Bachelor-Arbeit

Entwicklung eines menschenähnlichen Agenten für ein Computerspiel mit dem Ziel, den Turing-Test zu bestehen

Betreuer: Prof. Dr.-Ing. Sanaz Mostaghim

Vorgelegt von: Xenija Neufeld
Abgabedatum: 22. Oktober 2014

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Pseudocodeverzeichnis

Formelverzeichnis

Glossar

- Bot – (von englisch “robot“) ein Agent, der in einem Computerspiel automatisiert spielt
- Deathmatch – Spielmodus, in dem das Ziel der Spieler darin besteht, möglichst viele Spielpunkte durch Töten anderer Spieler zu erhalten
- GameBots (auch GameBots 2004) – Modifikation des Spiels Unreal Tournament 2004, welche es ermöglicht, Daten aus dem Spiel auszulesen und Befehle an das Spiel zu senden
- Items – aufsammelbare Gegenstände, wie z.B. Waffen oder Munition
- Map – virtuelle Spielwelt
- Pogamut – Schnittstelle zwischen GameBots und der Programmierumgebung, welche eine große Bibliothek an Methoden anbietet
- Respawn – Erstehen eines Spielcharakters nach seinem simulierten Tod
- UT2K4 – Unreal Tournament 2004

1. Einleitung

Während der letzten Jahrzehnte ist die Videospielindustrie stark gewachsen. Computerspiele sind zu einem der wichtigsten Unterhaltungsmedien geworden. Ihre Qualität wird immer weiter verbessert, sodass viele von ihnen heute sehr realistische Simulationsumgebungen darstellen. So werden die Spiele immer öfter auch aus der wissenschaftlichen Sicht betrachtet. Bestimmte Situationen können schneller und günstiger in Spielen simuliert und neue Methoden getestet werden, als in anderen Umgebungen.

Um den Realismus in den Spielen zu steigern, versuchen Entwickler nicht nur die grafischen Qualitäten zu steigern, sondern auch die in den Spielen benutzten Methoden der künstlichen Intelligenz (KI) zu verbessern. Diese ist mittlerweile ein unverzichtbarer Teil vieler Videospiele. In modernen Spielen wird sie eingesetzt um Verhalten einzelner Spielcharaktere und –Einheiten oder ganzer Gruppen zu simulieren. Wenn diese jedoch ein für die Spieler nicht nachvollziehbares Verhalten aufweisen, verlieren viele Spieler an Motivation, das Spiel weiter zu spielen. Um gute Spielerfahrungen zu gewährleisten und die Immersion der Spieler zu steigern, ist es also sehr wichtig, dass das simulierte Verhalten besonders „glaubhaft“ erscheint. Werden Methoden der künstlichen Intelligenz eingesetzt, um das Verhalten von menschenähnlichen Spielcharakteren zu erzeugen, so bedeutet es, dass ihr Verhalten „menschlich“ sein soll.

„Menschlichkeit“ ist jedoch eine sehr subjektive Eigenschaft, die nicht gemessen werden kann. Um es trotzdem zu versuchen, wurde eine Variante des Turing-Test entwickelt, bei der statt der „Intelligenz“ einer Maschine die „Menschlichkeit“ eines Computerspiel-Charakters gemessen wird. Diese Variante wird in dem BotPrize-Wettbewerb angewandt. Als Testumgebung wird hierbei das Computerspiel Unreal Tournament 2004 verwendet. Hier können auf unterschiedliche Weisen entwickelte Agenten gegen mehrere menschliche Spieler spielen, wonach ihr Verhalten in Bezug auf die „Menschlichkeit“ beurteilt wird. Kann ein Agent davon überzeugen, dass er ein Mensch ist, so besteht er den modifizierten Turing-Test und gewinnt den BotPrize-Wettbewerb. Im Rahmen dieser Arbeit wurde ein solcher Agent entwickelt und nahm unter dem Namen OvGUBot (genannt nach der Otto-von-Guericke-Universität) teil.

1.1 Motivation

Obwohl der BotPrize-Wettbewerb bereits mehrere Male stattfand und viele Wissenschaftler verschiedene Agenten entwickelt haben, konnten nur zwei von ihnen den Turing-Test bestehen. Es wurden dabei sehr unterschiedliche Methoden ausprobiert, die ähnlich gute Ergebnisse bei den Wettbewerben erzielten. Das Verhalten der meisten Agenten erschien jedoch nicht „menschlich“ genug. Nach allen Wettbewerben gab es jede Menge an Feedback von Entwicklern der Agenten und Beobachtern der Spiele. All die Kommentare zeigten, wie unterschiedlich die Vorstellung von der „Menschlichkeit“ sein kann, aber auch auf welche Aspekte dabei im Allgemeinen geachtet wird. In dieser Arbeit soll ein Agent entwickelt werden, der an dem modifizierten Turing-Test teilnehmen und ihn bestehen sollte. Durch die Teilnahme könnten weitere Kriterien für die „Menschlichkeit“ eines Spielcharakters gesammelt werden. Außerdem kann dadurch untersucht werden, wie gut sich der BotPrize-Wettbewerb als Turing-Test eignet und ob die dabei entwickelten Agenten tatsächlich die Immersion der Spieler steigern können.

1.2 Ziele

Das primäre Ziel dieser Arbeit besteht darin, einen Agenten zu entwickeln, welcher an dem BotPrize-Wettbewerb in dem Jahr 2014 teilnimmt. Dabei ist es jedoch weniger wichtig, den modifizierten Turing-Test in der Form, in der er stattfindet, tatsächlich zu bestehen. Unter Berücksichtigung der Berechnung der „Menschlichkeit“ bei dem Test und der Ergebnisse

früherer Wettbewerbe, besteht das Ziel viel mehr darin, verhältnismäßig kleine Unterschiede in den Testergebnissen im Vergleich zu den menschlichen Spielern zu erreichen. Dies liegt vor allem daran, dass während der vergangenen Wettbewerbe selbst die Menschen den Test nicht immer bestehen konnten. So ist bereits ein Ergebnis, welches um nicht mehr als 10% von einem menschlichen Spieler abweicht, als ein zufriedenstellendes Resultat zu betrachten.

Um das Hauptziel zu erreichen, müssen einige konkrete Teilziele definiert werden. Damit der im Rahmen dieser Arbeit entwickelte Agent den Test bestehen kann, muss er ein „menschliches“ Verhalten aufweisen können. „Menschlichkeit“ wird an dieser Stelle und in diesem Zusammenhang definiert als die Fähigkeit, situationsbedingt zu handeln. Das heißt, der Agent sollte in der Lage sein, das Geschehen um ihn herum beobachten zu können und sich danach für entsprechende Aktionen entscheiden. Er sollte sich problemlos in der Gegend bewegen können und für unterschiedliche Interaktionen mit der Umgebung und anderen Spielern bereit sein. Wichtig ist dabei aber auch, dass das Verhalten nicht immer vorhersehbar sein sollte, denn Menschen handeln oft unberechenbar. Die wichtigste Eigenschaft, die einen Menschen „menschlich“ macht, zeichnet sich jedoch dadurch aus, dass sie Fehler machen. So sollte auch der Agent in manchen Fällen nicht unbedingt die günstigste Aktion ausführen, nicht den besten Weg wählen oder auch eine verzögerte Reaktion zeigen. Weiterhin sollen mit Hilfe des Turing-Tests weitere Eigenschaften herauskristallisiert werden.

1.3 Struktur der Arbeit

Nachdem in diesem ersten Kapitel der Arbeit eine grobe Einführung in das zugrundeliegende Thema gegeben und die Ziele definiert wurden, werden bestimmte Grundlagen in Kapitel 2 detaillierter erläutert. Hierfür wird dem Leser zuerst die Idee des originalen Turing-Tests beschrieben. Anschließend wird das Spielprinzip und die Besonderheiten des Spiels Unreal Tournament 2004, für welches der OvGUBot entwickelt wird, illustriert. Anschließend wird dem Leser ein kurzer Einblick in die gegebene Entwicklungsumgebung und die Verbindungen zwischen ihren wichtigsten Teilen GameBots und Pogamut gewährt. Das Kapitel schließt mit der detaillierten Beschreibung des BotPrize-Wettbewerbs ab.

In dem Kapitel 3 werden bisherige Arbeiten genauer untersucht und ihre Ergebnisse bei den bereits stattgefundenen BotPrize-Wettbewerben gezeigt. Die Agenten werden dabei nach ihren Architekturarten gruppiert. Hierbei wird zwischen regelbasierten Architekturen und solchen, die mit Hilfe von neuronalen Netzen oder evolutionären Algorithmen entwickelt wurden, unterschieden

Anschließend wird in dem Kapitel 4 die Entwicklung der OvGUBots beschrieben, Hierfür werden zuerst die wichtigsten Entscheidungen erläutert und begründet. Danach werden die wichtigsten Bestandteile der Architektur näher beschrieben und einige Beispiele von Spielsituationen gebracht,

Daraufhin wird in dem Kapitel 5 die Evaluierung des Agentenverhaltens beschrieben. Diese besteht aus zwei unterschiedlichen Teilen. Zuerst wird die Durchführung des BotPrize-Wettbewerbs beschrieben und die dabei erzielten Ergebnisse erläutert. Anschließend werden einige Kritikpunkte genannt. In dem zweiten Teil der Evaluierung werden einige lokal durchgeführte Testspiele beschrieben und auch die Ergebnisse dieser Tests gezeigt und weitere Anmerkungen genannt. Abschließend werden die Erkenntnisse dieser Arbeit in dem Kapitel 6 zusammengefasst, Schlussfolgerungen gezogen und weiterführende Aufgaben genannt.

2 Grundlagen

Bevor die Ideen und die Realisation dieser Arbeit beschrieben werden, ist es wichtig die zugrundeliegenden Komponenten zu kennen und ihre Funktionsweisen zu verstehen. So werden die wichtigsten Aspekte und Elemente in diesem Kapitel beschrieben.

2.1 Turing Test

1950 stellte der britische Wissenschaftler Alan Turing die Frage „können Maschinen denken?“. Um sie beantworten zu können, schlug er eine modifizierte Version des Imitationsspiels vor. Bei dem Imitationsspiel sollte ursprünglich eine Person C sich mit zwei anderen Personen A und B (eine Mann und eine Frau) unterhalten und anhand der Unterhaltung erkennen, welches Geschlecht jeweils A und B haben. Personen A und B sollten sich dabei in einem anderen Raum befinden als Person C und die Unterhaltung sollte über ein Schreibgerät stattfinden, sodass C die anderen Personen weder hören noch sehen konnte.

In der veränderten Version des Spiels schlug Alan Turing vor A oder B durch eine Maschine zu ersetzen. Somit sollte C anhand der Unterhaltung mit einem Menschen und einer Maschine erkennen können, welcher der Gesprächspartner die Maschine ist (vgl. Turing 1950). Sollte C nicht in der Lage sein, die Maschine von dem Menschen zu unterscheiden, könnte die Maschine somit als „menschlich“ gelten. Dieser Test ist heute als der „Turing-Test“ bekannt.

Im späteren Verlauf wurden diverse Variationen des Turing-Tests vorgestellt, wie zum Beispiel der „Loebner Prize“ (vgl. Loebner 2014), bei dem die Chateingaben aller Beteiligter in reeller Zeit sichtbar waren und somit auch alle Tippfehler und Korrekturen gesehen werden konnten. Eine weitere Version des Turing-Tests ist der BotPrize-Wettbewerb, der weiter in dieser Arbeit beschrieben wird.

2.2 Unreal Tournament 2004

Unreal Tournament 2004 (UT2K4) ist ein First-Person-Shooter, der 2004 als Teil der Unreal-Spielserie erschien. Genau wie in seinem Vorgänger Unreal Tournament 2003 hat UT2K4 einen Einzelspieler-Modus, zielt jedoch hauptsächlich auf den Mehrspieler-Modus ab. Die Spielwelt wird dabei von einem Server simuliert und die Spieler können sich als Clients mit dem Server verbinden. Jeder Spieler kann einen Server starten und diesen für ein lokales Spiel oder über eine Internetverbindung freigeben. Das Spiel hat mehr Waffenarten und Welten (Maps) als seine Vorgänger und bietet mehrere online Spielmodi. Die Maps stellen Welten mit unterschiedlichen (physikalischen) Eigenschaften dar. Manche der Welten sind Gebäude, andere sind begrenzte Flächen in einem Wald und andere wiederum stellen eine „Insel“ im Weltall dar, so, dass die simulierte Gravitation niedriger als auf der Erde ist. Die Spielmodi unterscheiden sich danach, ob in einem Team gespielt wird (zum Beispiel „Capture the Flag“), oder jeder Spieler nur für sich selbst verantwortlich ist. Einer solcher Spielmodi ist „Deathmatch“. Da dieser Modus für diese Arbeit relevant ist, wird er genauer beschrieben.

Bei einem Deathmatch-Spiel können mehrere Spieler auf einer Map nach dem Prinzip „jeder gegen jeden“ spielen. Jeder Spieler bekommt einen Punkt, wenn er einen anderen Charakter im Spiel tötet. Entsprechend verliert ein Spieler Punkte, wenn er von jemandem getötet wird, oder selbst (zum Beispiel durch seine eigene Granate) stirbt. Nach dem simulierten Tod erscheint der Spieler mit voller Gesundheit wieder im Spiel (wird „respawned“). Die Spielrunde ist dabei

entweder durch die Zeit oder durch die zu erreichende Punktanzahl begrenzt. Am Ende einer Spielrunde gewinnt derjenige, der die meisten Punkte (in der vorgegebenen Zeit) erzielt.

Jeder Spieler hat dabei die Möglichkeit, unterschiedliche Gegenstände (Items) auf der Map einzusammeln. Diese können sowohl Waffen, als auch Munition für die Waffen sein. Außerdem gibt es Items, die dem Spieler einen Vorteil gegenüber anderen Spielern gewähren (zum Beispiel Schild, „Double Damage“ oder Adrenalinkapseln) oder ihn heilen. Diese Gegenstände befinden sich immer an denselben Stellen einer Map und erscheinen einige Zeit später, wenn sie aufgehoben werden.

Der Screenshot in Abbildung 1 zeigt, wie das Spiel aus der Sicht des Spielers aussieht. In der Mitte des Bildes ist ein Gegner zu sehen. Im oberen Teil des Bildschirms kann der Spieler links die Zeit bis zum Ablauf der Spielrunde sehen und da drunter seine Punktzahl. Rechts sieht der Spieler die Anzahl der gesammelten Adrenalinkapseln. Im unteren Teil des Bildschirms ist mit dem blauen Kreuz die Gesundheit des Spielcharakters abgebildet und daneben alle möglichen Waffenarten. Die Waffen, die der Spieler noch nicht besitzt, sind schwarz ausgeblendet und die Waffe, mit der der Spieler gerade ausgerüstet ist, wird gelb hervorgehoben. Außerdem ist die Waffe selbst im Bild zu sehen und wird beim Schießen animiert. Die für die ausgewählte Waffe vorhandene Munition wird im rechten Teil des Bildes dargestellt. Jede Waffe hat zwei unterschiedliche Schussarten, die der Spieler jeweils mit der linken oder der rechten Maustaste ausführen kann.

Abbildung 1: Screenshot Unreal Tournament 2004

Um dem Spieler das Zielen zu erleichtern, wird in der Mitte des Bildschirms ein Fadenkreuz dargestellt. Jeder Spieler kann für seinen Spielcharakter einen Namen und einen Avatarkörper auswählen. Die Namen der Gegenspieler werden beim Anvisieren im oberen Teil des Bildschirms dargestellt.

Da die Zeit für eine Spielrunde begrenzt ist, die Maps im allgemein nicht sehr groß sind, im Durchschnitt bis zu 20 Spieler gleichzeitig spielen können und die Anzahl der Items auf einer Map begrenz ist, laufen alle Geschehnisse im Spiel sehr schnell ab. Je mehr Spieler spielen, desto größer ist die Wahrscheinlichkeit einen von ihnen auf der Karte zu treffen. Hierfür sollte der Spieler möglichst gut vorbereitet sein (genug Ausrüstung haben) um den Gegenspieler zu besiegen und somit einen Punkt zu kriegen. Im Normalfall dauert es weniger als eine halbe Minute, bis ein Gegner erscheint und genau so lange bis entweder der Gegnercharakter oder die Spielerfigur „getötet“ wird. Das heißt, jeder Spieler ist beinahe ständig in Bewegung auf der Suche nach weiteren Items oder Gegnern und braucht schnelle Reaktionszeiten im Falle eines Zusammentreffens wodurch das ganze Spiel sehr hektisch erscheint.

Außer dem offiziellen Spiel gibt es sehr viele Modifikationen (Mod) des Spiels, die teilweise von den Spielern selbst geschrieben wurden. Dies ist möglich, da der Quellcode des Spiels bis auf einige Ausnahmen frei verfügbar ist und somit modifiziert werden kann. Mit Hilfe eines dieser Mods ist es auch möglich Agenten (Bots) mit dem Spielserver zu verbinden und spielen zu lassen. Die genaue Beschreibung der dabei gebrauchten Architektur wird im folgenden Kapitel beschrieben.

2.3 Entwicklungsumgebung

Angefangen als ein Projekt an der Universität von Südkalifornien wurde die Modifikation des Spiels Unreal Tournament 2000 „GameBots“ entwickelt, um ein Forschungsgebiet für künstliche Intelligenz zu schaffen (vgl. Pogamut 2014). Da, wie bereits erwähnt, der Quellcode für Unreal-Spiele frei verfügbar war, war es möglich diese Modifikation in der für alle Spiele der Unreal-Serie benutzten Sprache Unrealscript zu schreiben. Diese Mod wurde und wird immer noch weiter entwickelt und erlaubt es als „GameBots 2004“ (im Folgenden kurz „GameBots“) Agenten mit dem Spiel Unreal Tournament 2004 zu verbinden. Es stellt ein Netzwerk-Text-Protokoll bereit, über welches es Nachrichten an die Spielbots sendet. Diese Nachrichten enthalten Informationen über das Spiel (z.B. Länge der Spielrunde), die Spielwelt (z.B. Positionen bestimmter Objekte) und die stattfindenden Events (z.B. Treffen eines Waffenprojektils oder Auftauchen eines Gegners). Events werden von GameBots selbst gesendet, andere Informationen können von dem Agenten angefragt werden. So kann zum Beispiel der Agent einen Strahl in die Welt schicken und bekommt von GameBots Informationen darüber, ob und in welchem Punkt der Strahl auf ein Objekt trifft. Es ist außerdem möglich einen durch GameBots mit Hilfe des A*-Algorithmus berechneten Pfad von einem Punkt in der Welt zu einem anderen zu erhalten. Desweiteren kann GameBots Anweisungen von Agenten erhalten, um z.B. die Waffe des Spielcharakters zu wechseln oder ihn zu Bewegen. Die Liste aller möglichen Nachricht kann unter dem angegeben Verweis angesehen werden (vgl. Bida 2014).

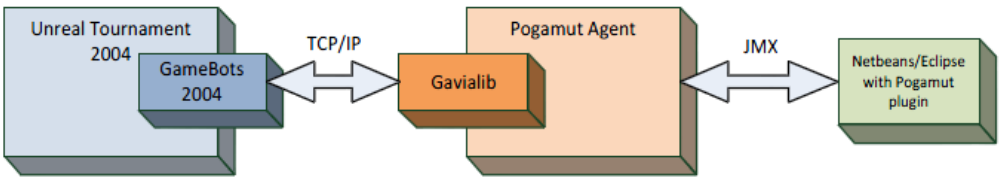

Abbildung 2: Pogamut-Architektur. In (Quelle: Muñoz 2012)

Wie in Abbildung 2 zu sehen ist, stellt GameBots als Server eine TCP/IP-Verbindung mit einer weiteren Plattform – dem Pogamut-Agenten. Pogamut wurde an der Charles Universität in Prag entwickelt und verbindet sich als Client mit GameBots. Es benutzt die Sprache UnrealScript um mit GameBots zu kommunizieren, stellt jedoch eine integrierte Entwicklungsumgebung (IDE) dar, die es erlaubt Bots in Java mit Hilfe von Eclipse[1] oder Netbeans[2] zu programmieren. Außerdem bietet es eine große Bibliothek an unterschiedlichen Methoden um Bots im Spiel zu kontrollieren. Dank dieser Bibliothek ist es Entwicklern möglich, das Verhalten eines Bots mit Hilfe unterschiedlicher Befehle zu programmieren.

Zusätzlich bietet GameBots viele Debugging-Hilfen an, die dem Entwickler direkt im Spiel visuelle Hinweise über viele Parameter des Bots und der Welt geben. So ist in Abbildung 3 der Navigationsgraph der Welt zu sehen. Dieser wird von GameBots für die Berechnung der Pfade benutzt und der Entwickler kann die Positionen und die Bezeichnungen einzelner Knoten sehen sowie die ein- und ausgehenden Kanten und überprüfen, wie sich sein Agent bewegt ohne das

[1] s. The Eclipse Foundation: Eclipse IDE https://www.eclipse.org/
[2] s. Oracle Corporation: NetBeans IDE https://netbeans.org/
[2] s. Oracle Corporation: NetBeans IDE https://netbeans.org/

Programm zu unterbrechen. Am oberen Bildschirmrand kann die Position des Charakters gesehen werden. Diese wird als ein Vektor im 3D-Raum dargestellt. Außerdem kann sich der Entwickler als ein Beobachter in der Welt bewegen und seine eigene Position anzeigen lassen.

Direkt am Charakter werden mit Hilfe von Balken die Gesundheits-, Adrenalin- und Schildwerte des Bots angezeigt. Wenn nötig, können die entsprechenden Werte außerdem durch Zahlen dargestellt werden.

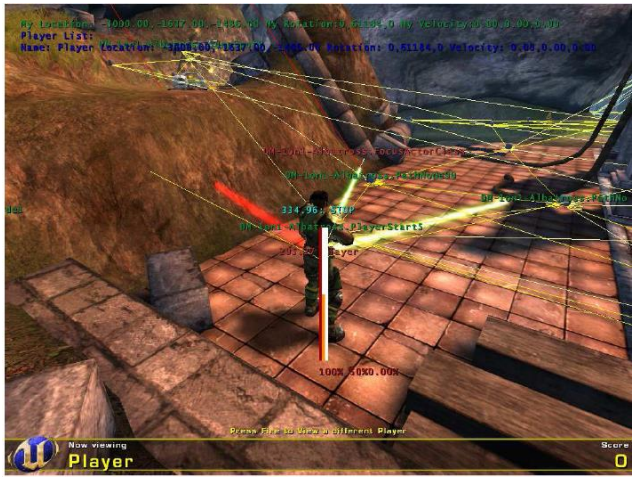

Abbildung 3: Debugging-Hilfen von GameBots 2004. In (Quelle: Bida/ Cerny / Gemrot / Brom 2012)

Auch die bereits erwähnten Strahlen können für den Entwickler sichtbar geschaltet werden, sodass nachvollziehbar wird, welche der Strahlen gerade auf Objekte treffen. Diese werden, wie in Abbildung 3 zu sehen ist, rot dargestellt. Mit Hilfe der Strahlen kann der Entwickler außerdem das Längenmaß der Unreal-Welt einschätzen. Abbildung 4 stellt zum Beispiel einen Strahl der Länge 50 „Unreal-Einheiten“ (UE) dar. Diese Maßangabe kann beim Programmieren für Distanzberechnungen benutzt werden und im Vergleich zu der Größe des Spielcharakters in Abbildung 4 kann abgeschätzt werden, wie lang ein „Unreal-Meter“ ist.

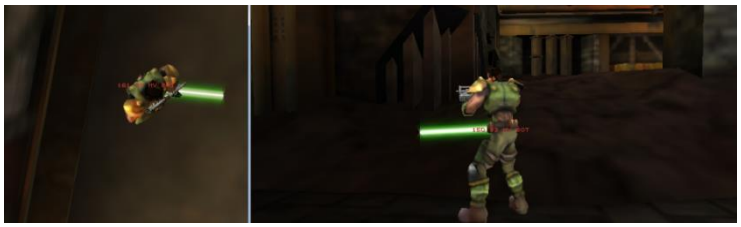

Abbildung 4: Längenmaß: Strahllänge 50 Unreal-Einheiten.

Alle Debugging-Tools von GameBots können beliebig ein- und ausgeschaltet werden, je nachdem, welche Informationen der Entwickler gerade benötigt.

Die Installation und die Verbindung zwischen UT2K4, Pogamut und der Entwicklungsumgebung sind sehr einfach gehalten. Wenn das Spiel und Netbeans (oder Eclipse) installiert sind, werden ihre Pfade auf dem Rechner bei der Installation von Pogamut automatisch gefunden und die Entsprechenden Dateiänderungen vorgenommen, beziehungsweise Bibliotheken installiert. So werden alle Komponenten verbunden und sind schnell betriebsbereit. Außerdem bieten die Entwickler von Pogamut mehrere Tutorials an, in denen grundlegende Befehle erklärt werden und der Quellcode von Beispielbots vorhanden ist, wodurch der Einstieg in die Entwicklung erleichtert wird[3].

[3] Die Tutorials können unter folgendem Verweis gefunden werden: Kadlec/Gemrot/Bida/Havlicek/Pibil/Zemcak/ Vansa 2014

2.4 BotPrize

Seit dem Vorschlag von Alan Turing war der Turing-Test für viele Wissenschaftler eine Herausforderung, aber auch eine Motivation weitere Varianten des Tests zu entwickeln. Es wurde erkannt, dass Computerspiele eine sehr gute Anwendungs- und Testumgebung für die Forschung im Bereich der künstlichen Intelligenz (KI) anbieten. Außerdem wurde eingesehen, dass „gute" KI in Spielen, die sich weder „übermenschlich", noch „vorhersehbar" verhält, die Immersion der Spieler steigert. Dabei bestand jedoch in der Spieleindustrie die Meinung, dass Computerspiel-KI sehr schwer zu entwickeln sei (vgl. Hingston 2009). So wurde 2008 zum ersten Mal der BotPrize-Wettbewerb – ein Turing-Test für Computerspiel-Bots durchgeführt mit der Hoffnung, neue Methoden zu finden, „gute" KI für Computerspiele zu entwickeln. Das Ziel des Wettbewerbs bestand darin, Agenten (Bots) nicht wie im ursprünglichen Turing-Test auf ihre Intelligenz zu testen, sondern auf ihre „Fähigkeit menschlich zu erscheinen" (vgl. Hingston 2010). Als Testumgebung wurde für den Wettbewerb das bereits beschriebene Spiel Unreal Tournament 2004 ausgewählt.

Die ursprüngliche Idee des Tests im Jahr 2008 bestand darin, jeweils einen Spieler als Richter gegen zwei Spielcharaktere – einen Menschen und einen Bot spielen zu lassen. Das heißt, es waren genau wie bei dem originalen Turing-Test immer drei Spielcharaktere an dem Test beteiligt. Diese konnten allerdings gleichzeitig mit den beiden anderen Charakteren interagieren und nicht nur jeweils paarweise, wie von Turing vorgeschlagen. Außerdem war die Aufgabe des Richters um einiges schwieriger, da hier nicht die Sprache, sondern das Verhalten der Charaktere im Allgemeinen beurteilt werden sollte. Um die Sprache nicht in Betracht zu ziehen, wurde der Spielchat ausgeschaltet. Die Namen aller Beteiligten wurden zufällig geändert und es wurde allen das gleiche Aussehen im Spiel zugewiesen. Der menschliche Spieler sollte dabei nicht versuchen, dem Richter zu helfen oder ihn bei der Bewertung zu stören, sondern versuchen das eigentliche Ziel des Spiels – das Gewinnen der Spielrunde anzustreben. Für die Bewertung der eingereichten Bots wurden mehrere Spielrunden à 10 Minuten auf unterschiedlichen Maps gespielt, so, dass jeder Richter einmal jeden Bot und jeden menschlichen Spieler getestet hat. Um genug Zeit für die Beobachtung der Spieler zu haben, wurde der Waffenschaden auf 20% des Normalen gesetzt, so, dass mehr Zeit für die Eliminierung eines Spielers nötig war. Am Ende jeder Spielrunde bewertete der Richter die Menschlichkeit der beiden Spielcharaktere auf einer Scala von „dieser Spieler ist kein menschlicher Bot" bis hin zu „dieser Spieler ist ein Mensch" (vgl. Hingston 2009). Um den Test zu bestehen, musste ein Bot dabei vier der fünf Richter von seiner Menschlichkeit überzeugen. Der Spieler und der Computer, auf dem der Agent lief, befanden sich dabei in einem anderen Raum als der Schiedsrichter. Im Jahr 2008 konnte der AMIS Bot zwei der fünf Schiedsrichter überzeugen, was jedoch nicht genug war, um den BotPrize zu gewinnen (genauere Ergebnisse werden im Kapitel 3Stand der beschrieben.)

