale Zustand des Spielers benutzt wird, Spielmechaniken

zu verändern. Grundlegend können wir zwei Use-Cases für den Einsatz von Emotionserkennung in Spielen identifizieren.

Adaptive Spiele: Das Spiel verwendet die erkannten Emotionen, um den Verlauf des Spiels zu verändern. Die Emotionen können als Steuerung für Charaktere verwendet werden. Ein Charakter kann gezeigte Emotion imitieren oder sich beispielsweise nur dann bewegen, wenn der Benutzer für die Steuerung vorgesehene Emotion zeigt. Anhand gezeigter Emotionen kann der Geschichtsverlauf des Spiels gesteuert und verändert werden. Spielmechaniken können abhängig von Emotionen an die Bedürfnisse eines Benutzers angepasst werden. Der Schwierigkeitsgrad kann dynamische angepasst werden, um den Spielspaß zu maximieren. KI-Agenten können Figuren überzeugender porträtieren und anhand der Emotionen des Spielers reagieren.

Qualitätssicherung: Die Analyse der Emotionen kann dabei helfen, Probleme im Spieldesign zu identifizieren. Es kann festgestellt werden, ob Spielkonzepte wie beabsichtigt funktionieren. Nicht funktionierende Ansätze können frühzeitig erkannt und überarbeitet werden, wodurch die Qualität des Produktes verbessert werden kann. Generell kann der emotionale Einfluss auf den Spieler besser abgeschätzt und mit der beabsichtigten Wirkung abgeglichen werden.

2.2.1. Beispiele für Affective Gaming

In diesem Abschnitt werden ausgewählte Veröffentlichungen im Forschungsbereich Affective Gaming vorgestellt. So gibt es Arbeiten, die sich rein auf physiologischen Sensoren konzentrieren.

In Negini u. a. (2014) wird ein First Person Shooter vorgestellt, in dem sich der Spieler gegen Zombies wehren muss. Über den affektiven Zustand des Benutzers werden die Fähigkeiten des Spielers, der Gegner und die Umgebung angepasst. Der affektive Zustand des Spielers wird durch das Messen von GSR (Galvanic Skin Response) bestimmt. Die Arbeit Vachiratamporn u. a. (2014) stellt ein Survival Horror Spiel vor, welches mit EEG und EKG arbeitet. Die Aufgabe des Spielers ist es, einem Monster zu entkommen und acht Seiten einzusammeln, die in der Spielwelt verteilt sind. Nach dem Einsammeln aller Seiten, werden die Fähigkeiten des Monsters erhöht, die ihm dabei helfen den Spieler zu fangen. Anhand des affektiven Zustands werden die Spielereignisse und deren Häufigkeit angepasst.

Parnandi u. a. (2013) ist ein Fahrsimulator, in dem die Steuerung des Autos und die Umgebung durch die Messergebnisse eines EDA-Sensors angepasst werden.

Im Folgenden wird die Arbeit von Nacke u. a. (2011) und das kommerziell erhältliche Spiel Nevermind (vgl. Lobel u. a. (2016)) genauer vorgestellt. Beide Arbeiten verwenden eine große Bandbreite verschiedener Sensoren, die häufig bei Arbeiten im Affective Gaming genutzt werden.

2.2.2. Biofeedback game design: using direct and indirect physiological control to enhance game interaction

In der Arbeit von Nacke u. a. (2011) wurde ein 2D Plattformer entwickelt, der mit physiologischen Sensoren gesteuert wird. Ziel war es diese Sensoren zu klassifizieren, und zu bestimmen, wie sich diese am besten einsetzen lassen. Es soll herausgestellt werden, welche Sensoren am besten zur passiven Eingabe und welche Sensoren zur aktiven Eingabe durch den Benutzer geeignet sind. Die Arbeit beschäftigte sich mit zwei konkreten Fragestellungen, die mittels einer Studie untersucht wurden.

- Wie reagieren Benutzer, wenn Sensoren eingesetzt werden, um klassische Steuerungsmöglichkeiten zu erweitern.
- Welche Arten von Sensoren eignen sich am Besten für welche Aufgaben innerhalb eines Spiels.

Spielszenario

Es wurde ein 2D Einzelspieler Spiel entwickelt, welches sich hauptsächlich über einen Standard-Controllerinput steuern lässt. Bei dem Spiel handelt es sich um einen Shooter, indem der Spieler eine Reihe von Hindernissen und Gegner überwinden muss, beispielsweise KI-Gegner, bewegende Plattformen und einen Bosskampf. Durch Speicherpunkte kann der Spieler seinen Fortschritt speichern. Um den Spieler dazu zu motivieren seinen Gegner zu besiegen, werden Speicherpunkte nur aktiv, wenn alle Gegner bis zu diesem Speicherpunkt hin besiegt wurden.

Steuerung über Sensoren

Das Spiel besitzt fünf verschiedene Spielmechaniken die über Sensoren direkt oder indirekt kontrolliert werden (siehe 2.2.2). Es wurden zwei verschiedene Sensorkonfigurationen während der Studie getestet (siehe 2.1). Die veränderbaren Spielmechaniken werden im Folgenden aufgelistet:

1. Größe der Gegner.

2. Anpassung der Waffen. In dem Spiel stehen dem Spieler drei Waffen zur Verfügung, eine dieser Waffen ist ein Flammenwerfer, dessen Flammenlänge durch die Sensoren angepasst werden kann.
3. Geschwindigkeit und Sprunghöhe des Charakters.
4. Schneefall und Boss Schwierigkeit. Die Stärke von Schneefall wird durch die Sensoren gesteuert. Ein starker Schneefall bedeutet eingeschränkte Sicht für den Spieler. Zusätzlich wird die Bewegungsgeschwindigkeit des Bossgegners angepasst, welcher sich bei niedrigem Schneefall langsamer bewegt.
5. Medusa Gaze ist ein Powerup, das durch den Spieler eingesammelt werden kann und 20 Sekunden aktiv ist. Das Powerup ermöglicht es dem Spieler Gegner und bewegende Plattformen einzufrieren. Zur Steuerung wurde ein Eyetracker verwendet. Die Gegner und Plattformen werden über die Blickrichtung anvisiert.

Es wurden nahezu alle im Fachbereich beliebten Sensoren in der Studie untersucht und durch verschiedene Konfigurationen deren Eignung als Eingabegeräte geprüft (siehe 2.1.

- Eyetracker wird als direkter Input verarbeitet.
- Elektromyografie (EMG) wird als direkter Input verwendet werden.
- Elektrodermale Aktivität (EDA) verwendet als indirekter Input.
- Elektrokardiografie (EKG) kann nicht direkt durch den Benutzer kontrolliert werden und wird als indirekte Steuerung verwendet.
- Atmung (RESP) kann zu einem gewissen Maße direkt durch den Benutzer beeinflusst werden.
- Temperatur Sensor angebracht an der Haut ein indirekter Sensor aber innerhalb der Studie als direkte Steuerungsart verwendet z. B. durch Pusten zur Erhöhung der gemessenen Temperatur durch den Sensor.

Zusammenfassung

Die Autoren geben zwei Hauptaussagen in ihrem Fazit, wenn es darum geht, Spiele mit physiologischen Sensoren zu erweitern. Erstens sollten die Interpretation der direkten physiologischen Sensoren auf natürliche Art eine Aktion in der Spielwelt reflektieren (Medusa Gaze). Zweitens werden indirekte Sensoren am besten verwendet, um die Spielumgebung zu verändern (Schneefall).

Spielmechanik	Konfiguration 1	Konfiguration 2
Größe der Gegner	RESP	GSR
Anpassung der Waffen	EKG	EMG
Geschwindigkeit und Sprunghöhe	TEMP	EKG
Schneefall und Boss	GSR	RESP
Medusa Gaze	Gaze	Gaze

Tabelle 2.1.: Sensorkonfiguration

2.2.3. Nevermind

Nevermind ist eines der wenigen bereits existierenden kommerziellen Spiele, das Techniken und Theorien aus dem Affective Computing einsetzt. Entwickelt wurde das Spiel von Flying Mollusk (siehe Reynolds (2016)). Das Spielszenario von Nevermind ist ein First-Persion Horror Game, in dem durch das Messen des Herzschlages die Spielumgebung angepasst wird .

Spielszenario

Der Spieler schlüpft in die Rolle eines Neuroprober, ein Therapeut in der Zukunft. Dieser kann in die Träume seiner Patienten eindringen, welche an ernsthaften psychischen Krankheiten leiden.

Jeder dieser Träume stellt ein Level dar. Ziel eines Levels ist es, die Ursache für das psychische Leiden des Patienten zu finden. Um das Ziel zu erreichen, muss der Spieler durch das Level navigieren und Rätsel lösen. Durch das Lösen dieser Rätsel bekommt der Spieler Hinweise, die ihm helfen, das finale Rätsel am Ende eines Levels zu lösen. Das Lösen des finalen Rätsels erklärt den Ursprung der Krankheit des Patienten.

Ziel ist es, den Spieler unter Stress zu setzen und ihm zu vermitteln, wie er seinen Stress kontrolliert. Der Spieler soll üben, stressvolle Situationen in seinem täglichen Leben zu bewältigen.

Verwendung von Sensoren

In Nevermind wird die Herzrate des Benutzers gemessen und die Varianz der Herzrate verwendet um das Spiel zu verändern. Dabei wird die negative Erregung als Indikator für Stress des Benutzers interpretiert. Eine vollständige Auflistung aller verwendbaren Sensoren findet sich unter Reynolds (2016).

Über den Stress wird der Schwierigkeitsgrad des Spiels gesteuert. Als visuelles Feedback für den Spieler wird ein Statik Effekt über im Bild angezeigt. Je höher der Stress, desto intensiver

dieser Effekt. Zudem wird die Umgebung dem Spieler gegenüber zunehmend feindlicher. So wird die Sicht eingeschränkt, die Beweglichkeit vermindert oder dem Spieler Schaden zugefügt.

Zusammenfassung

Die Arbeit beschrieb ein Horrorspiel, in dem physiologische Sensoren verwendet wurden, um die Spielerfahrung für den Benutzer zu individualisieren und die Immersion innerhalb der Spielwelt zu erhöhen. Ziel war es den Benutzer zu trainieren Stress volle Situationen zu bewältigen und seine Emotionen zu kontrollieren, um so im Alltag ähnlich Stress volle Situationen besser zu verarbeiten. Der emotionale Input des Spielers wird sinnvoll eingesetzt um die Umgebung abhängig von dem Stresslevel des Benutzers zu verändern. Die Umgebung hat einen großen Anteil an der Immersion die ein Spiel vermitteln kann, jedoch hat der emotionale Input des Benutzers keine Auswirkungen auf das verhalten von KI-Agenten.

2.2.4. Zusammenfassung

Die vorgestellten Arbeiten beschreiben, wie Sensoren und Techniken des Affective Computing eingesetzt werden können, um die Spielerfahrung zur erweitern. Es wird eine große Variation von Sensoren verwendet, um verschiedene Spielmechaniken anzupassen. Jedoch werden die gemessenen Sensordaten oft nicht direkt mit Emotionen assoziiert und interpretiert. Die Sensordaten werden teilweise als Stellgrößen für einfache Regler verwendet. Ein großes Augenmerk sollte auf die angemessene Verwendung dieser Sensoren gelegt werden, weil diese zwar nicht für jegliche Art von Interaktion geeignet sind, in Maßen und richtig eingesetzt jedoch die Spielerfahrung verbessern können.

Es wurde gezeigt, das sich Arbeiten im Bereich des Affective Gaming häufig darauf konzentrieren, Teile der Spielwelt und/oder den Schwierigkeitsgrad des Spiels in Abhängigkeit des affektiven Zustands anzupassen. Es gibt viele Arbeiten, die dies bereits erfolgreich umgesetzt haben. Das Ziel dieser Arbeit ist es, das Verhalten der KI-Agenten durch emotionalen Input zu verändern. Dieser Aspekt wird oft nicht beachtet trägt jedoch einen großen Anteil zur gesamten Spielerfahrung bei. So muss ebenfalls über die Verwendbarkeit und Verfügbarkeit für potenzielle Benutzer nachgedacht werden. Betrachtet man diesen Aspekt, scheinen Spracherkennung und kamerabasierte Systeme am vielversprechendsten. So findet man heutzutage Mikrofone und Kameras in nahezu allen Smartphones und Laptops.

Ein generelles Problem des Affective Gaming ist es, dass jedes Projekt sein eigenes Spiel implementiert, was einen Vergleich der verschiedenen Lösungsansätze erschwert. Zudem müssen erhebliche Ressourcen aufgewendet werden, um ein Spiel umzusetzen, das in der Regel

keine Konkurrenz zu den in der Industrie entwickelten Spielen darstellen. In einem Bereich, der sich primär mit der Immersion von Spielern beschäftigt, ist dies besonders hinderlich. So müssten für optimale Versuchsbedingungen der Sound, die Grafik und die Spielmechaniken auf dem neusten Stand der Technik sein. Dies ist jedoch ein Anspruch, der für Einzelpersonen oder kleine Projektgruppen kaum umsetzbar ist. In anderen Bereichen der künstlichen Intelligenz werden, auch in Kooperation mit der Industrie, Plattformen entwickelt wie Wymann u. a. (2017) und DeepMind (2017), die den Forschern diese Hürde abnehmen und eine vergleichbare Versuchsumgebung für alle Forschungsgruppen bereitstellen. So können sich die Forschungen auf die eigentliche Fragestellung konzentrieren.

2.3. Companion Technology

Das Konzept der Companion Technologie ist ein System, das dem Benutzer als Gefährten dient. Der Sonderforschungsbereich Transregio 62 der Universität Ulm beschäftigt sich intensiv mit dem Thema der Companion Systeme. Aus der Beschreibung der Ziele (vgl. 62 (2017)) dieser Forschungsgruppe lassen sich vier Merkmale entnehmen, die ein solches System erfüllen sollte:

1. Die Funktionalität des Systems wird auf den jeweiligen Benutzer individuell abgestimmt. Das System orientiert sich dabei an den Fähigkeiten, Vorlieben, Anforderungen und aktuellen Bedürfnissen des Benutzers.

2. Der Companion stellt sich auf die Situation und emotionale Befindlichkeit des Benutzers ein.

3. Der Companion ist stets verfügbar, kooperativ und vertrauenswürdig.

4. Der Companion tritt seinem Nutzer als kompetente und partnerschaftliche Dienstleister gegenüber.

Die oben genannten Punkte beschreiben ein System, das dem Benutzer als Assistent oder Gesellschaft dient. Es nicht verbindlich, dass dieses System durch einen physikalischen Körper repräsentiert wird. So kann es sich bei einem Compainion beispielsweise um einen Assistenten Handeln, der einem hilft, seine Termine zu planen oder ein virtueller Begleiter innerhalb eines Spiels, der dem Spieler beim Lösen schwierige Aufgaben unterstützt. Eines der wichtigsten Merkmale ist der Punkt 4. Der Benutzer sollte dem Companion nie überdrüssig werden, ihn als Belastung oder als störend ansehen. Eines der bekanntesten Beispiele ist der Office Assistant Clippy, welcher von Microsoft 1997 in Office eingeführt und schlussendlich mit dem Erscheinen von Office XP eingestellt wurde (vgl. Microsoft (2001)). Dieser galt bei vielen Benutzern als

lästig und störend anstatt als hilfreicher Begleiter. Der Punkt 2 unterstreicht die Wichtigkeit von Emotionserregung für Companion Systeme. Dies betrifft sowohl ein adaptives Verhalten beim Erkennen von Emotionen, als auch das Darstellen von Emotionen als Reaktionen auf Benutzerinteraktionen.

2.3.1. Problemstellungen

Companion Technology ist ein interdisziplinäres Feld und stellt vielseitige Problemstellungen an eine Vielzahl von Forschungsbereichen der Informatik, Robotik, Psychologie und weiterer Fachbereiche. So müssen Sensordaten zur Erkennung der Umwelt entwickelt werden. Die anfallenden Daten müssen erfasst und ausgewertet werden. Um sich auf Anforderungen und aktuellen Bedürfnissen des Benutzers einstellen zu können, bedarf es emotionales Empfinden und Empathie durch Emotionserkennung, sowie Algorithmen zur Planung und Entscheidungsfindung, um intelligente Entscheidungen zu treffen. Nur ein lernendes System kann sich auf die individuellen Bedürfnisse der Benutzer einstellen und vorherige Erfahrungen in die Entscheidungsfindung und Pläne mit einbeziehen. Für Konversation und physische Interaktion sollte das System über moderne Techniken der Human-Computer Interaktion verfügen.

2.3.2. Anwendungsbereiche

Eine umfangreiche Übersicht über den Themenbereich der Companion Technology wurde von Biundo u. a. (2016) erstellt. Diese beinhaltet eine ausführliche Liste an Referenzen zu Forschungsgruppen. Aktuelle und beendete Projekte können dem Dokument entnommen werden. Die Anwendungsbereiche der Companion Technologie werden in folgende Kategorien unterteilt:

Robotik: Beispiele für Companions, die unter den Bereich der Robotik fallen sind intelligente Spielzeuge, Haushaltsroboter und Gesundheitspflege. Beispiele für Haushaltsroboter sind u.a Saugroboter, Küchenassistenten, Barkeeper. In die Kategorie der Gesundheitspflege fallen Systeme die Alzheimerpatienten unterstützen z.B. Rehabilitations- und Trainingsroboter, haustierähnliche soziale Roboter zu Therapiezwecken wie z.B. Essstörungen oder Demenz. Aber auch Roboter, die beim Essen, Anziehen und Zähneputzen helfen oder bei anderen alltäglichen Aufgabe Menschen mit eingeschränkten motorischen oder geistigen Fähigkeiten helfen können.

Intelligente Umgebungen und Häuser: Systeme, die ihr Verhalten an die gegebenen Situationen und Benutzervorlieben anpassen. Diese können auch bei Unfällen im Haushalt

die Ambulanz kontaktieren, Kochrezepte vorschlagen, die Beleuchtung verändern oder Musik abspielen.

Fahrassistenten: Autos, die sich an die individuellen Vorlieben ihres Fahrers anpassen, den Sitz einstellen, die Musik steuern oder Fahr- und Navigation-Assistenz bieten. Diese Systeme können auf die Umgebung reagieren und automatisch Bremsen und Hindernissen ausweichen oder dem Fahrer beim Parken unterstützen. Weitere automatische Fahrsysteme haben die Möglichkeit ihre Geschwindigkeit an die Verkehrsregeln aber auch an die Präferenzen des Fahrers anzupassen, z.B abhängig vom Stresslevel oder emotionalen Zustand des Benutzers.

2.3.3. Zusammenfassung

Bei der Angabe der Anwendungsbereiche wurde von Biundo u. a. (2016) der Bereich der interaktiven Medien wie Computerspiele ausgelassen. Interaktive Welten in denen Spieler unzählige Stunden verbringen und sich drin verlieren, sind überaus geeignet für den Einsatz von KI-Agenten als Companion. Die KI in Computerspielen könnten als Teilmenge des Anwendungsreichs der Robotik angesehen werden. Anders als bei der Robotik fallen einige Problemstellungen, die bei Robotern überwunden werden müssen, weg. Die Sensorik und Verfügbarkeit von Umgebungsinformationen spielen bei Computerspielen keine Rolle, da zu jedem Zeitpunkt alle Benötigten Information zur Verfügung stehen. Da alle motorischen Fähigkeiten simuliert werden, muss keine Motorik entwickelt und angesteuert werden. Die Problemstellungen der Emotionserkennung und Entscheidungsfindung sowie die Mensch-Maschinen-Interaktion über Gesichtsausdrücke, Sprache und Gestiken bleiben erhalten.

Durch die Anwendung der Ansätze aus der Compainion Technologie können virtuelle Charaktere erstellt werden, die über die Funktion eines simplen Gegners hinausgehen. Diese besitzen das Potenzial die Immersion des Spiels zu erhöhen, indem sie dem Spieler als hilfreicher Begleiter zur Seite stehen, glaubwürdiger reagieren und so unter Umständen sogar ermöglichen, dass Spieler eine emotionale Bindung zu diesem Charakter aufzubauen, ähnlich wie bei einem Filmcharakter.

3. Das Emotion Bike

Das Emotion Bike Projekt ist eine Arbeitsgruppe an der HAW Hamburg, die sich mit der Erforschung von Mensch-Maschine-Interaktion befasst. Aus diesem Grund wurde eine Laborumgebung konzeptioniert und entwickelt. Mithilfe verschiedenster Messtechniken können Gestiken und Emotionen von Probanden erfasst und analysiert werden. Hierzu werden Kameras eingesetzt, die den Probanden filmen. Zusätzlich werden Sensoren aus der medizinischen Diagnostik wie Körpertemperatur, Puls (PPG) und Hautleitfähigkeit (EDA) eingesetzt. Über einen Eyetracker können die Augenbewegungen und Pupillenerweiterung gemessen werden. Der Versuchsaufbau beinhaltet ein Ergometer über das ein Computerspiel gesteuert werden kann. Es wird versucht, durch den Einsatz physikalischer und virtueller Interaktion den Probanden emotional zu involvieren. Ziel ist es, zu untersuchen, wie sich die Technologie auf die individuellen Bedürfnisse eines Benutzers einstellen kann. Hierfür werden die durch Versuche erfassten Daten ausgewertet und die Beziehung zwischen der Interaktion mit dem System und der erfassten Emotionen analysiert. Die Emotionserfassung wird mit Hilfe von Algorithmen und Techniken aus dem Affective Computing durchgeführt. Hierfür wurden externe Tools als auch Eigenentwickelungen herangezogen.

3.1. System und Versuchsaufbau

Der Aufbau des Emotion Bike besteht aus vier Komponenten, die über eine Netzwerkschnittstelle miteinander kommunizieren (Abbildung 3.1). Über den Exergame Controller kann das Spiel gesteuert werden. Das Bild 3.2 zeigt den Versuchsaufbau. Zu sehen ist das Ergometer mit einer selbst erstellten Lenkvorrichtung, die für das Emotion Bike entwickelt wurde. Die Daten des Ergometers und der Lenkvorrichtung werden über einen Rasberry Pi mit Gertboard verarbeitet und anschließend in das Netzwerk gesendet. Durch den stationären Aufbau wurde sichergestellt, dass sich das Gesicht des Probanden während einer Versuchsdurchführung immer im Kamerafokus befindet. Die Visualisierung empfängt die Daten, die von dem Exergame Controller gesendet wurden. Diese werden anschließend in Lenkbefehle und Beschleunigungswerte für die Simulation umgewandelt. Es handelt sich hierbei um eine Fahrradsimulation, welche den

Probanden vor verschiedene Aufgaben stellt, die es zu lösen gilt. Im Spiel vordefinierte Ereignisse werden sobald vom Benutzer ausgelöst an das Data Aquisition System übertragen. Eine Kamera erfasst während der Versuchsdurchführung das Gesicht des Probanden. Die Videodaten werden für die spätere Auswertung lokal gespeichert. Über Biosensoren wird die Körpertemperatur, Puls, Hautleitfähigkeit gemessen. Die Daten der Sensoren werden über Netzwerk an das Data Aquisition System übertragen und gespeichert. Alle Daten werden mit einem Zeitstempel versehen, sodass bei der Analyse der zeitliche Zusammenhang zwischen Sensordaten und Spielereignissen nicht verloren geht. Es findet keine Datenanalyse oder Emotionserkennung in Echtzeit statt. Die gespeicherten Daten können nach der Versuchsdurchführung analysiert werden. Eine detailliertere Beschreibung der Architektur und des Spiels wird in der Arbeit von Müller u. a. (2015) geschildert.

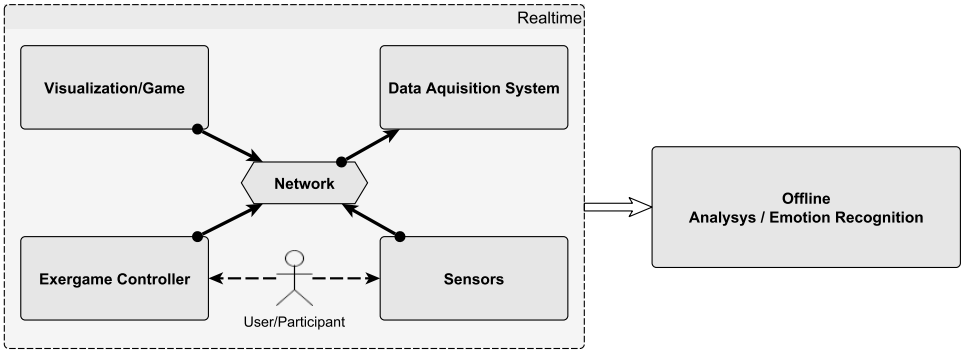

Abbildung 3.1.: Emotion Bike Architektur

3.2. Erste Fallstudie

Die erste Veröffentlichung innerhalb des Emotion Bike Projektes ist die Veröffentlichung EmotionBike: A Study of Provoking Emotions in Cycling Exergames (siehe Müller u. a. (2015)). Ziel dieser Untersuchungen war es zu ermitteln, ob eine direkte Korrelation zwischen den gezeigten Emotionen einer Person und den Ereignissen innerhalb des Spiels existiert. Des Weiteren wurde analysiert, ob vorherige Annahmen über den emotionalen Einfluss der Ereignisse und Situationen zutreffen und welche personenbezogenen Unterschiede festzustellen sind. Zusätzlich wurde untersucht, welche Unterschiede die einzelnen Probanden in ihrer emotionalen Reaktion und Intensität zeigten.

Abbildung 3.2.: Emotion Bike Versuchsaufbau

3.2.1. Versuchsdurchführung

Bei den Versuchen wurden die Probanden gebeten, fünf verschiedene Szenarien des Fahrradsimulators zu spielen. Während der Versuchsdurchführung wurde das Gesicht der Probanden mit einer Kinect 2[1] aufgezeichnet. Zu Beginn der Versuche wurden die Probanden gebeten, einen Fragebogen auszufüllen. Nachdem ausfüllen des Fragebogen, wurden die Probanden darüber informiert, dass sie die Versuche jederzeit abbrechen können. Die Testpersonen wurden während der gesamten Versuchsdurchführung von einem Projektmitglied begleitet. Als Erstes wurden die Testpersonen mit der Steuerung vertraut gemacht. Sie wurden instruiert, wie die Steuerung funktioniert. Anschließend wurde ein Testszenario gestartet, in dem die Testpersonen mit der Steuerung vertraut machen konnten. Nachdem die Testpersonen ihre Bereitschaft erklärt hatten, wurden die Versuche durchgeführt. Während der Versuche wurden die gezeigten Emotionen von einem Beobachter protokolliert. Das erstellte Protokoll diente als Referenzgrundlage für die anschließenden Analysen. Nachdem Abschluss jedes Szenarios wurden die Probanden gebeten, eine persönliche Einschätzung über die gefühlten Emotionen zu geben.

Probanden: Die Versuche wurden mit 11 Probanden durchgeführt, drei weibliche und acht männliche Testpersonen nahmen an den Versuchen teil. Die Alterspanne der Probanden lag zwischen 19 und 41 Jahren, das Durchschnittsalter bei 27 Jahren.

Der vorher ausgefüllte Fragebogen beinhaltete allgemeine personenbezogene Fragen wie Geschlecht, Alter, körperliche Fitness und Erfahrungen mit Videospielen. Zusätzlich wurde ein Fragebogen ausgefüllt, dessen Ziel es ist die Charaktereigenschaften einer Person zu bestimmen. Durch Beantwortung des Fragebogen konnten die Probanden nach dem Big Five Personality Model, in Gruppen kategorisiert werden. Die Kategorien und eine kurze Beschreibung kann der Tabelle 3.1 entnommen werden.

Die Analyse der Fragebögen zeigte, dass sieben Personen regelmäßig Videospiele konsumieren. Diese Testpersonen gaben an, zwischen 2 bis 3 Stunden pro Woche zu spielen und Kategorisierten sich selber als Gelegenheitsspieler. Der Rest der Probanden gab an, dass sie nicht regelmäßig spielen. Ebenfalls wurde erfasst, ob die Testpersonen Erfahrungen mit alternativen Steuerungsmöglichkeiten haben. Als Beispiele wurden den Personen die Wii Remote, Microsoft Kinect oder Lenkradsteuerung genannt. 7 von 11 Probanden bejahten diese Frage, es konnte kein Unterschied zwischen den Gruppen festgestellt werden. Alle Probanden waren mit einem Fahrrad vertraut und hatte keine Probleme bei der Versuchsdurchführung.

[1] https://developer.microsoft.com/en-us/windows/kinect

Kategorie	Beschreibung
Openness to Experience	Die Neigung, dass eine Person einfallsreich, neugierig, einfühlsam und/oder kreativ ist.
Conscientiousness	Die Neigung, dass eine Person organisiert, gründlich und/oder zuverlässig zu sein.
Extroversion	Die Neigung, dass eine Person sozial, offen und/oder durchsetzungsfähig ist.
Agreeableness	Die Neigung, dass eine Person freundlich, großzügig, sympathisch und/oder selbstlos ist.
Neuroticism	Die Neigung, dass eine Person angespannt, ängstlich und/oder launisch ist.

Tabelle 3.1.: Big Five Personality Model

Versuchsszenarien: Für die Versuche wurden 5 Szenarien vorbereitet. Jedes der Szenarien hatte das Ziel eine spezifische Emotion in den Probanden hervorzurufen. In der Tablle 3.2 sind die Szenarien, Kurzbeschreibung und die erwarteten Emotionen aufgelistet. Generell ist die Aufgabe für den Spieler in jedem Szenario, mit dem Fahrrad von der Startlinie bis zur Zielleine zu gelangen. Auf dem Weg zur Ziellinie werden dem Probanden verschiedene Hindernisse in den Weg gestellt.

3.2.2. Fazit

Das Videomaterial, das während der Versuche aufgenommen wurde, wurde mit der Emotionserkennungssoftware CERT (siehe Littlewort u. a. (2011)) analysiert. Die Software analysiert die Einzelbilder und bewertet die Wahrscheinlichkeit, dass ein Gesichtsausdruck eine von acht Emotionen darstellt. Für die Emotionen Freude, Ekel, Wut, Furcht, neutral, traurig, Überraschung, und Verachtung werden die Wahrscheinlichkeiten im Bereich von 0 bis 1 berechnet. Da sich die Wahrscheinlichkeiten der Gesichtsausdrücke zwischen Personen signifikant unterscheiden können, wurden die Ergebnisse der Analysesoftware, mit den Ergebnissen des Beobachters und der Selbsteinschätzung der Probanden verglichen.

Die Wahrscheinlichkeitswerte wurden mit den erwarteten Emotionen verglichen. Hierfür wurde ein Zeitfenster, um die Ereignisse, die durch das Spiel generiert wurden, festgelegt. Die Dauer des Zeitfensters wurde auf 3 Sekunden festgelegt, 0.5 Sekunden vor dem Eintreten eines Ereignisses und 2.5 Sekunden danach. Innerhalb dieses Zeitfensters wurde die Emotion mit der höchsten Wahrscheinlichkeit betrachtet, wobei die Wahrscheinlichkeit einen Wert von 0.5 überschreiten musste. Die detaillierten Auswertungen und erstellten Diagramme sind in

Szenario	Erwartete Emotion	Beschreibung
Level1	Joy	Hierbei handelt es sich um ein Dorf mit einem Rundkurs. Auf der Straße laufen Teddybären in allen möglichen Richtungen. Ziel ist es den Teddybären auszuweichen und möglichst keine zu verletzten (Abb. B.1a). Das Ereignis einer Kollision wird von dem System für die Auswertung gespeichert. Bei einer Kollision wird das Rad zum Stillstand gebracht und der Benutzer muss vom Stillstand aus das Rad erneut beschleunigen. Dadurch resultiert ein erhöhter Widerstand in den Pedalen des Ergometers.
Level2	Surprise, Joy	Der Spieler wird mit einem Slalom konfrontiert. Über den Kurs wurden 20 Münzen verteilt. Das Ziel ist es, die 20 Münzen einzusammeln, bevor die Ziellinie durchquert werden kann (Abb. B.1a). Durchquert der Spieler die Ziellinie vorzeitig, wird er an den Anfang des Levels teleportiert.
Level3	Surprise, Anger, Joy, Fear	Um die Ziellinie zu erreichen, muss der Spieler einen Graben überwinden (Abb. B.2). Der Spieler kann hierfür einen Beschleuniger und eine Sprungschanze verwenden. Wird der Beschleuniger durchquert, reicht die Geschwindigkeit des Rads aus, um über die Sprungschanze die andere Seite und damit die Ziellinie zu erreichen. Es erfordert Geschicklichkeit, die andere Seite zu erreichen. Aus diesem Grund wird nach jedem Versuch die Breite der Sprungschanze angepasst, um den Spieler die Aufgabe zu erleichtern.
Level4	Surprise, Fear	Der Spieler muss einen Gebirgspass hochfahren, um zum Ziel zu gelangen (Abb. B.3b). Während der Auffahrt erhöht sich der Widerstand der Pedalen, wodurch der Spieler körperlich höher beansprucht wird. Ist der Spieler oben angekommen, wird er von Spinnen angegriffen. Diese werden hochgewachsenes Gras vor dem Spieler versteckt, sodass sie erst kurz vor der Attacke erkennbar werden.
Level5	Disgust, Fear, Joy, Surprise	Der Spieler befindet sich in einem dunklen und dichten Wald. Durch leuchtende Münzen wird der Spieler zum Ziel geleitet (Abb. B.3b). Kurz bevor der Spieler das Ziel erreicht, tauchen plötzlich Monster aus dem nichts auf und ein lautes Kreischen ertönt. Der Spieler wird für kurze Zeit festgehalten und ist handlungsunfähig.

