Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Masterarbeit

Multimodaler Zugang zu einer virtuellen Welt für Sehende und Sehgeschädigte

Linda Kerkhoff - 534216

Hochschule für Technik und Wirtschaft Berlin

Internationale Medieninformatik

Prof. Dr.-Ing. David Strippgen	1. Prüfer, HTW
Prof. Dr.-Ing. Carsten Busch	2. Prüfer, HTW
Dipl.-Ing.(FH) Florian Berger	Sachverständiger Betreuer

Vorgelegt am 10. Oktober 2013

Abstract

Computer games have reached a high status in the modern digital society and culture. The main reception of a game happens via the visual channel. Thus, people with no or limited vision are automatically excluded from these cultural assets.

Whether research in this fiels is focused on games for blind people or games of multi-modal access, it is mostly concentrated on the development of one or few different games. There seem to be no efforts of developing the means or tools to simplify the creation of such games. What is missing is the approach to combine developed concepts in a tool to enable the creation of a multitude of games.

In the present work, it is hypothesized that it is possible to create or to extend an existing game engine to enable the creation of multi-modal accessible games. Namely to allow the development of games whose output can be received by using only the visual or the auditory senses.

The existing open source engine Fabula already provides a visual interface. In the present work it is extended with an auditory-only user interface. To evaluate the extended game engine an adventure game for two players is designed suitable for visual and auditory perception. The design of the game is followed by its implementation using the extended game engine.

The resulting game was evaluated in a small case study. It has been proven that the extension of the game engine allows the creation of multi-modal accessible games for sighted and non sighted users. The Access is guaranteed by providing an auditory-only as well as a visual-only client.

Kurzbeschreibung

In unserer Gesellschaft haben Computerspiele einen großen Stellenwert und sind zu einem festen Bestandteil der Kultur geworden. Die hauptsächliche Rezeption eines Spiels erfolgt dabei über den visuellen Kanal. Menschen ohne oder mit eingeschränktem Sehvermögen werden dadurch automatisch von diesem Kulturgut ausgeschlossen.

Unabhängig davon, ob ein Forschungsprojekt sich mit der Erstellung von Spielen für blinde Menschen oder der Entwicklung von multimodal zugänglichen Spielen widmet, werden meist ein einzelnes oder wenige Spiele unterschiedlicher Genre entwickelt. Was fehlt ist der Ansatz, entwickelte Konzepte in einem Werkzeug zu vereinen, sodass die Möglichkeit gegeben ist, mit diesem Werkzeug eine Vielzahl von Spielen erstellen zu können.

In dieser Arbeit wird die Hypothese aufgestellt, dass es möglich ist, eine vorhandene Game-Engine so zu erweitern, dass mit dieser ein rein auditiv und rein visuell zugängliches Spiel erstellt werden kann.

Zur Belegung dieser These erfolgt die Konzeption eines visuell und auditiv zugänglichen Spiels, sowie die Erweiterung einer quelloffenen Game-Engine, so dass diese eine auditive und eine visuelle Benutzerschnittstelle für den alleinigen Zugang zum Spiel gewährt. Außerdem erfolgt die Implementierung des Spielkonzepts mit der Engine als Grundlage für die Evaluierung in einer klein angelegten Fallstudie.

Die bestehende Game-Engine Fabula konnte im Rahmen dieser Arbeit um eine auditive Benutzerschnittstelle erweitert werden. Die Konzeption und die Implementierung eines Adventure-Spiels für zwei Spieler mit Fabula ist gelungen. Dieses Spiel kann sowohl unter der Verwendung der visuellen, als auch der auditiven Schnittstelle gespielt werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

In unserer Gesellschaft haben Computerspiele einen großen Stellenwert und sind zu einem festen Bestandteil der Kultur geworden. Insbesondere Kindern und Jugendlichen eröffnen sie den Zugang zur medialen Welt und fördern die wichtige Auseinandersetzung mit Computern und Technologien. Die hauptsächliche Rezeption erfolgt dabei über den visuellen Kanal, während der auditive Kanal in der Regel nur dazu dient, die realistische Darstellung zu verbessern [14]. Menschen ohne oder mit eingeschränktem Sehvermögen werden dadurch automatisch von diesem Kulturgut ausgeschlossen.

Es gibt in der Forschung verschiedene Ansätze auf die besonderen Bedürfnisse einzugehen, um diese Menschen zu integrieren. So wurde beispielsweise eine Vielzahl von existierenden Computerspielen unterschiedlicher Genre adaptiert und so verändert, dass sie von visuell beeinträchtigten Personen gespielt werden können. Zum Teil geht das nur unter Zuhilfenahme von spezieller Hardware, die haptische Rückmeldung gibt, wie zum Beispiel Force Feedback bei einem Joystick oder speziell entwickelte Handschuhe die visuelle Informationen in haptische Rückmeldung verwandeln [27].

Der Fokus der meisten Entwicklungen liegt hierbei darauf, ein Spiel für Menschen mit Sehschädigung zu kreieren. Es gibt nur wenige integrative Ansätze, die sich darauf konzentrieren einen multimodalen Zugang zu schaffen [14]; also beispielsweise neben einem visuellen Kommunikationskanal, wie etwa die zwei- oder dreidimensionale Darstellung auch einen auditiven Kanal als vollwertigen Zugang bereit zu stellen.

Dieser Ansatz wird mit universal zugänglichen Spielen weiter vertieft. Diese Spiele sind dafür ausgelegt, dass Menschen mit unterschiedlichen Beeinträchtigungen sie spielen können. Dabei werden vier Gruppen möglicher Beeinträchtigungen nach der Einteilung der WHO *(World Health Organisation)* berücksichtigt: Menschen mit eingeschränktem Sehvermögen, eingeschränktem Hörvermögen, mit motorischen oder kognitiven Beeinträchtigungen (vgl. [28]). Dabei kann ein Spieler auch Einschränkungen aus mehr als einer Gruppe haben. Vor Beginn des Spiels werden die Bedürfnisse des Spielers in einem Profil erfasst und die Schnittstellen des Spiels angepasst, so kann zum Beispiel ein motorisch eingeschränkter Mensch ein solches Spiel nur über das Betätigen einer einzelnen Taste bedienen. Die Auswahl an geeigneten Spielkonzepten ist für diesen Ansatz naturgemäß eingeschränkt und muss für alle Beeinträchtigungen anpassbar sein.

Unabhängig davon, ob ein Forschungsprojekt sich mit der Erstellung von Spielen für blinde Menschen, der Entwicklung von multimodal oder universal zugänglichen Spielen widmet, lässt sich eine Gemeinsamkeit feststellen: Alle Projekte entwickeln ein oder wenige Spiele oder Prototypen unterschiedlicher Genre, um Konzepte und Technologien für die Umsetzung zu evaluieren. Wenn mehrere Spiele umgesetzt werden, stützen sich diese auf gemeinsame Konzepte, jedoch nicht auf eine gemeinsame Umsetzung. Was fehlt ist der Ansatz, entwickelte Konzepte in einem Werkzeug zu vereinen, so dass die Möglichkeit geboten ist, mit diesem Werkzeug mehrere Spiele, im Idealfall auch aus unterschiedlichen Genre, erstellen zu können.

Ein solches Werkzeug könnte eine Game-Engine sein, mit der die Erstellung von multimodal oder universal zugänglichen Spielen möglich ist. Bei universal zugänglichen Spielen ist zwar die Anzahl an geeigneten Spielkonzepten begrenzt, allerdings können sich durch die Kombination von den vier möglichen Beeinträchtigungen des Spielers entsprechend viele Umsetzungsvarianten mit speziellen Schnittstellenanforderungen ergeben. Für jede dieser Schnittstellen ist eine tiefgehende Auseinandersetzung mit den Beeinträchtigungen und den sich daraus ergebenden Bedürfnissen der Spieler nötig um die Schnittstellen adäquat zu definieren und umzusetzen. Der dazu benötige Aufwand ist im Rahmen einer einzigen Arbeit nicht zu leisten. Ein naheliegender Schritt wäre es die Herausforderung der Entwicklung einer Game-Engine für multimodal oder universal zugängliche Spiele auf ein abgeschlossenes Teilproblem herunterzubrechen. Zum Beispiel in dem eine Game-Engine für multimodal zugängliche Spiele entwickelt wird, welche mindestens zwei Wahrnehmungsschnittstellen bedient. Als Schnittstellen würden sich der visuelle und der auditive Wahrnehmungskanal anbieten, weil hierfür ausreichend evaluierte Beispiele zur Verfügung stehen.

Mit der Umsetzung einer solchen Game-Engine könnte ein Weg aufgezeigt werden, wie der Zugang zu einem Spiel über zwei Wahrnehmungsschnittstellen gleichwertig möglich gemacht werden kann. Die Game-Engine könnte zum einen die Grundlage für zukünftige Forschungsarbeiten bieten, welche einen ähnlichen Ansatz verfolgen oder den Ansatz dieser Arbeit erweitern wollen. Zum anderen wäre die Game-Engine für Entwicklungen von multimodal zugänglichen Spielen geeignet und könnte deren Entwicklungsaufwand reduzieren. Dabei wäre ein positiver Nebeneffekt, dass Spiele erstellt werden können, welche von sehenden und visuell eingeschränkten Menschen gleichermaßen und sogar miteinander gespielt werden können. Auf diesem Weg würde eine echte Integration erfolgen und ein Austausch zwischen Menschen ermöglicht, der sonst

so sonst nicht statt finden könnte.

1.2 Forschungsfragen

Es gibt derzeit keine Game-Engine für die Entwicklung von mulitmodal zugänglichen Spielen. Wie kann eine Game-Engine gestaltet oder eine bestehende Game-Engine erweitert werden, sodass die Erstellung von multimodal zugänglichen Spielen möglich ist? Welches Spielkonzept eignet sich für eine Umsetzung unter Verwendung der Game-Engine, das sowohl visuell, als auch auditiv zugänglich ist? Wie kann aus diesem Spielkonzept mit der Game-Engine ein Spiel entwickelt werden? Wie kann mit Hilfe dieses Spiels die Game-Engine bezüglich ihrer Eignung für die Erstellung von multimodal zugänglichen Spielen qualitativ evaluiert werden?

1.3 Hypothese

Es ist möglich eine multimodale Game-Engine zu erstellen, oder eine vorhandene Game-Engine so zu erweitern, so dass mit dieser ein auditiv und visuell zugängliches Spiel erstellt werden kann, welches über die Verwendung beider Schnittstellen getrennt voneinander spielbar ist. Der Begriff der Spielbarkeit umfasst hierbei, dass jede Spielmechanik des Spiels grundsätzlich zugänglich ist.

1.4 Methodik

Das Vorgehen zur Belegung zur Prüfung der Hypothese gliedert sich in die folgenden Schritte. Zunächst wird der aktuelle Forschungsstand von Spielen für visuell beeinträchtigte Menschen und von multimodal zugänglichen Spielen erarbeitet. Darauf aufbauend erfolgt der Entwurf eines sowohl visuell als auch auditiv zugänglichen Spiels. Im Anschluss daran folgt die Planung und Implementierung einer multimodalen Game-Engine oder die Umsetzung einer Erweiterung einer bestehenden Game-Engine zu einer multimodalen Game-Engine. Mit dieser Engine wird dann das zuvor konzipierte Spiel konkret für die Verwendung der visuellen als auch der auditiven Schnittstelle umgesetzt. Diese Umsetzung dient dann im letzten Schritt als Grundlage für die qualitative Evaluierung der Hypothese in einer Fallstudie.

2 State of the Art

2.1 Spiele für Sehgeschädigte

2.1.1 Überblick

Eine umfassende Sammlung an Spielen für sehgeschädigte oder blinde Menschen stellt die Seite audiogames.net [11] bereit. Neben der Auflistung der Spiele gibt es auch ein Forum, des Weiteren können Reviews für Spiele eingesehen oder verfasst werden. In der Sammlung sind Spiele unterschiedlichster Genre zu finden, die unterschiedliche Arten des Zugangs ermöglichen. Die Vielzahl der Genre von Computerspielen wird in [28] in acht Genre unterteilt: *First-person Shooter* (FPS), *Strategy games* – in Echtzeit oder Rundenbasiert, *Sport games*, *Role- playing games* (RPG), *Puzzle Spiele*, *Racing games* – dazu gehören auch Simulatoren, *Dance/Rhythm games* und *Adventure games*. Für jedes dieser Genre sind Vertreter auf audiogames.net zu finden.

2.1.2 Role-playing games

Unter den Spielen finden sich auch textbasierte Browser-Spiele, welche keine spezielle Audioausgabe verwenden, wie zum Beispiel Hogwarts Live [25], ein MMORPG (*Massively Multiplayer Online Role-Playing Game*), welches thematisch in der Welt der Harry Potter Romane von J. K. Rowling angesiedelt ist. Der Spieler kann beispielsweise verschiedene Orte aufsuchen, seinen Avatar ausrüsten und mit ihm Kämpfe bestreiten und so Erfahrung sammeln, die für das Erreichen des nächsten Rangs erforderlich ist. Ein normaler Screenreader, der den Zugang zu Webseiten durch das Vorlesen der Inhalte ermöglicht, reicht aus um das rein textbasierte Spiel zu spielen.

2.1.3 Adventure

Aus dem Genre Adventure sind Audio-Abenteuer wie beispielsweise „Der Tag wird zur Nacht“ [7] und „Sarah and the Castle of Witchcraft and Wizardry“ [20] zu finden. Bei beiden Spielen kann der Spieler sich dabei frei durch eine virtuelle Welt bewegen, Hindernisse und Wände werden akustisch dargestellt, ebenso wie die Schritte des Spielers, um die Orientierung im Raum zu ermöglichen. Die Steuerung beider Spiele erfolgt über die Tastatur, ist aber unterschiedlich komplex, angepasst an den Umfang der Spielmechaniken. Im Spiel „Der Tag wird zur Nacht“ muss der Spieler einen Fluchtweg aus einer Stadtvilla finden, um den Lavaströmen eines ausbrechenden Vulkans zu entkommen. Für die Fortbe-

wegung werden nur drei Pfeiltasten benutzt: linke Pfeiltaste für eine Linksdrehung, rechte Pfeiltaste für eine Rechtsdrehung und Pfeiltaste oben um vorwärts zu laufen. Interaktionen mit Objekten wie zum Beispiel das Öffnen einer Tür erfolgen über das Betätigen der Entertaste. Das Spiel „Sarah and the Castle of Witchcraft and Wizardry“ ist ebenfalls an die Harry Potter Romane angelehnt und umfasst mehr Spielmechaniken als „Der Tag wird zur Nacht“. So verfügt der Spieler hier über ein Inventar, kann Zaubersprüche in einer Liste sammeln, ebenso wie Passwörter. Die Bewegung kann wesentlich präziser erfolgen, so ergibt das Betätigen der linken Pfeiltaste eine kleine Drehung nach links, linke Pfeiltaste in Kombination mit Control eine große Drehung nach links und linke Pfeiltaste zusammen mit Shift gedrückt einen seitlichen Schritt nach Links. Eine vollständige Drehung der Spielfigur wird über die Space-Taste erreicht. Bei der Bewegungsrichtung wird zwischen vorwärts und rückwärts unterschieden und die Schrittgeschwindigkeit kann erhöht werden. Außerdem trifft der Spieler auch auf *non-player character* (NPC), die dem Spieler weiterhelfen oder bekämpft werden müssen. Unterschiedliche Charaktere verfügen dabei über unterschiedliche Stimmen und Gegenstände oder Zaubersprüche haben ihre eigene Klänge. Ein Hilfsmenü kann jederzeit aus dem Spiel aufgerufen werden, dabei werden die einzelnen Menüeinträge vorgelesen. Der Spieler hat so beispielsweise die Möglichkeit, sich die unterschiedlichen Geräusche und ihre Zuordnungen anzuhören, unter anderem kann auch die Anleitung zur Bewegung der Spielfigur abgerufen werden.

2.1.4 Strategy games

Ein Vertreter des Genres Strategie ist „Tacticle Battle“ von Ian Reed, das auf seiner Internetseite [21] heruntergeladen werden kann und ein rundenbasiertes Strategiespiel ist. Das Spielfeld besteht aus einer Karte, die in ein Raster aus quadratischen Flächen unterteilt ist, welche – ähnlich wie bei einem Schachbrett – über Indizes identifiziert werden. Mit Hilfe der Pfeiltasten kann der Spieler über das Spielfeld navigieren und Informationen zu dem Inhalt des Feldes in akustischer Form erhalten. Dabei werden der Inhalt und die Indizes des Feldes vorgelesen. Feldinhalt können Informationen über die Beschaffenheit der Landschaft sein, ob sich auf dem Feld beispielsweise ein Berg befindet und das Spielfeld folglich nicht von einer Spielfigur besetzt werden kann. Ein weiterer möglicher Inhalt ist eine Spielfigur; ein Feld kann nur von einer Spielfigur gleichzeitig besetzt werden. Bei der akustischen Repräsentation eines besetzen Feldes wird der Name der Figur vorgelesen, gefolgt von einem charakteristischen Geräusch – zum Beispiel Wolfsgeheul bei einem Werwolf – und endet mit dem

Vorlesen der Indizes. Zu Beginn des Spiels befinden sich an den seitlichen Spielfeldrändern jeweils zwei konkurrierenden Parteien, bestehend aus mehreren Spielfiguren (Einheiten). Der Spieler tritt gegen eine vom Computer gesteuerte Truppe an. Ziel des Spiels ist es, alle gegnerischen Einheiten zu zerstören, bevor die eigene Truppe zerstört wird. Pro Runde kann der Spieler für jede Einheit eine spezifische Anzahl an Aktionspunkten für unterschiedliche Aktionen einsetzen. Beispiele möglicher Aktionen sind das Bewegen von Figuren auf dem Spielfeld, Angriffe und Heilen. Die Figuren unterscheiden sich in ihren Fähigkeiten, so können einige effektiver heilen, aber nicht so gut kämpfen, andere Einheiten sind Krieger, die höhere Schadenresistenz und stärkere Angriffskräfte besitzen. Auch die gegnerische Truppe setzt sich aus unterschiedliche Figuren mit verschiedenen Fähigkeiten zusammen. Ist der Spieler an der Reihe, kann er wie beschrieben über das Spielfeld navigieren und mit der Entertaste eine seiner Spielfiguren auswählen und für diese Figur das Aktionsmenü öffnen, als Rückmeldung wird der Name des Menüs vorgelesen. Das Aktionsmenü besteht aus einer Liste, der jeweils aktive Eintrag wird vorgelesen. Über die Pfeiltasten nach oben und unten können die Listeneinträge durchblättert werden. Über die Entertaste kann eine Aktion bestätigt werden. Verlässt der Spieler das Menü und kehrt zurück zum Spielfeld, wird diese Information ebenfalls vorgelesen. Das Bewegen einer Spielfigur erfolgt ohne Hilfe des Aktionsmenüs. Nachdem der Spieler zu einem Feld mit einer eigenen Spielfigur navigiert ist, kann er diese über die Pfeiltasten in Kombination mit der Shift-Taste direkt auf ein angrenzendes Feld bewegen, sofern dieses nicht blockiert und somit besetzt ist. Eine Besonderheit des Spiels ist, dass Spieler eigene Karten entwerfen und spielen können. In der für diese Arbeit gespielten Version 1.11 von „Tacticle Battle“ stehen initial drei Spielmodi mit unterschiedlichen Karten zur Auswahl.

2.1.5 Racing games

Ein Vertreter des Genres Racing games ist „Topspeed 3“ [8], ein klassisches Rennspiel. Die Steuerung erfolgt initial über drei Pfeiltasten: Links für eine Linkskurve, rechts für eine Rechtskurve und hoch, um zu beschleunigen. Allerdings können die Tastenbelegung im Optionsmenü geändert werden, auch besteht die Wahl einen Joystick anzuschließen, welcher sich schnell und einfach im Menü kalibrieren lässt. Die Orientierung im Spiel erfolgt über das Motorengeräusch und einen Copiloten, der Kurven und ihre Beschaffenheit ansagt, ob es zum Beispiel eine Haarnadelkurve oder eine sanfte Kurve ist. Außerdem kann zwischen den Optionen 3D-Audio und Stereo-Ausgabe gewählt werden.

2.1.6 Verschiedenes

Die Spiele „Audiofrogger“, „Mosquitos“, „MatrixShot“ werden in [23] vorgestellt. Alle drei Spiele wurden mit 3D Sound umgesetzt, können allerdings auch mit Stereo-Kopfhörern gespielt werden. Bei „Audiofrogger“ handelt es sich um ein Remake des Spiels „Frogger“. Wie im Original muss der Spieler am Bildschirm eine mehrspurige Straße überqueren und hören, ob Fahrzeuge von links oder rechts kommen. Die beiden anderen Spiele werden nicht nur am Monitor gespielt sondern erfordern mehr Raum. „Mosquitos“ wird in einer 360° Umgebung mit Hilfe von Headtracking, Kopfhörern und einem sogenannten Stylus Pen[1] als Eingabegerät gespielt. Dieses Gerät fungiert im Spiel als Mückenspray und der Spieler muss Mücken auditiv lokalisieren und mit dem Spray bekämpfen. Im weitesten Sinne gehört dieses Spiel noch zum Genre FPS. „MatrixShot“ verwendet neben Kopfhörern und Headtracking auch eine Kamera. Aus Richtung der Kamera werden auf den Spieler virtuelle, akustisch dargestellte Geschosse abgegeben, denen der Spieler nach links oder rechts ausweichen muss, die Ausweichbewegung wird von der Kamera erfasst und ausgewertet.

2.1.7 Dance/Rhythm games

„Finger Dance“ ist eine Adaption des Spiels „Dance Dance Revolution“ (DDR) und aus dem Genre Dance/Rhythm games. Im Original wird für das Spiel eine sensorische *Tanzmatte* als Eingabegerät benutzt. Auf der Matte befinden sich mehrere Felder, die im Takt der Musik ausgelöst werden, üblicherweise in dem der Spieler einen Fuß auf die entsprechende Position setzt. Bei „Finger Dance“ wird das Gameplay des Originals derart verändert, dass es mit der Tastatur gespielt werden kann, dabei ersetzt eine Taste ein Feld der Matte und die Anzahl der möglichen Tasten wird auf vier reduziert. Anstelle der visuellen Anzeige, welche Tasten gedrückt werden müssen, wird dies über akustische, gut unterscheidbare Töne dargestellt. Dabei werden für die Differenzierung zwei Merkmale in Kombination benutzt, die Höhe des Tons und die Richtung, also die Ausgabe für ein Ohr oder einen Lautsprecher (vgl. [17]).

2.1.8 Zusammenfassung

Zu jedem der aufgeführten acht Genre gibt es Spiele für blinde Menschen, allerdings ist die Anzahl im Vergleich zu Spielen ohne Zugang wesentlich kleiner und oft sind die Spiele schon älter. Zusammenfassend lässt sich feststellen, dass ein häufiger gewählter Ansatz für Audio Spiele ist, ein bestehendes Spiel oder

[1] Eingabestift, der im Raum bewegt werden kann und an das Trackingsystem angeschlossen ist.

Spielidee für blinde und sehgeschädigte Menschen zugänglich zu machen, seltener sind Neukonzeptionen. Textbasierte Spiele sind naturgemäß ohne eine Anpassung mit Hilfe eines Screenreaders spielbar. Bei anderen Spielen gibt es verschiedene Ansätze die Spiele zugänglich zu machen. Elemente des Gameplays können entfallen oder reduziert werden. Bei „Finger Dance“ zum Beispiel gibt es vier Tasten, die Tanzmatte des Originals verfügt aber über mehr Felder. Bei diesem Beispiel wird auch die Art der Interaktion mit dem Spiel vereinfacht und das Gameplay damit deutlich verändert: Anstelle der Tanz-Bewegung, welche die Benutzung der Matte erfordert, interagiert der Spieler nur mit seinen Händen und der Tastatur. Ein weiterer gängiger Ansatz ist die Umwandlung von visuellen Stimuli in auditive oder taktile. Die Ersetzung von visuellen Stimuli zu auditiven wurde zum Beispiel bei der Portierung von „Frogger“ zu „Audiofrogger“ angewendet. In Kapitel 2.2 werden weitere Beispiele vorgestellt, bei deren Umsetzung visuelle Stimuli durch haptische ersetzt wurden. Bei einer auf Audio basierenden Interaktion werden in [22] die drei Grundarten akustischer Signale – Sprache, Musik und natürliche oder künstliche Geräusche – unterschieden und verschiedenen Aufgabenbereichen zugeordnet. Sprache eigne sich gut dafür, eine große Menge an Informationen zu übertragen, wie zum Beispiel die Beschreibung einer Szene im Spiel. Musik dagegen erzeuge Stimmung und Atmosphäre. Natürliche Geräusche seien gut geeignet für Beschreibungen von Objekten und künstliche Geräusche als Signale. Letztere Gruppe wird auch als Sonifikation bezeichnet. Dabei erfolgt diese je nach Spieltyp mit dreidimensionalem oder Stereo-Sound. Neben der Vertonung des eigentlichen Spiels ist ein auditives Spiel Menü Standard: Die einzelnen Menüeinträge werden vorgelesen, hilfreich kann ein Geräusch sein, welches das Ende des Menüs nach dem letzten Menüeintrag und vor dem ersten Menüeintrag darstellt.

2.2 Multimodal zugängliche Spiele

2.2.1 Überblick

Neben den Spielen die speziell auf die Bedürfnisse von sehgeschädigten Menschen eingehen, existieren auch ein paar wenige multimodal zugängliche Spiele und Prototypen, die sowohl auf sehende als auch auf blinde Spieler ausgerichtet sind. Diese Spiele enthalten nicht nur eine taktile oder auditive Schnittstelle, sondern auch eine visuelle. Um eine Grundlage für die Eingrenzung des Begriffs „multimodal zugänglich“ zu schaffen wird zunächst die menschliche Interaktion mit Spielen modellhaft beschrieben, gefolgt von der Erklärung des Begriffs. Daran schließt sich die Vorstellung von Vertretern multimodal zugänglichen Spielen

an, sowie eine Bewertung, inwiefern diese dem Anspruch der multimodalen Zugänglichkeit genügen.