Seit 2008 wurde der Test bereits fünf Mal durchgeführt, wobei sich die Testregeln im Zeitverlauf änderten. Im sechsten Wettbewerb im Jahr 2014, an dem der im Rahmen dieser Arbeit entwickelte OvGUBot teilnahm, sollten die menschlichen Spieler die Rolle der Schiedsrichter selbst übernehmen. In insgesamt fünf unterschiedlichen Runden sollten gleiche Anzahl an Bots und Menschen gemeinsam spielen. Das heißt, der Testverlauf war nicht mehr dem, des originalen Turing-Tests ähnlich und auch nicht der ursprünglichen Idee des BotPrizes. Hierbei konnten alle Beteiligten gleichzeitig miteinander interagieren und wurden nicht mehr in Tripel eingeteilt. Die Notwendigkeit zusätzliche Richter/Beobachter einzubeziehen entfiel 2010 mit der Entscheidung den Bewertungsprozess zum Teil des Spiels zu machen (vgl. Hingston 2010). Durch die Modifikation einer Waffe im Spiel war es möglich andere Spielcharaktere in reeller Zeit als Bot oder Mensch zu bewerten. Hierzu musste der Spieler, der nun auch ein Richter war, mit der „Link Gun" entsprechend mit der linken oder der rechten Maustaste (unterschiedliche Feuermodi) auf seinen Gegner schießen. Dabei änderte sich die Farbe des Gegnernamens in gelb (Bot), beziehungsweise blau (Mensch) (s. Abbildung 5), damit sich jeder Spieler daran erinnern konnte, ob und wie er andere Spieler bereits bewertet hat. Außerdem war es möglich seine Meinung im Verlauf der Spielrunde zu ändern und den Gegner anders zu

bewerten. Jeder Spieler hatte dabei unendliche viel Munition für die Bewertungswaffe. Die Bewertungen und die Veränderungen der Namensfarben waren nicht für andere Spieler sichtbar. Alle Bewertungen wurden dabei in einer Logdatei auf dem Spielserver dokumentiert um sie anschließend auszuwerten. Durch diese große Modifikation des Tests „veränderte sich das Spiel von einem reinen „Deathmatch“ zu einem Bewertungsspiel“ (übersetzt durch den Autoren, Schrum/Karpov/Miikkulainen 2012, S.124).

Abbildung 5: Screenshot: Darstellung der Bewertung im Spiel.
Gelb – als Bot bewerteter Charakter (li.), blau – als Mensch bewerteter Charakter (re.)

Auch hier wurde der Waffenschaden auf 40% des Standards reduziert (vgl. Human-like Bots 2014), damit die Spieler genug Zeit haben konnten ihre Gegenspieler zu beobachten und zu bewerten. Für Bots war es außerdem auch möglich, die Bewertungswaffe zu benutzen, allerdings ohne Ihre Bewertungen in die Ergebnisse einfließen zu lassen. In dieser Phase sollten also die Richter aus der Erste-Person-Perspektive bewerten. Dabei sollte das Spiel jedes einzelnen Spielercharakters aufgenommen werden um anschließend die zweite Phase der Bewertung zu ermöglichen. Die zweite Phase bestand daraus, die aufgenommenen Spiele als Videos weiteren unbeteiligten Personen über eine Crowdsourcing-Plattform zu zeigen und sie wiederum alle Spieler nach ihrer Menschlichkeit beurteilen zu lassen. In dieser Phase wurden also auch die Richter, die im Spiel selbst beurteilt haben, bewertet. Schließlich sollte die „Menschlichkeit“ (Humanness) jedes Beteiligten aus der Bewertung beider Phasen aus der folgenden Formel resultieren (vgl. Human-like Bots 2 2014):

Formel 2.4.1: Menschlichkeitsberechnung (Humanness)

$$Humanness = FPA * FPWF + TPA * TPWF$$

Humanness – „Menschlichkeit“ eines bestimmten Spielcharakters
FPA (First-Person Assessment) – Ergebnisse der Erste-Person-Bewertungsphase
TPA (Third-Person Assessment) – Ergebnisse der Dritte-Person-Bewertungsphase, entstanden mit Hilfe einer Crowdsourcing-Plattform
FPWF (First-Person Weighting Factor) =0,5 – Gewichtung der Erste-Person-Bewertungsphase im Gesamtergebnis
TPWF (Third-Person Weighting Factor) =0,5 – Gewichtung der Dritte-Person-Bewertungsphase im Gesamtergebnis

Die Ergebnisse der einzelnen Bewertungsphasen und jedes Richters sollten dabei nach der Signalentdeckungstheorie (eng.: signal detection theory) die Genauigkeit des Richters berücksichtigen. Diese stellte den Durchschnitt aus den richtigen Bewertungen („Hits“) und den falschen Bewertungen („False alarms“, „Misses“) nach der folgenden Formel:

Formel 2.4.2: JR_j – Genauigkeit eines Richters

$$JR_j = \frac{Hits - (Misses + FalseAlarms)}{N_j}$$

j – Index des Richters
JR_j (Judge reliability) – Verlässlichkeit/Genauigkeit des Richters j
N_j – Anzahl aller durch den Richter j vorgenommenen Bewertungen

Mit den folgenden Bedeutungen der richtigen und der falschen Ergebnisse:

	Bewertet als „Mensch“	Bewertet als „Bot“
Spieler ist ein Mensch	Hit	False Alarm
Spieler ist ein Bot	Miss	Hit

Anschließend sollte die relative Genauigkeit des Richters wie folgt berechnet werden:

Formel 2.4.3: JRR_j – relative Genauigkeit eins Richters

$$JRR_j = \frac{JR_j}{AvgJR}$$

JRR_j (Judgement relative reliability) – relative Verlässlichkeit/Genauigkeit des Richters j
AvgJR – arithmetisches Mittel der Genauigkeiten (JR) aller Richter berechnet durch:

Formel 2.4.4: AvgJR – arithm. Mittel der Richtergenauigkeiten

$$AvgJR = \frac{\sum_{j=1}^{J}(JR_j)}{J}$$

J – Anzahl aller Richter

Mit Hilfe der relativen Genauigkeiten aller Schiedsrichter und den dazugehörigen „Menschlichkeitswerten“ sollte anschließend für jeden Spielcharakter einzeln das Ergebnis der ersten Bewertungsphase (FPA_i) berechnet werden:

Formel 2.4.5: FPA_i – Ergebnis der Erste-Person-Bewertung

$$FPA_i = \frac{\sum_{j=1}^{n}(JRR_j * Humanness_{i,j})}{J * N_{i,j}}$$

FPA_i – Ergebnis der ersten Bewertungsphase für Spieler i
$N_{i,j}$ – Anzahl aller durch den Richter j am Spieler i vorgenommenen Bewertungen

Formel 2.4.6: $Humanness_{i,j}$ – „Menshclichkeit“ eines Spielers laut einem Richter

$$Humanness_{i,j} = \frac{Miss_{i,j}}{J * N_{i,j}}$$

$Humanness_{i,j}$ – Anteil der „Menschlichkeit“ des Spielers i laut Richter j
Missi,j – Anzahl der Bewertungen des Bots i als „Mensch“ durch den Richter j

Die Ergebnisse der zweiten Phase (TPA) sollten nach demselben Prinzip unter Berücksichtigung der Richtergenauigkeit berechnet werden.

3 Stand der Wissenschaft

Seitdem der BotPrize-Wettbewerb zum ersten Mal im Jahr 2008 stattfand, haben viele Wissenschaftler sich vorgenommen, Agenten zu entwickeln, die den „Turing-Test für Bots“ bestehen sollten. Hierfür wurden unterschiedliche Ansätze und Methoden aus dem Forschungsbereich der künstlichen Intelligenz genommen und an die gestellte Aufgabe angepasst. Die Grenze zur „Menschlichkeit“ konnte jedoch mehrere Jahre lang nicht überschritten werden. Erst im Jahr 2012, als der Wettbewerb zum fünften Mal stattfand, konnten zwei Bots MirrorBot und UT^2 die Jurymitglieder von ihrer Menschlichkeit überzeugen. Die Ergebnisse der früheren Wettbewerbe sind in der vom Entwickler des MirrorBot Mihai Polceanu erstellten Grafik in Abbildung 6 zu sehen. Dabei ist jedoch zu beachten, dass in den ersten zwei Jahren der Test- und Bewertungsablauf anders als in den Folgejahren war. Erst ab 2010 wurde die im Kapitel BotPrize beschriebene Bewertungswaffe und die beschriebene Formel zur Berechnung der „Menschlichkeit“ benutzt. So wurden die Ergebnisse von 2008 und 2009 für die Darstellung umgerechnet (vgl. Polceanu 2013).

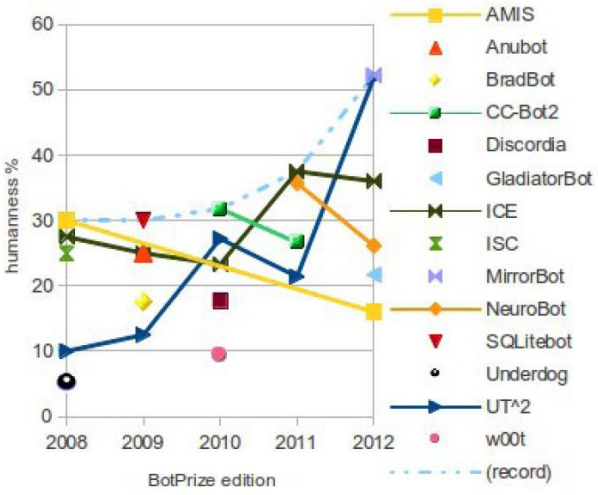

Abbildung 6: Bisherige Ergebnisse des BotPrize-Wettbewerbs. In (Quelle: Polceanu 2013).

Im folgenden Kapitel werden einige, der bis heute entwickelten Unreal Tournament-Bots beschrieben. Diese werden bereits von Mihai Polceanu in seinem Paper über den MirrorBot nach ihren Ansätzen in zwei unterschiedliche Kategorien eingeteilt: regelbasierte Systeme und Agenten, die mit Hilfe von neuronalen Netzen entwickelt wurden (vgl. Polceanu 2013). Zu den regelbasierten Architekturen gehören AMIS, SQLiteBot, CC-Bot2 und MirrorBot. Die Bots NeuroBot, ICE und UTˆ2 benutzen neuronale Netze in ihren Architekturen. Aus der Abbildung 6 ist ersichtlich, dass sich die Ergebnisse der Bots im Verlauf der Jahre insgesamt verbessert haben, wobei Repräsentanten beider Kategorien Rekordergebnisse geliefert haben.

Zu den genannten Architekturarten kann allerdings noch eine dritte Kategorie hinzugefügt werden – die mit Hilfe von genetischen Algorithmen entwickelten Bots. Auch wenn bis jetzt noch kein solcher Bot an dem BotPrize-Wettbewerb teilgenommen hat, so wird doch anschließend der Expert-Bot als ein wichtiger Repräsentant der evolutionär entwickelten Bots vorgestellt. Dieser Bot galt außerdem als eine große Hilfe für diese Arbeit. Einige Beispiele der unterschiedlichen Agenten werden im Folgenden kurz beschrieben.

3.1 Regelbasierte Architekturen

3.1.1 AMIS

Der Gewinner des BotPrize-Wettbewerbs im Jahr 2008 AMIS-Bot wurde als Nachfolger des Loque-Bots speziell für den Wettbewerb entwickelt. Der Loque-Bot bestand dabei aus vier unterschiedlichen Verhaltensweisen: Kampfverhalten, „Power-Up-Verhalten“ um dringend nötige Gegenstände aufzusammeln, Verfolgungsverhalten um Gegner zu verfolgen und ein Verhalten, bei dem der Bot zwischen nützlichen Items herumgelaufen ist. Für das Management dieser Verhalten gab es drei unterschiedliche Hilfsklassen: LoqueTravel, welche Entscheidungen über aufzusammelnde Items getroffen hat, LoqueMemory, welche auf Events gehört und den Agenten informiert hat und LoqueWeaponInfo, welches Informationen über bekannte Waffen gespeichert hat (vgl. Stolba 2014). Mit seinem sehr aggressiven Kampfverhalten und guten Waffenkenntnissen war der LoqueBot ein Agent, der ein „so guter Deathmatch-Spieler sein sollte, wie nur möglich.“(übersetzt durch den Autor, Stolba 2014) Da jedoch ein „zu guter“ Bot nicht menschlich erschien, wurden einige Änderungen im AMIS-Bot vorgenommen um ihn für den BotPrize schwächer, langsamer und somit „menschlicher“ erscheinen zu lassen.

Zum einen wurde das „Waffenwissen“ des Agenten geändert. Ursprünglich beinhaltete es Informationen über alle möglichen Waffen, sodass der LoqueBot in jeder Situation die optimale Waffe gewählt hat und sie am besten benutzen konnte. Bei dem AMIS-Bot wurden diese Informationen erst im Laufe des Spiels gesammelt, so dass der Bot „lernen“ konnte, welche Waffe unter welchen Umständen zu benutzen ist.

Zum anderen wurde versucht die „Glaubwürdigkeit“ des Bots zu verbessern. Hierzu gehörte z.B. die Zielgenauigkeit des Agenten, welche reduziert wurde, so, dass der AMIS-Bot manchmal auch danebengeschossen hat. Außerdem wurde die Bewegung des Bots insofern verändert, dass sie nicht mehr flüssig verlief, sondern mit einer bestimmten Wahrscheinlichkeit unterbrochen wurde. Laut der Aussage der Entwickler des AMIS-Bots war die Bewegung allerdings immer noch verbesserungsfähig nach dem ersten Wettbewerb (vgl. Stolba 2014).

3.1.2 SQLitebot

Den zweiten Wettbewerb im Jahr 2009 gewann der regelbasierte Agent SQLitebot. Die Besonderheit dieser Architektur bestand in der Benutzung einer SQL-Datenbank als eine Art „Langzeitgedächtnis“(vgl. Cothran 2014). Der Bot wurde entwickelt, basierend auf einem Beispielbot der Pogamut-Entwickler und übernahm das bereits erwähnte Kampfverhalten des AMIS-Bots aus dem Vorjahr. Allerdings wurde an dieser Stelle erkannt, dass zwar die meisten bisher entwickelten Bots sich Orte von Items „merken“, aber keine weiteren wichtigen Punkte in der Welt. Aus diesem Grund wurde beschlossen SQLite zu benutzen um „Hotspots“ auf der Karte über längere Zeit zu merken. Solche Hotspots konnten Orte sein, an denen der Agent andere Spieler zuletzt gesehen oder getötet hat, oder selbst getötet wurde. Der Vorteil von SQLite bestand dabei darin, dass das Schreiben und Lesen in und aus Dateien in einem Format möglich war, welches einfache Anfragen sowohl im Spiel als auch offline erlaubte. Außerdem nahmen diese Operationen nur Millisekunden ein, sodass keine Verzögerungen im Spiel entstanden. So war es bei Bedarf möglich mehrere Orte aus der Datenbank zu erhalten, z.B. alle wichtigen Hotspots, die der Agent in den letzten 45 Sekunden gesehen hat und dann über die weiteren Aktionen zu entscheiden(vgl. Cothran 2014).

Ein weiteres als menschlich betrachtetes Verhalten, das bei der Architektur neu hinzukam, war das Zurückziehen. Für den SQLitebot wurden im Vorfeld für alle Spielwelten Tabellen erstellt, in denen für jeden Navigationspunkt der Welt gespeichert wurde, welche anderen Punkte von ihm aus sichtbar waren. So konnte der Agent, wenn seine Gesundheitswerte eine bestimmte Minimumschwelle erreichten, aus der Tabelle einen Punkt auswählen, der vom Standpunkt seines Gegners nicht sichtbar war, auswählen und sich dorthin zurückziehen. Dies war bis dahin

von keiner anderen Architektur bekannt. Die Sichtbarkeitstabellen wurden dabei ebenfalls mit Hilfe von SQLite gespeichert.

Um weitere Verbesserungen des „menschlichen Erscheinens“ zu erzielen wurden an manchen Stellen mehr Zufallsvariablen eingeführt, so dass z.B. die Bewegungen während eines Kampfes weniger vorhersehbar und repetitiv waren. Außerdem wurde versucht „sinnlose“ Verhaltensweisen zu verhindern, welche den Bot als „unmenschlich“ charakterisieren würden, wie zum Beispiel längeres Herumlaufen in der Nähe eines unerreichbaren Ziels.

3.1.3 CC-Bot2

Im Jahr 2010 wurde der CC-Bot2 vorgestellt. Dieser wurde von einer Gruppe von Wissenschaftlern entwickelt, die sich mit dem Bewusstsein von Maschinen beschäftigten und die kognitive Architektur CERA-CRANIUM entwickelt haben (vgl. Arrabales 2 2014). Die Architektur erlaubt es, autonome Systeme wie mobile Roboter mit Hilfe von unterschiedlichen kognitiven Komponenten zu kontrollieren. Basierend auf dieser Architektur entstand der Gewinner des BotPrize-Wettbewerbs 2010.

Das System setzt sich aus zwei unterschiedlichen Bestandteilen zusammen. Die CERA-Komponente ist dabei das Kontrollelement des Systems und besteht aus vier in Abbildung 7 abgebildeten Schichten. Die unterste Schicht ist für die Sensomotorik des Agenten verantwortlich und entspricht bei dem CC-Bot2 in ihrer Funktionsweise der Pogamut-Architektur. Hier findet also die Kommunikation mit dem Spiel statt. Die zweite Schicht – physikalische – erstellt abstraktere Repräsentationen der Sensorinformationen und leitet diese an die oberen Schichten. Umgekehrt wandelt sie außerdem Befehle der oberen Schichten in einzelne Aktionen um, die an die unterste Schicht weitergereicht und vom Agenten ausgeführt werden können. In der dritten Schicht werden die Sensordaten zusammengefasst und mit Bezug auf die Ziele (Missionen) des Agenten weitergereicht. Die oberste Kernschicht ist dafür verantwortlich zu entscheiden, welche der erhaltenen kognitiven Informationen zu dem Zeitpunkt am relevantesten sind und entsprechende Mechanismen zu regulieren (vgl. Arrabales/ Muñoz/ Ledezma/ Gutierrez/ Sanchis 2012, S. 181). Zwischen den Schichten findet also immer sowohl eine Bottom-Up- als auch eine Top-Down-Kommunikation statt.

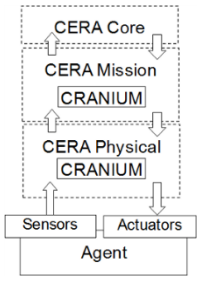

Abbildung 7: CERA-CRANIUM-Architektur.
In (Quelle: Arrabales/ Muñoz/ Ledezma/ Gutierrez/ Sanchis 2012, S. 181)

Die CRANIUM-Einheit verwaltet eine Vielzahl an Prozessoren mit unterschiedlichen Funktionen. Um die Fortbewegung des Agenten zu ermöglichen werden z.B. die in Abbildung 8 genannten Prozessoren berücksichtigt und aktiviert. Bei der Implementierung des CC-Bot2 erstellt CRANIUM für jeden aktiven Prozessor einen eigenen Thread, sodass diese parallel zueinander und unabhängig voneinander laufen können.

Name	Layer	Description
NotMoving	Physical	Detects when the bot is not moving during some time
StuckDetector	Physical	Detects when the bot is stuck in a given place
RandomMove	Physical	Moves randomly trying to unstuck the bot
RandomNavigation	Physical	Random movement among the navigation points. A wander behaviour
MoveCloserPosition	Physical	Fires a percept when the bot is close to a target position
RandomJumpGenerator	Physical	Performs a jump action randomly
AttackingMovement	Physical	Realizes a random movement when is in attacking mode

Abbildung 8: Einige Prozessoren, die bei der Generierung der Fortbewegung des CC-Bot2 berücksichtigt werden. In (Quelle: Arrabales/ Muñoz/ Ledezma/ Gutierrez/ Sanchis 2012, S. 186)

Wie in Abbildung 7 zu sehen ist, benutzt der CC-Bot2 zwei getrennte CRANIUM-Komponenten, die in der physischen und in der Missionsschicht als Subsysteme funktionieren und unterschiedliche Informationen als Eingaben benutzen (vgl. Arrabales/ Muñoz/ Ledezma/ Gutierrez/ Sanchis 2012, S. 182ff). Die Entscheidung darüber, welche der Prozessoren aktiviert werden sollen, wird basierend auf einer heuristischen Berechnung danach getroffen, wie viel sie zu der derzeitigen Mission beitragen können (vgl. Arrabales/ Muñoz 2014).

Zu dem Zeitpunkt der Entwicklung des CC-Bot2 wurde von allen möglichen kognitiven Fähigkeiten vor allem der Aufmerksamkeits-Mechanismus implementiert. Dieser sollte dafür sorgen, dass situationsabhängig das System entscheiden konnte, welche Sensoreingaben zurzeit wichtig waren und sich nur auf diese konzentrieren. Im Gegensatz zu Standardlösungen, bei denen immer dieselbe Anzahl an Informationen berücksichtigt wird, konnten hier bestimmte Werte verworfen werden. Somit war es unter anderem möglich, die CPU-Auslastung bei Bedarf zu reduzieren und dadurch Verzögerungen zu verhindern (vgl. Arrabales/ Muñoz/ Ledezma/ Gutierrez/ Sanchis 2012, S. 189).

Der größte Unterschied zu anderen Architekturen war hierbei, dass durch die kognitiven Fähigkeiten dieses Systems es keine festen Regeln für das Verhalten des Bots gab. Die Entwicklung einer solchen Architektur ist somit zwar sehr interessant für den Forschungsbereich des maschinellen Bewusstseins, stellt aber Spieleentwickler, die kein Vorwissen auf diesem Gebiet haben, vor große Schwierigkeiten. Aufgrund der Komplexität der Architektur ist es sehr schwierig diese zu verstehen und zu debuggen, vor allem, wenn mehr kognitive Fähigkeiten wie Emotionen oder ein episodisches Gedächtnis implementiert werden (vgl. Arrabales/ Muñoz/ Ledezma/ Gutierrez/ Sanchis 2012, S. 189).

3.1.4 MirrorBot

Im Jahr 2012 nahm MirrorBot als ein weiterer regelbasierter Agent an dem BotPrize-Wettbewerb teil. Der Entwickler ging hierbei von den Beobachtungen aus der Psychologie über die Spiegelung (eng.: „mirroring“) im menschlichen Verhalten aus. Es wurde berücksichtigt, dass unbewusste Nachahmung eine wichtige Grundlage im sozialen Verhalten der Menschen bildet. Die Fähigkeit bestimmte Aktionen zu wiederholen und aus ihnen weitere Verhaltensweisen zu lernen, spielt eine wichtige Rolle in zwischenmenschlichen Beziehungen. Bezogen auf den Wettbewerb wurde besonders nach der Einführung der Bewertungswaffe und der Reduktion des Schadens anderer Waffen „die Aggression im Spiel teilweise durch freundliche Interaktionen zwischen den Spielern ersetzt und die soziale Interkation gefördert“ (übersetzt durch den Autor, Polceanu 2013). Aus diesem Grund wurde bei der Entwicklung des MirrorBot versucht ein Spiegelverhalten zu implementieren, um ein Bewusstsein des Agenten vorzutäuschen.

Das System wurde dabei aus zwei unterschiedlichen Modulen aufgebaut, welche in Abbildung 9 zu sehen sind. Ein Modul sollte, wie viele andere Architekturen, nach vorgegebenen Regeln das Verhalten des Bots generieren, wenn der Bot keinen oder einen aggressiven Gegner vor sich

hatte. Sobald jedoch ein freundlicher (nicht schießender) Gegner vor dem Agenten erschien, wurde das zweite Modul aktiviert, welches das Verhalten des Gegenspielers beobachten und nach einiger Zeit widerspiegeln sollte.

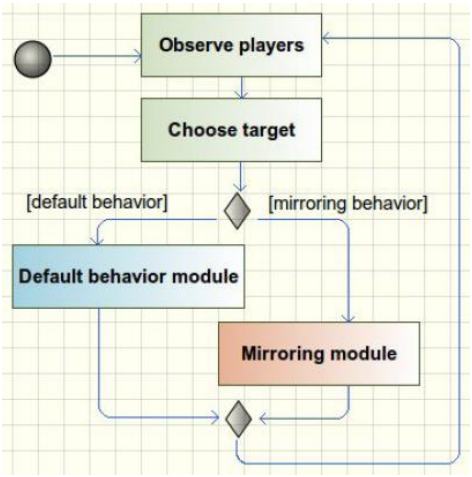

Abbildung 9: Übersicht der Verhaltensweisen des MirrorBot. In (Quelle: Polceanu 2013).

Das in Abbildung 10 abgebildete Standard-Modul des Agenten kontrollierte dabei getrennt die Fortbewegung und das Ziel- beziehungsweise Schießverhalten des Bots. Um ein Ziel auszuwählen, hat der MirrorBot den aggressivsten der sichtbaren Gegner ausgewählt. Bei einem sich bewegenden Gegner hat der Bot nicht die aktuelle Position des Gegners fokussiert, sondern einen durch die Geschwindigkeit des Gegners berechneten Zukunftsort. Falls der Agent keinen Gegner sichten konnte, hat er einen Punkt zwischen den zwei nächsten Navigationspunkten „angeschaut“, was besonders bei der Bewegung um eine Ecke „menschlicher“ erscheinen sollte.

Die Schusskomponente des Moduls war dafür verantwortlich über die Wahl der Waffe und des Schussmodus zu entscheiden. So wurde z.B. die Bewertungswaffe ausgerüstet, wenn der Bot „seine Entscheidung über die Menschlichkeit des Spielers änderte“ (übersetzt durch den Autor, Polceanu 2013). Für die Bewertung eines Charakters wurde die Mehrheit der Bewertungen anderer Spieler benutzt.

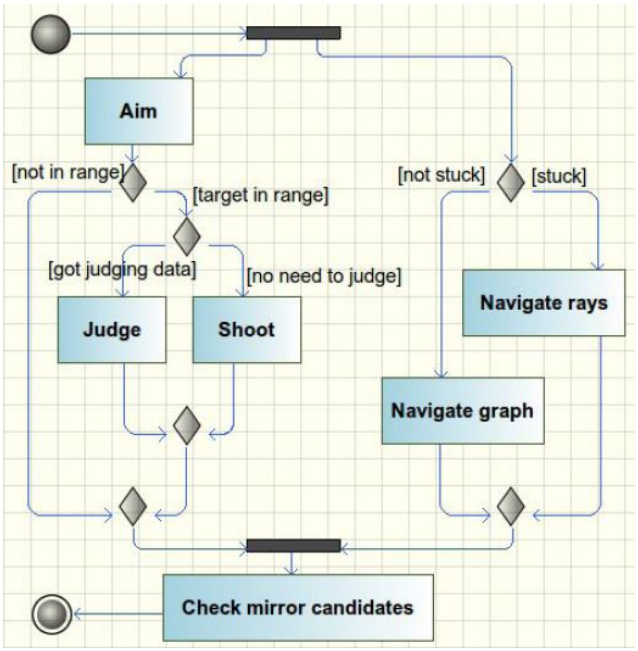

Abbildung 10: MirrorBot's Default-Verhalten. In (Quelle: Polceanu 2013).

Für die Fortbewegung des Agenten wurden zwei unterschiedliche Navigationssysteme implementiert. Standardweise wurde der Navigationsgraph benutzt um den Agenten in der Welt zu bewegen. Für die Fälle, dass der Spielcharakter stecken blieb oder aus einem anderen Grund den Graphen nicht benutzen konnte, wurde die Fortbewegung mit Hilfe von Strahlen ermöglicht. Hierfür wurden insgesamt 16 horizontale Strahlen vom Agenten aus in unterschiedliche Richtungen gesendet um so die Umwelt auf Hindernisse „abzutasten". Außerdem wurden 8 weitere Strahlen mit einem 45°-Winkel nach unten gesendet um mögliche Lücken im Boden zu erkennen und ihnen auszuweichen.

Alle diese Komponenten wurden deaktiviert sobald der Agent einen passenden Kandidaten gefunden hat, um sein Verhalten zu imitieren. Danach wurde das Verhalten nach einzelnen Keyframes aufgenommen. Hierfür wurden Informationen über den Spieler wie seine Position, Rotation, ausgerüstete Waffe etc. gespeichert. Nach einer Verzögerung von ca. 200 Millisekunden hat der Agent das Aufgenommene Verhalten, außer den Vorwärts- und Rückwärtsbewegungen spiegelverkehrt wiederholt. Die Vorwärtsbewegung des Gegners wurde dabei als eine Rückwärtsbewegung ausgeführt (und entsprechend umgekehrt) um den Abstand zwischen den Spielcharakteren ungefähr gleich zu halten. Das „Spiegeln" wurde entweder dann unterbrochen, wenn der Bot attackiert wurde, der Gegner sich umdrehte, oder der Bot länger als acht Sekunden lang den Spieler imitierte (vgl. Polceanu 2013).