Tabelle 3.2.: Testszenarien des Emotion Bike Spiel

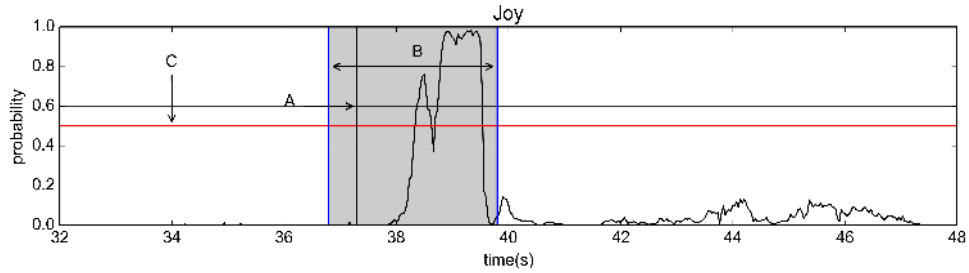

Abbildung 3.3.: Level 5 Jumpscare Event (Quelle Müller u. a. (2015)). A: Durch das Spiel erzeugtes Ereignis. B: Zeitfenster um das Ereignis herum. C: Schwellwert für erkannte Emotionen

der Veröffentlichung Müller u. a. (2015) zu finden. Die Abbildung 3.3 zeigt die Analyse eines Versuch für das Szenario Level5.

Die Untersuchungen der Fallstudie lassen die Vermutung zu, dass eine direkte Verbindung zwischen Ereignissen im Spiel und über Gesichtsausdrücke gezeigte Emotionen bestehen. So konnten innerhalb des Zeitfensters um ein Ereignis Messspitzen festgestellt werden. Die gezeigten Emotionen entsprachen jedoch nicht immer den Emotionen, die bei der Erstellung der Szenarien erwartet wurden. Zudem wurde festgestellt, dass die Intensität und die gezeigten Emotionen zwischen den Testpersonen variieren. Es wurde festgestellt, dass die emotionale Auswirkung von Ereignissen mit ihrer Häufigkeit abnehmen kann. Durch die Bestimmung der Persönlichkeiten konnten die individuellen Unterschiede bei der Analyse mit einbezogen werden. Es zeigte sich, das die extrovertierten Probanden intensivere emotionale Reaktionen ausdrückten.

Die Analyse der Emotionen kann ein hilfreiches Werkzeug bei der Entwicklung von Spielen sein. Ein Spiel kann angepasst werden, um die durch die Designer beabsichtigten Reaktionen des Spielers zu erzeugen. Elemente und Spielideen, die zu Frust führen, können frühzeitig erkannt und behoben werden. Bereits implementierte Mechaniken und Elemente der Geschichte können auf ihre Wirksamkeit überprüft werden. Durch eine Echtzeiterfassung wäre es möglich, gänzlich neue Spielerfahrungen, durch emotionsbasierte Spielmechaniken entwickeln, welche die Immersion des Spielers erhöhen könnten. So können auf jeden Spieler individuell angepasste Szenarien generiert und dynamische Geschichten erzählen werden, als auch die Entwicklung emotionaler KI-Agenten, welche ihr Verhalten anhand der gezeigten Emotionen des Spielers anpassen. Dieser kann eine direkte emotionale Reaktion aufgrund seiner Handlungen oder anderen Ereignissen im Spiel erwarten, wodurch eine angemessene Reaktion anhand der Emotionen des Spielers möglich wird.

Physiological Data Analysis for an Emotional Provoking Exergame (vgl. Müller u. a. (2016)) ist die zweite Veröffentlichung, die innerhalb des Emotion Bike Projektes entstanden ist. Ziel dieser Arbeit war die Analyse der physiologischen Daten. Generell wurden die Versuche aus Müller u. a. (2015) nochmals durchgeführt und anschließend die physiologischen Daten nach der in Abschnitt 3.2.1 vorgestellten Methode untersucht. Die Ergebnisse dieser Arbeit festigten dabei die Resultate der vorherigen Untersuchungen. Es wurde gezeigt, dass die vorgestellte Methode unabhängig von den Sensoren erfolgreich angewendet werden kann. Weiter wurde gezeigt, dass durch den Einsatz von physiologischen Sensoren die Erkennung von False Positives durch die Gesichtserkennung minimiert werden kann. Emotionen, die unbewusst oder bewusst vorgetäuscht werden, können durch physiologischen Sensoren erkannt werden. Beispielsweise kann die Emotion Freude falsch erkannt werden. Es wurde nachgewiesen, dass das Empfinden von Frustration oft durch ein Lächeln überspielt wird, welches durch kamerabasierte Systeme falsch erkannt werden würde. Durch den Abgleich mit physiologischen Sensoren können solche False Positives erkannt und ausgefiltert werden. Des Weiteren können die Erkennungsraten bei introvertierten Personen erhöht werden.

Nachteil dieser Sensoren ist ihre Verfügbarkeit für Konsumenten, Kosten, in Inbetriebnahme und Anbringung an den Körper. Solche Sensoren besitzen eine hohe Zugangsbeschränkung für den Verbraucher, weshalb wir uns in dieser Arbeit auf kamerabasierte Systeme zur Emotionserkennung fokussieren. Heutzutage werden Laptop und Smartphones nahezu ausnahmslos mit einer Kamera ausgeliefert, wodurch eine hohe Verfügbarkeit ohne Zusatzkosten sichergestellt wird. Es müssen keine Vorkehrungen, wie beispielsweise das Anbringen der Sensoren am Körper, vorgenommen werden. Als einzige Bedingung muss erfüllt sein, dass sich der Benutzer im Fokus der Kamera befindet und frontal erfasst werden kann. Der Einsatzbereich der Computerspiele erfüllt diese Vorbedingung automatisch, wenn die Kamera über oder unter dem Monitor installiert wird, da sich der Benutzer beim Spielen direkt dem Monitor zuwendet und seine volle Aufmerksamkeit schenkt. Sollte der Benutzer nicht erkannt werden, kann dies durch visuelle Hinweise dem Benutzer vermittelt werden.

3.3. Towards More Robust Automatic Facial Expression Recognition in Smart Environments

Ziel dieser Arbeit Bernin u. a. (2017) ist es, die Erkennungsraten von Gesichtserkennungssoftware zu erhöhen. Hierfür wurden vier Softwares zur Emotionserkennung miteinander verglichen. Alle Programme sind kommerziell und/oder frei erhältlich. Es werden drei Datenbanken mit gekennzeichneten Videomaterial verwendet. Das Videomaterial wurde auf alle

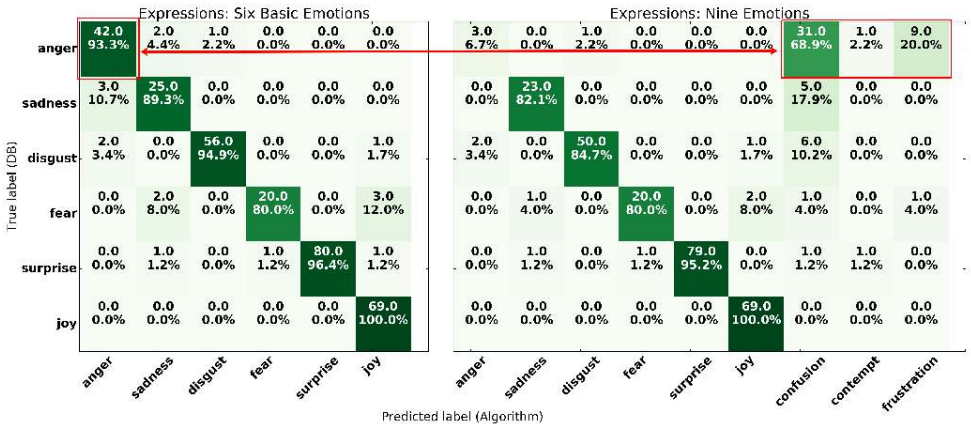

Abbildung 3.4.: Auswertung der Erkennungsraten Affectiva, InSight, CERT, Emotient. (Quelle Bernin u. a. (2017))

Datensätze, die mit den primär Emotionen Freude, Angst, Wut, Traurig, Überraschung und Ekel gekennzeichnet wurden, beschränkt.

Die Ergebniswerte der einzelnen Programme (Affectiva, InSight, CERT, Emotient) wurden zwischen 0 und 1 normalisiert, um vergleichbare Werte erzeugen zu können. Alle Programme wurden untereinander und mit dem Kennzeichnungen für die Videos verglichen (Abb. 3.4). Die y-Achse zeigt die in Datenbanken gekennzeichneten Emotionen und die x-Achse die durch eine der getesteten Software erkannten Emotionen. Es wurde nicht weiter spezifiziert, welche der aufgelisteten Software in dem Beispiel zu sehen ist. Erkennbar an der roten Umrandung ist das mit der Anzahl der zu erkennenden Emotionen sich die Erkennungsrate verringert. Anhand dieser Beobachtung wurde die Methode des Application Specific Clustering vorgestellt. Die Grundidee ist es, die verschiedenen Emotionen, abhängig von Kontext eines Programms, zu gruppieren. Unter Verwendung dieser Methode kann die Erkennungsrate erhöht werden. Bei vielen Programmen ist eine Differenzierung nach positiven und negativen Emotionen bereits ausreichend.

3.3.1. Fallbeispiele

Folgend werden die drei in dem Paper genannten Fallbeispiele aufgelistet:

Benutzerfreundlichkeitsstudien: In Benutzerfreundlichkeitsstudien sind Entwickler daran interessiert zu erfahren, ob Benutzer bei der Bedienung eines Programms positive, negative oder neutrale Emotionen zeigen. Die Emotion Joy kann als positive Emotion kategorisiert werden. In der Kategorie der negativen Emotionen können die Emotion Fear,

Anger, Disgust und Sadness zusammengefasst werden. In die Kategorie der neutralen Emotionen können Suprise und nicht erkannte Emotionen zusammengefasst werden.

Cockpit Szenario: Ein Cockpit System, wie es beispielsweise in einem Auto oder Flugzeug vorkommen könnte, sollte in der Lage sein, dem Benutzer automatische Unterstützung zu bieten, wenn diese benötigt wird. Als Indikator hierfür könnte ein negativer emotionaler Zustand, wie Stress oder die schwindende Konzentration, verwendet werden. Dieser Alarmzustand des Systems kann durch die Emotionen Anger, Fear, Disgust und Supprise repräsentiert werden. Keine erkannten Emotionen, Joy und Sadness können als normal Zustand zusammengefasst werden.

Lernende Systeme: Ein Beispiel ist der virtuelle Lehrer, der erkennen sollte, wann ein Schüler sich im Flow befindet oder nicht. Der Flow-Zustand beschreibt eine Person, die sich im Zustand absoluter Konzentration befindet. Der Zustand des Flows kann durch das Zusammenfassen der Emotionen Spurprise, None und Joy beschrieben werden. Die Emotion Sadness wird in diesem Szenario ignoriert, da sie als nicht relevant für das gewählte Szenario angesehen wird.

3.3.2. Zusammenfassung

In der Abbildung 3.5 sind alle drei beschriebenen Szenarien und deren Erkennungsraten dargestellt. Es wurden die Erkennungsraten mit allen sechs Emotionen der Clustering Methode gegenübergestellt. Es ist zu erkennen, das durch die Gruppierung der Emotionen die Erkennungsrate deutlich erhöht werden kann und im Bereich zwischen 32% bis 69% gegenüber der Erkennung einzelner Emotionen (12% bis 31%) liegt. Es ist zu beachten, dass eine Gruppierung der Emotionen nicht für jedes Programm geeignet ist. Der Ansatz der Gruppierung von Emotionen stellt eine interessante Möglichkeit dar, die Erkennungsraten zu erhöhen, solange keine genauere Differenzierung in die einzelnen Emotionen benötigt wird. Es muss untersucht werden, in welcher Form diese Methode bei der Umsetzung von emotionalen KI-Agenten eingesetzt werden kann und welche Auswirkungen es auf die Modellierung und das resultierende Verhalten des Agenten hat.

App.	Cluster	InSight	Affectiva	Emotient	CERT
usability	positive	9	97	49	97
usability	neutral	79	60	98	14
usability	negative	201	278	180	515
usability	overall	289	435	327	626
usability	**detect. rate**	**0.32**	**0.48**	**0.36**	**0.69**
cockpit	alarm	262	316	240	199
cockpit	normal	204	221	264	312
cockpit	overall	466	437	504	511
cockpit	none	148	103	150	9
cockpit	**detect. rate**	**0.51**	**0.48**	**0.55**	**0.56**
learning	inflow	262	302	265	126
learning	panic	92	90	70	58
learning	overall	354	392	395	184
learning	**detect. rate**	**0.39**	**0.43**	**0.43**	**0.34**
6 emotions overall		112	175	164	279
6 emotions detect. rate		**0.12**	**0.19**	**0.18**	**0.31**

Abbildung 3.5.: Application Specific Clustering. (Quelle Bernin u. a. (2017))

4. Künstliche Intelligenz in Spielen

Was ist künstliche Intelligenz und wie wird diese in Videospielen eingesetzt? In diesem Kapitel wird eine kurze Einführung in die künstliche Intelligenz gegeben. Diese basiert hauptsächlich auf dem Buch von Russell u. a. (2003). Anschließend werden die gebräuchlichsten KI-Techniken der Computerspielindustrie betrachtet.

4.1. Vier Arten künstlicher Intelligenz

Die KI kann grundsätzlich in vier verschiedene Ansätze kategorisiert werden:

Systeme die wie Menschen handeln: Beschreibt ein System, das Aktivitäten ausführt, die dem menschlichen Denken zugeordnet werden können. Dazu gehören Entscheidungsfindung, Problemlösungen und Lernen. Als ein Test für ein solches System wird oft der Turing-Test genannt. Das System gilt dann als ein System, das wie ein Mensch handelt, wenn es diesen Test erfolgreich absolviert.

Ziel des Turing-Tests ist es herauszufinden, ob ein Mensch das Verhalten des Systems von dem Verhalten eines Menschen unterscheiden kann. Die Testdurchführung findet ohne Sicht- und Hörkontakt statt. Der Proband kann so nicht erkennen, ob es sich um einen Menschen oder um eine Maschine handelt. Der Proband stellt dem System und einer weiteren Person eine Reihe von Fragen. Anschließend muss die Testperson anhand der gegebenen Antworten abschätzen, ob es sich bei einem der Gesprächspartner, um eine Person oder einen Computer handelt. Kann der Proband dies nicht bestimmen, hat das System den Test bestanden.

Um den Test erfolgreich bestehen zu können, muss ein Programm die folgenden Anforderungen erfüllen:

1. Verarbeitung natürlicher Sprache: Natürliche Sprache in Text und/oder Audio Form müssen verarbeitet werden können, damit das System die Fragen des Testers verarbeiten kann.

2. Wissensrepräsentation: Das System muss die Fähigkeit haben Wissen über sich und seine Umwelt zu speichern.
3. Entscheidungsfindung: Das System muss bekanntes Wissen verarbeiten und anwenden können, um Fragen zu beantworten und Problemlösungen zu generieren.
4. Maschinelles Lernen: Durch maschinelles Lernen muss das System in der Lage sein, sich an veränderte Umstände anzupassen. Es muss anhand von Reaktionen auf eigene- und Benutzerinteraktion neue Strategien und Problemlösungen entwickeln.

Des Weiteren gibt es einen totalen Turing-Test der physikalische Interaktion beinhaltet. Das System muss entsprechend um die folgende Fähigkeiten erweitert werden:

1. Computervision: Das Erkennen von Objekten in der Umgebung durch den Einsatz von Sensoren z.B. Laser/Infrarot Abstandsmesser, Kamera etc.
2. Robotik: Interaktion mit physikalischen Objekten sowie das Bewegen innerhalb eines Raumes.

Die oben genannten sechs Disziplinen, die als Anforderungen durch den Turing-Test und den totalen Turing-Test entstanden sind, bilden einen Großteil des Fachbereichs KI ab.

Systeme die wie Menschen denken: Um ein System zu entwickeln das denkt wie ein Mensch, muss ein psychologisches Modell entwickelt werden. Das Modell muss den menschlichen Denkprozess akkurat simulieren. Dafür muss die Arbeitsweise des menschlichen Gehirns verstanden werden. Dies kann durch psychologische Experimente erreicht werden oder, indem man versucht, seine eigenen Gedanken erfassen, während sie entstehen. Mithilfe eines solchen Modells ist es möglich zu versuchen ein Computerprogramm zu entwickeln, das den menschlichen Denkprozess simuliert. Stimmt das Verhalten des Systems bei gleichem Input mit dem Verhalten eines Menschen überein, ist anzunehmen, dass das System zumindest ähnlich wie ein Mensch denkt. Dabei ist zu beachten, dass nicht alle Menschen sich in einer gleichen Situation auch gleich verhalten.

Mit dieser Problemstellung beschäftigt sich der Forschungsbereich der Kognitionswissenschaften. Es werden Modelle aus der KI und experimentelle Techniken aus der Psychologie zusammengebracht, um Theorien über die Arbeitsweise des menschlichen Verstandes zu entwickeln. So ist ein Algorithmus, der gut geeignet ist eine Aufgabe zu lösen, nicht auch gleichzeitig ein gutes Modell des menschlichen Denkprozesses.

Systeme die rational denken: Ein System, das immer zu korrekten Schlüssen führt, wenn ihm eine korrekte Vorbedingung übergeben wurde. Diese Art von Systemen basieren auf

logischer Notation. Mithilfe dieser Notation werden Aussagen zu allen Arten von Dingen in der Welt und die Beziehung zwischen diesen Dingen dargestellt. Nach Russell u. a. (2003) gibt es bei diesem Ansatz zwei große Hindernisse. So stellt es sich als schwierig heraus, informelles Wissen aufzunehmen und durch formale Begriffe darzustellen. Es wird hervorgehoben, dass das Problem zwar theoretisch lösbar ist, jedoch nicht in die Praxis umgesetzt werden kann. Weil bereits Systeme mit einer geringen Menge an Fakten die Rechenleistung von Computern ausreizten. Durch Richtlinien, die dem System gegeben werden, kann dem Problem entgegengesteuert werden. Beispielsweise, kann dem System vorgegeben werden welche Schlussfolgerungen zuerst ausprobiert werden sollen.

Systeme die rational Handeln (Rationale Agenten): Zusammengefasst ist ein Agent ein Computerprogramm, welches so handelt, das er das beste Ergebnis erreicht oder das beste erwartete Ergebnis, falls es Unsicherheiten gibt.

> A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

So wird erwartet, dass dieser selbstständig agiert, seine Umgebung wahrnehmen kann, über einen längeren Zeitraum besteht, sich an Änderungen und Möglichkeiten anpasst und die Ziele eines Anderen weiterführen kann. Das logische Denken ist wie in dem Ansatz (siehe 4.1) eine Ergänzung des Denkprozesses eines rational handelnden Agenten. Es gibt Situationen, in denen nicht beweisbar das Richtige getan werden kann, jedoch dennoch etwas getan werden muss. So ist es auch möglich, rational zu agieren, ohne eine Schlussfolgerung zu treffen: Diese werden als Reflexreaktionen bezeichnet. Der Agent muss die Möglichkeit haben, Wissen intern darzustellen und basierend auf diesem Wissen logische Schlüsse zu ziehen. Der Denkprozess eines Agenten kann durch die Prozessabfolge Wahrnehmen, Planen, Handeln beschrieben werden (siehe 4.1).

Umgangssprachlich wird unter dem Begriff künstliche Intelligenz, Systeme die rational Handeln bzw. rationale Agenten verstanden. Es ist zu beachten das sich die vier Kategorien nach Russell u. a. (2003) nicht gegenseitig ausschließen. Betrachtet man die Arbeiten aus der Companion Technologie, werden Software Agenten oder Roboter entwickelt, die sowohl die Fähigkeiten eines rationalen Agenten besitzen, als auch Aspekte von Systemen die wie Menschen handeln. So werden alle der genannten sechs Eigenschaften in Computerspielen und Compainion Systemen heutzutage eingesetzt. Ziel ist es jedoch selten gewesen den Turing-Test zu bestehen, sondern Systeme zu schaffen die, die Interaktionen zwischen Menschen und Maschinen verbessern.

4.2. Agenten

Allgemein ist ein Agent eine Software oder ein System, welches seine Umgebung durch Sensoren wahrnimmt und mit Hilfe von Aktoren in dieser Umgebung handelt (vgl. Russell u. a. (2003)). Der grundlegende Modell eines Agenten ist in der Abbildung 4.1 dargestellt. Im ersten Schritt werden über die Sensoren die Umwelt wahrgenommen. Anschließend wird aufgrund dieser Wahrnehmungen eine Aktion ausgewählt. Die Aktion wird anschließend über Aktuatoren ausgeführt. Dieses Modell wird oft auch unter dem Begriff Wahrnehmen/Planen/Handeln (Sense/Plan/Act) geführt.

Anhand dieser Definition hat ein menschlicher Agent Augen, Ohren, Nase als Sensoren und Hände, Füße, Mund und andere Körperteile als Aktuatoren. Ein Roboter Agent besitzt beispielsweise eine Kamera und Laser/Infrarot Entfernungsmesser als Sensoren und verschiedene Motoren wie beispielsweise Greifarm oder Räder als Aktuatoren. Ein Softwareagent kann seine Umgebung über Tastatureingaben, Kamera oder Spielcontroller wahrnehmen und auf seine Umgebung über Sound und dem Darstellen von Bildern/Daten wirken.

Es wird die Annahme gemacht, dass ein Agent seine Aktionen wahrnehmen kann, aber nicht immer dessen Auswirkung. Der Begriff Wahrnehmung wird verwendet, um die wahrgenommenen Eingaben des Agenten zu jedem beliebigen Zeitpunkt zu beschreiben. Die Wahrnehmungssequenz ist die komplette Historie aller vom Agenten erkannten Wahrnehmungen. So kann die Entscheidung eine Aktion zu wählen, von der kompletten bis zu diesem Zeitpunkt erkannten Wahrnehmungssequenz abhängen.

Das Verhalten eines Agenten wird durch die Agentenfunktion abgebildet. Diese Funktion ordnet eine jede Wahrnehmungssequenz einer Handlung zu. Die Agentenfunktion ist eine abstrakte mathematische Beschreibung des Agenten. Das Agentenprogramm ist die eigentliche Implementation, welche auf der Agentenarchitektur ausgeführt wird.

Diese Definition dient jedoch nicht der absoluten Charakterisierung von Systemen in Agenten und Nichtagenten. Vielmehr dient diese als Analysewerkzeug für Systeme.

4.2.1. Leistungsbewertung und Rationalität

Ein Agent wird als rational bezeichnet, wenn er das Richtige tut. Die Frage ist, wer entscheidet, was das richtige Verhalten ist und wann eine gewählte Aktion als richtig gilt.

Grob gesagt, ein Agent tut immer genau dann das Richtige, wenn die Aktion des Agenten dazu führt, so erfolgreich wie möglich zu sein. Eine allgemeingültige Leistungsbewertung über den Erfolg des Agenten gibt es nicht. Ein Agent erzeugt abhängig von seinen Wahrnehmungen eine Aktionssequenz. Aufgrund dieser Aktionen durchläuft die Umwelt des Agenten eine Folge

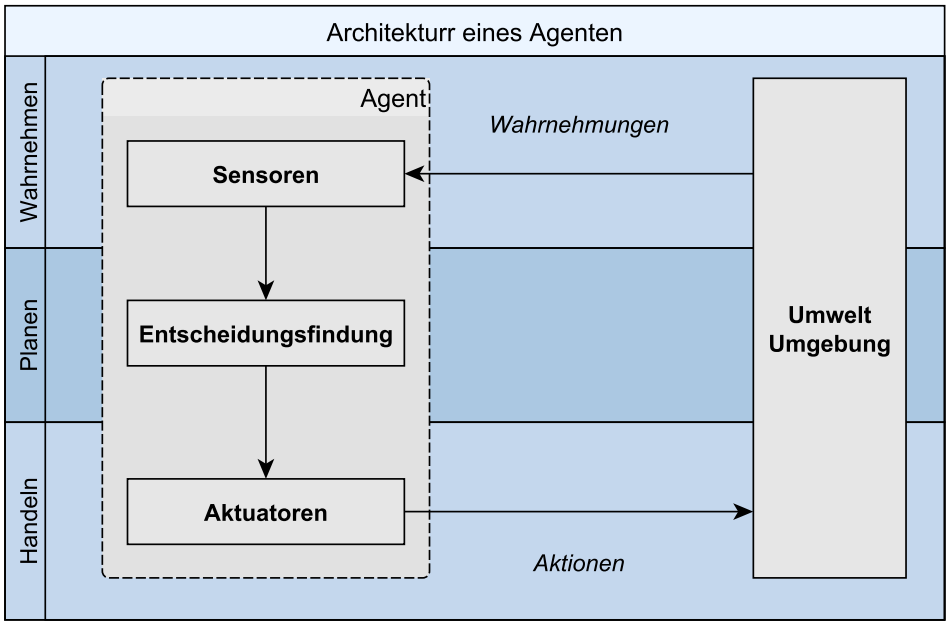

Abbildung 4.1.: Grundlegende Architektur eines Agenten

von Zuständen. Entspricht die Folge dem angestrebten Verhalten, hat sich der Agent richtig verhalten. Die Leistungsbewertung kann nur durch den Entwickler vorgenommen werden. Es bedarf Fachwissen über die gestellte Aufgabe, um zu ermitteln, wie gut diese erfüllt wurde und ob Verbesserungen oder Änderungen am Verhalten nötig sind.

Die Rationalität eines Agenten ist von vier Bedingungen abhängig.

- die Leistungsbewertung
- das Vorwissen
- die möglichen Aktionen
- der bisherigen Wahrnehmungsfolge

Nach Russell u. a. (2003) ergibt sich dadurch die folgende Definition für einen Agenten:

> For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Für jede mögliche Wahrnehmungsfolge sollte ein rationaler Agent eine Aktion auswählen, von der erwartet wird, dass sie die Leistungsbewertung des Agenten maximiert, basierend auf dem vorhandenen Wissens und der Wahrnehmungsfolge über die der Agent verfügt.

4.2.2. Vier grundlegende Arten von Agenten

Einfache Reflex-Agenten: Der einfache Reflex-Agent wählt seine Aktionen anhand der aktuellen Wahrnehmungen (siehe 4.1). Die vorhergegangene Wahrnehmungsfolge wird ignoriert. Die Entscheidungsfindung eines solchen Agenten kann durch einfache „if-then“ Regeln umgesetzt werden. Der Agent bekommt durch die Sensoren seine Wahrnehmungen, diese werden interpretiert und der Zustand generiert. Anschließend wird eine für den Zustand passende Regel gefunden. Anhand dieser Regel wird eine passende Aktion ausgewählt, welche anschließend die Aktuatoren steuert.

```
1 public Action SimpleReflexAgent(percept)
2 {
3   state = InterpInput(percept)
4   rule = RuleMatch(state,rules)
5   action = RuleAction[rule]
6   return action
7 }
```

Listing 4.1: Einfacher Reflex-Agent

Nachteil dieses Agenten ist, dass er nur reagieren kann, wenn die Umgebung in der sich der Agent befindet, vollständig beobachtbar ist. Sprich, sobald eine der benötigten Wahrnehmungen fehlt, kann der Agent nicht mehr funktionieren. Entweder reagiert er nicht mehr oder landet in einer Endlosschleife, je nach Implementierung und abhängig davon, ob ein Notfallplan für solche Fälle vorgesehen ist. Die fehlenden Wahrnehmungen können durch defekte Sensoren hervorgerufen werden oder aus dem einfachen Grund, das die benötigte Wahrnehmung in einem vorherigen Zeitschritt passiert ist.

Modellbasierte Reflex-Agenten: Modellbasierte Refelx-Agenten unterscheiden sich hauptsächlich von Einfachen reflexbasierten Agenten darin, dass sie einen internen Zustand verwalten. Dieser erlaubt es ihnen Wahrnehmungen bzw. Wahrnehmungsverläufe zu verwalten, die nicht Teil des aktuellen Zustands sind. Um den internen Zustand über die Zeit zu pflegen, sind zwei Arten von Wissen über die Umwelt erforderlich. Erstens, wie entwickelt sich die Welt unabhängig von dem Agenten. Beispielsweise beobachtet der Agent eine Ampel. Wenn diese auf Rot steht und die Sicht versperrt ist, muss der

Agent wissen, dass diese über die Zeit die Ampelphase von Rot nach Grün wechselt. Zweitens, wie wirken sich die Aktionen des Agenten auf die Umwelt aus. Bewegt sich beispielsweise der Agent 5 Meter nach vorne, muss der Agent wissen, dass er seine Position verändert hat und sich nun an einem Punkt Position += vorwärts Vektor * 5 Meter befindet. Dieses Wissen wird als Modell der Umwelt bezeichnet.

```
1 private State state;
2 public Action StatefulReflexAgent(percept)
3 {
4   state = UpdateState(state, action, percept)
5   rule = RuleMatch(state, rules)
6   action = RuleAction[rule]
7   return action
8 }
```

Listing 4.2: Modellbasierter Reflex-Agent

Zielbasierte Agenten: Es gibt Situationen, in denen eine Reflexhandlung nicht ausreichend ist. Beispielsweise, ein selbstfahrendes Auto steht an einer Kreuzung. Das Bremsen an der roten Ampel kann ein Reflex-Agenten richtig entscheiden. Jedoch nicht, wenn es darum geht, in welcher Richtung das Fahrzeug abbiegen soll, dies ist abhängig von dem Fahrziel. Folglich benötigt der Agent neben den Zustandsinformationen ein oder mehrere Ziele, die wünschenswerte Situationen für den Agenten beschreiben. Die Ziele können dabei von verschiedener Komplexität sein. So kann ein Ziel direkt durch eine Handlung erreicht werden oder eine komplette Aktionssequenz benötigen. Das Finden solcher Aktionsfolgen wird algorithmisch durch Suchen und Planen erreicht. Beispielsweise das Suchen eines Weges zwischen Start und Fahrziel. Oft wird hierfür ein heuristischer Ansatz wie z. B. AStar verwendet. Der Pseudocode 4.3 zeigt einen möglichen zielbasierten Agenten. Dieser prüft, welche der im aktuellen Zustand möglichen Aktionen mit dem Ziel vereinbar sind und wählt eine entsprechende Aktion/Aktionsfolge aus. Die Aktionsfolge muss kontinuierlich überprüft werden, denn im Verlauf der Durchführung kann ein Ziel oder eine Aktion innerhalb der Aktionsfolge nicht mehr erreichbar sein. Bleiben wir bei dem Beispiel der Routenplanung. So ist es möglich, dass eine Straße gesperrt ist und die Route deshalb neu geplant werden muss.

```
1 private State state;
2 private Goals goal;
3 public Action GoalBasedAgent(percept)
4 {
```

```
5   state = UpdateState(state, actions, percept)
6   plan = planActions(state, goal)
7   return plan
8 }
```

Listing 4.3: Zielbasierter Agent

Nutzenbasierte Agenten: Diese Art von Agenten verwenden Nutzenfunktionen (Utility Functions), die den Nutzen von Aktionen bewerten. Der Nutzen einer Aktion wird oft als reale Zahl zwischen 0 und 1 repräsentiert. Diese Zahl stellt den Nutzen einer Aktion oder Aktionsfolge dar. Berechnet ein selbstfahrendes Auto eine Route zwischen dem Startpunkt und einem Ziel, besteht die Möglichkeit mehrere Routen zu erhalten, die zum Ziel führen. Über eine Nutzenfunktion kann nun bestimmt werden, welche dieser Routen ausgewählt wird. Die Nutzenfunktion könnte die Routen beispielsweise in Abhängigkeit von der Streckenlänge bewerten und die schnellste Route auswählen. Handelt es sich um ein Taxi, könnte aber durchaus auch eine langsamere Strecke ausgewählt werden, da diese den Gewinn maximieren würde. Generell sind Nutzenfunktion immer dann sinnvoll, wenn mehrere Aktionen oder Aktionsfolgen zur Auswahl stehen, die in Konflikt zueinander stehen. Der Pseudocode 4.4 zeigt beispielhaft, wie ein solcher Agent aussehen könnte.

```
1 private State state;
2 private Goals goal;
3 public Action GoalBasedAgent(percept)
4 {
5   state = UpdateState(state, actions, percept)
6   plans = planActions(state, goal)
7   utilities = calculateUtility(state, plans)
8   plan = pickPlanWithHighestUtility(plans, utilities)
9   return plan
10 }
```

Listing 4.4: Nutzenbasierter Agent

Lernende Agenten: Der Bereich der lernenden Agenten ist sehr umfangreich und nicht essenziell für diese Arbeit. Für eine ausführliche Beschreibung wird auf die Quelle Russell u. a. (2003) verwiesen. Kurz gesagt, durch maschinelles Lernen wird der Agent befähigt seine Leistung unabhängig von seinem Ausgangswissen zu maximieren, er kann lernen in unbekannten Umgebungen zu agieren. Prinzipiell kann jeder der bisher genannten Agenten um die Funktionalität des Lernens erweitert werden.

4.2.3. Zusammenfassung

Zusammengefasst ist ein Agent ein System oder eine Software, welches seine Umgebung durch Sensoren wahrnimmt und mit Hilfe von Aktoren in dieser Umgebung handelt. Über eine Agentenfunktion wird das Verhalten des Agenten beschrieben. Über eine Leistungsfunktion wird das Handeln des Agenten bewertet. Das Agentenprogramm ist die Implementation der Agentenfunktion. Es existieren vier grundlegende Arten von Agenten, und jede dieser Arten kann um maschinelles Lernen erweitert werden. Die grundlegenden Arten schließen sich jedoch nicht gegenseitig aus. Ein einfacher Reflex-Agent kann eine Nutzenfunktion anwenden, um alle möglichen Aktionen zu bewerten oder maschinelles Lernen Nutzen.

4.3. Entscheidungsfindung

Die Entscheidungsfindung ist einer der Hauptbereiche künstlicher Intelligenz und notwendig damit ein Agent handeln kann. Es ist eines der Haupteigenschaften von unabhängig agierenden Charakteren in Computerspielen. Im Allgemeinen besitzt ein Charakter in einem Videospiel eine begrenzte Anzahl von Aktionen, die dieser ausführen kann. Diese werden durch das Game Design bestimmt und variieren je nach Spiel und Charakter. Ohne Entscheidungsfindung kann jedoch kein vernünftiges Verhalten für einen Charakter modelliert werden.