In [28] wird die Interaktion mit Spielen an Hand eines dreistufigen Modells erklärt. Die erste Stufe ist die Wahrnehmung von Stimuli. Dabei wird in primäre und sekundäre Stimuli unterschieden. Während die primären Stimuli für die Spielbarkeit eines Spiels unerlässlich sind, sind sekundäre Stimuli für die Atmosphäre im Spiel verantwortlich und für die Spielbarkeit vernachlässigbar. Die zweite Stufe ist, dass dem Spieler ermöglicht wird, mehrere Reaktionsmöglichkeiten gegeneinander abzuwägen und eine Reaktion auszuwählen. Die Auswahl der Reaktionsmöglichkeiten wird vom Genre eines Spiels bestimmt, in einem Racing Game beispielsweise wären mögliche Reaktionen *Lenken*, *Bremsen* oder *Beschleunigen*, in einem Adventure könnten mögliche Aktionen beispielsweise das *Untersuchen*, *Aufnehmen* oder *Verwenden* eines Gegenstandes sein. Die letzte Stufe des beschriebenen Modells ist die Ermöglichung der Eingabe. Dabei wird zwischen digitaler und analoger Eingabe unterschieden, genauer ob die Eingabe diskret über die Betätigung von Buttons erfolgt oder kontinuierlich, wie beispielsweise bei der Verwendung eines Joysticks.

2.2.2 Begriffsklärung

Die Definition des Begriffs *multimodaler Zugang*, beziehungsweise *multimodale Schnittstelle* erfolgt in der Literatur mit unterschiedlicher Gewichtung der beteiligten Systemkomponenten. So legt Sharon Oviatt den Schwerpunkt ihrer Definition einer *multimodalen Schnittstelle* auf die Verwendung unterschiedlicher Eingabemodalitäten und nicht auf verschiedene Ausgabe-Modi (vgl. [18, S. 418]). In dieser Arbeit steht jedoch nicht die Eingabe auf multimodale Art und Weise im Fokus, sondern der Aspekt, dass die Ausgabe über verschiedene Modalitäten – folglich multimodal – erfolgen kann. Dies entspricht der Verwendung des Begriffs der Multimodalität von Archambault und Olivier, die „alternative Modalitäten“ für blinde Menschen als „non visuelle Modalitäten basierend auf auditiven und taktilen Sinnen“ beschreiben (vgl. [2]).

Der Begriff *multimodaler Zugang* bezieht sich in dieser Arbeit allein auf die Ausgabemodalitäten, was für Spiele konkret bedeutet, dass zur Rezeption der Spielausgabe mehrere Wahrnehmungskanäle verwendet werden. Dabei gilt zusätzlich die Einschränkung, dass jeder Kanal alleine stehend vollen Zugang zu einem Spiel gewähren muss. Diese Auslegung bezieht sich auf die erste Stufe des vorgestellten Interaktionsmodells, also auf die Bereitstellung von primären Stimuli. Die Rezeption der Ausgabe von klassischen Computerspielen erfolgt üblicherweise über die visuelle, auditive und taktile Wahrnehmung, wobei die

beiden letzteren oft nur als sekundäre Stimuli – für die atmosphärische Untermalung – eingesetzt werden. Damit ein Spiel multimodal zugänglich ist, müssen letzten Endes primäre Stimuli für mindestens zwei der genannten Wahrnehmungsarten bereitgestellt werden.

2.2.3 Audio Space Invaders

Das Gameplay des Spiels „Audio Space Invaders“ [14] aus dem Jahr 2000 ist an das Spielprinzip von „Space Invaders“ angelehnt: Der Spieler steuert ein Raumschiff und muss gegnerische Raumschiffe abschießen. Dabei gibt es eine visuelle dreidimensionale Repräsentation, in denen die gegnerischen Flugobjekte vom Spieler visuell wahrgenommen und somit abgeschossen werden können. Sehende Menschen können das Spiel auch ohne die akustische Darstellung spielen, somit sind die auditiven in diesem Fall sekundäre Stimuli. Allerdings können auch blinde Menschen das Spiel nur mit der akustischen Ausgabe spielen. Über 3D- oder Stereo-Sound können die gegnerischen Flugzeuge auditiv lokalisiert werden. „Audio Space Invaders“ stellt folglich primäre Stimuli sowohl visuell als auch auditiv bereit und ist somit nach der obenstehenden Definition ein multimodal zugängliches Spiel.

2.2.4 Terraformers

Das Spiel „Terraformers“ [26] ist aus dem Jahr 2004, eine Mischung aus den Genre Adventure und FPS und spielt in der Zukunft. Über das sogenannte Terraforming werden fremde Planeten für Menschen erkundet und bewohnbar gemacht. Von einem solchen Planeten kommt ein Notruf und der Spieler ist Teil einer Rettungsmission, bei der die Aufgabe darin liegt, die Ursache des Notrufs zu klären. Beim Anflug des Planetens wird das Flugzeug angegriffen und es kommt zur Bruchlandung. Nur der Hauptcharakter überlebt und ist bei der Mission auf sich alleine gestellt. Das Spiel enthält sowohl eine grafische dreidimensionale Repräsentation der virtuellen Welt als auch eine akustische. Für Menschen, deren Sehvermögen eingeschränkt ist, die aber nicht vollständig blind sind, gibt es einen Modus, der den Kontrast der grafischen Darstellung stark erhöht. Bei Bedarf kann die grafische Darstellung jedoch auch ganz ausgestellt werden, um den Rechenaufwand zu verringern. Damit die akustische Darstellung korrekt erfolgt, müssen die Kopfhörer – beziehungsweise Lautsprecher – richtig angeschlossen sein.

Die akustische Repräsentation enthält Raumklang, zusätzlich werden die Schritte der Spielfigur dargestellt. Dabei unterscheiden sich die Schrittgeräusche

abhängig vom Bodenmaterial. Außerdem verfügt jedes *Gameobject* über eine sprachliche Darstellung und *3D sound icons*, ein individueller Klang oder eine kurze Klangfolge. Für die auditive Navigation werden mehrere Techniken verwendet, die der Spieler zu Beginn des Spiels in einem Trainingsteil erlernt.

Der Hauptcharakter erhält zu Beginn seiner Mission einen Anzug, der mit einem *personal digital assistant* (PDA) ausgestattet ist. Der PDA gibt in sprachlicher Form Hinweise zum Spiel und führt durch das Training. Außerdem warnt der PDA, wenn der Spieler auf Feinde triff. Zur besseren Navigation im Raum kann der Spieler auf einen *Sound Compass*, ein *Sonar* und ein *Global Positioning System* (GPS) zugreifen. Der Sound Compass zeigt dem Spieler mit einer anhaltenden Klangfolge an, wo Norden ist. Schaut der Hauptcharakter in diese Himmelsrichtung, ist der Kompass sowohl links als auch rechts zu hören; wird die Figur nach Osten gedreht, ist der Kompass nur auf dem linken Ohr – beziehungsweise aus dem linken Lautsprecher – zu hören. Zusätzlich kann der Spieler über den Ziffernblock der Tastatur den Hauptcharakter in die vier Himmelsrichtungen und vier Nebenhimmelsrichtungen ausrichten. Eine weitere Funktion des Sound Compass ist die Möglichkeit, über eine Taste die Information über die aktuelle Ausrichtung des Hauptcharakters in sprachlicher Form abzufragen.

Das Sonar gibt den ungefähren Abstand zu dem Objekt an, welches sich vor dem Hauptcharakter befindet. Dafür wird ein anhaltendes Piepen verwendet. Bewegt die Figur sich auf ein Objekt zu, wird der Ton des Sonars höher. Zusätzlich kann mit einer Taste überprüft werden, um was für ein Objekt es sich handelt. Außerdem gibt das Sonar automatisch Rückmeldung, wenn der Spieler einem Schussziel gegenüber steht und schießen kann. Bei Bedarf können das Sonar und der Sound Compass ein- und ausgeschaltet werden. Das GPS kann sprachlich die exakte Position des Hauptcharakters und die der wichtigen Objekten in seiner Umgebung in einem globalen zweidimensionalen Koordinatensystems ausgeben. Die Achsen des Koordinatensystems werden von den beiden Himmelsrichtungen Westen und Norden gebildet – eine Position besteht folglich immer aus einem West-Wert und einem Nord-Wert. Die Hauptcharakter besitzt einen Rucksack, der als Inventar dient und in dem gesammelte Objekte verstaut werden. Dabei wird der Inhalt des Rucksacks in Listenform repräsentiert. Öffnet der Spieler den Rucksack wird pro Listeneintrag ein Ton abgespielt, damit die Länge der Liste auch auditiv erfasst werden kann. Bei der Navigation durch die Liste werden die Namen der Objekte vorgelesen. Das gleiche Prinzip wird auch bei den *Communication Devices* verwendet. Der Spieler kann an diesen Nachrichten abrufen, die Hilfestellung und Aufgaben für den Spielverlauf geben. Neben der sprachlichen Umsetzung werden die Listen auch in Textform dargestellt.

Ein weiteres wichtiges Element sind Türschlösser die geöffnet werden müssen. Es gibt verschiedene Arten von Schlössern, die sich nach einem ähnlichen Prinzip öffnen lassen. Im gleichen räumlichen Abschnitt sind akustische Hinweise für einen Code hinterlegt. Die Türschlösser verfügen über mehrere Eingabemöglichkeit, jede davon ist individuell akustisch dargestellt. Mit Hilfe des gehörten Hinweises kann der Schlüssel erraten und eingegeben werden. Ist der Schlüssel korrekt, wird akustisches Feedback über ein Geräusch einer sich öffnen Tür gegeben. Ein zweites Prinzip sind *Audio Keys*, die jeweils eine eigene Tonabfolge besitzen und im Level gefunden und im Inventar abgelegt werden können. Zu einem solchen Schlüssel gibt es ein entsprechendes Schloss, welches sich öffnet, wenn der Schlüssel damit verwendet wird.

Im Grundansatz ist „Terraformers“ ein multimodal zugängliches Spiel, allerdings fehlen an einigen Stellen primäre visuelle Stimuli. So kann der Spieler die Anleitung im Trainingsteil allein durch die Stimme des PDAs empfangen. Es fehlt eine zusätzliche textuelle Darstellung, wie bei der Umsetzung des Nachrichtensystems oder des Inventars. Ein weiteres Hindernis sind die Audio Keys, die neben der akustischen Darstellung im Trainingsteil noch über ein grafisches 3D-Modell visualisiert werden – im eigentlichen Spiel fehlen diese Modelle jedoch. Die Lokalisierung der Schlüssel ist daher nur über die Benutzung des GPS möglich. Auch für die akustischen Rätsel der anderen Türschlösser wird keine visuelle Alternative geboten. Akustische Stimuli – primäre und sekundäre – werden jedoch in vielfältiger Form bereit gestellt. Der Spieler hat die Wahl unterschiedlicher Navigationshilfen und kann diese parallel verwenden, oder bei Bedarf ausschalten. Sonar, GPS und Audio Compass bieten dem Spieler verschiedene Ansätze, um aus der akustischen Repräsentation der virtuellen Welt ein mentales Modell zu erstellen.

2.2.5 AudioQuake

„AudioQuake“ von 2005 ist ein Wrapper für den FPS „Quake“(1996), der das Spiel um einen auditiven Zugang erweitert, indem zusätzlichen zur akustischen Ausgabe des Spiels eine Text-To-Speech-Engine die Konsolenausgaben der Quake Engine vorliest und darüber hinaus wichtiges akustisches Feedback über den virtuellen Raum bereitstellt (vgl. [4]). Neben der normalen Soundausgabe erweitert der Wrapper das Spiel um zahlreiche akustische Hinweise als Navigationshilfe, aufgelistet in Tabelle 1. Der Spieler kann die Navigationshilfe konfigurieren, indem er das Abspielen einzelner Warnungen per Tastendruck ein- und ausschalten kann. Alle in der Tabelle aufgelisteten Warnung werden mit Hilfe von synthetischen Tonabfolgen verdeutlicht, mit Ausnahme der Schrittge-

räusche. Die Tonabfolgen können im speziellen Fall auch eine räumliche Information tragen. Ein einfaches Beispiel hierfür sind „Z Level Warnings“, bei denen die Z-Richtung über eine steigende Tonhöhe für Aufwärtsbewegungen eine fallende Tonhöhe für Abwärtsbewegungen verwendet werden. Neben den optionalen Navigationshilfen, gibt es weitere Hilfen, die jederzeit per Tastendruck angefordert werden können. Gleiches gilt für die akustische Wiedergabe von Statusinformation und dem Waffenmenü, aufgeführt in Tabelle 2.

Hinweis	Taste	Bedeutung
Wall Warnings	w	Warnung vor Wänden (seitlich und frontal)
Wall Touch Warnings	t	Warnung wenn die Spielfigur eine Wand berührt
Z Level Warnings	z	Interaktionen im Raum auf der Z-Achse (z.B. beim hoch- oder runtergehen von Treppenstufen, Benutzen von Plattformen und des Jetpacks)
Hazard Drop/Ledge Warnings	h	Warnung vor Absturzgefahr die sich vor oder hinter der Spielfigur befindet
Side Hazard Warnings	i	Warnung vor Absturzgefahr die sich seitlich von der Spielfigur befindet
Corner Warnings	x	Hinweis auf Ecken und Kreuzungen
EtherScan RADAR Monster	t	Hinweis auf Monster
EtherScan RADAR Enemy	y	Hinweis auf gegnerische Mitspieler
EtherScan RADAR Friends	k	Hinweis auf befreundete Mitspieler
Footsteps	f	Wiedergabe von Schrittgeräuschen
D5k (Detection 5000)	k	Hinweis auf Gegenstände

Tabelle 1: *Aktivierbare akustische Hinweise in AudioQuake*

Eine weitere Navigationshilfe, um sich im virtuellen Raum besser orientieren zu können, wird dem Spieler gegeben, über die Möglichkeit bis zu 20 Wegpunkte (*waypoints*) in einem Level zu setzen. Waypoints werden automatisch nummeriert und der Spieler kann sie bei Bedarf auch wieder entfernen. Allerdings können waypoints nicht beliebig gelöscht werden, sondern nur absteigend in der Erstellungsreihenfolge, beginnend beim zuletzt hinzugefügten, dann der waypoint, welcher davor erstellte, usw. Befindet sich der Spieler in der Nähe eines waypoints, wird diese Information zusammen mit der Nummer des Weg-

Taste(n)	Information
c	Kompass, Ausgabe der Blickrichtung als Himmelsrichtung
j	Informationen ob der Spieler einen Sprung ausführen kann
l	In Blickrichtung wird freier Raum mit einem Rauschen angezeigt
k	Beschreibung von *drops* in unmittelbarer Nähe zum Spieler
9	Ansage des Lebensenergie-Status und der Rüstung
0	Auflistung der verfügbaren Munition
1-8	Durchschalten und auswählen von Waffen
TAB	Ansage des Punktestandes

Tabelle 2: *Abfragbare Navigationshilfen und Statusinformationen in AudioQuake*

punktes vorgelesen. Allerdings bleiben waypoints nur im laufenden Spiel und lokal für ein Level bestehen, so bald das Level verlassen wird, oder die Spielfigur stirbt und das Level neu geladen wird, sind auch die waypoints nicht mehr vorhanden.

Im Spiel „Quake“ hat der Spieler mehrere Möglichkeiten die Steuerung der Bewegung der Spielfigur einzugeben. Dabei können die Eingabegeräte Tastatur und Maus jeweils für sich genommen oder in Kombination benutzt werden. Die Blickrichtung bestimmt auch die Schussrichtung, vertikales Zielen ist mit Hilfe der Tastatur möglich, jedoch nicht zwingend notwendig, da dies zu einem gewissen Maß automatisch erfolgt. In „AudioQuake“ kann die Eingabe zur Bewegungssteuerung ausschließlich über die Tastatur erfolgen. Die Steuerung erfolgt dabei analog zur Tastatursteuerung von „Quake“, mit einer Anpassung der Drehung der Spielfigur. Das Drehen erfolgt nicht kontinuierlich entsprechend zur Länge des Tastendrucks, sondern um einen festgelegten Winkel, der in den Optionen eingestellt werden kann und initial 22,5° beträgt.

Der in „Quake“ integrierte Chat wird auch über die Konsole ausgegeben und gehört somit zu den Ausgaben, die in „AudioQuake“ automatisch von der Text-To-Speech-Engine vorgelesen werden. Für das Vorlesen der Ausgabe stehen dem Spieler weitere Steuerungsmöglichkeiten per Tastatur zur Verfügung. So kann zwischen den vorgelesenen Nachrichten geblättert, die letzte Nachricht wiederholt, oder das Vorlesen aller wartenden Nachrichten auf die letzte wartende Nachricht reduziert werden.

An dieser Stelle wird deutlich, dass die Anzahl von akustischen Hinweisen und vorgelesenen Informationen in „AudioQuake“ relativ umfangreich ist. Dabei kann es schwer werden zwischen den einzelnen Informationen zu un-

terscheiden, insbesondere wenn viele akustische Signale gleichzeitig auftreten. Auch die Zuordnung von Geräuschen und die damit übertragende Information muss bei vielen akustischen Hinweisen erst erlernt werden. Insbesondere wenn ein akustischer Hinweis nicht aus einem natürlichen Geräusch besteht. Bei der Verwendung von natürlichen Geräuschen, ist das nicht nötig, die Zuordnung von Schrittgeräuschen als Bewegungsinformation beispielsweise erfolgt instinktiv richtig.

Mit den akustischen Hinweisen ist es möglich, sich in der virtuellen Welt zu bewegen und Gegner abzuschießen. Die grundlegenden Spielmechaniken sind demnach auch auditiv verfügbar, womit „AudioQuake“ zu den multimodal zugänglichen Spielen zu zählen ist. Allerdings werden visuelle und auditive Stimuli nicht immer zur gleichen Zeit gegeben, womit ein Ungleichgewicht entsteht, wenn das Spiel im Mehrspielermodus gespielt wird. Während ein Gegner schon im Sichtfeld eines Spielers erkennbar ist, kann es sein, dass der „EtherScan RADAR“ diesen Gegner noch nicht erkannt und akustisch signalisiert hat. Um diesem Ungleichgewicht entgegen zu treten, war ursprünglich ein „Fairplay-System“ geplant (vgl. [4]).

2.2.6 AudiOdyssey

Aus dem Jahr 2007 ist der Prototyp „AudiOdyssey“ [10] aus dem Genre Dance/ Rhythm games und kann entweder mit der Wiimote oder der Tastatur gespielt werden. Der Spieler schlüpft in die Rolle eines angehenden DJs. Im Spiel ist ein Grundrhythmus zu hören, ein Signal gibt an, wann eine Taste gedrückt werden muss. Das Signal ist akustisch als Beat zu hören, die entsprechende Taste wird von einer Sprecherin angesagt. Dabei sind Signale für die linke Pfeiltaste auf dem linken Ohr zu hören, für die rechte Pfeiltaste auf dem rechten Ohr und für die Pfeil-oben-Taste auf beiden Ohren. Zusätzlich kann der Spieler sich über die Leertaste die Taste ansagen lassen. Visuell ist das Signal als weißer Punkt zu sehen, dieser Punkt erscheint für jede Taste an einer unterschiedlichen Stelle. Anhand der visuellen Stimuli erschließt sich jedoch nicht, welcher Punkt welcher Taste zugeordnet ist. Wird eine Taste im richtigen Moment gedrückt, wird eine Tonfolge abgespielt und das DJ-Pult leuchtet auf. Das drücken einer falschen Taste wird auditiv durch einen Fehler-Ton angezeigt, visuell ist es nur durch das Ausbleiben des positiven Feedbacks (Aufleuchten des DJ-Pults) wahrzunehmen. Wurde fünf Mal in Folge die Taste im richtigen Takt gedrückt, wird der Beat dem Grundrhythmus hinzugefügt und ein neuer Beat kann erspielt werden. Wurden alle Beats erfolgreich erspielt, ist ein Lied abgeschlossen. Insgesamt enthält der Prototyp zwei Lieder von unterschiedlichem Schweregrad.

Im Ansatz ist „AudiOdyssey“ multimodal zugänglich, allerdings erfolgt die Spielanleitung rein akustisch, ebenso wie die Zuordnung der Tasten nur auditiv eindeutig erfasst werden kann. Dadurch fehlen primäre visuelle Stimuli. Wenn die Zuordnung der Tasten allerdings durch Ausprobieren oder Erinnern bekannt ist – was bei nur zwei Lieder schnell der Fall ist – kann das Spiel auch ohne akustische Ausgabe gespielt werden. Eine visuelle Spielanleitung müsste allerdings in jedem Fall noch ergänzt werden, um die Kriterien des multimodalen Zugangs zu erfüllen.

2.2.7 Blind Hero

Das Spiel „Blind Hero“ [27] ist aus dem Jahr 2008 und eine Adaption des Musik- und Rhythmus-Spiels „Guitar Hero“. Im Original „Guitar Hero“ kann der Spieler das Spielen mit einer Gitarre nachempfinden. Zur Eingabe wird ein Controller verwendet, dessen Form einer Gitarre nachgebildet ist, am Gitarrenhals sind nebeneinander fünf farbige Tasten angebracht. Um ein Lied zu spielen muss der Spieler im richtigen Moment die richtigen Tasten anschlagen. Auf dem Bildschirm ist eine räumliche Darstellung eines fünfsaitiges Gitarrenhals zu sehen, jede Saite entspricht dabei einer Taste. Entlang der Saiten bewegen sich Punkte und Linien auf den Spieler zu, dabei symbolisieren Punkte einen einfachen Anschlag und Linien das Halten der Saite, beziehungsweise Taste. Durch die räumliche Darstellung wird dem Spieler zusätzlich eine Vorschau auf die nächsten anzuschlagenden Tasten gegeben. Überqueren die Punkte den vordersten Bund des dargestellten Gitarrenhalses, muss der Anschlag oder das Halten der Controller-Tasten erfolgen. Passiert das im richtigen Takt, wird visuelles und akustisches Feedback gegeben und der Punktestand des Spielers erhöht sich. Der Spieler kann die gleichen Lieder in drei Schwierigkeitsstufen spielen, je schwieriger die Stufe ist, desto mehr Tasten muss der Spieler bedienen.

Für die blinden-gerechte Adaption muss der Quellcode des Spieles angepasst werden. Da der Quellcode von „Guitar Hero“ nicht frei verfügbar ist, wurde das Open-Source Spiel „Frets on Fire“ als Grundlage für die Adaption verwendet – ein in Python geschriebener Klon von „Guitar Hero“ (vgl. [27]). Bei „Blind Hero“ werden zu den visuellen primären Stimuli aus „Frets on Fire“ taktile primäre Stimuli ergänzt. Auditive Stimuli sind schon in Form der Wiedergabe der Lieder vorhanden und können nicht zur Steuerung verwendet werden. Für die Erzeugung der taktilen Stimuli kommt besondere Hardware zum Einsatz: Ein Handschuh, der an vier Fingern Motoren besitzt um haptische Signale zu geben. Jeder Finger entspricht dabei einer Taste. Da der Daumen zum Halten des Gitarrencontrollers benötigt wird, empfängt dieser keine Signale. Durch diese

Umsetzung wird das ursprüngliche Gameplay verändert, es können nur vier der fünf Tasten des Controllers zum Spielen benutzt werden. Außerdem können die Motoren keine taktile Vorschau auf die kommenden Tasten geben, außer durch die Verwendung von mehreren Motoren pro Finger, was die Lernkurve zu Beginn des Spieles zu steil ansteigen lassen und den Spaß am Spielen vermindern würde (vgl. [27]). Ein weiteres Problem seien die Kosten des Handschuhs und dass ein solcher Handschuh auch für unterschiedliche Handgrößen erstellt werden müsste.

In Hinsicht auf die Multimodalität des Zugangs lässt sich sagen, dass „Blind Hero“ primäre Stimuli sowohl visuell als auch taktil bereitstellt und somit ein multimodal zugängliches Spiel ist – mit der Einschränkung, dass die Vorschau auf kommende Tasten nicht als primäre Stimuli betrachtet werden. Ein gleichberechtigter Zugang erfolgt nicht, vermutlich der Grund warum die Tests in [27] mit blinden Menschen und sehenden Menschen mit verbundenen Augen durchgeführt wurden.

2.2.8 Rock Vibe

„Rock Vibe“ [1] aus dem Jahr 2009 ist ebenfalls ein Rhythmus Spiel und eine Adaption des Spiels „Rock Band“. Dieses Spiel ermöglicht im gleichen Stil wie „Guitar Hero“ das Spielen von Instrumenten nachzuempfinden, allerdings ist es nicht nur auf Gitarren- Instrumente begrenzt, sondern kann auch mit einem Schlagzeug Controller oder Mikrofon und Gesang gespielt werden. Die Adaption des Spiels implementiert nur einen taktilen Ansatz für den Schlagzeug-Controller. Dieser ist ähnlich wie ein Schlagzeug aufgebaut und verfügt über vier Trommeln und ein Fußpedal. Das Gameplay gleicht dem des Spiels „Guitar Hero“, ein Spieler muss nach Erhalt eines Stimulis im richtigen Takt die Trommeln schlagen oder das Pedal bedienen. Die Darstellung der visuellen Stimuli erfolgt ebenfalls analog zu der bei „Guitar Hero“. Die Adaption „Rock Vibe“ ersetzt visuelle Stimuli teilweise durch taktile und teilweise durch auditive. Beim Spielen eines Liedes werden die visuellen Signale wann welcher Teil des Schlagzeugs bedient werden soll durch taktile Stimuli ersetzt. Dafür wurden fünf kleine Vibrationsmotoren mit Klettverschluss-Bändern am Spieler befestigt, jeweils am linken und rechten Unter- und Oberarm für die vier Trommeln und einer am Knöchel für das Pedal (vgl. [1]). Das visuelle Feedback, ob das richtige Element des Controllers im richtigen Takt gespielt wurden, wurde durch akustisches Feedback ersetzt, bei Misserfolg ertönt ein Fehlerton, bei Erfolg wird eine Schlagfolge abgespielt. Der Titel des Liedes wird am Anfang und der erreichte Punktestand am Ende eines Liedes sprachlich wiedergegeben. Neben dem eigentlichen Spiel

wurde auch eine akustische Menüführung implementiert und das Menü um eine Kalibrierung der Vibrationsmotoren erweitert.