Probleme, die der Entwickler bei diesem Ansatz jedoch sah, lagen in der Imitation des Verhaltens eines anderen Bots. Wenn dieser zum Beispiel steckenblieb, oder sich nicht bewegte, erkannte der MirrorBot diesen als einen friedlichen Spieler an und wiederholte sein Verhalten. Da dieses jedoch als sehr „botähnlich" war, könnte der MirrorBot von einem menschlichen Spieler auch als ein Bot bewertet werden. Um dies zu verhindern wurde überprüft, ob der imitierte Spieler sich „zu still" verhielt und das Spiegeln wurde gegebenenfalls unterbrochen.

Mit dieser Kombination von „einfachen" Regeln und der Imitation des menschlichen Verhaltens schaffte MirrorBot es als einer der zwei ersten Bots den modifizierten Turing-Test zu bestehen. Im Jahr 2012 erzielte er einen Rekord von 52,2% der „Menschlichkeit" und landete auf dem ersten Platz des BotPrize-Wettbewerbs (vgl. 2K BotPrize 2014).

3.2 Neuronale Netze und Evolutionäre Algorithmen

3.2.1 NeuroBot

Der NeuroBot nahm zum ersten Mal an dem BotPrize-Wettbewerb im Jahr 2011 teil. Das Ziel des Entwicklerteams bestand darin, zu zeigen, „dass biologisch inspirierte neuronale Mechanismen eine effektive Möglichkeit sein können, um ein System mit einem menschenähnlichen Verhalten zu konstruieren" (übersetzt durch den Autor, Fountas/ Gamez / Fidjeland 2011). Die Architektur wurde basierend auf der Theorie des globalen Arbeitsraums (eng.: Global Workspace Theory) aufgebaut. Die Grundidee der Theorie besteht darin, dass viele Prozesse darum konkurrieren, ihre Informationen in den „globalen Arbeitsraum" zu reichen. Der „Gewinnerprozess" kann schließlich seine Informationen an alle anderen Prozesse weiterleiten, was wiederum einen neuen Konkurrenzzyklus auslöst (vgl. Fountas/ Gamez / Fidjeland 2011). (Das Prinzip der Theorie des globalen Arbeitsraums wurde bereits bei der Entwicklung des im Kapitel CC-Bot2 beschriebenen CC-Bot2 angewendet.) Im Falle des NeuroBots wurde ein neuronaler globaler Arbeitsraum implementiert, bei dem ein „gepulstes" neuronales Netz (eng.: spiking neural network SNN) aus ca. 20000 Neuronen (vgl. Fountas 2011, S.21) aufgebaut wurde. Der Unterschied eines solchen Netzwerks im Vergleich zu einem herkömmlichen künstlichen neuronalen Netz ist, dass die Neuronen hierbei nicht bei jedem Zyklus feuern, sondern erst bei Erreichen eines bestimmten Schwellenwerts. Einen Vorteil dieses Netzwerks sahen die Entwickler unter anderem darin, dass das Ergebnis „biologisch realistischer" (vgl. Fountas 2014) sein sollte. Das neuronale Netz wurde mit Hilfe des NeMo-

Simulators[4] umgesetzt, welcher vor allem für die Forschung im Bereich der Neurowissenschaften benutzt wird.

Als Eingaben für die Neuronen wurden die Daten des Spiels verwendet. Hierfür gab es mehrere Sensormodule, die, wie in Abbildung 11 abgebildet, die Daten an die Knoten des globalen Arbeitsraumes weitergeleitet haben (vgl. Fountas 2011, S.25ff).

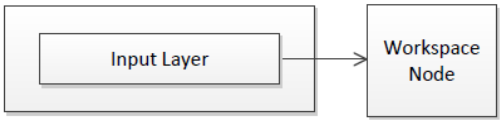

Abbildung 11: Architektur eines Sensormoduls. In (Quelle: Fountas 2011, S.25).

Außerdem wurden insgesamt neun Motorikmodule implementiert, die für die elementaren GameBots-Befehle wie MOVE, JUMP etc. verantwortlich waren. Diese Module waren wiederum über ihre Outputlayer mit den Knoten des Arbeitsraumes verbunden (s. Abbildung 12).

Abbildung 12: Architektur eines Motorikmoduls. In (Quelle: Fountas 2011, S.26).

Zusätzlich wurden neun unterschiedliche Verhaltensmodule implementiert. Diese bekamen die Pulsmuster (eng.: spiking patterns) der Knoten als Eingaben. Im Folgenden werden die Module kurz vorgestellt:

- Moving Module: bestimmt, ob der Spielcharakter sich bewegen sollte
- Navigation Module: ermöglicht eine Fortbewegung entlang von Wänden und verhindert Kollisionen
- Exploration Module: kontrolliert die Blickrichtung des Spielcharakters
- Firing Module: bestimmt, ob der Spielcharakter schießen sollte
- Jumping Module: bestimmt, ob der Spielcharakter springen sollte
- Chasing Module: ermöglicht die Suche und die Verfolgung eines Gegners, wenn das Firing Module: aktiviert und der Gesundheitszustand des Spielcharakters gut ist
- Fleeing Module: kontrolliert das Fliehen vor einem Gegner, wenn der Gesundheitszustand des Charakters schlecht ist
- Recuperating Module: kontrolliert die Bewegung zu einem Gesundheits-Aufbau-Item, wenn der Gesundheitszustand des Charakters schlecht ist
- Look Item Module: kontrolliert die Bewegung zu einem sichtbaren Item
- U-turn Module: ermöglich eine komplette Drehung des Charakters, wenn dieser von mehreren Wänden umgrenzt ist (in einer Ecke steht)

Obwohl der NeuroBot bei dem Wettbewerb gegen Bots, die bereits vorher teilgenommen haben, konkurrierte, verfehlte er nur knapp den ersten Platz. Die Entwickler sehen das Problem vor allem in den letzten zwei Spielrunden des Wettbewerbs. Aufgrund von Internetverbindungsproblemen musste die Geschwindigkeit, mit der das neuronale Netz aktualisiert wurde, reduziert werden, wodurch der NeuroBot verzögert reagierte (vgl. Fountas 2011, S.48).

[4] s. NeMo: http://nemosim.sourceforge.net/

3.2.2 UT^2 2010

Bereits in den ersten zwei Wettbewerbsjahren nahm ein an der Universität von Texas entwickelter Bot am BotPrize teil. Als UT^2 ist dieser allerdings erst seit dem Wettbewerb im Jahr 2010 bekannt, bei dem er den zweiten Platz erreichte. Im Jahr 2012 konnte der Bot sogar zusammen mit dem bereits beschriebenen MirrorBot den Turing-Test bestehen.

Auch wenn der UT^2-Bot als ein mit Hilfe eines neuronalen Netzes entwickelter Agent kategorisiert wird, kann dieser im Prinzip in alle drei Architekturkategorien zugeteilt werden. Das zugrundeliegende System entspricht einer typischen verhaltens- beziehungsweise regelbasierten Architektur. Diese bestand nach dem Wettbewerb im Jahr 2010 aus acht unterschiedlichen Verhaltensmodulen. Wie in Abbildung 13 zu sehen ist, konnten die Module (blau) mit Hilfe von sechs verschiedenen Controllern (orange) unterschiedliche Aktionen (gelb) ausführen (vgl. Schrum/ Karpov/ Miikkulainen 2012, S. 125ff).

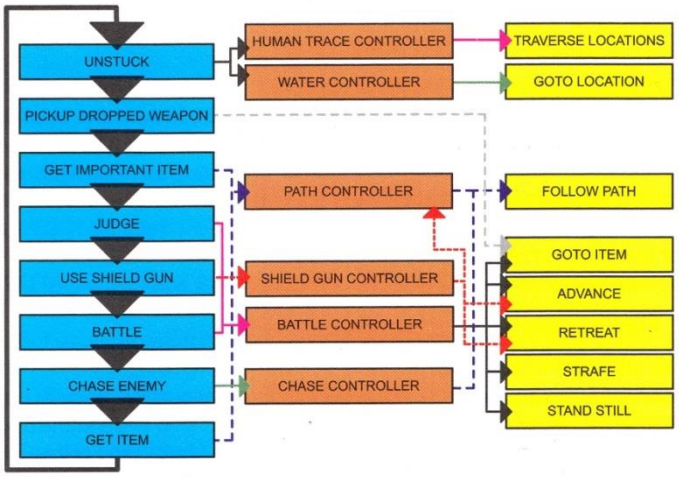

Abbildung 13: Architektur des UT^2-Bots. In (Quelle: vgl. Schrum/ Karpov/ Miikkulainen 2012, S.126).

Jedes der Module konnte durch eigene Bedingungen ausgelöst werden. In jedem Zeitschritt wurde über die Module in der in Abbildung 13 gezeigten Reihenfolge (von oben nach unten) iteriert und überprüft, welcher der Modulauslöser aktiviert wurde.

Der wichtigste Controller war hierbei der Battle-Controller. Dieser regelte das Verhalten des Bots während einer Konfrontation mit einem Gegner und trug somit sehr viel dazu bei, wie „menschlich“ der Bot erschien. Der Controller implementierte ein künstliches neuronales Netz. Als Eingaben für die Neuronen wurden spezielle Sensoren benutzt. Die in Abbildung 14 dargestellten Kreissegmente lieferten z.B. Informationen darüber, ob sich ein Gegner innerhalb des Sichtfeldes des Bots aufhielt. Außerdem wurden insgesamt 22 Strahlen zur Abtastung der Umwelt auf Hindernisse und andere Agenten benutzt. Das Netzwerk hatte insgesamt acht mögliche Ausgaben, wovon fünf zur Bewegung benutzt wurden und drei weitere angegeben haben, ob der Bot schießen sollte, mit welcher Waffe er ausgerüstet werden sollte und ob ein Sprung nötig war. Während all dieser Aktionen fokussierte sich der Bot immer auf den Gegner (vgl. Schrum/ Karpov/ Miikkulainen 2012, S.126).

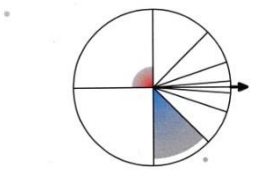

Abbildung 14: Kreissegment-Sensoren des UT^2. Blickrichtung des Agenten dargestellt durch den Pfeil, graue Punkte – Gegner, farbige Segmente sind aktiviert durch die Gegner. In (Quelle: vgl. Schrum/ Karpov/ Miikkulainen 2012, S. 129).

Das Kampfverhalten des Bots wurde durch Neuroevolution trainiert. Das beschriebene neuronale Netz wurde hierfür zuerst durch eine sehr einfache Struktur dargestellt, die keine versteckten Schichten (eng.: hidden layers), sondern nur Input- und Outputknoten besaß. Unter Benutzung von drei unterschiedlichen Mutationsoperatoren wurde das Netz anschließend gelernt. Ein Operator veränderte dabei die Gewichte einzelner Netzwerkverbindungen, ein Operator addierte neue Verbindungen und der letzte Operator spaltete die vorhandenen Knoten. Es wurde entschieden keine Crossover-Operatoren zu benutzen (vgl. Schrum/ Karpov/ Miikkulainen 2012, S. 133ff). In der Fitnessfunktion wurden dabei folgende Werte berücksichtigt: zugefügter Schaden, Zielgenauigkeit, erhaltener Schaden, Anzahl der Kollisionen mit der Umwelt, Anzahl der Kollisionen mit anderen Agenten. Jedoch kritisierten die Entwickler des Bots selbst die hohe Anzahl der Ziele. Das Training fand in fünf unterschiedlichen Spielwelten statt. Der Agent hat dabei zwischen 100 und 500 Sekunden lang gegen fünf Beispielbots von Pogamut gespielt. Es wurden insgesamt 60 Generationen des Agenten mit einer Populationsgröße von 20 Individuen trainiert. Die Entwickler des UT^2-Bots erkannten, dass diese Größen für dieses Problem sehr klein waren. Aufgrund von langen Evaluierungszeiten entschieden sie sich jedoch gegen eine Evolutionierung in größeren Maßstäben.

In der Fitnessfunktion für den Wettbewerb 2010 wurde die Bewertungswaffe noch nicht berücksichtigt, sodass der Bot „nur“ darauf trainiert wurde, ein möglichst gutes Kampfverhalten zu entwickeln. In den Videos aus dem BotPrize 2012, in dem der UT^2 die 50%-Schwelle überschreiten konnte, ist jedoch zu sehen, wie der Bot selbst die Bewertungswaffe benutzt (vgl. Schrum/ Karpov 2014).

Eine weitere Besonderheit des UT^2 bestand darin, dass der Bot einen speziellen Controller hatte, der die Bewegungen des Agenten im Falle des Steckenbleibens regelte. Da die Standardnavigation von Pogamut nicht immer reibungslos abläuft, kann es passieren, dass ein Agent irgendwo steckenbleibt und nicht mehr mit Hilfe der Pfadberechnung weiterkommen kann. Aus diesem Grund wurden für den UT^2 mehrere Spiele von Menschen gespielt, wobei alle Events des Spiels und Spielerdaten in einer Log-Datei gespeichert wurden. Danach wurden aus dieser Datei bestimmte Informationen über Spieler (wie Position, Geschwindigkeit, Rotation etc.) extrahiert und in einer SQLite-Datenbank gespeichert. Der „Human Trace Controller“ des Agenten konnte auf diese Datenbank zugreifen, falls der Bot steckenblieb und das Verhalten eines Menschen in einem ähnlichen Fall „nachspielen“. So konnte die Wahrscheinlichkeit vermindert werden, dass der Bot aufgrund dieses „typischen Bot-Verhaltens“ als Bot anerkannt wurde (vgl. Karpov/ Schrum/ Miikkulainen 2012, S.155ff).

3.2.3 Expert Bot

Der Expert-Bot wurde an der Universität von Granada entwickelt. Wie bereits erwähnt, nahm dieser Bot bis jetzt an keinem Wettbewerb teil. Da er jedoch auch mit dem Ziel entwickelt wurde möglichst menschlich zu wirken und sein Quellcode frei zur Verfügung stand, diente dieser Bot als Grundlage bei der Ausarbeitung des im Rahmen dieser Arbeit entwickelten OvGUBots (s. Kapitel 4.1.2). Die Verhaltensregeln des Expert-Bots wurden mit Hilfe eines spanischen Unreal-Tournament-Expertenspielers erstellt, wodurch der Bot seinen Namen erhielt. Die zugrundeliegende Architektur entsprach einem endlichen Zustandsautomaten mit zwei Zustandsschichten (vgl. Mora/ Aisa/ Caballero/ Garcıa-Sanchez/ Merelo/ Castillo/ Lara-Cabrera 2013). Wie in Abbildung 15 zu sehen ist, gab es insgesamt fünf primäre Zustände, in einem von denen sich der Bot zu jederzeit befinden musste. Die Zustände waren:

- Attack: um einen Gegner, der nicht wegläuft zu attakieren
- Hunt: um einen weglaufenden Gegner zu verfolgen
- Retreat: um vor einem Gegner wegzulaufen
- Camp: um an einem Ort auf einen Gegner im Hocken zu warten
- Greedy: um nützliche Items in der Welt zu sammeln

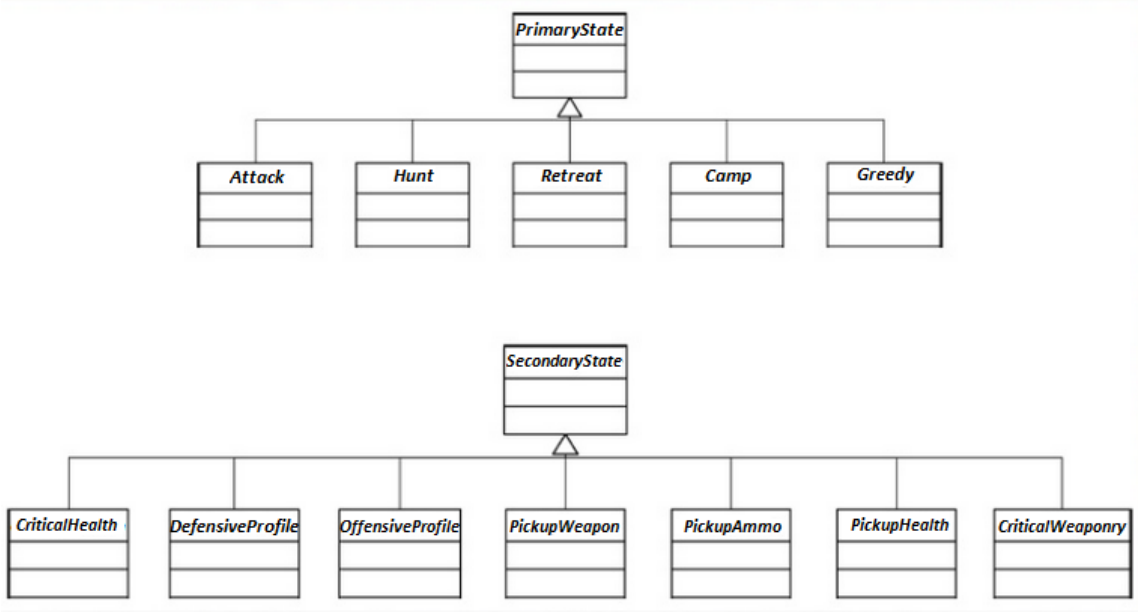

Abbildung 15:Primäre und sekundäre Zustände des Expert Bots. (In Anlehnung an Mora 2014).

Die sekundären Zustände waren dagegen optional und führten zusätzliche Aktionen innerhalb des gewählten Zustandes aus. D.h. ein sekundärer Zustand wurde zusätzlich zu einem primären Zustand nur dann angenommen, wenn dies notwendig war. So wurde zum Beispiel der Zustand „CriticalHealth“ zusätzlich zu dem Zustand „Retreat“ betreten, wenn der Gesundheitszustand des Agenten sehr schlecht war. Die Architektur war so gestaltet, dass in jedem Zeitschritt bestimmte Faktoren in einem Entscheidungsbaum überprüft wurden und der passende primäre Zustand (und eventuell ein sekundärer Zustand) ausgewählt wurde. Die Entscheidungsregeln wurden aufbauend auf vorigen Arbeiten und des Expertenwissens aufgestellt (vgl. Mora/ Aisa/ Caballero/ Garcıa-Sanchez/ Merelo/ Castillo/ Lara-Cabrera 2013).

Außer den Informationen über den Bot selbst, wurden Informationen über den Gegner gespeichert. Es ist allerding nicht möglich solche Informationen von GameBots, beziehungsweise dem Spiel zu erhalten, da der Agent nicht „allwissend“ sein darf. Aus diesem Grund wurden die Parameter des Gegners wie sein Gesundheitszustand, seine Munition usw. abgeschätzt. Der Expert-Bot ging also am Anfang des Spiels davon aus, dass die Gesundheitswerte des Gegner maximal sind. Sobald der Expert-Bot aber den Gegner beschoss, wurde der Gesundheitszustand des Gegners ausgehend von dem zugefügten Schaden aktualisiert. Dies war möglich, weil der Expert-Bot nur in 1-gegen-1-Spielrunden trainiert wurde und davon ausgehen konnte, dass kein weiterer Spieler den betrachteten Gegner angegriffen hat.

Nachdem der Expert-Bot in zwei Spielwelten jeweils zwei Spiele gegen den Standard-UT2K4-Bot zur Evaluierung gespielt hat, wurden einige Entscheidungsparameter optimiert. Hierfür wurde der Bot auf zwei unterschiedliche Weisen mit Hilfe eines evolutionären Algorithmus trainiert. In einer Version wurden Chromosomen mit 143 Genen benutzt, in der anderen mit nur 26 Genen. Bei der ersten Variante bestimmten 126 der Gene über die Auswahl der Waffen. Diese Funktion erfüllten in der zweiten Variante nur noch neun Gene. Die restlichen Gene beider Varianten bezogen sich auf Gesundheitswerte des Bots, Distanzangaben, Itemprioritäten und eine Zeitangabe für das Verfolgen und Weglaufen vor dem Gegner. Die Fitnessfunktionen berücksichtigten dabei wie oft der Bot innerhalb eines Spiels gestorben ist, wie oft er seinen Gegner getötet hat, den erhaltenen und den zugefügten Schaden. Bei der ersten Variante wurden 30 Generationen mit 30 Individuen in 15-minütigen Spielrunden entwickelt. Die zweite Variante hatte 50 Generationen à 30 Individuen und es wurden Spiele von je fünf Minuten gespielt. Dabei wurde in beiden Fällen ein Elitismusmechanismus zur Selektion gewählt, der jeweils vier besten Individuen der Generation behielt. Außerdem wurde das uniforme Crossover

und eine Mutation angewendet, die mit der Wahrscheinlichkeit 1/Chromosomgröße den Wert eines Gens um 10% veränderte (vgl. Mora/ Aisa/ Caballero/ Garcıa-Sanchez/ Merelo/ Castillo/ Lara-Cabrera 2013).

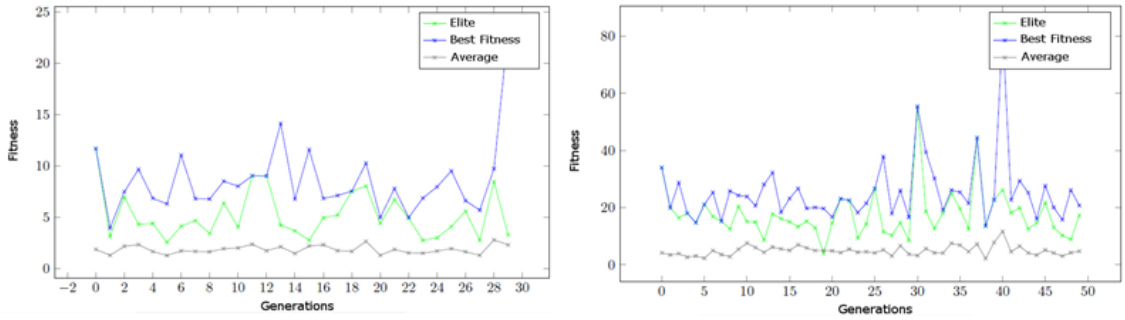

Abbildung 16: Fitnesswerte der ersten Variante (143 Gene li.) und der zweiten Variante (26 Gene re.) In (Quelle: Mora/ Aisa/ Caballero/ Garcıa-Sanchez/ Merelo/ Castillo/ Lara-Cabrera 2013).

Wie in Abbildung 16 zu sehen ist, veränderten sich die durchschnittlichen Fitnesswerte im Verlauf beider Experimente jedoch kaum. Die Entwickler des Expert-Bots führten diese Erkenntnis darauf zurück, dass die Werte nicht nur von dem Bot selbst abhingen, sondern auch sehr stark von den Gegneraktionen, die nicht kontrollierbar waren (vgl. Mora/ Aisa/ Caballero/ Garcıa-Sanchez/ Merelo/ Castillo/ Lara-Cabrera 2013). Aus diesem Grund entschieden sie sich für eine dritte Variante des Experiments, die mehr auf die Ausbeutung (eng.: exploitation) guter Lösungen abzielen sollte. Hierfür wurden in die Fitnessfunktion Werte eingefügt, die besser die Leistung des Bots spiegelten (z.B. wie oft er ein Schild aufgehoben hat oder wie oft er eine der „starken“ Waffen benutzt hat.) Es wurden wieder Chromosomen der Länge 26 benutzt und 50 Generationen à 30 Individuen entwickelt. Um den Selektionsdruck zu erhöhen wurden die besten 15 Individuen für die nächste Generation behalten und die restlichen Individuen wurden durch fitnessproportionale Selektion ausgewählt. Die Crossover- und Mutationsoperatoren blieben gleich.

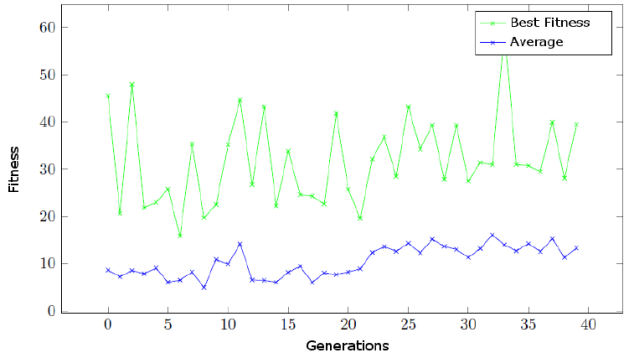

Abbildung 17: Fitnesswerte der dritten Variante (26 Gene, veränderte Fitnessfunktion, Selektion). In (Quelle: Mora/ Aisa/ Caballero/ Garcıa-Sanchez/ Merelo/ Castillo/ Lara-Cabrera 2013).

Wie die Abbildung 17 zeigt, war ein kleiner Anstieg der Durchschnittswerte der Fitness bei der letzten Variante erkennbar. Trotzdem fanden die Entwickler, dass die „Evolution nicht so gut wie gewünscht“ (übersetzt durch den Autor, Mora/ Aisa/ Caballero/ Garcıa-Sanchez/ Merelo/ Castillo/ Lara-Cabrera 2013) war und auch nach der Meinung des Expertenspielers war die Leistung des Bots schlechter als erwartet.

4 Entwicklung des OvGUBots

4.1 Überlegungen und Entscheidungen

4.1.1 Regelbasierte Architektur

Bevor erste konkrete Ansätze eines Agenten gemacht wurden, wurde überlegt, worauf seine Architektur basieren könnte. Hierfür wurden die bereits beschriebenen Beispiele, ihre Besonderheit und die Ergebnisse der vorigen Wettbewerbe betrachtet. Letztendlich fiel die Entscheidung auf eine regelbasierte Architektur, in der die unterschiedlichen Verhaltensmuster durch Zustände repräsentiert werden sollten. Die Vorteile einer solchen Architektur wurden unter anderem darin gesehen, dass die Regeln in einer solchen Architektur für Menschen leicht verständlich sein könnten. So könnten sie einfach während der Entwicklung formuliert werden und in Fehlerfällen wäre es relativ einfach die Fehler zu finden und zu korrigieren. Dies wäre aber zum Beispiel bei der Entwicklung eines neuronalen Netzes kaum möglich gewesen. Ein wichtiges Argument gegen ein neuronales Netz oder die Entwicklung mit Hilfe von genetischen Algorithmen bestand außerdem darin, dass für diese Methoden eine Fitnessfunktion nötig gewesen wäre, um den Agenten verbessern zu können. Da das wichtigste Ziel bei der Entwicklung des Agent jedoch darin bestand, ihn möglichst „menschlich“ erscheinen zu lassen und „Menschlichkeit“ nicht gemessen und durch Formeln dargestellt werden kann, könnte sie nicht in der Fitnessfunktion berücksichtigt werden. In einer regelbasierten Architektur könne jedoch die Kommentare und Anmerkungen der vorigen Wettbewerbe dazu genutzt werden, Regeln zu erstellen, die den Agenten „menschlich(-er)“ erscheinen lassen würden.

Die Entscheidung den OvGUBot auf diese Weise zu entwickeln wurde schließlich dadurch bekräftigt, dass alle in dem Kapitel 3.1 beschriebenen Agenten, die eine regelbasierte Architektur besaßen, in den jeweiligen Wettbewerbsjahren Topleistungen erbringen konnten. Dies zeigte, dass es möglich war, mit festformulierten Regeln Verhaltensweisen darzustellen, die oft „menschlicher“ erschienen als die Verhalten anderer Agenten. Um den OvGUBot also so zu entwickeln, dass sein Verhalten für „menschlich“ gehalten werden könnte, mussten anschließend die „richtigen“ Regeln zusammengefasst und ausformuliert werden.

4.1.2 Expert Bot als Grundlage für den OvGUBot

Als Orientierungshilfen bei der Entwicklung des OvGUBots dienten mehrere regelbasierte Architekturen. Besonders in den späteren Phasen war es wichtig, die Regeln und Abläufe mehrerer Systeme zu vergleichen. Für den Einstieg in die gegebene Programmierumgebung war jedoch (nach den Beispielbots von Pogamut) der bereits im Kapitel 3.2.3 beschriebene Expert Bot besonders hilfreich. Der Quellcode für den Expert Bot stand frei (und vollständig) zur Verfügung, sodass der Bot sofort getestet und einzelne Befehle nachvollzogen werden konnten.

Im Vergleich zu anderen im Kapitel 3 beschriebenen Architekturen hatte der Expert Bot ähnliche Zustände, wie die meisten Implementierungen und konnte somit die wichtigsten Verhaltensweisen bereits annehmen. Durch die mit Hilfe eines Expertenspielers erstellten Regeln lieferte der Bot gute Ergebnisse in 1-gegen-1-Spielen gegen die nativen UT2K4-Bots bereits vor der Optimierung der Parameter durch die evolutionären Algorithmen.

Außerdem war der Quellcode für diesen Agenten nicht nur frei verfügbar, sondern unterstand auch der GNU General Public License[5], die es erlaubte, diesen Quellcode zu benutzen und weiterzuentwickeln.

Aufgrund all dieser Vorteile bot die Architektur des Expert Bots eine gute Grundlage für die Entwicklung des OvGUBots, sodass manche Teile davon bis in die finale Version übernommen wurden. Allerdings wurden auch sehr viele Änderungen daran vorgenommen, die detaillierter im folgenden Kapitel beschrieben werden.