Es existieren viele verschiedene Ansätze, die das Problem der Entscheidungsfindung lösen. Wir kategorisieren diese in planende und reaktive Ansätze. Möglich ist auch eine Kombination aus beiden Ansätzen. Welcher dieser Ansätze sich für ein Computerspiel eignet, ist abhängig von der Problemstellung. Es muss gewährleistet sein, das die KI auf Ereignisse innerhalb der Spielwelt und auf Eingaben des Benutzers adäquat reagieren kann. Der Algorithmus muss für zeitkritische Anwendungen geeignet sein und gut Skalieren. Abhängig vom Spiel ist es möglich, das die Anzahl gleichzeitig aktiver KI-Agenten in die Hunderte geht. Da es sich um eine Echtzeitanwendung handelt, gibt es ein Zeitbudget, das eingehalten werden muss. Bei heutigen Computerspielen beträgt die akzeptable Mindestbildrate 30 Bilder pro Sekunde. Folglich darf die Rechenzeit nicht mehr als 30 ms pro Bild betragen. Dieses Zeitbudget beinhaltet Grafik, Audio, Input, Spiellogik und KI.

Eine große Herausforderung ist es, die Charaktere glaubwürdig wirken zu lassen. Die Reaktionen eines Charakters und Entscheidungen müssen nachvollziehbar sein und realistisch wirken. Im optimalen Fall sollten sich die Entscheidungen eines virtuellen Charakters nicht von denen eines Menschen unterscheiden lassen. Ein langes Zögern, bevor eine Aktion ausgeführt wird, Merkwürdiges, unlogisches oder Fehlverhalten wirken sich negativ auf die Immersion des Spielers aus. Die virtuellen Charaktere wirken uninteressant oder lassen den Spieler an

der Intelligenz seiner Kontrahenten zweifeln. Um den Spieler überzeugende Charaktere und Immersion bieten zu können, müssen virtuelle Charaktere in der Lage sein korrekt auf Situationen zu reagieren. Sie müssen eigene Ziele verfolgen und danach handeln. Wir beschränken uns bei der Entscheidungsfindung auf reaktive Verfahren, welche im Bereich der Computerspiele gebräuchlich sind.

4.3.1. Finite State Machines - FSM

Eine Finite State Machine (endlicher Automat) ist ein Modell, mit dem das Verhalten von Agenten beschrieben werden kann. Die Finite State Machine besteht aus Zuständen und Transitionen. Die Zustände beschreiben das Verhalten; die Transitionen beschreiben die Bedingungen für einen Zustandswechsel. Auch wenn das Konzept der Finite State Machine schon lange existiert werden diese noch häufig in der modernen Softwareentwicklung eingesetzt. Sie bieten ein einfach zu verstehendes Konzept, das sich schnell implementieren und testen lässt. Bei der Entwicklung von rudimentärer KI sind FSM oft die bevorzugte Lösung.

Einfache Game AI - FSM

Das Diagramm 4.2 zeigt, wie man mit einer Finite State Machine Verhalten modelliert werden kann. Das gezeigte Beispiel besteht aus fünf Zuständen, die das Verhalten eines Agenten für ein Spiel Action Spiel beschrieben könnten.

Die Transitionen bestimmen, ob und wann sich der aktuelle Zustand ändern kann. Jeder Zustand besitzt eine ausgehende und eine eingehende Transition. Generell kann ein Zustand beliebig viele eingehende und ausgehende Transition besitzen. Das Beispiel beginnt mit dem initial Zustand „Patrol At Location A“. In diesem Zustand läuft die KI eine vorgegebene Anzahl von Positionen ab. Nachdem alle Positionen besucht wurden, wird die Transition „Done A“ ausgelöst. Der Zustand „Patrol At Location B“ unterscheidet sich von „Partol A“ durch die Positionen, die durchlaufen werden. Sind in diesem Zustand alle Positionen abgelaufen wurden, so wird wieder in den Zustand „Patrol At Location A“ gewechselt. Die beiden Zustände modellieren einen zyklischer Wechsle zwischen zwei zu patrouillierenden Pfaden. Aus beiden Zuständen kann die KI entscheiden, ob sie den Spieler angreift. Die Transitionen „Player In Range“ und „!Player In Range“ sorgen für den Zustandswechsel in den Zustand „Attack“. Die Transition wird ausgelöst, sobald sich der Spieler innerhalb einer vordefinierten Entfernung zur KI befindet. Die Negation diese Bedingung sorgt für den Zustandswechsel zurück zu den Zuständen „Patrol At Location A“ oder „Patrol at Location B“ je nach vorherigem Zustand. Befindet sich die KI im Zustand „Attack“ und die Lebenspunkte sinken auf 0 wird die KI in den Zustand „Dead“ übergehen. Dieser Zustand beendet die KI-Routine und löscht den Agenten.

Nachteile der FSM

Die in diesem Beispiel gezeigte Finite State Machine lässt erkennen, dass bei komplexerem Verhalten die Anzahl der Zustände und Transitionen stark ansteigen. Je nach Abstraktionsgrad der Zustände kann bereits ein einfaches Verhalten in komplexen Automaten resultieren. Was

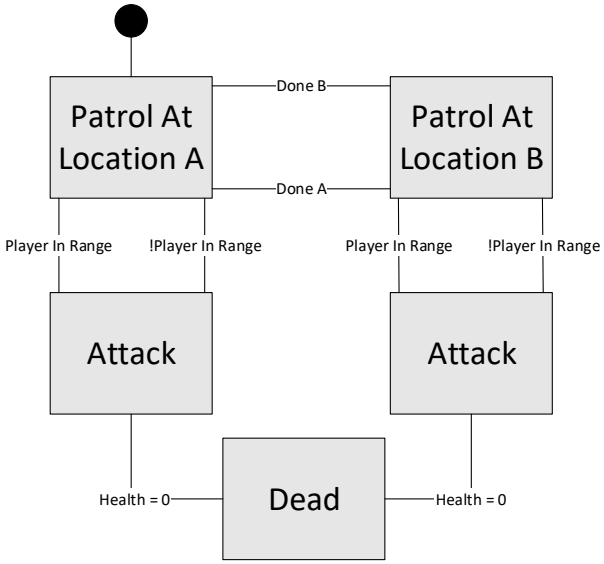

Abbildung 4.2.: Finite State Machine Beispiel

dazu führt, dass das Verhalten nur noch schwer zu Verwalten und zu Pflegen ist. Bereits bei Zustandsautomaten mit einer geringen Menge an Zuständen wird das Hinzufügen neuer Zustände schwer zu realisieren ohne den Automaten umzustrukturieren. Das Erweitern einer Finite State Machine wird fehleranfällig. Zudem lässt sich häufig wiederkehrendes Verhalten schlecht wiederverwenden. Anhand des Beispiel 4.2 sehen wir das ein Zustand mit identischen Verhalten und Transitionen öfter vorkommen kann (Abb. 4.2 Attack). Grund hierfür ist, das es nicht anders möglich ist, in den vorhergegangen Zustand „Patrol At Position A“ oder „Patrol At Position B“ zurückzukehren.

Hierarchical State Machine - HFSM

Hierarchical State Machine nehmen sich der in 4.3.1 dargestellten strukturellen Problemen der Finite State Machines an. So ist es möglich, eigenständiges oder wiederkehrendes Verhalten als Subautomaten zu definieren, wodurch die Anzahl der Transitionen durch Hierarchisierung minimiert werden. Zudem können Automaten einfacher umstrukturiert werden.

In der Abbildung 4.3 ist das Verhalten aus 4.3.1 als Hierarchical State Machine modelliert. Durch das Modellieren des Subautomaten „Patrol FSM“ kann die Anzahl der Zustände, bereits in diesem kleinen Beispiel, um eins und die Anzahl der Transitionen um zwei reduziert werden. Möglich ist dies durch den „History State“ H, aus dem beim Eintreten in den Subautomaten der

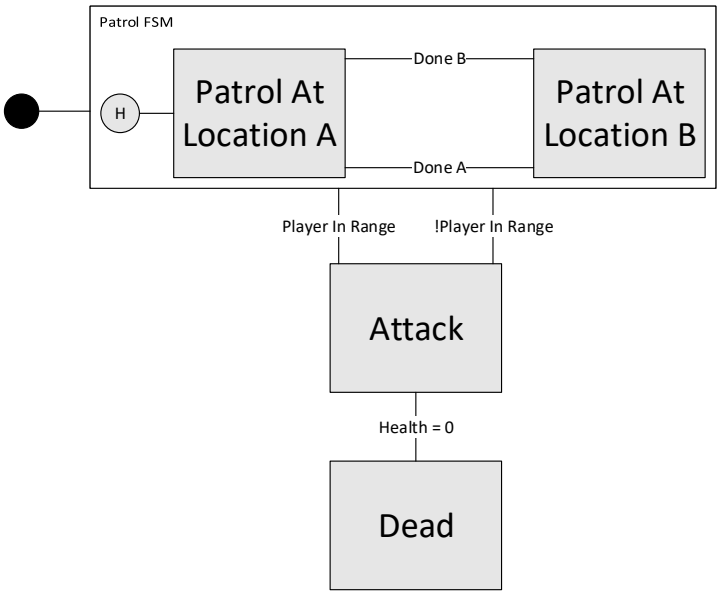

Abbildung 4.3.: Hierarchical State Machine Bsp.

Startzustand entnommen wird. Initial ist der Startzustand „Patrol At Location A“, wird nun die Transition „Player In Range“ aus dem Zustand „Patrol At Location B“ heraus geschaltet, so wird dieser im „History State“ gespeichert. Bei einer Transition zurück in die „Patrol FSM“ durch die Transition „!Player In Range“ wird nun der Zustand „Patrol At Location B“ Startzustand.

4.3.2. Behavior Trees - BT

Behavior Trees oder kurz (BTs) zählen zum jetzigen Zeitpunkt zu einer der weitest verbreiteten Techniken zur Implementierung von Computerspiele KI (künstliche Intelligenz). AAA Spiele [1] wie z.B. Halo2, Halo3, The Devision oder GTA verwenden Behavior Trees zur Umsetzung von Agenten (vgl. Johansson und Dell'Acqua (2012)). Es handelt sich hierbei um eine praxiserprobte Technik, die einen intuitiven Aufbau hat.

Aufgrund ihrer Struktur bieten Behavior Trees viele Möglichkeiten zur Anpassung und Erweiterung an gegebene Anforderungen. Durch die Baumstruktur ist eine verständliche Visualisierung zum Erstellen und modellieren von Verhalten möglich. Somit bieten Behavior Trees auch Personen ohne umfangreiche Programmierkenntnis, die Möglichkeit diese zu verwenden und Verhalten von Agenten zu modellieren, Voraussetzung sind hierfür gute Editoren und vordefinierte Bausteine zum Modellieren von Verhalten, die den Benutzer dabei unterstützen.

Trotz der umfangreichen Verwendung von Behavior Trees innerhalb der Industrie und in der Forschung, gibt es bis zum jetzigen Zeitpunkt keine einheitliche Definition. Zur Beschreibung des Aufbaus von Behavior Trees werden die Definitionen aus Ogren (2012) Marzinotto u. a. (2014) und Johansson und Dell'Acqua (2012) herangezogen. Diese Definitionen unterscheiden sich minimal in der Nomenklatur und Funktionsweise.

Grundverhalten

Es handelt es sich bei einem Behavior Tree um eine Baumstruktur. Während der Ausführung werden alle Knoten innerhalb des Baumes evaluiert - beginnend mit dem obersten Knoten (Wurzel) in der Hierarchie. Anschließend führt dieser, falls vorhanden, seine Kinderknoten aus. Ein Knoten kann sich dabei in einem von drei Zuständen befinden, bestehend aus:

Running: Der Status „Running" beschreibt einen Knoten, der nach dem Zeitpunkt der Ausführung seine Operationen noch nicht abgeschlossen hat.

Success: „Success" beschreibt ist der Status für einen Knoten, der seine Operation erfolgreich ausführen konnte.

Failure: Der Status „Failure" beschreibt einen Knoten, der seine Operation nicht ausführen konnte.

[1] Spiele mit einem großen Budget

Knoten

Ein Behavior Tree besteht aus zwei verschiedenen Basisknoten, den Selectors und den Actions. Diese wiederum, werden erweitert um spezifischere Funktionen abzubilden. Nachfolgend eine Beschreibung der am häufigsten verwendeten Arten von Knoten:

Action: Actions sind immer Blattknoten und führen eine Aktion aus. Ob eine Action erfolgreich ausgeführt wurde, wird durch den zurückgegebenen Status bestimmt. Wann eine Aktion als erfolgt gilt, hängt von der jeweiligen Aktion ab. Es ist allerdings erforderlich, dass nach jeder Ausführung der Status zurückgegeben wird.

Condition: Es handelt sich um einen Blattknoten der eine bestimmte Bedingung prüft. Was geprüft wird, hängt von der Implementation ab. Das Verhalten ist äquivalent zu einer If-Anweisung. Wird eine Bedingung erfüllt, so befindet sich die Condition in dem Zustand „Success“. Wenn die Bedingung nicht erfüllt wurde, wird der Status „Failure“ zurückgegeben. Die Evaluierung einer Bedingung erfolgt sofort, eine Condition kann sich nicht im Zustand „Running“ befinden.

Selectors: Selectors sind keine Blattknoten, die Funktion besteht darin, eines oder mehrerer Kinderknoten anhand eines festgelegten Verhaltens zu selektieren. Die Kinder des Selector werden sequenziell aktualisiert, bis einer der Knoten einen Status „Running“ oder „Success“ zurückgibt. Der Status des Kindes wird anschließend im Baum nach oben weitergereicht. Der Selector gibt den Status „Failure“ zurück, wenn keiner seiner Kinder sich im Status „Running“ oder „Success“ befindet.

- **Sequence Selector:** Ein Sequence Selector arbeitet alle seine Kinderknoten in einer festen Reihenfolge ab. Beginnend vom ersten bis zum letzten Kind. Der Sequence Selector gibt den Status „Success“ zurück, solange alle seine Kinder diesen Status besitzen. Sobald sich ein Kind in dem Status „Failure“ befindet, gibt der Sequenz Selector diesen Status an seinen Vaterknoten zurück. Die Ausführung aller nachfolgender Kinder wird gestoppt. Die Sequenz beginnt bei der nächsten Ausführung von vorne. Gibt eines der Kinder in der Sequenz den Status Running zurück, so wird bei der nächsten Ausführung die Sequenz bei diesem Kind fortgeführt.
- **Priority Selector:** Ähnlich wie der Sequenz Selector führt der Priority Selector alle seine Kinder in einer festen Reihenfolge aus, beginnend mit dem ersten Knoten. Der Priority Selector übernimmt dabei den Zustand des Kindes, solange sich dieser im Zustand Running oder Success befindet. Wenn ein Kind Failure zurückgibt, so

wird der nächste Knoten in der Reihenfolge ausgewählt. Der Selector befindet sich im Zustand Failure sobald, sich alle seine Kinder den Zustand Failure befinden.

Parallel Selector: Ein Parallel Selector führt alle seine Kinder gleichzeitig aus. Hierbei ist zu beachten, dass sich nicht um Multithreading handelt, sondern auf den Ausführungszyklus des Baumes Bezug genommen wird. Generell gibt der Parallel Selector den Status eines oder mehrerer Kinder wieder. Dies steht jedoch in Abhängigkeit zu der Konfiguration des Selectors. Mögliche Konfigurationen für „Running“, „Failure“ und „Success“ können in Abhängigkeit von Variablen angegeben werden. Diese geben an, wie viele Kinder den Zustand z.B. „Success“ erreichen müssen, damit der Parallel Selector ebenfalls ein diesen Status meldet.

Decorator: Der Decorator ist eine Kombination aus Condition und Selector, wobei ein Decorator nicht mehr als ein Kind besitzt. Die Bedingung gibt dabei vor, ob das Kind ausgeführt wird. Wird das Kind ausgeführt, so gibt der Decorator den Zustand des Kindes zurück oder „Failure“, falls die Bedingung zur Ausführung nicht erfüllt wurde.

Grafische Notation

Da es für die grafische Notation keine einheitliche Definition gibt, wird die folgende Notation für diese Arbeit verwendet (siehe Grafik 4.4).

Action: Rechteck mit Beschriftung.

Condition: Raute mit Beschriftung.

Selector: Rechteck mit Beschriftung gefolgt von den Symbolen „|?“.

Sequence Selector: Rechteck mit Beschriftung gefolgt von dem „|“ Symbol.

Priority Selector: Rechteck mit Beschriftung gefolgt von den Symbolen „|*“.

Parallel Selector: Rechteck mit Beschriftung gefolgt von den Symbolen „||“.

Decorator: Rechteck mit Beschriftung und eingeschlossen Knoten (Action oder Selector).

Beispiel für einen Behavior Tree

Im Folgenden wird das Verhalten eines Behavior Trees am Beispiel der in Abschnitt 4.3.1 beschriebenen KI erläutert. Der Aufbau des Behavior Trees wird in Abbildung 4.5 mit der

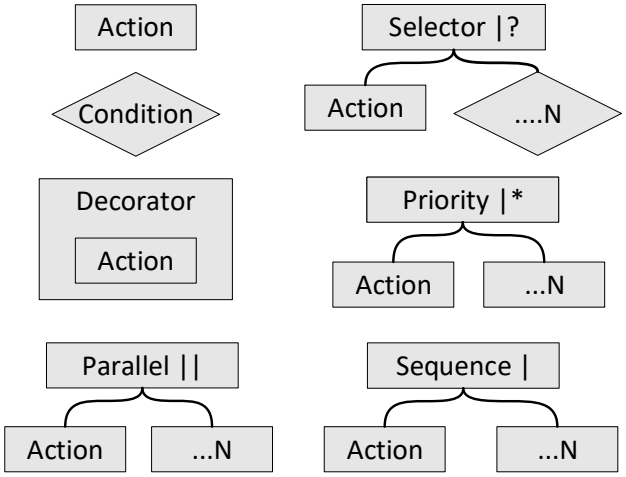

Abbildung 4.4.: Notationen für die grafische Darstellung eines Behavior Trees

in Abschnitt 4.3.2 beschriebenen Notation dargestellt. Das Verhalten des Agenten ist in drei Teilaufgaben unterteilt und wird wie folgt interpretiert:

Als Wurzelknoten wird ein Priority Selector verwendet, d.h. bei jedem Update des Baumes werden die Teilbäume in der Priorität von links nach rechts abgearbeitet. Nachdem ein Zyklus erfolgreich abgearbeitet wurde, wird der Baum bei der nächsten Ausführung neu evaluiert. Die erste Priorität hat der Knoten „Dead“, „Dead“ besteht aus einem Decorator und einer Action. Der Decorator prüft, ob die Bedingung „Health = 0“ erfüllt ist. Ist die Bedingung erfüllt, wird das Verhalten beendet und die KI terminiert. Zweite Priorität hat das Verhalten „Attack“ - dies ist ebenfalls ein Decorator. Bedingung für die Ausführung von „Attack“ ist „Player In Range“. Ist die Bedingung erfüllt, wird der Spieler angegriffen. Als letzte Priorität wird der Teilbaum ausgeführt, der für das Patrouillieren verantwortlich ist. Der Teilbaum besteht aus dem Sequenz Selector „Patrol“ und den Aktionen „Patorl A“ und „Patrol B“. Solange keine der beiden anderen Prioritäten erfüllt wird, werden diese zyklisch von links nach rechts ausgeführt. Die hier beschriebene Implementation unterscheidet sich in ihrem Verhalten minimal von dem in Abschnitt 4.3.1 beschriebenen Verhalten. Anders als bei den ersten Beispielen ist es möglich, von „Patrol“ in den Zustand „Dead“ zu gelangen.

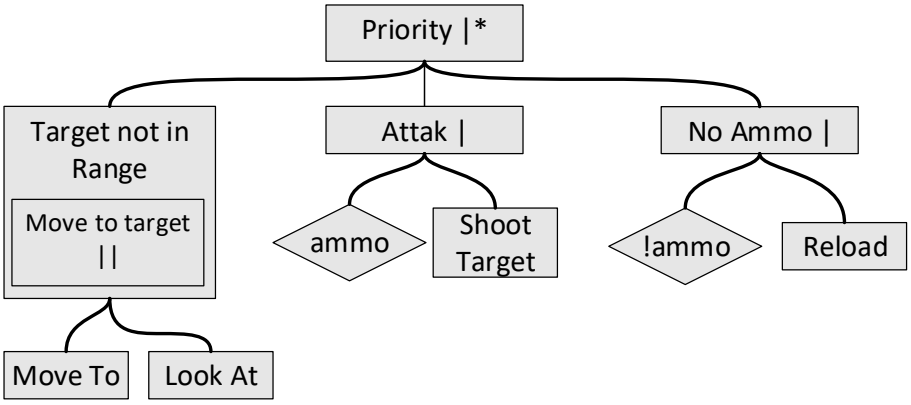

Abbildung 4.5.: Behavior Tree: Beispiel AI

4.3.3. Utility basierte Systeme

Anhand der vorher beschriebenen Architekturen kann man erkennen, dass die Entscheidungen, die getroffen werden, fast ausschließlich auf booleschen Ausdrücken basieren. Die Transitionen der FSM oder HFSM wählen die Aktionen (Zustände) je nachdem ob eine Transition erfüllt ist oder nicht. Behavior Trees verhalten sich ähnlich: hier wird anhand von vier vordefinierten Zuständen entschieden, wie der Baum durchlaufen wird. Es handelt sich um eine diskrete Menge an Bedingungen die zur Auswahl einer Aktion führen. Jedoch gibt es zahlreiche Fragestellungen, wo eine boolesche Bedingung nicht ausreicht oder geeignet wäre. So könnte eine Entscheidung abhängig von vorherberechneten Erfolgsaussichten sein. Beispielsweise kann die Wahrscheinlichkeit bei einer Konfrontation als Sieger hervorzugehen, abhängig von dem emotionalen Zustand, der Persönlichkeit oder jegliche andere Kombination aus kontinuierlichen Werten sein.

Das generelle Konzept von Utility basierten Systemen besteht daraus, jede mögliche Aktion durch einen nummerischen Wert zu bewerten und anhand des Ergebnis die beste Aktion auszuwählen. In einem Utility System werden kontinuierlich Werte gemessen, gewichtet, kombiniert und bewertet. Aus vielen Aktionen wird eine Menge von potenziell präferierten Aktionen gewählt. Anschließend wird die für die aktuelle Situation günstigste Aktion gewählt.

Einerseits kann man Utility Systeme benutzen, um vorhandene Transitionskosten zu ergänzen. Anderseits ist es durchaus möglich, komplette Systeme basierend auf der Berechnung von Utility Werten zu entwickeln. Dies betrifft vor allem Umgebungen mit einer Vielzahl möglicher Aktionen, die sehr aufwendig zu modellieren sind. Als auch Systeme wo Entscheidungen von vielen einzelnen miteinander konkurrierenden Faktoren abhängig sind. In diesen Umgebungen kann ein Utility basiertes System zur Entscheidungsfindung sinnvoll eingesetzt werden.

Ein prominentes Beispiel für ein solches Utility System ist die Sims. In die Sims wählt der einzelne Agent anhand seiner Situation und internem Status eine Menge von präferierten möglichen Aktionen. Für jede Aktion wird die Utility berechnet, dieser Wert ist normalisiert. Im einfachsten Fall wird die Aktion mit dem höchsten Utility Wert gewählt, oder aus den ersten 3 bis 4 Aktionen mit dem höchsten Utility Wert zufällig einer gewählt. Sollte es Aktionen mit gleichem Utility geben, wird auch hier über eine Zufallsnummer oder andere Selektionsmechanismen wie z. B. Gewichtung die nächste Aktion gewählt. So sollte, wenn die Eignung einer Aktion steigt, auch dessen Utility Wert steigen.

Ob und wann ein Utility System den anderen Architekturen zu bevorzugen ist, hängt im Endeffekt von dem Spiel ab. In der Regel werden diese Systeme häufig bei Echtzeit Strategie Spielen oder Rollenspiel verwendet. Zum Beispiel für Spiele, bei denen entschieden werden muss, welche Aktion als nächstes ausgewählt werden soll um die Finanzen in einer Wirtschafts-

simulation am effektivsten verwalten. Rollenspiele, bei denen die KI berechnen muss, welcher Zauberspruch am effektivsten ist, um einen eine andere KI oder einen Spieler zu besiegen. Würden von einem Utility Basiertem oder einem hybriden System Profitieren.

Utility basierte Architekturen können oft flexibler auf veränderte Situationen reagieren, angepasst oder erweitert werden. Die Berechenbarkeit des Verhalten von Utility basieren Agenten ist ebenfalls geringer als bei anderen Architekturen.

Im Gegensatz zu anderen Systemen können Utility Systeme unübersichtlich und leicht unberechenbar werden. Sie sind nicht intuitiv zu verstehen und für Designer oder andere Personen, ohne technisches Wissen, schwer zu modifizieren. Es ist schwer, die Wirkung von Veränderungen des Systems einzuschätzen und zu überblicken. In manchen Situationen ist ein vordefiniertes Verhalten der KI nötig, jedoch ist dies mit Utility Systemen nur schwer umzusetzen. Ein weiterer Nachteil ist, das Utility Systeme oft mehr Performance in Anspruch nehmen. Dies kann bei Spielen, die nur einen geringen Anteil ihrer Rechenleistung für die KI zur Verfügungsstellen können, ein Problem.

Utility

Utility bezeichnet einen einzeln einheitlichen Wert, der die Nützlichkeit einer Aktion innerhalb eines Kontextes beschreibt. Es ist zu beachten, dass Utility nicht das gleiche ist wie ein nummerischer Wert. Ein Wert beschreibt eine messbare Menge wie z. B. der Preis für ein Auto, ein Fahrrad, eine Monatskarte oder die Zeit um von zu Hause zur Arbeit zu kommen. Die Utility wiederum misst, wie sehr etwas verlangt wird oder wie nützlich eine Aktion ist.

Beispielsweise, wenn man sich für ein Transportmittel für den Weg zur Arbeit entscheiden muss und die Auswahl zwischen einem Fahrrad und einem Auto besitzt. Das Rad kostet einmalig 200 Euro und das gebrauchte Auto kostet einmalig 800 Euro. Mit dem Rad benötigt man eine Stunde zur Arbeit, mit dem Auto nur 15 Minuten. Hier ist der Nutzen (Utility) für eine Entscheidung sich ein Rad zu kaufen abhängig von der eigenen finanziellen Lage und der Gewichtung der Zeitersparnis. So ist der Utility Wert für das Rad bei einem Millionär sicherlich gering, weil dieser möglicherweise Zeit mit einem höheren Wert als Geld bemisst. Andererseits besitzt man nicht viel Geld, hat aber viel Zeit, so kann das Rad einen hohen Utility Wert besitzen. Der Utility Wert ist also abhängig vom Kontext und kann sich über die Zeit ändern.

Maximum Expected Utility

Das wichtigste bei Utility Systemen ist das Berechnen der Utility Werte. Für jede Aktion die eine KI ausführen kann, muss ein Utility Wert berechnet werden. Anschließend kann dann anhand

der Utilitys eine Entscheidung getroffen werden. In der Regel die Aktion mit dem höchsten Utility Wert. Da die meisten Spiele nichtdeterministisch sind, ist es schwer die optimale Aktion zu ermitteln. Oft sind die Information über die Folgen einzelnen Aktionen nicht ausreichen. Da es in Echtzeitsystemen oft nicht möglich ist, alle Auswirkungen eines möglichen Zuges zu betrachten, muss eine Aktion gewählt werden von der man glaubt das diese die geeignetste ist. Es wird also die beste Vermutung, abgängig von den unvollständigen Informationen die zur Verfügung stehen, getroffen.

Eines der meist verwendeten Verfahren ist es die Utility mit der Wahrscheinlichkeit von jedem infrage kommenden Ergebnis zu multiplizieren und zu summieren. Die Expected Utility wird wie folgt berechnet:

$$EU = \sum_{n=1}^{N} D_i P_i \tag{4.1}$$

D ist der Utility Wert und P die Wahrscheinlichkeit das, das Ereignis eintritt. Die Wahrscheinlichkeit ist normalisiert, der Wert hat einen Bereich zwischen 0 und 1, sodass die Summe aller Wahrscheinlichkeiten einen Wert von 1 ergibt. Dies wird für jede mögliche Aktion ausgeführt und die Aktion mit dem höchsten Wert wird ausgewählt.

Utility und Behavior Trees

Behavior Trees haben eine fundamentale Limitierung, wenn es darum geht, eine Entscheidung aus mehreren möglichen Aktionen zu wählen. Durch das Verwenden von Utility basierten Entscheidungen können einige der Probleme, die ein Behavior Tree hat, gelöst werden. So kann unter Verwendung, der hier vorgestellten Utility Systeme, ein Utility Selektor entworfen werden. Dieser kann anhand von Utility Werten einen Kinderknoten selektieren. Dadurch kann die Dynamik erhöht werden, ohne die Vorteile von Behavior Trees aufzugeben.

5. Agentenarchitektur

Dieses Kapitel beschreibt die Architektur des emotional agierenden KI-Agenten. Es werden alle Komponenten der Architektur und deren Implementation beschreiben, ebenfalls werden wichtige Designentscheidungen und Implementierungsdetails besprochen.

5.1. Systemübersicht

In der Abbildung 5.1 ist die Gesamtarchitektur mit allen Subsystemen des Agenten skizziert. Folgend wird nun eine kurze Übersicht der Aufgaben und Funktionen der Subsysteme gegeben.

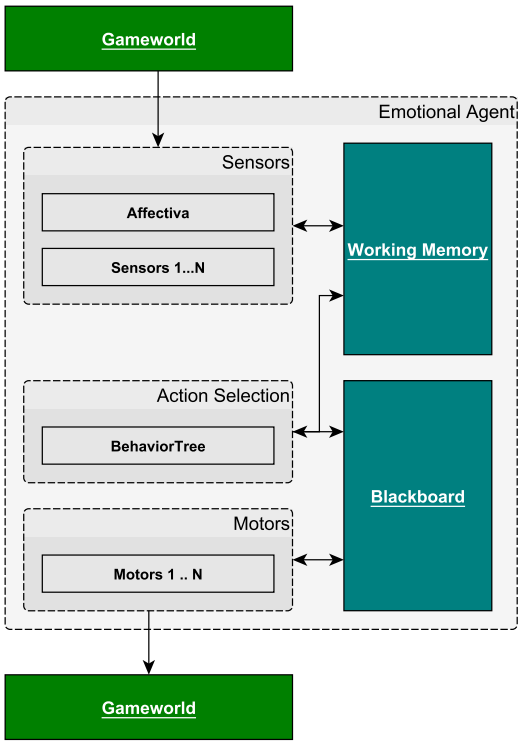

Abbildung 5.1.: Agentenarchitektur

Blackboard: Dient als zentrale Kommunikationseinheit zwischen der Action Selection und den Motors. Das Blackbord biete eine lose Koppelung der Komponenten, wodurch das System flexibel erweiterbar ist. Die lose Koppelung ermöglicht es Komponenten leicht zu warten, ohne das Gesamtsystem zu verändern.

Working Memory: Ist im Kern identisch mit dem Blackboard. Dient zur Kommunikation zwischen den Sensoren und der Action Selection.

Behavior Tree: Expliziere Implementation eines Action Selection Algorithmus. Alle Entscheidungen, die ein Agent treffen kann, werden über einen Behavior Tree modelliert.

Affectiva: Affectiva (2017) ist ein Software Development Kit zur Echtzeiterkennung von Emotionen über kamerabasierte Sensoren. Affectiva erkennt bis zu 21 verschiedene Gesichtsausdrücke und sieben verschiedene Emotionen. Das Software Development Kit von Affectiva ist frei verfügbar, wobei es Einschränkungen bei kommerziellen Gebrauch gibt.

Motors: Motors dienen zur Interaktion mit der Umwelt (Spielewelt). Sie implementieren die Aktionen, die ein Agent ausführen kann. Beispiele für Aktionen, die von einem Motor ausgeführt werden, sind: Abspielen von Animationen, Bewegungen im Raum, Interaktion mit Objekten, dem Spieler und anderen Agenten.

Sensors: Detektieren den Zustand der Umwelt. Sie dienen als Augen und Ohren des Agenten. Alle Informationen, die dem Agenten zur Verfügung stehen, werden durch die Sensoren bereitgestellt und gegebenenfalls gefiltert.

5.2. Anforderungen an die Architektur

Wiederverwendbarkeit: Komponenten müssen wiederverwendbar sein. Eine Sammlung von Werkzeugen, die genutzt werden kann, um Verhalten zu modifizieren oder als Basis für neue Agenten verwendet zu werden.

Skalierbarkeit: Im Kontext der Spieleentwicklung gibt es ein striktes Zeitbudget, das nicht unterschritten werden darf. Die Anforderung für ein flüssiges Spielerlebnis beträgt 30 Bilder pro Sekunde, welches einem Zeitbudget von 33 ms pro Bild entspricht. Bei den meisten Spielen wird jedoch eher ein Budget von 16 ms angestrebt (60 Bilder pro Sekunde). Die Vorgaben bei Virtual Reality sind meist noch strikter um Motion-Sickness entgegenzuwirken. Für eine optimale Spielerfahrung wird eine Bildrate von 90 Bildern pro Sekunde empfohlen.

Benutzbarkeit: Die Architektur sollte einfach und verständlich strukturiert sein. Spieleentwicklung ist ein hoch interdisziplinäres Feld: Das Verhalten von KI-Agenten wird oft von Designern modelliert und nicht von den KI-Entwicklern selbst. Deshalb ist es wichtig eine Architektur zu entwickeln, die im realen Produktionsbetrieb mit Hilfe von spezialisierten Editoren Anwendern leicht zugänglich gemacht wird.

Daten Getrieben: Lose Koppelung einzelner Subsysteme durch ein Datengetriebes System. Erhöht die Testbarkeit des Systems und die Wiederverwendbarkeit einzelner Komponenten. Ermöglicht außerdem das Modellieren mittels grafischer Tools.