Hinsichtlich der Multimodalität des Zugangs lässt sich sagen, dass zwar insgesamt drei unterschiedliche Wahrnehmungsarten benutzt werden, aber nur visuell alle primären Stimuli dargestellt werden. Für die blinden-gerechte Umsetzung werden für primäre Stimuli auditive und taktile Signale verwendet, die für sich alleine genommen keinen Zugang zum Gameplay gewähren.

2.2.9 Zusammenfassung

Wenige Spiele, welche einen multimodal zugänglichen Ansatz verfolgen, ermöglichen einen vollwertigen multimodalen Zugang nach der Begriffseingrenzung in Kapitel 2.2.2. Einige der vorgestellten Vertreter benutzen spezielle Hardware, deren Verwendung sich nicht auf andere Spielprinzipien anwenden lässt. Die Konzepte der Spiele „Blind Hero“, „Rock Vibe“ und „AudiOdyssey“ sind sehr genrespezifisch und lassen sich nur teilweise auf andere Genre übertragen. So ist die Ersetzung von visuellem durch haptisches Feedback auf wenige Signale begrenzt und erfordert zusätzlich spezielle Hardware. Ein komplexeres Spiel, welches die Navigation durch eine virtuelle Welt erfordert, ist mit diesen Mitteln kaum umzusetzen. Der haptische Kanal kann nützlich sein, um die Wahrnehmung einer virtuellen Welt zu verstärken, wie es auch durch den Einsatz von force feedback in vielen Spielen praktiziert wird. Jedoch reicht dieser Kanal nicht aus, um komplexere Zusammenhänge darzustellen; zumindest nicht so lange es keine interaktiven plastischen Modelle gibt, die eine Spielwelt vollständig abbilden können. In Verbindung mit einer akustischen Repräsentation, wie es auch in den vorgestellten Beispielen geschieht, kann haptisches Feedback durchaus visuelle Signale ersetzen, insbesondere wenn die Menge der zu ersetzenden Signale gering ist.

Die Spiele „Audio Space Invaders“, „AudioQuake“ und „Terraformers“ verfolgen einen multimodal zugänglichen Ansatz, indem sie neben einer visuellen Schnittstelle auch eine akustische Schnittstelle bereitstellen. Aus diesen Spielen lassen sich einige gute Konzepte für die Ersetzung von visuellen durch auditive Stimuli ableiten, die größtenteils auch auf andere Spielgenre übertragen werden können. Das Spiel „AudioQuake“ zeigt als Erweiterung zu einem bestehenden Spiel – welches sich primär auf visuelle Stimuli stützt – gut auf, welches Feedback auch akustisch bereit gestellt werden muss. Neben der auditiven Darstellung der Menüs ist eine Wahrnehmung des virtuellen dreidimensionalen Raumes wichtig. Bewegt sich der Spieler, muss er hören können, dass die Bewegung erfolgt, wie weit er von Hindernissen entfernt ist und ob er auf einen

Abgrund zu läuft. Die Verwendung von Radaren die mit Hilfe von *Earcons*[2] anzeigen, wo sich beispielsweise Hindernisse befinden, wird sowohl bei „Terraformers“ als auch bei „AudioQuake“ eingesetzt. Dabei sollte beachtet werden den Spieler nicht mit zu vielen gleichzeitigen akustischen Signalen zu überfordern.

Auch die Verwendung von Text-To-Speech-Engines zur Sprachsynthese haben beide Ansätze gemein. So können Inhalte auch dynamisch zur Laufzeit generiert werden. In Verbindung mit dieser Technologie lassen sich akustische Listen erstellen, welche sich nicht nur für die Ersetzung von visuellen Menüs außerhalb des eigentlichen Spiels, sondern für die Darstellung eines Inventars oder des Inhalts eines Raumes eignen. Jedes der drei visuell und auditiv zugänglichen Spiele verfügt über einen auditiven Kompass, der die Orientierung im Raum erleichtert. Objekte im Raum werden über 3D-Sound ortbar gemacht. Das in „Terraformers“ verwendete GPS hilft dem Spieler zusätzlich ein konkretes mentales Bild von dem virtuellen Raum und Objekten, die sich darin befinden, zu erlangen. Dadurch, das die Orientierungshilfen optional und individuell ein- und ausschaltbar sind, kann der Spieler selbst entscheiden, ob er die Räume spielerisch erkunden möchte, oder ob er anhand von seiner Koordinate und den Koordinaten eines Objekts gezielt darauf zu gehen möchte. Bei der Konzeption eines multimodalen Spiels ist es wichtig, alle Spielmechaniken für alle Wahrnehmungskanäle bereitzustellen. Das wurde bei dem Spiel „Terraformers“, das überwiegend multimodal spielbar ist, bei den akustischen Rätseln deutlich. Diese Rätsel sind über eine visuelle Schnittstelle allein nicht zu lösen.

2.3 Fazit

Die Erarbeitung des States of the Art hat gezeigt, dass es bereits eine Vielzahl von Audiospielen aus unterschiedlichen Genre gibt. Spiele, die einen multimodal zugänglichen Ansatz verfolgen und konsequent zu Ende führen gibt es sehr wenige. Eine Game-Engine für die Erstellung von multimodal zugänglichen Spielen existiert nicht. Der Ansatz der bei „AudioQuake“ gewählt wurde, kommt dem einer Game-Engine mit multimodalen Zugang relativ nah, weil die bestehen *ZQuake-Engine* angepasst und um einen Audio-Wrapper erweitert wurde. Allerdings ermöglicht dieser nur, das vorhandene Spiel „Quake“ akustisch zugänglich zu machen. Außerdem ist der Wrapper losgelöst von der eigentlichen Engine, was weitere Schwierigkeiten mit sich bringt. Es war recht zeitaufwän-

[2] Earcons sind synthetische abstrakte Töne oder Tonfolgen, welche in strukturierter Kombination Informationen übermitteln können (vgl. [6]). Ein bekanntes Beispiel dafür sind akustische Rückmeldungen eines Betriebssystems, bei Fehler oder Hinweismeldungen. Dabei unterscheiden sich die Earcons für Fehler und Hinweise und allein anhand des akustischen Signals erkennt der Benutzer, ob es sich um einen Fehler oder Hinweis handelt.

dig, das Spiel „AudioQuake“ für einen Test zum Laufen zu bringen. Das liegt zum einen daran, dass „Quake“ an sich alt ist, aber aus dem Wrapper heraus gestartet werden muss, um akustisches Feedback zu erhalten. Das schließt die Verwendung herkömmlicher Emulatoren aus. Zielplattformen für „AudioQuake“ sind Linux, Windows XP und Mac OS X Leopard (vgl. [3]).

„Quake“ ist unter Linux darauf ausgelegt, den OSS-Soundserver zu verwenden. Unter Ubuntu ist dieser Soundserver mittlerweile durch den ALSA-Soundserver ersetzt. Theoretisch besteht die Möglichkeit den OSS-Soundserver zu installieren, praktisch ist das allerdings nur mit unverhältnismäßigen Aufwand unter Ubuntu möglich. Es gibt den Wrapper „ALSA-OSS“, der dem Spiel eine OSS-Schnittstelle bereitstellt und die Ausgabe entsprechend an den ALSA-Soundserver weiterleitet, so kann „Quake“ mit Soundausgabe gestartet werden. Allerdings ist die von „AudioQuake“ verwendete Text-To-Speech-Engine nicht kompatibel mit diesem Wrapper, das Spiel startet und die Sounds aus dem Spiel sind zu hören, die Ausgabe der Text-To-Speech-Engine hingegen nicht. Ohne Verwendung des Wrappers kann das Audio-Menü von „AudioQuake“ geöffnet und das Spiel gestartet werden. Allerdings kommt es zu einem Spielabsturz, sobald der erste spielinterne Sound abgespielt werden soll, da dafür nicht der richtige Soundserver zur Verfügung steht. Unter Windows 7 lässt sich das Spiel aus „AudioQuake“ heraus weder mit einem 32-bit, noch mit 64-bit System starten. Letzteres schließt sich aus, weil „Quake“ damit nicht kompatibel ist. Unter Windows 7 ist es nicht möglich eine Konsolen-Anwendung im Vollbildmodus zu starten, was allerdings für „Quake“ erforderlich wäre. Unter Windows XP lässt sich die benötigte Text-To-Speech-Engine, das Spiel und auch der Wrapper problemlos installieren. Allerdings hat der zeitliche Aufwand, um das Spiel auf aktuellen Betriebssystemen zum Laufen zu bringen und das Scheitern dieses Vorhabens gezeigt, dass eine betriebssystem- und plattformunabhängige Lösung angestrebt werden sollte. Außerdem sollte das Audio-Interface fester Bestandteil der Game-Engine sein, damit die beschriebenen oder ähnliche Kompatibilitätsprobleme mit dem genannten Audio-Wrapper nicht auftreten können. Falls eine bestehende Game-Engine verwendet werden soll, sollte der Quellcode dieser Engine frei verfügbar und erweiterbar sein. Ein weiterer Grund der für eine Open-Source-Lösung spricht, ist die Wiederverwendbarkeit für weiterführende Forschungsprojekte.

Für die Verwendung und Erweiterung einer bereits bestehenden Game-Engine mit visueller Schnittstelle spricht, dass es solche Game-Engines schon gibt und kein wissenschaftlicher Mehrwert durch die Implementierung einer solchen geschaffen würde. Der Fokus der Entwicklung würde somit auf der Entwick-

lung einer auditiven Schnittstelle liegen. Eine für diesen Ansatz geeignete Game-Engine ist Fabula [5]. Fabula ist eine Open Source Game-Engine, geeignet für die Genre Adventure, Rollenspiel, Strategiespiel und Interactive Storytelling. Dabei ist Fabula eine semantische Game-Engine ohne festgelegten Ausgabemodus. Die Ausgabe eines Spieles kann daher auf unterschiedliche Arten erfolgen, zum Beispiel visuell (2D, 3D, Text) oder auch auditiv, wenn dafür ein entsprechender Client existiert. Aktuell existiert ein 2D-Client für die visuelle Ausgabe eines Spiels. Mit Verwendung und Erweiterung dieser Engine könnten folglich Spiele der vier unterstützen Genre mit multimodalen Zugang erstellt werden. Da es für Fabula keine verfügbaren Spiele gibt, die sich für eine multimodal zugängliche Umsetzung eignen, muss ein solches konzipiert und entwickelt werden.

Für die Konzeption eines solchen Spiels ergeben sich aus dem erarbeiteten States of the Art konkrete Konzepte für die Gestaltung einer auditiven Schnittstelle. Dazu gehören die Verwendung von Geräuschen aus unterschiedlichen Klassen, namentlich Sprache, künstliche Töne und Earcons, Musik, sowie die Verwendung von natürlichen Geräuschen für unterschiedliche Teilaufgaben. Es hat sich gezeigt, dass Sprache am Besten dazu geeignet ist, längere Informationen zu übermitteln. Schon bei einem sehr kleinen Spiel ist die Menge an anfallenden Texten, welche vertont werden müssen, relativ groß. Der zeitliche Aufwand um diese Texte einsprechen zu lassen ist hoch, die dabei entstehenden Audiodateien benötigen viel Speicherplatz und falls sich etwas im Spiel ändert, können die Dateien nicht flexibel verändert werden, sondern müssen in der Regel neu eingesprochen werden. Inhalte, welche sich erst zur Laufzeit ergeben können nicht dynamisch erstellt und wiedergegeben werden. Um dieser Herausforderung zu begegnen, bietet sich der Einsatz von Sprachsynthese in Form einer Text-To-Speech-Engine an, die dynamisch, auch zur Laufzeit Text in Sprache umwandeln kann.

In Verbindung mit dieser Technologie, kann eine weitere Komponente, die sich sowohl in vielen Audio-Spielen als auch in der auditiven Schnittstelle von „Terraformers“ bewährt hat umsetzen: Auditive Listen. Diese können durchblättert werden, dabei wird jeder Eintrag einer Liste vorgelesen. Bei Bestätigung wird der Inhalt des aktiven Listeneintrages vorgelesen, wie zum Beispiel die räumliche Position eines Objektes, in einer Liste die alle Objekte eines Raumes enthält. Allerdings ist es bei der akustischen Wiedergabe besonders bei längeren Texten wichtig, dass der Spieler nicht jeden Eintrag vollständig anhören muss. Menüs die häufig verwendet werden sind dem Spieler nach kurzer Spielzeit vertraut und sollten daher nur kurze, aber prägnante Informationen enthalten. Außerdem ist es für die Orientierung im Spiel wichtig, dass zu Beginn einer Lis-

tennavigation der Titel dieser Liste vorgelesen wird und das nach dem Verlassen einer Liste akustisches Feedback gegeben wird, wo der Spieler sich befindet. In „Terraformers“ werden Listen dieser Art verwendet und um einen Ton erweitert, der das Ende einer Liste anzeigt. Beim Durchblättern der einzelnen Einträge wird am Ende der Liste nach dem letzten Eintrag ein kurzer Ton abgespielt, bevor das Vorlesen mit dem ersten Listeneintrag fortgeführt wird. Gerade bei umfangreichen Listen mit vielen Einträgen ist das hilfreich um einen Überblick über die aktuelle Position in der Liste zu bewahren. Akustische Listenmenüs eigenen sich gut um Menüs und auch komplexere Inhalte darzustellen.

Ein weiteres Konzept für auditiv zugängliche Spiele ist die Verwendung von *Earcons*. Earcons eignen sich um ein kurzes und schnelles akustisches Feedback zu geben. Allerdings ist es bei der Verwendung zu beachten, dass sich verschiedene Eracons deutlich genug voneinander abheben. Bei „AudioQuake“ ist die Anzahl von Warntönen die über Earcons signalisiert werden relativ groß, was zum einen den Nachteil mit sich bringt, dass es einige Zeit braucht, die richtige Zuordnung von Earcon und Information zu lernen. Zum anderen kann es passieren, dass relativ viele akustische Hinweise zeitgleich auftreten, was zur Überlagerungen führt und einzelne Earcons nicht mehr deutlich zu erkennen sind. Deshalb erscheint es sinnvoll, wenn viele Eracons in einem Spiel verwendet werden,dem Spieler selbst entscheiden zu lassen, welche akustischen Hinweise er hören möchte. Außerdem sollte der Einsatz von Earcons so gewählt werden, dass dem Spieler nicht zu viele Hinweise gleichzeitig gegeben werden, so dass er die so übertragenden Informationen zuordnen kann. Ein gutes Gebiet für den Einsatz von Earcons könnte auditives Feedback sein, wenn der Spieler auf ein Hindernis trifft. In dem Spiel „Sarah and the Castle of Witchcraft and Wizardry“ erfolgt dieses Feedback über die sprachliche Ausgabe von „Autsch“, was etwas aufdringlich ist, gerade wenn viele Kollisionen in kurzer Zeit aufeinander folgen. Dies kann aber relativ häufig der Fall sein, wenn ein Spieler sich ohne visuelles Abbild durch eine virtuelle Welt bewegt. Ein kurzes Earcon signalisiert in sehr schnell und weniger aufdringlich, dass der Spieler nicht weitergehen kann.

Für die Charakterisierung von Objekten eignen sich Earcons allerdings weniger. In einem Spiel können relativ viele unterschiedliche Objekte existieren und es wäre eine große Lernleistung erforderlich, für alle diese Objekte eine abstrakte Tonfolge zu erinnern, um daraus die richtigen Informationen abzuleiten. Intuitiver für dieses Gebiet sind natürliche Geräusche, dabei muss der Spieler keine neue Zuordnung von Geräusch und Bedeutung lernen. Ein Beispiel hierfür wäre die akustische Darstellung eines Hundes durch die Wiedergabe von „Bell“-Geräuschen. Allerdings ist zu Bedenken, dass nicht jedes Objekt über ein

eindeutiges Geräusch verfügt und zugeordnet werden kann. Von daher sollten Objekte auch über einen Namen verfügen, der akustisch repräsentiert werden kann, etwa durch Vorlesen des Namens durch eine Text-To-Speech-Engine oder das Abspielen einer Sound-Datei, welche eine sprachliche Beschreibung enthält.

Bewegungsfeedback kann dagegen gut durch natürliche Geräusche alleine erfolgen; Schrittgeräusche werden intuitiv einer sich bewegenden Figur zugeordnet. Wenn die Schritte nach einer Richtungseingabe des Spieler abgespielt werden, kann dieser die Schrittgeräusche der Bewegung seiner Spielfigur zuordnen. Diese Art des des Bewegungsfeedbacks für „Laufen“ lässt sich natürlich auch auf andere Bewegungsarten anwenden. Im *Racing Game* „Top Speed 3“ wurden zum Beispiel Motorengeräusche benutzt, um die Fortbewegung des Rennwagens zu verdeutlichen. Anhand der Geräusche war nicht nur klar, dass der Rennwagen fährt, zusätzlich waren Rückschlüsse auf Beschleunigung und Geschwindigkeit möglich. Diese Rückschlüsse sind auch bei Schrittgeräuschen möglich, in dem man die Wiedergabe beschleunigt. Das Bewegungsfeedback der eigenen Spielfigur sollte sich dabei klar von dem anderer Spieler unterscheiden. Das kann durch die Verwendung unterschiedlicher Geräusche für unterschiedliche Figuren oder über eine räumliche Positionierung der Geräusche erreicht werden. Dabei hat die Recherche des States of the Art ergeben, dass für einen räumlichen Klangeindruck nicht zwangsweise die Audiowiedergabe mit Hilfe von Surround-Sound-Ausgabegeräten erfolgen muss, sondern eine Stereo-Wiedergabe, vorzugsweise in Verbindung mit Kopfhörern ausreicht (vgl. [23]).

Nicht alle vorgestellten Spiele erfordern die Navigation durch einen virtuellen Raum. Bei den beschriebenen Musik- und Rhythmusspielen entfällt dieser Punkt vollständig, da der Spieler sich mehr oder minder statisch an der gleichen Stelle befindet. Bei fast allen anderen vorgestellten Spielen ist die Navigation durch eine virtuelle Welt jedoch Bestandteil des Spiels. Dafür werden je nach Genre unterschiedliche Konzepte angewendet: Von einem einfachen Koordinatensystem bei „Tacticle Battle“, das schlicht die Koordinaten der Karte vorliest, über die Orientierung allein über Motoren- und Fahrtgeräusche, bei „Top Speed 3“ bis hin zu einer mit Earcons vertonten virtuellen Umgebung bei „AudioQuake“. Einige Spiele verwenden auch eine Kombination von Sprache, Earcons und natürlichen Geräuschen, wie zum Beispiel „Terraformers“. Neben den natürlichen Geräuschen und Earcons, die auf Barrieren hinweisen, kann der Spieler auch auf ein Koordinatensystem zurückgreifen und kann so seine Position und die von Objekten in seiner Umgebung bestimmen. Ideal ist es, wenn so ein Koordinatensystem optional verfügbar ist und nicht bei jedem Schritt angesagt wird, so wird der Spieler nicht um das Gameplay-Element des Entdeckens von Gegen-

ständen gebracht und hat trotzdem die Sicherheit, falls exploratives Vorgehen nicht zielführend ist, Hilfestellung bei der Orientierung zu erhalten.

Ein weiteres Konzept, das die Orientierung im virtuellen Raum fördert ist die Verwendung eines *Audio-Kompasses*, der dem Spieler die Himmelsrichtung seiner Blickrichtung ansagt. Zusätzlich besteht die Möglichkeit, die Spielfigur in eine bestimmte Himmelsrichtung zu drehen. Diese Funktion wird sowohl in „AudioQuake“ als auch in „Terraformers“ angewendet und lässt sich gut auf drei- und zweidimensionale Spielkonzepte übertragen, wenn sie aus einer Egoperspektive gespielt werden. Für Spiele, die in einer anderen Perspektive angelegt sind, macht ein Audio-Kompass nur dann Sinn, wenn die Bewegungssteuerung eine Drehung der Spielfigur um ihre eigene Achse enthält. Ist dies nicht der Fall, bleibt die Blickrichtung einer Spielfigur konstant und würde immer der gleichen Himmelsrichtung entsprechen.

Während der Erarbeitung des States of the Art konnten eine Vielzahl von Herangehensweisen für die Umsetzungen einer auditiven Schnittstelle und zum Teil auch für die Umsetzung eines multimodal zugänglichen Spiels evaluiert werden. Die daraus gewonnen Erkenntnisse können in den Entwurf eines eigenen multimodal zugänglichen Spiels einfließen. Einige Konzepte, wie zum Beispiel die Verwendung einer Text-To-Speech-Engine, sind so grundlegend, dass sie sicher für die Umsetzung benutzt werden. Bei anderen Konzepten, wie zum Beispiel der Verwendung eines Audio-Kompasses bleibt noch abzuwägen, ob sie sich für die zu berücksichtigenden Genre der Game-Engine Fabula eignen.

3 Entwurf eines multimodal zugänglichen Spiels

3.1 Vorüberlegungen zum Vorgehen

Das Spiel sollte für zwei Spieler ausgelegt sein, die kooperativ miteinander spielen. So kann der direkte Vergleich des auditiven und des visuellen Zugangs zum Spiel erfolgen und überprüft werden. Ausgehend von dieser Überlegung muss ein geeignetes Genre gefunden werden. Mit Fabula lassen sich Spiele aus den Genre Adventure, Rollenspiel, Strategiespiel und Interactive Storytelling realisieren. Grundsätzlich ist jedes dieser Genre für ein multimodal zugängliches Spiel-Konzept geeignet. Bei dem Genre Interactive Storytelling hängt es von der Umsetzung ab, wie viel Interaktion möglich ist. Es gibt interessante Ansätze für mobile Geräte, bei denen eine Geschichte interaktiv in einer Augmented Reality erzählt wird und die Spieler reale Orte aufsuchen müssen, um der Handlung zu folgen. Bei einer Umsetzung für einen Computer sind die Möglichkeiten der Interaktion schon begrenzter. Mit den Mitteln von Fabula wäre die Darstellung der Geschichte mit Hilfe von Videospiel-Sequenzen und über eine textuelle Ausgabe denkbar. Für die Gestaltung der auditiven Schnittstelle hieße das, dass die Videosequenzen mit weiteren sprachlichen Informationen versehen werden müssten, ähnlich wie bei der Audiodeskription von Filmen. Allerdings eignet sich das Genre eher für einen Einzelspielermodus und da es keine direkte Manipulation der virtuellen Welt gibt, würde ein Spiel in erster Linie rein sprachliche Elemente enthalten und nicht die vollen Möglichkeiten einer auditiven Schnittstelle erschöpfen.

Eine Gemeinsamkeit von Adventure-Spielen und Rollenspielen ist die Interaktion der Spielfigur in einer virtuellen Welt, häufig beinhaltet dies die Möglichkeit, sich in der Welt zu bewegen und mit anderen Spielern, NPCs und Objekten zu interagieren. Bei einem Rollenspiel gibt es neben der Interaktion mit einer virtuellen Welt in der Regel ein System um seinen Charakter auszubauen und zu erweitern, Der bisher bestehende 2D-Client von Fabula müsste allerdings noch um die visuelle Anzeige des Charaktersystems erweitert werden. Da das aber nicht Schwerpunkt dieser Arbeit sein soll, würde es sich anbieten den Teil zu implementieren, der auch schon mit dem visuellen 2D-Client abgebildet ist: Die Interaktion mit Gegenständen, Spielern und NPCs in einer virtuellen Welt. Das würde auf Spiele der Genre Strategie und Adventure zutreffen, bei beiden Genre kann der Spieler sich durch die Welt bewegen, lediglich die Arten der Interaktion mit Objekten, anderen Spielern und NPCs würden sich unterscheiden. Bei einem Abenteuer-Spiel wären typische Interaktionen das Ansehen, Aufsammeln, Benutzen von Objekten, sowie das Sprechen mit anderen Spielern oder

NPCs. Bei einem Strategiespiel könnten die Interaktionen so aussehen, wie bei „TacticalBattle“, demnach wären Angreifen, Heilen und Verteidigen mögliche Interaktionen. Da der 2D-Client dabei besser die Anforderungen eines Adventures trifft, wird dieses Genre für die Konzeption gewählt. Für die Konzeption des Spiels ergibt sich daraus, dass die genannten Interaktionen für dieses Genre als Grundlage für die Auswahl der Spielmechaniken dient.

Die Benutzerschnittstelle des Spiels soll möglichst mit einem Standardaufbau erfolgen, dass heißt die Verwendung von Tastatur und Maus für die Eingabe und für die Ausgaben des Spiels Bildschirm beziehungsweise Stereo-Kopfhörern. Der Verzicht auf spezielle Ein- und Ausgabegeräte bringt zum einen den Vorteil, dass potentielle Spieler Zugang zum Spiel haben, ohne Neuanschaffungen tätigen zu müssen. Zum anderen können die Ergebnisse dieser Arbeit leichter von anderen Forschungsprojekten aufgegriffen, getestet und bei Bedarf praktisch nachvollzogen werden.