4.2 Implementierung

Aufbauend auf der Architektur des Expert Bots und unter Berücksichtigung der Anmerkungen der Entwickler anderer Bots, sowie des Jury-Feedbacks voriger BotPrize-Wettbewerbe konnten viele wichtige Verhaltensmuster herausgestellt werden, die ein Bot unbedingt annehmen musste um „menschlich“ zu wirken. Die Implementierung dieser Verhaltensweisen und die damit zusammenhängenden Entscheidungen werden nun einzeln erläutert.

4.2.1 Zielsuche und Navigation

Das größte Problem, welches die meisten Bots „unmenschlich“ erscheinen ließ, war die Fortbewegung. Sobald diese nicht reibungslos ablief, der Bot steckenblieb oder sich zu stark an dem Navigationsgraphen orientierte, wurde dies von den Richtern des BotPrize-Wettbewerbs bemerkt und der Bot wurde entsprechend bewertet. Aus diesem Grund war es zuerst wichtig, eine möglichst flüssige Navigation des OvGUBots zu ermöglichen.

Wie bereits in Kapitel 2.3 erwähnt, beinhalten die Spielwelten in UT2K4 Navigationsgraphen, die ein Agent zur Pfadfindung nutzen kann. Mit Hilfe von GameBots bestand bei der Implementierung die Möglichkeit den A*-Algorithmus[6] von GameBots oder den Floyd-Warshal-Algorithmus[7] zu benutzen. Beide Algorithmen wurden durch GameBots selbst berechnet und brauchten lediglich einen Start- und einen Zielknoten als Eingaben. Systemtechnisch funktionieren beide Algorithmen gleich gut. Die nativen Bots von UT2K4 benutzen jedoch den A*-Algorithmus, der auch in der Architektur des Expert Bots eingebaut war. Aus diesen Gründen wurde entschieden, bei der A*-Variante zu bleiben. Für die Navigation in der Welt brauchte ein Agent außer einer Variante der „PathPlaner“-Klasse auch eine Klasse, die für die Navigation entlang des Pfades verantwortlich sein sollte. Bei der Architektur des Expert Bots wurde für diese Zwecke der „LoqueNavigator“ – Teil der Pogamut-Bibliothek – benutzt. Er ermöglichte es, die Spielfigur flüssig von einem Knoten zum nächsten zu bewegen und überprüfte dabei, ob es sich bei den Knoten um spezielle Stellen wie Fahrstühle oder Orte zum Springen handelte.

Die Navigation mit Hilfe der „LoqueNavigator“-Klasse funktionierte somit gut, allerding lief der Agent dabei immer genau entlang einer Graphkante. Da die Knoten und somit auch die Kanten sich meistens jedoch in der Mitte eines breiten Weges befinden (s. Abbildung 18 links), nahm der Agent immer denselben Weg. Ein Mensch würde sich allerdings jedes Mal etwas anders bewegen. Aus diesem Grund wurde für den OvGUBot eine leicht veränderte Variante des LoqueNavigators implementiert, die es ermöglichte, die Spielfigur mit leichten Abweichungen von den Orten der Knoten fortzubewegen. Da sich jedoch die Items auf den Graphknoten befanden und gesichert werden musste, dass der Bot ein Item im Vorbeilaufen aufheben konnte, durfte die Abweichung von einem Knoten maximal dem Kollisionsradius des Spielcharakters entsprechen. So wurde bei der Navigation durch den Graphen zu dem Standort des jeweils nächsten Navigationspunktes ein zufälliger Offset in x- und y-Richtung hinzugefügt,

[5] s. Free Software Foundation: GNU General Public License http://www.gnu.org/copyleft/gpl.html
[6] s. Lester, P.: A* Pathfinding for Beginners http://www.policyalmanac.org/games/aStarTutorial_de.html
[7] s. Lang, H.W.: Floyd-Warshall-Algorithmus http://www.iti.fh-flensburg.de/lang/algorithmen/graph/warshall.htm

der aber nicht den Kollisionsradius überschreiten durfte. Dadurch konnte ein möglicher Weg durch den in Abbildung 18 gezeigten Graphabschnitt wie im rechten Teil der Darstellung aussehen. Der Agent würde sich nicht mehr exakt entlang der Graphkanten bewegen (grüne Linie) und würde trotzdem alle Items aufheben können, da sie sich in seiner Reichweite (rote Ellipsen) befinden würden.

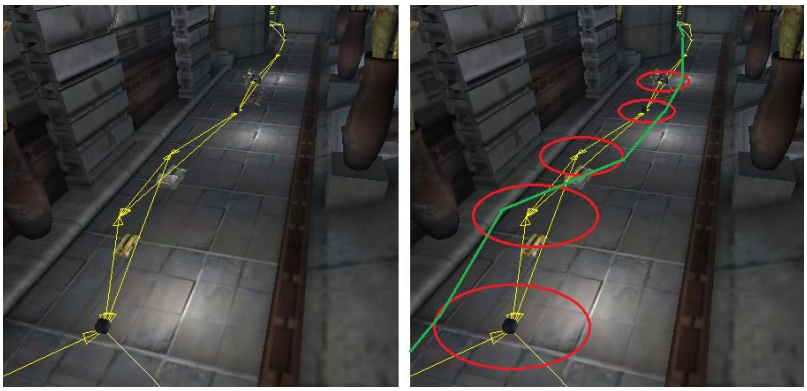

Abbildung 18: Navigation ohne (li.) und mit Abweichung von dem Nevigationsgraphen (re.).
Gelb: Kanten des Graphen, grün: möglicher Pfad mit Abweichungen, rot: Kollisionsradien eines Spielcharakters.

Um die Laufziele sinnvoll auszuwählen, wurde in der Architektur des Expert Bots in der „EstimateDestination"- Methode in jedem Zeitschritt überprüft, ob sich ein anderer Spieler in der Nähe des Agenten befand. Falls dies nicht der Fall war, wurde aus allen sichtbaren Items das beste Item zum Aufsammeln ausgewählt. Die Prioritäten der Items wurden in der Anfangsversion des Expert Bots durch den Expertenspieler festgelegt. Nach der Evolutionierung wurden die Prioritäten aus den entsprechenden Genen ausgelesen. Falls kein Item zurzeit zu sehen war, wurde einer der nächsten Navigationspunkte zufällig ausgewählt. Sollte jedoch ein Gegner in der Nähe des Spielcharakters sein, wurde ein Navigationspunkt nur dann als Ziel gewählt, wenn es der Standort eines Schildes war.

Der Agent hat sich außerdem zu dem sichtbaren Item nur dann bewegt, wenn er sich nur in einem primären und keinem sekundären Zustand befand. Falls aber ein sekundärer Zustand gewählt wurde, wurde die zustandsspezifische Bewegung ausgeführt. So hat sich der Agent zum Beispiel auf einen Gegner zubewegt, falls er sich in dem sekundären Zustand „OffensiveProfile" befand oder aber ein „Gesundheits-Item" gesucht hat, wenn er im Zustand „PickupHealth" war (vgl. Abbildung 15.) Das heißt, in solchen Fällen wurde an mehreren Stellen im Quellcode nach einem nützlichen Gegenstand gesucht. Dies sollte bei der Implementierung des OvGUBots geändert werden, so, dass nur in der „EstimateDestination"-Methode das zurzeit am meisten benötigte Item gesucht wurde. Um trotzdem entscheiden zu können, ob gerade ein Gesundheitsitem oder Munition mehr benötigt wird, wurde bei der Überprüfung der Itemprioritäten der derzeitige Gesundheitszustand und die Waffen bzw. Munition des Agenten berücksichtigt.

Die Wichtigkeit der einzelnen Gegenstände wurde für den OvGUBot aus der für den UT^2-Bot erstellten Waffenpräferenztabelle und den, durch den Experten angegebenen Item-Prioritäten des Expert Bots ermittelt. Diese konnten aus den Quellcodes beider Bots ausgelesen werden. Im „Greedy"-Zustand des Expert Bots wurden die Prioritäten aller Items in Abhängigkeit seiner Gesundheit. Für den UT^2-Bot gab es eine Tabelle, nach der entschieden wurde, welche Waffe ausgerüstet werden sollte. Daraus ließ sich schließen, welche Waffenarten öfter benutzt wurden und somit eher aufgehoben werden sollten. Die Waffenpräferenzen wurden in Abhängigkeit des Abstandes zu dem Gegner angegeben. Aus diesen zwei Tabellen resultierten die in Tabelle 1 dargestellten Prioritäten der Gegenstände für den OvGUBot. Die in den ersten fünf Zeilen angegebenen Items tragen zur Verbesserung des Gesundheitszustands bzw. des Schutzes der Spielerfigur bei und sind deshalb in den meisten Fällen (außer bei maximalem Gesundheitswert) wichtiger als Waffen oder Munition.

Tabelle 1: Item-Prioritätentabelle des OvGUBots.

Item	**Priorität**
Super Health Pack	100 (0, wenn Gesundheitswert=200)
Super Shield Pack	100
Shield Pack / U-Damage Pack	90
Health Pack	90 (0, wenn Gesundheitswert>=100)
Mini Health Pack	80 (0, wenn Gesundheitswert=200; 45, wenn weniger als 50% der Munition vorhanden)
Flac Canon	70
Minigun	60
Shock Rifle	50
Rocket Launcher	50
Bio Rifle	40
Lightning Gun	30
Assault Rifle	20
Sniper Rifle	10

Nach der oberen Tabelle war es nun möglich die Gegenstände in der Spielwelt nach ihrer Wichtigkeit auszuwählen und den OvGUBot zu ihnen laufen zu lassen. Genau wie bei dem Expert Bot wurde hierbei, falls mehrere Gegenstände die gleiche Priorität hatten, der Gegenstand mit der kleinsten Distanz zu dem Agenten gewählt. Dies bedeutete jedoch, dass wenn der Agent zu einem Item mit einer sehr hohen Priorität lief, er keine weiteren Items beachtet oder aufgehoben hat. Befände sich also der Agent in der Abbildung 19 am Punkt A und liefe er zu dem „Health Pack“ (Priorität 90) am Punkt B, würde er die Items an den Punkten C, D und E ignorieren bis er das „Health Pack“ aufgehoben hätte. Dieses Verhalten könnte den Bot also bei einer näheren Betrachtung als „unmenschlich“ charakterisieren, da ein Mensch sich mehrere Ziele gleichzeitig merken kann und somit normalerweise nützliche Items auf seinem Weg zu einem wichtigen Ziel aufheben kann.

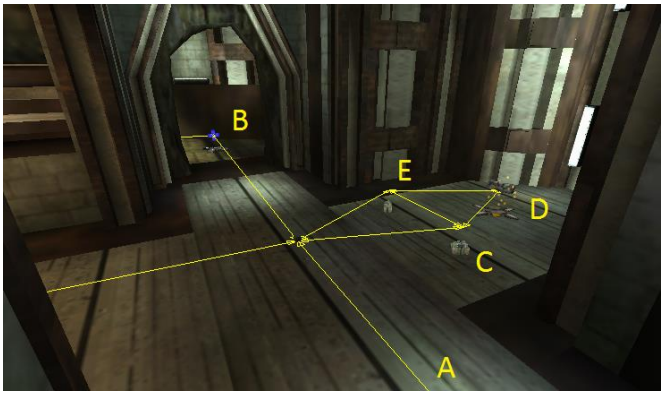

Abbildung 19: Mehrere sichtbare Items.

Damit der OvGUBot also dieses „typische Botverhalten“ nicht annimmt, wurde entschieden zwei Zielpunkte zu speichern. Hierfür wurde eine Variable angelegt, die immer das aktuelle Ziel speicherte und eine weitere, die ein mögliches nächstes Ziel speichern konnte (nur wenn ein weiteres Ziel vorhanden war.) Durch diese Variablen war es möglich, das wichtigste Item (hier B) sich zu merken und die Items in der Nähe unter Berücksichtigung ihrer Priorität und der Distanz zu ihnen als Zwischenziele einzufügen.

Tabelle 2 stellt den Ablauf von Aktionen und Entscheidungen für den in Abbildung 19 gezeigten Fall dar. Das „aktuelle Ziel“ stellt dabei immer das Item dar, zu welchem sich der Bot zum Zeitpunkt t bewegt. In der letzten Spalte wird begründet, warum die gewählten Itemstandorte anderen Punkten vorgezogen wurden.

Tabelle 2: Auswahlreihenfolge sichtbarer Items des OvGUBots.

Zeitpunkt t	Aktuelles Ziel	Nächstes Ziel	Aktion	Begründung
1	B	-	Läuft weiter, sieht C, E, wählt C als erstes Ziel, B wird zum nächsten Ziel	C ist näher als E und B hat eine höhere Priorität als E
2	C	B	Erreicht C, sieht D, wählt D als erstes Ziel, B bleibt nächstes Ziel	D und E sind gleich nah, aber D und B haben höhere Prioritäten als E
3	D	B	Erreicht D, wählt E als erstes Ziel, B bleibt nächstes Ziel	E ist näher als B und es sind keine weiteren Items zu sehen um B als nächstes Ziel zu ersetzen
4	E	B	Erreicht E, wählt B als aktuelles Ziel	Außer B ist kein weiterer Punkt sichtbar
5	B	-	Erreicht B, sucht weiter	

Nachdem die oben beschriebenen Änderungen vorgenommen wurde, sollte sich der Agent flüssig in der Welt bewegen können. Dies war jedoch nicht immer der Fall, da aus unterschiedlichen Gründen der Bot doch manchmal stecken bleiben konnte und sich nicht mehr bewegt hat. Möglicher Grund dafür wäre z.B., dass GameBots merklich lange gebraucht hat um einen Pfad zu berechnen oder überhaupt kein Pfad zu einem Punkt gefunden werden konnte

(vermutlich weil die Graphdaten veraltet waren). Außerdem könnte der Bot auf seinem Weg in eine Lücke fallen und würde trotzdem versuchte dem berechneten Pfad zu folgen obwohl er vor einer Wand stand. Um diese Fälle abzufangen und das „unmenschliche“ Verhalten des Bots zu verschleiern, wurde in jedem Zeitschritt abgefragt, ob sich der Bot bewegte. Der einzige Fall, in dem sich der Bot nicht bewegen sollte, war, wenn er sich auf einer Fahrstuhlplattform befand. Falls er sich also nicht in der Nähe eines Fahrstuhls befand und sich trotzdem nicht bewegte, wurde zuerst abgefragt, ob er nah genug an dem aktuellen Ziel war und deswegen stehen blieb. In dem Fall wurde sofort ein neues Ziel gesucht und ein neuer Pfad berechnet. Dies war allerdings selten der Fall, da der Pfad zum nächsten Ziel bereits kurz vor dem Erreichen des aktuellen Ziels berechnet wurde. Das größte Problem bestand aber, wenn sich der Agent noch sehr weit von dem aktuellen Ziel befand und sich trotzdem nicht bewegte. Dies bedeutete, dass der Weg zu diesem Ziel (zurzeit) nicht berechnet werden konnte. Als Lösung für dieses Problem wurde entschieden, das unerreichbare Ziel eine Zeit lang zu blockieren und nach einem anderen Ziel, welches erreichbar war, zu suchen. Hierfür wurde der aktuelle Zielpunkt zu einer Liste unerreichbarer Orte hinzugefügt, die bei der Suche nach weiteren Zielen nicht berücksichtigt wurden. Da davon ausgegangen werden konnte, dass kein Pfad nur von der aktuellen Position des Agenten zu dem verworfenen Ziel berechnet werden konnte (aber der Punkt möglicherweise von anderen Orten aus erreichbar war), musste der Zielpunkt aus der Liste wieder gelöscht werden. Dies passierte, wenn die Liste eine bestimmte Anzahl an blockierten Punkten beinhaltete. (Nach einigen Testläufen erwies sich eine Anzahl von fünf Orten als sinnvoll.) Damit der Bot während der Zeit, in der ein neues Ziel gesucht und ein neuer Pfad berechnet wurde, nicht ohne Bewegung stand, wurden leichte Bewegungen reingebracht. Hierfür wurde zuerst mit Hilfe von Strahlen überprüft, ob sich eine Wand vor dem Agenten befand, so, dass er sich von ihr wegdrehen musste. Es wurden genau wie bei dem Expert Bot insgesamt acht Strahlen von dem Bot aus zur Abtastung der Umwelt gesendet. Wenn der Agent nicht vor einer Wand stand, drehte er sich zufälligerweise nach links und rechts in einem Winkel zwischen 0 und 30°. Dadurch wurde ein „Umherschauen“ simuliert, was den Bot wiederum „menschlicher“ erschienen ließ, da Menschen während des Spiels auch manchmal stehen bleiben um die Gegend genauer zu betrachten.

4.2.2 Zustände

Nach den beschriebenen Optimierungen der Zielsuche des Agenten und seiner Fortbewegung stellte sich die Frage, ob und wie vollständig die in der Architektur des Expert Bots implementierten Zustände für den OvGUBot behalten werden sollten. Die Grundstruktur konnte so behalten werden, dass die einzelnen Zustände von der Elternklasse „PrimaryState“ erbten. Diese beinhaltete Methoden für die Waffenauswahl und Waffenwechsel, sowie eine Methode für die Fortbewegung, die in den einzelnen Zuständen überschrieben wurde.

Aufgrund von gezogenen Vergleichen mit anderen Architekturen stand fest, dass folgende Zustände auf jeden Fall implementiert werden mussten: „Attack“, „Retreat“ und „Greedy“, da der Bot in der Lage sein sollte wichtige Gegenstände aufzusammeln, wenn kein Gegner in der Nähe war. Anderenfalls war es wichtig, den Gegner zu attackieren, wenn die Möglichkeit dazu gegeben war oder vor ihm wegzulaufen, falls der Bot „zu schwach“ war. Zusätzlich zu diesen drei Verhaltensmustern musste bedacht werden, dass im Gegensatz zu dem OvGUBot der Expert Bot nicht für den BotPrize-Wettbewerb entwickelt wurde und der Aspekt der Bewertung nicht berücksichtigt wurde. Da sich aber, wie bereits in Kapitel 2.4 erwähnt, das Spielprinzip bei dem Wettbewerb durch die Bewertungswaffe sehr stark änderte und das Spiel viel langsamer und weniger aggressiv ablief, musste auch das Verhalten des Bots entsprechend angepasst werden.

Wie auf manchen Videos aus den vorigen Wettbewerben zu sehen ist (z.B. vgl. Karpov 2014), haben sich die Richter öfters zurückgezogen und von einem ruhigen Platz aus andere Spieler beobachtet. So konnten sie mehr Zeit gewinnen und dann eine Entscheidung bezüglich der „Menschlichkeit“ beobachteter Spieler treffen. Zu dem Zeitpunkt der Entwicklung des OvGUBots war bereits bekannt, dass es bei dem BotPrize-Wettbewerb 2014 eine

Bewertungsphase geben wird, in der alle Spielcharaktere (Menschen und Bots) aus der dritten Perspektive von unbeteiligten Personen bewertet werden (s. Kapitel 2.4). Aus diesem Grund konnte davon ausgegangen werden, dass während dieser Phase das „Beobachterverhalten" genau wie in den vorigen Videos zu bemerken sein würde. Um also vor allem diejenigen, die in der zweiten Phase alle Spieler auf ihre „Menschlichkeit" bewerten zu überzeugen, wurde entschieden für den OvGUBot zu den bereits genannten Zuständen einen „Observe"-Zustand hinzuzufügen. In diesem sollte der Bot ähnlich den Richtern andere Spieler beobachten ohne sie zu attackieren.

Damit deutlich sein konnte, dass der Agent gerade einen Spieler beobachtet, war es außerdem wichtig, dass er den Spieler verfolgen konnte, falls dieser aus dem Sichtfeld verschwand. Somit war die Verfolgung eines Spielers sowohl beim Attackieren, als auch beim Beobachten – also in zwei unterschiedlichen Zuständen wichtig. Aus diesem Grund wurde entschieden, auf den bei dem Expert Bot implementierten Zustand „Hunt" (vgl. Abbildung 15) als solchen zu verzichten und die Verfolgung als eine Methode in der „PrimaryState" zu implementieren. Diese konnte aus den Zuständen „Attack" und „Observe" aufgerufen werden, wenn der Zielspieler aus dem Sichtfeld verschwand. In allen anderen Zuständen war eine Verfolgung nicht notwendig.

Zusätzlich wurde in den Videos vergangener Wettbewerbe beobachtet, dass die menschlichen Spieler öfters stehen geblieben sind, sich umgeschaut haben und auf andere Spieler in bestimmten Orten gewartet haben. Damit der OvGUBot auch in der Lage sein konnte solche Orte zu finden und dieses Verhalten zu simulieren, wurde anschließend entschieden einen „Hide"-Zustand zu implementieren. Darin sollte sich der Bot in einem kleinen Bereich hin und her bewegen, umschauen und eine kurze Zeit lang auf Gegenspieler warten.

Schließlich wurde der bei dem Expert Bot implementierte Zustand „Camp" verworfen (vgl. Abbildung 15). Dieser diente dazu, dass der Expert Bot sich an den Orten, an denen Spieler nach dem simulierten Tod wieder erschienen, hinhocken und warten konnte. Der Zustand wurde zum einen verworfen, da in allen Videos bisheriger Spieler kein Spieler gesehen wurde, der die Hock-Funktion benutzt hat. Zum anderen war das Warten in dem „Camp"-Zustand so umgesetzt, dass der Agent währenddessen nur in eine Richtung geschaut hat ohne sich zu bewegen. Dieses Verhalten könnte jedoch als sehr „unmenschlich" anerkannt werden, weil sich Menschen, auch wenn sie stehen bleiben, sich drehen und ihre Umgebung anschauen. Das starre Hocken und Warten aus dem „Camp"-Zustand wurde also durch Hin-und-Her-Pendeln und die Umschau-Funktion in dem neuen „Hide"-Zustand ersetzt.

Somit ergaben sich insgesamt folgende fünf Zustände, die unterschiedlich Verhaltensweisen des Agenten ermöglichen sollten und von der Klasse „PrimaryState" erbten: „Attack", „Retreat", „Greedy", „Observe" und „Hide". Anschließend wurde überlegt, inwiefern die bei dem Expert Bot implementierte sekundäre Zustände benötigt wurden. Diese wurden, wie bereits in dem Abschnitt 4.2.1 beschrieben, nur manchmal angenommen und dienten überwiegend dem Aufsammeln bestimmter derzeit benötigter Gegenstände. Da jedoch die Auswahl der nötigen Items bei dem OvGUBot bereits geändert wurde, wurde an dieser Stelle entschieden, die sekundären Zustände des Expert Bots komplett zu verwerfen und die primären Zustände um fehlende Funktionen zu erweitern. Um zu gewährleisten, dass je nach der Verfassung des Agenten der richtige Zustand betreten wurde und um die rechnerische Leistung zu verbessern, wurde außerdem die Auswahl des aktuellen Zustandes geändert.

Wie bereits erwähnt wurden bei dem Expert Bot in jedem Zeitschritt diverse Werte des Agenten überprüft und anhand dieser der anzunehmende Zustand ermittelt. Dies Geschah in einer Extraklasse („Skynet") innerhalb der „Behave"-Methode und wurde ähnlich einem Entscheidungsbaum durch viele ineinander verschachtelte If-Else-Schleifen umgesetzt. Der Nachteil einer solchen Implementierung besteht jedoch darin, dass unabhängig davon, ob und welche der Bedingungen einer If-Else-Schleife erfüllt werden, alle Befehle innerhalb der Schleife jedes Mal auf den Stack-Speicher geladen und bei dem Verlassen der Schleife wieder gelöscht werden. Durch die vielen verschachtelten Schleifen wurde der Stack bei dem Expert

Bot in jedem Zeitschritt unnötig ausgelastet. Um dies zu vermeiden und die Entscheidungsregeln übersichtlicher zu halten wurde entschieden für jeden Zustand festzulegen, durch die Erfüllung welcher Bedingungen dieser betreten werden konnte. Hierfür wurde in jedem Zustand die Methode „ConditionsAreTrue“ implementiert, in der für diesen Zustand wichtige Parameter überprüft wurden und ein Boolean-Wert zurückgegeben wurde. In der „Behave“-Methode wurden somit die „ConditionsAreTrue“-Methoden der einzelnen Zustände der Reihe nach aufgerufen. Sobald die Bedingungen für einen Zustand zutrafen, wurde dieser ausgewählt ohne die Bedingungen anderer Zustände zu überprüfen und den Stack-Speicher unnötig auszulasten.

Dies bedeutete jedoch, dass die Zustände, die zuerst überprüft wurden, weitere Zustände blockieren könnten. Aus diesem Grund war es wichtig, die Reihenfolge der Zustände bei der Überprüfung nach ihrer Dringlichkeit festzulegen. Hierfür wurden folgende Überlegungen aufgestellt: die Bedingungen der Zustände, die eine Interaktion mit Gegnern erforderten (attackieren, weglaufen, beobachten), mussten vor den Bedingungen der Zustände, in denen der Agent alleine war (verstecken und sammeln), überprüft werden. (Die Bedingungen und Regeln für einzelne Zustände werden später bei der Beschreibung dieser genannt.)

„Retreat“-Bedingungen mussten dabei vor den „Attack“-Zustandsbedingungen überprüft werden, denn falls der Agent attackiert wurde und nicht zurückschießen konnte, musste er weglaufen, sodass die Prüfung der „Attack“-Bedingungen nicht mehr nötig war. „Observe“ war dabei weniger wichtig als „Retreat“, denn auch hier galt: bevor der Bot einen Gegner beobachtet, sollte zuerst überprüft werden, ob seine Lebens- oder Munitionswerte nicht so schlecht sind, dass er vor dem Gegner vorsichtshalber weglaufen sollte. Der Zustand „Observe“ sollte jedoch vor „Attack“ kontrolliert werden. Falls ein Gegner sichtbar war, vor dem der Bot nicht weglaufen musste und der den Bot nicht attackiert hat, sollte zuerst überprüft werden, ob der Bot den Gegner beobachten sollte. War es nicht der Fall, könnte dann überprüft werden, ob der Bot den Gegner attackieren sollte oder eventuell auf keine Interaktion mit ihm eingehen sollte.

Falls die Bedingungen eines der ersten drei Zustände nicht erfüllt waren, sollte anschließend überprüft werden, ob der Agent sich verstecken könnte oder Items aufsammeln sollte. „Greedy“ war somit der Zustand mit der niedrigsten Wichtigkeit und wurde als eine Arte „Default-Zustand“ betreten, wenn die Bedingungen der anderen vier Zustände nicht erfüllt wurden. Damit der „Greedy“-Zustand außerdem betreten werden konnte, falls ein bestimmtes Item gerade benötigt wurde (z.B. Lebensitem, Munition), wurden die Gesundheits- und Munitionswerte des Agenten in allen Zuständen überprüft.

Abbildung 20: Reihenfolge der Überprüfung der Zustandsbedingungen von oben nach unten.

Aus den oben beschriebenen Überlegungen ergab sich die in der Abbildung 20 dargestellte Reihenfolge für die Überprüfung der Zustandsbedingungen (von oben nach unten). So wurde nun in der ursprünglichen „Behave“-Methode in jedem Zeitschritt zuerst die „ConditionsAreTrue“-Methode der Klasse „Retreat“ aufgerufen. Falls diese den Boolean-Wert „true“ zurückgab, wurde der Zustand „Retreat“ ausgewählt und die „Behave“-Methode verlassen. Anderenfalls wurde die „ConditionsAreTrue“-Methode der Klasse „Observe“

aufgerufen usw. Auf diese Weise waren außerdem, genau wie bei der Implementierung des Expert Bots, Übergänge von jedem Zustand in jeden anderen Zustand möglich.

Durch diese Umsetzung war es allerdings möglich, dass der Bot sehr lange in einem Zustand blieb, wenn sich die überprüften Variablen nicht änderten. Dies wäre allerdings ungünstig für die Zustände „Observe“ und „Hide“. In „Observe“ würde es bedeuten, dass der Bot sehr lange einen Spieler beobachten würde ohne ihn anzugreifen oder weiterzulaufen, wodurch dieser ihn als Bot identifizieren könnte. In dem „Hide“-Zustand könnte es sogar dazu kommen, dass sobald der Bot einen guten Punkt zum Verstecken gefunden hat, er diesem überhaupt nicht mehr verlassen würde, bis ein Gegner innerhalb seines Blickfeldes erschien. Um dies zu vermeiden wurden in der Klasse „PrimaryState“, von der alle Zustände erben, zwei zusätzliche Variablen „stateStartTime“ und „timeStateLeft“ eingefügt um für jeden Zustand zu speichern, wann dieser zuletzt betreten bzw. verlassen wurde. Zusätzlich wurden zwei weitere Variablen eingefügt, durch die für jeden Zustand einzeln festgelegt werden konnte, wie lange sich der Bot maximal innerhalb dieses Zustandes befinden konnte („secondsToBeInState“) und wie viel Zeit vergehen musste, bis der Bot wieder diesen Zustand betreten durfte („minTimeBetweenSameState“). Mit Hilfe dieser vier Parameter konnte gesichert werden, dass sich der Bot weder zu lange in den Zuständen „Hide“ und „Observe“ befand, noch zu oft sich versteckt oder andere Spieler beobachtet hat. Die Zeiten für diese zwei Zustände wurden zuerst aufgrund der Erfahrung festgelegt und anschließend in einigen Testspielen gegen mehrere Spieler angepasst.