5.3. C4 Architektur

Die C4 Architektur ist zur Verhaltenssimulation von Agenten in virtuellen Welten entwickelt worden. Die Architektur wurde durch Rabin (2002) und Isla u. a. (2001) beschrieben. Entwickelt wurde diese durch das Media Laboratory am MIT (Massachusetts Institute of Technology). Die Architektur ist in der Abbildung 5.2 dargestellt.

Die C4 Architektur wurde durch die Möglichkeiten und Einschränkungen realer Tiere inspiriert. Die Anforderungen an die Architektur werden im Folgenden aufgelistet:

- Unterstützung von Grafik mit einer Bildrate von mindesten 20 Hz pro Sekunde.
- Unterstützung von Eingabegeräten wie Maus, Tastatur und Mikrofon, sowie andere exotischere Eingabegeräte wie Kinect, leap motion etc.
- Unterstützung von netzwerkbasiertem Rendering.
- Skalierbarkeit zur Unterstützung einer angemessenen Anzahl an Agenten, die mit der Umwelt und untereinander interagieren können.
- Das Erleichtern der Erstellung und Verwaltung von Agenten.

Die folgende Punkte wurden deshalb als Hauptziele festgelegt:

- Der Agent sollte einfach zu designen und zu implementieren sein.
- Reaktives Verhalten: Der Agent besitzt ein reaktives Verhalten, vergleichbar mit dem anderer autonomer Agenten.
- Lernende Agenten: Die Agenten sollen die Möglichkeit besitzen anhand von Bestrafung und Belohnungen als Reaktion aus der Umwelt ihr Verhalten anzupassen.
- Erweiterbarkeit: Die Architektur sollte einfach erweiterbar sein, um Forschung in verschiedenen Bereichen zu unterstützen.

Generell ist die Architektur so organisiert, dass die einzelnen Komponenten der C4 Architektur über ein internes Blackboard miteinander kommunizieren. Jede der Komponenten kann von diesem Blackboard lesen und schreiben. Die einzelnen Komponenten der C4 Architektur (Abb. 5.2) werden im folgenden Abschnitt beschrieben.

World Model: Generell ist das World Model mit dem Verhalten und Funktionen, die durch die meisten Game Engines bereitgestellt werden, zu vergleichen. Die Aufgabe des World

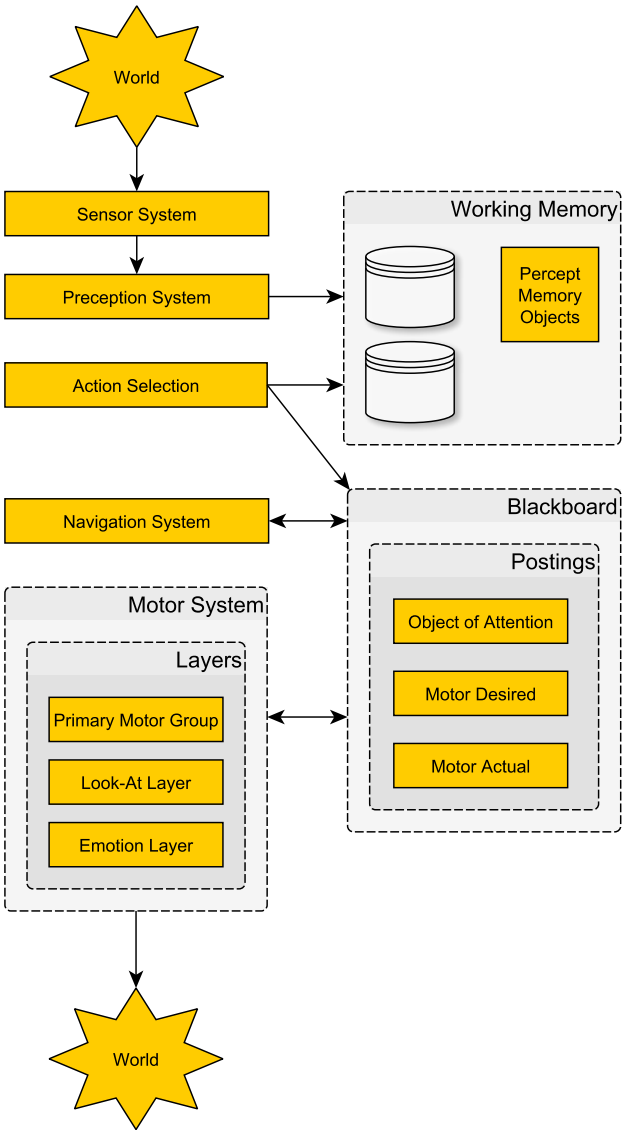

Abbildung 5.2.: C4 Architektur nach Isla u. a. (2001)

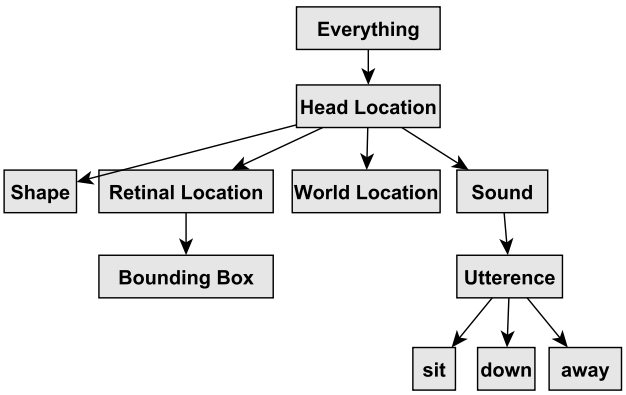

Abbildung 5.3.: C4 Perception System nach Isla u. a. (2001)

Model ist es, eine Liste aller Agenten und Objekte innerhalb der Spielwelt zu verwalten. Ein Event Blackboard für den Agenten bereitzustellen, das Ereignisse und den Zustand der Spielwelt an die Sensoren und Wahrnehmung des Agenten weiterleitet. Des Weiteren wird das Netzwerk und Darstellung verwaltet. World Events treten in der Form von Data Records auf, die von den Sensory und Preception Komponenten des Agenten verarbeitet werden können. Der Inhalt eines Data Record ist frei definierbar. Ob und wie diese Daten verarbeitet werden, hängt von den Sensoren und der Wahrnehmung des Agenten ab. Innerhalb eines Updates werden die Events an die einzelnen Komponenten des Agenten weitergeleitet. Die meisten Komponenten wiederum produzieren daraufhin Ereignisse, die in die Spielwelt zurückgegeben werden, um diese zu verändern. Die Änderungen werden verarbeitet und der daraus resultierende Zustand im nächsten Update-Schritt an die Komponenten weitergegeben. Das schließt Operationen, wie das Aktualisieren der UI, Netzwerk und Redering mit ein.

Sensory System: Das Sensory System verarbeitet den Zustand und Ereignisse des World Model. Das Sensorsystem dient als Filter zwischen dem Agenten und dem World Modell durch das alle Data Records laufen. Anders als bei Sensoren in der realen Welt, die eine begrenzte Anzahl an Informationen besitzen, hat man innerhalb eines Spiels Zugriff auf den kompletten Zustand des Spiels und theoretisch auch alle relevanten Daten. Folglich ist die Aufgabe des Sensory System, die Anzahl der Daten zu limitieren. Die Architektur trennt die Aufgabenbereiche deutlich. Bei Spielen ist es oft nicht gewünscht einen Agenten so zu modellieren, dass dieser komplettes Wissen über seine Umwelt besitzt und dadurch mit unmenschlicher Effizienz operiert. Vielmehr geht es darum, glaubwürdiges

Verhalten zu simulieren. Das Sensory System ist für das Sehen des Agenten zuständig. Das System muss in der Lage sein, die Informationen so zu Filtern, dass die wahrgenommenen Informationen glaubwürdig erscheinen. Agenten, die z. B. hinter einer Mauer stehen, sollten von einem solchen Sensory System nicht detektiert werden, auch wenn diese Information theoretisch zur Verfügung steht. Es kann aber durchaus auch angebracht sein, um ein gewünschtes Verhalten zu erhalten. Anders, als in der Robotik, wo alle Daten möglichst effizient und komplett verarbeitet werden müssen. Bei Spielen hat generell das resultierende Verhalten im Vordergrund. Die Priorität ist abhängig vom Game Design.

Perception System: Das Perception System hat die Aufgabe, zu entscheiden, welche der Informationen, die durch das Sensory System geliefert werden, weiter verarbeitet werden. So kann das Sensory System alle sichtbaren Objekte liefern: Diese werden weiter verarbeitet und alle für den Agenten uninteressanten Informationen herausgefiltert. Es wird nicht beschrieben wie die Daten des Sensory Systems verarbeitet werden, welche Daten verarbeitet werden, in welcher Form diese weitergegeben werden und wie ein Percept implementiert werden muss. Ein Percept kann ein simples „switch case..“, ein „if then... else ...' oder ein komplexes Subsystem sein. Percept System ist hierarchisch angeordnet (siehe 5.3), so können Percepts immer weiter verfeinert werden. Die Ergebnisse werden als Percept Memory Objects gespeichert.

Action Selection: In der Spezifikation wird ein Action System und Action Tuples beschrieben. Diese beiden Komponenten sind sehr spezifisch für die im dort vorgestellte Entscheidungsfindung. Beide Komponenten wurden deshalb als Action Selection zusammengefasst. Anhand der Daten, die das Perception System liefert, kann der Agent entscheiden, welche Aktionen ausgeführt werden sollen. Die C4 Architektur ist unabhängig von der implementierten Entscheidungsfindung, sodass eine Menge verschiedeneren Action Selection Algorithmen möglich sind. Vorgesehen ist, dass die Ergebnisse der Action Selection in ein Blackboard geschrieben werden, was wie auch der Working Memory ein genereller Datenspeicher ist über den die Komponenten miteinander kommunizieren. Diese Postings werden von dem Navigation System und dem Motor System gelesen. Je nach Inhalt des Postings wird dann das Navigation System oder das Motor System eine Aktion ausführen.

Working Memory: Der Working Memory ist ein genereller Datenspeicher, er stellt das Gedächtnis des Agenten dar. Ziel ist es, den aktuellen erfassten Zustand der Umwelt zu speichern. Diese Daten dienen als Basis für die Action Selektion. Da alle verarbeiteten Sensordaten innerhalb des Working Memory gespeichert werden, können komplexe

Anfragen durch die Action Selection durchgeführt werden. Beispielsweise kann die Position eines jeden erfassten Agenten abgefragt und berechnet werden, welcher die geringste Distanz zum Agenten hat. Es kann auch eine Historie von Ereignissen geführt werden, wenn den Daten ein Timestamp hinzugefügt wird. Das Working Memory kann so zu einer komplexen Datenbank von Sensordaten werden.

Navigation System: Ist für die Navigation des Agenten zuständig z.B. Pathfinding, Steering und Obstacle Avoidance. Zusätzlich hat das Navigation System die Möglichkeit das Motor System zu unterbrechen. Befindet sich ein Posting auf dem Blackboard, beispielsweise um ein Item vom Boden aufzuheben, so muss das Navigation System erkennen, ob der Agent vor dem Objekt steht und dieses aufheben kann. Ist das nicht der Fall, muss sich der Agent erst vor das Objekt bewegen. Das Navigation System muss in dem Fall unterbrechen, den Agenten an die richtige Position bewegen und anschließend die Aktion für das Aufheben ausführen. Das Navigation System entscheidet also letztendlich, welche Aktionen vom Motor System ausgeführt werden können und welche Aktionen das Navigation System benötigen.

Motor System: Das System steuert alle Interaktionen mit der Umwelt. Beispiele für ein Motorsystem sind z.B das Animationssystem, Sound, grafische Veränderungen, UI Controls und weitere Aktionen, wie das Benutzen eines Gegenstandes. Die einzelnen Komponenten des Motor Systems entscheiden anhand von Postings auf dem Blackboard, ob diese ausgeführt werden. Damit die einzelnen Komponenten des Motor Systems und das Navigationssystem zusammenarbeiten können, wird der Status einer Aktionen als Postings auf dem Blackboard hinterlegt.

5.4. Agent

Der Agent implementiert die in 5.3 vorgestellte C4 Agentenarchitektur in abgewandelter Form. Im Folgenden werden die Unterscheide zwischen der C4 Architektur und der für den Agenten verwendeten Architektur beschrieben.

5.4.1. Architekturunterschiede zu C4

Bei der Entwicklung von KI-Agenten für Computerspiele ist zu beachten, dass es viele unterschiedliche Agenten geben kann, deren Grundverhalten und Aktionsmöglichkeiten bis zu einem gewissen Grad gleich sind. Diese können sich jedoch auch minimal oder drastisch in ihren Fähigkeiten und Aktionsmöglichkeiten unterscheiden. Deshalb ist es wichtig eine

möglichst modulare Architektur zu entwickeln, die es erlaubt, Komponenten im großen Maße wiederzuverwenden und die flexibel genug ist, neues Verhalten hinzuzufügen, ohne bereits bestehendes Verhalten modifizieren zu müssen. Deshalb wird die Architektur des Sensory System, Perception System, Navigation Systems und des Motor Systems in ein komponentenorientiertes System transformiert. Dadurch wird die Wiederverwendbarkeit der Sensoren und Motoren erhöht, sodass diese nach dem Baukastenprinzip hinzugefügt und entfernt werden können. Das Erweitern der Funktionalität von Agenten durch die Entwicklung neuer Sensoren und Motoren wird deutlich erleichtert.

Änderungen am Sensory System und Perception System: Die beiden Systeme wurden zu einer Komponente zusammengeführt. Jede Sensor Komponente enthält sowohl das Sensory System als auch das Perception System, welche in 5.3 vorgestellt wurden. Ziel der Umstrukturierung ist es, das System weiter zu entkoppeln und die Wiederverwendbarkeit zu erhöhen. Anstelle eines Gesamtsystems, welches alle für den Agenten interessanten Daten aus dem World Model extrahiert und anschließend über ein Perception System filtert, werden diese Aufgaben nun über einzelne Komponenten verteilt, sodass Erfassung und Filterung dieser Daten leichter in Subsysteme aufgeteilt werden. Eine Sensor Komponente liefert immer nur einen Sensorwert, der bereits gefiltert wurde. Ein Positionssensor beispielsweise, liefert nur Positionen und filtert diese, um die für diesen Agenten wichtigen Informationen zu liefern. Es können jedoch mehrere Positionssensoren existieren, die je nach Fähigkeiten des Agenten, verschiedene genaue oder andere Position Ergebnisse liefern. So kann es beispielsweise einen Positionssensor geben, der alle möglichen Agenten innerhalb eines Bereiches liefert. Möchte man nun einen Agenten erstellen, der nur die Position des am nächsten stehenden Agenten bestimmt, so kann diese eine Komponente modifiziert werden oder ein zusätzlicher Sensor erstellt werden. Wodurch der vorherige Sensor erhalten bleibt und das Verhalten der bereits bestehenden Agenten nicht verändert wird.

Änderungen am Navigation System und Motor System: Wurden zu einzelnen Motor Komponenten zusammengefasst, die unabhängig voneinander über ihre Ausführung entscheiden. Diese werden wie bei der C4 Architektur von der Action Selection über das Blackboard gestreut. Es liegt in der Verantwortung der Action Selection dafür zu sorgen, das keine konkurrierenden Aktionen gleichzeitig ausgeführt werden. Auch hier bietet der komponentenbasierte Ansatz mehr Flexibilität und Unabhängigkeit.

5.4.2. Update-Schritt

Der Komplette Update-Schritt, den ein Agent jeden Frame durchführt, ist in 5.4 und 5.1 illustriert. Zuerst werden die Sensoren, Motoren, Blackboard und Working Memory Initialisiert. Danach beginnt der eigentliche Update-Schritt. Es werden alle Sensoren aktualisiert, deren Update-Funktion aktiviert ist. Anschließend wird die Action Selection ausgeführt. Als letztes werden alle Motoren aktualisiert. Anschließend beginnt der Vorgang mit dem nächsten Frame von vorne.

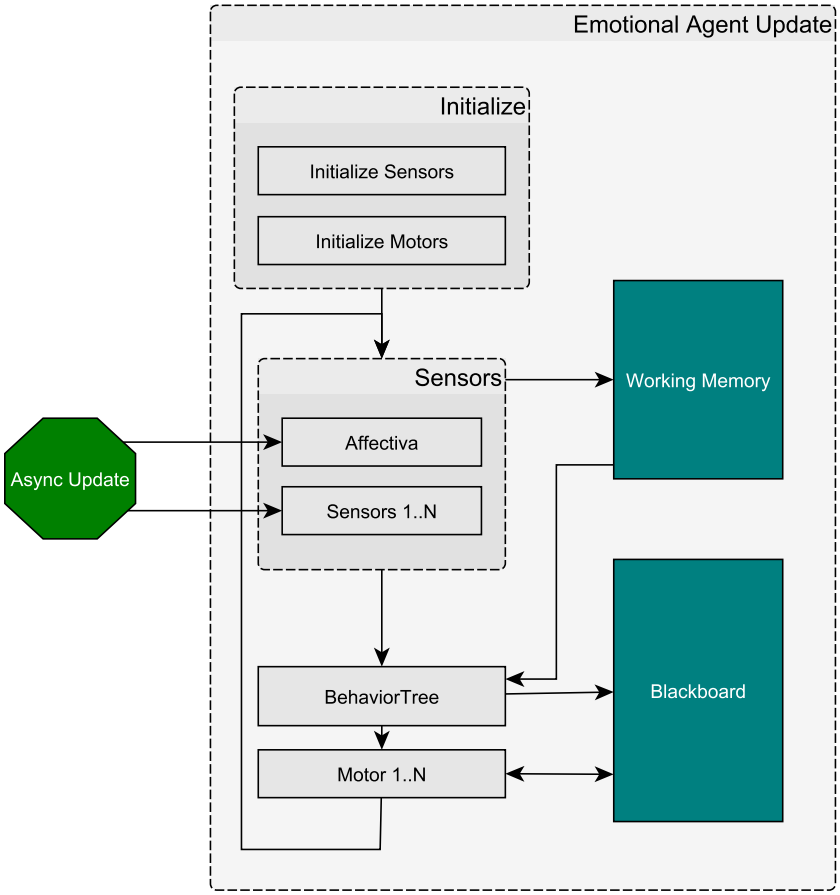

Abbildung 5.4.: Emotional Agent Update-Schritt

```
foreach (var sensor in Sensors)
{
  if (!sensor.UpdateEnabled()) continue;
  sensor.UpdateSensor(deltaTime);
```

```
5 }
6 var status = Tree.Update();
7 foreach (var motor in Motors)
8   motor.UpdateMotor();
```

Listing 5.1: Agent Update

5.5. Sensors

Die Aufgabe der Sensors ist es Daten aus der Umwelt zu verarbeiten und zu filtern, um sie dem Agenten für den Entscheidungsprozess zur Verfügung zu stellen. Die Daten werden anschließend im Working Memory des Agenten gespeichert. Welche Daten ein Sensor erfasst, hängt von der Implementation des Sensors ab. Die Repräsentation der Daten ist nicht einheitlich und variiert zwischen den Sensor Implementationen.

In der Grafik 5.5 wird das Interface eines Sensors beschrieben. Der Sensor besitzt drei Funktionen, „UpdateEnabled, „UpdateSensor und „Initialize“. Um Daten schreiben zu können, besitzt jeder Sensor eine Referenz auf den Working Memory.

Initialize: In dieser Funktion werden alle Operationen, die der Sensor für die Initialisierung benötigt, ausgeführt. Die Funktion wird einmalig aufgerufen, wenn sich der Agent initialisiert.

UpdateEnabled: Über die Funktion teilt der Sensor dem Agent mit, ob er synchron aktualisiert werden muss. Damit ist es dem Entwickler überlassen ob die„UpdateSensor“ Funktion implementiert wird. Beispielsweise Sensoren, die asynchron aktualisiert werden. Eine Webcam besitzt eigenen Takt, der unabhängig vom dem des Agenten ist. Die Daten der Webcam werden asynchron aktualisiert können aber mit jedem Aktualisierungsschritt des Agenten gelesenen werden. Weiter kann es Sensoren geben, die zeitgesteuert aktualisiert werden. Diese Sensoren würden sich eigenständig aktualisieren und benötigen kein synchrones Update.

UpdateSensor: Wird von dem Agenten benutzt um den Sensor synchron zu aktualisieren. Der Sensor detektiert für ihn relevante Informationen aus der Umwelt, diese werden anschließen gefiltert, konvertiert und in den Working Memory geschrieben.

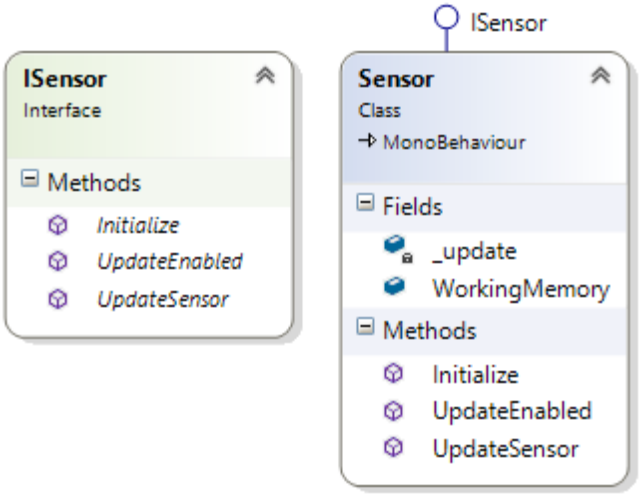

Abbildung 5.5.: Klassendiagramm Sensors

5.6. Motors

Ein Motor hat die Aufgabe die von der Action Selection generierten Aufträge auszuführen. Sie sind für die Interaktion mit der Umwelt zuständig. Bei jedem Update überprüft ein Motor, ob für ihn wichtige Kontrollnachrichten auf dem Blackboard hinterlegt wurden. Ist eine Kontrollnachricht auf dem Blackboard hinterlegt, wird diese entfernt und der Motor erfüllt seine Aufgabe. Der Motor teil dann der Action Selection seinen Status über das Blackboard mit. Hierfür generiert der Motor eine Statusnachricht; diese kann sich von Motor zu Motor unterscheiden. So gibt es Motors die ihre Operation innerhalb eines Aufrufs abschließen. Ein solcher Motor würde keine Statusnachricht auf dem Blackboard generieren. Andere Motoren benötigen mehrere Updates bis eine Aktion beendet ist. Einige delegieren Aufträge an Subsysteme wie das Animations-System und warten auf Kontrollnachrichten von diesen Subsystemen. Ein Solcher Motor würde eine Statusnachricht auf dem Blackboard hinterlegen. Beispielsweise könnte die Statusnachricht ein Wert sein, der angibt das der Auftrag noch bearbeitet wird oder ein nummerischer Wert der den Fortschritt der Operation beschreibt.

Damit der Motor auf das Blackboard zugreifen kann, besitzt die Implementierung eine Referenz auf das Blackboard. Ein Motor muss die folgenden Funktionen (siehe 5.6) Implementieren:

Initialize: Dient zur Initialisierung des Motors durch den Agenten (Optional).

UpdateMotor: Diese Funktion wird vom Agenten aufgerufen um den Motor zu aktualisieren. In diesem Aufruf wird das Blackboard nach einer Statusnachricht überprüft. Wenn

eine vorhanden ist, wird die Operation ausgeführt. Wird bereits eine Operation ausgeführt, kann diese weiter bearbeitet werden und die Statusnachricht auf dem Blackboard aktualisiert werden.

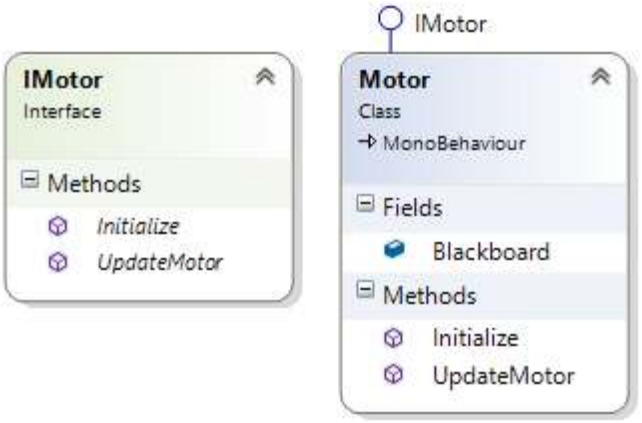

Abbildung 5.6.: Klassendiagramm Motors

5.7. Blackboard

Informationen sind eines der Fundamente der künstlichen Intelligenz. Der Agent muss gesammeltes Wissen und seinen internen Zustand Speicher und verwalten können. Er muss sich mit anderen Agenten oder Subsystemen koordinieren und kommunizieren können. Das Blackboard bietet eine einfach zu verstehende und implementierbare Möglichkeit diese Daten zu verwalten und zu verarbeiten.

5.7.1. Generelle Architektur

Im Folgenden wird nun das generelle Konzept vom Blackboards vorgestellt. Die hier beschriebene Architektur wurde durch Rabin (2002) und Erman u. a. (1980) definiert.

In der Abbildung 5.7 ist das System nach Rabin (2002) veranschaulicht. Die hier beschrieben Architektur besteht aus drei verschiedenen Komponenten:

Blackboard: Das Blackboard ist eine Sammlung von Daten auf denen andere Komponenten arbeiten. Es ist möglich diese Daten unsortiert zu speichern oder in Kategorien zu strukturieren. Dadurch brauchen Komponenten nur den Bereich überprüfen, an dem sie interessiert sind.

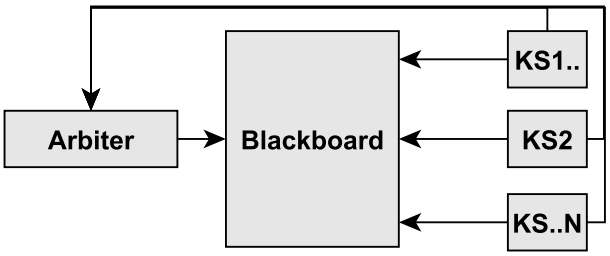

Abbildung 5.7.: Blackboard Architekture

Knowledge Source (KS): Sind Komponenten, die auf den Daten des Blackbords arbeiten. Sie werden als vielfältig unterschiedliche Programme beschreiben, die unabhängig voneinander sind. Diese Komponenten sind Spezialisten, die zusammenarbeiten, um ein Problem zu lösen. Jede der Komponenten löst eine einzige Unteraufgabe innerhalb des Gesamtsystems. Sie werden als „Indipendent condition-action module“ beschrieben. Sie besitzen eine Bedingung, wenn diese erfüllt wird, wird eine Aktion erzeugt. Im Regelfall sind die KS inaktiv und warten auf Ausführung durch den Arbiter. Die einzelnen Aktionen des KS verändern die Daten des Blackboards. Ein KS kann vorhandene Daten verändern, neue Daten generieren oder Kontrollsignale für andere KS hinterlegen. Durch diese Veränderungen können die Bedingungen von anderen KS erfüllt werden. Es ist den KS nur erlaubt über das Blackboard miteinander zu kommunizieren.

Zum Beispiel existiert im Spielkontext ein KS1 das einen Pfad zwischen den Punkten P1 und P2 berechnet. Ein weitere KS2 das die Anfangs- und Endpunkte für den Pfad ermittelt. Beide KS müssen Daten austauschen und miteinander kommunizieren um, einen geeigneten Pfad zu finden. Beide KS besitzen eine Bedingung und eine Aktion, die ausgeführt wird, sobald die Bedingung erfüllt ist. Die Bedingung für das KS2 ist es geeignet Punkte für die Pfadfindung zu ermitteln, darauffolgende werden diese Punkte über das Blackboard publiziert. Das KS1 wartet auf zwei geeignete Punkte, um anschließend einen Pfad zu generieren.

Arbiter: Die Aufgabe des Arbiters ist es, zu entscheiden welche Knowledge Source ausgeführt werden sollen. Die Strategie, nach der einzelne KS ausgeführt werden sollen, unterscheidet sich in Abhängigkeit von der Implementation. Eine mögliche Strategie ist, die KS in Abhängigkeit ihrer Priorität und durch Prüfung von Vorbedingungen nacheinander auszuführen. Gibt es mehrere KS, die auf die gleichen Ressourcen zugreifen und diese ver-

ändern, ist das Ergebnis abhängig von der Reihenfolge in der die KS ausgeführt werden. Es liegt in der Verantwortung des Entwicklers eine geeignete Strategie zu entwickeln.

Generell wird für einen einzelnen Update-Schritt eine Liste relevanter KS selektiert und an den Arbiter weitergegeben. Der Arbiter entscheidet dann anhand seiner Auswahlstrategie, welches der KS ausgeführt wird. Danach kann der daraus resultierende neue Zustand des Blackbords auf eine Terminierungsbedingung überprüft werden. Wenn die Bedingung nicht erfüllt, wurde beginnt der Vorgang von vorne, bis der Zustand erreicht wird, der zur Terminierung führt.

In der Praxis wird die oben beschriebene Architektur jedoch meist nicht strikt befolgt. In den meisten Fällen wird auf den Arbiter verzichtet, sodass eine beliebige Anzahl von KS ausgeführt werden können. So werden Blackboards meist nicht mehr als Problemlöser eingesetzt, wie in Erman u. a. (1980) beschrieben. Vielmehr werden sie zur Steuerung und Koordinierung von Subsystemen eingesetzt, zur Kommunikation mit anderen Agenten und zum Speicher des eigenen Zustands. Gründe hierfür sind, dass es sich bei einem KS um ein umfangreiches Subsystem handeln kann, welches das Blackboard zur Koordination mit den anderen Subsystemen einsetzt. Trotz der Unterschiede zwischen der beschriebenen Architektur und den Ansätzen aus der Praxis gelten für diese Systeme weiterhin die folgenden fundamentalen Funktionen, durch Rabin (2002) und Erman u. a. (1980) beschrieben wurden:

- Ein KS muss nicht wissen, wann und wie die Daten von anderen Komponenten in das Backboard gespeichert werden.
- Ein KS muss nicht wissen, von wem die Daten in das Blackboard geschrieben wurden.
- Ein KS kann Daten von dem Blackboard lesen und schreiben.

Betrachtet man diese drei Punkte, entsprechen sie den Funktionen des Publish/Subscribe Entwurfsmuster. Es existiert ein oder mehrere Publisher, diese verteilen Daten an eine spezifizierte Gruppe. Der Publisher besitzt kein Wissen darüber, an welche Empfänger die Daten gesendet werden oder ob es überhaupt Empfänger gibt. Ein Subscriber bekundet sein Interesse an Daten die über eine spezifische Gruppe veröffentlicht werden. Der Subscriber weiß nicht, wer der Publisher ist oder ob ein Publischer existiert. Dieses Entwurfsmuster wird unter anderem bei Netzwerksystemen wie ActiveMQ (vgl. Foundation) oder Source-Specific-Multicast (vgl. Holbrook und Cain (2006)) angewendet.

5.7.2. Implementation

Genauso wie das in 5.7 vorgestellte Blackboard bietet die Implementation die folgenden Funktionalität:

- Lesen: Zu jedem Zeitpunkt kann jede Komponente, die Zugriff auf das Blackbord hat, von diesem lesen.
- Schreiben: Zu jedem Zeitpunkt kann jede Komponente, die Zugriff auf das Blackboard hat, Daten schreiben.
- Daten: Die Art der Daten wird von dem Blackboard nicht Festgelegt. Es können Texte, Zahlen, Listen oder auch Objekte in dem Blackbord gespeichert werden.

Das Klassendiagramm 5.8 zeig die Implementation des Blackboards. Das Blackboard ist eine in Memory Database in der Key-Value Pairs gespeichert werden. Die Daten werden in einem Dictionary gespeichert, welches einen String und den generischen Datentyp „BlackboardProperty“ enthält. Der Name String ist ein Schlüssel zur Identifizierung der Daten. Eine Komponente, die Daten lesen oder schreiben möchte, muss einen Schlüssel verwenden, der den anderen Komponenten im System bekannt ist. Die „BlackboardProperty“ ist eine Container Klasse die den eigentlichen Inhalt kapselt. Diese einheitliche Schnittstelle ermöglicht dem Blackboard typsicheres arbeiten. Obwohl es sich bei dem Blackboard um einen generischen Speicher handelt, können so nur typisierte Daten gelesen und geschrieben werden. Dadurch werden Laufzeitfehler minimiert, die oft durch generische Zugriffe hervorgerufen werden.

Vorteile

Das Blackbord kann als interner und externer Speicher eingesetzt werden. Es können nicht nur einzelne Komponenten, sondern auch Agenten untereinander kommunizieren. Da das Blackboard nicht vorschreibt, welche Art von Daten gespeichert werden, ist es sehr modular einsetzbar. Es ist nicht mehr nötig spezifische Implementation, die abhängig von vordefinierten Datenstrukturen und Einsatzbereichen sind, zu implementieren. Da die Agenten oder Komponenten nicht abhängig voneinander sind, haben Änderung in einen der Komponenten oder Agenten keine Auswirkungen auf die anderen Module. Dies erhöht die Wartbarkeit einzelner Komponenten erheblich. Dies ist besonders vorteilhaft in Systemen, die häufig überarbeitet und angepasst werden. Diese Struktur ist optimal für die Spieleentwicklung, in der Verhalten und Gameplay während der Entwicklung oft überarbeitet oder komplett neu definiert werden muss.