Die Handlung des Spielkonzepts wird im thematischen Genre „Fantasy“ angesiedelt. Dieses Genre ist in vielen Medien vertreten, nicht nur in Videospielen, sondern auch in der Literatur und in Filmen und sollte daher leicht zugänglich sein. Bestimmte Rollen und Stereotypen sind etabliert und benötigen daher keiner ausführliche Beschreibung, um ein Bild des Charakters zu erzeugen. Für das Spiel werden unterschiedliche Rollen für die Hauptfiguren ausgewählt, dessen verschiedene Eigenschaften eine logische Trennung im Handlungsverlauf ermöglicht. So kann das Spiel einmal im Mehrspielermodus als auch in Teilen im Einzelspielermodus getestet werden. Die verwendeten Rollen sind eine Fee, die fliegen kann und ein Gnom, welcher sehr klein ist. So kann der Gnom in Teile der virtuellen Welt vordringen, für welche die Fee zu groß ist und die Fee dagegen kann Teile erkunden, die sie nur erreicht, weil sie fliegen kann. Für die Konzeption wird nur ein kleiner Teil eines vollständigen Spiels benötigt, der Ausschnitt der Handlung, der umgesetzt werden soll, wird im folgenden Kapitel beschrieben.

3.2 Handlungsbeschreibung

Zu Beginn des Spiels kann jeder Spieler einen Charakter wählen. Dabei stehen die Fee Cassandra und der Gnom Kuni zur Auswahl. Beide Wesen befinden sich in Anlehnung an ein klassisches narratives Konzept[3] auf einer Reise, die sie durch den Zauberwald führt. Die Handlung des Prototyps ist allerdings erstmal auf den Ausschnitt begrenzt, in dem die Protagonisten versuchen den Eingang zum Zauberwald zu finden.

[3] vgl. Abschnitt 2.1 „Narrative Structure“ in [9]

Der Eingang liegt auf der anderen Seite eines Flusses, die alte Brücke, die über den Fluss führt, ist eingestürzt. Cassandra könnte einfach über den Fluss fliegen, dafür müssten allerdings erst ihre Flügel repariert werden. Da Cassandra ein kleines Gewichtsproblem hat, reicht die Kraft ihrer Flügel nicht aus, um ihr eigenes Gewicht und das von Kuni zu tragen. Dieser weiß allerdings von einem alten Gnomengang durch die Felsen. Dieser ist jedoch mit einem Schloss verschlossen und öffnet sich erst, wenn der Schlüssel gefunden und ins das Schloss gesteckt wurde. Da Cassandra fliegt und Kuni den Geheimgang nimmt, trennen sich kurz die Wege beider Protagonisten.

Kuni findet sich auf einer Wiese wieder, auf der Blumen blühen und seltsame Laute aus einem nahen Busch hörbar sind. Bei nähere Betrachtung findet er dort einen Elf, der seine Pflicht-Schicht im Harfe-Spielen absolviert und den Posten auf keinen Fall verlassen darf. Leider hat der Elf schrecklichen Durst. Kuni findet einen Blütenkelch und kann darin Tau von den Blumen der Wiese einsammeln und den Kelch zum Elf bringen. Aus Dankbarkeit schenkt dieser ihm eine Saite der Harfe und erklärt ihm den Weg zum Eingang zum Zauberwald.

Währenddessen hat Cassandra den Fluss überquert und findet ein Stück Kuchen und sammelt dieses als Wegzehrung für die lange Reise ein. Außerdem findet sie eine Brechstange mit dessen Hilfe sie eine Schatzkiste aufbrechen kann. Darin findet sich kein Schatz, sondern nur eine kaputte Laute.

Vor dem Zauberwald treffen Cassandra und Kuni wieder zusammen. Allerdings gelingt es ihnen erst nicht, den Wächter des Waldweges zu überzeugen sie in den Wald zu lassen. Beide besitzen kein Gold um den Wegzoll zu zahlen. Bei einem Stückchen Kuchen, kommt der Wächter ins Plaudern und erzählt, dass er eigentlich Wandermusiker war, bis ihn Räuber überfallen und seine Laute gestohlen haben, seitdem bewache er den Eingang und verlange Zoll, um sich irgendwann wieder ein Instrument leisten zu können. Cassandra versucht es mit der kaputten Laute, allerdings fehlt ihr eine Saite und daher will der Wächter sie nicht haben. Glücklicherweise ist Kuni in dem Besitz einer Saite. Beide Objekte vereint ergeben ein funktionierendes Instrument, welches der Wächter dankend annimmt und sich sofort auf und davon macht um wieder dem Minnesang zu frönen. Damit ist der Weg in den Zauberwald für Kuni und Cassandra frei. Hier endet der Teil der Handlung, der durch den Prototypen abgedeckt wird. Im weiteren Verlauf dieser Arbeit wird für das Spiel von nun an auch der Arbeitstitel „Zauberwald“ verwendet.

3.3 UI-Design der Clients

3.3.1 Überblick

Für die visuelle Rezeption des Spiels wird das Fabula-Plugin *PygameUserInterface* verwendet, eine Benutzerschnittstelle die auf *Pygame*[4] basiert. Das Plugin ist, wie bereits erwähnt, für die Umsetzung eines Adventure-Spiels geeignet erzeugt eine rein visuelle Repräsentation des Spiels. In diesem Unterkapitel werden die einzelnen funktionalen Bestandteile der Schnittstelle beschrieben und für jede ihrer Funktion eine auditive beziehungsweise blinden-gerechte Entsprechung konzipiert; auf die technische Umsetzung der auditiven Schnittstelle wird näher in Kapitel 4 eingegangen.

3.3.2 Eingabe

Benutzereingaben werden im PygameUserInterface über die Maus und die Tastatur entgegen genommen. Die hauptsächliche Steuerung im Spiel erfolgt allerdings mit der Maus. Die Verwendung einer Maus ist nur in wenigen Fällen blinden-gerecht und kann im Audio-Client nicht als Haupt-Eingabegerät für die Spielsteuerung eingesetzt werden. Stattdessen bietet sich die Steuerung über die Tastatur an. Die einzelne Tastenbelegungen werden bei den Anwendungsfällen beschrieben, sollten jedoch konfigurierbar angelegt werden, so dass der Spieler bei Bedarf die Belegung ändern kann. Im Audio-Client ist zusätzlich ein Hilfe-Menü geplant, in welchem die Zuordnung von Tasten und Funktion hinterlegt sind und das über das Drücken von „F1“ erreicht werden kann.

3.3.3 Grafische und akustische Widgets

Alle textuellen Informationen müssen für den Audio-Client in Sprache umgewandelt werden, dies wird mit Hilfe einer Text-To-Speech-Engine realisiert. Dabei werden alle Sprachinhalte mit der gleichen Stimme vorgelesen. Es wäre wünschenswert, wenn die Wahl dieser Stimme dem Spieler überlassen werden könnte. Da das Spiel für zwei Spieler ausgelegt ist, müssen die Benutzerschnittstellen Eingabefelder enthalten, damit Netzwerkkonfigurationen, wie zum Beispiel die IP-Adresse, eingegeben werden können. Im PygameUserInterface kann das über ein gewöhnliches Textfeld gelöst werden, das Tastatureingaben entgegennimmt und die entsprechenden Buchstaben dem Text der Eingabe hinzufügt und dies auch anzeigt. Im Audioclient hingegen muss ein auditives Textfeld

[4] Pygame ist eine Sammlung von Python-Modulen aufbauend auf die SDL Bibliotheken für die Entwicklung von Spielen (vgl. [24]).

erstellt werden. Ein solches Textfeld besitzt ein bezeichnendes Label, dass vorgelesen wird, wenn das Eingabefeld in den aktiven Modus übergeht. Während die Eingabe erfolgt, wird jede gedrückte Taste einzeln vorgelesen und das entsprechende Zeichen dem Eingabetext angefügt. Nach Beendigung der Eingabe über eine Bestätigungstaste wird der vollständige Eingabetext einmal vorgelesen. Im visuellen Client werden Aktionen durch das Betätigen von Buttons angestoßen. Da im Audio-Client nicht mehrere Auswahlmöglichkeiten auf einmal dargestellt werden können, werden Button-Funktionen in ein auditives Listenmenü verschoben. Audiolisten lesen analog zum Texteingabefeld bei Aktivierung einen bezeichnenden Namen vor, gefolgt von dem ersten Eintrag der Liste. Über die vertikalen Pfeiltasten können die Einträge durchblättert werden, der jeweils aktive Eintrag wird dabei vorgelesen. Mit der Pfeiltaste Rechts oder über die Returntaste wird ein Eintrag bestätigt und die entsprechende Aktion ausgeführt. Mit der Pfeiltaste Links kann eine Audioliste verlassen werden. Wenn beim Durchblättern der Liste der letzte Eintrag erreicht wurde und in die gleiche Richtung weitergeblättert wird, soll ein kurzer Hinweiston das Ende der Liste anzeigen und ohne weitere Eingabe des Benutzers zum ersten Eintrag der Liste gesprungen und dieser vorgelesen werden.

Bei der Verwendung mehrerer Audiolisten soll die Steuerung immer auf die gleiche Art über die gleichen Tasten erfolgen, um eine konsistente Menüführung zu gewährleisten. Wenn das eigentliche Spiel geladen wird, wird im PygameUserInterface das Anwendungsfenster schwarz überblendet und ein „Loading“-Schriftzug angezeigt. Analog dazu soll im Audio-Client der Beginn des Ladens über das Abspielen eines Laden-Sounds erfolgen. Auf die gleiche Art und Weise soll das Ende des Ladevorgangs erfolgen. Aus der visuellen Anzeige erschließt sich im PygameUserInterface, dass die virtuelle Welt betreten wurde. Um die gleiche Erkenntnis im Audio-Client zu ermöglichen werden nach dem Ladevorgang zum Raum passende Hintergrundgeräusche in einem Loop abgespielt. Diese Hintergrundgeräusche sollen immer vorhanden sein, so lange der Spieler sich in dem Raum befindet.

3.3.4 Navigation in der virtuellen Welt

Die virtuelle Welt ist in einzelne Räume unterteilt, welche wiederum aus gleichgroßen Feldern aufgebaut sind. Jedes Feld verfügt im PygameUserInterface über eine bildliche Darstellung, ebenso wie Objekte, Spieler und NPCs. Im Audio-Client dagegen besitzen Felder eine textuelle Beschreibung. Objekte, Spieler und NPCs verfügen über ein beschreibendes, natürliches Geräusch und eine aussagekräftige textuelle Benennung. Im Gegensatz zum visuellen Client kön-

nen diese Eigenschaften nicht permanent und gleichzeitig angezeigt werden, weil sich die unterschiedlichen akustischen Ausgaben einfach nur überlagern würden. Die akustische Darstellung von Feldern wird in Unterkapitel 3.3.9 gesondert beschrieben.

Das Abspielen von akustischen Beschreibungen von Objekten und Spielfiguren wird auf einen Teilausschnitt des virtuellen Raums begrenzt. Bei jeder Bewegung der Spielfigur, wird dieser Teilausschnitt aktualisiert. In Abbildung 1 ist dieser Teilauschnitt dargestellt. Das Feld, auf dem die Spielfigur sich befindet ist grau unterlegt, der hörbare Teilausschnitt der virtuellen Welt setzt sich aus diesem und den grün unterlegten, acht umgebenden Feldern zusammen. Befindet sich ein Objekt oder eine Spielfigur in diesem Ausschnitt, wird das natürliche beschreibende Geräusch ausgegeben. Befinden sich mehrere Objekte in diesem Ausschnitt, werden deren Geräusche gleichzeitig abgespielt. Um dennoch eine räumliche Zuordnung von Objekt und Feld zu ermöglichen, ist jedem Feld eine unterschiedliche Lautstärke für die Wiedergabe der Geräusche zugeordnet. Dabei setzt sich die Gesamtlautstärke aus zwei Kanälen zusammen, einem linken und einem rechten. Die Lautstärke ist für beide Kanäle am höchsten für das Feld auf dem sich die Spielfigur befindet. Das Feld links davon spielt Sounds nur auf dem linken Kanal ab, das Feld rechts davon nur auf dem rechten. Die obere Reihe an Feldern wird als vor dem Spieler liegend gewertet und hat damit insgesamt eine höhere Lautstärke als die untere Reihe, die als hinter dem Spieler liegend eingeordnet werden kann. Da die Steuerung nicht für eine Ego-Perspektive, sondern für eine Vogelperspektive konzipiert ist, gibt es im Prinzip keine Blickrichtung, das heißt die Zuordnung von Lautstärke und Feld bleibt konstant.

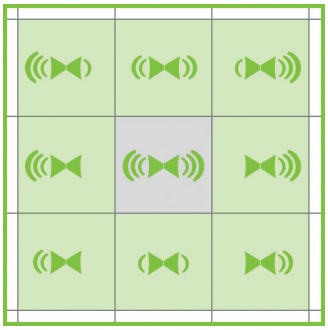

Abbildung 1: *Lautstärke Zuordnung für umgebende Felder der Spielfigur*

Im Audio-Client erfolgt die Steuerung über die WASD-Tasten, wobei sich die Spielfigur je Tastendruck um ein Feld nach oben (W-Taste), links (A-Taste),

unten (S-Taste) oder rechts (D-Taste) bewegt. Um Rückmeldung über die Bewegung zu erhalten, werden Schrittgeräusche abgespielt. Die Steuerung der Bewegung der Spielfigur erfolgt im visuellen Client mit Hilfe der Maus: Der Benutzer klickt auf eine Position, die Spielfigur bewegt sich auf das Feldauf dem der Klick ausgelöst wurde, sofern das Feld nicht blockiert ist. Dabei wird die bildliche Darstellung auf die entsprechende Position bewegt. Versucht der Spieler sich an einen Ort zu Bewegung, der außerhalb des Raumes liegt, blockiert oder nicht begehbar ist, wird das im PygameUserInterface durch ein kurzes Aufblinken des Anwendungsfensters dargestellt. Im Audio-Client wird ein Earcon abgespielt, dass die Information tragen soll, dass der Spieler auf ein Hindernis gestoßen ist und seine zuletzt eingegebene Bewegung nicht erfolgen kann. Im PygameUserInterface werden die Haupt-Spielfiguren Kuni und Cassandra mit einfacher Animation versehen, die vier Ansichten pro Spielfigur bereit stellt. Bewegt eine Spielfigur sich nach unten, ist die Vorderansicht zu sehen, bewegt sie sich nach oben, die Hinteransicht. Geht die Figur zur Seite wird die entsprechende Seitenansicht angezeigt. In Abbildung 2 ist zum Beispiel die Hinteransicht von Cassandra zu sehen.

3.3.5 Inventar

Das Inventar ist der Ort, in dem Gegenstände, welche der Spieler eingesammelt hat, aufbewahrt werden. Die Gegenstände können mit anderen Objekten in der virtuellen Welt benutzt werden. Im PygameUserInterface wird das Inventar visuell durch eine gelb umrandete, hellbraune Fläche dargestellt, die immer Teil des Anwendungsfenster ist, zu sehen im unteren Bereich von Abbildung 2. Gegenstände können, sofern sie beweglich sind und sich in der Nähe der Spielfigur befinden, einfach mit Hilfe von Drag and Drop auf diese Fläche gezogen werden. Dann werden sie automatisch ins Inventar verschoben und dort angezeigt. Die bildliche Darstellung wird aus dem Raum entfernt.

In der Abbildung ist ebenfalls der Vorgang des Benutzen eines Objektes aus dem Inventar mit einem Objekt in der virtuellen Welt illustriert. Dieser Vorgang wird ebenfalls per Drag and Drop vorgenommen. Während des ziehen des Gegenstandes verbleibt ein Abbild davon im Inventar, während eine Kopie davon neben dem Mauszeiger angezeigt wird. Bewegt sich dabei der Mauszeiger über Objekte in der virtuellen Welt, wird die Darstellung des Objekt visuell hervorgehoben, in dem die entsprechende Grafik aufgehellt angezeigt wird. Im Audio-Client wird das Drag und Drop zum Aufheben eines Gegenstandes durch eine weitere Interaktion im Interaktionsmenü realisiert. Für die Repräsentation des Inventars ist eine Audioliste vorgesehen, die sich über die Taste „q“ öffnen

Abbildung 2: *Screenshot: Drag and Drop im PygameUserInterface*

lässt. Wenn sich im Inventar Objekte befinden wird ein Sound abgespielt, der das Öffnen des Inventars signalisiert. Der Sound besteht aus einem natürlichen Geräusch (Aufnahme vom Aufschlagen eines Buchs). Wenn das Inventar keine Objekte enthält wird ein sprachlicher Sound abgespielt, der die Information abspielt. Die Audioliste kann wie beschrieben durchblättert werden. Pro Objekt im Inventar gibt es einen Listeneintrag, der die textuelle Repräsentation des Objektes enthält. Wird ein Eintrag ausgewählt, wird dem Spieler eine weitere Beschreibung des Objekts vorgelesen. Anschließend befindet er sich wieder in der Audioliste, die er über die Pfeiltaste Links verlassen kann. Während der Spieler im auditiven Inventar navigiert, sollen die Hintergrundgeräusche des Raums pausiert werden. So wird dem Spieler signalisiert, dass er sich in einem Menü befindet und nicht im Raum, was bedeutet, dass er auch keine Eingaben zur Bewegung eingeben kann.

3.3.6 Interaktionsmenü

Im Spiel sind die folgenden Arten von Interaktionn möglich: Ansehen, Ansprechen, Manipulieren, Aufheben von Gegenständen und Benutzen von Gegenständen mit anderen Gegenständen. Im PygameUserInterface sind die letzten beiden Interaktionen, wie im vorangegangenen Kapitel beschrieben, per Drag and Drop realisiert, wenn sich die Objekte in den umliegenden Feldern der Spielerpositi-

on befinden. Die drei anderen Interaktionen können über ein visuelles Interaktionsmenü aufgerufen werden. In Abbildung 3 ist dieses Interaktionsmenü zu

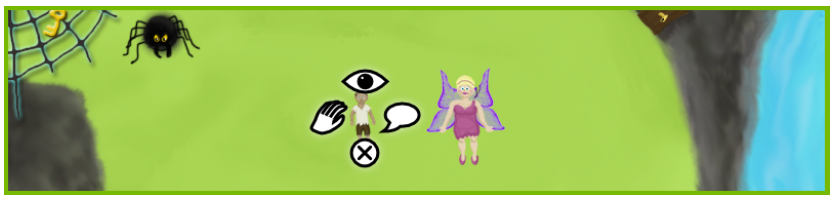

Abbildung 3: *Screenshot: Interaktionsmenü im PygameUserInterface*

sehen nachdem der Spieler von Fee Cassandra einen Rechtsklick auf Gnom Kuni ausgeführt hat. Dieses Menü erscheint immer, wenn der Benutzer auf einem Gegenstand, Spieler oder NPC einen Rechtsklick mit der Maus ausführt, unabhängig davon, ob sich der Gegenstand in der direkten Umgebung der Spielfigur befindet. Das visuelle Menü enthält Symbole für die einzelnen Interaktionen. Über einen Klick mit der linken Maustaste auf eines der Symbole wird die entsprechende Aktion ausgeführt. Im Audio-Client dagegen erfolgt der Zugang zu jeder der fünf möglichen Interaktions-Arten gleich, über ein Interaktionsmenü in Form einer Audioliste. Dabei ist bei dieser Liste besonders, dass die einzelnen Einträge nicht mit Hilfe der Text-To-Speech-Engine vorgelesen werden, sondern Sounds mit aufgenommener Sprache enthalten. Die Anzahl der Interaktionen verändert sich über das Spiel hinweg nicht, die Anzahl der Einträge und die Art der Einträge der Audioliste für das Inventar beispielsweise schon. Über die Verwendung von aufgenommenen Sounds statt Sprachsynthese erfolgt eine klare Trennung von statischen und generischen Inhalten. Wird eine Interaktionsart ausgewählt, muss der Spieler danach ein Zielobjekt oder eine Zielperson auswählen. Mit Ausnahme der Interaktion „Benutzen von Gegenständen mit anderen Gegenständen“, bei dem zuerst der zu benutzende Gegenstand ausgewählt wird und danach das Zielobjekt, beziehungsweise die Zielperson, dargestellt in Tabelle 3. Die Auswahl von Gegenstand und Zielobjekt erfolgen wieder über Audiolisten.

Die Einträge enthalten hier – analog zur Audioliste des Inventars – die textuelle Repräsentation der Objekte, Spieler oder NPCs. Inhaltlich sollen in der Liste nur Objekte enthalten sein die den Bedingungen der gewählten Interaktion entsprechen. Wenn also beispielsweise die Audioliste für die Interaktion „Aufheben“ geöffnet wird, soll diese Liste nur Einträge von Objekten enthalten, die sich auf den umliegenden Feldern der Spielfigur befindet und beweglich sind. Die Interaktionen „Ansehen“ und „Ansprechen“ beziehen sich auf alle Objekte im

Ansehen		
Ansprechen		
Aufheben		
Benutzen mit →	Auswahl Gegenstand →	Auswahl Ziel
Manipulieren		
Beenden		

Tabelle 3: *Menüfluss Audio-Client: Interaktion für „Benutzen mit“*

gesamten Raum, mit der Einschränkung, dass für die Interaktion „Ansprechen“ nur andere Spieler oder NPCs zur Verfügung stehen. Die Listeneinträge für die Auswahl von Objekten für die Interaktion „Manipulieren“ und für den zu benutzenden Gegenstand für die Interaktion „Gegenstand benutzen mit Ziel“ sollen sich nur aus Gegenständen, die sich auf den umliegenden Feldern befinden und Gegenständen aus dem Inventar des Spielers zusammensetzen. Die Audioliste zu Wahl eines Zielobjektes oder der Zielfigur für die Interaktion „Gegenstand benutzen mit Ziel“ soll ebenfalls nur Objekte in der nahen Umgebung und des Inventars mit einschließen. Alle diese Audiolisten müssen sowohl mit einem Menüsound, der beim Öffnen der Liste abgespielt wird, sowie einem Sound der abgespielt wird, falls die Liste leer ist, versehen werden.

Da bei diesem Design mehrere Audiolisten ineinander verschachtelt sein können, muss das Verhalten beim Verlassen der Liste über die Verlassen-Taste definiert werden. Das gewünschte Verhalten lässt sich am Besten an dem konkreten Beispiel aus Tabelle 3 erläutern. Befindet sich der Spieler in der Ziel-Auswahl für die Interaktion „Benutzen mit“ und betätigt die Zurücktaste, soll der Spieler automatisch ins die Audioliste zur Auswahl des Gegenstandes zurück kehren. Diese Liste soll dann mit den aktuellen Gegenständen gefüllt werden, den Menü-Sound abspielen und den ersten Listeneintrag vorlesen. Falls keine Objekte mehr bereit stehen, soll direkt das Interaktionsmenü wieder aufgerufen werden und dessen Menüsound und erster Menüeintrag abgespielt werden. Während der Spieler sich im Interaktionsmenü oder einem der Submenüs befindet, soll analog zur Audioliste des Inventars keine Bewegung im Raum möglich sein, folglich werden die Hintergrundgeräusche für den Raum auch nicht abgespielt.

3.3.7 Auswahl von optionalen Sätzen

Wenn der Spieler eine andere Spielfigur oder NPC anspricht, sollen die Benutzerschnittstellen zwei unterschiedliche Fälle als Reaktion darauf implementieren. Zum einen die simple Ausgabe eines Satzes oder aber eine Auswahl, welche die

Wahl zwischen mehreren möglichen Sätzen ermöglicht und die Eingabe der Auswahl des Spielers erfordert. Im PygameUserInterface wird dieses Auswahl über eine visuelle Liste angezeigt, dessen einzelne Sätze über einen Mausklick angewählt werden können, dargestellt in Abbildung 4. Über den Ok-Button kann der Benutzer seine Auswahl bestätigten.

Abbildung 4: *Screenshot: Satzauswahl im PygameUserInterface*

Im Audio-Client wird für das Auswahlelement wieder eine Audioliste verwendet. Die Einträge der Liste setzen sich aus den optionalen Sätzen zusammen und werden von der Text-To-Speech-Engine vorgelesen, wenn der Spieler die Liste durchblättert. Da eine Auswahl eines Satzes erfolgen muss, kann diese Audioliste nicht über die Pfeiltaste Links verlassen werden. Die Wahrnehmung von einem Sprachereignis wird unter anderem im nächsten Unterkapitel beschrieben.

3.3.8 Wahrnehmung von Ereignissen

Die Interaktionen die durch das Menü ausgewählt werden führen zu unterschiedlichen Aktionen. Die Darstellung dieser Aktionen wird im Folgenden beschrieben. Wenn sich ein Spieler ein Objekt ansieht, wird eine Information über das Objekt an die Clients weitergeleitet. Im PygameUserInterface erscheint in der oberen Bildhälfte eine graue Box mit der textuellen Darstellung der Information. Auf eine ähnlich Weise werden die Gesprächs-Ereignisse dargestellt. Im

unteren Teil des Anwendungsfenster erscheint eine Box mit dem gesprochenen Text. Dabei ist dem Text vorangestellt, welche Figur spricht. Im Audio-Client werden beide Ereignisse durch das Vorlesen der textuellen Information durch die Text-To-Speech-Engine ausgegeben. Analog zum visuellen Client wird bei Gesprächsereignisse dem Text, der Name des Sprechers vorangestellt.

Im visuellen Client muss das Aufheben und Aufnehmen eines Objektes ins Inventar nicht eigenständig behandelt werden, da das durch das Drag and Drop und erscheinen im Inventar deutlich wird. Im Audio-Client ist das Inventar kein dauerhaft präsenter Bestandteil der Schnittstelle. Dem Benutzer muss folglich, ähnlich wie beim Ansehen eines Objektes, eine sprachliche Rückmeldung gegeben werden, dass ein Objekt ins Inventar aufgenommen wurde.