Ähnlich vielen anderen Implementierungen wurde auch bei dem Expert Bot das Schießverhalten getrennt von dem Bewegungsverhalten umgesetzt (vgl. z.B. BattleController von UT^2 in Kapitel 3.2.2). Die Schießfunktion wurde in der Elternklasse „PrimaryState“ eingebaut, da der Bot innerhalb mehrerer Zustände schießen können sollte. Dieses Prinzip wurde auch für den OvGUBot behalten, sodass aus den zwei Zuständen „Attack“ und „Retreat“ auf die Schießfunktion zugegriffen wurde. Hierdurch regelten die einzelnen Zustände also nur das Bewegungsverhalten des Agenten. Die grundlegenden Funktionen einzelner Zustände sowie ihre Besonderheiten werden im Folgenden kurz beschrieben.

4.2.2.1 Retreat

Um den OvGUBot vor dem Gegner weglaufen zu lassen, wurde bei seiner Implementierung auf die bereits für den Expert Bot geschriebene Hilfsmethode „getBestRunZone“ gegriffen. Diese Methode lieferte den nächsten Navigationspunkt zu dem Spieler, der gleichzeitig am weitesten von dem Gegner entfernt war. Hierfür wurden die Positionen des Bots sowie des Gegners übergeben und aus allen möglichen Navigationspunkten der Map wurde der passende ermittelt und zurückgegeben. Zu diesem Punkt wurde der Bot anschließend geschickt. Die Berechnung der Zielpunkte und die Bewegung zu ihnen fand solange statt, wie der Bot sich in dem „Retreat“-Zustand befand. Dieser konnte abgebrochen werden, wenn der Bot entweder starb, oder mehr als eine bestimmte Zeit lang nicht beschossen, was darauf deutete, dass der Gegner aufgehört hat, den Bot zu verfolgen.

Nach einigen Testspielen ist jedoch aufgefallen, dass der Bot beim Weglaufen seinen Blick immer auf den Gegner fokussiert hat. Dies wirkte jedoch sehr unrealistisch wenn der Bot rückwärts lief und dabei trotzdem nicht gegen Wände stieß. Aus diesem Grund wurde beschlossen, den Bot beim Weglaufen in die Laufrichtung schauen zu lassen. Da menschliche Spieler sich jedoch manchmal umdrehen, um nach dem Verfolger zu schauen und einschätzen zu können, wie weit der Gegner ist, wurde auch für den OvGUBot eine Möglichkeit implementiert, während des Rückzugs sich umzudrehen. So drehte sich der Bot für einige Sekunden in Richtung des Gegners, falls er seit mehr als 5 Sekunden nicht mehr getroffen wurde. Dieses Verhalten sollte eine Überprüfung der Gegnerposition simulieren.

4.2.2.2 Observe

Das Verhalten des Agenten in dem „Observe“-Zustand musste aus zwei unterschiedlichen Bewegungsarten bestehen. Falls der Bot seinen Zielspieler sah, sollte er diesen beobachten und sich dabei in seiner Sichtweite bewegen. Wenn der Gegner jedoch aus der Sichtweite des Bots verschwand, musste dieser ihn verfolgen um ihn weiterhin beobachten zu können.

Die Verfolgungsfunktion wurde, wie bereits erwähnt, in der Klasse „PrimaryState“ implementiert und konnte aus dem Beobachtungszustand aufgerufen werden. Hierbei wurde ein Pfad zu der Position berechnet, an der der verfolgte Gegner zuletzt gesehen wurde. Zu diesem Punkt sollte sich der Bot bewegen in der Annahme, den Gegner von dort aus wieder sehen zu können. Falls der Gegner jedoch von dieser Position aus nicht mehr zu sehen war, verließ der Bot den „Observe“-Zustand.

Wenn der Agent den gegnerischen Spieler sah und dieser still stand, sollte der Bot mit einer kleinen Wahrscheinlichkeit (20%) auch stehen bleiben und diesen im Stehen beobachten. Anderenfalls sollte der Bot sich zufälligerweise nach links oder rechts bewegen. Diese schleichenden Pendelbewegungen werden oft von menschlichen Spielern ausgeführt, wenn sie sich nicht gezielt zu einem Punkt bewegen. Da Menschen jedoch ihre Umgebung sehen und einschätzen können, ob in ihrer Nähe Hindernisse vorhanden sind, würden sie bei den Pendelbewegungen nicht gegen eine Wand laufen oder in Bodenlücken fallen. Damit auch der OvGUBot beim Beobachten eines Spielers nicht gegen Wände läuft oder in die Tiefe stürzt, wurde entschieden, mit Hilfe von Strahlen die Umgebung des Agenten abzutasten und seine Bewegungen damit zu kontrollieren.

Abbildung 21: 10 Strahlen zur Abtastung der Umgebung des Agenten. Grün: nicht kollidierende Strahlen; rot: kollidierende Strahlen.

In dem Quell-Code des Expert Bots wurden bereits acht Strahlen benutzt, die auf der Hüfthöhe des Agenten seinen Umkreis in Winkeln von 45° horizontal überprüft haben (s. Abbildung 21). Mit ihrer Hilfe war es möglich, herauszukriegen, ob und wie weit sich um den Agenten herum Hindernisse befanden und somit Kollisionen mit Wänden und anderen Objekten zu verhindern. Falls jedoch links von ihm eine Bodenlücke war, konnte dies nicht erkannt werden, sodass der Bot weiterhin nach links laufen würde. Um dies zu lösen, wurden zwei weitere Strahlen hinzugefügt, die – wie in Abbildung 21 zu sehen ist – nach links und rechts unten von dem Charakter zeigen und es ermöglichen, die Umgebung nicht nur nach Hindernissen, sondern auch nach Bodenlücken zu überprüfen. So konnte nun eine flüssige Bewegung des Agenten in dem Beobachtungszustand gesichert werden.

Anschließend wurde überlegt, welche weiteren Verhaltensmuster für Menschen typisch sind, wenn sie andere Spieler beobachten. Aufgrund der frühen Videos und Beschreibungen wurde angenommen, dass menschliche Spieler ihre Gegner testen könnten, um anhand ihrer Reaktionen ihre „Menschlichkeit“ beurteilen zu können. So könnte zum Beispiel ein Mensch einzelne Schüsse in die Nähe des Gegners abfeuern und sehen, wie dieser darauf reagiert. Nach den Annahmen mancher Jurymitglieder würde ein menschlicher Gegner in diesem Fall erkennen, dass der erste Spieler selbst ein Mensch ist und diesen mit Hilfe der Bewertungswaffe bewerten. Aufgrund dieser Überlegungen wurde anschließend die Funktion „testPlayer“ implementiert, die es dem OvGUBot in dem Beobachtungszustand erlaubte, seine Gegenspieler

zu testen. Hierbei sollte der Agent mit einer kleinen Wahrscheinlichkeit einige Schüsse mit der Bewertungswaffe etwas höher als der Gegner abschießen und falls der Gegner daraufhin mit der Bewertungswaffe antwortete, ihn bewerten. Die Bewertungsfunktion selbst wurde wiederum in der Klasse „PrimaryState“ implementiert, da sie in mehreren Zuständen aufgerufen werden konnte und wird in dem Kapitel 4.2.3 beschrieben.

Aufgrund eigener Erfahrungen und des Feedbacks anderer Spieler während Testspielrunden sollte der Bot sich nicht zu lange in dem Beobachtungszustand befinden, da sein Verhalten sonst als „zu passiv“ beurteilt werden könnte. So wurde die maximale Zeit des Befindens in dem Beobachtungszustand auf 15 Sekunden begrenzt. Außerdem sollte der Agent mindestens 20 Sekunden lang ein anderes Verhallten zeigen, bevor er den „Observe“-Zustand wieder betreten durfte. Falls er einen Spieler kürzlich bewertet hat, sollte er außerdem erst nach mindestens 90 Sekunden denselben Spieler beobachten können. Dies war damit verbunden, dass durch die Bewertung der Bot zeigen würde, dass er bereits eine „Meinung“ über den Spieler gebildet hat und ihn vorerst nicht mehr beobachten muss.

4.2.2.3 Attack

Die Bewegungen in dem „Attack“-Zustand mussten im Prinzip ähnlich denen, im „Observe“-Zustand sein. Menschliche Spieler bewegen sich normalerweise in einem Kampf von einer Seite zur anderen um somit den Projektilen des Gegners auszuweichen. In diesem Sinne wurde das Pendelverhalten, welches im vorigen Abschnitt bereits beschrieben wurde, auch für das Attackieren übernommen. Auch die Verfolgung des Ziels war genauso wichtig beim Attackieren, wie beim Beobachten, sodass auch dieses Verhalten übernommen werden konnte. So konnte es möglich sein, den Gegner zu verfolgen, falls dieser weglief, zu fokussieren und gleichzeitig seinen Schüssen auszuweichen.

4.2.2.4 Hide

Wie bereits erwähnt, bestand die Idee hinter dem „Hide“-Zustand darin, den Agenten in einem Versteckplatz auf Gegner warten zu lassen um sie dann zu überraschen. Um einen solchen Platz zu finden, musste der Aufbau der Welt bedacht werden, sodass der Agent einen Ort auswählen konnte, der schlecht von anderen Orten aus sichtbar wäre. Enge Flure oder Raumecken, die durch andere Objekte verdeckt sind, wären gute Beispiele für Versteckorte. An dieser Stelle wurde entschieden, die von Pogamut angebotene Methode der Sichtbarkeitsmatrix (Visibility Matrix) zu nutzen, um herauszufinden, welche Orte einer Map am wenigsten gesehen werden konnten. Eine Sichtbarkeitsmatrix speicherte dabei für jeden Navigationspunkt Informationen darüber, von welchen anderen Orten dieser sichtbar war. Diese Matrix musste somit einmalig vor dem Start einer Spielrunde für jede benutzte Welt erstellt werden. Um herauszufinden, ob ein Navigationspunkt B von dem Punkt A aus sichtbar war, wurde hierfür ein Strahl von A nach B gesendet und überprüft, ob dieser mit irgendwelchen Objekten kollidierte. Aus den Informationen der Sichtbarkeitsmatrix konnte nun für jeden Punkt entnommen werden, wie viele Punkte von ihm aus sichtbar waren, beziehungsweise von wie vielen aus dieser sichtbar war. So war es nun möglich, die Punkte zu finden, die von den wenigsten Orten aus gesehen werden konnten.

Sobald ein solcher Punkt in der Sichtweite des Agenten war und der Agent dabei keinen anderen Spieler sah, sollte er sich zu diesem Versteck bewegen. Da solche Punkte jedoch, wie bereits erwähnt, oft in einer Ecke sind, könnte es als „unmenschlich“ empfunden werden, wenn der Bot an dem Punkt einfach stehen bleiben würde um auf Gegner zu warten, ohne selbst eine gute Sicht zu haben. Aus diesem Grund war es wichtig, den Agenten zwischen den Nachbarpunkten des Verstecks pendeln zu lassen um dadurch das „Umschauen und Warten“ zu simulieren. Hierfür wurde eine Liste aller Nachbarn des Punktes aus dem Navigationsgraph erstellt, sobald der „Hide“-Zustand betreten wurde. Solange sich der Agent in diesem Zustand befand, wurde ein Nachbarpunkt aus der erstellten Liste zufällig gewählt (außer dem Punkt, an dem sich der Agent gerade befand) und der Bot zu diesem Punkt geschickt. Angekommen an

dem Ort sollte sich der Agent eine kurze Zeit lang (5 Sek.) an diesem Ort befinden um das Warten zu simulieren. Dabei sollte er sich außerdem in Winkeln von maximal 3° nach links oder rechts drehen und mit einer Wahrscheinlichkeit von 40% kleine Schritte zur Seite machen um somit das Umschauen zu verdeutlichen. Um zu garantieren, dass der Agent dabei nicht in Richtung einer Wand schaute, wurde auch hier mit Hilfe der bereits erwähnten Strahlen die Umgebung überprüft und der Charakter gegebenenfalls von der Wand weggedreht.

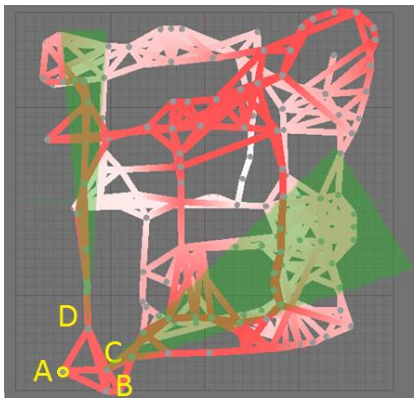

Abbildung 22: Beispiel eines Versteckpunktes. Sicht des Navigationsgraphen von oben.
A - Versteckpunkt; B/C/D – Punkte, zwischen denen der Agent pendelt; grün - Sichträume des Agenten von den Punkten D/C.

In Abbildung 22 ist zur Veranschaulichung die Heatmap eines Navigationsgraphen dargestellt, wobei die grauen Punkte die Navigationspunkte/Knoten und die weiß-roten Linien die Kanten darstellen. Hier könnte beispielsweise der Punkt A als Versteckpunkt gewählt werden, da dieser in einer Ecke liegt und von wenigen anderen Orten aus gesehen wird. Sobald der Agent den Versteckzustand betreten würde, würden die Punkte B, C und D als Nachbarn des Punktes A erkannt werden und der Agent würde zwischen ihnen pendeln. Wenn der Agent an den Punkten C und D stehen bleiben und sich „umschauen“ würde, könnte er zum Beispiel längere Flure in seinem Blickfeld (grün dargestellt) behalten und somit Gegner früh erkennen.

Auch für diesen Zustand wurde die maximale Aufenthaltsdauer nach einigen Testspielen auf 25 Sekunden beschränkt, damit sich der Agent nicht zu lange in einem Bereich aufhielt. Außerdem wurde die Mindestzeit, die vergehen musste, bevor der „Hide“-Zustand erneut betreten werden durfte, auf 15 Sekunden gesetzt. So konnte sich der Agent weit genug von dem vorigen Versteckort entfernen um ein neues Ziel zu finden.

Da menschliche Spieler in der Lage sind ihre Umgebung stets zu erkunden und unabhängig von ihrer derzeitigen Mission kleine Aufgaben auszuführen, würden sie beim Verstecken oder Beobachten anderer Spieler in ihrer Nähe auftauchende Items nebenbei aufsammeln. Aus diesem Grund wurde sowohl in dem „Hide“-Zustand, als auch in den Zuständen „Retreat“, „Observe“ und „Attack“ stets überprüft, ob ein Item mit einer hohen Priorität (vgl. Kapitel 4.2.1) sich in der unmittelbaren Nähe des Agenten befand und dieses gegebenenfalls aufgesammelt ohne den Zustand zu verlassen.

4.2.2.5 Greedy

Der „Greedy“-Zustand sollte, wie bereits erwähnt, immer dann betreten werden, wenn die Bedingungen für keinen der anderen Zustände erfüllt waren oder wenn der Agent bestimmte Gegenstände benötigte. Dieser Zustand war bereits in der Implementierung des Expert Bots vorhanden, allerdings wurde darin zusätzlich zu der „EstimateDestination“- Methode (vgl. Kapitel 4.2.1) nochmals nach dem besten Item gesucht. Da die Zielsuche jedoch, wie in Kapitel 4.2.1 beschrieben, für den OvGUBot bereits seine Bedürfnisse und Werte berücksichtigte und das passende Item unabhängig von dem aktuellen Zustand gefunden wurde, war eine erneute Itemsuche in dem „Greedy“-Zustand nicht notwendig. So musste der OvGUBot in diesem

Zustand lediglich zu diesem Item entlang des berechneten Pfades laufen. Da der Zustand außerdem als Default-Zustand galt, brauchte dieser keine Begrenzungen der Aufenthaltsdauer oder Bedingungen, sodass keine „ConditionsAreTrue“-Methode in der „Greedy“-Klasse implementiert werden musste.

4.2.2.6 Zustandsbedingungen

Im Folgenden werden die Bedingungen einzelner Zustände als Pseudocodes der „ConditionsAreTrue“-Methoden der ersten vier Zustandsklassen beschrieben. Diese werden in der Reihenfolge aufgelistet, in der sie aufgerufen und überprüft werden, sodass ersichtlich sein kann, wie die einzelnen Bedingungen zusammenhängen.

Pseudocode 1: Retreat.ConditionsAreTrue

```
Input: double game time, int health

Output: Boolean showing whether or not to enter the state „Retreat“

1.  if players are visible and weaponry state < 5 then

2.      return true

3.  endif

4.  if bot last hit < 15s. ago and (health < 30 or weaponry state < 5) then

5.      return true

6.  endif

7.  if bot last hit > 15s. ago then

8.      return false

9.  endif
```

Pseudocode 2: Observe.ConditionsAreTrue

```
Input: AgentInfo info, double game time

Output: Boolean showing whether or not to enter the state „Observe“

1.  if bot last hit < 15s. ago then

2.      return false

3.  endif

4.  if no target found to observe then

5.      return false

6.  endif

7.  if bot is at last seen target’s position but there is no visible enemy then

8.      return false

9.  endif

10. if bot judged the target < 90s. ago then

11.     return false

12. endif
```

```
13. if bot.Deaths >= 10 and target killed the bot >= bot.Death/2 times then
14.     return false
15. endif
16. if bot was observing < minTimeBetweenSameState then
17.     return false
18. endif
19. if bot is observing > secondsToBeInState then
20.     return false
21. else
22.     return true
23. endif
```

Pseudocode 3: Attack.ConditionsAreTrue

```
Input: AgentInfo info, double game time

Output: Boolean showing whether or not to enter the state „Attack"

1.  if bot last hit < 15s. ago then
2.      return true
3.  endif
4.  if bot last hit > 15s. ago and (health < 40 or weaponry state < 10) then
5.      return false
6.  endif
7.  if no target selected to attack then
8.      return false
9.  endif
10. if bot has "Double Damage" or bot's armor >=50 then
11.     return true
12. endif
13. if time last met target < 20s. and  bot is at last seen target's position
    but there is no visible enemy then
14.     return false
15. endif
16. if time last met target < 20s. then
17.     return true
18. else
19.     return false
20. endif
```

Pseudocode 4: Hide.ConditionsAreTrue

```
Input: double game time, int health

Output: Boolean showing whether or not to enter the state „Hide"

1.  if no hiding point found then
2.      return false
3.  endif
4.  if health <60 or weaponry state < 5 then
5.      return false
6.  endif
7.  if shooting players are visible then
8.      return false
9.  endif
10. if bot was hiding < minTimeBetweenSameState then
11.     return false
12. endif
13. if bot is hiding > secondsToBeInState then
14.     return false
15. else
16.     return true
17. endif
```

Wie in den oberen Pseudocodes zu sehen ist, müssen nicht nur die Zeit, die Gegner und die Gesundheit des Agenten bestimmte Werte annehmen um die Zustandsbedingungen zu erfüllen, sondern auch die Variable „weaponry state" (Waffenzustand). Diese wurde eingeführt um herauszufinden, wie viel Munition der Agent im Vergleich zu der maximal möglichen Munitionsanzahl zurzeit besitzt. Somit konnte eingeschätzt werden, ob der Agent noch in der Lage war, Gegner zu attackieren ohne nach Gegenständen zu suchen oder er sich auf die Suche nach Items begeben musste. Für die Berechnung des aktuellen Waffenzustands wurde für jede der acht in Kapitel 4.2.1 genannten Waffen im Spiel (ausgenommen Bewertungswaffe und die „Shield Gun", da diese keine Munition benötigte) die Anzahl der vorhandenen Munition relativ zu der maximalen Anzahl dieser berechnet (vgl. Formel 4.2.2.6.1). Anschließend wurden die relativen Munitionsstände mit einem Gewichtsfaktor multipliziert und zu dem Waffenzustand zusammengerechnet (vgl. Formel 4.2.2.6.2). Dabei haben die vier Waffen, die bei der Benutzung höhere Prioritäten hatten (vgl. Tabelle 1) Gewichtsfaktoren von 0,15 bekommen und Waffen, die der Agent seltener benutzen würde, einen Gewichtsfaktor von 0,10 (vgl. Formel 4.2.2.6.3).

Formel 4.2.2.6.1: RelAmmo(Weapon$_i$) – relativer Munitionszustand einer Waffe

$$RelAmmo(Weapon_i) = \frac{Ammo(Weapon_i)}{MaxAmmo(Weapon_i)}$$

Ammo(Weapon$_i$) – Anzahl der vorhandenen Munition der Waffe mit dem Index i (i∈{1,..,8})
MaxAmmo(Weapon$_i$) – maximale Anzahl der Munition der Waffe mit dem Index i
RelAmmo(Weapon$_i$) – relativer Munitionsstand der Waffe mit dem Index i berechnet aus der Anzahl der vorhandenen Munition relativ zu der maximalen Anzahl der Munition dieser Waffe

Formel 4.2.2.6.2: WeaponryState – aktueller Waffenzustand

$$WeaponryState = \sum_{i=1}^{8} RelAmmo(Weapon_i) * W_i$$

WeaponryState – Angabe über den aktuellen Waffenzustand des Agenten zusammengesetzt aus der Summe der gewichteten relativen Munitionsstände aller Waffen wobei gilt:

Formel 4.2.2.6.3: W$_i$ – Gewicht des rel. Munitionsstandes einer Waffe

$$W_i = \begin{cases} 0{,}15 & if\ Weapon_i \in \{Flac\ Canon, Minigun, Rocket\ Launcher, Shock\ Rifle\} \\ 0{,}10 & if\ Weapon_i \in \{Bio\ Rifle, Lighting\ Gun, Assault\ Rifle, Sniper\ Rifle\} \end{cases}$$

W_i – Gewicht des relativen Munitionsstandes der Waffe mit dem Index i, wobei die Gewichte der Waffen mit höherer Priorität höher sind als die, derjenigen mit einer niedrigeren Priorität

Nach diesen Berechnungen hätte der Agent einen Waffenzustand von 100, wenn er für jede Waffe die maximale Munitionsanzahl hätte, sodass die Variable „weaponry state“ als eine Angabe über den Waffenzustand in Prozent gesehen werden konnte. In den oben aufgeführten Pseudocodes ist zu sehen, dass zum Beispiel der Zustand „Retreat“ betreten wurde, wenn der Waffenzustand des Agenten niedriger als 5 war und der Zustand „Hide“ dabei nicht betreten werden konnten (vgl. Pseudocode 1 und Pseudocode 4). Diese Bedingungen entstanden dadurch, dass jeder Spielcharakter beim Start einer Spielrunde und nach seinem Tod bereits eine Grundmenge an Munition besaß, welche einen Waffenzustands-Wert von 7,5 ergab. Mit diesem Wert war es noch möglich Gegner zu attackieren, sodass ein Rückzug erst ab einem niedrigeren Wert nötig war. Bei einem Wert von weniger als 5 sollte sich der Bot jedoch auf keinen Fall verstecken, sondern nach Munitionsitems suchen, da er anderenfalls beim Auftauchen eines Gegners nicht in der Lage wäre, diesen lange genug zu attackieren. Der Wert wurde dabei nach jedem Respawn des Agenten neuberechnet, sowie wenn der Spielcharakter ein Munitions- oder Waffenitem aufhob und wenn er aufhörte zu schießen. Außer für die Überprüfung der Zustandsbedingungen wurde der Wert des Waffenzustands außerdem bei der im Kapitel 4.2.1 beschriebenen Suche nach dem besten Item berücksichtigt.

Die Größen der Gesundheitswerte, die in den Bedingungen einzelner Zustände überprüft wurden, ergaben sich nach einigen Testspielen und Befragungen anderer Spieler. Mit den endgültigen Werten und Bedingungen sollte der Agent einen wenig offensiven Charakter zeigen, sondern mehr auf das Beobachten und Bewerten konzentriert sein. Die Entscheidung den Charakter so zu konzipieren hing vor allem mit der bereits beschriebenen Beobachtung zusammen, dass das Ziel des Spiels bei dem Turing-Test stark verschoben wurde und nicht mehr das Erzielen der meisten Punkte im Vordergrund stand.

4.2.3 Weitere Implementierungsdetails

Nachdem das Grundgerüst aus den einzelnen Zuständen implementiert wurde, mussten viele einzelne Zusätze und Optimierungen des Botverhaltens hinzugefügt werden. Die Meisten dieser Änderungen wurden aufgrund des Feedbacks und der Beschreibungen voriger Arbeiten vorgenommen. So wurde zum Beispiel eine gewisse Schussungenauigkeit hinzugefügt, die in

der Implementierung des Expert Bots nicht vorhanden war. Bisher konnten manche Bots sehr einfach erkannt werden, wenn sie unabhängig von der Entfernung zum Gegner und ihrer Geschwindigkeit den Gegner sehr genau trafen, was nicht der Fall bei menschlichen Spielern war. So wurde für den OvGUBot beim Schießen ganz nach dem Vorbild des UT^2-Bots ein Offset zu der Position des Gegners hinzugefügt, der von den Geschwindigkeiten des Agenten sowie des Gegners und der Entfernung zu dem Gegner abhing.

Eine weitere große Änderung an dem Quellcode des Expert Bots bestand in der Speicherung der Informationen über Gegner. Ein menschlicher Spieler kann sich die Namen und Avatare anderer Spieler merken und sich zum Beispiel daran erinnern, wie oft er bestimmte Gegner bereits getroffen hat, wie sie sich ihm gegenüber verhalten haben und entsprechend bei erneuten Begegnungen agieren. Er kann einen bestimmten Gegner über längere Zeit verfolgen oder vor ihm weglaufen ohne ihn mit anderen Spielern zu verwechseln oder sich von ihnen ablenken zu lassen. Damit ein Bot „menschlich“ wirkt, ist es für ihn also wichtig auch dieses „Erinnerungsvermögen“ zu simulieren und gewisse Gegnerinformationen zu speichern. Da der Expert Bot jedoch nur in 1-gegen-1-Spielen trainiert wurde, musste er nicht zwischen mehreren Gegnern unterscheiden. Außerdem konnte hierbei davon ausgegangen werden, dass sich Werte wie die Gesundheit des Gegners nur dann änderten, wenn der Expert Bot diesen attackierte, da sonst kein Spieler ihn verletzt haben könnte. Aus diesem Grund wurden bei der Implementierung des Expert Bots Werte wie Gesundheitszustand und Waffenzustand des einen einzigen Gegners abgespeichert. Diese wurden am Anfang der Spielrunde sowie bei Interaktionen mit dem Gegner geschätzt und aktualisiert.

Der OvGUBot wurde jedoch mit dem Ziel entwickelt, gegen mehrere Gegner zu spielen. Um ihm eine Unterscheidung der Gegner zu ermöglichen und genauere Informationen über sie zu speichern, wurde die Klasse „EnemyInfo“ stark verändert. Es wurde nun je eine Instanz dieser Klasse für jeden Spieler in einer Hashmap angelegt, die durch die eindeutigen UnrealIDs einen schnellen Zugriff auf die Informationen ermöglichte. Die Gesundheits- und Waffenzustände anderer Spieler konnten nun nicht mehr so einfach geschätzt werden, da diese sich gegenseitig attackieren könnten, wenn sie nicht in dem Sichtfeld des OvGUBot waren, sodass er diese Änderungen nicht mehr verfolgen konnte. Es wurden aber Informationen hinzugefügt, die mehr über das Verhalten der anderen Spieler gegenüber dem Bot verraten sollten. So wurden zum Beispiel die Zeitpunkte gespeichert, an denen der Gegner von dem OvGUBot zuletzt beobachtet oder bewertet wurde, beziehungsweise diesen selbst bewertete oder attackierte. Außerdem wurde für jeden Spieler mit Hilfe mehrerer Hashmaps gespeichert, wie oft dieser von anderen Spielern getötet wurde oder sie selbst tötete und wie oft er als Bot, beziehungsweise Mensch bewertet wurde. Vor allem die Informationen über die Bewertungen anderer Spieler wurden als wichtig angesehen um sie später bei der Bewertung durch den OvGUBot zu nutzen (s.u.). Das Wissen darüber, wie oft ein Spieler den OvGUBot getötet hat, wurde zum Beispiel genutzt, um die Aggressivität dieses Spielers einzuschätzen. So sollte der Bot beispielsweise einen aggressiven Spieler nicht mehr beobachten wollen, sondern ihn gleich attackieren, was in der Bedingung für den „Observe“-Zustand in Pseudocode 2 Z.13 gesehen werden kann. Die Position, an der ein Spieler zuletzt von dem OvGUBot gesehen wurde, wurde weiterhin gespeichert und bei der bereits beschriebenen Verfolgung des Spielers benutzt.