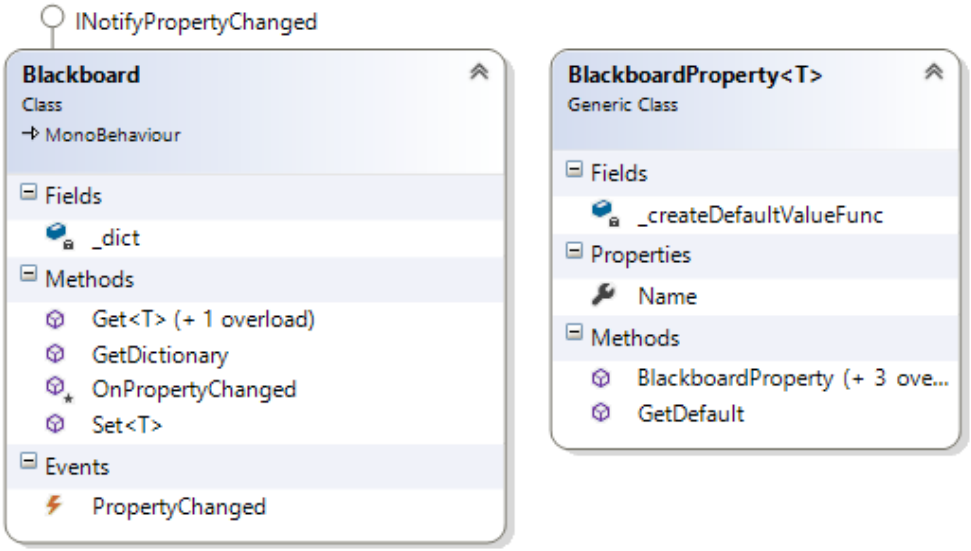

Abbildung 5.8.: Blackboard Klassendiagramm

5.8. Behavior Tree

In diesem Abschnitt wird die Implementation des in Kapitel 4.3.2 beschriebenen Behavior Trees erläutert. Es wird exemplarisch beschrieben, wie das System mit Unit Tests auf Funktionsweise überprüft wurde.

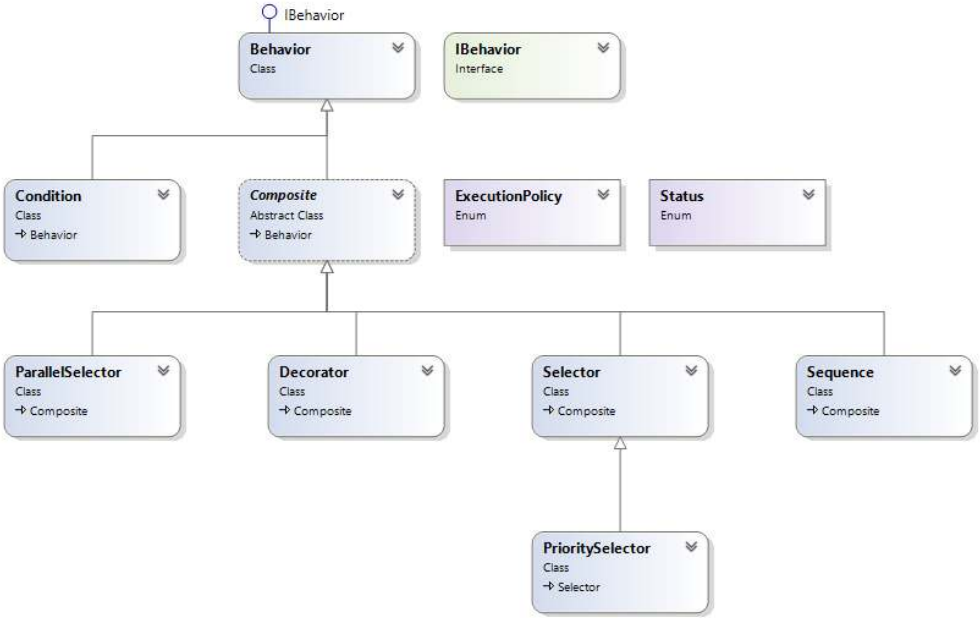

Abbildung 5.9.: Behavior Tree Klassen Diagramm

Architektur: Die Abbildung 5.9 zeigt die Klassenhierarchie, aus der sich der Behavior Tree zusammensetzt. Alle Klassen in dem Behavior Tree erben von der Behavior, die das Interface IBehavior (siehe 5.2) implementiert. Diese Klasse beschreibt das Grundverhalten eines jeden Knoten innerhalb des Behavior Trees. Eine Aktion erbt direkt von dieser Klasse z.B. das „MockBehavior“ oder Condition. Als weitere Basisklasse stellt die Composite Klasse die Grundfunktionen für alle Selektoren und den Decorator dar. Sie erbt ebenfalls von Behavior und implementiert u. a. die Datenstrukturen zum Verwalten von Kinderknoten.

Im Folgenden werden die Klassen Behavior, Composite und Selector im Detail erläutert:

Behavior: Jedes Behavior besitzt die folgenden Eigenschaften: Den Namen, eine eineindeutige ID, ein Parent und den aktuellen Status. Für die Implementation wurde der Status noch um einen weiteren Zustand erweitert. Der Status „Invalid“ gibt an, dass ein Behavior noch nicht initialisiert wurde. Das Hauptverhalten der Behavior Klasse ist über die Funktion „Tick“

implementiert (siehe 5.3). Diese ist für alle Ausprägungen von Knoten wie z. B. Selector, Action, Condition, Decorator identisch.

1. Prüfen den aktuellen Status, ist der Status „Invalid“ führe die „Initialize“ Funktion aus.
2. Führe die „Update“ Funktion aus.
3. Ist der aktuelle Status ein anderer als „Running“ führe „Terminate“ aus.
4. Gib den aktuellen Status zurück.

Die Funktionen „Initialize, Update, Terminate und Reset“ sind virtuelle Funktionen und können von einer Unterklasse implementiert werden.

```
public interface IBehavior{
  String Name { get; set; }
  int Id { get; set; }
  IBehavior Parent { get; set; }
  Status BtStatus { get; set; }
  void Initialize();
  Status Tick();
  Status Update();
  void Terminate(Status status);
  void Reset();
}
```

Listing 5.2: IBehavior Interface

```
public Status Tick(){
  if (BtStatus == Status.BtInvalid){
    Initialize();
  }
  BtStatus = Update();
  if (BtStatus != Status.BtRunning){
    Terminate(BtStatus);
  }
  return BtStatus;
}
```

Listing 5.3: Behavior Tick Funktion

Composite: Das Composite ist die Basisklasse aller Knoten, die über Kinder verfügen, dazu gehören u. a. alle Selektoren und der Decorator. Die Composite Klasse erbt das Grundverhalten von Behavior und wird um die Verwaltung von Kinderknoten erweitert. Hierfür wird als zusätzliche Eigenschaft eine Liste über IBehaviors hinzugefügt. Zusätzlich zu der neuen Datenstruktur besitzt ein Composite vier wesentliche Zugriffsfunktionen auf diese Datenstruktur.

- „GetChild(int index)“ liefert ein Kind mit dem angegeben Index zurück.
- „ChildCount“ liefert die Aktuelle Anzahl der Kinder.
- „[int index]“ Indizierungsoperator, dieser wird überladen, sodass man von außen auf die Kinder eines Composite zugreifen kann, wie bei einem Array (siehe 5.4).
- "T ‘Add()“ ist eine generische Factory Funktion zum Hinzufügen von Kindern. Erstellt mithilfe des Standardkonstruktors eine Klasse des angegeben Typen und liefert eine Instanz zurück. Die neu erstellte Instanz wird als Kind hinzugefügt. Die „Add“ Funktion wird beschränkt, indem man nur Objekte hinzufügen kann, die eine Reihe von Kriterien erfüllen. Es muss sich um eine Klasse handeln, die über einen Standardkonstruktor verfügt und von IBehavior erbt.

```
public T Add<T>() where T : class, IBehavior, new(){
  var t = new T {Parent = this};
  Children.Add(t);
  return t;
}
public Composite this[int i]{
  get{
    return Children[i] as IBehavior;
  }
}
```

Listing 5.4: Composite

Selector: Der Selector wählt eines seiner Kinder aus, beginnend mit dem Ersten. Die Kinder werden nacheinander ausgeführt, bis ein Kind ein „Success“ oder „Running“ als Status zurückgibt. Hat eines oder keines der Kinder den Status „Success“ wird bei der nächsten Ausführung die Selektion der Kinder neu begonnen - ansonsten wird das Kind ausgeführt, welches zuletzt den Status „Running“ hatte (siehe 5.5).

```
while (true){
  Status childStatus = GetChild(_currentSelectedChild).Tick();
  if (childStatus != Status.BtFailure){
    if (childStatus == Status.BtSuccess){
      CurrentSelectedChild = 0;
    }
    return childStatus;
  }
  if (++CurrentSelectedChild != ChildCount) continue;
  CurrentSelectedChild = 0;
  return childStatus;
}
```

Listing 5.5: Selector Update Funktion

Testing In diesem Abschnitt wird am Beispiel der Behavior Klasse erklärt, wie dabei vorgegangen wurde, die Funktionalität der Implementation zu testen. Für alle Klassen des Behavior Trees wurden wurden Unit Tests mithilfe des Unity Test Tools erstellt.

Um die Behavior Klasse zu testen, wurde eine Pseudoklasse erstellt. Die Klasse (siehe 5.6) erbt von Behavior, und simuliert eine Action. Mit diesem Test wird einerseits das Grundverhalten

jedes Knoten als auch das Gesamtverhalten der Actions getestet. Generell handelte es sich bei der Implementation um eine testgetriebene Entwicklung. Die konkreten Klassen wurden anhand der zuvor erstellen Test implementiert. So war es während der Implementation möglich schrittweise die einzelnen Funktionen zum Implementieren und über die Tests zu validieren.

Hierfür wurden der Pseudoklasse drei Zähler von Typ Integer hinzugefügt: „i_InitializedCalled“, „i_UpdateCalled“ und „i_TerminateCalled“. Die Variablen werden mit dem Ausführen der entsprechenden Funktionen hochgezählt. Weitere Eigenschaften sind „t_status“ und „t_terminateStatus“, über den Status „t_terminate“ kann bestimmt werden, mit welchem Status ein Behavior terminiert wurde. So sollte es beispielsweise nicht vorkommen, dass ein „t_terminateStatus = BtRunning“ existiert. Der „t_status“ wird nicht aktiv von der Pseudoklasse gesetzt es handelt sich um eine Variable, die während der Tests explizit von außen gesetzt wird. Durch das setzten der Variable kann der Status verändert werden. Der Wert der Variable wird durch die Update Funktion zurückgeliefert.

```
1  public override void Initialize(){
2    ++i_InitializedCalled;
3  }
4  public override Status Update(){
5    ++i_UpdateCalled;
6    return t_status;
7  }
8  public override void Terminate(Status status){
9    ++i_TerminateCalled;
10   t_terminateStatus = status;
11 }
```

Listing 5.6: MockBehavior Klasse

Insgesamt wurden 32 verschiedene Unit Tests durchgeführt, um das gesamte System zu testen. Die Benennung der einzelnen Tests beginnt mit dem Namen der Funktion, die getestet wird, was getestet wird und welches Ergebnis erwartet wird. Nachfolgend werden beispielhaft einige Tests (siehe 5.7) für die Behavior Klasse der Reihenfolge nach beschrieben.

Tick_TestInitialize_SUCCESS: Der erste Test sollte prüfen, ob und wie oft die „Initialize“ Funktion aufgerufen wird. Es wurde geprüft, ob der Zählerstand von „i_initialized“ mit dem Erwartungswert übereinstimmt.

Tick_TestUpdate_SUCCESS: Dieser Test sollte abprüfen ob und wie oft die Update Funktion nach einem „Tick“ aufgerufen wird.

Tick_TestUpdate1000_SUCCESS: Hier wurde untersucht ob das, was in Test 2 angenommen wurde, auch noch nach 1000 aufrufen zutrifft.

Tick_TestTerminate_SUCCESS: In dem letzten Test wurde überprüft, ob der Status im speziellen „BtTerminate“ richtig gesetzt wird. Hierfür wurde der Status nach jedem Aufruf der „Tick“ Funktion verändert, und die Veränderungen mit dem erwarteten Status verglichen.

```
1  public void Tick_TestInitialize_SUCCESS(){
2    MockBehavior bt = new MockBehavior();
3    Assert.AreEqual(0, bt.i_InitializedCalled);
4    bt.Tick();
5    Assert.AreEqual(1, bt.i_InitializedCalled);
6    bt.Tick();
7    Assert.AreEqual(1, bt.i_InitializedCalled);
8  }
9  public void Tick_TestUpdate_SUCCESS(){
10   MockBehavior bt = new MockBehavior();
11   Assert.AreEqual(0, bt.i_UpdateCalled);
12   Status st = bt.Tick();
13   Assert.AreEqual(1, bt.i_UpdateCalled);
14   bt.Tick();
15   Assert.AreEqual(2, bt.i_UpdateCalled);
16 }
17 public void Tick_TestUpdate1000_SUCCESS(){
18   MockBehavior bt = new MockBehavior();
19   for (int i = 0; i < 1000; i++)
20   {
21     Assert.AreEqual(i, bt.i_UpdateCalled);
22     bt.Tick();
23   }
24 }
25 public void Tick_TestTerminate_SUCCESS(){
26   MockBehavior bt = new MockBehavior();
27   bt.Tick();
28   Assert.AreEqual(0, bt.i_TerminateCalled);
29   Assert.AreEqual(Status.BtInvalid, bt.t_terminateStatus);
30   bt.t_status = Status.BtSuccess;
31   bt.Tick();
32   Assert.AreEqual(1, bt.i_TerminateCalled);
```

```
33   Assert.AreEqual(Status.BtSuccess, bt.t_terminateStatus);
34 }
```

Listing 5.7: Test für die Behavior Klasse

5.9. Affectiva und Affdex API

Das Ziel von Affectiva (2017) ist es, emotionale Intelligenz in die digitale Welt zu bringen, um Technologie zu humanisieren.

> At Affectiva, our mission is to bring emotional intelligence to the digital world; we are humanizing technology.

Hierfür stellt die Firma Affectiva ein Software Development Kit zur verfügen welches Emotionserkennung in Echtzeit durchführen kann. Das System benötigt eine handelsübliche Kamera, wie sie heutzutage in jedem Laptop oder Smartphone zu finden ist.

Das SDK besitzt folgende Features:

- Das Erkennen und verfolgen von menschlichen Gesichtern in Bildern.
- Messen und verfolgen von emotionalen Gesichtsausdrücken in Echtzeit.
- Erfassen von 20 verschiedenen Gesichtsausdrücken in Echtzeit.
- Das erkennen von Attributen der physischen Erscheinung, wie das Geschlecht und ob eine Person eine Brille/Sonnenbrille trägt.
- Identifiziert 12 verschiedene Emoji, die aus den Gesichtsausdrücken hergeleitet werden.

Das System erkennt die Emotionen mit Hilfe eines Classifiers der mit 3.2 Millionen Gesichtern aus 75 verschiedenen Ländern trainiert wurde. Das SDK steht für sieben verschiedene Plattformen bereit, darunter Android, iOS, Linux, macOS, Unity, Web und Windows.

5.9.1. Affdex Unity SDK

Um mit dem Unity SDK von Affectiva arbeiten zu können, müssen drei Klassen in das Projekt übernommen werden:

Detector: Der Detector erfasst die Gesichtsausdrücke. Die Bilder müssen dem Detector Bild für Bild jeden Frame zugeliefert werden. Der Detector kann dann das Bild analysieren und das Ergebnis der Emotionserkennung zurückliefern. Der Detector bietet Konfigurationsmöglichkeiten für die Emotionserkennung und Gesichtsausdrücke. So lässt sich einstellen welche der zur Verfügung stehenden Emotionen und Gesichtsausdrücke erfasst werden sollen.

Camera Input: Um das Bild von der Webcam zu bekommen, bietet das SDK das Camera Input Skript. Alternativ kann auch ein eigenes Skript geschrieben werden. Für die Kamera selber werden Standardauflösungen im 4:3 Format von 320x240, bis 1024x768 empfohlen. Die empfohlene Sample Rate beträgt zwischen 5 und 20 Bildern pro Sekunde. Durch das Reduzieren der Sample Rate und Auflösung der Bildquelle kann die Prozessorauslastung des Affdex SDK auf die Zielplattform optimiert werden.

Image Result Listener: Über den Image Result Listener können die durch den Detector erzeugten Daten abgerufen werden. Hierbei handelt es sich um eine Abstrakte Klasse, für die folgende Funktionen vom Benutzer implementiert werden müssen:

onFaceFound(float timestamp, int faceId): Die Methode wird aufgerufen, sobald das Affdex SDK ein Gesicht erkannt hat. Parameter sind ein Unix Zeitstempel der den Zeitpunkt angibt, wann das Gesicht erkannt wurde. Der zweite Parameter ist ein Integer, der die ID für das Gesicht angibt. Über die ID werden die Daten für das entsprechende Gesicht indiziert.

onFaceLost(float timestamp, int faceId): Die Methode wird aufgerufen wenn das Affdex SDK ein bereits erkanntes Gesicht wieder verliert. Parameter sind ein Unix Zeitstempel der den Zeitpunkt angibt, wann das Gesicht verloren wurde. Der zweite Parameter ist ein Integer der die ID für das Gesicht angibt. Über die ID werden die Daten für das entsprechende Gesicht indiziert.

onImageResults(Dictionary<int, Face> faces): Die Methode „onImageResults“ wird jedesmal aufgerufen, wenn ein neues Ergebnis für ein erkanntes Gesicht bereitsteht. Auf die Daten kann über die ID eines Gesichts zugegriffen werden. Die Daten für ein einzelnes Gesicht sind in der „Face“ Struktur gespeichert. Die „Face“ Struktur enthält Ergebnisse über die erkannten Gesichtsausdrücke, Emotionen und Kopfmessungen und Merkmalspunkte.

Affdex Datenstruktur

Im Folgenden werden die von der Affdex API gelieferten Ergebnisstrukturen der Emotionserkennung genauer beschrieben und wie diese Ergebnisse zu interpretieren sind. Alle Werte beziehen sich immer auf das zuletzt analysierte Bild.

Gesichtsausdrücke: Die Gesichtsausdrücke werden als Zahlenwerte in die Datenstruktur 5.8 geschrieben. Jeder Wert in der Datenstruktur beschreibt die Wahrscheinlichkeit für den Gesichtsausdruck den dieser repräsentiert. Die Wahrscheinlichkeit besitzt einen

Wertebereich Zwischen 0 und 100, wobei 100 der maximalen Wahrscheinlichkeit von 100% entspricht.

```
struct Expressions
{
  float Smile;
  float InnerEyeBrowRaise;
  float BrowRaise;
  float BrowFurrow;
  float NoseWrinkler;
  float UpperLipRaiser;
  float LipCornerDepressor;
  float ChinRaiser;
  float LipPucker;
  float LipPress;
  float LipSuck;
  float MouthOpen;
  float Smirk;
  float EyeClosure;
  float Attention;
};
```

Listing 5.8: Affdex Expressions Struktur

Emotionen: Die Datenstruktur 5.9 repräsentiert die erkannten Emotionen. Jeder Wert in der Datenstruktur beschreibt die Wahrscheinlichkeit für den Gesichtsausdruck und die Emotion. Die Wahrscheinlichkeit besitzt einen Wertebereich Zwischen 0 und 100, wobei 100 der maximalen Wahrscheinlichkeit von 100% entspricht. Die Emotionen Joy, Fear, Disgust, Sadness, Anger, Surprise und Contempt zählen zu den Basisemotionen die durch Paul Ekman geprägt wurden Ekman (1992). Die Werte Valence (Wertigkeit) und Engagement (Beteiligung, Involvierung) gehören nicht direkt zu den Emotionen. Der Wertebereich von Valence stellt eine Ausnahme dar. Der Wert gibt an ob ein Gesichtsausdruck Positiv oder Negativ ausgerichtet ist. Die Valence hat einen Wertebereich von -100 bis 100, wobei positive Werte einen positiv ausgerichteten Ausdruck beschreiben und negative Werte einen negativ ausgerichteten Ausdruck beschreiben.

```
struct Emotions
{
  float Joy;
  float Fear;
```

```
5   float Disgust;
6   float Sadness;
7   float Anger;
8   float Surprise;
9   float Contempt;
10  float Valence;
11  float Engagement;
12 };
```

Listing 5.9: Affdex Emotions Struktur

Kopfmessungen: Die Datenstruktur „Measurements“ enthält zwei Eigenschafren: Die Orientierung des Kopfes und die Distanz zwischen den beiden äußeren Augenwinkeln. Die Orientierung wird durch die Eigenschaften „Pitch“, „Yaw“ und „Roll“ (Steigung, Winkel und Rotation), beschrieben.

Feature Points: Diese Struktur ist eine Liste von 34 Gesichtspunkten, die bei der Emotionserkennung genutzt werden. Ein Einzelner „Feature Point“ besitzt eine ID zur Identifizierung und X, Y-Werte die dessen Position im Bild angeben. Diese können beispielsweise dazu genutzt werden die Gesichtsausdrücke des Benutzers auf einen Avatar zu Projizieren.

5.9.2. Affdex als Sensor für den Emotionalen KI-Agenten

Der Affdex Sensor für den Emotionalen KI-Agenten implementiert das Interface für die Sensoren (siehe 5.5) und erbt von der in 5.9.1 beschriebenen abstrakten Klasse ImageResultListener. Der Sensor arbeitet unter der Annahme, dass eine Person nur eine Emotion innerhalb eines Frames haben kann. Es wird immer die Emotion mit der höchsten Wahrscheinlichkeit ausgewählt. Kommt es zu einer Kollision zwischen den Wahrscheinlichkeiten mehrerer Emotionen wird die mit der höchsten Priorität ausgewählt. Die Prioritäten der Emotionen werden durch den Benutzer festgelegt.

Die Aufgabe des Sensors kann wie folgt zusammengefasst werden:

1. Empfange die durch Affdex generierten Daten sobald diese verarbeitet werden können.
2. Filtere die Daten.
3. Ignoriere „Valence“ und „Engagement“, da diese keine konkrete Emotion darstellen.
4. Es werden nur Emotionen mit einer Wahrscheinlichkeit >= Threshold betrachtet. Threshold ist ein benutzerdefinierter Untergrenze, vorgegeben 60%.

5. Ermittle die Emotion mit der höchsten Wahrscheinlichkeit und Cache diesen Wert.
6. Wenn die Wahrscheinlichkeit von der aktuell höchsten Emotion und der aktuellen Emotion innerhalb eines vorgegebenen Bereichs liegen, prüfe Priorität der Emotionen.
7. Schreibe die Emotion mit der höheren Priorität oder fahre fort mit 8.
8. Schreibe die Emotion mit dem aktuell höchsten Wert in den Working Memory bei jedem Sensor Update.

Die Filterfunktion des Affdex Sensors als Pseudocode:

```
foreach (var emotion in face.Emotions){
 //Skip invalid emotions
 if (emotion == Valence || emotion == Engagement) continue;
 if (emotion.Value < EmotionThreshold) continue;
 _emotionResult.Add(emotion);
 var checkPrio = (
   Abs(_highestEmotion.Value - emotion.Value) <= PriorityThreshhold
 );
 if (checkPrio){
   var swap = CheckEmotionPriority(emotion);
   if (swap) // swap emotions based on Priorety{
     _highestEmotion = emotion;
     continue;
    }
 }
 if (_highestEmotion.Value < emotion.Value){
   _highestEmotion = emotion
 }
}
```

Listing 5.10: Affdex Sensor Filterfunktion

6. Game Design

Im Game Design werden die Konzepte des Spiels vorgestellt. Dies beinhaltet die Charaktere und deren Fähigkeiten sowie das Szenario in dem die Charaktere agieren und weitere Spielmechaniken.

Im zweiten Teil, der Implementierung werden die verwendeten Technologien diskutiert, mit denen das Spiel umgesetzt wurde sowie die Verwendung externer Bibliotheken, 3D Modelle, Animationen und Sounds. Des Weiteren werden wichtige Aspekte der Implementierung und die Umsetzung emotional adäquat reagierenden KI-Agenten besprochen.

6.1. Ziele des Game Design

Um ein realistisch umsetzbares Konzept zu entwickeln, werden folgende Bedingungen für das Spiel festgelegt:

- Für die Durchführung von Versuchen sollten nicht mehr als 30 Minuten veranschlagt werden. Dadurch haben die Testpersonen genügend Zeit sich mit dem Spiel zu befassen und Fragebögen vor und nach der Versuchsdurchführung zu beantworten.
- Die Spielzeit sollte nicht Länger als 10 Minuten sein. Das gibt dem Spieler die Zeit sich mit dem Spielprinzip vertraut zu machen. Der emotionale Aspekt des Spiels sollte dem Spieler innerhalb dieser Zeit vermittelt werden können. Der Aufwand für die Probanden sollte damit in einem realistischen Rahmen liegen.
- Der emotionale Agent sollte langlebig sein und Aspekte eines Companion beinhalten.
- Das Spielprinzip sollte leicht verständlich sein und dennoch anspruchsvoll genug, um den Spieler zu fordern.
- Erstellen eines Spielabschnittes, der die Spielmechaniken eines Gesamtspiels repräsentiert (Vertical Slice).
- Das Erstellen von Grafiken wie Texturen, 3D Modelle, Animationen und Audio liegen außerhalb der Absichten dieser Arbeit, weshalb auf extern erstellte Assets zugegriffen

wird. Diese sind kostenlos und die Verwendung der Assets in diesem Projekt verstößt nicht gegen die Lizenzbedingungen der Urheber.

6.2. Das Spielszenario

Das Spiel basiert auf dem etablierten Genre der Schleichspiele, welche maßgeblich durch die Metal Gear Solid Spielserie beeinflusst wurden, die bis in die 90er Jahre zurückreicht (siehe Konami (1999)). Somit bietet das hier vorgestellte Szenario ein bereits bewiesenes Spielprinzip, dessen Spielmechaniken von einem Großteil der Spieler positiv angenommen wurde.

Der Spieler begibt sich in die Rolle eines Geheimagenten. Sein Auftrag ist es eine gegnerische Anlage zu infiltrieren, zur Seite steht ihn sein treuer Begleiter. Zusammen müssen sie durch die Anlage navigieren, um ins obere Stockwerk, die Kommandozentrale, zu gelangen. Auf den Weg dahin müssen sie Sicherheitsanlagen und Wachen überlisten oder diese ausschalten. Die Wahl des Vorgehens ist dem Spieler überlassen. Die Fähigkeiten des Spielers und seines Begleiters ergänzen sich, sodass die Operation nur zum Erfolg geführt werden kann, wenn beide zusammenarbeiten.

6.3. Die Charaktere und interaktive Objekte

6.3.1. Die Wache

Die Wache ist der Hauptkontrahent des Spielers und seines Begleiters. Die Wachen sind mit Betäubungsgewähren bewaffnet und patrouillieren in der Umgebung. Es ist beabsichtigt, dass die Wache bei dem Spieler negative Emotionen auslösen könnte. Jedoch ist nicht garantiert, dass die Spieler emotional reagieren. Die Emotionen sind stark von der Persönlichkeit des Spielers abhängig. Anhand der Vorversuche (siehe 3) hat sich gezeigt, dass einige Personen weniger oder keine der angestrebten Emotionen ausdrücken. Diese Personen zeigten oft einen Gesichtsausdruck, der als konzentriert beschrieben werden kann. Des Weiteren kann nicht vorhergesagt werden, ob die angestrebte Emotion mit der tatsächlich empfundenen Emotionen übereinstimmt. Jede Wache hat eine feste Patrouille, die sie abläuft. Sie beschützen wichtige Gegenstände, die der Spieler benötigt, um voranzukommen oder blockieren dem Weg zum Ziel. Sie besitzen zwei Sensoren: Einen Sichtsensor der es ihnen ermöglicht den Spieler zu wahrzunehmen, wenn sich dieser in deren Sichtfeld aufhält und einen Geräuschsensor der es ihnen ermöglicht die Bewegungsgeräusche des Spielers zu registrieren. Wenn sie den Spieler gesichtet haben, lösen die Wachen Alarm aus, was alle anderen Wachen alarmiert. Verliert die Wache Sichtkontakt, bewegt sie sich zu der letzten bekannten Position, an dem sie den Spieler

gesehen hat. Nach Ablauf einer bestimmten Zeit erlischt der Alarm, wodurch alle Wachen wieder anfangen zu patrouillieren. Befindet sich der Spieler im Sichtfeld, wird er angegriffen und nach wenigen Treffern betäubt. Der Spieler wacht dann am Anfang des Levels wieder auf und der Fortschritt geht verloren.

6.3.2. Der Spieler

Der Spieler hat die Fähigkeit zu schleichen um, sich unbemerkt an Wachen vorbei zu bewegen. Er kann Schalter drücken, um Türen zu öffnen oder Alarmsysteme zu deaktivieren. Der Spieler verfügt über keine offensiven Fertigkeiten, um Wachen auszuschalten. Hierfür ist er auf seinen Begleiter angewiesen. Er kann seinem Begleiter Befehle geben, die dieser dann ausführt. Diese beschränken sich auf die einfachen Kommandos „Folgen“, „Stop“ und „Helfen“.

6.3.3. Der Begleiter

Der Begleiter ähnelt in seinem Verhalten dem eines Hundes. Er hört auf die einfachen Kommandos des Spielers. Seine Aufgabe ist es positive Emotionen bei dem Spieler auszulösen und ihm als hilfreicher Begleiter zur Seite zu stehen.

Für die Durchführung von Experimenten werden zwei Varianten des Begleiters implementiert. Die erste Variante ist ein Begleiter, der auf die Emotionen des Spielers reagiert. Die zweite Variante implementiert einen rational agierenden Begleiter. Der rationale Agent kann für die Kontrollgruppe bei Versuchsdurchführungen verwendet werden. Der rationale Agent reagiert nicht auf Emotionen und wird immer auf die Anweisungen des Spielers reagieren und diesen folgen. Er folgt somit den Anforderungen (vgl. 2.3), die an einen Companion gestellt werden strikter als die erste Variante.

Das Verhalten des Begleiters wird in zwei Ebenen unterteilt: Einer Sachebene und einer emotionalen Ebene. Die Sachebene beinhaltet alle Aktionen, die nicht durch Emotionen beeinflusst werden. Die emotionale Ebene beinhaltet alle Aktionen des Begleiters, die durch die Emotionen des Spielers beeinflusst werden können.

Sachebene

Bei der Sachebene ist zu beachten, dass die Aktionen selber keine emotionale Komponente enthalten, jedoch durch die emotionale Ebene gesteuert werden können.

- „Stop“: Der Begleiter bleibt stehen. Der Spieler kann sich entscheiden den Begleiter zurückzulassen, wenn er der Meinung ist, dass er nicht auf ihn angewiesen ist.

- „Folgen“: Der Begleiter folgt dem Spieler.
- „Helfen“: Der Begleiter hilft dem Spieler. Er hat die Möglichkeit Wachen auszuschalten und dem Spieler dabei zu helfen Orte zu erreichen, die dem Spieler unzugänglich sind. Zum Beispiel beim Deaktivieren von Laserschranken oder beim Entfernen von Hindernissen, die den Weg versperren.

Emotionale Ebene

Bei der Entscheidungsfindung wurde darauf geachtet, den Einfluss, den die Emotionen des Spielers haben, auf einige wenige Aspekte zu fokussieren. Bei der Entscheidungsfindung wurde darauf geachtet, den Einfluss, den die Emotionen des Spielers haben, auf einige wenige Aspekte zu fokussieren. Es wurde darauf geachtet, dass sich die Spielelemente, die sich durch Emotionen beeinflussen lassen, natürlich anfühlen und in einem sinnvollen Kontext sind (vgl. 2.2.2). Sie sollten nicht das Kernspiel bestimmen, sondern die klassischen Spielmechaniken ergänzen. Ein exzessiver Einsatz der Emotionserkennung könnte einen negativen Effekt auf die Wahrnehmung des Spielers haben und sich erzwungen oder lästig anfühlen.

Der emotionale Agent hat die Fähigkeit den Kommandos des Spielers zu folgen, eine andere Aktion auswählen oder das Kommando zu ignorieren.

Vertrauen: Der Begleiter besitzt einen numerischen Wert, der das Vertrauen zu dem Spieler widerspiegelt. Dieser Wert verändert sich abhängig von den gezeigten Emotionen des Spielers. Die Emotionen werden mit einem Application Specific Clustering (siehe 3.3) kategorisiert, um die Erkennungsrate zu erhöhen. Die Emotionen Joy und Surprise werden als positive Emotionen kategorisiert. Als negative Emotionen werden Fear, Anger, Disgust und Sadness zusammengefasst. Der Wert, durch den das Vertrauen repräsentiert wird, erhöht sich, wenn der Begleiter eine positive Emotion über seinen Emotionssensor wahrnimmt und verringert, wenn er eine negative Emotion wahrnimmt. Der Wert besitzt ein konstantes Minimum (min = 0) und ein Maximum, welches nicht überschritten werden kann. Das Maximum ist konfigurierbar, da eine geeignete Einstellung in Abhängigkeit von der Spiellänge bestimmt werden muss. Ist der Wert zu groß, wird der Spieler keine Veränderung feststellen, ist das Maximum zu niedrig angesetzt, wird das Verhalten zu sprunghaft und dadurch störend. Die Auswirkungen von erkannten Emotionen auf den Wert sind konfigurierbar, sodass durch experimentelles Vorgehen ein geeigneter Faktor gefunden werden kann. Betrachtet man die Art der Interaktion, handelt es sich bei dem Vertrauen nicht um eine sofortige Reaktion. Der Wert ist als lang-

fristige, akkumulierte Reaktion konzipiert, wobei die Langfristigkeit immer in Relation zur Versuchsdauer steht und abhängig von dieser konfiguriert wird.

Das Vertrauen wirkt auf die Entscheidungsfindung des Begleiters. Sinkt der Wert unter eine bestimmte Schwelle, hört der Begleiter nicht mehr auf den Spieler. Der Spieler kann dann nur noch durch positive Emotionen das Vertrauen wieder erhöhen. Im nicht emotionalen Modus ist der Wert konstant, sodass der Begleiter zu jeder Zeit auf die Befehle des Spielers reagiert. Optisch wird der Wert durch die dreistufige Farbcodierung eines grafischen Symbols über dem Charakter dargestellt. Ein rotes Symbol signalisiert ein geringes Vertrauen, der neutrale Zustand wird durch die Farbe Gelb dargestellt und ein grünes Symbol bedeutet absolutes Vertrauen.