3.3.9 Orientierungshilfen im Audioclient

Im Gegensatz zum visuellen Client kann der Spieler im Audio-Client nicht ohne gezielte Eingabe Informationen über die virtuelle Welt erhalten. Nur die Objekte, welche sich in seiner Umgebung befinden werden durch den Audio-Client automatisch dargestellt. Damit der Spieler sich auch einen Gesamteindruck über den Raum verschaffen kann wird allen Felder des Raums ein Tupel bestehend aus einer x- und y-Koordinate gegeben. Dabei soll für jeden Raum einzeln ein lokales Koordinaten System erstellt werden, so dass jeder Raum das Feld (0, 0) enthält. Eine Orientierungshilfe ist das Abfragen der eigenen Position als Koordinaten-Tupel. Dies erfolgt beim Betätigen der x-Taste, auf das das Vorlesen der Tupel erfolgt. Eine weitere Hilfsfunktion erreicht der Benutzer über die y-Taste. Es öffnet sich eine Audioliste, die alle Objekte des Raumes enthält. Wenn ein Objekt in der Liste ausgewählt wird, wird ebenfalls das entsprechende Koordinatentupel vorgelesen.

4 Implementierung

4.1 Ausgangszustand Fabula

4.1.1 Architektur Fabula

Fabula ist eine plattformunabhängige Open-Source Game-Engine, die in Python (Version 3.1) implementiert ist und sich zur Zeit im Entwicklungsstadium einer Alpha-Version[5] befindet. Fabula basiert auf einer Client-Server Architektur, dabei verwaltet der Server den Spielzustand und der Client besorgt die Repräsentation dieses Spielzustands in Form einer Benutzerschnittstelle, welche auch für die Entgegennahme und Weiterleitung von Benutzereingaben verantwortlich ist. Abbildung 5 stellt die Architektur und Komponenten von Fabula dar. Die Kommunikation von Server und Client kann lokal oder über IP-Netzwerke

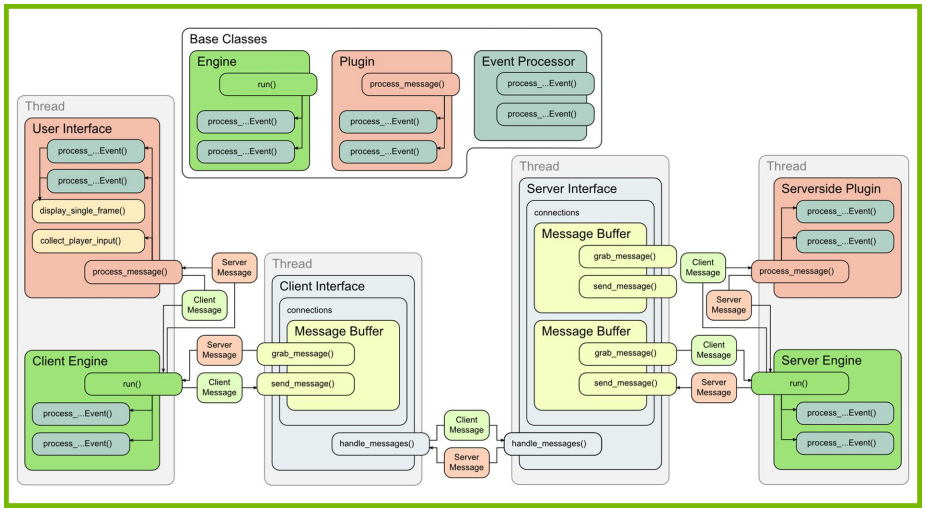

Abbildung 5: *Architektur Fabula und interne Kommunikation*[6]

erfolgen. Dabei kommunizieren Server und Client über ein *Event*-basiertes Protokoll. Mehrere Events, die parallel verarbeitet werden sollen, können in einer *Message* zusammengefasst übertragen werden. Es besteht demnach die Möglichkeit Spiele sowohl mit einem Einzelspieler- als auch in einem Mehrspielermodus zu implementieren. Dabei kann bei letzterer Variante über ein Netzwerk zusammen gespielt werden. Abgesehen von der generellen Eignung für das Genre erfüllt Fabula demnach auch die Ansprüche der Mehrspieler-Unterstützung aus der Konzeption des multimodal zugänglichen Spiels.

[5] Grundlage dieser Arbeit ist der aktuelle, unveröffentlichte Entwicklungsstand von Fabula, verfügbar unter `https://bitbucket.org/flberger/fabula/`, Commit 7d99c44, Bookmark audioclient.

[6] Original-Grafik von Florian Berger, 2011

4.1.2 Abstraktion der Spielwelt

Fabula besitzt ein abstraktes Modell der Spielwelt, welches unabhängig von der Repräsentation besteht. Es ist Aufgabe des jeweiligen Clients diese Abstraktion in einer Schnittstelle für den Benutzer abzubilden. Die Abstraktion der Spielwelt basiert auf die Unterteilung der Welt in räumliche Teilabschnitte. Ein solcher Teilabschnitt heißt *Room* und ist in gleichgroße Felder unterteilt; ein Feld wird als *Tile* bezeichnet und kann begehbar sein oder nicht, was über die Typisierung als *FLOOR* oder *OBSTACLE* erreicht wird. In einem Room können mehrere Objekte existieren. Dabei wird ein Objekt als *Entity* bezeichnet und kann entweder vom Typ Gengenstand (*Item*), Spieler (*Player*) oder NPC (*NPC*) sein. Neben dem Typ kann eine Entity als beweglich oder unbeweglich definiert werden. Außerdem kann festgelegt werden, ob die Entity passierbar ist, oder nicht. Jede Entity ist räumlich einem Tile zugeordnet, ist die Entity passierbar, kann eine andere Entity dieses Tile betreten, anderfalls nicht.

Die von Fabula implementierte Abstraktion ist sehr gut geeignet für die Umsetzung des in der Konzeption gewählten Genres „Adventure“. Abbildung 6 illustriert die Abstraktion eines Rooms und der sich darin befindenden Entities.

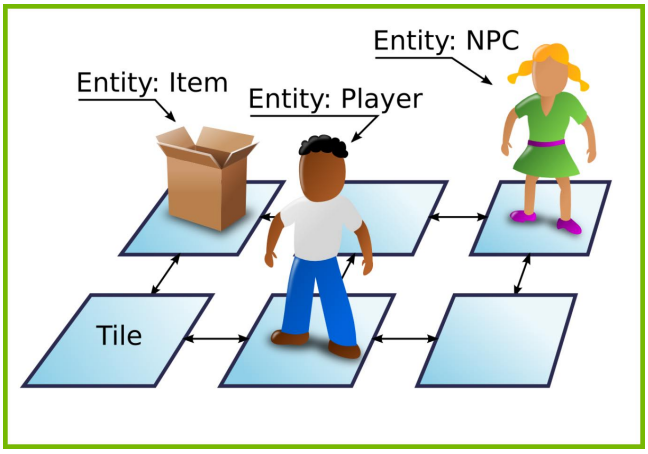

Abbildung 6: *Abstraktion der Spielwelt*[7]

Jede Room-Instanz verfügt über ein Rack, in dem alle Entities, welche aus einem Room entfernt wurden, abgelegt werden. Das Rack ermöglicht die Zuordnung von Besitzern zu Entities.

[7] Original-Grafik von Florian Berger [5]

4.1.3 Fabula Plugins

Fabula stellt ein Plugin-System bereit mit dessen Hilfe die Implementierung erleichtert wird. So gibt es ein *UserInterface*-Plugin, von dem auch der derzeitige visuelle Client *PygameUserInterface* abgeleitet ist. Diese Schnittstelle basiert auf *Pygame* (Version 1.9.1), mit dessen Hilfe ein Anwendungsfenster, das Rendering des Spielzustandes und die Entgegennahme von Benutzereingaben ermöglicht wird. Des Weiteren existiert das serverseitge Plugin *DefaultGame*, welches ein Grundgerüst für ein Fabula Spiel bereitstellt und grundlegende Funktionen, wie die Verarbeitung der Events bereits implementiert. Das Plugin-System ist sehr gut geeignet, um Erweiterungen für die Game-Engine zu entwickeln. Daher ist es möglich, ohne Anpassung der vorhandenen Struktur der Engine, ein *AudioUserInterface* als Plugin umzusetzen. Die vorhandenen Plugins für die visuelle Benutzerschnittstelle und die serverseitige Implementierung des DefaultGames sind eine gute Entwicklungsgrundlage, um schnell ein lauffähiges Spiel zu erstellen, so kann mehr Zeit auf die Entwicklung der Audio-Schnittstelle verwendet werden.

4.1.4 Events

Über *Events* kann die Spielwelt beeinflusst werden. Dabei leitet sich ein Großteil der Events entweder von der Klasse *AttemptEvent* oder der Klasse *ConfirmEvent* ab. Die Events dieser beiden Klassen werden benutzt um Benutzeraktionen abzubilden und implementieren dabei Möglichkeiten der Interaktion eines klassischen Adventure-Spiels, was sich sehr gut mit der Konzeption des multimodal zugänglichen Spiels deckt. Im Client werden die Benutzereingaben in *Attempt-*

Aktion	AttemptEvent	ConfirmEvent
Bewegen	TriesToMoveEvent	MovesToEvent
Ansehen	TriesToLookAtEvent	PerceptionEvent
Aufheben	TriesToPickUpEvent	PicksUpEvent
Benutzen	TriesToDropEvent	DropsEvent
Manipulieren	TriesToManipulateEvent	ManipulatesEvent
Ansprechen	TriesToTalkToEvent	CanSpeakEvent

Tabelle 4: *AttemptEvents und mögliche ConfirmEvents*

Events übersetzt, die an den Server übermittelt werden. Das Server-Plugin enthält die Spiellogik und prüft anhand dieser die Anfrage und sendet entweder ein Event der Klasse ConfirmEvent oder ein *AttemptFailedEvent* an den Client zu-

rück. In Tabelle 4 sind die möglichen Aktionen mit den dazugehörigen Attempt-Events aufgeführt, sowie die entsprechenden ConfirmEvents die vom Server bei erfolgreicher Prüfung an den Client zurück gesendet werden können. Dabei ist diese Zuordnung nur beispielhaft, weil die Auswahl eines passenden Confirm-Events in vielen Fällen von der Spiellogik abhängt. So kann zum Beispiel auf ein *TriesToTalkToEvent* auch mit einem *SaysEvent* geantwortet werden. Nur wenn für ein bestimmtes AttemptEvent keine adäquate Antwort implementiert ist, sendet der Server automatisch ein AttemptFailedEvent an den Client. Der Client muss das Ergebnis einer Anfrage repräsentieren. Der Ablauf und die beteiligten Komponenten lässt sich unter zu Hilfenahme von Abbildung 5 nachvollziehen.

Neben den Events für die verschiedenen Aktionen gibt es auch die Klasse der Server-Events. Über diese können eine Room-Instanz und die darin befindlichen Entities und Tiles direkt manipuliert werden. So können einem Room beispielsweise Entities über ein *SpawnEvent* hinzugefügt werden und über ein *DeleteEvent* gelöscht werden. Außerdem können Tiles über ein *ChangeMapElementEvent* verändert oder ersetzt werden. Server-Events werden auch benötigt, wenn ein Client sich zum Server verbindet. Mit der Anmeldung des Clients wird ein *InitEvent* zum Server übermittelt, der dem Client daraufhin mit einem *ServerParametersEvent* und *EnterRoomEvent* antwortet und für den Room über ChangeMapElementEvents die Tiles des Rooms und über SpawnEvents die entsprechenden Entities übermittelt. Auch hier ist es die Aufgabe des Clients auf diese Events mit einer geeigneten Repräsentation für den Benutzer zu reagieren.

4.1.5 Asset-Manager

Fabula stellt einen *Asset-Manager* bereit, der das Laden und Cachen von Assets aus Dateien übernimmt. Ein lokaler Dateipfad oder eine Netzwerk-Adresse als URI kann als Pfad zum Laden der Ressource angegeben werden. Der Asset-Manager erstellt aus der angegebenen Quelle ein Datei-ähnliches Objekt, das dem *AssetsDictionary* einer Entity unter einem Schlüssel, der aus dem MIME-Type des Dateiformats des Assets besteht, angehängt werden kann. So wird die Verwendung der Ressource an ein Objekt gebunden und muss nicht bei jedem Zugriff neu geladen werden. Allerdings kann eine Enity durch diese Implementierung nur ein einziges Asset pro Dateiformat besitzen. In Tabelle 5 werden alle Dateiformate und ihre entsprechenden MIME-Types dargestellt, die momentan in Fabula als Asset verwendet werden können.

Dateiformat	MIME-Type
mp3	audio/mpeg
ogg	audio/ogg
wav	audio/vnd.wave
gif	image/gif
jpg	image/jpeg
jpeg	image/jpeg
png	image/png
txt	text/plain

Tabelle 5: *Dateiformate und MIME-Types*

4.2 Vorgehensweise

4.2.1 Erstellung der Raumpläne

In Fabula besteht die Möglichkeit einen Room mit allen enthaltenen Tiles und Entities über eine besonders formatierte Text-Datei zu definieren, welche die Dateiendung *.floorplan* besitzt. Jedes Tile eines Rooms wird in einer Zeile dieser Datei definiert. Dabei werden dem Tile die Position als Tupel aus x- und y-Koordinate übergeben, sowie der Tile-Typ und die URI für die Assets des Tiles. Befindet sich auf dem Tile eine Entity, wird diese in der gleichen Zeile definiert, separiert mit einem Tab und den entsprechenden Daten der Entity. Dahinter können weitere Entities folgen, wiederum mit einem Tab von der vorherigen Definition getrennt. Der erste Schritt der Implementierung ist die Erstellung der *Floorplans* für die vier Room-Instanzen von Zauberwald: *default.floorplan* für den Startraum, *room_cassandra.floorplan* für den Raum, der nur von Cassandra betreten werden kann, *room_kuni.floorplan* für den entsprechenden Raum für Kuni und *room_entry.floorplan* für den Raum in dem Kuni und Cassandra sich wieder begegnen und sich der Eingang zum Zauberwald befindet. Das Server-Plugin kann eine Floorplan Datei lesen und daraus einen Room erstellen. Über die Floorplans erfolgt die Definition der Spielwelt Abstraktion von Zauberwald. Der nächste Schritt ist die Implementierung der Spiellogik in Form eines Server-Plugins, damit die Basis für die Entwicklung der Clients geschaffen ist.

4.2.2 Serverseitige Implementierung von Zauberwald

Als Grundlage für die Umsetzung des Server-Plugins wurde das Plugin *DefaultGame* verwendet. Die Klasse *ZWServerPlugin* leitet sich daher von *DefaultGame* ab und ist im Modul *zauberwald.py* definiert, welches vollständig im Anhang C.1

dieser Arbeit angehängt ist. Das *ZWServerPlugin* erweitert das *DefaultGame* um zwei Listen, die mögliche Sätze der beiden Hauptcharaktere Kuni und Cassandra verwalten. Die Sätze der Listen werden für die gemeinsame Konversation benutzt, wenn die Clients der Figuren Kuni und Cassandra miteinander sprechen. Dabei werden die Listen dynamisch aktualisiert, wenn zum Beispiel Kuni seinen Vorstellungssatz gesprochen hat, wird dieser aus den möglichen Sätzen Kunis entfernt und aus Cassandras Sätzen wird die Frage „Mit wem habe ich die Ehre?“ gelöscht. So wird die mögliche Unterhaltung zwischen Kuni und Cassandra dem Spielverlauf angepasst.

Die Methode `respond(self, event)` von *DefaultGame* wird in der Regel bei jeder Verarbeitung eines *AttemptEvents* mit Ausnahmen der *TriesToMoveEvents* aufgerufen. Um einen Großteil der Spiellogik zu implementieren wird diese Methode im *ZWServerPlugin* überschrieben. Das erfolgt indem das übermittelte Event mit allen möglichen Events der Spiellogik verglichen wird und bei Übereinstimmung eine Reaktion als passendes Event an die Server-Engine übergeben wird. Werden nicht nur ein Event, sondern mehrere Events für die Erstellung der passenden Antwort benötigt, werden diese Events in eine Message verpackt und in die Warteschlange für Messages anghängt. Das folgende Code-Beispiel zeigt den Vergleich mit einem *TriesToLookAtEvent* für Cassandra und dem Gegenstand Brecheisen und die entsprechende Antwort auf dieses Event, in diesem Fall ein *PerceptionEvent*.

```
elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                         target_identifier='pry')):

    event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                          perception='Ein Brecheisen, das könnte (...)')]
```

Einige Events sind fest mit einem Room verbunden, wie zum Beispiel das Aufheben von Gegenständen. Damit diese von der Spiellogik nur in Betracht gezogen werden, wenn das AttemptEvent in diesem Room erfolgt, wird die Spiellogik zusätzlich über Hilfmethoden in Room-spezifische Logik aufgeteilt. Diese Hilfsmethoden sind nach dem Schema `_respond_room_[roomname]` benannt. Die Methode `process_TriesToPickUpEvent` wird außerdem so überschrieben, dass das daraus resultierende Event auch über den Aufruf der `respond` Methode mit der Spiellogik abgeglichen wird.

Damit die Clients zwischen den Room-Instanzen wechseln können werden einige Tiles als Teleportierfelder behandelt. Um diese Funktion zu realisieren wird die `process_message` Methode des *DefaultGames* überschrieben und mit der Überprüfung von TriesToMoveEvents für die gewählten Tiles erweitert. Da-

bei unterscheiden sich die Teleportierfelder für Kuni und Cassandra, nur wenn der richtige Client versucht sich auf ein entsprechendes Tile zu bewegen wird der Client teleportiert. Dieser Vorgang setzt sich aus dem Löschen dieses Clients und eventuell verbliebenen Events aus dem alten Room und dem Laden des neuen Rooms zusammen. Das Laden eines Rooms ist in die Hilfsmethode `_load_room` ausgegliedert, welche die Client-ID, den Namen des neuen Rooms sowie eine *location* erwartet. Eine weitere Anpassung des DefaultGame erfolgt durch das Überschreiben der Methode `process_TriesToDropEvent`. Im DefaultGame können Items auf andere Items oder auf Tiles abgelegt werden. In Zauberwald ist das Ablegen von Items auf andere Items begrenzt.

Das Starten des Servers ist in das Modul `start_zauberwald_server.py` ausgelagert (siehe Anhang C.2). Wird das Skript aus einer Konsole heraus gestartet, kann eine IP-Adresse über den optionalen Parameter `-a` angegeben werden.

4.2.3 Implementierung des Zauberwald PygameUserInterface

Die visuelle Benutzerschnittstelle für Zauberwald ist in der Klasse `ZWPygameUI` implementiert. Sie leitet sich vom Plugin PygameUserInterface ab und benutzt dieses fast unverändert. Lediglich die Methode `get_connection_details` wird überschrieben, sodass beim Start des Clients eine veränderte GUI angezeigt wird. Anstelle eines Eingabefeldes für den *login*-Namen und des OK-Buttons werden zwei Buttons implementiert, um das Spiel entweder als Cassandra oder als Kuni zu starten. Die Klasse `ZWPygameUI` ist ebenfalls im Modul `zauberwald.py` enthalten. Auch das Starten des visuellen Clients ist in ein eigenes Skript ausgelagert (`start_zauberwald_pygame_client.py` – Anhang C.3). Mit der Implementierung der visuellen Schnittstelle, der vorangegangenen Abstraktion der Spielwelt, der Umsetzung des Server-Plugins und der Spiellogik kann *Zauberwald* bereits gespielt werden. Der nächste Schritt ist die Implementierung der auditiven Schnittstelle.

4.2.4 Grundlage des Audio-Clients

Der Schwerpunkt der Implementierung liegt in der Erweiterung von Fabula, um einen auditiven Zugang zu Spielen zu ermöglichen. Die Umsetzung des *AudioUserInterfaces* erfolgt als Fensteranwendung, obwohl eine auditive Schnittstelle auch ohne ein graphisches Fenster implementiert werden könnte. Bei einer reinen Konsolenanwendung muss der Benutzer jedoch Tastatureingaben über die Enter-Taste bestätigen. Die Verwendung von Pygame, das unter anderem auch das Fenster verwaltet, ermöglicht die direkte Verarbeitung des Anschla-

gens einzelner Tasten. Pygame bietet außerdem die Möglichkeit Sounddateien zu laden und die Ausgabe zu steuern. Das *Mixer*-Modul von Pygame stellt zu diesem Zweck Funktionen wie das Abspielen, Pausieren und Loopen bereit. Audio-Dateien können als Objekt des Typs *Sound* gespeichert werden und entweder direkt oder über die Verwendung eines *Channel*-Objekts abgespielt werden. Letzteres ermöglicht das Abspielen mit unterschiedlicher Lautstärke für den linken und den rechten Stereo-Kanal.

Die aktuell veröffentlichte – und bisher von Fabula benutzte – Pygame Version 1.9.1 von 2009 enthält einen Bug, der die Verwendung der Klasse *Channel* zum Abspielen von Sounds unmöglich macht (siehe [13]). Da diese Funktion jedoch für die Implementierung des Audio-Client benötigt wird, muss Pygame auf die Version 1.9.2a0[8] von Juli 2010, die einen Bugfix für das Problem enthält, upgegraded werden. Durch den Umstieg auf die neuere Version konnten bisher sowohl unter Windows als auch unter Linux keine Einschränkung des Funktionsumfangs anderer Fabula-Komponenten, welche auf Pygame aufbauen, festgestellt werden.

Für die Einbindung einer Text-To-Speech-Engine wird das Python-Modul *Pyttsx*[9] verwendet. Das Modul liefert eine plattformübergreifende Schnittstelle aus Python zu den gängigen Text-To-Speech-Engines herkömmlicher Plattformen. Unter Windows ist das *SAPI5*, unter Mac OS X der *NSSpeechSynthesizer* und unter Linux *eSpeak* und jede weitere Platform, die kompatibel mit *eSpeak* ist (vgl. [19]). Die aktuelle Pyttsx Version 1.1 ist für die Verwendung mit der Python Version 3.2 geeignet und musste für die Verwendung mit Fabula erst für Python 3.1 konvertiert werden.

Unter Windows 7 ist jedoch SAPI5 nicht mehr verfügbar, stattdessen gibt es den sogenannten *Narrator*, der zwar SAPI5-Stimmen verwendet, initial aber nur über eine englische Vorlesestimme verfügt (vgl. [16]) und die Installation einer kostenfreien deutschen Stimme aus Lizenzgründen nicht zulässt.[10] Allerdings kann auch unter Windows *eSpeak* für die Sprachsynthese verwendet werden. So konnte die Sprachsynthese aus Text auch unter Windows 7 realisiert werden. Weitere Schwierigkeiten, die sich bei der Verwendung der Text-To-Speech-Engine während der Entwicklung ergeben haben, werden gesondert in Kapitel 4.3 behandelt.

[8] Diese Version ist zwar auf dem FTP-Server von Pygame (`http://www.pygame.org/ftp/`) verfügbar, jedoch ist sie keine öffentliche Release-Version.

[9] `https://pypi.python.org/pypi/pyttsx`

[10] Windows 8 bietet laut [15] wieder die Sprachen Englisch (Vereinigte Staaten und Vereinigtes Königreich), Französisch, Deutsch, Japanisch, Koreanisch, Mandarin (vereinfachtes und traditionelles Chinesisch) und Spanisch für die Sprachausgabe mit *Narrator* an.

4.2.5 Implementierung von Audio-Widgets

Nachdem nun alle Voraussetzungen für die Sprachsynthese und das Abspielen von Sounds erfüllt sind, kann die Implementierung der auditiven Schnittstelle angegangen werden. Vor der Implementierung des eigentlichen auditiven Plugins ist die Umsetzung von Widgets sinnvoll. Dies sind auditive, generische Schnittstellen-Komponenten wie zum Beispiel eine auditive Menüliste. Dieser Schritt erfolgt in Anlehnung an die Verwendung von GUI-Widegts für grafische Benutzerschnittstellen.

Die Basis für eine auditive Menüliste wird von der Klasse `AudioMenuList` gebildet. Sie legt das Grundgerüst fest und enthält neben einer Liste für die Einträge einen *menu_sound*, einen *empty_sound*, einen *channel* und kann mit den Callback-Methoden `_on_entry_selected` und `_on_exit` versehen werden. Dabei sollte *menu_sound* einen aussagekräftigen, aber kurzen Menütitel enthalten, der beim Öffnen des Menüs abgespielt wird. Der *empty_sound* sollte eine Meldung enthalten, dass die Liste des Menüs leer ist. Der *channel* wird für das Abspielen der mit der Liste assoziierten Sounds benutzt. Die Funktion `try_to_open_menu_list` kann von außen aufgerufen werden, um eine Menüliste zu öffnen. In der Funktion wird geprüft, ob die Liste des Menüs leer ist. Ist das der Fall wird die Wiedergabe des *empty_sounds* gestartet und gewartet bis der Sound vollständig abgespielt wurde. Das folgende Code-Beispiel implementiert dieses Verhalten.

```
if self.empty_sound is not None:

    # Play back menus list empty sound
    #
    self.channel.play(self.empty_sound)

    # Wait till sound is complete
    #
    while self.channel.get_busy():
        pass
```

Das Ergebnis von `try_to_open_menu_list` ist im Fall einer leeren Liste *False*. Ist die Liste nicht leer, wird die Methode `open_menu_list` aufgerufen, die den *menu_sound* wiedergibt. Ableitende Menülisten sollten diese Funktion mit der Wiedergabe des ersten Listeneintrags erweitern. Über die Callback-Funktion `_on_entry_selected` kann der Liste ein Verhalten für den Fall, dass ein Eintrag selektiert wird zugeordnet werden. Die Methode `_on_exit` definiert das Verhalten, wenn eine Liste ohne Auswahl eines Eintrag verlassen wird.