Im Zusammenhang mit der Speicherung der Spielerinformationen wurde im weiteren Schritt die Auswahl des nächsten Zielgegners implementiert, die bisher auch nicht für den Expert Bot notwendig war. Hierbei wurde überlegt, nach welchen Kriterien ein menschlicher Spieler auswählen würde, wen er als nächstes attackieren oder beobachten würde. Die Entscheidung wäre einfach, wenn nur ein Spieler sichtbar wäre. Bei der Auswahl zwischen mehreren Spielern würde ein Mensch wiederum nach dem Verhalten der anderen Spieler ihm gegenüber entscheiden. Er würde zum Beispiel eher einen Gegner attackieren, der ihn selbst vor kurzer Zeit angegriffen hat, als denjenigen, der sich ihm gegenüber friedlich verhalten hat. Um eine nachvollziehbare Auswahl des Zielspielers auch dem OvGUBot zu ermöglichen, wurden die bereits erwähnten Informationen über die Zeitpunkte der Interaktionen mit den Spielern benutzt. Hatte der Bot noch keinen Gegner, sei es weil die Spielrunde erst angefangen hat, oder weil der

vorige Zielgegner gestorben ist, so musste der Bot einen neuen Spieler auswählen. Hierfür wurden zuerst alle Spieler in Betracht gezogen, die der Agent innerhalb der letzten 15 Sekunden gesehen hat. Diese Zeit schien unter der Berücksichtigung des schnellen Spielverlaufs sinnvoll zu sein. Aus diesen Spielern wurde danach der Spieler ausgesucht, der den Agenten zuletzt bewertet oder attackiert hat. Falls keiner der Spieler mit dem Agenten interagiert hatte, wurde derjenige ausgewählt, der sich am nächsten zu dem Agenten befand und für ihn sichtbar war.

Da bereits aus den vorigen Arbeiten bekannt war, dass der Aspekt der Fokussierung auf einen Gegner für einen Spieler sehr wichtig war um „menschlich“ zu wirken, wurde versucht auch den OvGUBot so zu implementieren, dass er über eine längere Zeit sich auf einen Spieler konzentrierte. So wurde generell ein Spieler so lange als Ziel behalten, bis entweder er, oder der Bot getötet wurde. Die Fokussierung auf einen Gegner sollte jedoch mit einer Ausnahme abgebrochen werden können. Falls der Agent einen Zielspieler ausgewählt hatte, dieser ihn aber länger nicht (mehr) attackierte und der Agent von einem anderen Spieler angegriffen wurde, konnte der Agent sein Ziel wechseln. Diese Entscheidung war ebenfalls damit verbunden, dass ein Mensch erkennen würde, dass es für ihn von höherer Priorität ist, denjenigen zu eliminieren, der ihn angreift, als jemanden zu verfolgen, der ihm keinen Schaden zufügt.

Der letzte wichtige Punkt, der die „Menschlichkeit“ des OvGUBots (in dem BotPrize-Wettbewerb) steigern sollte, war die Fähigkeit andere Spieler als „Mensch/Bot“ zu bewerten. Wie bereits in dem Kapitel 4.2.2.2 erwähnt, wurde hierfür eine Bewertungsfunktion in der Klasse „PrimaryState“ implementiert, die aus den Zuständen „Attack“ und „Observe“ aufgerufen werden konnte. Dabei gab es zwei Möglichkeiten den Zielspieler zu bewerten, je nachdem „wie sicher der OvGUBot war“. Das heißt, zum einen konnte die Bewertungsfunktion mit einem Booleanwert als Parameter aufgerufen werden, der angab, ob der Spieler als ein „Bot“ bewertet wird. Zum anderen konnte eine leicht veränderte Funktion aufgerufen werden, in der anhand der gespeicherten Informationen darüber, wie der Gegner von anderen Spielern bewertet wurde, entschieden wurde, wie er zu bewerten ist. Die erste Variante sollte zum Beispiel aufgerufen werden, wenn der Agent selbst von dem Gegner bewertet wurde, da hierbei davon ausgegangen werden konnte, dass der Gegner eher ein Mensch ist. Die zweite Variante wurde aufgerufen nachdem der Agent einen Spieler lange genug beobachtet hatte. Bei dem Wettbewerb war es erlaubt, seine Meinung und Bewertung beliebig oft zu ändern. Es wurde allerding in beiden Funktionen darauf geachtet, dass der Agent keinen Spieler öfter als einmal in 120 Sekunden bewertete, da es sonst zu „unmenschlich“ erscheinen würde, wenn er seine „Meinung“ zu oft ändern würde.

Das Bewertungsmodell mit den zwei unterschiedlichen Funktionen wurde vor allem aus dem Grund so entwickelt, dass angenommen wurde, dass die Spieler sehen könnten „wie“ sie bewertet werden, wenn mit unterschiedlichen Modi der Bewertungswaffe auf sie geschossen wurde. Allerdings hat sich bereits nach der ersten Spielrunde des BotPrize-Wettbewerbs herausgestellt, dass die Bewertungswaffe so modifiziert wurde, dass die Projektile beider Schussmodi gleich aussahen. Somit konnten also andere Spieler zwar sehen, dass sie bewertet wurden, aber nicht wie. Eine einfache Bewertungsfunktion mit demselben Schussmodus für alle Spieler hätte also an dieser Stelle gereicht.

Außer den hier beschriebenen grundlegenden Elementen wurden viele weitere kleine Details an dem Code des Expert Bots geändert oder zu ihm hinzugefügt. Einige Methoden und Parameter wurden im Verlauf des BotPrize-Wettbewerbs optimiert, sodass die finale Version des OvGUBots in dem Quellcode im Anhang gesehen werden kann. Im Folgenden werden der Verlauf und die Ergebnisse des BotPrize-Wettbewerbs selbst, sowie weiterer Tests beschrieben.

5 Evaluierung

Wie bereits am Anfang dieser Arbeit erwähnt, wurde der OvGUBot vor allem entwickelt um den angepassten Turing-Test für Computerspiele bei dem BotPrize-Wettbewerb zu bestehen. Der Wettbewerb war besonders deswegen wichtig, weil er schon mehrmals durchgeführt wurde und die dabei erzielten Ergebnisse mit denen, der letzten Jahre verglichen werden konnten. Außerdem nahmen mehrere andere Bots an dem Wettbewerb teil, sodass bestimmte Bedingungen vorherrschten und die Möglichkeit bestand, die Agenten untereinander zu vergleichen. Da jedoch nicht auf alle Informationsdetails des Wettbewerbs zugegriffen werden konnte, wurden anschließend eigene Tests durchgeführt. So war es möglich mit den teilnehmenden Spielern zu sprechen und zu erfahren, welche Aspekte für sie bei der Bewertung des Bots wichtig waren. Beide Abläufe und ihre Ergebnisse werden in diesem Abschnitt beschrieben.

5.1 BotPrize-Wettbewerb 2014

5.1.1 Durchführung

Der BotPrize-Wettbewerb wurde im Jahr 2014 zum sechsten Mal durchgeführt. Es nahmen insgesamt vier Wettbewerber aus vier unterschiedlichen Ländern daran teil, von denen drei sich zum ersten Mal beworben haben. Neben dem OvGUBot waren es der in Südkorea entwickelte BotTracker und der in Spanien entwickelte NizorBot (vgl. Human-like Bots 3 2014). Der vierte Teilnehmer war MirrorBot, der bereits im Jahr 2012 nach einer erfolgreichen Teilnahme den Turing-Test bestand (vgl. Kapitel 3.1.4). Außerdem haben zwei weitere Agenten – ADANN und CCBot – partizipiert, die jedoch keinen Anspruch auf die Auszeichnung des Wettbewerbs hatten, da sie von seinen Veranstaltern entwickelt wurden.

Genau wie in den Vorjahren bestand der Wettbewerb aus zwei unterschiedlichen Bewertungsphasen, die in Kapitel 2.4 bereits beschrieben wurden. Die erste Phase bestand daraus, dass die eingereichten Agenten und menschliche Spieler in mehreren Spielrunden gemeinsam spielen sollten und die letzten dabei aus der Erste-Person-Perspektive alle anderen Spieler bewerten sollten. Im Gegensatz zu den vorigen Wettbewerben wurden dieses Jahr die Botprogramme jedoch nicht an die Veranstalter geschickt und lokal gestartet. Stattdessen mussten die Entwickler zu vereinbarten Zeitpunkten ihre Agenten mit dem externen Spielserver verbinden. Eine weitere organisatorische Änderung bestand darin, dass nun die menschlichen Spieler nicht mehr unabhängige Drittpersonen waren, sondern die Entwickler der Bots selbst mitgespielt und mitbewertet haben. Das heißt jeder Entwickler musste seinen Agenten mit dem Server verbinden und sich selbst in das Spiel einloggen und aktiv am Bewertungsprozess mitwirken. Dies galt jedoch nicht für die Veranstalter. Diese haben nur ihre Agenten mit dem Spiel verbunden, sodass insgesamt zehn Spielcharaktere (sechs Bots und vier Menschen) sich im Spiel befanden.

Es wurden insgesamt fünf Spielrunden von je 15 Minuten gespielt. Die Spiele fanden vom 12. Juni bis 26. Juni statt, sodass zwischen zwei Runden jeweils mindestens ein Tag lag, an dem die Entwickler die Möglichkeit hatten, ihre Agenten weiterhin zu verändern und zu verbessern. Während der Spielrunden bestand das Ziel jedes menschlichen Spielers darin, möglichst alle anderen Spielcharaktere mit Hilfe der bereits beschriebenen Bewertungswaffe als Mensch oder Bot zu kennzeichnen. Dabei konnte sich jeder beliebig oft umentscheiden. Auch Bots konnten die Bewertungswaffe nutzen, allerdings wurden ihre Stimmen nicht in die Berechnung der

Ergebnisse einbezogen. Es wurden die, in dem Kapitel 2.4 beschriebenen Modifikationen an dem Spiel vorgenommen und auch die Berechnung wurde wie beschrieben durchgeführt.

Für die Spielrunden wurden fünf unterschiedliche Maps gewählt, die sich in ihren Merkmalen stark unterscheidet haben, wodurch das Verhalten der Spieler in sehr unterschiedlichen Umgebungen beobachtet werden konnte. Die Maps wurden in der folgenden Reihenfolge gespielt: Grendelkeep, Deck 17, TokaraForest, Calandras, Grendelkeep. Grendelkeep und Deck17 ähnelten sich optisch sehr stark, da beide ein leeres Industriegebäude von innen darstellten. Deck17 war allerdings sehr viel kleiner als die erste Map, wodurch es viel öfter zu Interaktionen mit anderen Spielern kam und wenig Raum zum Ausweichen oder Zurückziehen gab. TokaraForest stellte dagegen eine absolut andere Spielwelt dar. Es handelte sich hierbei um eine sehr große Map, die aus einem Wald mit überdimensionierten Bäumen bestand. Durch die vielen Ebenen und weite offene Flächen konnten die Spieler einen guten Überblick über einen großen Teil der Welt haben. Außerdem wurde eine geringere Schwerkraft auf dieser Map simuliert, sodass alle Spielcharaktere sehr hoch springen konnten und langsam runtergefallen sind. Die vierte Map Calandras entsprach in ihrer Größe ungefähr der ersten Map und stellte zerstörte Gebäude dar, sodass sowohl offene Flächen, als auch enge Korridore vorhanden waren. In der fünften Spielrunde wurde auf derselben Map gespielt, wie in der ersten Runde, wodurch es möglich war, die Fortschritte der Bots (und Menschen) über die Zeit zu sehen.

Während der Spielrunden wurden alle Spielcharaktere von den Veranstaltern aus der Dritte-Person-Perspektive beobachtet und ihre Spielweisen auf Videos aufgezeichnet. Diese Videos sollten Drittpersonen gezeigt werden, um die einzelnen Spielcharaktere in der zweiten Phase der Bewertung als Bot oder Mensch von ihnen bestimmen zu lassen. Hierfür wurde jeweils ein ca. 2 Minuten langes Video von jedem Spielcharakter aus der ersten Spielrunde über eine Crowd-Sourcing-Plattform geteilt. Der Ausschnitt von 2 Minuten wurde laut Veranstaltern in der Mitte der gespielten 15 Minuten zufällig gewählt. Alle zehn Videos können unter (HumanLike BotCom 2014) angesehen werden. Die Teilnehmer konnten gegen eine kleine Entlohnung die Videos einzeln anschauen und beliebig viele davon bewerten. An dem zweiten Part der Bewertung nahmen insgesamt 232 Personen teil (vgl. Arrabales 2014). Dabei hat jedoch nicht jeder Teilnehmer alle 10 Videos bewertet, was daran gesehen werden kann, dass während der Erstellung dieser Arbeit die zehn Videoaufnahmen durchschnittlich ca. 35 Youtube-Aufrufe hatten (Stand 1.10.2014). Da jeder Nutzer der verwendeten Plattform teilnehmen konnte, ist außerdem nichts über diese Personen bekannt und es kann somit nicht angegeben werden, wie viel Erfahrung mit Computerspielen diese Personen hatten.

5.1.2 Ergebnisse

Während der Spielrunden haben alle Spielcharaktere zufällige Namen und Avatare bekommen, sodass die menschlichen Spieler nicht erkennen konnten, wen sie wie bewertet haben. Die Bewertungsergebnisse wurden auf dem Spielserver in einer Logdatei gespeichert und im Nachhinein von den Veranstaltern des BotPrize-Wettbewerbs ausgewertet. Um jedoch zu vermeiden, dass irgendwelche Rückschlüsse aus den Ergebnissen der ersten Bewertungsphase gezogen werden konnten, wurden die Resultate zusammen mit den Ergebnissen der zweiten Bewertungsphase erst am 29.08.2014 auf der IEEE Conference on Coputational Intelligence vorgestellt.

Die Werte der „Menschlichkeit“ einzelner Spieler wurden hierbei, wie es in dem Kapitel 2.4 beschrieben ist, berechnet. Dabei zählten die Werte der Erste-Person-Bewertung (FPA) und der Dritte-Person-Bewertung (TPA) zu gleichem Anteil. Jeder dieser Werte setzte sich wiederum aus den gewichteten Mitteln der Bewertungen einzelner Richter zusammen, wobei das Gewicht der Stimme eines Richters von seiner Genauigkeit/Zuverlässigkeit abhing (vgl. Formel 2.4.1 bis Formel 2.4.6).

Abbildung 23 stellt die Ergebnisse der ersten Bewertungsphase dar. Die ersten sechs Spieler sind dabei die eingereichten Bots und die letzten Spieler sind ihre Entwickler. In der zweiten Spalte sind die nicht gewichteten durchschnittlichen Resultate der Erste-Person-Bewertungen. Die dritte Spalte zeigt dabei die Resultate, die unter Berücksichtigung der Zuverlässigkeit einzelner Jurymitglieder berechnet wurden.

Calculating FPA (First-Person Assessment)

Weighted First-Person Humanness Ratio

BotName	Humanness	FPA
MirrorBot	0.4996406	0.20164771
BotTracker	0.4231043	0.20070203
OvGUBot	0.3164826	0.10545765
NizorBot	0.2980527	0.11821633
ADANN	0.2432864	0.08351664
CCBot	0.1685606	0.06214746

BotName	Humanness	FPA
Player	0.5417464	0.1932813
tmchojo	0.5177169	0.1775752
Xenija	0.3847691	0.1713976
Juan_CVC	0.3172348	0.1237229

Abbildung 23: Ergebnisse der Erste-Person-Bewertungen.
6 erste Spieler sind Bots, 4 letzte Spieler sind Menschen. Humanness – nichtgewichtetes Mittel der Erste-Person-Bewertungen. FPA – gewichtetes Mittel der Erste-Person-Bewertungen. In (Quelle: Arrabales 2014).

Während der ersten Bewertungsphase hat in dem BotPrize-Wettbewerb der Gewinner des vorigen Wettbewerbs MirrorBot die besten Ergebnisse unter allen Agenten erzielt, aber trotzdem die 50%-Schranke der „Menschlichkeit“ verfehlt. In Abbildung 23 ist zu sehen, dass sogar bei den ungewichteten Ergebnissen kein Bot es geschafft hat, die Richter von seiner „Menschlichkeit“ zu überzeugen und 50% zu erreichen. Es ist jedoch zu beachten, dass nur zwei Menschen selbst mehr als 50% „Menschlichkeit“ erzielt haben und somit nicht behauptet werden kann, dass die Schranke zwischen „menschlich“ und „unmenschlich“ bei 50% liegt. Die Unterschiede zwischen den ersten zwei Bots und den Menschen sind sehr klein, sodass bereits hierbei gesagt werden kann, dass diese Bots während der Spielrunden sich sehr menschlich verhalten haben. Dies wird besonders dadurch bestätigt, dass unter Berücksichtigung der Richterzuverlässigkeit die Agenten MirrorBot und BotTracker sogar einen höheren FPA-Wert bekommen haben, als alle menschlichen Spieler. Interessant ist, dass MirrorBot dabei sogar den Menschen „Player“ überholt hat (vgl. Abbildung 23 rote Markierung). OvGUBot konnte in der ersten Bewertungsphase einen „Menschlichkeitswert“ von ~0.32 erzielen und war mit einem FPA-Wert von ~0,11 auf dem vierten Platz unter allen Bots.

Bei dem Vergleich der Resultate der letzten vier Bots ist auffällig, dass die Unterschiede ihres FPA-Werts zu dem niedrigsten FPA-Wert eines Menschen, der bei ~0.12 liegt, kleiner sind als die Unterschiede zu den FPA-Werten der ersten zwei Bots. Dass die letzten vier Agenten einen so „großen“ Unterschied zu den ersten zwei Agenten haben, könnte damit erklärt werden, dass während der Spielrunden bekannt war, dass die Anzahl der Menschen und Bots ungefähr gleich war. Aus diesem Grund haben die meisten Teilnehmer, laut ihren Aussagen, versucht ungefähr fünf Spielcharaktere als Mensch und fünf andere als Bot zu kennzeichnen. Ihre Entscheidung hing also nicht nur von dem Verhalten der anderen Spieler ab, sondern auch von der Anzahl der bereits bewerteten Spieler. So kann angenommen werden, dass durch ihr weniger „menschliches“ Verhalten im Vergleich zu anderen Spielern die Bots NizorBot, OvGUBot, ADANN, CCBot die niedrigsten „Menschlichkeitswerte“ bekommen haben. Dabei könnte der Spieler „Juan_CVC“ womöglich oft als Bot gekennzeichnet werden, sodass die Verteilung „5 Menschen/5Bots“ während der Spielrunden aufging.

Weiterhin fällt auf, dass die FPA-Werte sich stark von den ungewichteten „Menschlichkeitswerten“ unterscheiden. Dies gibt an, dass die Zuverlässigkeit der Jury im

Durchschnitt nicht sehr hoch war, was wiederum bedeutet, dass die menschlichen Spieler bei ihren Bewertungen oft falsch lagen und Bots und Menschen verwechselt haben. Für alle Agenten ist es also ein positives Zeichen und bestätigt, dass es während der Spiele generell schwer fiel, ihr Verhalten von dem menschlichen zu unterscheiden.

Im Gegensatz zu der Erste-Person-Bewertung war den Benutzern der Crowd-Sourcing-Plattform bei der Teilnahme an der Dritte-Person-Bewertung nicht bekannt, wie viele Menschen und Bots an dem Spiel teilnahmen. Sie wurden lediglich gebeten, sich die Videos anzuschauen und zu entscheiden, ob jeweils der gefilmte Spieler ein Mensch oder ein Bot war. Anschließend haben sie ihre Entscheidung mit einem kurzen Kommentar begründet. Da jedoch, wie bereits erwähnt, die Bewertung jedes Videos einzeln stattgefunden hat und nicht jeder Teilnehmer alle Spieler bewertet hat, konnte an dieser Stelle nicht die Zuverlässigkeit/Genauigkeit der Richter berücksichtigt werden. So wurde bei der Berechnung der Ergebnisse der Dritte-Person-Bewertung nicht wie geplant und in dem Kapitel 2.4 beschrieben, der gewichtete Durchschnitt der einzelnen Wertungen genommen, sondern lediglich der relative Anteil der Bewertungen eines Bots als „Mensch“, wie in es der Formel 5.1.2.1 dargestellt ist.

Formel 5.1.2.1: $TPA_{i,j}$ – Ergebnis der Dritte-Person-Bewertung

$$TPA_{i,j} = \frac{Miss_{i,j}}{N_{i,j}}$$

$Miss_{i,j}$ – Anzahl der Bewertungen des Bots i als „Mensch“ durch den Richter j
$N_{i,j}$ – Anzahl aller durch den Richter j am Spieler i vorgenommenen Bewertungen
I – Anzahl der Spieler
J – Anzahl der Richter (232)
$TPA_{i,j}$ – Ergebnis der zweiten Bewertungsphase für Spieler i durch den Richter j

Die herausgekommenen Ergebnisse waren sehr positiv für alle Agenten. Wie in der Abbildung 24 zu sehen ist, haben dabei alle Spieler bis auf den Bot ADANN eine Wertung von mehr als 50% bekommen, wobei OvGUBot einen TPA-Wert von ~0,61 erzielte. Diese Zahlen können so verstanden werden, dass fast alle Spieler mehr als 50% der Richter davon überzeugt haben, dass sie eher Menschen als Bots sind. Auch das Ergebnis des Spielcharakters ADANN liegt nur knapp unter den 50%.

Calculating TPA (Third-Person Assessment)

Crowdsourcing Judging

BotName	FPA	TPA	H++
Xenija	0.17139763	0.8235294	0.4974635
MirrorBot	0.20164771	0.7333333	0.4674905
Player	0.19328127	0.6315789	0.4124301
tmchojo	0.17757519	0.6470588	0.4123170
NizorBot	0.11821633	0.7058824	0.4120493
BotTracker	0.20070203	0.5909091	0.3958056
CCBot	0.06214746	0.7058824	0.3840149
Juan_CVC	0.12372294	0.6190476	0.3713853
OvGUBot	0.10545765	0.6086957	0.3570767
ADANN	0.08351664	0.4761905	0.2798536

Abbildung 24: Ergebnisse beider Bewertungsphasen.

FPA – gewichtetes Mittel der Erste-Person-Bewertungen, TPA – gewichtetes Mittel der Dritte-Person-Bewertungen, H++ – Gesamtergebnis als Durchschnitt von FPA und TPA. In (Quelle: Arrabales 2014).

Die Unterschiede zwischen den TPA-Werten einzelner Spieler sind sehr klein und bis auf die Werte der Spielcharaktere Xenija und ADANN gibt es keine bemerkenswerten Ausreißer. Dies kann wieder darauf hinweisen, dass die Verhaltensweisen der Menschen und Bots sehr ähnlich

waren. Es ist allerdings schwierig, genau zu bestimmen, wie diese Zahlen zustande gekommen sind und wonach hierbei geurteilt wurde. Besonders unter der Berücksichtigung der Tatsache, dass nicht alle Richter alle Videos gesehen haben, können Vergleiche zwischen den Werten schlecht gezogen werden. Die von den Teilnehmern der zweiten Bewertungsphase hinterlassenen Kommentare für die Spielcharaktere Xenija und OvGUBot können hierfür in dem Anhang A angeschaut werden. Interessant ist hierbei, wie unterschiedlich die Wahrnehmung der einzelnen Beobachter war. Zum Beispiel wurde der OvGUBot für einen Menschen gehalten mit der Begründung „movement and quick responses“, andere Beobachter schrieben dazu jedoch „week response to attack“ oder „the characters playing had a lot of delay reactions like bots…“ und bewerteten den Charakter als Bot (vgl. Anhang A). Die Reaktionen des Bots wurden hierbei von verschiedenen Personen also sehr unterschiedlich wahrgenommen, was zu unterschiedlichen Bewertungen geführt hat. Dies geschah auch mit der Wahrnehmung der Bewegung des Agenten, so schrieb ein Beobachter „movement seemed fluid like controlled by machine…“ und ein anderer dagegen „the character sometimes stops to act along the level of the game for no reason…“ (vgl. Anhang A). Diese Beispiele zeigen, wie schwer es ist, ein Verhalten zu simulieren, welches von allen Beobachtern als „menschlich“ anerkannt wird, da die Bewertungen doch sehr von den Wahrnehmungen der Beobachter abhängen.

Die Bewertungen waren jedoch nicht nur von unterschiedlichen Wahrnehmungen abhängig. Sogar in Fällen, in denen das Verhalten des Bots ähnlich wahrgenommen wurde, wurde seine „Menschlichkeit“ unterschiedlich bewertet. So wurde der OvGUBot zum Beispiel aufgrund der Vielzahl von zufälligen Aktionen als ein Bot bewertet und mit dem Kommentar versehen „Random and unnecessary shooting/jumping/movement…“. Die zufälligen Aktionen wurden aber von einem anderen Beobachter als sinnvoll bewertet, sodass der OvGUBot als Mensch markiert wurde. Das Kommentar hierbei lautete:

„This virtual player was controlled by a Human because there were a few moments when this player would shoot at the walls. I can see how even an AI player can shoot and hit the wall by missing a player but when you have aimed at the wall and fire, that seems more like a Human thing to do. Also, there was a moment when the player would jump on the moving platform then jump off, jump on and then jump off again. With whatever is controlling this character there seems to be some thought going behind this and some form of strategy which is another reason why I think it's a Human.“ (vgl. Anhang A)

Die Auswertung der Kommentare ist an dieser Stelle also sehr schwierig. Sehr viele von ihnen sind subjektiv und können sich teilweise widersprechen. Im Allgemeinen können die Kommentare jedoch weiterhelfen, da deutlich wird, auf welche Aspekte Beobachter mehr achten. So wurde in den vorhandenen Kommentaren sehr oft auf die Bewegung eingegangen und nach ihr beurteilt und erst danach wurde das Schießverhalten und andere Details bewertet.

Obwohl der OvGUBot einige negative Kommentare bezüglich seiner zufälligen Aktionen und seiner Bewegungsweise erhalten hat, konnte er doch in der zweiten Bewertungsphase mehr als die Hälfte der Jury-Mitglieder von seiner Menschlichkeit überzeugen. Dieses Ergebnis kann als zufriedenstellend bewertet werden, vor allem unter Berücksichtigung der Tatsache, dass das Video während der ersten von fünf Spielrunden aufgezeichnet wurde. Im Verlauf des Wettbewerbs wurden einige Details an seinem Verhalten angepasst, wodurch das Ergebnis in einer späteren Spielrunde womöglich noch besser aussehen würde, was jedoch an dieser Stelle nicht überprüft werden kann. Insgesamt hat der Agent eine Wertung von ~0,357 bekommen (vgl. Abbildung 24) und bekam somit den vierten Platz unter allen teilnehmenden Bots in dem BotPrize-Wettbewerb 2014 und den neunten Platz unter allen Spielern. Auch das Gesamtergebnis kann als befriedigend angesehen werden. Auch wenn der Agent keine 50% „Menschlichkeit“ erzielen konnte, so ist das Ergebnis doch im Vergleich zu anderen Werten sehr positiv. Wie bereits erwähnt, kann die 50%-Schwelle dabei nicht mehr als die Grenze zwischen „Bot“ und „Mensch“ angesehen werden, da auch keiner der Menschen sie überschreiten konnte. Insofern kann an dieser Stelle behauptet werden, dass alle Teilnehmer des

Wettbewerbs im Jahr 2014 gute Leistungen erbracht haben und den Turing-Test bei einer anderen Modifikation bestehen könnten.

5.1.3 Anmerkungen

Die Ergebnisse des BotPrize-Wettbewerbs zeigen, dass der angepasste Turing-Test für Computerspiele nicht so gut, wie gedacht, funktionieren kann. Dies liegt einerseits daran, dass die Vorstellung von der „Menschlichkeit“ sehr subjektiv ist und diese somit schwer mit Hilfe von Formeln gemessen werden kann. Andererseits können einige Aspekte der Durchführung des Wettbewerbs/Turing-Tests als solchen eventuell verbessert werden.

Wie sehr die Bewertung der „Menschlichkeit“ von der Wahrnehmung einzelner Personen und ihren Vorstellungen abhängt, wurde bereits in dem vorigen Abschnitt angesprochen. Es ist dabei natürlich unumstritten, dass unterschiedliche Meinungen von unterschiedlichen Menschen berücksichtigt werden sollen, allerdings sollte darauf geachtet werden, dass all diese Menschen dieselben Informationen bekommen und sich gleichmäßig an der Bewertung beteiligen. Dies war vor allem bei der zweiten Bewertungsphase nicht der Fall. Es wurde hierbei nicht sichergestellt, dass jeder Beobachter die Videos von allen Spielern ansah und bewertete. Somit können diese Bewertungen kaum miteinander verglichen werden. An dieser Stelle wäre es möglicherweise nicht die Videos einzeln zu verlinken und bewerten zu lassen, sondern gleich alle zehn zusammen.