Emotionales Bestärken: Das Bestärken ist eine weitere Variante, mit der der Spieler emotionalen Einfluss auf die Entscheidungsfindung des Begleiters nehmen kann. Das emotionale Bestärken wird aktiv durch den Spieler ausgelöst, indem er die bestärken Taste drückt. Dies führt zu einer unmittelbaren Reaktion durch den Begleiter. Nachdem die Taste gedrückt wurde, werden die durch den Spieler gezeigten Emotionen akkumuliert. Die positiven und negativen Emotionen werden unabhängig voneinander gezählt. Wird die Obergrenze, an gezeigten positiven Emotionen erreicht, wird ein positives Bestärken durch den Agenten erkannt. Wird der Schwellwert an negativen Emotionen erreicht, wird ein negatives Bestärken wahrgenommen. Innerhalb jedes Frames wird bestimmt, ob es sich bei einer gezeigten Emotion um eine positive oder negative Emotion handelt und die zugehörige Variable inkrementiert. Kann keine Emotion eindeutig erkannt werden, wird keine der Variablen hochgezählt. Dieser Vorgang ist zeitlich begrenzt mit tMax = 5s, das Zeitfenster ist abhängig von den Obergrenzen für gezeigte negative und positive Emotionen sowie der Framerate der Kamera. Liefert die Kamera 20 Bilder pro Sekunde, können maximal 20 positive oder negative Emotionen pro Sekunde erkannt werden.

Bestärken ausgewählter Aktionen: Der Begleiter merkt sich die letzte Aktion, die er aufgrund eines Befehls des Spielers ausgeführt hat. Der Spieler hat nun die Möglichkeit den Begleiter zu bestärken oder zu tadeln je nachdem wie zufrieden er mit der Aktion des Begleiters war.

Abhängig von einer Wahrscheinlichkeitstabelle wählt der Begleiter eine Aktion basierend auf dem vom Spieler zuvor gegebenen Kommando aus. In dieser Wahrscheinlichkeitstabelle befindet sich die korrekte Aktion sowie 1 bis N im Kontext nicht sinnvollen Aktionen. Für die nicht sinnvollen Aktionen werden Animationen verwendet. Beispielsweise weißt der Spieler den Begleiter an zu folgen, dieser winkt dem Spieler stattdessen

zu. Die korrekte Aktion besitzt die höchste Wahrscheinlichkeit. Mit einer geringerer Chance kann jedoch auch eine der anderen Aktionen ausgeführt werden.

Durch das Bestärken kann die Wahrscheinlichkeit der ausgewählten Aktion erhöht oder verringert werden, sodass diese seltener oder häufiger vom Begleiter gewählt wird. Wenn ein positives Bestärken erkannt wurde, wird die Wahrscheinlichkeit dieser Aktion erhöht, ein negatives Bestärken verringert den Wert. Der Grundwert und die dazugehörige Aktion können konfiguriert werden. Über die gewichteten Wahrscheinlichkeiten (siehe 6.1) wird die nächste Aktion berechnet.

```
var total = SumProbability(Items);
var randomPoint = Random.value * total;
foreach (var item in Items){
 if (randomPoint < item.getWeight())
   return item;
 randomPoint -= item.getWeight();
}
return Items[Items.Count -1];
```

Listing 6.1: Selektion der nächsten Aktion mittels gewichteter Wahrscheinlichkeit

Durch dieses Verhalten wird dem Begleiter eine gewisse Sturheit verliehen. Ziel ist es, das der Begleiter dadurch den Eindruck eines eigenen Willens vermitteln kann. Es sollen sowohl negative Emotionen provoziert werden, sollte der Begleiter nicht auf den Spieler hören, als auch positive Emotionen, wenn der Begleiter die Kommandos des Spielers befolgt. Im nicht emotionalen Modus wählt der Begleiter immer die passende Aktion aus.

Emotional abhängige Aktionen: Einige Passagen des Levels sind durch Kisten blockiert und unpassierbar für den Spieler. Der Begleiter kann diese Kisten aus dem Weg räumen, indem er diese zerstört. Der Spieler muss dem Begleiter befehlen die Kisten zu zerstören. Nachdem der Befehl gegeben wurde, wartet der Begleiter darauf, dass der Spieler ihn entsprechend motiviert. Über der Passage wird durch ein Emoticon angezeigt, welche Emotion der Spieler nachahmen muss. Hierfür muss der Spieler die Bestärken-Taste drücken. Wurde die Emotion erfolgreich erkannt, räumt der Begleiter das Hindernis aus dem Weg. Wurde die korrekte Emotion nicht erkannt, hat der Spieler die Möglichkeit den Vorgang zu wiederholen. Im nicht emotionalen Modus wird das Hindernis sofort nach Absetzen des Befehls entfernt.

6.3.4. Weitere interaktive Objekte

Security System: Zusätzlich zu den Wachen existieren Überwachungskameras. Diese Melden ihre Sichtungen an die Wachen, indem sie Alarm auslösen. Sie besitzen nur einen Sichtsensor und können nicht schießen. Laserzäune blockieren den Weg des Spielers. Sie töten den Spieler sofort, wenn sie durchquert werden. Die Intention ist es, so Spannung zu erzeugen und Emotionen des Spielers auszulösen. Wird der Spieler vom Überwachungssystem erfasst, ertönt eine Sirene und Warnleuchten werden aktiviert. Die Wachen werden auf den Spieler aufmerksam und bekämpfen ihn. Dadurch könnten negative Emotionen ausgelöst werden. Überwindet der Spieler das Hindernis, sollte dies positive Emotionen auslösen.

Schalter und Druckplatten: Durch Schalter und Druckplatten lassen sich Türen öffnen und Laserzäune deaktivieren. Druckplatten müssen dauerhaft belegt werden, wodurch entweder der Spieler oder der Begleiter auf der Druckplatte stehen muss, damit diese aktiv bleibt.

Schlüsselkarte: Schlüsselkarten können vom Spieler eingesammelt werden. Manche Türen lassen sich nur durch eine Schlüsselkarte öffnen.

Lebenskugeln: Hat der Spieler Lebenspunkte verloren, können diese durch das Einsammeln der Lebenskugel wiederhergestellt werden.

6.3.5. Protokollierung des Spielverlaufs

Während eines Spieldurchlaufes fallen eine große Menge verschiedenster Daten an, die für eine Analyse gespeichert und ausgewertet werden können. Die Daten müssen in einen zeitlichen und kausalen Zusammenhang miteinander gebracht werden. Die gesamte Architektur wird auf dem gleichen System ausgeführt, wodurch die Zeiten aller Datenbestände synchron sind. Die Zeit wird von Beginn bis zum Ende der Versuchsdurchführung gemessen. Wobei der Start der Simulation Zeitpunkt 0 ist. Es können alle relevanten Daten des Spiels und der Emotionserkennung gespeichert werden. Theoretisch kann mit Hilfe dieser Daten die gesamte Simulation aufgezeichnet und wiedergegeben werden. Wichtige Daten für die Analyse der Versuche werden im Folgenden aufgelistet.

Emotionserkennung: Die Daten aus der Emotionserkennung werden wie folgt im CSV-Format abgespeichert (siehe 6.2).

```
1 Time;Joy;Fear;Disgust;Sadness;Anger;Surprise
```

```
2 ;Contempt;Valence;Engagement;Cluster
```

Listing 6.2: Emotionserkennung Log

Wobei „Time“ die gemessene Zeit seit Start der Simulation ist und „Cluster“ angibt, ob es sich um eine positive, negative oder neutrale Emotion handelt.

Vertrauenswert: Hier wird neben einem Zeitstempel die Höhe des Wertes gespeichert. So kann der zeitliche Verlauf des Vertrauenswertes und seine Relation zu den erkannten Emotionen analysiert werden.

Entscheidungsfindung: Die Protokollierung der Entscheidungen, die der Begleiter getroffen hat. Wichtig sind hier Entscheidungen, die auf Basis des emotionalen Zustands des Spielers getroffen wurden. Für den Prototyp ist das die Entscheidung, ob eine Aktion aufgrund des Vertrauens ausgewählt wurde oder nicht, sowie das Bestärken einer Aktion durch den Spieler und emotional abhängige Aktionen.

6.3.6. Zusammenfassung

Das in diesem Abschnitt beschriebene Game Design beschreibt ein in sich kohärentes Spiel, welches dem Genre der Schleichspiele oder Action Adventure zugeordnet werden kann.

Das Spiel beinhaltet sowohl emotionale als auch nicht emotionale Interaktionen, die Einfluss auf Entscheidungsfindung des KI-Agenten haben. Die emotionale Interaktion ist eindeutig von der nicht emotionalen Interaktion unterscheidbar. Der emotionale Aspekt der Entscheidungsfindung kann bei Bedarf aktiviert und deaktiviert werden, damit Versuche durch eine Kontrollgruppe möglich sind. Es werden keine Features entfernt, lediglich die Art wie diese ausgelöst werden, wodurch die Features gut vergleichbar bleiben. Zusätzlich ist es möglich, nur einzelne emotionale Features zu aktivieren, damit diese gesondert untersuchen werden können.

Es wird angenommen, dass ein Prototyp mit einer Spiellänge von ca. 10 Minuten lang genug ist um die emotionalen Reaktionen des Spielers und das Zusammenspiel mit dem Begleiter untersuchen zu können. Die Spiellänge kann bei Bedarf verlängert werden, indem neue Level hinzugefügt werden oder die Schwierigkeit erhöht wird. Um die Schwierigkeit zu erhöhen, kann beispielsweise die Anzahl an Wachen angepasst oder die Lebenspunkte des Spielers verringert werden.

Die Daten der Emotionserkennung können verwendet werden, um die Emotionen über den gesamten Spielverlauf zu analysieren. Zusätzlich kann untersucht werden, wie sich das Clustering in positive und negative Emotionen auf die Erkennungsrate auswirkt. Über die

Entscheidungen des Agenten und Fragebögen von Probanden kann untersucht werden, wie die im Game Design angedachten emotionalen Interaktionen auf die Wahrnehmung des Spiels und die Immersion des Spielers auswirken.

Zu beachten ist, dass neben der Spielmechanik auch Sound und Grafik Einfluss auf die Immersion haben. Ein optisch einheitliches und grafisch aufwendiges Spiel mit qualitativ hochwertigem Sound und Musik ist jedoch zeitlich, finanziell und im Rahmen dieser Arbeit nicht umsetzbar.

6.4. Implementierung des Prototyps

In diesem Kapitel wird die Implementation des Prototyps beschrieben. Dieser setzt das zuvor in Kapitel 6 beschriebene Game Design um. Als Erstes werden die verwendeten Technologien und Ressourcen beschrieben und anschließend die wichtigsten Aspekte der Implementation diskutiert.

6.4.1. Game Engine

Für die Implementation des Prototyps wurde die Unity Engine verwendet, welche bereits im EmoBike Projekt verwendet wurde (vgl. 3). Benutzt wurde die Personal Edition (vgl. Unity (2017d)) der Engine, welche für kleinere kommerzielle und nicht kommerzielle Projekte frei verfügbar ist. Bis auf einige wenige Funktionen, die für die Umsetzung des Prototyps nicht relevant sind, bietet die Personal Edition den vollen Funktionsumfang einer Professionellen Game Engine.

Funktionsweise und Kernkonzepte

Szenen Aufbau: Grundlegend wird ein mit Unity erstelltes Spiel in Szenen unterteilt. Eine Szene beinhaltet alle Modelle, Sounds und Skripte, die das Spiel ausmachen. Es können 1 bis N Szenen erstellt werden, welche auch oft als Level bezeichnet werden. Eine Szene ist in einer Baumstruktur organisiert (Abb. 6.1), jeder Knoten in dieser Struktur muss ein „GameObject“ sein.

GameObject: Diese Klasse ist der grundlegende Container für jede Funktion in einer Szene und kann 0 bis N Kinder besitzen, wobei jedes Kind auch ein „GameObject“ ist. Das „GameObject“ besitzt 1 bis N „Components“ (Abb. 6.2), welche die Funktionalitäten des Objektes bereitstellen. Es ist festgelegt, dass ein Objekt mindestens die „Transform“ Komponente besitzt, diese gibt die Position, Skalierung und Rotation innerhalb der Szene

Abbildung 6.1.: Unity3d Scene Hierarchy

an. Beispiele für weitere Standardkomponenten sind der „Renderer“, „Collider“ und „Animator“, wobei der „Renderer“ für das Rendern eines 3D Modells verantwortlich ist. Über den „Collider“ werden die Maße und Formen eines 3D Modells für die Physik Engine bereitgestellt. Über den „Animator“ können, falls vorhanden, Animationen angesteuert werden. Weitere Standardkomponenten sowie benutzerdefinierte Komponenten können einem „GameObject“ zugewiesen werden.

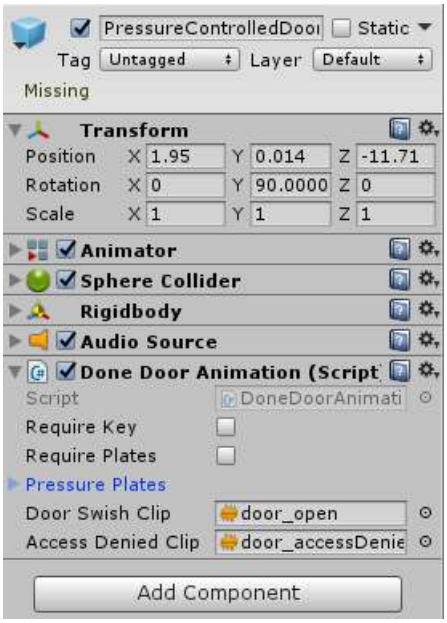

Abbildung 6.2.: Unity3d GameObject Components

Funktionsname	Beschreibung
Awake	Wird nach dem Instanziieren eines „GameObjects“ aufgerufen. Diese Funktion dient zur Initialisierung. Sie wird einmalig während des Lebenszyklus des Objekts aufgerufen.
Start	Diese Funktion dient ebenfalls der Initialisierung des „MonoBehaviors“, sie wird jedoch nach der „Awake“ Funktion aufgerufen und jedes Mal wenn das Objekt aktiviert wird. Sie kann genutzt werden, wenn Referenzen auf anderen „MonoBehaviors“ benötigt werden, welche bereits initialisiert sein müssen.
Update	Diese Funktion wird einmal pro Ausführungszyklus aufgerufen. In dieser Funktion wird der Großteil der Spiellogik ausgeführt.
OnCollisionEnterStayExit	Diese Gruppe von Funktionen werden nach einer Kollisionserkennung ausgeführt. Hierfür benötigt das „GameObject“ die Komponenten „Rigidbody“ und „Collider“. Die Komponente „Rigidbody“ wird von der Physik Engine benötigt und gibt an das es sich bei einem Objekt um ein Physik Objekt handelt. Ein Physik Objekt kann durch die Physik Engine in seiner Position und Rotation manipuliert werden, sowie Kollisionsereignisse empfangen. Diese Ereignisse werden asynchron von der Physik Engine ausgelöst.
OnTriggerEnterStayExit	Ein „Collider“ kann als „Trigger“ konfiguriert werden. So werden Kollisionen anderer Objekte mit dem „Collider“ zwar registriert, jedoch werden keine Manipulation der Objekte durch die Physik Engine vorgenommen. Dies ist hilfreich, um festzustellen, ob ein Objekt einen bestimmten Bereich innerhalb der Spielwelt erreicht oder passiert hat.

Tabelle 6.1.: MonoBehavior wichtige Standardfunktionen

MonoScript: Das „MonoScript“ ist eine Möglichkeit eine benutzerdefinierte Komponente zu erstellen. Bis auf wenige Ausnahmen werden alle benutzerdefinierten Komponenten über „MonoScripts“ erstellt. Programmiert wird das „MonoScripts“ in Mono, einer quelloffenen Version der Programmiersprache C#. Um aus dem Skript eine Komponente zu machen die von Unity verwendet werden kann, muss die Klasse von „MonoBehavior“ erben.

Das „MonoBehavior“ stellt eine Reihe von Funktionen zur Verfügung, die nach jedem Aktualisierungsschritt der Engine ausgeführt werden. Es ist dem Programmierer überlassen, welche dieser Funktionen er implementiert. Die wichtigsten Funktionen sind in der Tabelle 6.1 dargestellt. Eine ausführliche Beschreibung über den Ausführungszyklus der Engine ist in der Dokumentation (siehe Unity (2017a)) beschrieben.

Es steht dem Entwickler frei, innerhalb eines „MonoScripts“ Bibliotheken, andere Klassen oder „MonoScrips“ zu referenzieren und zu verwenden. Damit der eigene Code ausgeführt werden kann, muss mindestens ein „MonoBehavior“ existieren, welches einem „GameObject“ zugewiesen wurde und eine der Standardfunktionen implementiert. Ein „MonoScript“ kann 1 bis N „GameObjects“ zugewiesen werden. Nach dem Start erstellt das „GameObject“ eine eigene Instanz der Klasse, die durch das „MonoScript“ beschriebenen wird.

Animationssystem: Um Animationen in Unity abspielen zu können, wird für jedes Modell, welches Animationen benötigt, ein Animation Controller erstellt (Abb. 6.3). Der Animation Controller steuert die Animationen über einen Zustandsautomaten (siehe 4.3.1). Der Controller ist Teil der Unity Engine und wird über einen grafischen Editor erstellt. Über Parameter können die Transitionen des Zustandsautomaten im Code angesteuert werden. Mögliche Parameter sind „Integer“, „Bool“, „Float“ und „Trigger“. Der Parameter „Trigger“ ist ein boolescher Wert, der sich nach einer Transition automatisch zurücksetzt. Ein Zustand besteht entweder aus einer Animation, die beim Erreichen des Zustandes abgespielt wird oder aus einem Subautomaten.

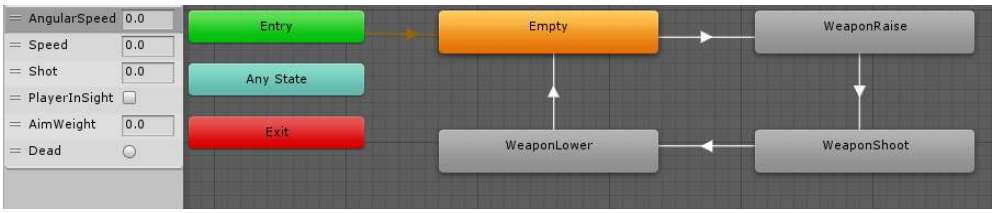

Abbildung 6.3.: Unity3d Animation Controller

Navigationssystem: Damit KI-Agenten intelligent durch die Spielwelt navigieren können, benötigen sie Wissen über die Umgebung und begehbaren Bereiche. Zu diesem Zweck stellt Unity ein Navigationssystem zur Verfügung. Das Navigationssystem besteht aus vier Komponenten:

NavMesh: Ist eine Datenstruktur, welche die begehbaren Bereiche der Spielwelt speichert. Das „NavMesh“ ist eine Schicht, die auf die Geometrie von begehbaren Objekten gelegt wird. Diese wird durch einen blauen Bereich dargestellt (Abb. 6.4). Die Datenstruktur besteht aus Polygonen. Alle Objekte, die Teil der Navigation sein sollen, werden über den Editor als statische Objekte markiert. Wenn das „NavMesh“ erstellt wird, werden die Oberflächen nach vorkonfigurierten Kriterien geprüft, um festzustellen, welche Bereiche sich als begehbar eignen. Unter anderem werden der

Radius und die Höhe des Agenten sowie die Steigung der Oberfläche berücksichtigt. Optional kann eine Liste von „Navigation Areas“ erstellt werden. Eine „Area“ kann einer oder mehreren Oberfläche zugewiesen werden, um diese mit Kosten zu belegen. Die Kosten der „Area“ wird bei der Wegfindung berücksichtigt.

NavMesh Obstacle: Ist für dynamische Objekte angedacht, die einen begehbaren Bereich versperren. Der versperrte Bereich verändert das „NavMesh“ zur Laufzeit. Beispielsweise ist eine Tür, die sich öffnen und schließen lässt, ein „NavMesh Obstacle“. Ist die Tür geschlossen, muss das „NavMesh“ angepasst werden, da dieser Bereich nicht mehr begehbar ist. Ansonsten würde ein Agent einfach durch die Tür laufen. Andersherum muss das „NavMesh“ angepasst werden, wenn sich die Tür öffnet, damit der Agent nicht einer offenen Tür ausweicht oder sein Ziel nicht erreichen kann.

Off-Mesh Link: Eine Verbindung zwischen zwei Punkten auf einem „NavMesh“ oder zwischen mehreren voneinander getrennten „NavMeshes“. Wenn zwei Punkte auf einem „NavMesh“ nicht direkt miteinander verbunden sind, kann über einen „Off-Mesh Link“ eine Verbindung erstellt werden. Beispielsweise kann eine Grube überquert werden indem sich der Agent zwischen zwei „Off-Mesh Links“ bewegt. Erreicht der Agent einen „Off-Mesh Link“, kann während der Transition zum gegenüberliegenden Punk eine Sprung Animation abgespielt werden.

NavMesh Agent: Berechnet über das „NavMesh“ den Pfad von der aktuellen Position zu einer Zielposition. Der Pfad wird mittels A* Algorithmus berechnet. Der Agent hat die Fähigkeit anderen Agenten und „NavMesh Obstacles“ auszuweichen. Er kann die Position eines Objektes automatisch verändern oder über Code gesteuert werden.

6.4.2. Verwendete Ressourcen

Die für den Prototypen verwendeten 3D Modelle, Partikel Systeme, Texturen, Animationen und Sounds wurden über den Unity Asset Store erworben (siehe Unity (2017c)). Der Asset Store ist eine von Unity Technologies zur Verfügung gestellte Onlineplattform, auf der Entwickler alle möglichen Ressourcen, die zum Erstellen von Spielen benötigt werden, anbieten. Diese sind entweder kostenlos oder können käuflich erworben werden. Für den Prototypen wurden ausschließlich kostenlose Ressourcen verwendet. Des Weiteren sind einige der Audiodateien über Freesound (2017) erworben, eine Onlineplattform über die kostenlos Audiodateien vertrieben werden.

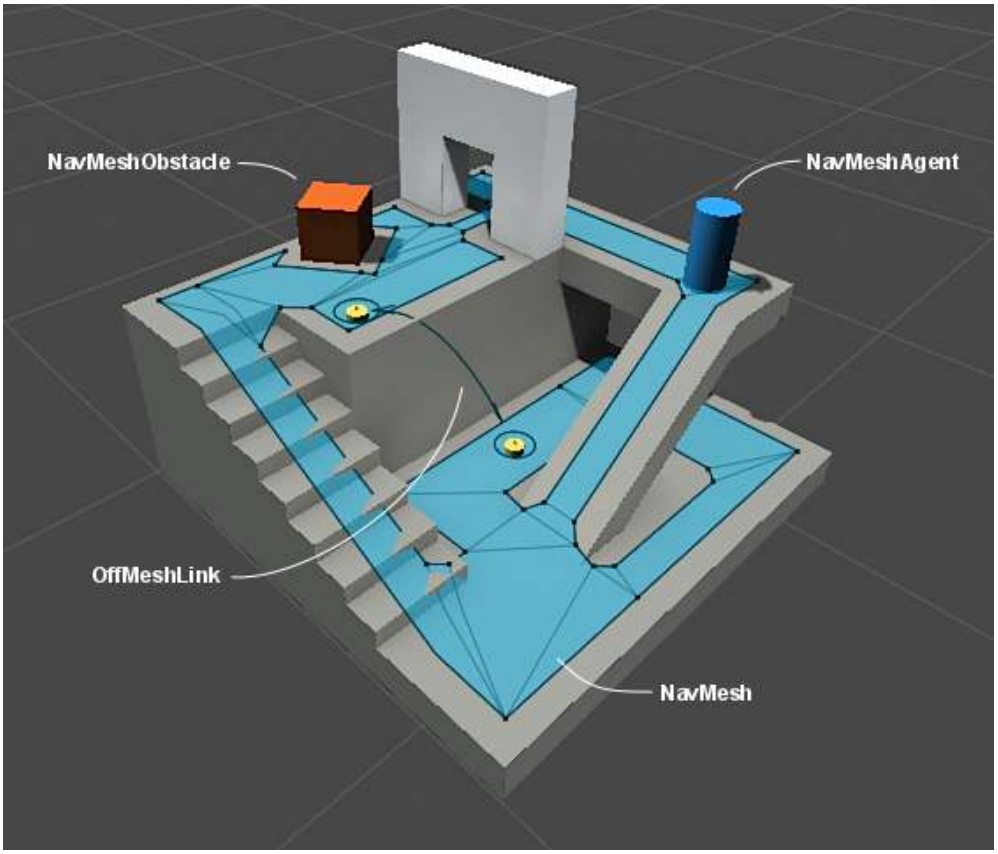

Abbildung 6.4.: NavMesh Übersicht. (Quelle: Unity (2017b))

6.4.3. Steuerung

Hauptfigur: Der Spieler übernimmt die Rollen von Ethan (Abb. 6.5) welcher sich über einen Xbox360 Controller steuern lässt. Der Spieler kann sich in X und Z Richtung frei bewegen. Mit dem linken Analogstick wird die Rotation und Geschwindigkeit des Charakters angepasst. Durch Gedrückthalten der Schultertaste LB wird das Schleichen aktiviert. In diesem Modus kann Ethan nicht von den Wachen gehört werden. Befindet man sich in der Nähe eines Schalters, wird dieser durch drücken von LB ausgelöst. Mit der A-Taste kann der Spieler die Aufmerksamkeit auf sich ziehen. Nach Drücken der Taste wird eine Audiodatei und eine entsprechende Animation abgespielt.

Abbildung 6.5.: Ethan Model

Befehle an den Begleiter: Die Befehle an den Begleiter werden wie folgt über den Controller ausgelöst:

- X-Taste: Folgen
- B-Taste: Stop
- Y-Taste: Helfen
- RB-Taste: Bestärken

User Interface: Das User Interface (Abb. 6.6) wurde so simpel wie möglich zu gestaltet, um den Fokus nicht vom Spiel abzulenken. Die Lebenspunkte des Spielers werden oben in der Ecke durch eine grüne Füllleiste (1) angezeigt. Wird der Spieler getroffen, wird die Füllleiste entsprechend angepasst und der fehlende Wert durch einen rot eingefärbten Abschnitt der Leiste angezeigt, wobei eine komplett grün ausgefüllte Leiste 100 % Lebenspunkte darstellt.

Wird die Bestärken Taste gedrückt, zeigt ein Text und ein Emoticon die aktuell erkannte Emotion an (2). Sowohl Text als auch Emoticon sind nur sichtbar, solange das Bestärken aktiv ist. Über eine Textbox, unten links (3) werden dem Spieler zu jedem Zeitpunkt die Steuerung des Charakters und Begleiters angezeigt.

Das Spiel wird aus der Vogelperspektive gespielt. Die Kamera folgt dem Spieler (4) automatisch, so das der Spielercharakter immer im Zentrum vom Bild ist. Der Begleiter (5) hat keinen Einfluss auf die Kameraperspektive. Das Kamerabild des Spielers wird oben rechts (6) angezeigt. Dies hilft festzustellen, ob sich der Spieler im Fokus der Kamera zur Emotionserkennung befindet. Wenn der Spieler Alarm ausgelöst hat und sich nicht mehr im Sichtfeld einer Wache oder einer Kamera befindet, wird ein Timer (7) eingeblendet der 5 Sekunden herunterzählt. Danach ist der Alarm beendet und der Timer wird ausgeblendet.

6.4.4. Charaktere und interaktive Objekte

Überwachungskameras: Eine Überwachungskameras ist fest positioniert. Über eine Animation schwenkt die Kamera von rechts nach links und wieder zum Ausgangspunkt zurück. Die Kamera besitzt ein „Spotlight," ein Licht, das in einem Kegel vom Ursprung aus strahlt. Dadurch wird der Bereich, den die Kamera erfasst, für den Spieler visualisiert. Wo das Licht auf ein Objekt trifft, wird eine rote Fläche sichtbar. Zusätzlich wird über ein „Light Cookie", eine Textur, ein Muster an diesen Stellen auf den Boden und die Objekte gezeichnet (Abb. 6.7). Über einen Sichtkegel, welcher durch einen „Collider" definiert ist, wird festgestellt, ob der Spieler den Sichtbereich betritt oder verlässt. Der „Collider" ist in dem Bild 6.7 durch die grünen Linien gekennzeichnet. Betritt der Spieler den Bereich und ist nicht durch andere Objekte in der Spielwelt verdeckt, wird der Alarm ausgelöst. Über die Physik Engine wird mittels eines „Raycast" geprüft, ob der Spieler durch andere Objekte verdeckt wird. Ein „Raycast" ist eine Gerade zwischen zwei Punkten. Das Ergebnis des „Raycasts" ist das erste Objekt, welches diese Gerade schneidet.

Abbildung 6.6.: User Interface und Spielerperspektive

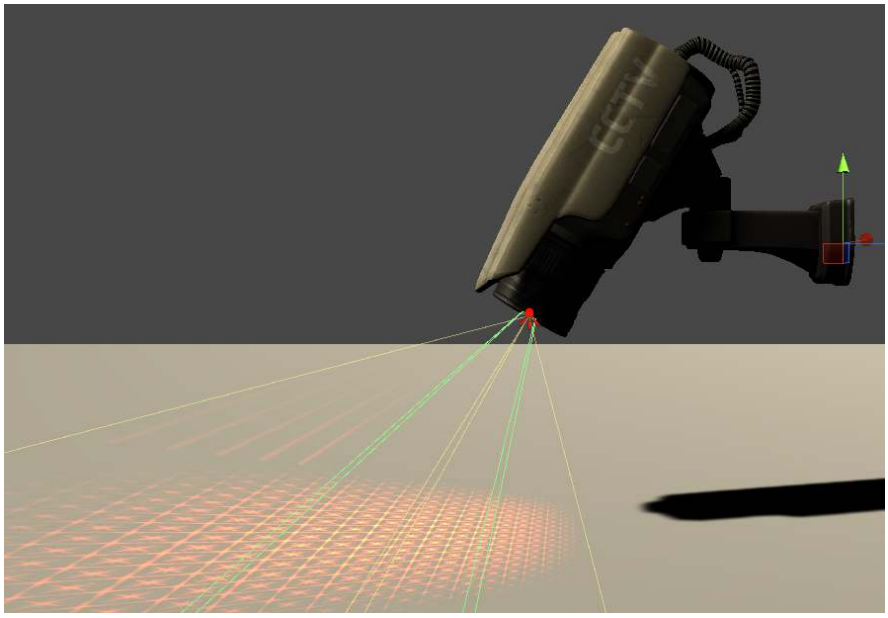

Abbildung 6.7.: Überwachungskamera Model

Laserzaun: Der Laserzaun (Abb. 6.8) detektiert den Spieler über einen „Collider“. Im Gegensatz zu der Überwachungskamera wird kein Alarm ausgelöst. Sobald der Spieler den Zaun passiert, wird dieser auf der Stelle getötet. An der Position des Spielers wird ein Partikel System instanziiert, um den Kontakt mit dem Laserzaun zu visualisieren. Des Weiteren wird ein elektrisches Geräusch und eine Sterbeanimation für den Spieler abgespielt. Zusätzlich ist ein optionales Verhalten für den Laserzaun implementiert, welches den Laser in einem zeitlichen Abstand ein- und ausschaltet. Somit kann der Laserzaun auf vier verschiedene Arten als Spielelement eingesetzt werden:

1. Der Laserzaun ist permanent aktiv und dient dazu, die Bewegungsfreiheit des Spielers einzuschränken.
2. Der Laserzaun kann durch einen Schalter deaktiviert werden, sodass die Bewegungsfreiheit des Spielers temporär eingeschränkt ist, um den Spielverlauf zu lenken. Der Spieler muss zuerst den Schalter erreichen, um weiter zu kommen. Der Schalter kann beispielsweise durch andere Hürden und Gegner blockiert sein, die erst besiegt werden müssen.
3. Der Laserzaun ist zeitgesteuert, der Spieler muss seine Bewegungen richtig timen, um einen oder mehrere dieser Zäune zu passieren.
4. Der Laserzaun lässt sich nur mithilfe einer Druckplatte deaktivieren. Sobald der Spieler die Druckplatte verlässt, wird der Zaun wieder aktiviert. Der Spieler ist auf die Kooperation seines Begleiter angewiesen, um den Zaun zu deaktivieren.

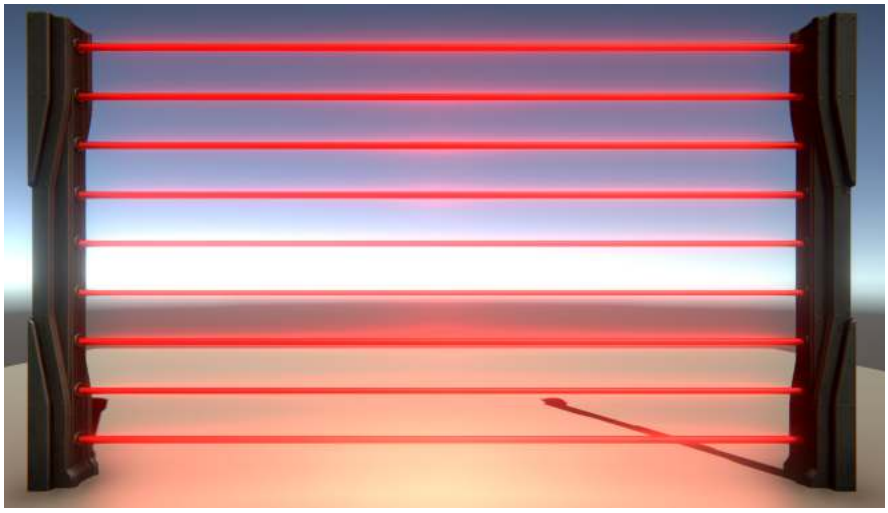

Abbildung 6.8.: Laserzaun Model

Schalter: Über die Schalter können Spielelemente wie Laserzäune, Überwachungskameras und Türen bedient werden. Der Schalter verfügt über zwei Zustände, freigeschaltet und gesperrt (Abb. 6.9). Im gesperrten Zustand ist das mit dem Schalter verbundene Objekt aktive. Im freigeschalteten Zustand ist das mit dem Schalter verbundene Objekt deaktiviert. Mehrere Schalter können in eine Gruppe zusammengefasst werden, so dass alle Schalter aktiviert werden müssen, bevor das verbundene Objekt deaktiviert wird. Für die Schalter wird eine Reihenfolge festgelegt, in der sie aktiviert werden müssen. Wird ein Schalter außerhalb der festgelegten Reihenfolge aktiviert, werden alle Schalter zurückgesetzt. Wodurch Kombinationsrätsel umgesetzt werden, die den Spieler am Voranschreiten hindern, bis sie gelöst werden.