Die Basis-Klasse implementiert die Verarbeitung von Tastatureingaben. Die

Methode `process_key_input` erwartet den *Key* eines *KEYDOWN-Events* von Pygame. Das folgende Codebeispiel zeigt einen Auschnitt der `process_key_input` Methode.

```
# Scrolling through entries with up and down arrow keys
#
if key in ENTRY_DOWN_KEYS or key in ENTRY_UP_KEYS:

    self.scroll_list(key)

# Call callback function for entry selected
#
elif (key in SELECT_ENTRY_KEYS
      and self.callback_on_entry_selected is not None):

    self.callback_on_entry_selected(self)
```

Die Definitionen der Tastenbelegung erfolgt außerhalb der Klasse unter Verwendung von Python-Listen. So können an einer zentralen Stelle alternative Tasten ergänzt oder die Zuordnung der Tasten geändert werden. Bei Betätigung der Tasten für das Durchblättern der Liste wird die Methode `scroll_list` aufgerufen, in welcher der interne Listenindex entsprechend der Eingabe aktualisiert wird. Allerdings kann der Eintrag noch nicht vertont werden, weil nicht klar ist, welche Objekte in der Liste enthalten sind. Dies geschieht erst in den konkreten Klassen `SoundMenuList`, `EntityMenuList` und `TextMenuList`, die von `AudioMenuList` abgeleitet sind.

Die Liste einer `SoundMenuList` besteht aus *Sounds*. Beim Durchblättern der Liste wird der Sound des aktuellen Eintrags über den *channel* der Liste abgespielt. Außerdem wird die Methode `open_menu_list` überschrieben und so erweitert, dass die Liste nach dem Abspielen des *menu_sounds* die Wiedergabe des ersten Sounds in der Liste auslöst.

Eine `EntityMenuList` verfügt über das zusätzliche Attribut *tts_engine*, über das der Liste eine Pyttsx-Engine zur Sprachsynthese übergeben wird. Ein Listeneintrag besteht aus einem Tupel, welches eine Entity und einen String enthält. Da diese Art von Listen verwendet wird, um Entitys für ein AttemptEvent auszuwählen, kann der String je nach Art des Events und der Herkunft der Entity unterschiedliche Informationen beinhalten. Beim Scrollen der Liste wird entweder die ID der Entity oder ihr *text/plain*-Asset – sofern vorhanden – von der Text-To-Speech-Engine vorgelesen, das folgende Code-Beispiel implementiert diese Funktion.

```
entity_text = entity.identifier
```

```
# Check if entity provides a name
#
if ('text/plain' in entity.assets.keys()
    and entity.assets['text/plain'].data is not None):

     entity_text = entity.assets['text/plain'].data[0]

# Let the TTS engine read the entry at index
#
self.tts_engine.say(entity_text)
    self.tts_engine.runAndWait()
```

Analog zur `SoundMenuList` wird auch in der `EntityMenuList` die Methode `open_menu_list` überschrieben und mit dem Vorlesen des ersten Menüeintrages erweitert, wenn die Wiedergabe des *menu_sounds* abgeschlossen ist.

Eine `TextMenuList` verfügt ebenfalls über das zusätzliche Attribut *tts_engine*. Die Liste erwartet einen String pro Listeneintrag. Wird die Liste gescrollt, liest die Text-To-Speech-Engine den Eintrag zum aktuellen Index der Liste vor. Auch diese Liste erweitert `open_menu_list`, so dass initial beim Öffnen der erste Listeneintrag vorgelesen wird.

Eine zweite Art von Audio-Widgets wird durch die Klasse `AudioTextField` implementiert. Sie verfügt über ein Attribut *label*, das einen String erwartet, analog zum *menu_sound* einer auditiven Menüliste. Außerdem benötigt das Textfeld einen Eingabetext und eine Text-To-Speech-Engine, die über das Attribut *tts_engine* zugewiesen werden kann. Der Eingabetext ist mit einem leeren String initialisiert und kann in den Fällen, in denen eine Vorausfüllung des Feldes sinnvoll ist, mit einem entsprechenden String initialisiert werden. Das Textfeld kann über die Methode `focus_audio_text_field` fokussiert werden und gibt in diesem Fall das *label* wieder. Die Methode `process_key_input` verarbeitet selbständig ankommende *KEYDOWN-Events* und bleibt so lange im Eingabemodus, bis die Eingabe über die Enter-Taste abgeschlossen ist. Während der Eingabe werden alle Events, die durch das Betätigen von Tasten erfolgen als Zeichen gelesen, sofern sie im UTF-8 Zeichensatz enthalten sind und dem Eingabetext angehängt. Außerdem liest die *tts_engine* das eingegebene Zeichen vor. Ausnahmen bilden nur die bereits erwähnte Enter-Taste und die Backspace-Taste, bei dessen Betätigung das letzte Zeichen des Eingabetextes gelöscht wird. Die Klassen aller Audio-Widgets befinden sich im Modul `audioui.py` (siehe Angang C.5).

4.2.6 Implementierung der Audio-Entity

In dem Modul `audioui.py` ist außerdem noch die Klasse `AudioEnity` implementiert die eine kleine Erweiterung einer normalen Fabula-Entity ist. Wenn

eine Entity über ein Asset des Typs *audio/ogg* verfügt und im Dateinamen der String „steps“ enthalten ist, wird der Typ der Entity auf diese Klasse umgeändert. In der Klasse wird die `process_MovesToEvent` Methode so umgeschrieben, dass bei jedem *MovesToEvent* der Sound der Entity abgespielt wird. Der Sound der Entity sollte folglich Schrittgeräusche oder ähnliches enthalten. Kuni, Cassandra und der Wächter des Eingangs zum Zauberwald erfüllen die Vorraussetzung und werden als `AudioEnity` behandelt. Der Wächter und Kuni besitzen Schrittgeräusche, während Cassandra ein Flügelschlagen wiedergibt. Mit Hilfe dieser Erweiterung werden im Spiel die Bewegungen von AudioEntities hörbar gemacht.

4.2.7 Implementierung des AudioUserInterface

Mit der Implementierung der Audio-Widgets und der AudioEntity wurden Komponenten entwickelt, die für die Verwendung im auditiven Plugin bereitstehen. Dessen Implementierung erfolgt ebenfalls im Modul `audioui.py` in der Klasse `AudioUserInterface`. Diese ist vom `UserInterface`-Plugin von Fabula abgeleitet und muss die Entgegennahme von Benutzereingaben, Events des Servers, sowie die akustische Repräsentation der Spielwelt realisieren. Dafür müssen die Methoden des `UserInterface`-Plugins sinnvoll überschrieben werden.

Bei der Initialisierung der Klasse wird eine Text-To-Speech-Engine sowie das Pygame-Mixer-Modul für die Wiedergabe von Sounds initialisiert. Das *AudioUserInterface* verfügt über verschiedene Sounds und Channels, sowie Audiomenülisten. Zusätzlich enthält es verschiedene Eingabezustände, und einen Stack für deren Verwaltung. Dies ist nötig, weil Widgets einer auditiven Schnittstelle anders als Widgets einer visuellen Schnittstelle nicht gleichzeitig nebeneinander angezeigt werden können. Weil die Bedienung der Widgets jedoch konsistent über die gleichen Tasten erfolgen soll, werden die Eingabezustände verwendet, um Benutzereingaben dem richtigen Bedienelement zuordnen zu können. Im Quellcode sind die Zustände unter dem Namen *Input State* zu finden. Die Anzahl der möglichen Input States des AudioUserInterfaces umfassen je einen Input State für jedes Listenmenü, sowie den Input State *IN_ROOM*, wenn der Benutzer kein Menü angewählt hat und sich frei in der Spielwelt bewegen kann. Außerdem gibt es eine Instanzvariable *input_state_stack* von Typ Python-Liste. Das letzte Element dieser Liste enthält immer den aktuellen Zustand des Clients und liegt damit zu oberst auf dem Stack.

Wird der Audio-Client gestartet, beginnt die eigentliche Repräsentation mit Hilfe des Audio-Clients. Initial erhält dieser eine Menüliste zur Eingabe des *logins*, Starten des Spiels und Beenden des Spiels. Die Menüliste ist mit dem Input

State *CONNECTION_DETAILS* assoziiert, der initial der einzige Zustand des *input_state_stacks* ist. Die Erstellung des Menüs wird durch das Überschreiben der Methode `get_connection_details` erreicht. Diese Methode kann vom System das optionale Flag *prompt_connector* erhalten – ist dieses Flag gesetzt, wird die Menüliste mit der Ansage der IP-Adresse und einem auditiven Textfeld zu Eingabe der IP-Adresse erweitert. Die Methode muss ein Tupel aus Benutzername und IP-Adresse zurückgeben. Wird keine IP-Adresse gesondert angegeben, erfolgt die Verwendung einer default Adresse. Wählt der Benutzer den Eintrag zum Starten des Spiels aus, wird der Client beim Server angemeldet, der die Initialisierung des Rooms beginnt und den Client über ein EnterRoomEvent darüber informiert.

Die Methode `process_EnterRoomEvent` reagiert auf dieses Event mit der Wiedergabe der Instanzvariable *loading_sound* vom Typ Sound. Anschließend müssen die vom Server übermittelten Spawn- und ChangeMapElementEvent für die Darstellung des Rooms verarbeiten werden. Dies geschieht in den überschriebenen Methoden `process_ChangeMapElementEvent` und `process_SpawnEvent`. In der zuletzt genannten wird die Darstellung einer Entity für den Audio-Client erstellt. Besitzt eine Entity Assets vom Typ *audio/ogg* und *text/plain* werden diese Assets mit Hilfe des Asset-Managers geladen und als *value* zum entsprechenden MIME-Key in das Asset-Dictionary der Entity gespeichert. Enthält die *URI* eines *audio/ogg* Assets den String „steps“ wird die Klasse der Entity zu `AudioEntity` geändert. Da die Methode nicht nur initial verwendet wird, wenn ein Room erstellt wird, sondern auch wenn eine Entity aus dem Inventar wieder in den Room gelangt, wird vor dem Laden und Speichern geprüft, ob das Asset-Dictionary nicht schon einen Wert (demzufolge eine Datei) zum übergebenen MIME-Key besitzt. Am Ende der Methode wird der Sound der Entity über den Aufruf von `_play_sound_surrounding_entity` wiedergeben, sofern sie in der Nähe des Spielers des Clients ist. Wenn die Entity vom Typ *PLAYER* ist und die Identität nicht mit der des Clients übereinstimmt, handelt es sich folglich um die Spielfigur eines anderen Spielers und es wird ein *PerceptionEvent* mit der Information generiert, dass sich ein anderer Spieler im Raum befindet.

Der Prozess des Ladens der Assets aus `process_SpawnEvent` läuft analog für Tiles in der Methode `process_ChangeMapElementEvent` ab. Es werden ebenfalls die Assets der Typen *audio/ogg* und *text/plain* – sofern vorhanden – geladen und unter dem entsprechenden MIME-Key in das Asset-Dictionary eines Tiles abgelegt. Die aktuelle Implementierung des Audio-Clients greift nicht auf die Assets direkt zu, sondern verwendet nur das Koordinaten Tupel, das die Position des Tiles entspricht. Da aber zukünftige Implementierungen oder Spiele eine

textuelle oder auditive Repräsentation eines Tiles verwenden könnten, wird die Bereitstellung dieser Assets vom Audio-Client übernommen.

Das RoomCompleteEvent kündigt an, dass alle Events zur Erstellung des Rooms erfolgt sind und dieser über die beiden zuletzt genannten Methoden im Client etabliert wurde. In der Methode `process_RoomCompleteEvent` wird daher nur noch der Sound *loading_complete* wiedergegeben, bevor der Zustand *IN_ROOM* dem *input_state_stack* zugewiesen wird und Benutzereingaben verarbeitet werden können. Der Eingabezustand *CONNECTION_DETAILS* wird dabei vom Stack gelöscht, da er nur für die Anmeldung des Clients verwendet wird.

Aus dem neuen Eingabezustand kann nun die Bewegung im Raum oder Aktionen über das Öffnen der Audiomenüs erfolgen. Der Audio-Client verfügt neben dem *connection_details_menu* über die in Tabelle 6 aufgeführten Menülisten. Die Erstellung der Menüs ist jeweils in die Hilfsmethoden `_set_up_[Name]` ge-

Menüliste Name	Funktion
interaction_sound_menu	Auswahl einer Interaktion
select_item_menu	Auswahl eines Gegenstandes für *use*
select_target_menu	Auswahl für das Ziel eines AttemptEvents
select_sentence_menu	Auswahl eines Satzes bei einem CanSpeakEvent
see_room_menu	Abfrage aller Entities im aktuellen Room
inventory_menu	Abfrage aller Entities im Inventar
help_menu	Abfrage der Tastaurbelegung

Tabelle 6: *Auditive Menülisten des Audio-Clients*

kapselt. So kann bei der Verwendung des Plugins die Funktion einer einzelnen Liste geändert werden, ohne dass die Initialisierungsmethode des Clients oder alle Liste verändert werden müssen. Im Folgenden wird auf die einzelnen Menüs und ihre Verwendungsweise näher eingegangen.

Das Interaktionsmenü ist vom Typ `SoundMenuList`, die Einträge der Liste bestehen demnach aus den Sounds für die möglichen Interaktionen *look_at*, *talk_to*, *pick_up*, *use*, *manipulate* und *cancel* (zum Verlassen des Menüs). Während das Menü aktiv ist, befindet sich der Audio-Client im Input State *INTERACTION*. Die Verwendung von Sounds überlässt die Entscheidung ob Sprachaufnahmen oder Töne, beziehungsweise Tonfolgen verwendet werden sollen dem Entwickler eines Spiels. Lediglich die Assets für die sechs genannten Interaktionen müssen als ogg-Dateien bereitgestellt werden. Gleiches gilt auch für die *menu_sounds* und *empty_sounds* aller Menülisten.

Das folgende Code-Beispiel zeigt die Initialisierung des Interaktionsmenüs,

die in der `_set_up_interaction_menu` Methode erfolgt. Vor diesem Auschnitt erfolgte die Definition der Callback-Funktionen sowie die Initialisierung der Sounds für die Listeneinträge und das Anhängen dieser an das Listenobjekt *interaction_sounds*.

```
# Sound to be played when interaction menu is opened
#
file = self.assets.fetch('interaction_menu.ogg')
menu_sound = pygame.mixer.Sound(file)

interaction_sound_menu = SoundMenuList(interaction_sounds,
                                       menu_sound,
                                       self.attempt_failed_sound,
                                       self.channel_system,
                                       callback_on_interaction_selected,
                                       callback_on_interaction_menu_exit)
```

In der Callback-Funktion `callback_on_interaction_selected` wird für jedes Listenelement ein Verhalten im Falle der Auswahl definiert. Für alle Interaktionen wird der Client in den Input State *SELECT_TARGET* überführt, mit Ausnahme der Interaktion *use*, in diesem Fall ist der neue Input State *SELECT_ITEM*. Wird der Eintrag *cancel* selektiert, wird die Exit-Callback Funktion aufgerufen. In den meisten Menülisten ist die Funktion so implementiert, dass der aktuelle Input State vom *input_state_stack* entfernt wird.

```
def callback_on_item_menu_exit(select_item_menu):

    # Going back to previous input state
    #
    self._pop_input_state_from_stack()

    return
```

Die Bearbeitung des Stacks obliegt den Methoden `_push_input_state_on` und `_pop_input_state_from_stack`, welche das korrekte Hinzufügen oder Entfernen eines States ermöglichen und mit dem aktualisierten *input_state_stack* die Methode `_transition_to_input_state` aufrufen, welche den eigentlichen Zustandsübergang implementiert. In dieser Methode wird geprüft, welcher der aktuelle Zustand ist. Dabei ist der Ausgangszustand *IN_ROOM*. Bei einem Zustandswechsel in den Raum wird der *input_state_stack* verworfen und nur mit dem aktuellen Zustand belegt. Für alle Zustände, die mit einem Listenmenü assoziiert sind, und keine festen Listeneinträge, wie das Interaktionsmenü besitzen, werden die Listeneinträge über die entsprechenden Hilfsmethoden generiert. Konnte die Liste für den aktuellen Input State generiert werden, wird der

Versuch unternommen diese zu öffnen. Das hat im Fall einer leeren Liste zur Folge, dass der *empty_sound* wiedergeben wird und die Exit-Callback-Funktion aufgerufen wird, die ihrerseits wieder dafür sorgt, dass der aktuelle Zustand vom Stack entfernt wird.

So werden auch auch die Menüs für die Auswahl eines Items oder eines Ziels erstellt. Für beide Listen wird die Klasse `EntityMenuList` verwendet. Die Methode zur Erstellung der Listeneinträge für die Auswahl des Ziels berücksichtigen je nach aktuell ausgewählter Aktion unterschiedliche Entities. So werden für die Interaktion *Ansprechen* alle Entities der Typen *Player* und *NPC* des aktuellen Rooms ausgewählt, mit Ausnahme der mit dem Client assoziierten Player-Entity. Für die Interaktion *Ansehen* werden alle Entities des Rooms, sowie die Entities des Inventars der Liste hinzugefügt. Für die Zielauswahl der Interaktionen *Benutzen*, *Aufheben* und *Manipulieren* stehen dagegen nur die Entities aus einem Ausschnitt des Rooms zur Verfügung. Dieser Ausschnitt wird durch das Tile, auf dem sich der Player des Clients befindet, sowie den acht angrenzenden Tiles bestimmt. Für das *Aufheben* erfolgt zusätzlich die Einschränkung, dass die Entity beweglich sein muss. Bei der Auswahl eines Items für die Interaktion *Benutzen* erfolgt ebenfalls die räumliche Begrenzung der Items des Rooms.

Wird die Auswahl eines Ziels bestätigt, wird in der Callback-Funktion für die Selektion eines Eintrags der Menüliste ein entsprechendes AttemptEvent zusammengesetzt und an die für den Server bestimmten Messages übergeben. Das folgende Code-Beispiel zeigt die Erstellung eines TriesToLookAtEvents.

```
# attempt_look_at
#
if self.interaction_sound_menu.list_index == self.ATTEMPT_LOOK_AT:

    interaction_event = fabula.TriesToLookAtEvent(self.host.client_id,
                                                  selected_target)

    self.message_for_host.event_list.append(interaction_event)
```

Das Listenmenü *see_room_menu* enthält alle Entities, die sich im Room befinden. Bei Auswahl einer Entity wird das Koordinaten-Tupel des Tiles vorgelesen, auf dem sich die Entity befindet. Als stellvertretendes Beispiel für den Zugriff auf Entities wird die Methode für die Erstellung der Listeneinträge aufgeführt.

```
def _get_room_entities(self):

    items = []

    for entity in self.host.room.entity_dict.values():
```

```
            if entity.identifier is not self.host.client_id:

                items.append((entity, entity.identifier))

        return items
```

Die Liste für das *inventory_menu* ist ebenfalls eine `EntityMenüList`, die Methode `_get_rack_items`, welche zur Erstellung der Listeneinträge verwendet wird, wird im folgenden Code-Beispiel gezeigt und soll den Zugriff auf Entities des Inventars stellvertretend aufzeigen:

```
    def _get_rack_items(self):

        items = []

        for entity in self.host.rack.entity_dict.values():

            # Check if the entity is owned by someone and if client is
            # that someone
            #
            if (entity.identifier in self.host.rack.owner_dict.keys()
                and self.host.rack.owner_dict[entity.identifier] is self.host.client_id):

                items.append((entity, entity.identifier))

        return items
```

Die verbleibenden Menüs *select_sentence_menu* und *help_menu* sind vom Typ `TextMenuList`. Ersteres stellt eine Besonderheit dar, da dieses Menü nicht durch die Betätigung einer Taste geöffnet wird, sondern als Reaktion auf ein CanSpeakEvent des Servers in der Methode `process_CanSpeakEvent`. Das Event enthält eine Liste mit möglichen Sätzen, die für das Füllen der Liste des Menüs verwendet wird. Weitere `process_[ConfirmEvent]` Methoden werden im Folgenden beschrieben. Ein PerceptionEvent enthält das Attribut *perception*, welches in der entsprechenden `process` Methode von der Text-To-Speech-Engine vorgelesen wird. Ähnlich wird mit einem SaysEvent verfahren, die Engine liest den Namen des Sprechers, gefolgt von dem Attribut *text* des Events vor. Auf ein PicksUpEvent wird mit dem Vorlesen der Nachricht, dass das betroffene Item ins Inventar aufgenommen wurde, reagiert. Der *attempt_failed_sound* des AudioUserInterfaces wird wiedergegeben, wenn ein AttemptFailedEvent eintrifft.

In der Methode `process_deleteEvent` wird geprüft, ob die zu löschende Enity vom Typ *PLAYER* ist und sich die Identität von der Identität des Clients unterscheidet. Ist das der Fall, wurde die Spielfigur eines anderen Mitspielers gelöscht und ein *PerceptionEvent*, mit der *perception* welche Spielfigur den Raum

verlassen hat, wird unmittelbar an die Methode `process_PerceptionEvent` übergeben.

Die Methode `process_MovesToEvent` ermöglicht die akustische Wahrnehmung der Entities, die sich im Umfeld des mit dem Client assoziierten Players befinden. Das Umfeld setzt sich zusammen aus der Position des Tiles auf dem der Player sich befindet und den Positionen der acht angrenzenden Tiles. Jeder Position wird in der Initialisierung des AudioUserInterfaces ein *channel* zugeordnet, sowie eine Lautstärke für den linken und den rechten Kanal einer Stereo-Ausgabe. Der folgende Code-Auschnitt zeigt die Initialisierung der Kanäle, sowie die Verwendung in der von `process_MovesToEvent` aufgerufenen Methode `_play_sound_surrounding_entity`.

```
# One channel for each surrounding position including clients position
# Used to play sounds for items in the surrounding positions - for
# example when moving
#
self.surrounding_position_channels = [(pygame.mixer.Channel(0), 0.7, 0.5),
                                      (pygame.mixer.Channel(1), 0.8, 0.8),
                                      [...]
                                      (pygame.mixer.Channel(7), 0.8, 0.0),
                                      (pygame.mixer.Channel(8), 1, 1)]

[...]

location = self.host.room.entity_locations[entity.identifier]

    if (location in surrounding_positions
        and entity.identifier is not self.host.client_id
        and 'audio/ogg' in entity.assets.keys()
        and entity.assets['audio/ogg'].data is not None):

        fabula.LOGGER.debug("Playing Sounf of: {}".format(entity.identifier))

        # channel consists of a mixer.channel, a value for left volume and
        # a value for right volume.
        #
        channel = {surrounding_positions[0] : self.surrounding_position_channels[0],
                   surrounding_positions[1] : self.surrounding_position_channels[1],
                   [...]
                   surrounding_positions[7] : self.surrounding_position_channels[7],
                   surrounding_positions[8] : self.surrounding_position_channels[8]
                  }[location]

        channel[0].play(entity.assets['audio/ogg'].data)
        channel[0].set_volume(channel[1], channel[2])
```

Im AudioUserInterface kann nicht ganz auf die Verwendung von konkreten sprachlichen Inhalten verzichtet werden, wie beispielsweise bei der Benachrichtigung für den Benutzer, dass ein Item ins Inventar aufgenommen wurde (siehe

Code-Beispiel). Das Plugin verwendet für solche Fälle einheitlich die englische Sprache.

```
self.in_inventory_msg = "{} in inventory"
[...]
picked_up_perception = self.in_inventory_msg.format(item_name)
```

4.2.8 Implementierung des Zauberwald AudioUserInterface

Die auditive Schnittstelle für Zauberwald ist in der Klasse `ZWAudioUI` des Moduls `zauberwald.py` implementiert und ist vom Plugin `AudioUserInterface` abgeleitet. Die Methode `get_connection_details` wird überschrieben und die Einträge der initialen Liste auf Deutsch übersetzt. Die Möglichkeit einen *login* einzugeben wird durch die Einträge „Als Kuni spielen“ und „Als Cassandra spielen“ ersetzt. Wählt der Benutzer ersteren aus, startet das Spiel mit dem *login* „kuni“, letzterer resultiert in einen Spielstart als „cassandra“. Außerdem werden die Benachrichtigungen *in_inventory_msg*, *taker_in_room_msg* und *taker_left_room_msg*, sowie die Listeneinträge des Hilfemenüs ins Deutsche übersetzt. Das Starten des auditiven Zauberwald-Clients erfolgt über die Ausführung des Skripts `start_zauberwald_audio_client.py`, das dieser Arbeit als Anhang C.4 angehängt ist.

4.3 Erreichen von Plattformunabhängigkeit

Um den größtmöglichen Zugang zu der Erweiterung von Fabula zu einer Engine für multimodal zugängliche Spiele zu gewähren, ist die Plattformunabhängigkeit ein wichtiger Schritt. Die Verwendung von Python als Programmiersprache bietet dafür eine gute Basis. Python-Anwendungen können – ähnlich wie Java-Anwendungen – auf verschiedenen Plattformen laufen, ohne dass systemspezifische Anpassungen des Quellcodes vorgenommen werden müssen. Ein großer Teil der Unterschiede zwischen Betriebssystemen und Rechnerarchitekturen werden durch den Python-Interpreter und die Python-System-Module abstrahiert. Das ist naturgemäß nicht mehr gegeben, wenn eine Python-Anwendung auf betriebssystemspezifische Bibliotheken oder Anwendungen zugreift.

Pygame stellt eine Python-Anbindung für die *SDL*-Multimediabibliothek zur Verfügung. SDL ist eine plattformübergreifende Entwicklungsbibliothek die den Zugriff auf Audio- und Grafikhardware, sowie eine Vielzahl von Eingabegeräten bereitstellt (vgl. [12]). Dabei werden alle gängigen Plattformen unterstützt. Damit erfüllt Pygame den Anspruch der Plattformunabhängigkeit.