Ein Kritikpunkt ist auch an den Videoaufnahmen selbst zu bringen. Die Videos waren jeweils ca. zwei Minuten lang, obwohl die Spielrunden je 15 Minuten lang dauerten. Auch wenn die Zeit bei allen Spielern zufällig ausgesucht wurde und jeder Spielcharakter gleich behandelt wurde, so wäre es doch sinnvoller, das Verhalten der Spieler während der ganzen Spielrunde aufzunehmen und bewerten zu lassen. Es ist nicht ausgeschlossen, dass manche Spielcharaktere eine bestimmte Strategie während der Spielrunden haben könnten und sich zu manchen Zeitpunkten anders verhalten haben, als zu anderen. Durch die Videoausschnitte konnten jedoch diese Entwicklungen den Beobachtern vorenthalten bleiben wodurch ihre Bewertung möglicherweise anders ausfiel, als bei der Betrachtung des Gesamtbildes.

In diesem Zusammenhang ist außerdem zu kritisieren, dass in die Erste-Person-Bewertung alle fünf Spielrunden eingingen und in die Dritte-Person-Bewertung nur die erste Spielrunde. Dabei zählten jedoch die Teilergebnisse beider Bewertungsphasen gleichermaßen bei der Berechnung des Gesamtergebnisses. Es ist zwar verständlich, dass es schwierig wäre, so viele Jury-Mitglieder dazu zu bringen, die Videos mit einer Länge von je 15 Minuten von zehn Spielern aus fünf Spielrunden zu bewerten. Eine alternative Lösung könnte jedoch in den nächsten Wettbewerben sinnvoll sein. Zum Beispiel könnten die Beobachter sich zeitgleich mit den Spielern mit dem Spielserver verbinden und das Spielgeschehen sofort beobachten. Dies würde außerdem einige Arbeitsschritte den Veranstaltern des Wettbewerbs ersparen.

Ein weiterer Punkt, der an dieser Stelle angesprochen werden sollte, ist die Beteiligung der Entwickler der Bots an den Spielen. Auch wenn alle Spieler durch die Veränderung ihrer Namen und Avatare anonymisiert wurden, so kannte doch jeder Entwickler die Besonderheiten bei dem Verhalten seines Agenten und konnte sie möglicherweise in bestimmten Situationen wiedererkennen. So kann nicht ausgeschlossen werden, dass die Bewertungen in der ersten Phase des Wettbewerbs immer fair waren. Zwei Spieler sind zum Beispiel über die Spielrunden hinweg dadurch aufgefallen, dass einer sich als einziger hingehockt hat und der andere sehr aggressiv gegenüber allen Spielern war. Es ist nicht bekannt, ob diese Spielcharaktere Bots oder Menschen waren, aber im Falle von Bots, könnten ihre Entwickler sie leicht identifizieren. Laut den Veranstaltern des Wettbewerbs haben in den vergangenen Jahren unbeteiligte Personen gegen die Bots gespielt, doch konnten dieses Jahr nicht genügend Bereitwillige gefunden werden, sodass die Entwickler hinzugezogen wurden. Für die nächsten Wettbewerbe ist es sicherlich sinnvoll wieder unbeteiligte Personen an dem Spiel teilnehmen zu lassen.

Auch wäre es besser, die Botprogramme wieder den Veranstaltern zu schicken und sie lokal starten zu lassen. Bei der Verbindung mit dem externen Spielserver kam es beim OvGUBot zum Beispiel zu sehr starken Verzögerungen. Die Pfadberechnung durch GameBots konnte teilweise nicht durchgeführt werden, wodurch der Agent mehrere Sekunden lang an einer Stelle stehen blieb. Dies war jedoch nicht der Fall bei einer Verbindung mit einem lokalen Server. Nach einigen Gesprächen mit den Veranstaltern und anderen Teilnehmern, stellte sich heraus, dass ähnliche Verzögerungen auch bei ihnen auftraten. Aus diesem Grund sind lokale Spielrunden eine bessere Variante, solange die Verbindungsprobleme während des Wettbewerbs nicht behoben werden können.

Weiterhin sollte bei der Planung der Spielrunden besser darauf geachtet werden, welche Maps hierfür benutzt werden oder die Teilnehmer sollten besser darüber informiert werden. So wurde zum Beispiel vor dem Anfang des Wettbewerbs bekannt gegeben, dass nur auf offiziellen Maps gespielt wird, zu denen die in der dritten Spielrunde benutzte Map Calandras allerdings nicht gehörte. Dies ergab zum Beispiel für den OvGUBot ein Problem, da dieser, wie in dem Kapitel 4.2.2.4 beschrieben, in dem „Hide“-Zustand eine vorerstellte Sichtbarkeitsmatrix von jeder Map benötigte, um nach Versteckpunkten zu suchen. Für die Map Calandras wurde jedoch keine Matrix erstellt, da diese Map nicht beim Kauf des offiziellen Spiels vorhanden war und von allen Teilnehmern vor dem Start der Spielrunde heruntergeladen werden musste. Aus diesem Grund war es dem OvGUBot nicht möglich, sich zu verstecken, sodass nur die anderen Zustände betreten werden konnten.

Die wichtigste Frage besteht jedoch in der Tauglichkeit des BotPrize-Wettbewerbs als Turing-Test, oder sogar vielmehr in der Tauglichkeit des gewählten Spiels. Wie bereits in dem Kapitel 2.4 beschrieben, wurde der Wettbewerb gestartet, um Methoden zu finden, mit denen Agenten entwickelt werden könnten, die sich sehr „menschlich“ in einem Spiel verhalten würden und somit die Immersion der Spieler steigern könnten. Das ursprüngliche Ziel war es also, Bots für die unveränderte Variante eines Spiels zu entwickelt. Durch die Anpassung an den Turing-Test und die Einführung des Bewertungsaspekts während des Spielens wurde das Ziel des Spiels jedoch sehr stark verändert. Auch die Spielumgebung und -Bedingungen entsprachen durch den reduzierten Schaden nicht den Originalen. Die meisten Bots, die für den Wettbewerb entwickelt wurden, hatten dabei als Ziel eher die Simulation des Richterverhaltens, als eines Spielers während eines Standard-Deathmatch-Spiels. Die menschlichen Spieler mussten nun auch nicht mehr die Spielrunde durch die meisten Punkte gewinnen, sondern möglichst genau die Menschen von den Bots unterscheiden.

An dieser Stelle kann für die durchgeführten Modifikationen die Tatsache sprechen, dass bei dem ursprünglichen Test die Teilnehmer auch wissen, dass sie gerade bewertet werden und dadurch ihr Verhalten womöglich von dem Normalen abweicht. Allerdings muss der Mensch, der dabei mit dem Prüfer kommuniziert sich nicht für den Prüfer ausgeben (vgl. Kapitel 2.1). So wäre es womöglich sinnvoller, die Bots nicht während des Spiels zu bewerten, oder dies zumindest nicht so offensichtlich zu tun, um ihr Verhalten dadurch nicht in eine unerwünschte Richtung zu verändern.

Auch wäre es besser, die Konditionen des Spiels nicht zu verändern. In Unreal Tournament 2004 wurden sie für den Wettbewerb so stark verändert, dass das normalerweise sehr schnelle Shooter-Spiel zu einem langsamen Bewertungsspiel wurde. Es ist verständlich, dass die Bewertungen schwierig wären, wenn unter den „normalen“ Bedingungen nicht genug Zeit vorhanden wäre, um das Verhalten der Spieler bewerten zu können. So wäre es an dieser Stelle möglicherweise sinnvoller, statt Unreal Tournament 2004 ein Spiel zu nehmen, welches standardmäßig langsamer verläuft, aber ähnliche Bedingungen bietet. Als Beispiel könnte das Spiel „Battlefield 1942“, oder ein anderes Spiel der Battlefield-Reihe sich dafür eignen. Diese Spiele verfügen auch über einen Deathmatch-Modus, legen jedoch mehr Wert auf Taktik, als auf Schnelligkeit. Durch eine gute Taktik könnten die Bots hierbei sogar noch besser ein „menschliches“ Verhalten zeigen. So wäre es möglich, sie sowohl während des Spiels, als auch im Nachhinein in dem Turing-Test zu bewerten, ohne das Spielprinzip zu ändern und die Bots

könnten genauso weiterhin in das Spiel übernommen werden um gegen Online-Spieler zu spielen und ihre Immersion zu steigern.

5.2 Lokale Tests

5.2.1 Durchführung

Nach den Spielrunden des BotPrize-Wettbewerbs wurden anschließend einige Testspiele mit dem OvGUBot lokal durchgeführt. Dies diente vor allem dazu, mehr Feedback über das Verhalten des Bots, sowie über den Turing-Test von Personen zu holen, die sich nicht mit der Entwicklung von Bots beschäftigten, jedoch Shooterspiele spielten. Hierbei wurden ähnliche Bedingungen wie in dem BotPrize-Wettbewerb geschaffen. Es wurden insgesamt fünf Spielrunden à 15 Minuten auf unterschiedlichen Maps gespielt. Dabei wurden Maps gewählt, die denen aus dem Wettbewerb von ihrer Größe und Struktur ähnelten. So wurden die Welten Grendelkeep, Antalus, Osiris, Curse4, TokaraForest ausgewählt. Es nahmen fünf Personen an den Tests teil, jedoch konnten in jeder Spielrunde immer nur fünf Spieler inklusive OvGUBot teilnehmen. Um zu sehen, ob sich die Wahrnehmung der Menschen ändert, wenn sie nicht wissen, ob und wie viele Bots gerade mit ihnen Spielen, haben in zwei Runden je zwei unterschiedliche Bots und drei Menschen gegeneinander gespielt. Die menschlichen Spieler haben dabei vor jeder Spielrunde auf einem Zettel eine zufällige Aufgabe bekommen. Entweder durften sie in dieser Runde mitspielen, oder sollten ein Einzelspieler-Spiel starten, um den Eindruck zu erwecken. Ihre Aufgabe durften sie nicht erläutern, sodass kein anderer Spieler Bescheid wusste, wer und wie viele von den anderen vier Spielern sich gerade im Spiel befinden. Die Plätze derjenigen Spieler, die ein Einzelspieler-Spiel gespielt haben, wurden mit Bots „aufgefüllt“. Die Verteilung der Spieler kann Tabelle 3 entnommen werden.

Wie bereits in dem Kapitel 4.2.2.6 erwähnt, wurden bestimmte Werte, wie der Waffen- oder der Gesundheitszustand des OvGUBots überprüft, um zu entscheiden, welchen Zustand der Agent betreten sollte. Sein Verhalten hing also stark von diesen Werten ab. Um zu sehen, ob sich die Meinung der menschlichen Spieler über das Verhalten des Bots ändert, wurden diese Werte so verändert, dass der Bot ein aggressiveres Verhalten zeigen sollte, als dies normalerweise der Fall war. Der Agent mit den modifizierten Werten hat zusammen mit dem Bot mit den „Standard-Werten“ und drei Menschen in der zweiten Spielrunde gespielt. Außerdem nahm an der dritten Spielrunde der Agent UT^2, welcher 2012 den Turing-Test bestand (vgl. Kapitel 3.2.2), zusammen mit dem OvGUBot teil. Hierbei sollten die Unterschiede zwischen diesen zwei Agenten betrachtet werden.

Außerdem sollten weitere Meinungen zu den Modifikationen an dem Spiel, die für den BotPrize-Wettbewerb durchgeführt wurden und in dem Kapitel 5.1.3 bereits kritisiert wurden gesammelt werden. Hierfür wurde die erste Spielrunde ohne die Bewertungswaffe und im Standard-Deathmatch-Modus gespielt. In allen anderen Spielrunden wurde anschließend die Bewertungswaffe eingeführt und der Schaden – genau wie im Wettbewerb – auf 40% gesenkt. So „verlangsamte“ sich das Spielgeschehen auch in den lokalen Testspielen.

Die Spielweisen aller Spielcharaktere wurden dabei jeweils aus der Erste-Person-Perspektive auf Videos aufgenommen um sie im Nachhinein wiederum von weiteren Personen auswerten zu lassen. An dieser Stelle wäre es sinnvoller, die Spielcharaktere aus der Dritte-Person-Perspektive zu beobachten. So könnte nämlich besser überprüft werden, wie das Verhalten des jeweiligen Spielcharakters von Drittpersonen wahrgenommen wird. Hierfür müsste allerdings die gleiche Anzahl an Beobachtern, wie an Spielern mit dem Spielserver verbunden sein, was jedoch aus Mangel an Helfern nicht möglich war.

Auf den Videos waren jedoch nach der Aufnahme Informationen zu sehen, welche die einzelnen Spieler als Mensch oder Bot verraten würden. So sahen zum Beispiel die Fadenkreuze der menschlichen Spieler anders, als die der Bots aus. Außerdem wurde bei den Bots am rechten

Rand des Bildschirmes ihr Name angezeigt, dafür jedoch keine Nachricht, wenn sie ihre Waffe gewechselt haben. Aus diesem Grund mussten bestimmte Stellen des gezeigten Bildschirms verdeckt werden um somit die Spieler nicht zu verraten. Wie die Abbildung 25 zeigt, konnte dabei die Spielumgebung trotzdem gut genug gesehen werden.

Abbildung 25: Screenshot aus einem Video, welches die Spielweise eines Testspielers aus der Erste-Person-Perspektive zeigt. Schwarze Rechtecke verdecken Informationen, die den Spieler als Mensch oder Bot identifizieren würden.

Die Videos von je fünf Spielern aus allen fünf Spielrunden wurden anschließend auf ungefähr die gleiche Länge von ca. zehn Minuten geschnitten. Dies war nötig, weil manche Spieler zum Beispiel am Anfang der Spielrunde das Spielmenü geöffnet haben, was ein Bot nicht tun könnte. So mussten solche Stellen am Anfang rausgeschnitten werden.

Anschließend wurde je ein Bewertungsbogen von zehn weiteren Personen ausgefüllt, in dem für jeden Spieler angegeben werden konnte, wie „menschlich“ dessen Verhalten erscheint und welche Merkmale bei den Entscheidungen besonders wichtig waren. Die Liste der in Frage kommenden Merkmale wurde in Anlehnung an die, durch den Entwickler des Mirror-Bots herausgefilterten Merkmale, erstellt (vgl. Polceanu 2013). Der Bewertungsbogen ist im Anhang B zu sehen. Alle bewertenden Personen, hatten genau wie die Testspieler Erfahrungen mit Computerspielen und wussten, was das Ziel eines Deathmatch-Spiels ist.

5.2.2 Ergebnisse

Bei den lokalen Testspielen konnte der OvGUBot die bewertenden Personen kaum von seiner Menschlichkeit überzeugen. Tabelle 3 zeigt die Bewertungsergebnisse, die aus den einzelnen Bewertungsbögen zusammengefasst wurden. Damit die teilnehmenden Personen in unterschiedlichen Runden nicht erkannt werden konnten, wurden sie in den Videos als „Spieler 1 – 5“ angegeben (Spalte 3). Welche Personen sich hinter den Spielcharakteren verbargen, kann in der zweiten Spalte abgelesen werden. Die Werte in der Tabelle geben die von den zehn Richtern geschätzte „Menschlichkeit“ der Spieler in Prozent an sowie den daraus resultierenden Durchschnitt. Die Werte der teilgenommen Bots sind dabei grün hervorgehoben. Außerdem OvGUBot nahm in der zweiten Runde der bereits erwähnte modifizierte, leicht aggressivere OvGUBot2 teil und in der dritten Runde der UT^2-Bot.

Tabelle 3: Ergebnisse der Bewertungen der Spielermenschlichkeit anhand der Videos von lokalen Testspielen.

Runde	Spieler	Richter/ Video	A	B	C	D	E	F	G	H	I	J	K	Ø
R1	Person1	Spieler 1	60	35	75	90	60	80	65	50	70	78	70	**66,64**
	OvGUBot	Spieler 2	20	40	25	10	0	20	20	15	35	17	10	**19,27**
	Person2	Spieler 3	100	36	90	100	70	75	80	70	80	70	90	**78,27**
	Person3	Spieler 4	100	29	90	100	100	75	85	70	90	66	95	**81,82**
	Person5	Spieler 5	100	38	90	40	20	50	55	60	70	22	75	**56,36**
R2	Person3	Spieler 1	90	32	90	25	100	85	70	55	85	57	90	**70,82**
	OvGUBot2	Spieler 2	10	28	25	10	40	30	20	10	15	30	45	**23,91**
	OvGUBot	Spieler 3	0	28	25	0	20	30	25	5	20	17	20	**17,27**
	Person1	Spieler 4	100	42	60	90	80	95	75	70	85	68	90	**77,73**
	Person2	Spieler 5	90	35	75	80	50	80	70	60	70	66	75	**68,27**
R3	UT^2	Spieler 1	0	31	25	40	10	40	20	10	25	17	45	**23,91**
	OvGUBot	Spieler 2	20	27	25	25	20	40	25	15	25	22	50	**26,73**
	Person4	Spieler 3	90	35	50	100	80	80	70	60	95	66	75	**72,82**
	Person2	Spieler 4	100	40	75	90	90	90	80	65	90	73	85	**79,82**
	Person5	Spieler 5	80	36	90	80	70	95	75	60	80	70	95	**75,55**
R4	Person4	Spieler 1	80	37	75	90	40	90	65	75	70	45	90	**68,82**
	Person2	Spieler 2	100	29	25	100	20	90	60	50	65	45	80	**60,36**
	OvGUBot	Spieler 3	10	29	25	0	0	5	10	5	25	0	20	**11,73**
	Person1	Spieler 4	100	38	70	90	90	85	80	75	80	66	95	**79,00**
	Person5	Spieler 5	100	34	70	80	80	80	75	70	80	63	80	**73,82**
R5	Person4	Spieler 1	70	34	75	90	90	95	70	65	90	75	80	**75,82**
	Person2	Spieler 2	90	32	60	70	80	90	65	55	85	66	85	**70,73**
	Person5	Spieler 3	100	30	60	100	90	95	75	60	95	71	95	**79,18**
	OvGUBot	Spieler 4	0	28	25	0	0	30	15	0	20	17	20	**14,09**
	Parson1	Spieler 5	90	34	50	90	70	90	70	50	90	60	85	**70,82**

Anhand dieser Ergebnisse ist erkennbar, dass nicht nur der OvGUBot, sondern auch der UT^2-Bot, der den Turing-Test im Jahr 2012 bestand, die Beobachter nicht überzeugen konnten. Beide Agenten hatten eine durchschnittliche „Menschlichkeit“ von ca. 20%, wogegen die menschlichen Spieler im Durchschnitt zu 70% „menschlich“ wirkten. Dabei ist auffällig, dass keine besonderen Unterschiede zwischen der ersten Spielrunde, die keine Modifikationen hatte und den restlichen Runden, in denen die Bewertungswaffe benutzt wurde und der Schaden reduziert wurde, vorhanden sind. Auch der Unterschied zwischen dem OvGUBot und seiner modifizierten aggressiveren Version in der zweiten Spielrunde ist nicht bemerkenswert. Dass die Unterschiede zwischen den Menschen und den Bots so viel höher sind, als bei den Ergebnissen des BotPrize-Wettbewerbs, kann mehrere Ursachen haben.

Zum einen haben an dem Wettbewerb mehrere Bots teilgenommen, die sich in ihrem Verhalten möglicherweise ähnelten, sodass es für Beobachter schwer war unter den vielen sich ähnlich verhaltenden Spielcharakteren Bots rauszusuchen. Da während der lokalen Spieler der OvGUBot aber größtenteils als einziger Bot gegen mehrere Menschen gespielt hat, ist sein Verhalten vermutlich stärker aufgefallen.

Zum anderen kann eine Ursache der Bewertungsaspekt in dem Spiel selbst sein. Während des Wettbewerbs haben alle Teilnehmer versucht andere Spieler möglichst genau zu bewerten und haben dadurch vermutlich langsamer und zurückhaltender gespielt. In den lokalen Spielrunden konnten die Spieler zwar auch die Bewertungswaffe benutzen, es war ihnen allerdings weniger wichtig dabei genau zu sein, da sie ihre Zuverlässigkeit/Genauigkeit nicht steigern mussten. Somit konnten sie sich womöglich erlauben offensiver und schneller zu spielen, sodass der Spaßfaktor hierbei den Bewertungsfaktor überwog. Der OvGUBot wurde jedoch mit dem Ziel entwickelt, andere Spieler zu bewerten und sich eher zurückzuhalten, sodass sein Verhalten dadurch den Beobachtern möglicherweise mehr auffiel. Dies würde auch die Kritik an dem BotPrize-Wettbewerb hinsichtlich des verschobenen Spielziels stärken.

Der wichtigste Grund dafür, dass die Bots so „unmenschlich“ wirkten, liegt jedoch vermutlich darin, dass die Videos aus der Erste-Person-Perspektive aufgenommen. Die menschlichen Spieler konnten dabei selbst ihr Spiel aufnehmen, sodass das Video genau das gezeigt hat, was sie sehen konnten. Da die Agenten jedoch selbst nicht „sehen“ konnten, musste ihr Verhalten mit Hilfe der „Zuschauer“-Funktion (Spectator) in dem Spiel aufgenommen werden. Hierdurch war die Kamerasicht leicht verändert im Vergleich zu der, der menschlichen Spieler. So war die Kamera zwar immer in die Blickrichtung des Bots gedreht, wirkte jedoch immer etwas starr. Abbildung 26 zeigt zum Beispiel links die „Sicht“ eines Bots. Dieser schießt dabei genau auf den Gegner, der durch den gelben Pfeil dargestellt ist. Allerdings ist das Fadenkreuz („X“) nicht über dem Gegner, sondern leicht daneben. Im rechten Bildteil ist die Sicht eines menschlichen Spielers, der genau auf den markierten Gegner schießt, wobei sein Fadenkreuz diesen verdeckt. Dass die Unterschiede der Kameraführung den Beobachtern aufgefallen sind, lässt sich anhand der folgenden Kommentare erkennen: „z.B. auf der letzten Map schaut Einer nach unten von den Pfaden aus. Das ist etwas, was ein Bot nicht machen würde“ , „steife Kamera-Führung“ (s. Anhang C). Insofern wäre es tatsächlich sinnvoller, die Verhaltensweisen der Spielcharaktere aus der Dritte-Person-Perspektive aufzunehmen, um dieses Problem zu vermeiden. Dies war jedoch, wie bereits im vorigen Kapitel erwähnt, unter gegebenen Umständen nicht möglich.

Abbildung 26: Unterschied der Kamerasicht aus der Perspektive eines Bots (li.) und eines Menschen (re.). „X“ in der Mitte ist das Fadenkreuz, das Ziel ist durch den gelben Pfeil dargestellt.

Die Wichtigkeit der unterschiedlichen Verhaltensmerkmale für die Beobachter der Spiele kann Tabelle 4 entnommen werden. Hierbei wurde von jedem Beobachter für jedes Merkmal/Verhaltensmechanismus angegeben, wie wichtig dieses für sie bei der Entscheidung nach der „Menschlichkeit“ der Spieler war. Die Wertung ging von 0 (unwichtig) bis 10 (sehr wichtig). Die Durchschnittbewertungen können der letzten Spalte entnommen werden. Als besonders wichtig hat sich hierbei – genau wie in den Kommentaren zu den Wettbewerb-Videos – die Bewegung der Spieler herausgestellt. Aber auch die Fokussierung auf einen Gegner und seine Verfolgung haben große Rollen gespielt (vgl. grün hervorgehobene Zellen). Unter Berücksichtigung dieser Aspekte, kann nachvollzogen werden, warum der OvGUBot weniger „menschlich“ erschien. Auch wenn während der lokalen Spielrunden der Bot keine Pfadfindungsprobleme wie während des Wettbewerbs hatte, so bewegte er sich doch sehr nah an

den Navigationspunkten entlang. Dies fiel manchen Beobachtern auf und wurde sogar durch ein Kommentar vermerkt: „Folgt den grauen Steinen“ (s. Anhang C). Dass der Bot sich nicht stark genug auf den Gegner zu fokussieren schien, kann wiederum an der Kamerasicht und seinem zurückhaltenden Verhalten liegen.

Tabelle 4: Ergebnisse der Wichtigkeit unterschiedlicher Verhaltensmerkmale bei der Bewertung der „Menschlichkeit“ 0 (unwichtig) – 10 (sehr wichtig).

Merkmal / Richter	A	B	C	D	E	F	G	H	I	J	K	Ø
Blieb im Spiel stehen (z.B. um sich umzuschauen. Damit ist nicht das Steckenbleiben gemeint)	6	6	3	5	0	9	4	8	6	1	5	**4,82**
Benutzte die „Dodge“-Fähigkeit (sehr schnelle Bewegungen)	0	9	5	1	2	2	1	2	8	3	2	**3,18**
Ausweichen/Weglaufen, wenn im Nachteil	2	8	8	8	9	6	7	5	6	9	7	6,82
Verstecken um den Gegner auszutricksen	0	8	8	7	8	6	8	5	8	3	7	6,18
Fokussierte sich auf den Gegner	4	6	7	10	10	4	6	5	8	7	8	6,82
Imitation des Gegnerverhaltens	0	8	3	1	0	5	7	2	0	3	2	**2,82**
Langzeitgedächtnis	0	8	3	3	0	7	4	5	3	4	2	**3,55**
Limitierte Schussgenauigkeit	3	5	5	5	3	0	4	1	2	4	7	**3,55**
Ressourcensammeln basierend auf den Bedürfnissen	3	7	3	5	5	2	3	7	2	6	3	**4,18**
Situations-beobachtung	7	9	7	2	9	5	10	9	4	1	8	6,45
Reibungslose Bewegung	2	6	9	10	7	7	6	5	5	8	10	6,82
Zielverfolgung	2	7	7	10	7	8	7	4	6	8	9	6,82
Waffenauswahl kontextabhängig	4	8	3	1	5	2	6	4	1	2	6	**3,82**
Hohe Aggressivität	5	7	3	2	0	0	7	1	0	2	4	**2,82**
Erzielte viele Punkte im Spiel	0	7	3	0	0	2	1	1	2	6	0	**2,00**

Hohe Aggressivität und erzielte Punkte haben bei der Bewertung der Spieler im Vergleich zu anderen Mechanismen/Eigenschaften eher geringere Rollen gespielt. Dies zeigt, dass die Beobachter vermutlich davon ausgingen, dass Spieler mit unterschiedlich viel Erfahrung an den Spielrunden teilnehmen könnten und schwächere Leistungen nicht unbedingt „Unmenschlichkeit“ bedeuteten. Allgemein ist es jedoch auch hier schwierig, bestimmte Mechanismen herauszufiltern. Für viele von ihnen unterscheiden sich die Einschätzungen der Wichtigkeit sehr stark abhängig von der bewertenden Person. So ist zum Beispiel das

„Stehenbleiben im Spiel“ für Person E völlig unwichtig und Person F sieht diese Eigenschaft als eine der wichtigsten. Dies unterstreicht wieder, wie unterschiedlich die Vorstellung von „Menschlichkeit“ sein kann und wie subjektiv die Wahrnehmung des Verhaltens dabei ist.

5.2.3 Anmerkungen der Spieler

Die Benutzung der Bewertungswaffe wurde nicht nur während des BotPrize-Wettbewerbs kritisiert. Auch während der Testspiele gab es mehrmals negative Kritik seitens der Spieler bezüglich des Bewertungsaspekts. Die Bewertungsfunktion wurde allgemein als sehr „ablenkend“ und störend empfunden. Die Spieler bemerkten, dass es allgemein sehr schwierig war, ihre Gegner zu bewerten, da wenig Zeit zum Beobachten und Handeln vorhanden war. Bei den meisten Versuchen einen Gegner zu bewerten ist entweder der Spieler selbst oder der Gegner vorher gestorben. So musste sich der Spieler den Gegnernamen merken und versuchen, ihn erneut auf der Map aufzusuchen. Dies lenkte jedoch stark davon ab, auf andere Gegner zu achten und mit ihnen zu interagieren.

6 Schlussfolgerungen

6.1 Zusammenfassung

In der vorliegenden Arbeit wurde gezeigt, wie schwierig es ist, einen Agenten zu entwickeln, der „menschlich“ genug erscheint, um den Turing-Test zu bestehen. Vielfältige Versuche wurden bereits vorher unternommen, doch kaum einem Bot ist es bis heute gelungen, in dem Computerspiel Unreal Tournament 2004 für einen menschlichen Spieler gehalten zu werden.

In dieser Arbeit wurde der zum sechsten Mal stattgefundene BotPrize-Wettbewerb vorgestellt. Dieser sollte als eine angepasste Variante des Turing-Tests herausfinden, ob die auf verschiedene Weisen entwickelten Agenten während mehrerer Spielrunden mehrere Richter/Beobachter von ihrer Menschlichkeit überzeugen können. Es wurden einige der Teilnehmer der Wettbewerbe früherer Jahre genauer betrachtet. Die Architekturen der entwickelten Agenten ließen sich dabei in drei unterschiedliche Kategorien einteilen, wobei Vertreter jeder Kategorie bereits Topleistungen erzielen konnten. Die meisten der vorgestellten Agenten waren regelbasierte Systeme, deren Verhaltensweisen durch Zustände, ähnlich wie in einem Zustandsautomaten implementiert wurden. Einige der Agenten wurde (anschließend) mit Hilfe von neuronalen Netzen optimiert.