Abbildung 6.9.: Schalter Model (links: Freigeschaltet und rechts: Gesperrt)

Druckplatten: Die Druckplatten (Abb. 6.10) verhalten sich ähnlich wie Schalter, der Unterschied ist, dass diese bei Berührung freischalten. Sobald der Kontakt mit der Druckplatte gelöst wird, sperrt die Platte wieder, wodurch der Spieler auf den Begleiter angewiesen ist, um die Druckplatte freigeschaltet zu lassen, während der Spieler einen Laserzaun, Tür oder Überwachungskamera passiert.

Kisten: Die Kisten sind ein weiteres Hindernis, das der Spieler überwinden muss. Sie können eingesetzt werden, um einen Weg zu blockieren. Die einzige Möglichkeit den Weg freizuräumen ist es, dem Begleiter den Befehl zu geben die Kisten zu zerstören. Hierfür gibt der Spieler den „Helfen“ Befehl. Ist das Vertrauen hoch genug nimmt der Begleiter den Befehl an und bewegt sich auf die Kisten zu. Anschließend wartet der Begleiter auf ein Bestärken durch den Spieler. Ein Emoticon über den Kisten zeigt an, welche Emotion

Abbildung 6.10.: Druckplatten Model (links. freigeschaltet und rechts. gesperrt)

der Spieler nachahmen muss, damit der Begleiter motiviert ist, die Kisten zu zerstören. Um das Bestärken zu aktivieren, drückt der Spieler die zugehörige Taste. Der Spieler hat nun fünf Sekunden Zeit, die Emotion zu imitieren, wurde die Emotion richtig erkannt zerstört der Begleiter die Kisten und der Weg ist frei.

Abbildung 6.11.: Kisten Model

Wache: Die Wache implementiert die in Kapitel 5 beschriebene Agentenarchitektur. Das 3D Model der Wache (Abb. 6.12) stellt einen Sicherheitsroboter dar, welcher mit einem generischen, humanoiden Skelett modelliert wurde. Das Animationssystem hat so die Möglichkeit, verschiedenste generische Animationen auf das Model zu übertragen.

Über einen Sichtsensor wird die Position des Spielers ermittelt. Wenn der Spieler erkannt wird, wird die Position in den Working Memory des Agenten geschrieben. Der Spieler wird erkannt, wenn er sich im Sichtbereich der Wache befindet und von keinen anderen Objekten verdeckt wird. Der Sichtbereich wird durch einen „Collider“ begrenzt, welcher als „Trigger“ konfiguriert ist. Innerhalb dieser Sphäre wird der Sichtbereich der Wache weiter eingeschränkt, sodass die Wache den Spieler nur dann wahrnehmen kann, wenn er sich in einem Winkel von +-55°, ausgehend vom Vorwärtsvektor der Wache, befindet. Der Sichtbereich ist exemplarisch in der Abbildung 6.13 dargestellt, wobei die grüne Linie den „Collider“, der rot ausgefüllte Kreis die Wache und die roten Linien den Sichtradius darstellen. Betritt der Spieler den Bereich des „Colliders“, befindet sich jedoch nicht im Sichtradius der Wache, kann er durch Geräusche wahrgenommen werden. Es wird überprüft, ob der Spieler läuft oder über die Rufen-Taste ein Geräusch von sich gibt.

Die Wache besitzt zwei Motoren, einen Bewegungsmotor und einen Angriffsmotor. Beide werden angesteuert, indem ein Ziel in das Blackboard eingetragen wird. Der Bewegungsmotor bewegt die Wache von seiner derzeitigen Position zu einer Zielposition, so lange ein Ziel vorhanden ist und der boolsche Wert „Movement“ auf dem Blackboard wahr ist. Der Angriffsmotor schießt so lange auf das gesetzte Ziel, bis das Ziel keine Lebenspunkte mehr besitzt oder die Aktion abgebrochen wird, indem das Ziel vom Blackboard gelöscht wird.

In der Abbildung 6.14 ist der Behavior Tree der Wache beschrieben. Dieser ist simple strukturiert, da der Sichtsensor bereits einen erheblichen Teil der KI-Logik enthält. Die Wurzel des Baumes ist ein Selector, welcher die Knoten von links nach rechts validiert. Der erste Knoten „Can Shoot“ ist ein Dekorator der prüft, ob der Spieler gesichtet wurde und sich in Schussreichweite befindet. Ist dies der Fall, wird die Aktion „Shoot“ ausgeführt. Der zweite Knoten „Can Chase“ prüft, ob sich der Spieler im Sichtbereich befindet, ist dies der Fall wird der Bewegungsmotor über die Aktion „Chasing“ angesteuert, so dass sich die Wache auf den Spieler zubewegt. Der letzte Knoten „Partolling“ prüft, ob die Wache einen Wegpunkt seiner Patrouille erreicht hat, ist dies der Fall wird der nächste Position aus einer Liste von Wegpunkten gewählt. Ist die Position noch nicht erreicht, wird der Bewegungsmotor mit dem aktuellen Wegpunkt angesteuert.

Lebenskugeln: Um die verlorenen Lebenspunkte wiederherstellen zu können, kann der Spieler Lebenskugeln einsammeln (Abb. 6.15). Diese stellen die gesamten Lebenspunkte des Spielers wieder her. Beim Aufsammeln wird die Lebenskugel konsumiert und verschwin-

Abbildung 6.12.: Wachen Model

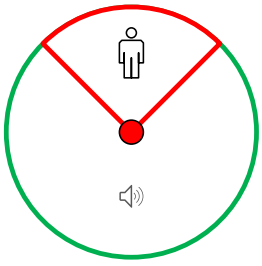

Abbildung 6.13.: Sichtbereich der Wachen

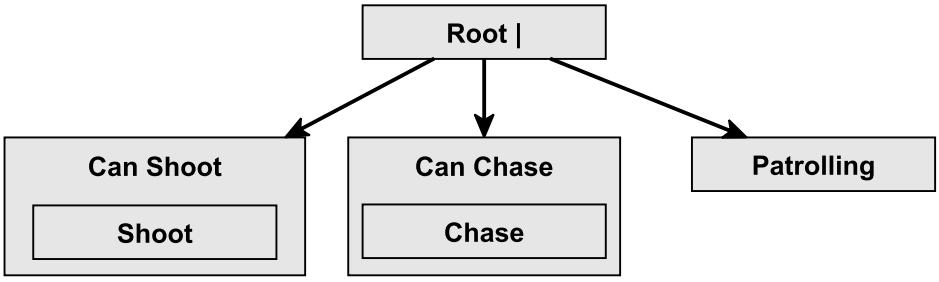

Abbildung 6.14.: Behaviro Tree der Wachen

det. Hat der Spieler bereits volle Lebenspunkte, bleibt die Lebenskugel erhalten und kann später eingesammelt werden.

Abbildung 6.15.: Lebenskugel

6.4.5. Begleiter

Der Begleiter ist, wie auch die Wachen, nach der in Kapitel 5 beschriebenen Agentenarchitektur implementiert. Der Begleiter ist ein humanoider Roboter (Abb. 6.16) der indirekt über die Befehle des Spielers gesteuert wird. Im Folgenden werden die Sensoren, Motoren und der Behavior Tree des Begleiters beschrieben.

CommandSensor: Über diesen Sensor werden die Befehle des Spielers erkannt. Während eines Ausführungszyklus werden alle konfigurierten Tasten abgefragt und das Ergebnis als Enumeration (siehe 6.3) in den Working Memory des Agenten geschrieben. Wobei die Werte None, Interact und Chicken nur intern genutzt werden.

```
1 public enum Command
```

Abbildung 6.16.: Begleiter Model

```
{
  None,
  Stay,
  Follow,
  Help,
  Interact,
  Chicken
}
```

Listing 6.3: Befehlsschlüssel

AffdexSensor: Der „AffdexSensor“ verarbeitet die über die Affdex Api (siehe 5.9) bereitgestellten Daten. Über einen Filterfunktion wird die erkannte Emotion ermittelt (siehe 5.10). Anschließend wird bestimmt, ob es sich um eine positive oder negative Emotion handelt. Der „AffdexSensor“ verwendet zwei Listen. Eine Liste enthält alle Emotionen, die als positiv kategorisiert und eine Liste der Emotionen, die als negativ kategorisiert werden. Über einen Vergleich wird bestimmt, in welcher der beiden Listen sich die aktuell erkannte Emotion befindet. Das Ergebnis wird als „EmotionCluster“ Objekt zur Weiterverarbeitung im Working Memory des Agenten gespeichert. Der „EmotionClus-

ter“ (siehe 6.4) ist eine Datenklasse, welche die Eigenschaften der erkannten Emotion speichert, wobei „Value“ der Wahrscheinlichkeitswert der Emotion ist.

```
public enum Cluster{
  Neutral,
  Positive,
  Negative
}
public Emotions Emotion;
public float Value = 0;
public Cluster Cluster;
```

Listing 6.4: EmotionCluster Datenklasse

PossiblePositions: Der Sensor bestimmt die Position des Begleiters wenn er dem Spieler folgt. Ziel ist es, dass der Begleiter immer eine valide Position in der Nähe des Spielers besitzt, die den Weg des Spielers nicht blockiert. Hierfür werden, ausgehend von dem Spieler, drei feste Positionen auf ihre Eignung überprüft. Die Positionen sind links, rechts und hinter dem Spieler angeordnet. Es wird geprüft, ob sich zwischen dem Spieler und der Position andere Objekte befinden und ob die Position durch ein anderes Objekt blockiert wird. Dieser Vorgang ist in der Abbildung 6.17 exemplarisch dargestellt. Die möglichen Positionen für den Begleiter sind (1), (2) und (3), wobei sich zwischen der Position (3) und dem Spieler eine Wand (4) befindet, wodurch diese Position nicht mehr valide ist und die Position (1) ausgewählt wird. Prinzipiell ist es möglich, so viele Positionen wie gewünscht zu konfigurieren. Die Positionen werden in der Reihenfolge (1), (3) und (2) priorisiert. Die drei Positionen werden in einer Liste über die Datenklasse „PossiblePosition“ (siehe 6.5) in dem Working Memory des Begleiters gespeichert.

```
public GameObject Position;
public bool Valid;
```

Listing 6.5: PossiblePosition Datenklasse

InteractableSensor: Der Sensor erfasst alle in der näheren Umgebung befindlichen Objekte, mit denen der Begleiter interagieren kann. Dazu gehören Wachen, Druckplatten und Kisten. Diese werden in einer Liste im Working Memory des Begleiters gespeichert. In der Datenklasse „Interactable“ (siehe 6.6) wird gespeichert, wie der Begleiter mit dem „Interactable“ interagieren kann. Wobei der Wert „Shoot“ angibt, ob sich der Begleiter nur an die Position des „Interactable“ bewegen muss oder ob er darauf schießen muss

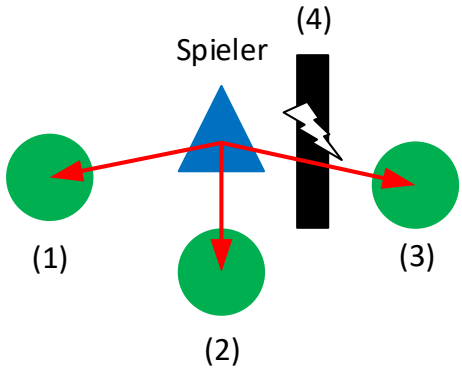

Abbildung 6.17.: Mögliche Positionen für den Begleiter

sobald er in Schussreichweite ist. Über der Wert „NeedsReinforcement“ wird angegeben, ob der Begleiter auf ein Bestärken durch den Spieler wartet, bevor die Aktion ausgeführt wird. Über den Enumeration „ReinforcementType“ wird angegeben, ob ein positives oder negatives Bestärken benötigt wird.

```
1 public bool Shoot;
2 public bool NeedsReinforcement;
3 public EmotionCluster.Cluster ReinforcementType;
```

Listing 6.6: Interactable Datenklasse

ReinforcementSensor: Die durch den „AffdexSensor“ erfasste Emotion wird über diesen Sensor weiterverarbeitet. Drückt der Spieler die Bestärken-Taste, erfasst der Sensor über einen maximalen Zeitraum von 5 Sekunden die Emotionen. Sobald während dieses Zeitraums die Obergrenze an positiven oder negativen, erkannten Emotionen erreicht wird, wird das Bestärken erfolgreich beendet und das Ergebnis als Enumeration im Working Memory gespeichert. Wird innerhalb des Zeitraums keine Obergrenze erreicht, endet das Bestärken ohne Ergebnis.

MovementMotor: Der Motor bewegt den Agenten von seiner aktuellen Position an die gewünschte Zielposition. Über ein Boolean, welches über das Blackboard abgerufen wird, kann der Motor aktiviert und deaktiviert werden. Über einen Vektor wird die Zielposition auf das Blackboard geschrieben. Wenn die Zielposition erreicht ist, stoppt

der „MovementMotor“ automatisch. Um einen geeigneten Pfad von der aktuellen Position zum Ziel zu ermitteln, wird das Navigation System (siehe 6.4.1) von Unity verwendet.

ShootingMotor: Dieser Motor steuert die Animationen und Audio wiedergaben, die für das Schießen benötigt werden. Der Begleiter schießt zwei Laserstrahlen aus seinen Augen. Der Schaden wird abhängig von der Entfernung zum Ziel berechnet, wobei der minimale und maximale Schaden konfigurierbar ist. Der Motor wird angesteuert indem das Objekt, auf das geschossen werden soll, auf das Blackboard geschrieben wird. Der Motor schießt so lange, bis das Objekt zerstört oder vom Blackboard entfernt wird. Jedes Objekt, das als Ziel angegeben wird, benötigt die Klasse „BaseHealth“, auf die der „ShootingMotor“ zugreift. „BaseHealth“ (siehe 6.7) ist die Basisklasse, die für das Verhalten der Lebenspunkte von Objekten zuständig ist. Eine erweiterte Klasse wird für die Wachen, Spieler und Kisten verwendet.

```
1  public float Health = 100f;
2  public bool _isDead = false;
3  public void TakeDamage(float amount){
4    Health -= amount;
5    if (Health > 0f) return;
6      _isDead = true;
7  }
8  public bool IsDead(){
9    return _isDead;
10 }
```

Listing 6.7: BaseHealth Klasse

Behavior Tree: Der gesamte Behavior Tree ist in der Abbildung C.1 zu sehen. In diesem Abschnitt wird die Verhaltensmodellierung des Begleiters anhand von ausgewählten Ausschnitten aus dem Behavior Tree beschrieben. Die Abbildung 6.18 zeigt übersichtlich, wie der Begleiter seine Informationen verarbeitet.

Die Wurzel des Baumes ist ein Parallel Selector, der die drei Unterbäume „ChangeColor“, „Command“ und „Reinforcement“ ausführt. Die „ChangeColor“ Aktion ändert anhand des aktuellen Vertrauens die Farbe von dem Herz (siehe 6.16) welches über dem Kopf von dem Begleiter schwebt. Somit wird dem Spieler Feedback über den aktuellen Vertrauenszustand gegeben, die Farben sind in drei Stufen unterteilt (siehe 6.2).

Über den „Command“ Sequence Selector wird entschieden, ob ein Befehl vom Spieler akzeptiert oder verworfen wird. Ob der Befehl akzeptiert wurde, wird über den Listen

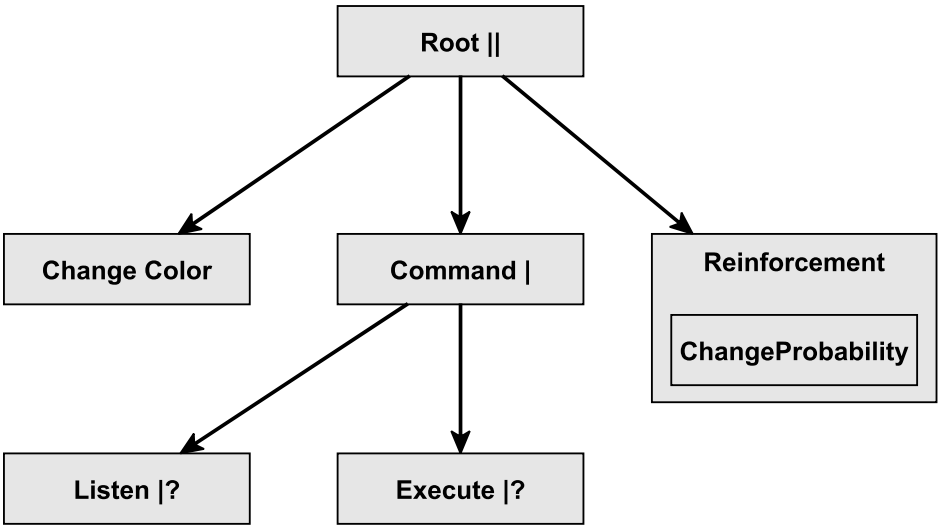

Abbildung 6.18.: Begleiter Behavior Tree generelles Verhalten

Vertrauen	Farbe
< 33%	Rot
< 66%	Gelb
> 66%	Grün

Tabelle 6.2.: Herz, Farbgebungen basierend auf dem Vertrauen

Selector Unterbaum entschieden. Der Selector „Execute“ und dessen Unterbaum führt dann anschließend den Befehl des Spielers aus.

Über die Aktion „Reinforcement“ wird der „ProbabilityTable“ (siehe 6.8) des letzten Befehls angepasst. Diese Klasse enthält die Aktionen, die aufgrund eines Befehls getroffen werden können. Für jeden möglichen Befehl kann eine solche Tabelle angelegt und in dem Working Memory des Begleiters gespeichert werden. Existiert kein „ProbabilityTable“ für einen Befehl wird das „ReinforcementResult“ konsumiert und keine Änderung vorgenommen.

```
1 private List<ProbabilityContainer> _probabilityList;
2 private ProbabilityContainer _lastResult;
3 public ProbabilityContainer GetChildByProbability()
4 public void ChangeProbability(float amount)
```

Listing 6.8: Probability Table Klasse

Die Klasse „ProbabilityTable“ verwaltet eine Liste von „ProbabilityContainer“ welche einen Befehl und einen Wahrscheinlichkeitswert besitzen (siehe 6.9). Über die Funktion „ChangeProbability“ verändert die Aktion „Reinforcement“ den Wahrscheinlichkeitswert der zuletzt ausgewählten Aktion des „ProbabilityTable“, wobei die Wahrscheinlichkeit der zuletzt ausgewählten Aktion um einen angegebenen Wert, abhängig von dem „ReinforcementResult“, erhöht oder verringert wird. Die Wahrscheinlichkeiten der anderen, möglichen Befehle werden proportional zum angegeben Wert verringert oder erhöht (siehe 6.10).

```
1 public float Probability = 0;
2 public Companion.Command Command = Companion.Command.None;
```

Listing 6.9: Probability Container Klasse

```
1 foreach (var child in _probabilityList){
2   if (child  == _lastResult)
3     child.Probability += amount;
4   else
5     child.Probability -= amount / (_probabilityList.Count - 1);
6 }
```

Listing 6.10: ChangeProbability Funktion

Der Unterbaum „Listen“ (Abb. 6.19) prüft, ob ein Befehl akzeptiert wird. Der Decorator „DontListen“ prüft, ob das Vertrauen unter dem Wert von 33% liegt. Wenn die Bedingung zutrifft, wird die „DontListen“ Sequenz ausgeführt, andernfalls wird der Befehl

akzeptiert. Wurde der Befehl akzeptiert, prüft die Aktion „ChangeQueryByProbaility“, ob ein „ProbabilityTable“ für den beantragten Befehl existiert. Ist ein „ProbabilityTable“ vorhanden, wird über die Funktion „GetChildByProbarbility“ die auszuführende Aktion gewählt. Anschließend wird die „Accept“ Sequenz ausgeführt, welche dem Benutzer ein Feedback gibt, das der Befehl akzeptiert wurde und den auszuführenden Befehl in den Working Memory des Agenten schreibt.

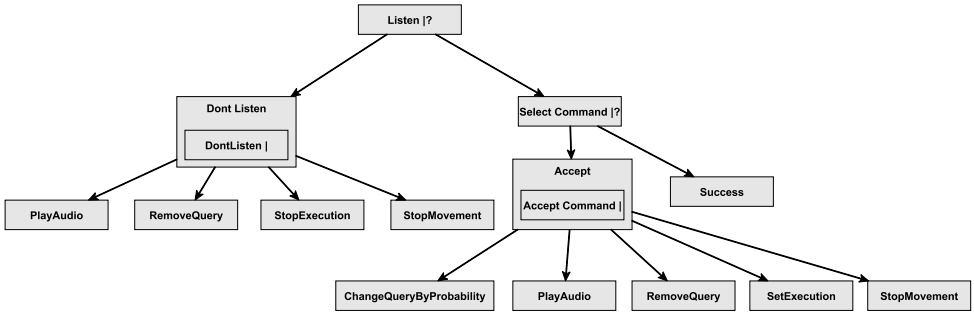

Abbildung 6.19.: Begleiter Behavior Tree Listen Unterbaum

Anhand von dem „Help“ Befehl (Abb. 6.20) wird die Ausführung von Befehlen beschrieben. Als erste Priorität wird geprüft, ob sich eine Wache in Reichweite befindet. Ist dies der Fall, wird über die Aktion „EnemyVisible“ der „ShootingMotor“ angesteuert und die Wache bekämpft. Andernfalls wird geprüft, ob sich ein „Interactable“ in Reichweite des Begleiters befindet. Ist dies der Fall, wird das „Interactable“ das dem Spieler am nächsten ist ausgewählt und die benötigten Aktionen ausgeführt. Handelt es sich um ein „Interactable“, das ein „Reinforcement“ benötigt, wird über die Aktion „WaitForReinforcement“ auf ein „ReinforcementResult“ gewartet. Das Warten wird durch das Ausführen eines anderen Befehls unterbrochen.

6.4.6. Level Design

Das für die Vorversuche verwendete Level wird anhand einiger Schlüsselszenen erläutert, um den der Ablauf des Spiels zu skizzieren. Zu Beginn des Spiels befindet sich der Spieler (Abb. C.2) alleine in einem Raum. Der Raum ist sicher, sodass der Spieler nicht gleich bei Spielbeginn von den Wachen angegriffen wird.

Die erste Aufgabe des Spielers ist es seinen Begleiter aus dem Gefängnis zu befreien (Abb. C.3). Von dem Start aus kann der Spieler nur in eine Richtung fortbewegen. Verlässt er den Startbereich, muss er sich an zwei Wachen bis zum Gefängnis vorbei schleichen. Dem Spieler bleibt keine andere Möglichkeit, da er sich nicht gegen die Wachen erwehren kann. Ist der

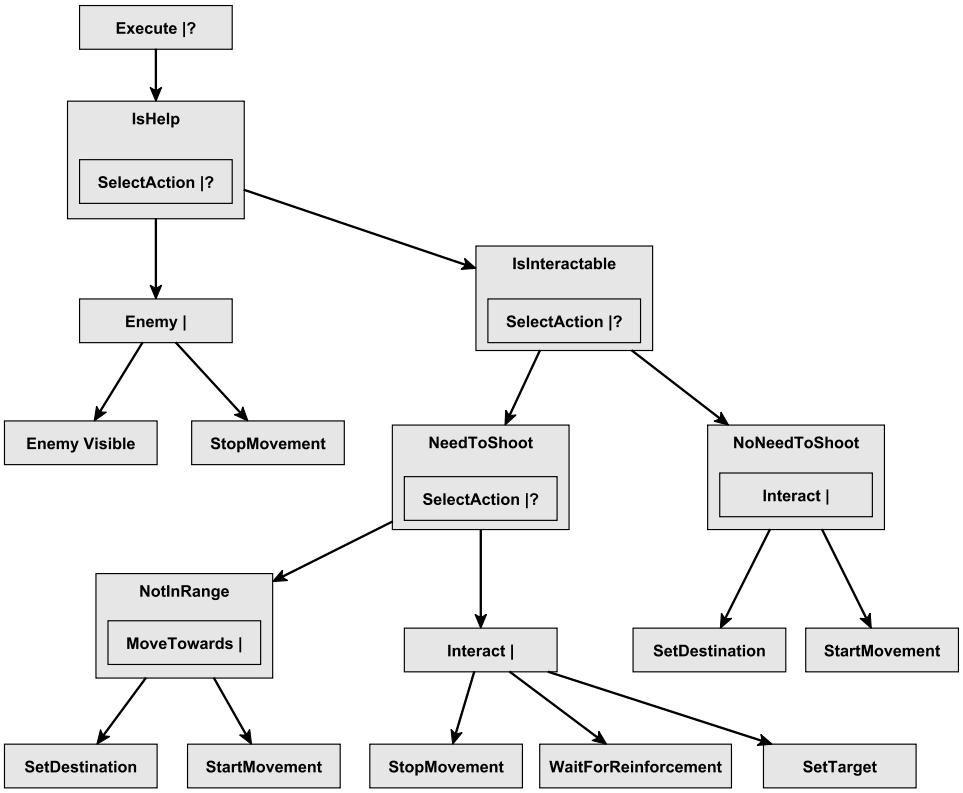

Abbildung 6.20.: Begleiter Behavior Tree Unterbaum für den Help Befehl

Spieler im Gefängnis angekommen, muss er ein Schalterrätsel lösen, um den Begleiter aus dem Gefängnis zu befreien. Hierfür müssen die Schalter in einer bestimmten Reihenfolge nacheinander betätigt werden.

Anschließend kann der Spieler mit der Hilfe des Begleiters die Wachen ausschalten (Abb. C.4) oder sich an ihnen vorbei schleichen. Dadurch, dass der Spieler den Begleiter befreit hat, kann er nun Laserzäune deaktivieren und Schalter bzw. Druckplatten (Abb. C.5) betätigen, die das Weiterkommen zuvor verhindert hatten.

Im weiteren Spielverlauf wird der Weg durch Kisten (Abb. C.7) versperrt. Diese kann der Spieler beseitigen, indem er den Begleiter um Hilfe bittet. Andere Hindernisse, wie die zeitgesteuerten Laserzäune (Abb. C.6) verlangen Feingefühl und Timing von dem Spieler. Das Ende des Levels wird durch eine verschlossene Tür blockiert. Der Spieler kann das Level erst beenden, wenn er eine Schlüsselkarte ergattern konnte. Durch Tests wurde eine Spielzeit von 5 Minuten ermittelt, diese kann jedoch abhängig von dem Spieler variieren.

6.4.7. Zusammenfassung

Die Implementation setzt alle im Game Design (siehe 6) beschriebenen Spielmechaniken um. Sowohl die Wache als auch der Begleiter verwenden die in Kapitel 5 beschriebene Agentenarchitektur. Die an das Spiel gestellten Anforderungen konnten erfolgreich eingehalten werden. Der Agent lässt sich bei Bedarf so konfigurieren, dass er nicht auf die Emotionen des Spielers reagiert. Somit ist es möglich, Versuche mit und ohne emotionale Interaktion durchzuführen. Durch erste Tests wurde eine Spiellänge von ca. 5 Minuten für das Level ermittelt. Somit wird bei einer Versuchsdurchführung mit Kontrollgruppe eine Gesamtspiellänge von 10 Minuten nicht überschritten.

7. Versuche und Auswertung

7.1. Bedingungen an den Versuchsaufbau

Um die bestmöglichste Versuchsumgebung zu schaffen, sind im Hinblick auf die Gesichtserkennung eine Reihe von Vorbedingungen zu erfüllen.

Beleuchtung: Durch Abdunkelung des Raumes und künstliches Licht werden optimale Lichtverhältnisse für die Versuchsdurchführung geschaffen. Die Beleuchtung darf weder zu dunkel noch zu hell sein, da beides das Gesicht im Kamerabild verdecken würde. So werden Fehler bei der Gesichtserkennung durch Veränderung von Witterungsverhältnissen und Tageszeit ausgeschlossen.

Bewegungen des Probanden: Die Aufmerksamkeit des Probanden sollte während der gesamten Versuchsdurchführung auf dem Spiel liegen. Ablenkungen durch den Versuchsdurchführenden, werden auf ein Minimum reduziert. So reagiert der Versuchsdurchführende nur auf Fragen des Probanden und nimmt keine aktive Rolle während der Versuche ein.

Kamera Positionierung: Die Positionierung der Kamera ist entscheidend für die Erkennungsrate. Um ein optimales Ergebnis erzielen zu können, muss das Gesicht des Probanden frontal und zentriert im Bild positioniert werden und einen möglichst großen Bereich des Bildes abdecken. Aus diesem Grund wird vor jedem Versuch die Position der Kamera an den aktuellen Probanden angepasst.

Verdeckung des Gesichts: Gegenstände und Bekleidung, die das Gesicht verdecken sind zu vermeiden. Brillen, Mützen oder jegliche andere Art von Kopfbedeckung, die das Gesicht merklich verdecken können, müssen während der Versuchsdurchführung abgelegt werden. Das Tragen von Brillen sollte kein Problem darstellen. Um das Risiko von fehlerhaften Versuchen zu mindern, werden während der Vorversuche trotzdem Personen, die ohne Brille spielen können, bevorzugt. Personen mit ausgeprägter Gesichtsbehaarung werden von den Vorversuchen ausgeschlossen. Verdeckungen des Gesichts durch

Wegdrehen oder Kratzen stellen keine größeren Probleme dar, da diese nicht häufig und zeitlich begrenzt auftreten.

7.2. Fragebögen

Fragebogen 1: Vor der Versuchsdurchführung muss jeder Proband einen kurzen Fragebogen mit zehn Fragen beantworten (siehe A.1 und A.2). Die Beantwortungszeit des Fragebogens wird auf ca. 2 bis 3 Minuten geschätzt. Der Fragebogen soll Aufschluss darüber geben, ob es sich bei dem Probanden um eine Person handelt, die Videospiele spielt, wie lange die Person spielt und was diese Person bei Videospielen bevorzugt. Unterteilt sind die Fragen in die Kategorien Grafik, Sound, Charaktere, Geschichte und Spielmechaniken.

Fragebogen 2: Nach Abschluss der Untersuchungen muss jeder Proband einen weiteren Fragebogen ausfüllen (siehe A.3 und A.4). Dieser Fragebogen soll Aufschluss darüber geben, wie den Probanden das Spiel gefallen hat. Insbesondere wie die emotionale Interaktion mit dem Begleiter die Wahrnehmung des Spiels beeinflusst. Außerdem ist von Bedeutung ob diese Aspekte der Interaktion den Probanden gefallen hat und ob diese als sinnvolle Erweiterung der klassischen Spielmechaniken wahrgenommen wird oder negative Auswirkungen auf das Spiel hat. Der Proband hat die Möglichkeit durch zwei offene Fragen, hervorheben, was ihn an dem Spiel gut und was ihm nicht gefallen hat. Zudem wird ganz direkt gefragt, ob diese Art der Interaktion nach Einschätzung des Probanden Auswirkungen auf deren Immersion hat. Der Fragebogen wird sowohl für die Versuche mit emotionaler Interaktion als auch für die Versuche ohne emotionale Interaktion verwendet. Besonders interessant ist hier der Vergleich beider Gruppen bei den Fragen, welche die emotionalen Interaktion betreffen. Die Beantwortungszeit des Fragebogens für beide Durchläufe wird auf ca. 5 bis 6 Minuten geschätzt.

7.3. Versuchsdurchführung

Zu Beginn der Versuchsdurchführung werden die Probanden gebeten den Fragebogen 1 auszufüllen. Anschließend wird die Steuerung und die emotionalen Interaktionen erklärt. Danach wird das Spiel mit dem rationalen Agenten gestartet. Der Proband hat nun fünf Minuten Zeit zu spielen. Sollte er es innerhalb dieser Zeit nicht schaffen das Level zu beenden wird es ihm freigestellt aufzuhören. Nach dem Beenden des ersten Durchlauf wird der Proband gebeten den Fragebogen 2 auszufüllen. Der Proband wird nun darüber aufgeklärt, dass die Emotionserkennung deaktiviert war und keine emotionalen Interaktionen möglich waren. Nachdem

der Fragebogen 2 ausgefüllt wurde, wird das Spiel mit dem emotional reaktiven Agenten gestartet. Dem Probanden werden nochmals die Steuerung und die emotionale Interaktion mit dem Begleiter erklärt. Nach weiteren fünf Minuten Spielzeit wird der Proband gebeten, den Fragebogen 2 nochmals auszufüllen. Abschließend kann der Proband, wenn er möchte, weitere für ihn wichtige Punkte ansprechen, die seiner Meinung nach nicht über den Fragebogen abgedeckt wurden.

7.4. Auswertung der Versuche

Durch die Erfahrungen, die innerhalb des Emotion Bike Projekt erworben haben, war zu erwarten, dass sowohl das Game Design als auch die Implementation mehrere Iterationen benötigen. So muss das Spiel durch kleinere Versuchsdurchführungen evaluiert werden, bevor größere Versuchsreihen mit einer adäquaten Anzahl an Probanden durchgeführt werden können. Bei dem Emotion Bike dauerte es ca. ein Jahr bevor das Spiel und alle Komponenten fehlerfrei funktionierten. In dieser Zeit wurde das Game Design anhand von kleineren Studien immer wieder getestet und anhand des Feedbacks überarbeitet.

Die ersten durchgeführten Versuche dienten vorrangig dazu, die Funktionalität des implementierte Spiels, das Game Design und die emotionale Interaktion zu evaluieren. So wurden diese erst einmal nur mit drei Probanden durchgeführt. Wobei alle Probanden männlich und im Alter von 27 und 28 und 43 Jahren waren.