Pyttsx ist das einzige Python-Modul, das den Ansatz einer allgemeinen Anbindung für unterschiedliche Standard-Text-To-Speech-Engines verfolgt. Es kapselt die spezifischen Details einer Engine in Treibermodulen, wodurch die Verwendung von Text-to-Speech-Engines plattformunabhängig wird und leicht zu erweitern ist. Die Unterstützung der Sprachsynthese-Anwendungen aller gängigen Plattformen sollte somit grundsätzlich die Verwendung des AudioUserInterfaces auf diesen Plattformen ohne Anpassung ermöglichen.

Allerdings ergaben sich während der Implementierung mehrere Probleme bei der Verwendung von Pyttsx. Die Dokumentation des Moduls ist relativ kurz gehalten und setzt sich allein aus den Docstrings der Methoden zusammen. Zusätzlich werden lediglich minimale Beispiele für die Verwendung der Engine zum Sprechen eines Satzes bereitgestellt. Die einfache Handhabung kann anhand dieser Beispiele nachvollzogen werden, es fehlt jedoch eine Beschreibung des erweiterten Funktionsumfangs. Hinzu kommt, dass sich die aufgeführten Beispiele nicht fehlerfrei auf unterschiedlichen Betriebssystemen ausführen lassen.

Leider funktioniert das Stoppen der Sprachausgabe zu einem beliebigen Zeitpunkt nicht, wie in der Dokumentation beschrieben, sondern ist sowohl unter Windows 7 aus auch unter Ubuntu 12.04 fehlerhaft. Daher besteht nur die Möglichkeiten, neue Texte für die Sprachsynthese in die Warteschlange der Engine einzureihen. Dadurch, dass während des Sprechens im Pygame-Thread weiterhin die Benutzereingaben angenommen werden, kann die Warteschlange sehr umfangreich werden und es kommt zu großen Verzögerungen zwischen der auslösenden Benutzereingabe und der Wiedergabe des Textes.

Die Lösung, Benutzereingaben nicht zu verarbeiten, während die Wiedergabe erfolgt, würde eine inkonsistente Benutzerführung zur Folge haben, da auf Tastatureingaben nicht immer ein auditives Feedback erfolgen würde, ohne das sich der Sinn dafür dem Benutzer erschließen würde. Der Ansatz das Problem über die Verwendung von einem separaten Thread für die Steuerung der Sprachausgabe zu lösen, scheitert daran, dass sich im Versuch der Implementierung gezeigt hat, dass Pyttsx nicht threadsafe ist und demnach nicht von mehreren parallelen Threads gemeinsam genutzt werden kann. Die aufgefunden Fehler und die Lücken in der Dokumentation wurden dem Autor des Moduls berichtet, um das Modul und seine Verwendung in Zukunft zu verbessern.

Über die Recherche zur Lösung der Probleme, hat sich herausgestellt, dass sich durch eine kleine Modifikation des SAPI5-Treibers von Pyttsx auch die aktuelle Microsoft Speech Platform (Version 11) und die dafür erhältlichen Stimmen mit Pyttsx verwenden lassen. Dies ermöglicht zumindest unter Windows 7 die

Verwendung von Stimmen, die wesentlich verständlicher und natürlicher klingen, als die für eSpeak verfügbaren Stimmen. Allerdings wäre es wünschenswert, wenn in Zukunft für die Microsoft Speech Platform ein eigener Treiber bereitgestellt würde, da SAPI5 vermutlich von dieser abgelöst wird.

5 Evaluierung

5.1 Testaufbau

Für die Evaluierung wird eine Fallstudie mit dem Prototypen des Spiels durchgeführt. Dafür werden zwei Testplätze mit je einem netzwerkfähigen Computer, einer Standardtastatur und Maus für die Eingabe, einem Bildschirm für visuelle Ausgaben und schließenden Stereo-Kopfhörern für die akustische Ausgabe des Spiels ausgestattet. Beide Computer der Testplätze verwenden Windows 7 als Betriebssystem und erfüllen die Mindestanforderungen für die Benutzung von aktuellen, interaktiven Multimediaanwendungen.[11] Auf beiden Computern ist Python in der Version 3.1.4 (32 Bit), Pygame in der Version 1.9.2a und eine für Python 3.1 portierte Version von Pyttsx 1.1, sowie die in dieser Arbeit erweiterte Version der Fabula Engine vorhanden. Während der Durchführung der Tests befinden sich die Computer im gleichen lokalen Netzwerk und einer der Computer übernimmt zusätzlich die Rolle des Servers für Zauberwald. Ein Testlauf wird jeweils mit zwei Testpersonen gleichzeitig durchgeführt, die dabei den gleichen Client für die Repräsentation des Spiels benutzen.

5.2 Testablauf

Die beiden Testpersonen erhalten gemeinsam eine verbale Spielanleitung, in der eine kurze Beschreibung der Spielfiguren Kuni und Cassandra, sowie ihrer Aufgabe (den Eingang zum Zauberwald finden) erfolgt. Außerdem wird die jeweilige Spielsteuerung erläutert. Die Vorlage für die gesamte Anleitung ist dieser Arbeit als Anhang B angefügt. Nach der Anleitung und der Rollenverteilung, werden die Testperson an die Testplätze gebeten, um sich dort kurz mit dem Testaufbau vertraut zu machen. Wenn dies erfolgt ist, startet die Spielphase.

Die Testpersonen starten gleichzeitig den jeweiligen Client aus einer vorbereiteten Konsole und beginnen das Spiel. Während der Spielphase soll möglichst auf verbale Kommunikation unter den Testspielern außerhalb des Spiels verzichtet werden. Für dringende Fragen steht der Testleiter zur Verfügung. Nach etwa 20 Minuten wird die Spielphase unterbrochen, und die Testpersonen nacheinander in einem separaten Raum befragt. Grundlage der Befragung stellt der Fragebogen im Anhang A dar.

[11] Spezifikationen: PC1: Windows 7 (64 Bit) Home Premium SP 1, Intel(R) Core(TM) i7-3517U CPU 2.4 GHz, 8 GB RAM; PC2: Windows 7 (64 Bit) Professional SP 1, Intel(R) Core(TM) i5-2410M CPU 2.3 GHz, 4 GB RAM

5.3 Testdurchführung

Es wurden zwei Testläufe mit insgesamt vier Testpersonen durchgeführt. Alle Testdurchläufe erfolgten mit dem Zauberwald-Audio-Client. Für die Sprachsynthese wurde die Text-To-Speech-Engine *eSpeak* in Kombination mit der Stimme *eSpeak-DE* verwendet, eine männliche, deutsche Stimme dieser Engine. Die voreingestellte Sprechrate betrug 200 Wörter pro Minute.

5.4 Angaben zu den Testpersonen

Das Alter der Testpersonen liegt in einer Spanne von 25 bis 31 Jahren. Alle Testpersonen sprechen und verstehen fließend Deutsch. Alle Testpersonen verfügen über eine uneingeschränkte Sehfähigkeit und verwenden *täglich* einen Computer. Zwei der Testpersonen spielen *selten* Computerspiele, eine *mehrmals in der Woche* und eine *täglich*. Das generelle Interesse an Computerspielen wird von einer Testperson als *sehr hoch*, von einer weiteren als *hoch* und von zwei Testpersonen als *gering* beschrieben.

Eine blinde Testspielerin wurde ebenfalls als Testperson angefragt. Leider ist ihre Teilnahme in dem Zeitfenster, in welchen der Prototyp die nötige Reife für aussagekräftige Untersuchen erreicht hatte, aus Gründen, welche nicht bei der Autorin lagen, nicht zustande gekommen.

5.5 Feedback der Testpersonen

Für das Feedback zum Spiel und der Verwendung des Clients sind im Fragebogen vier Fragen, sowie freier Raum für weitere Anmerkungen vorhanden. Tabelle 7 zeigt das Ergebnis der konkreten Fragen.

	sehr gut	gut	mittelm.	schlecht	gar nicht
Orientierung		•	• •	•	
	sehr gut	gut	mittelm.	schlecht	gar nicht
Steuerung		• •	• •		
	zu leicht	leicht	anspruchsv.	schwer	zu schwer
Schw.-Grad			•	• • •	
	ja	eher ja	neutral	eher nicht	nein
Spaß		• • • •			

Tabelle 7: *Feedback der Fallstudie*

Es hat sich ergeben, dass sich eine Testperson *gut* im Spiel orientieren konnte, zwei Testpersonen geben für die Orientierung *mittelmäßig* an und eine Testperson konnte sich *schlecht* orientieren. Auf die Frage wie gut die Testperson mit der Steuerung zurecht kommen, geben zwei Testpersonen *gut* und zwei *mittelmäßig* an. Der Schwierigkeitsgrad des Spiels wird von einer Testperson als *anspruchsvoll* beurteilt und von drei Personen mit *schwer*. Alle Testpersonen beantworten die Frage, ob ihnen das Spiel Spaß gemacht hat mit *eher ja*.

Die Anmerkungen des freien Teils der Befragung werden im Folgenden erläutert. Drei Testpersonen empfanden die Soundeffekte zur Beschreibung von Gegenständen als gut, teilweise jedoch als nicht eindeutig genug. Die Stimme der Text-To-Speech-Enging wurde von allen Testpersonen als schwer verständlich beschrieben, so dass Inhalte zum Teil zweimal abgerufen werden mussten, um sie zu verstehen. Zwei Testpersonen haben es als schwierig herausgestellt, sich anhand von Koordinaten in der virtuellen Welt zu orientieren. Eine Person hätte sich die automatische Ansage der Position bei jeder Bewegung gewünscht. Zwei Testpersonen haben eine Einleitung im Spiel vermisst, welche die Atmosphäre und den Ort des Geschehens beschreibt. Die Menüführung mit Listen wurde von einer Person positiv herausgestellt, eine andere empfand die Verwendung der Menüs anfangs als schwierig, nach einer Eingewöhnungszeit jedoch leichter. Eine Testperson hat angemerkt, dass sie nicht das Gefühl hatte, mit einer zweiten Person zusammen zu spielen, sondern eher für sich alleine. Eine Testperson hat Schwierigkeiten mit dem Aufheben von Objekten im Spiel angeführt. Eine Person empfand den akustischen Unterschied zwischen Menü und Welt als zu gering und hat die Verwendung von Hintergrundmusik für die Menüs vorgeschlagen. Das Feedback, wenn die Spielfigur gegen ein Hindernis läuft, war dem Empfinden einer Testperson nach nicht ausreichend, sie hätte sich eine Steigerung des Feedbacks bei mehrfacher Kollision mit dem gleichen Hindernis gewünscht. Außerdem hatte eine Person Schwierigkeiten mit der Vorbelegung der Tasten und den Wunsch geäußert, Aktionen auch über das Betätigen der Taste e bestätigen zu können. Eine Testperson hat angemerkt, dass sie gerne Rückmeldung darüber erhalten hätte, wenn die Spielfigur des anderen Spielers den aktuellen Raum betritt oder verlässt.

6 Erkenntnisse

6.1 Resultate

Im Rahmen dieser Arbeit wurde der Stand der Wissenschaft für auditive und multimodal zugängliche Spiele erhoben und dargestellt. Ein Ergebnis der Recherche ist, dass bisher nur wenige multimodal zugängliche Spiele implementiert wurden, die auch für mehrere Modalitäten einen vollen Zugang gewähren. Darüber hinaus existiert bislang keine Game-Engine für die Erstellung solcher Spiele.

Die über auditive Spiele und Zugänge gewonnenen Erkenntnisse sind sowohl in den Entwurf einer auditiven Benutzerschnittstelle als auch in die Konzeption eines Spiels mit visuellem und auditivem Zugang eingeflossen. Es wurden wiederverwendbare, auditiv wahrnehmbare Komponenten für die entworfene Benutzerschnittstelle konzipiert und implementiert. Die dafür benötigte Anbindung einer Text-To-Speech-Engine sowie einer Schnittstelle für das Abspielen von Sounds wurden über die Verwendung von plattformunabhängigen und quelloffenen Bibliotheken erreicht.

Die Fabula Game-Engine wurde in Form eines Fabula-Plugins mit diesen Funktionalitäten in erweitert. Das Plugin kann als auditiver Client für mit Fabula erstellte Spiele eingesetzt werden. Zusammengenommen mit dem schon bestehenden visuellen Client kann die Repräsentation dadurch multimodal erfolgen, wobei jeder Client für sich genommen, dem Anspruch eines vollwertigen Zugangs gerecht wird.

Des Weiteren konnte ein Konzept für ein multimodal zugängliches Spiel entwickelt und mit der erweiterten Fabula Engine umgesetzt werden. Dafür wurde die Spiellogik implementiert, visuelle Inhalte produziert und auditive Inhalte durch das Einsprechen von Texten erstellt. Auditive Inhalte aus der Gruppe der natürlichen Geräusche und musikalische Elemente wurden aus frei verfügbaren Quellen bezogen.[12]

In einer klein angelegten Fallstudie konnte der auditive Client für Fabula durch seine praktische Verwendung auf der Grundlage des entwickelten Spiels evaluiert werden.

[12] Alle nicht eingesprochenen oder durch Sprachsynthese generierten auditiven Inhalte stammen von `http://www.freesound.org` und wurden unter der Wahrung der für sie geltenden creative commons Lizenzen verwendet. Bei der Auslieferung des Quellcodes und den Spielressourcen erfolgt eine Auflistung der Urheber in der Datei README.

6.2 Diskussion

6.2.1 Diskussion der auditiven Schnittstelle

Da der Umfang der Fallstudie sehr begrenzt ist, ist es nicht möglich allgemeingültige Erkenntnisse aus ihr zu ziehen, allerdings können unter Vorbehalt gewisse Annahmen und Rückschlüsse getroffen werden. Um diese sicher zu Bestätigen, wären Tests mit einer größeren Anzahl an Testpersonen aus unterschiedlicheren demografischen Gruppen notwendig. Es wäre vor allem wünschenswert, dafür auch Personen der potentiellen Zielgruppe sehgeschädigter Menschen gewinnen zu können um die Eignung der Benutzerschnittstelle für diese nicht nur abschätzen, sondern auch wissenschaftlich fundiert belegen zu können.

Die Testpersonen, die an der Fallstudie teilgenommen haben, sind alle uneingeschränkt sehfähig, damit sind sie mit der auditiven Benutzung eines Computers ebenso wenig vertraut, wie mit rein auditiven Spielen. Für die Evaluierung kann das allerdings auch als Vorteil interpretiert werden. Wenn es ungeübten Menschen gelingt, ein Spiel auf Anhieb mit dem Audio-Client zu spielen, kann das als gutes Indiz für ein funktionierendes Konzept der Bedienung gewertet werden.

Die Evaluierung der Fallstudie hat gezeigt, dass Testspieler mit unterschiedlichen Spielerfahrungen beim Spielen von Zauberwald mit dem Audio-Client die grundsätzliche Navigation in der virtuellen Welt bewältigen konnten, wenn auch mehr oder weniger gut. In Anbetracht des ungewohnten Konzepts, sich ein mentales Model von einem Raum nur anhand von akustischen Informationen zu schaffen, ist dieses Ergebnis zu erwarten gewesen. Es zeigt allerdings auch, dass genug akustische Hinweise gegeben werden, um im Schnitt eine mittelmäßige Orientierung für Ungeübte gewährleisten zu können. Aus der Befragung nach Durchführung der Tests ist außerdem hervorgegangen, dass die Orientierung mit zunehmender Spieldauer verbessert werden konnte, also die Orientierungsfähigkeit aufgrund auditiver Stimuli mit zunehmender Erfahrung gesteigert werden kann.

Mit dem Feedback zur Verwendung der Steuerung des Spiels verhält es sich ähnlich, nur dass die Testpersonen die Handhabung der Steuerung insgesamt besser bewertet haben. Auch in diesem Fall ist es positiv zu werten, dass Benutzer, die mit der auditiv geführten Bedienung nicht vertraut sind, diese auf Anhieb als mindestens mittelmäßig oder gut bewerten.

Der Schwierigkeitsgrad des Spiels wurde von den Testpersonen als schwer oder anspruchsvoll eingeordnet, in Anbetracht der Tatsache, dass alle Spieler auf die Frage, ob das Spiel Ihnen Spaß gemacht hat mit *eher ja* geantwortet

haben, lässt sich vermuten, das der Schwierigkeitsgrad des Spiels zwar fordernd, jedoch nicht überfordernd ist.

Basierend auf den genannten Beobachtungen und Indizien würde ich die technische Entwicklung der auditiven Schnittstelle als gelungen bezeichnen, auch wenn die Bedienbarkeit in einigen Punkten noch verbessert werden könnte. Dafür konnten während der Testdurchläufe wichtige Erkenntnisse gesammelt werden, außerdem können dem Feedback der Testpersonen nützliche Hinweise und Ideen entnommen werden. Die einzelnen Punkte werden im Folgenden erläutert.

Das Abspielen von Hintergrundmusik während der Benutzer sich in einem Menü befindet, wäre eine gute Erweiterung des Clients. In der momentanen Implementierung erfolgt im Menü nur direktes Feedback auf eine Benutzereingabe. Wenn längere Zeit keine Eingaben erfolgen, erhält der Benutzer praktisch keine Informationen aus dem Spiel.

Die Implementierung des *AudioUserInterface* ist zwar durch die Verwendung von globalen Variablen darauf ausgelegt, dass die Tastaturbelegung schnell und unkompliziert geändert werden kann, allerdings fehlt noch ein Optionsmenü in dem die tatsächlichen Änderungen auch vorgenommen werden können. Die Verwendung von Listen als Typ der globalen Variablen ermöglicht außerdem das Anlegen von parallel existierenden, alternativen Tastaturbelegungen. Das Optionsmenü könnte leicht mit einer weiteren auditiven Menüliste implementiert werden, hier wird der Vorteil der Erstellung von wiederverwendbaren auditiven Widgets deutlich.

Aus der Befragung hat sich ebenfalls ergeben, dass zu wenig Feedback darüber gegeben wird, was der andere Spieler macht. Dem Wunsch, darüber informiert zu werden, wenn der andere Spieler den Raum verlässt oder wieder betritt, sollte unbedingt nachgekommen werden. Beim visuellen Client sieht der Benutzer, wenn der andere Spieler den Raum betritt oder verlässt, dieses Feedback muss ebenfalls im Audio-Client verfügbar sein. Dieser Aspekt wurde erst durch die Durchführung der Fallstudie aufgedeckt. Da dieses Feedback allerdings als sehr wichtig erachtet wurde, wurde der Audio-Client letztendlich so erweitert, dass diese Rückmeldungen in der aktuellen Version auch akustisch erfolgen. Wenn der Spieler einen Raum betritt, in dem sich Mitspieler befinden, wird er darüber informiert, in dem der Name des Spielers angesagt wird. Dies erfolgt auch, wenn der Raum, in dem sich die Spielfigur befindet von einer Figur eines Mitspielers betreten wird. Analog dazu wird beim Entfernen eines Mitspielers aus dem aktuellen Raum verfahren – unabhängig davon, ob der Spieler den Raum durch das Teleportieren in einen anderen Raum verlässt, oder vom Server

entfernt wird, weil der dazu gehörende Client beendet wurde.

Die Kritik an der Verständlichkeit der Stimme der Text-To-Speech-Engine, welche von allen Testspielern geäußert wurde, finde ich berechtigt, die Lösung liegt jedoch nicht unmittelbar in meinen Händen. Unter Windows 7 und neueren Versionen von Windows könnte die Lösung in Form eines Pyttsx-Engine-Treibers für die *Microsoft Speech Platform* erfolgen. Für Linux und andere Plattformen, welche auf die Verwendung von *eSpeak* angewiesen sind, sehe ich keine Möglichkeit auf eine bessere Stimme zugreifen zu können.

Allerdings könnte grundsätzlich vor Beginn des Spiels ein Konfigurationsmenü für die Text-To-Speech-Engine vorgeschaltet werden, in dem der Benutzer aus allen Stimmen, die auf seinem System für die verwendete Text-to-Speech-Engine installiert sind, eine auswählen kann, die seinem Empfinden nach am besten zu verstehen ist. Außerdem sollte der Benutzer in diesem Menü die Sprechgeschwindigkeit der Engine einstellen können. Für Benutzer, welche mit Sprachsynthese nicht vertraut sind, ist eine geringere Anzahl von Wörtern pro Minute sinnvoll, während die Sprechgeschwindigkeit für geübte Hörer wesentlich höher sein kann. Leider fehlt hier das Feedback einer mit Sprachsynthese gut vertrauten Testperson.

Nach der Durchführung der Tests und des anschließenden Interviews haben die Testpersonen Zauberwald noch einmal mit dem visuellen Client gespielt. Dabei ist aufgefallen, dass der gleiche Spielfortschritt, der während der 20-minütigen Testphase unter Verwendung des Audio-Clients erzielt wurde, in etwa einem viertel der Zeit erreicht werden konnte. Die Ursache dafür könnte darin begründet sein, dass der Spieler schon die Verwendung von bestimmten Gegenständen erlernt hat und auch weniger Interaktion mit dem anderen Spieler benötigt, um die Rätsel zu lösen, weil er die benötigten Schritte zur Lösung schon kennt.

Ein weiterer Erklärungsversuch ist, dass bei allen vier Testpersonen die visuelle Wahrnehmung wesentlich geschulter ist, als die auditive. Ein sehender Mensch, ist nur in sehr seltenen Fällen darauf angewiesen, seine Umgebung über akustische oder taktile Sinne zu erfassen. Blinde Menschen dagegen, sind trainiert, ein mentales Modell ihrer Umwelt aus akustischen Informationen abzuleiten. Ein bekanntes Beispiel für so ein mentales Modell ist die Verwendung von Uhrzeiten um die Anordnung von Speisen auf einem Teller zu beschreiben. Auch das Abzählen von Schritten zur Orientierung und Abschätzung von Entfernung in gewohnten Umgebungen führt zu einer Vorstellung eines Raumes, welches dem abstrakten Bewegungsmodell von Fabula vermutlich recht nah kommt. Leider fehlen an dieser Stelle Testergebnisse, um die angestellten Vermutungen

belegen zu können.

Zudem wären auch weitere Tests, die gleichzeitig mit sehenden und blinden Menschen durchgeführt werden, ein guter Ansatzpunkt, um zu evaluieren, ob sich der blinden Mensch mit dem Audio-Client in einer vergleichbaren Geschwindigkeit wie ein sehender Mensch mit dem visuellen Client durch die virtuelle Welt bewegen kann.

Eine Testperson hatte angemerkt, dass sie ein zusätzliches Bewegungsfeedback in Form von Koordinaten begrüßt hätte. Genauer, dass nach jedem Schritt, den eine Spielfigur gegangen ist, die neue Position als Koordinate von der Text-To-Speech-Engine vorgelesen wird.

Bei der Konzeption wurde das Ansagen der Koordinaten, sowohl von der eigenen Position, als auch von den Objekten im Raum absichtlich so entworfen, dass die Informationen nicht permanent verfügbar sind, um den Spieler nicht um den Aspekt des explorativen Spielens zu bringen. Allerdings hat sich während der Fallstudie gezeigt, dass es mindestens zwei Arten gibt, sich die virtuelle Welt über den auditiven Sinn zu erschließen.

Zwei der Testpersonen sind so vorgegangen, dass sie den Raum erkundet haben, in dem sie darin herumgelaufen sind und auf die akustische Repräsentationen von Objekten geachtet haben. Die beiden anderen Testspieler sind dagegen so vorgegangen, dass sie sich die Positionen aller Objekte im Raum ansagen lassen haben, sobald sie einen Raum betreten haben. Die gehörten Positionen wurden dann versucht zu erinnern, um dann gezielt auf dem kürzesten Weg dorthin zu gelangen. Nach jedem Schritt wurde dann die aktuelle Position der Spielfigur mit dem Weg zu Ziel abgeglichen.

Das Verhalten von weitaus mehr Testpersonen müsste analysiert werden, um daraus stichhaltige Schlüsse auf unterschiedliche Verhaltensmuster zu zulassen. Allerdings ergibt sich aus der Fallstudie ein erstes Indiz dafür, dass es mindestens zwei sehr verschiedene Verhaltensmuster gibt. Daher wird in Betracht gezogen, die Ansage der Position nach jeder Bewegung auch in die Konfigurationsoptionen des Audio-Clients aufzunehmen.

Insgesamt betrachtet könnte die auditive Schnittstelle durch die Ermöglichungen von Konfigurationen unterschiedlicher Bereiche verbessert werden. Allerdings sollte abschließend nochmal festgehalten werden, dass das auditive Fabula-Plugin in seiner jetzigen Version alle grundlegenden Spielmechaniken von Fabula verarbeiten und auch auditiv repräsentieren kann und somit den vollen Zugang zu Spielen ermöglicht.

6.2.2 Diskussion der Erweiterung der Engine

Auch wenn in der Fallstudie keine Testläufe mit gleichzeitiger Verwendung des visuellen und des auditiven Clients durchgeführt wurden, konnte dies während der Implementierungsphase getestet werden. Die Tests erfolgten sowohl lokal, auf einem Rechner, als auch auf zwei, über ein lokales Netzwerk verbundenen Computern. Die Repräsentation erfolgt in beiden Clients annähernd simultan, wobei dies stark von der eingestellten Sprechgeschwindigkeit der Text-To-Speech-Engine, sowie der Lesegeschwindigkeit des Benutzers des visuellen Clients abhängt und somit auch variieren kann.