Der Expert Bot war dabei ein Vertreter der mit Hilfe eines genetischen Algorithmus entwickelten Agenten. Dieser wurde anschließend als Grundlage für die Entwicklung des im Rahmen dieser Arbeit entstandenen OvGUBots verwendet. Es wurde entschieden, einen regelbasierten Agenten zu entwickeln, da dadurch die Möglichkeit bestand, das Feedback voriger Wettbewerbe leicht durch bestimmte Regeln einzubringen. Außerdem zeigten die Ergebnisse der Vorjahre, dass regelbasierte Architekturen durchaus gute Resultate erzielen konnten. So wurden einige der Zustände des Expert Bots beibehalten und angepasst. Andere wurden wiederum entfernt oder durch neue ersetzt. Aufgrund der Beobachtung, dass das Ziel der Spiele während des Wettbewerbs darin bestand, andere Spieler zu bewerten, kamen die Zustände „Observe“ und „Hide“ zu den Verhaltensweisen des OvGUBots hinzu. Diese sollten es dem Agenten ermöglichen, außer sich aggressiv zu verhalten, auch seine Gegenspieler zu beobachten und zu bewerten. Weiterhin wurden einige Aspekte, wie zum Beispiel die Auswahl eines Ziels oder Speicherung der benötigten Informationen über Gegner implementiert, beziehungsweise angepasst.

Anschließend wurde der BotPrize-Wettbewerb 2014 beschrieben. Der OvGUBot nahm mit drei weiteren Agenten daran Teil und erzielte den vierten Platz. Hierbei spielten die Bots in fünf Spielrunden gegen mehrere menschliche Spieler, deren Rolle diesmal die Entwickler der Agenten selbst übernahmen. Die Menschen konnten während der Spielrunden alle Spieler als „Mensch/Bot“ bewerten, wodurch sich die Ergebnisse der ersten Phase ergaben. Anschließend wurden die Videos, die während dieser Spiele aufgenommen wurden, weiteren unabhängigen Personen gezeigt, woraufhin sie wiederum alle Spieler bewerten sollten. So wurden die Ergebnisse der zweiten Phase berechnet. Insgesamt ergaben sich jedoch für alle Teilnehmer Werte, die nicht die 50%-Grenze zur „Menschlichkeit“ überschritten. Laut der Definition des Turing-Tests könnte es bedeuten, dass sogar die Menschen den Test nicht bestanden. Aber gerade deswegen kann diese Definition nicht stimmen. Unter Berücksichtigung der Unterschiede zwischen den Resultaten der Menschen und der Bots ist eher zu sagen, dass alle Bots sehr „menschlich“ wirkten und somit den Turing-Test (nach einer anderen Definition) bestehen könnten. Aufgrund dieser Unstimmigkeiten und der Tatsache, dass die Entwickler ihre Bots mitbewerteten, wurde anschließend einige Kritik an dem Wettbewerb selbst gebracht.

Um zusätzlich weiteres Feedback von unbeteiligten Personen einzuholen, wurden im Anschluss an den Wettbewerb einige lokale Testspiele durchgeführt. Hierbei spielten jeweils fünf Spieler in insgesamt fünf Spielrunden unter ähnlichen Bedingungen, wie in dem Wettbewerb. Der größte Unterschied zu dem Wettbewerb bestand dabei darin, dass die Anzahl der Bots in einem Spiel nicht bekannt war. Die teilnehmenden Spieler hatten außerdem selbst keine Erfahrung mit der Entwicklung von Agenten für Computerspiele. Auch bei diesen Testdurchläufen wurden die Spiele aufgenommen und weiteren Personen zur Bewertung gezeigt. Im Gegensatz zu dem Wettbewerb konnten die Bots hierbei jedoch nicht von ihrer „Menschlichkeit“ überzeugen und wurden leicht erkannt. Dies lag unter anderem an der Perspektive, aus der die Videos aufgenommen wurden, aber auch daran, dass der OvGUBot ein sehr zurückhaltendes Verhalten aufwies, was ihn von anderen Spielern stark unterschied.

Sowohl bei dem BotPrize-Wettbewerb, als auch bei den lokalen Tests ist aufgefallen, dass die Bewegungsweise eines Spielcharakters sehr viel über seine „Menschlichkeit“ aussagt. So ist es besonders wichtig, dass sich ein Agent flüssig fortbewegen kann und Bewegungsmuster nicht wiederholt. Außerdem spielt die Fokussierung auf einen Gegner, zu der auch die Beobachtung und die Verfolgung gehören, eine sehr wichtige Rolle.

6.2 Kritische Reflexion

Die Art und Weise, auf die der OvGUBot in dieser Arbeit entwickelt wurde, zeigte sowohl Vor- als auch Nachteile. Durch die regelbasierte Architektur, war es einfach, bestimmte Verhaltensweisen zu implementieren. Es war möglich, die Bewegungsweisen des Agenten zwischen dem gezielten Folgen eines Pfades und dem Pendeln in der Nähe eines Ortes, umzuschalten. Jedoch stellte sich heraus, dass nicht immer Verlass auf die Pfadberechnung sein konnte, sodass Ablenkungen implementiert werden mussten, um das Steckenbleiben zu verbergen.

Die Idee, den Agenten in dem „Observe“-Zustand andere Spieler beobachten zu lassen, war allgemein sehr gut. An manchen Stellen hätte der Bot jedoch aggressiver sein können. So wurde doch manchmal kritisiert, dass der Agent einen Spieler erst attackierte, wenn dieser ihn zuerst angegriffen hat. Es wäre an dieser Stelle eventuell sinnvoller, mehr Informationen über die Umgebung und die Gegner zu nutzen, um sie in bestimmten Fällen eher zu attackieren.

Leider konnte der „Hide“-Zustand kaum betreten werden. Dies lag nicht nur an der inoffiziellen Map, die in dem BotPrize-Wettbewerb benutzt wurde, sondern vor allem daran, dass der Zustand nur dann betreten wurde, wenn kein Gegner von dem Agenten zu sehen war. Da die Maps jedoch relativ klein waren und sehr viele Spieler darauf gespielt haben, konnte der OvGUBot beinahe zu jeden Zeitpunkt mindestens einen Spieler „sehen“ und wählte somit einen anderen Zustand aus.

Allgemein kann festgehalten werden, dass viele Probleme, die sich bei der Entwicklung des Agenten ergaben, durch die angebotenen Bibliotheken von Pogamut entstanden sind. Pogamut wird jedoch immer weiter entwickelt und die Lösungen für die meisten Probleme konnten auf den Support-Seiten von Pogamut gefunden werden. So wird es für die nächsten Wettbewerbe möglicherweise bessere Bedingungen geben. Nach den gelösten Problemen können mittlerweile nur noch wenige Einzelheiten an dem Verhalten des Agenten oder der zugrundeliegenden Architektur kritisiert werden. Die Vorschläge für Verbesserungen werden in dem Kapitel 6.4 erläutert.

6.3 Fazit

Das Ziel dieser Arbeit bestand darin, einen Agenten zu entwickeln, der in einem Computerspiel von seiner „Menschlichkeit“ überzeugen könnte. Hierfür sollten die Ergebnisse des BotPrize-Wettbewerbs betrachtet werden. Es war jedoch weniger wichtig den Wettbewerb zu gewinnen, als vielmehr „gute“ Ergebnisse im Vergleich zu menschlichen Spielern zu erzielen.

Obwohl der OvGUBot den vierten Platz bei dem Wettbewerb erzielt hat, waren die Unterschiede in der Gesamtwertung sehr klein. In der zweiten Bewertungsphase konnte er sogar mehr als 50% der Beobachter von seiner „Menschlichkeit“ überzeugen. Was die Ergebnisse betrifft, kann somit eine positive Bilanz gezogen werden.

Anschließend ein paar Worte zu meinen persönlichen Eindrücken. Durch die Entwicklung des OvGUBots habe ich einen guten Eindruck davon bekommen, wie schwierig es sein kann, „glaubhafte“ Agenten für Computerspiele zu entwickeln. „Menschlichkeit“ ist nicht nur schwer zu messen, sondern kann auch von unterschiedlichen Personen anders wahrgenommen werden. Um das Verhalten des Agenten auf unterschiedliche Spieler anpassen zu können, müssen sehr komplexe Zusammenhänge erkannt werden. Die Arbeit an dem Agenten hat mir jedoch bereits viele Erkenntnisse auf diesem Gebiet geboten und war eine sehr interessante Erfahrung.

6.4 Ausblick

Auch wenn die Ergebnisse das OvGUBots bei dem BotPrize-Wettbewerb zufriedenstellend waren, wurde während der Spiele erkannt, welche Punkte an der Architektur zusätzlich verbessert werden könnten. Da die Bewegung eines Agenten eine sehr große Rolle bei der Bewertung seiner „Menschlichkeit“ spielt, müsste diese zu aller erst optimiert werden. Hierfür wird vorgeschlagen, nicht mehr den A*-Algortihmus von GameBots zu benutzen. Dies hat zwei unterschiedliche Gründe. Zum einen bewegt sich der Agent dabei zu nahe an den Navigationspunkten (trotz der implementierten Abweichungen), was erfahrenen Spielern auffallen kann. Zum anderen funktioniert die Pfadberechnung bei der Verbindung zu einem entfernten Server nicht immer und erzeugt starke Verzögerungen in dem Spielverlauf. Als Alternative könnte zum Beispiel die Fortbewegung mit Hilfe von Strahlen umgesetzt werden oder ein eigener A*-Algorithmus geschrieben werden.

Weiterhin sollte der „Hide“-Zustand überarbeitet werden. Zum einen sollte es möglich sein, Versteckpunkte auch ohne eine Sichtbarkeitsmatrix zu finden. Zum anderen sollte der Zustand auch dann betreten werden können, wenn andere Spieler in der Sichtweite des Agenten sind, aber sich zum Beispiel außerhalb seiner Reichweite befinden.

Außerdem ist es sehr wichtig, die Fokussierung auf unterschiedliche Gegner zu verbessern. Hierzu gehört zum Beispiel die Verfolgung eines Gegners, die eventuell früher stattfinden sollte. Der Agent sollte sich also nicht erst dann auf den Spieler zubewegen, wenn dieser außerhalb der Sichtweite ist, sondern versuchen, immer einen bestimmten Abstand zu dem Gegner zu halten. Auch bei dem Weglaufen in dem „Retreat“-Zustand sollten nicht nur die Position eines Gegners, sondern aller sich in der Nähe befindenden Gegner berücksichtigt werden. So kann sichergestellt werden, dass der Agent beim Rückzug vor einem Spieler nicht in die Hände eines anderen läuft.

Allgemein kann das Verhalten des Agenten etwas „aggressiver“ gestaltet werden. Hierfür wird empfohlen, Informationen über andere Spieler zu nutzen, um herauszufinden, wie aggressiv diese sich gegenüber Gegnern verhalten und das Verhalten des Agenten dementsprechend anzupassen.

Literaturverzeichnis

2K BotPrize, The: Humanness results http://www.botprize.org/result.html (2014-10-18)

Arrabales, R: BotPrize 2014 Results. Human-Like Bots Competition at IEEE CIG, http://de.slideshare.net/array2001/arrabales-bot-prize2014v2 (2014-10-18)

Arrabales, R. 2: CERA-CRANIUM http://www.conscious-robots.com/raul/machine-consciousness/34-machine-consciousness-research/67-cera-cranium.html (2014-10-18)

Arrabales, R. / Ledezma, A. / Sanchis, A.: ConsScale FPS: Cognitive Integration for Improved Believability in Computer Game Bots. In: Hingston, P. (Hrg.): Believable Bots. Can Computers Play Like People? Springer, Berlin/Heidelberg, 2012

Arrabales, R. / Muñoz, J.: The Awakening of Conscious Bots: Inside the Mind of the 2K BotPrize 2010 Winner http://aigamedev.com/open/article/conscious-bot (2014-10-18)

Arrabales, R. / Muñoz, J. / Ledezma, A. / Gutierrez, G. / Sanchis, A.: A Machine Consciousness Approach to the Design of Human-Like Bots. In: Hingston, P. (Hrg.): Believable Bots. Can Computers Play Like People? Springer, Berlin/Heidelberg, 2012

Bida, M.: Gamebots API list https://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php?page=Gamebots%20API%20list (2014-10-18)

Bida, M. / Cerny, M. / Gemrot, J. / Brom, C.: Evolution of GameBots project. In: Herrlich, M./ Malaka, R./ Masuch, M. (Hrg.): Entertainment Computing – ICEC 2012, Springer, Berlin/Heidelberg, 2012

Cothran, J.: Winning the 2K Bot Prize with a Long-Term Memory Database using SQLite http://aigamedev.com/open/articles/sqlite-bot (2014-10-18)

Fountas, Z.: Spiking Neural Networks for Human-like Avatar Control in a Simulated Environment, London, 2011

Fountas, Z.: Presentation. Spiking Neural Networks for Human-like Avatar Control in a Simulated Environment http://www.doc.ic.ac.uk/~zf509/Downloads/NeuroBot/MScThesis_presentation_slides.pdf (2014-10-18)

Fountas, Z. / Gamez, D. / Fidjeland, A.K.: A Neuronal Global Workspace for Human-like Control of a Computer Game Character. In: IEEE Conference on Computational Intelligence and Games, Seoul, 2011

Glavin, F.G. / Madden, M.G.: Incorporating Reinforcement Learning into the Creation of Human-Like Autonomous Agents in First Person Shooter Games. In: Proceedings of GAMEON 2011, the 12th annual European Conference on Simulation and AI in Computer Games, Galway, 2011

Hingston, P.: A New Design for a Turing Test for Bots. In: IEEE Symposium on Computational Intelligence and Games, IEEE, 2010

HINGSTON, P.: A TURING TEST FOR COMPUTER GAME BOTS. IN: IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 1, NO. 3, IEEE, 2009

HINGSTON, P.: BELIEVABLE BOTS. CAN COMPUTERS PLAY LIKE PEOPLE? SPRINGER, BERLIN/HEIDELBERG, 2012

HUMAN-LIKE BOTS: FREQUENTLY ASKED QUESTIONS, HTTP://HUMAN-MACHINE.UNIZAR.ES/?Q=RETECOG/CONTACT-0 (2014-10-18)

HUMAN-LIKE BOTS 2: HUMANNESS ASSESMENT METHOD, HTTP://HUMAN-MACHINE.UNIZAR.ES/?Q=RETECOG/HUMANNESS-ASSESMENT-METHOD (2014-10-18)

HUMAN-LIKE BOTS 3: TEAMS, HTTP://HUMAN-MACHINE.UNIZAR.ES/?Q=RETECOG/TEAMS (2014-10-18)

HUMANLIKE BOTCOM: HUMAN LIKE BOT COMPETITION (2014) HTTPS://WWW.YOUTUBE.COM/PLAYLIST?LIST=PLQVDNQWZ4VYEQMAIXYD0Z97ZC5UH1VWFM (2014-10-18)

KADLEC, R.: EVOLUTION OF INTELLIGENT AGENT BEHAVIOUR IN COMPUTER GAMES, PRAGUE, 2008

KADLEC, R. / GEMROT, J. / BIDA, M. / HAVLICEK, J. / PIBIL, R. / ZEMCAK, L. / VANSA, R.: POGAMUT 3 COOKBOOK HTTP://POGAMUT.CUNI.CZ/POGAMUT_FILES/LATEST/DOC/TUTORIALS/TUTORIALSBOOK.HTML (2014-10-18)

KARPOV, I.: BOTPRIZE 2010 - ROUND 3 - GAME 12 - DM-ICEHENGE - TRISTAN (HUMAN) - 1/4 HTTPS://WWW.YOUTUBE.COM/WATCH?V=LSAQNDP6DSW&LIST=UU3SAZPBXCQCCRZHPTH5BLLG (2014-10-18)

KARPOV, V. / SCHRUM, J. / MIIKKULAINEN, R.: BELIEVABLE BOT NAVIGATIONVIA PLAYBACK OF HUMAN TRACES. IN: HINGSTON, P. (HRG.): BELIEVABLE BOTS. CAN COMPUTERS PLAY LIKE PEOPLE? SPRINGER, BERLIN/HEIDELBERG, 2012

LOEBNER, H.G.: WHAT IS THE LOEBNER PRIZE? HTTP://WWW.LOEBNER.NET/PRIZEF/LOEBNER-PRIZE.HTML (2014-10-18)

MORA, A.M. / AISA, F. / CABALLERO, R. / GARCIA-SANCHEZ, P. / MERELO, J.J. / CASTILLO, P.A. / LARA-CABRERA, R: DESIGNING AND EVOLVING AN UNREALTOURNAMENT 2004 EXPERT BOT, GRANADA, 2013

MORA, A.: MODELLING HUMAN EXPERT BEHAVIOUR IN AN UNREAL TOURNAMENT 2004 BOT HTTP://DE.SLIDESHARE.NET/SLIDEMORA/MODELLING-26303325 (2014-10-18)

MUÑOZ FERNÁNDEZ, E.: IMPLEMENTATION OF THE ARTIFICIAL RECOGNITION SYSTEM DECISION UNIT IN A COMPLEX SIMULATION ENVIRONMENT, WIEN, 2012

PATRICK, M.: ONLINE EVOLUTION IN UNREAL TOURNAMENT 2004. IN: IEEE SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE AND GAMES, IEEE, 2010

POGAMUT: GAMEBOTS HTTP://POGAMUT.CUNI.CZ/MAIN/TIKI-INDEX.PHP?PAGE=GAMEBOTS (2014-10-18)

POLCEANU, M.: MIRRORBOT: USING HUMAN-INSPIRED MIRRORING BEHAVIOR TO PASS A TURING TEST. IN: IEEE CONFERENCE ON COMPUTATIONAL INTELLIGENCE AND GAMES, NIAGARA FALLS, 2013

SCHRUM, J. / KARPOV, V. / MIIKKULAINEN, R.: HUMAN-LIKE COMBAT BEHAVIOUR VIA MULTIOBJECTIVE NEUROEVOLUTION. IN: HINGSTON, P. (HRG.): BELIEVABLE BOTS. CAN COMPUTERS PLAY LIKE PEOPLE? SPRINGER, BERLIN/HEIDELBERG, 2012

SCHRUM, J. / KARPOV, I.: UT^2: WINNER OF 2012 BOTPRIZE IN UNREAL TOURNAMENT 2004 (2012) HTTP://NN.CS.UTEXAS.EDU/?BOTPRIZE2012 (2014-10-18)

STOLBA, M.: BOTPRIZE 2008 WINNING BOT HTTP://AMIS.MFF.CUNI.CZ/POGAMUT/TIKI-INDEX.PHP?PAGE=BOTPRIZE+2008+WINNING+BOT#AMIS_BOT (2014-10-18)

TURING, A.M.: COMPUTING MACHINERY AND INTELLIGENCE. IN: MIND, VOL. 59, 1950

Anhang

Anhang A: Kommentare der Teilnehmer der Dritte-Person-Bewertungsphase des BotPrize-Wettbewerbs.

Player	PlayerType	PlayerName	Guess	GuessReason
6	Human	Xenija	Human	he have good balance movement
6	Human	Xenija	Human	Moving like this character is not too difficult for gamers.
6	Human	Xenija	Human	The behaviour of the character is more close to Human controlled.He is moving but not continiously and he is attacking when he has a chance.
6	Human	Xenija	Human	The characater seemed to know that it had a purpose and what its objective was in order to be successful and it carried out the needed actions any other Human playing this game would think of to do.
6	Human	Xenija	Human	Speed & movement
6	Human	Xenija	Human	GOOD MOVEMENTS LOOKS REAL .....CAMERA ANGLE IS GOOD
6	Human	Xenija	Human	The virtual character maneuvered well and did not seem stiff as if programmed or in other words controlled by a machine.
6	Human	Xenija	Human	The virtual player was being controlled by a Human because the player seemed to be actively exploring the map searching out other opponent players. An AI player seems to react more than anything. Also, there were times when this player changed their weapon. It seemed as if the player was changing the weapon to best suit the situation. This player actively changed the weapon a few times making me think an AI wouldn't do something like this
6	Human	Xenija	Human	Realistic shooting behavior my primary aspect. Movements were normal, run from your spot if you can’t shoot right. The player actually used the yellow character between 30-32 seconds as a shield, which is a typical of a smart Human. Was busy with front enemy since he had to use the yellow guy as shield before he dies, he kept on even when there was an enemy behind who wasn’t even shooting. After the front enemy was down, went back for the bad guy behind. Pretty much Human with a very good eye though. A machine can’t make such clever judgments
6	Human	Xenija	Human	Human is being controll the character because it absorb so many enemy bullets
6	Human	Xenija	Human	Machine

6	Human	Xenija	Human	movement is good
6	Human	Xenija	Human	palyer is perfect,Human but shows machine abilities,accuracy,speed,timing
6	Human	Xenija	Human	Better reflexes
6	Human	Xenija	Machine	machine control a fast movement of character
6	Human	Xenija	Machine	wasnt very sure which one it could have been. however i ended up decided machine because they were very good at following the individuals and it just seemed like that kind of accuracy would be a machine.
6	Human	Xenija	Machine	The virtual character seem to be acting too fast that it focuses the enemies without any hesitation.
10	Bot	OvGUBot	Human	They are runing from laser shooting
10	Bot	OvGUBot	Human	machine
10	Bot	OvGUBot	Human	movement and quick responses
10	Bot	OvGUBot	Human	character action is slow who controls is not a good gamer if it controls by a computer it will be more fast
10	Bot	OvGUBot	Human	Human is being controll the character becuase character took more time for fighting
10	Bot	OvGUBot	Human	LITTLE LAG SHOULD BE FACED
10	Bot	OvGUBot	Human	i think a Human because they were running around a lot. they also were killed quite a bit which suggests Human.
10	Bot	OvGUBot	Human	Quite impressed with Character 10. My primary aspect of perception of behavior is based on the judgment of the player. In the first 19 seconds, the player wanted to collect an orb or something shining. He didn't just go for it, like some robot making calculations within milliseconds and still making a perfect short. The player made a perfect Human behavior of making a judgment on the enemy, how quick they short back at unknown entry, and the direction of the shooting and then made the run. The same thing happened on the 90th second. Such judgments were made by a Human player who knew was playing with a computer and just couldn't think like him. Only a Human can think like that in gaming.
10	Bot	OvGUBot	Human	.player moving very slowly,low accuracy,computer does not do that
10	Bot	OvGUBot	Human	This virtual player was controlled by a Human because there were a few moments when this player would shoot at the walls. I can see how even an AI player can shoot and hit the wall by missing a player but when you have aimed at the wall and fire, that seems more like a Human thing to

				do. Also, there was a moment when the player would jump on the moving platform then jump off, jump on and then jump off again. With whatever is controlling this character there seems to be some thought going behind this and some form of strategy which is another reason why I think it's a Human.
10	Bot	OvGUBot	Human	It is definetely an Human bot as it made certain mistakes while playing such as misinterpreting enemy's location and shooting on the wall and getting shot by the enemy at the same instant.
10	Bot	OvGUBot	Human	. The wy the character moves, and the reflexes seems that cames from Human
10	Bot	OvGUBot	Human	His behaviour was Humanlike, insecure.
10	Bot	OvGUBot	Human	An experienced player can play like that and it is a online game
10	Bot	OvGUBot	Machine	Non-stop repeated actions
10	Bot	OvGUBot	Machine	The movements were clitchy and at times very hard to see what was happening.
10	Bot	OvGUBot	Machine	The characters playing had a lot of delay reactions like bots, some have trouble following a path to a platform and some bots completely ignore other bots while walking by them.
10	Bot	OvGUBot	Machine	Because,this video seems that it controlled by an artificial person who is an experianced in this field.he controll every movement of this game.it gave me it must be a machine.
10	Bot	OvGUBot	Machine	the character sometimes stop to act along the level of the game for no reason and he gets shots without escape
10	Bot	OvGUBot	Machine	week response to attack
10	Bot	OvGUBot	Machine	because of the fast moments my decision is Machine
10	Bot	OvGUBot	Machine	Movement seemed fluid like controlled by machine and some movement seemed unnecessary.
10	Bot	OvGUBot	Machine	Random and unnecessary shooting/jumping/movement. Character seems not to interact with other players unless they shoot him first.

Anhang B: Bewertungsbogen der Menschlichkeit der Spieler in den lokalen Testspielen.

In der ersten Tabelle sind Videos von je 5 Spielern aus 5 Spielrunden verlinkt. Sieh Dir bitte zuerst die untere Tabelle mit den unterschiedlichen Verhaltensmechanismen an.
Schau danach die Videos an und versuche jeweils den Spieler, aus dessen Perspektive das Video aufgenommen wurde, nach seiner Menschlichkeit zu bewerten (0 – 100% „menschlich“) und trage die Bewertung in die erste Tabelle ein.
(Bemerkung: auch wenn Du aufgrund von bestimmten Aktionen dir sicher bist, dass der Spieler ein Mensch/Bot ist, gebe nicht unbedingt 100/0%, sondern überlege, zu wie viel % der Spieler sich sonst „menschlich“ verhalten hat.)
In jeder Spielrunde gibt es unterschiedlich viele Bots.
Lasse Dich nicht von den schwarzen Rechtecken in den Videos verwirren. Sie dienen dazu, gleiche Spieler in unterschiedlichen Runden nicht wiedererkennbar zu machen.
Wenn Du alle Spieler in allen Runden bewertet hast, trage in die zweite Tabelle ein, wie wichtig die beschriebenen Mechanismen bei Deinen Entscheidungen im Allgemeinen waren (0 – 10).
Schicke anschließend den ausgefüllten Bogen an xenija@acagamics.de Vielen Dank für Deine Teilnahme!

Spieler	Runde 1 Spieler 1	Runde 1 Spieler 2	Runde 1 Spieler 3	Runde 1 Spieler 4	Runde 1 Spieler 5	Runde 2 Spieler 1	Runde 2 Spieler 2	Runde 2 Spieler 3	Runde 2 Spieler 4	Runde 2 Spieler 5
Menschlichkeit *										
Spieler	Runde 3 Spieler 1	Runde 3 Spieler 2	Runde 3 Spieler 3	Runde 3 Spieler 4	Runde 3 Spieler 5	Runde 4 Spieler 1	Runde 4 Spieler 2	Runde 4 Spieler 3	Runde 4 Spieler 4	Runde 4 Spieler 5
Menschlichkeit *										
Spieler	Runde 5 Spieler 1	Runde 5 Spieler 2	Runde 5 Spieler 3	Runde 5 Spieler 4	Runde 5 Spieler 5					
Menschlichkeit *										

*Menschlichkeit des beobachteten Spielers in % eintragen
0% keine menschlichen Züge; ab 50% wird er als ein Mensch anerkannt (Turing Test bestanden); 100% absolut menschlich

Wichtigkeit 0 – 10**	**Mechanismus**
	Blieb im Spiel stehen (z.B. um sich umzuschauen. Damit ist nicht das Steckenbleiben gemeint)
	Benutzte die „Dodge“-Fähigkeit (sehr schnelle Bewegungen)
	Ausweichen/Weglaufen, wenn im Nachteil
	Verstecken um den Gegner auszutricksen
	Fokussierte sich auf den Gegner
	Imitation des Gegnerverhaltens
	Langzeitgedächtnis
	Limitierte Schussgenauigkeit
	Ressourcensammeln basierend auf den Bedürfnissen
	Situationsbeobachtung
	Reibungslose Bewegung
	Zielverfolgung
	Waffenauswahl kontextabhängig
	Hohe Aggressivität
	Erzielte viele Punkte im Spiel
	Andere (bitte beschreiben)

** Wie wichtig ist im Allgemeinen für dich der angegebene Mechanismus um die Entscheidung nach der Menschlichkeit zu treffen:

(unwichtig) 0 – 10 (sehr wichtig)

Anhang C: ergänzende Mechanismen und Kommentare zu den Videos der lokalen Testspiele.

Richter	
A	Rückwärtslaufen, Umkehren in Bewegung und Abstürze bei Sprüngen
A	Umschauen um die Map zu erkunden
A	Kamerabewegung
A	Reaktion auf Beschuss
A	Schießt nicht auf Gegner vor ihm (Spieler 2 Runde 2 – 0:20)
A	Folgt den grauen Steinen
A	Nutzung des Schilds zum Töten
A	z.B. auf der letzten Map schaut Einer nach unten von den Pfaden aus. Das ist etwas, was ein Bot nicht machen würde
E	Visuelle Trigger, z.B. Routenänderung nachdem ein Powerup zufällig ins Sichtfeld erscheint
F	steife Kamera-Führung
F	Verwendet Zoom der Waffe zum Schießen
F	Verwendet Zoom der Waffe zum Spähen
F	Strafing
F	Greift Absolut still stehende Charaktere nicht an

Anhang D: CD-ROM

Die beigefügte CD enthält den Quellcode des OvGUBots sowie eine PDF-Version dieser Arbeit. Außerdem ist der Quellcode des Expert Bots zu dem Anfang des Entwicklungszeitraumes des OvGUBots beigefügt (Stand 03.2014).

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, 22.10.2014…… …….