Alle gegebenen Antworten für den Fragebogen 1 sind in der Tabelle 7.1 aufgelistet. Die Auswertung des Fragebogens 1 ergab, dass der Proband 1 ein Gelegenheitsspieler und die Probanden 2 und 3 sich als Gamer identifiziert werden können. Durch Beobachtungen während der Versuche konnte festgestellt werden, dass der Proband 1 deutlichere Schwierigkeiten mit der Steuerung hatte. So lief er öfters gegen Wände und hatte Probleme sich die Tastenbelegung zu merken. Auch der Fortschritt innerhalb des Spiels lag hinter den der anderen Probanden, wobei nur Proband 3 das Spiel vollständig beenden konnte.

Alle Probanden gaben an, dass sie sich beim Spielen stark auf dass Spiel konzentrieren. Dies ist ein Indiz, dass diese Personen eher geneigt sind, sich in ein Spiel zu vertiefen. Das umgesetzte Spiel wird dem Genre der Action/Adventure Spiele zugeordnet. Bei der Frage 4, welche Art von Spiel bevorzugt wird, gab nur der Proband 2 an, dass er Action/Adventure Spiele bevorzugt. Einen Einfluss auf die Bewertung des Spiels konnte jedoch nicht festgestellt werden.

Betrachten wir die Bewertung der einzelnen Spielelemente wie Grafik, Sound und deren Wichtigkeit für die Probanden, konnte kein Einfluss auf die Bewertung des Spiels beobachtet

Frage	Proband 1	Proband 2	Proband 2
Frage 1	43 Jahre, Männlich	27 Jahre, Männlich	28 Jahre, Männlich
Frage 2	4	9	10
Frage 3	8	8	8
Frage 4	Strategie/Simulation	Action/Adventure	Ego-Shooter/Fahrsimulation
Frage 5	5	9	9
Frage 6	5	9	5
Frage 7	8	9	6
Frage 8	8	9	6
Frage 9	9	9	9
Frage 10	Spielmechaniken	Geschichte	Spielmechaniken

Tabelle 7.1.: Antworten Fragebogen 1

werden. Den Charakteren in Videospielen wurde von allen Probanden eine hohe Wichtigkeit zugeordnet und das Konzept des Begleiters als positiv empfunden. Ebenfalls besonders wichtig waren die Spielmechaniken und die Geschichte, wobei Proband 1 und Proband 3 angaben, dass sie die Spielmechaniken am meisten an einem Spiel fesseln. Das spiegelte sich auch in der Kritik nieder, da Proband 2 und 3 die Steuerung bemängelten.

Der erste Versuchsdurchlauf wurde mit dem rationalen Agenten durchgeführt, welcher nicht auf die Emotionen des Spielers reagiert. Die Antworten des Fragebogens 2 sind in der Tabelle 7.2 aufgelistet. Der zweite Versuchsdurchlauf wurde mit dem emotional reaktiven Agenten durchgeführt. Die Versuchsergebnisse können der Tabelle 7.3 entnommen werden.

Nach der ersten Versuchsdurchführung mit dem rationalen Agenten haben alle Probanden richtig erkannt, dass dieser nicht auf ihre Emotionen reagiert. Es war jedoch zu beobachten, dass alle Probanden bei der Beantwortung der Fragen 4 bis 7 des Fragebogens 2 Probleme hatten. Die Fragen gehen gezielt auf die emotionale Interaktion mit dem Begleiter ein und anhand der Beobachtung konnten diese Fragen nur negativ beantwortet werden. Die Probanden zögerten jedoch beim Ausfüllen des Fragebogens und mussten bestärkt werden ihre Beobachtungen anzugeben, ohne zu berücksichtigen, welches Ergebnis ihrer Meinung nach von ihnen erwartet werden könnte.

Die allgemeine Wahrnehmung des Spiels hat sich durch den Einsatz der Emotionserkennung nicht maßgeblich verändert. Der Proband 1 hat seine Bewertung sogar um einen Punkt nach unten korrigiert, während Proband 2 und Proband 3 die selbe Punktzahl vergeben haben.

Bei der Frage 2 wo nach den positiven Aspekten des Spiels gefragt wurde, haben alle Probanden angegeben, dass ihnen das Spielkonzept gefallen hat. Nach der zweiten Versuchs-

Frage	Proband 1	Proband 2	Proband 2
Frage 1	8	7	8
Frage 2	Ambiente, Spielidee, Grafik	Helfer, Konzept	Sound, Simple, Spannend
Frage 3	Die Lasertore	Bewegung zu ruckhaft	Steuerung zu „schwammig“ indirekt, Kamera nicht drehbar
Frage 4	2	1	1
Frage 5	3	1	1
Frage 6	3	1	1
Frage 7	3	1	1

Tabelle 7.2.: Antworten Fragebogen 2: Spieldurchlauf ohne Emotionserkennung

durchführung haben die Probanden 2 und 3 die emotionalen Interaktion als positiven Aspekt des Spiels angegeben.

Bei der Frage nach negativen Aspekten des Spiels haben die Probanden 2 und 3 nach beiden Durchläufen die Steuerung bemängelt. Der Proband 1 hat im ersten Versuchsdurchlauf die Laserzäune als negativen Aspekt angegeben. So bereiteten allen Probanden die zeitgesteuerten Laserzäune Probleme, lediglich der Proband 3 konnte dieses Hindernis überwinden.

Des Weiterem gab der Proband 1 an, dass ihm nicht gefallen hat, dass der Begleiter ihm im Stich gelassen hat. Grund hierfür war ein zu geringes Vertrauen, sodass der Begleiter nicht mehr auf die Befehle des Spielers reagiert hat. Dies wurde von dem Probanden 1 jedoch nicht wahrgenommen.

Generell hat der Proband 1 die emotionale Interaktion in beiden Durchläufen negativ bewertet. Das kann jedoch auch auf technische Probleme zurückgeführt werden. Die Emotionserkennung hatte Probleme die Emotionen von Proband 1 zu erkennen. Der Proband hatte einen kleinen Kinnbart, welcher Einfluss auf die Emotionserkennung gehabt haben könnte. Zudem war der Proband eher zurückhaltend und zeigte kaum aussagekräftige Gesichtsausdrücke. Dies wirkte sich besonders stark beim Beseitigen der Kisten aus (siehe 6.11). Bei dieser Interaktion muss der Spieler den Begleiter durch das Ausdrücken einer negative Emotion dazu bringen, die Kisten zu zerstören.

Die Probanden 2 und 3 haben die emotionale Interaktion mit dem Begleiter positiv bewertet und hatten den Eindruck, dass dieser adäquat auf ihre Emotionen reagierte. So wurde die emotionale Interaktion als sinnvolle Ergänzung der Spielmechaniken angesehen. Der potenzielle Einfluss auf die Immersion wurde bestätigt, wobei Proband 3 besonders stark zustimmte.

Frage	Proband 1	Proband 2	Proband 2
Frage 1	7	7	8
Frage 2	Spielidee, Konzept, Grafik	Konzept, Reaktion auf Emotionen	Sound, Simple, Spannend, Emotionen
Frage 3	Mein Companion hat mich im Stich gelassen.	Bewegung zu ruckhaft	Steuerung zu „schwammig“ indirekt, Kamera nicht drehbar
Frage 4	3	9	8
Frage 5	4	7	6
Frage 6	3	7	10
Frage 7	3	6	9

Tabelle 7.3.: Antworten Fragebogen 2: Spieldurchlauf mit Emotionserkennung

7.5. Zusammenfassung

Die Vorversuche haben gezeigt, dass das Spielprinzip positiv angenommen wird und die Implementation fehlerfrei funktionierte. Wir konnten zeigen das, das Spiel mit und ohne Emotionserkennung funktioniert. Die emotionale Interaktion wurde überwiegend positiv empfunden. Dies muss jedoch durch weiterführende Versuche mit einer größeren Anzahl an Probanden bestätigt werden. Wir haben ein System entwickelt und implementiert das die technischen Voraussetzungen geschaffen hat, emotionale Interaktionsmöglichkeiten mit und ohne Begleiter zu testen und deren Auswirkungen auf den Spieler zu untersuchen. Einzelne Interaktionen lassen sich separat ein- und ausschalten, so können isolierte Untersuchungen einzelner Interaktionen durchgeführt werden. Die Emotionserkennung kann auch komplett deaktiviert werden, sodass eine Untersuchung mit Kontrollgruppen durchgeführt werden kann.

Um zu verhindern, dass negative Emotionen ungewollt durch technische Mängel der Umsetzung ausgelöst werden, müssen anhand der Kritik durch die Probanden Anpassungen am Spiel durchgeführt werden. Eine der größten Kritikpunkte war die Steuerung, eine genauere Kontrolle der Bewegungsgeschwindigkeit über den linken Analogstick sollte die Steuerung zugänglicher machen. Zusätzlich würde die Taste für das Schleichen wegfallen, was die Steuerung vereinfacht. Es zeigte sich, dass das Betätigen der Schalter verbessert werden muss, indem der Bereich zur Aktivierung des Schalters vergrößert wird. Zusätzlich kann über Texteinblendung angezeigt werden, dass der Schalter aktiviert werden kann und welche Taste dafür gedrückt werden muss. Eine durch den Spieler kontrollierte frei rotierende Kamera würde es erleichtern die Wachen im Blickfeld zu behalten und ihre Bewegungen zu verfolgen.

Die Interaktionsmöglichkeiten des Begleiters mit den Druckplatten muss deutlicher kommuniziert werden. Das Icon eines Roboters auf den Druckplatten würde diese eindeutig kenn-

zeichnen. Das Bestärken sollte dem Benutzer deutlicher visualisiert werden, indem im Zentrum des Bildschirms eine Füllleiste für positive und negative Emotionen eingeblendet wird.

Die Schwierigkeit des Spiels muss reduziert werden. Am Anfang hatten die Probanden Schwierigkeiten den Wachen auszuweichen und zum Begleiter vorzudringen. Dies sollte durch das Entfernen einer Wache behoben werden. Die zeitgesteuerten Laserschranken war für alle Probanden das schwerste Hindernis, dies kann durch eine Veränderung der Ein- und Ausschaltzeiten ausgeglichen werden.

Es muss untersucht werden, ob eine Anpassung des Schwellwertes für die Emotionserkennung dazu führt, dass auch weniger stark ausgeprägte Emotionen zuverlässig erkannt werden können. Zusätzlich sollte analysiert werden, ob individuelle Schwellwerte für die einzelnen Emotionen die Erkennungsrate erhöhen.

8. Fazit

8.1. Zusammenfassung

Ziel dieser Arbeit war es, eine Plattform zu konzeptionieren und zu entwickeln um KI-Agenten für Computerspiele, die adäquat auf die Emotionen eines Spielers reagieren, untersuchen zu können. Zu Beginn wurden die Forschungsbereiche Affective Computing und Companion Technologie untersucht (siehe 2). Es wurde das Vorgehen bei der Emotionserkennung und verschiedene möglichen Sensoren beschrieben. Es hat sich herausgestellt, dass kamerabasierte Systeme das größte Potenzial haben, von Benutzer akzeptiert zu werden. So müssen diese nicht an den Körper angebracht werden, sie verfügen bereits über eine hohe Verbreitung und geringe Anschaffungskosten.

Anschließend wurden mögliche Anwendungsbereiche beschrieben. Im Speziellen wurde der Teilbereich des Affective Gaming genauer untersucht und anhand ausgewählter Projekte das Vorgehen und verwendeten Technologien aufgezeigt (siehe 2.2). Es wurde gezeigt, dass sich die meisten Arbeiten darauf konzentrieren, die Spielbedingungen für den Spieler anhand des affektiven Zustands zu verändern. Eine direkte Interaktion zwischen KI-Agenten und Spieler wurde in den meisten Arbeiten nicht betrachtet. Diese Interaktion hat jedoch großes Potenzial durch Emotionen auf natürliche und sinnvolle Weise ergänzt zu werden. Somit könnten Charaktere in Computerspielen noch lebensechter wirken und für den Menschen nachvollziehbarer handeln, wodurch noch überzeugendere Spielewelten erschaffen werden könnten.

Anhand von Vorversuchen wurde die Machbarkeit eines adäquat emotional agierenden Agenten untersucht. In Zusammenarbeit mit Doktoranden und Bachelorstudenten wurde im Rahmen des Emotion Bike Projekt eine Plattform entwickelt (siehe 3). Es wurde der Zusammenhang zwischen Ereignissen in Spielen und ausgedrückten Emotionen der Spieler analysiert. Als Spiel wurde ein Fahrradsimulator entwickelt und mit Hilfe verschiedenster Messtechniken die Emotionen von Probanden erfasst. Der Versuchsaufbau bestand aus einem Fahrradsimulator, der über ein Ergometer gesteuert wurde. In einer Fallstudie wurden die erfassten Kameradaten ausgewertet und in eine Beziehung mit den Spielereignissen gebracht.

Die Auswertung ließ vermuten, das ein direkter Zusammenhang zwischen den Ereignissen im Spiel und gezeigten Emotionen des Spielers existiert (siehe 3.2.1). Folglich kann auch ein KI-Agent Emotionen bei dem Spieler provozieren. Wenn der Agent Aktionen ausführt oder andere Ereignisse im Spiel stattfinden, kann der KI-Agent ein direktes emotionales Feedback durch den Spieler erwarten. So kann er sein Verhalten anhand des emotionalen Feedbacks durch den Spieler anpassen.

Im Kapitel 4 wurden die Begrifflichkeiten künstliche Intelligenz und KI-Agent definiert. Es wurde beschrieben, mit welchen Algorithmen KI-Agenten in Spielen umgesetzt werden. Anhand dieser Erkenntnisse wurde eine Architektur für den Agenten entwickelt (siehe 5). Die Architektur basiert auf der C4-Architektur (siehe 5.3) und verwendet Behavior Trees (siehe 5.8) für die Entscheidungsfindung.

Basierend auf der Agentenarchitektur und den Ergebnissen der Voruntersuchung wurde im Kapitel 6 ein Spielkonzept vorgestellt und implementiert. Der Spieler muss sich in einem Schleichspiel verschiedensten Herausforderungen stellen, hierfür ist er auf die Zusammenarbeit mit einem KI-Begleiter angewiesen. Der Begleiter vertraut dem Spieler abhängig von den gezeigten Emotionen. Über das Vertrauen wird die Kooperationsbereitschaft mit dem Spieler entschieden. Der Spieler kann mit Hilfe von Emotionen das Verhalten des Begleiters manipulieren.

Anhand der durchgeführten Versuche wurde der Prototyp getestet und die emotionale Interaktion validiert (siehe 7). Die Ergebnisse der Versuche zeigten, dass das Spielprinzip positiv angenommen wurde und die Implementation fehlerfrei funktionierte. Die emotionale Interaktion mit dem Begleiter wurde überwiegend positive empfunden. Zwei von drei Probanden hatten den Eindruck, dass dieser adäquat auf ihre Emotionen reagierte und eine sinnvolle Ergänzung der Spielmechaniken ist. Der Einfluss auf die Immersion wurde durch die Probanden teilweise bestätigt.

Es wurde eine Plattform geschaffen, die es ermöglicht emotionale Interaktionsmöglichkeiten mit und ohne Begleiter zu testen und deren Auswirkungen auf den Spieler zu untersuchen. Die Emotionserkennung lässt sich komplett deaktivieren, sodass eine Untersuchung mit Kontrollgruppen durchgeführt werden kann. Einzelne Interaktionen lassen sich separat ein- und ausschalten, so lassen sich die Interaktionen auch isolierte Untersuchungen. Nun können weiterführende Untersuchungen mit einer größeren Anzahl an Probanden aussagekräftige Ergebnisse erzielen.

8.2. Ausblick

Betrachten wir mögliche Erweiterungen und Bereiche, die eine fortführende Untersuchung benötigen, können wird drei Aspekte identifizieren:

1. Der KI-Agent
2. Das Spiel
3. Die emotionalen Interaktionen

Maschinelles Lernen ist ein Bereich der künstlichen Intelligenz, der oft nur wenig Beachtung im Gaming findet. Der KI-Agent könnte zu einem lernenden Agenten erweitert werden, der sich anhand des Verhaltens und der gezeigten Emotionen des Spielers an dessen individuellen Bedürfnisse anpasst. In dieser Arbeit wurde sich auf die reaktiven Entscheidungsfindungsalgorithmen konzentriert. Eine Untersuchung von planenden Algorithmen könnten die Vor- und Nachteile zu den reaktiven Algorithmen herausarbeiten.

Das Spiel kann um weitere emotionale und nicht emotionale Interaktionen erweitert werden. Zuerst sollte jedoch der Prototyp anhand des Feedback aus den Vorversuchen verbessert werden. Das Vertrauen könnte durch eine Emotionssimulation wie Wasabi (siehe Becker-Asano (2008)) ersetzt werden. Die Entscheidungsfindung könnte dann durch den affektiven Zustand des Agenten differenziertere Entscheidungen treffen. Für den Prototyp wurden passive als auch aktive emotionale Interaktionen implementiert. Über weitere Untersuchungen sollte bestimmt werden, welche Arten der emotionalen Interaktion von Spielern angenommen werden, sowie der Einfluss des Kontext und die Häufigkeit der Interaktion. Die Arbeit von Becker-Asano (2008) konnte bereits aufzeigen das sich durch eine emotionale Interaktion mit einem Avatar die Glaubwürdigkeit des Charakters und die Akzeptanz durch Benutzer erhöht.

Die Emotionserkennung vollzieht zur Zeit im Bereich Gaming einen ähnlichen Werdegang wie bereits die Gestenerkennung. Es besteht das Risiko, das diese in der Bedeutungslosigkeit verschwindet. Damals wurde es versäumt, Anwendungsbereiche zu finden, die einen wirklichen Mehrwert bedeuten. Die meisten Anwendungen kamen nie über den Status einer technischen Spielerei hinaus.

Weiterführende Versuche werden zeigen, inwieweit und in welche Arten von Spielen Emotionserkennung sinnvoll eingesetzt werden kann. Des Weiteren sollten die derzeitigen technischen Limitierungen der Emotionserkennung mit Kameras aufgezeigt werden. Anhand dieser Untersuchungen wird sich zeigen, ob die Emotionserkennung das Potenzial hat, einen ähnlich großen Einfluss auf die Spiellandschaft zu haben, wie damals der Wechsel von der 2D zu 3D Grafiken.

A. Versuche

1. Angaben zur Person

Alter:

Geschlecht:

Männlich Weiblich

2. Wie oft Spielen Sie oft Videospiele?

Nie **Manchmal** **Oft**

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

3. Wenn Sie ein Videospiel spielen, wie konzentriert sind Sie dabei auf das Spiel?

Leicht ablenkbar **Voll auf das Spiel konzentriert**

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

4. Welche Art von Spiel bevorzugen Sie?

Bitte maximal eine Antwort.

Ego-Shooter **Action/Adventure** **Strategie** **MMORPG** **RPG** **Simulationen** **Kampfspiele** **Puzzelspiele** **Fahrsimulationen**

5. Wie wichtig ist Ihnen Sound in Videospielen?

Nicht wichtig **Neutral** **Sehr wichtig**

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

Abbildung A.1.: Fragebogen 1 Teil 1

6. Wie wichtig ist Ihnen die Grafik in Videospielen?

Nicht wichtig — Neutral — Sehr wichtig

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

7. Wie wichtig ist ihnen die Geschichte in Videospielen?

Nicht wichtig — Neutral — Sehr wichtig

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

8. Wie wichtig sind Ihnen die Charaktere in Videospielen?

Nicht wichtig — Neutral — Sehr wichtig

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

9. Wie wichtig sind Ihnen die Spielmechaniken in Videospielen?

Nicht wichtig — Neutral — Sehr wichtig

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

10. Was fesselt Sie am meisten an einem Spiel?

Bitte maximal eine Antwort.

Sound ○ Grafik ○ Geschichte ○ Charaktere ○ Spielmechaniken ○

Abbildung A.2.: Fragebogen 1 Teil 2

1. Wie hat Ihnen das Spiel gefallen?

Nicht gut				Neutral					Gut
○	○	○	○	○	○	○	○	○	○

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

2. Was hat Ihnen an dem Spiel gefallen?

Verfassen Sie die Antwort in eigenen Worten.

3. Was hat Ihnen nicht an dem Spiel gefallen?

Verfassen Sie die Antwort in eigenen Worten.

4. Hatten Sie den Eindruck das der Begleiter adäquat auf Ihre Emotinen reagiert?

Stimme nicht zu									Stimme zu
○	○	○	○	○	○	○	○	○	○

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

Abbildung A.3.: Fragebogen 2 Teil 1

5. Wie empfanden Sie die emotionale Interaktion mit dem Begleiter?

Schlecht **Gut**

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

6. Hat die emotionale Interaktion die Spielmechaniken auf sinnvoll ergänzt?

Stimme nicht zu **Stimme zu**

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

7. Hatte die emotionale Interaktion mit dem Begleiter Einfluss auf Ihre Immersion?

Stimme nicht zu **Stimme zu**

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

Auf einer Skala von 1 bis 10, wobei 1 der kleinste und 10 der höchste Wert ist.

Abbildung A.4.: Fragebogen 2 Teil 2

B. Emotion Bike

(a) Level 1 (b) Level 2

Abbildung B.1.: Level 1 und 2

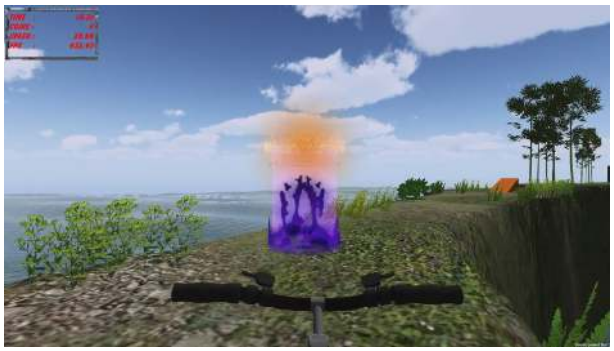

Abbildung B.2.: Level 3

(a) Level 4 (b) Level 5

Abbildung B.3.: Level 4 und Level 5

C. Implementation

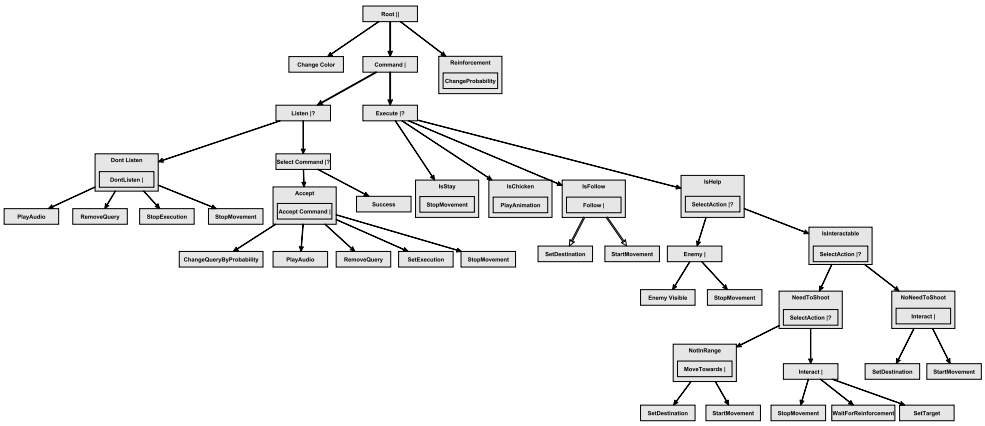

Abbildung C.1.: Kompletter Behavior Tree des Begleiter

Abbildung C.2.: Level Abschnitt: Level Start

Abbildung C.3.: Level Abschnitt: GefÃ¤ngnis

Abbildung C.4.: Level Abschnitt: Wachen bekÃ¤mpfen

Abbildung C.5.: Level Abschnitt: Druckplatten

Abbildung C.6.: Level Abschnitt: Laserzaun

Abbildung C.7.: Level Abschnitt: Weg durch Kisten blockiert

Literaturverzeichnis

[62 2017] 62, Transregio: *Eine Companion-Technologie fÃ¼r kognitive technische Systeme*. online. 2017. – URL http://sfb-trr-62.de/. – Zuletzt abgerufen: 11.02.2017

[Affectiva 2017] Affectiva: *Emotion Recognition Software and Analysis*. Online. 2017. – URL http://www.affectiva.com/. – Zuletzt abgerufen: 17.05.2017

[Becker-Asano 2008] Becker-Asano, Christian: *WASABI: Affect simulation for agents with believable interactivity*. Bd. 319. IOS Press, 2008

[Bernin u. a. 2017] Bernin, Arne ; Müller, Larissa ; ; Ghose, Sobin ; Luck, Kai von ; Wang, Qi ; Grecos, Christos ; Vogt, Florian: Towards More Robust Automatic Facial Expression Recognition in Smart Environments. In: *Proceeding of the PErvasive Technologies Related to Assistive Environments (PETRA) conference, Rhodos* ACM (Veranst.), 2017. – ISBN 978-1-4503-5227-7

[Biundo u. a. 2016] Biundo, Susanne ; Höller, Daniel ; Schattenberg, Bernd ; Bercher, Pascal: Companion-technology: An overview. In: *KI-Künstliche Intelligenz* 30 (2016), Nr. 1, S. 11–20

[DeepMind 2017] DeepMind, Technologies L.: *DeepMind and Blizzard to release StarCraft II as an AI research environment*. Online. 2017. – URL https://deepmind.com/blog/deepmind-and-blizzard-release-starcraft-ii-ai-research-environment/. – Zuletzt abgerufen: 27.05.2017

[Ekman 1992] Ekman, Paul: An argument for basic emotions. In: *Cognition & emotion* 6 (1992), Nr. 3-4, S. 169–200

[Ekman und Friesen 1977] Ekman, Paul ; Friesen, Wallace V.: Facial action coding system. (1977)

[Erman u. a. 1980] Erman, Lee D. ; Hayes-Roth, Frederick ; Lesser, Victor R. ; Reddy, D R.: The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. In: *ACM Computing Surveys (CSUR)* 12 (1980), Nr. 2, S. 213–253

[Foundation] Foundation, Apache S.: *Apache ActiveMQ*. – URL http://activemq.apache.org/

[Freesound 2017] Freesound: *freesound.org*. Online. 05 2017. – URL https://freesound.org/. – Zuletzt abgerufen: 10.05.2017

[Gilleade u. a. 2005a] Gilleade, Kiel ; Dix, Alan ; Allanson, Jen: Affective videogames and modes of affective gaming: assist me, challenge me, emote me. (2005)

[Gilleade u. a. 2005b] Gilleade, Kiel M. ; Dix, Alan ; Allanson, Jen: Affective videogames and modes of affective gaming: assist me, challenge me, emote me. In: *In The 2005 International Conference on Changing Views: Worlds in Play*, 2005, S. 547–554

[Holbrook und Cain 2006] Holbrook, H. ; Cain, B.: Source-Specific Multicast for IP. RFC 4607., 2006

[Huang u. a. 2015] Huang, Xiangyang ; Zhang, Shudong ; Shang, Yuanyuan ; Zhang, Weigong ; Liu, Jie: Creating Affective Autonomous Characters using Planning in Partially Observable Stochastic Domains. In: *IEEE Transactions on Computational Intelligence and AI in Games* (2015)

[Hume 2012] Hume, David: Emotions and moods. In: *Organizational behavior* (2012), S. 258–297

[Isla u. a. 2001] Isla, Damian ; Burke, Robert ; Downie, Marc ; Blumberg, Bruce: A Layered Brain Architecture for Synthetic Creatures. In: *IJCAI*, 2001, S. 1051–1058

[Johansson und Dell'Acqua 2012] Johansson, A. ; Dell'Acqua, P.: Comparing behavior trees and emotional behavior networks for NPCs. In: *Computer Games (CGAMES), 2012 17th International Conference on*, July 2012, S. 253–260

[Kanjo u. a. 2015] Kanjo, Eiman ; Al-Husain, Luluah ; Chamberlain, Alan: Emotions in context: examining pervasive affective sensing systems, applications, and analyses. In: *Personal and Ubiquitous Computing* 19 (2015), Nr. 7, S. 1197–1212

[Konami 1999] Konami, Digital E.: *Metal Gear Solid*. Online. 1999. – URL https://www.konami.com/games/eu/de/products/mgs/. – Zuletzt abgerufen: 15.05.2017

[Littlewort u. a. 2011] Littlewort, Gwen ; Whitehill, Jacob ; Wu, Tingfan ; Fasel, Ian ; Frank, Mark ; Movellan, Javier ; Bartlett, Marian: The computer expression recognition

toolbox (CERT). In: *Automatic Face & Gesture Recognition and Workshops* IEEE (Veranst.), 2011, S. 298–305

[Lobel u. a. 2016] Lobel, Adam ; Gotsis, Marientina ; Reynolds, Erin ; Annetta, Michael ; Engels, Rutger C. ; Granic, Isabela: Designing and utilizing biofeedback games for emotion regulation: The case of nevermind. In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* ACM (Veranst.), 2016, S. 1945–1951

[Marzinotto u. a. 2014] Marzinotto, A. ; Colledanchise, M. ; Smith, C. ; Ogren, P.: Towards a unified behavior trees framework for robot control. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, May 2014, S. 5420–5427

[Microsoft 2001] Microsoft: *Farewell Clippy: What's Happening to the Infamous Office Assistant in Office XP*. online. 2001. – URL https://news.microsoft.com/2001/04/11/farewell-clippy-whats-happening-to-the-infamous-office-assistant-in-#sm.00015yyebz3l9dt610rhseau3uiak. – Zuletzt abgerufen: 11.02.2017

[Müller u. a. 2016] Müller, Larissa ; Bernin, Arne ; Ghose, Sobin ; Gozdzielewski, Wojtek ; Wang, Qi ; Grecos, Christos ; Luck, Kai von ; Vogt, Florian: Physiological Data Analysis for an Emotional Provoking Exergame. (2016)

[Müller u. a. 2015] Müller, Larissa ; Zagaria, Sebastian ; Bernin, Arne ; Amira, Abbes ; Ramzan, Naeem ; Grecos, Christos ; Vogt, Florian: *EmotionBike: A Study of Provoking Emotions in Cycling Exergames*. S. 155–168. In: Chorianopoulos, Konstantinos (Hrsg.) ; Divitini, Monica (Hrsg.) ; Baalsrud Hauge, Jannicke (Hrsg.) ; Jaccheri, Letizia (Hrsg.) ; Malaka, Rainer (Hrsg.): *Entertainment Computing - ICEC 2015: 14th International Conference, ICEC 2015, Trondheim, Norway, September 29 - Ocotober 2, 2015, Proceedings*. Cham : Springer International Publishing, 2015. – URL http://dx.doi.org/10.1007/978-3-319-24589-8_12. – ISBN 978-3-319-24589-8

[Nacke u. a. 2011] Nacke, Lennart E. ; Kalyn, Michael ; Lough, Calvin ; Mandryk, Regan L.: Biofeedback game design: using direct and indirect physiological control to enhance game interaction. In: *Proceedings of the SIGCHI conference on human factors in computing systems* ACM (Veranst.), 2011, S. 103–112

[Negini u. a. 2014] Negini, Faham ; Mandryk, Regan L. ; Stanley, Kevin G.: Using affective state to adapt characters, NPCs, and the environment in a first-person shooter game. In: *Games Media Entertainment (GEM), 2014 IEEE* IEEE (Veranst.), 2014, S. 1–8

[Ogren 2012] Ogren, Petter: Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees. In: *AIAA Guidance, Navigation and Control Conference, Minneapolis, MN*, 2012

[Parnandi u. a. 2013] Parnandi, Avinash ; Son, Youngpyo ; Gutierrez-Osuna, Ricardo: A control-theoretic approach to adaptive physiological games. In: *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on* IEEE (Veranst.), 2013, S. 7–12

[Picard 1999] Picard, Rosalind W.: Affective Computing for HCI. In: *HCI (1)*, 1999, S. 829–833

[Picard 1995] Picard, Rosalind W.: Affective computing. (1995). – URL http://hd.media.mit.edu/tech-reports/TR-321.pdf. – Zuletzt abgerufen: 10.06.2017

[Rabin 2002] Rabin, Steve: *AI Game Programming Wisdom*. Rockland, MA, USA : Charles River Media, Inc., 2002. – ISBN 1584500778

[Reynolds 2016] Reynolds, Erin: *Nevermind*. 2016. – URL http://www.NevermindGame.com. – Zuletzt abgerufen: 03.03.2017

[Russell u. a. 2003] Russell, Stuart J. ; Norvig, Peter ; Canny, John F. ; Malik, Jitendra M. ; Edwards, Douglas D.: *Artificial intelligence: a modern approach*. Bd. 2. Prentice hall Upper Saddle River, 2003

[Unity 2017a] Unity, Technologies: *Execution Order of Event Functions*. Online. 05 2017. – URL https://docs.unity3d.com/Manual/ExecutionOrder.html. – Zuletzt abgerufen: 10.05.2017

[Unity 2017b] Unity, Technologies: *NavMesh Overview*. Online. 2017. – URL https://docs.unity3d.com/uploads/Main/NavMeshOverview.svg. – Zuletzt abgerufen: 21.05.2017

[Unity 2017c] Unity, Technologies: *Unity3d Asset Store*. Online. 05 2017. – URL https://www.assetstore.unity3d.com/en/. – Zuletzt abgerufen: 10.05.2017

[Unity 2017d] Unity, Technologies: *Unity3d Editionen Overview*. Online. 05 2017. – URL https://store.unity.com/. – Zuletzt abgerufen: 10.05.2017

[Vachiratamporn u. a. 2014] Vachiratamporn, Vanus ; Moriyama, Koichi ; Fukui, Ken-ichi ; Numao, Masayuki: An implementation of affective adaptation in survival horror games. In:

Computational Intelligence and Games (CIG), 2014 IEEE Conference on IEEE (Veranst.), 2014, S. 1–8

[Wymann u. a. 2017] Wymann, Bernhard ; EspiÃ©, Eric ; Guionneau, Christophe: *TORCS, the Open Racing Car Simulator*. Online. 05 2017. – URL http://torcs.sourceforge.net/index.php. – Zuletzt abgerufen: 27.05.2017

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 16. Juni 2017 Sebastian Zagaria