Bei der Verarbeitung von einem *SaysEvent* kann es allerdings zu einer störenden Verzögerung der Wiedergabe von Informationen kommen. Bei einer Einstellung der Sprechrate von 200 Wörtern pro Minute benötigt die Text-To-Speech-Engine in der Regel eine Sekunde weniger zur Sprachsynthese, als die Anzeige des Textes im visuellen Client dauert. Bei einzeln auftretenden *SaysEvents* fällt die Verzögerung kaum ins Gewicht, bei mehreren aufeinander folgenden *SaysEvents* kann es jedoch dazu führen, dass sich eine Verschiebung von mehreren Sekunden ergibt.

Da dieses Problem erst in einem weit fortgeschrittenen Stadium der Arbeit erkannt wurde, weil der Fokus auf der Evaluierung der auditiven Schnittstelle lag, konnte noch keine Lösung für die Behebung des Problems implementiert werden. Ein theoretischer Lösungsansatz besteht im Folgendem:

Da der Server eines Fabula Spiels über eine *action_time* Variable verfügt, welche die Dauer einer Aktion festlegt, und diese auch an verbundene Clients propagiert, hat jedes Fabula einen Ansatzpunkt für die zeitliche Synchronisation. Diese Dauer der Aktion kann verwendet werden, um das Einsetzen der Sprachsynthese zu verzögern, so dass es gleichzeitig mit der Darstellung des Textes im visuellen Client erfolgt. Bei der Implementierung eines Spiels sollte allerdings beachtet werden, dass diese Dauer nicht zu kurz angegeben wird, sodass die Sprachsynthese nicht in der angegebenen Zeit generiert werden kann. Die einzige Möglichkeit in diesem Fall die Sprachsynthese mit der übergebenen Dauer zu synchronisieren, wäre das Anheben der Sprechrate, was aber bewirken würde, dass die Ausgabe nicht mit der gewünschten Sprechgeschwindigkeit des Benutzer erfolgen würde.

Abgesehen von der Gefahr, dass die Wiedergabe und die Repräsentation eines *SaysEvents* in den beiden verschiedenen Clients zeitlich verzögert ablaufen kann, können die Clients ohne weiter Einschränkungen miteinander verwendet werden. Die stabile Ausführung des Spiels wird nicht durch das angeführte Problem gefährdet.

Die parallele Verwendung von zwei Audio-Clients miteinander, oder auch von zwei visuellen Clients, ist ebenfalls möglich. Dabei gilt für den ersten Fall die Einschränkung, dass zwei Audio-Clients nicht mit vollem Funktionsumfang lokal auf einem Rechner verwendet werden können, weil Pygame nur Sounds des aktiven Fensters wiedergeben kann, aber nur ein Fenster zu einem Zeitpunkt aktiv sein kann.

Die Umsetzung des Spiels Zauberwald funktioniert unabhängig von dem gewählten Zugang und büßt weder in der auditiven noch in der visuellen Version grundlegende Funktionen ein. Damit kann auch die Konzeption und Umsetzung eines mutlimodal zugänglichen Spiels als gelungen betrachtet werden.

In Hinsicht auf die, in der Einleitung dieser Arbeit aufgestellten Hypothese, lässt sich folglich sagen, dass die bestehende Engine Fabula erfolgreich erweitert und ein Spiel mit der erweiterten Engine erstellt werden konnte. Dieses Spiel kann über beide Schnittstellen getrennt voneinander und – mit der genannten Einschränkung – sogar gleichzeitig miteinander gespielt werden.

Dabei hat sich die Wahl der Engine insgesamt bewährt, zum einen Dank der gut erweiterbaren Struktur der Game-Engine, ausführlichen Dokumentation und ausführliche Kommentierung. Zum anderen um dem Anspruch zu genügen eine quelloffene Engine für multimodal zugängliche Spiele zu schaffen. In dieser Arbeit konnte ein Weg aufgezeigt, werden, wie eine Game-Engine erweitert werden kann, sodass sie für die Erstellung von multimodal zugänglichen Spielen verwendet werden kann. Des Weiteren konnte mit dieser Arbeit eine Grundlage für die Erforschung multimodal zugänglicher Spiele geschaffen werden.

Im Forschungsgebiet der Entwicklung universal zugänglicher Spiele stößt diese Arbeit allerdings an ihre Grenzen. Da diese Spiele es erfordern die unterschiedlichen Zugangsarten frei zu kombinieren, um nicht nur alle Beeinträchtigungen zu berücksichtigen, sondern auch die Kombination mehrerer. Der visuelle und der auditive Client stellen jedoch einen jeweils für sich abgeschlossenen Zugang dar. Die kombinierte Verwendung der Benutzerschnittstellen in einem Client ist aufgrund der unterschiedlichen Menüführung nicht möglich.

6.2.3 Weitere Anwendungsgebiete

Neben der Entwicklung weiterer Spiele der von Fabula unterstützen Genre sind auch andere Anwendungsszenarien denkbar.

So könnte zum Beispiel eine App für ein Smartphone entwickelt werden, die das Erstellen von auditiv wahrnehmbaren Routen erlaubt. Seit der Plattform-Version 1.6 verfügt beispielsweise das Betriebssystem Android über eine Text-To-Speech-Engine. Die App könnte so ausgeprägt sein, dass in einer visuellen

Karte von einem sehenden oder Blinden Orte von Interesse eingetragen und mit Hinweisen versehen werden. Beim Ablaufen der Route wird die Position des mobilen Geräts über GPS erfasst und bei Erreichen eines markierten Punktes der entsprechende Hinweis gegeben. Diese App ließe sich für unterschiedliche Zwecke einsetzen, so könnten blinde Menschen zum Beispiel Karten austauschen, in denen sie für blinde Menschen gut begehbare Wege eintragen, oder auf Gefahren hinweisen. Weiterhin ein sehender Mensch eine auditive Stadtführung für blinde Menschen anlegen, welche die für Blinde nicht wahrnehmbaren Gebäude oder Landmarken beschreibt.

Ein weiteres Szenario wäre die Erschaffung eines Trainingsprogramms für Feuerwehrmänner und -frauen um die Orientierung bei Dunkelheit zu trainieren, mit der sie sich konfrontiert sehen, wenn sie sich durch ein brennendes, von Rauch erfülltes Gebäude bewegen müssen. Das Erlernen von der Orientierung über die Anzahl von Schritten könnte gefördert werden und bei einem möglichen Einsatz hilfreich sein. Für das Trainingsprogramm könnten Räume und Gebäude nachgebildet werden, durch die sich dann der Benutzer ohne visuelles Feedback zu einem bestimmten Ziel und wieder zurück zum Ausgang bewegen muss.

Literatur

[1] T. Allman, R.K. Dhillon, M.A.E. Landau, und S.H. Kurniawan. Rock vibe: Rock band® computer games for people with no or limited vision. In *Proceedings of the 11th international ACM SIGACCESS conference on Computers and accessibility*, Assets '09, Seiten 51–58, New York, NY, USA, 2009. ACM.

[2] Dominique Archambault und Damien Olivier. How to make games for visually impaired children. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACE '05, Seiten 450–453, New York, NY, USA, 2005. ACM.

[3] M. T. Atkinson und S. Gucukoglu. Download. `http://agrip.org.uk/download`, letzter Zugriff am: 28.09.2013.

[4] M. T. Atkinson, S. Gucukoglu, C. H. C. Machin, und A. E. Lawrence. Making the mainstream accessible: redefining the game. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, Sandbox '06, Seiten 21–28, New York, NY, USA, 2006. ACM.

[5] F. Berger und W. Müller. Towards an open source game engine for teaching and research. In *Transactions on Edutainment VIII*, volume 7220 of *Lecture Notes in Computer Science*, Seiten 69–76. Springer Berlin Heidelberg, 07 2012.

[6] S. A. Brewster, P. C. Wright, und A. D. N. Edwards. An evaluation of earcons for use in auditory human-computer interfaces. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, CHI '93, Seiten 222–227, New York, NY, USA, 1993. ACM.

[7] T. Dannecker, M. Pasedag, C. Stoll, und H. Sturm. Der Tag Wird Zur Nacht. Hochschule der Medien, Stuttgart. 2003. [Download]. `http://www.dertagwirdzurnacht.de/`.

[8] L. de Ruijter, P. de Ruijter, B. Duvigneau, und D. Loots. Topspeed 3. Playing in the Dark. 2011. [Download]. `http://www.playinginthedark.net/topspeed3_e.php`.

[9] S. Donikian und J.N. Portugal. Writing interactive fiction scenarii with dramachina. In Stefan Göbel, Ulrike Spierling, Anja Hoffmann, Ido Iurgel, Oliver Schneider, Johanna Dechau, und Axel Feix, Hrsg., *Technologies for Interactive Digital Storytelling and Entertainment*, volume 3105 of *Lecture*

Notes in Computer Science, Seiten 101–112. Springer Berlin Heidelberg, 2004.

[10] E. Glinert und L. Wyse. Audiodyssey: an accessible video game for both sighted and non-sighted gamers. In *Proceedings of the 2007 conference on Future Play*, Seiten 251–252. ACM, 2007.

[11] Creative Heroes. Audiogames, your resource for audiogames, games for the blind, games for the visually impaired! `http://audiogames.net/`, letzter Zugriff am: 22.07.2013.

[12] S. Lantinga. Simple directmedia layer - homepage. `http://www.libsdl.org/`, letzter Zugriff am: 03.10.2013.

[13] L. Lindstrom und G. Lingl. [pygame] bug in mixer module with python 3.1 + pygame 1.9.1 ? `https://groups.google.com/forum/#!topic/pygame-mirror-on-google-groups/HSOlZXmGHcA`, letzter Zugriff am: 05.10.2013.

[14] R. McCrindle und D. Symons. Audio space invaders. In *Proceedings of the Third International Conference on Disability, Virtual Reality and Associated Technologies*, Seiten 59–65. Citeseer, 2000.

[15] Microsoft. Ausgeben von Text über die Sprachausgabe. `http://windows.microsoft.com/de-at/windows-8/hear-text-read-aloud-with-narrator`, letzter Zugriff am: 03.10.2013.

[16] Microsoft. Hear text read aloud with Narrator. `http://windows.microsoft.com/en-us/windows7/hear-text-read-aloud-with-narrator`, letzter Zugriff am: 03.10.2013.

[17] D. Miller, A. Parecki, und S. A. Douglas. Finger dance: a sound game for blind people. In *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*, Assets '07, Seiten 253–254, New York, NY, USA, 2007. ACM.

[18] S. Oviatt. Multimodal interfaces. In A. Sears und J.A. Jacko, Hrsg., *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, Second Edition*, Human Factors and Ergonomics, chapter 21, Seiten 413–432. Taylor & Francis, 2007.

[19] P. Parente. pyttsx - text-to-speech x-platform — pyttsx 1.2 documentation. `http://pyttsx.readthedocs.org/en/latest/index.html`, letzter Zugriff am: 03.10.2013.

[20] PCS Games. Sarah and the castle of witchcraft and wizardry. 2007. [Download]. http://www.pcsgames.net/.

[21] I. Reed. Blindaudiogames.com home page. http://blindaudiogames.com, letzter Zugriff am: 01.02.2013.

[22] N. Röber und M. Masuch. Interacting with sound: An interaction paradigm for virtual auditory worlds. In *ICAD*, 2004.

[23] N. Röber und M. Masuch. Leaving the screen: New perspectives in audio-only gaming. In *11th Int. Conf. on Auditory Display (ICAD)*. Citeseer, 2005.

[24] P. Shinners und Pygame Open Source Community. Wiki - pygame - python game development. http://www.pygame.org/wiki/about, letzter Zugriff am: 04.10.2013.

[25] D. Stern-Sapad und N. Moline. Hogwarts live. 2004. [Web Browser]. http://www.hogwartslive.com/.

[26] T. Westin. Game accessibility case study: Terraformers–a real-time 3d graphic game. In *Proceedings of the 5th International Conference on Disability, Virtual Reality and Associated Technologies, ICDVRAT*, Seiten 95–100, 2004.

[27] B. Yuan und E. Folmer. Blind hero: enabling guitar hero for the visually impaired. In *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility*, Seiten 169–176. ACM, 2008.

[28] B. Yuan, E. Folmer, und F.C. Harris. Game accessibility: a survey. *Universal Access in the Information Society*, 10(1):81–100, 2011.

A Fragebogen

Test-Fragebogen Zauberwald

Datum:

Startuhrzeit:
Enduhrzeit:
Spieldauer:

Angaben zur Person

Name:

Alter:

Sehfähigkeit: ☐ blind ☐ sehgeschädigt ☐ uneingeschränkt

Geschlecht: ☐ weiblich ☐ männlich ☐ unbestimmt

Fragen zum Testspiel

Wie gut konntest du dich im Spiel orientieren?

sehr gut ☐ gut ☐ mittelmäßig ☐ schlecht ☐ gar nicht ☐

Wie gut bist du mit der Steuerung zurecht gekommen?

sehr gut ☐ gut ☐ mittelmäßig ☐ schlecht ☐ gar nicht ☐

Hat dir das Spiel Spaß gemacht?

ja ☐ eher ja ☐ neutral ☐ eher nicht ☐ nein ☐

Wie würdest du den Schwierigkeitsgrad einschätzen?

zu leicht ☐ leicht ☐ anspruchsvoll ☐ schwer ☐ zu schwer ☐

Fragen zum Nutzungsverhalten

Wie oft benutzt du einen Computer?

selten ☐ mehrmals im Monat ☐ mehrmals in der Woche ☐ täglich ☐

Wie oft spielst du Computerspiele?

selten ☐ mehrmals im Monat ☐ mehrmals in der Woche ☐ täglich ☐

Wie würdest du dein Interesse an Computerspielen beschreiben?

sehr hoch ☐ hoch ☐ gering ☐ kein Interesse ☐

Für blinde Menschen

Welche Ein- und Ausgabegeräte benutzt du in Verbindung mit einem Computer?

Anmerkungen

B Spielanleitung

Informationen zum Spiel

Einleitung

Das Spiel ist ein Fantasy Adventure Spiel und wird von zwei Personen gemeinsam gespielt. Dafür übernimmt jeder von euch/Ihnen eine Rolle. Die Charaktere „Gnom Kuni“ und „Fee Cassandra“ stehen zur Auswahl.

[Rollenverteilung]

Aufgabenstellung

Eure/Ihre beiden Figuren befinden sich auf einer Reise die durch den Zauberwald führt. Dafür müssen Kuni und Cassandra allerdings zunächst den Eingang zum Zauberwald finden. Ziel des Spiels ist es, den Eingang des Zauberwaldes zu finden und diesen zu betreten. Um dieses Ziel zu erreichen, müssen kleine Rätsel gelöst werden. Dafür könnt ihr / können Sie gerne durch eure/Ihre Figuren miteinander sprechen.

[Erläuterung der Steuerung]

Steuerung Pygame-Client

- Maussteuerung
  - Links-Klick auf eine Position: Spielfigur bewegt sich dort hin
  - Rechts-Klick auf ein Objekt öffnet visuelles Interaktionsmenü
  - Drag&Drop Gegenstände können ins Inventar oder auf andere Objekte gezogen werden

Steuerung Audio-Client

- Menüsteuerung Listen
  - Pfeiltaste hoch: Einen Eintrag nach oben gehen
  - Pfeiltaste runter: Einen Eintrag nach unten gehen
  - Pfeiltaste rechts: Einen Eintrag auswählen
  - Pfeiltaste links: Das Menü verlassen
- Bewegung im Raum über wasd-Tasten

  - w: Spielfigur geht ein Feld nach oben
  - a: Spielfigur geht ein Feld nach links
  - s: Spielfigur geht ein Feld nach unten
  - d: Spielfigur geht ein Feld nach rechts
- Menüs im Spiel
  - q: Inventar öffnen
  - e: Interaktionsmenü öffnen
  - x: Position im Raum abfragen
  - y: Alle Gegenstände des Raums abfragen

Hinweise

- Während des Testspiels wenn möglich auf verbale Kommunikation mit dem anderen Spieler verzichten.
- Eventuell aufkommende Fragen bitte an den Testleiter stellen
- Nach etwa 20 Minuten wird die Spielphase beendet, der Hinweis dazu erfolgt durch den Testleiter.
- Die Spielsteuerung des Audio-Clients kann auch im Spiel mit der Taste F1 abgerufen werden
- Das Spiel wird über die Entertaste in einer Konsole gestartet

C Quellcode

C.1 zauberwald.py

```
"""Zauberwald Server-Side Plugin for Fabula

   Copyright 2013 Linda Kerkhoff
"""

# This file is part of Zauberwald.
#
# Zauberwald is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# Zauberwald is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# See <http://www.gnu.org/licenses/>.

# Work started on 05. Aug 2013.

import pygame
import fabula.plugins.pygameui
from fabula.plugins.audioui import AudioTextField
from fabula.plugins.audioui import TextMenuList

PLANES = fabula.plugins.pygameui.planes

# Constant Zauberwald player IDs
#
ID_KUNI = 'kuni'
ID_CASSANDRA = 'cassandra'

class ZWServerPlugin(fabula.plugins.serverside.DefaultGame):
    """This is the serverside Plugin for the Zauberwald game.

    Attributes:

    ZWServerPlugin.sentences_cassandra
        Sentences to save the potential conversation possibilities with Kuni.
        Initially Introduction sentences.

    ZWServerPlugin.sentences_kuni
        Sentences to save the potential conversation possibilities with
        Cassandra. Initially Introduction sentences.
    """

    def __init__(self, host):
        """Call base class __init__(), add custom logic.
        """

        # Call base class
        #
        fabula.plugins.serverside.DefaultGame.__init__(self, host)

        self.sentences_kuni = ['Hallo, ich bin der Gnom Kuni.',
                               'Wer bist du?']

        self.sentences_cassandra = ['Guten Tag, ich bin die Fee Cassandra.',
                                    'Mit wem habe ich die Ehre?']

        self.lute_is_repaired = False

        return

    def respond(self, event):
        """Add an Event from condition_response_dict corresponding to the Event given to message_for_host.

           If the Event is not found, return AttemptFailedEvent.
        """
```

```
71
72          # Save the room from where the event was to current_room
73          #
74          room = None
75
76          for current_room in self.host.room_by_id.values():
77
78              if event.identifier in current_room.entity_dict.keys():
79
80                  room = current_room
81
82          event_list = []
83          messages = []
84
85          # Call the proper respond method for the current room to handle
86          # room specific events
87          #
88          if room.identifier == 'default':
89
90              event_list = self._respond_room_default(event, room)
91
92          elif room.identifier == 'room_cassandra':
93
94              event_list = self._respond_room_cassandra(event, room)
95
96          elif room.identifier == 'room_kuni':
97
98              event_list = self._respond_room_kuni(event, room)
99
100         elif room.identifier == 'room_entry':
101             event_list = self._respond_room_entry(event, room)
102
103         # Check events which can happen in all rooms for example perception
104         # of items when they are carried around and SaysEvents between
105         # Cassandra and Kuni.
106         #
107         if event == fabula.TriesToTalkToEvent(identifier=ID_KUNI,
108                                               target_identifier=ID_CASSANDRA):
109
110             # sentences_kuni is changed according to game logic and contains the
111             # right phrases at the right time
112             #
113             event_list += [fabula.CanSpeakEvent(identifier=ID_KUNI,
114                                                 sentences=self.sentences_kuni)]
115
116         elif event == fabula.TriesToTalkToEvent(identifier=ID_CASSANDRA,
117                                                 target_identifier=ID_KUNI):
118
119             # sentences_cassandra is changed according to game logic and
120             # contains the right phrases at the right time
121             #
122             event_list += [fabula.CanSpeakEvent(identifier=ID_CASSANDRA,
123                                                 sentences=self.sentences_cassandra)]
124
125         elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
126                                                  target_identifier='pry')):
127
128             event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
129                                                   perception='Ein Brecheisen, das könnte noch nützlich sein.')]
130
131         elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
132                                                  target_identifier='cake')):
133
134             event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
135                                                   perception='Oh, ein Stückchen Kuchen, sieht das gut aus! '
136                                                              'Aber ich bin ja auf Diät.')]
137
138         elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
139                                                  target_identifier='lute_broken')):
140
141             event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
142                                                   perception='Ein wirklich schönes Instrument, leider fehlen ihre Saiten')]
143
144         elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
145                                                  target_identifier='lute')):
146
```

```
147                 event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
148                                                       perception='Eine wohlklingende Laute!')]
149
150             elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
151                                                      target_identifier='goblet_filled')):
152
153                 event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
154                                                       perception='Ein mit Tau gefüllter Blütenkelch.')]
155
156             if (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
157                                                    target_identifier='dew')):
158
159                 event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
160                                                       perception='Tau von den Blumen')]
161
162             elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
163                                                      target_identifier='goblet')):
164
165                 event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
166                                                       perception='Ein schöner Blütenkelch')]
167
168             elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
169                                                      target_identifier='string_harp')):
170
171                 event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
172                                                       perception='Die Saite einer Harfe')]
173
174             elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
175                                                      target_identifier='lute')):
176
177                 event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
178                                                       perception='Eine funktionierende Laute.')]
179
180             elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
181                                                    item_identifier='dew',
182                                                    target_identifier='goblet')):
183
184                 event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
185                                                       perception='Der Kelch ist jetzt mit dem Tau gefüllt')]
186
187                 if 'dew' not in room.entity_locations.keys():
188
189                     messages += [fabula.Message([fabula.DropsEvent(ID_KUNI,
190                                                                    self.host.rack.entity_dict['dew'],
191                                                                    room.entity_locations[ID_KUNI])])]
192
193                 if 'goblet' not in room.entity_locations.keys():
194
195                     messages += [fabula.Message([fabula.DropsEvent(ID_KUNI,
196                                                                    self.host.rack.entity_dict['goblet'],
197                                                                    room.entity_locations[ID_KUNI])])]
198
199                 messages += [fabula.Message([fabula.SpawnEvent(fabula.Entity('goblet_filled',
200                                                                              fabula.ITEM,
201                                                                              True,
202                                                                              True,
203                                                                              {'image/png': fabula.Asset(uri='goblet_filled.png',
204                                                                                                         data=None),
205                                                                               'audio/ogg': fabula.Asset(uri='goblet_filled.ogg',
206                                                                                                         data=None),
207                                                                               'text/plain': fabula.Asset(uri='goblet_filled.txt',
208                                                                                                          data=None)}),
209                                                                room.entity_locations[ID_KUNI] + (room.identifier,))]),
210                                             fabula.Message([fabula.DeleteEvent(identifier='goblet')]),
211                                             fabula.Message([fabula.DeleteEvent(identifier='dew')]),
212                                             fabula.Message([fabula.PicksUpEvent(identifier=ID_KUNI,
213                                                                                 item_identifier='goblet_filled')])]
214
215             elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
216                                                    item_identifier='goblet',
217                                                    target_identifier='dew')):
218
219                 event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
220                                                       perception='Warum sollte ich den Blütenkelch mit Tau benetzen, '
221                                                                  'der sieht ganz sauber aus.')]
222
```

```
223             elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
224                                                      target_identifier=ID_KUNI)):
225
226                 perception_of_kuni = 'Der Gnom Kuni.'
227
228                 if 'Hallo, ich bin der Gnom Kuni.' in self.sentences_kuni:
229
230                     perception_of_kuni = 'Ein kleiner Gnom, ob der wohl weiß, wo es zum Zauberwald geht?'
231
232
233                 event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
234                                                       perception=perception_of_kuni)]
235
236             elif event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
237                                                     target_identifier=ID_CASSANDRA):
238
239                 perception_of_cassandra = 'Die Fee Cassandra'
240
241                 if 'Guten Tag, ich bin die Fee Cassandra.' in self.sentences_cassandra:
242
243                     perception_of_cassandra = 'Hm, eine Fee...allerdings sieht sie doch ein wenig rundlich aus!'
244
245
246                 event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
247                                                       perception=perception_of_cassandra)]
248
249             # Game Logic done
250
251             if len(event_list):
252
253                 fabula.LOGGER.info("returning corresponding events")
254
255                 self.message_for_host.event_list += event_list
256
257             else:
258
259                 fabula.LOGGER.info("no action defined for event '{}', returning AttemptFailedEvent to host".format(event))
260
261                 self.message_for_host.event_list.append(fabula.AttemptFailedEvent(event.identifier))
262
263             self.queue_messages(*messages)
264
265             return
266
267         def process_message(self, message):
268             """Calls the base class and adds spawning functionality for teleporting tiles to switch rooms.
269
270                Returns ZWServerPlugin.message_for_host.
271             """
272
273             # Call base class
274             # This will add event to self.message_for_host
275             fabula.plugins.serverside.DefaultGame.process_message(self, message)
276
277             # Kuni teleporting
278             # (6, 0, 'default') -> Teleports Kuni to room_kuni
279             #
280             if (ID_KUNI in self.host.room_by_client.keys()
281                and self.host.room_by_client[ID_KUNI].identifier == "default"
282                and ID_KUNI in self.host.room_by_client[ID_KUNI].entity_locations.keys()
283                and self.host.room_by_client[ID_KUNI].entity_locations[ID_KUNI] == (6, 0)):
284
285                 if ('cobweb' in self.host.rack.owner_dict.keys()
286                     and self.host.rack.owner_dict['cobweb'] == ID_KUNI):
287
288                     # Cassandra needs cobweb to leave the room, Kuni is not allowed to leave
289                     #
290                     self.message_for_host.event_list.append(fabula.PerceptionEvent(ID_KUNI,
291                                                                                    'Du solltest vielleicht vorher Cassandra'
292                                                                                    'mit den Flügel helfen.'))
293                     self.process_TriesToMoveEvent(fabula.TriesToMoveEvent(ID_KUNI,
294                                                                           (5, 0)))
295                 else:
296
297                     self._remove_sentence(ID_KUNI, 'Die Tür zum Gnomenweg ist offen! Da gehts zum Zauberwald lang!')
298
```

```
                # Delete client
                #
                self.message_for_host.event_list.append(fabula.DeleteEvent(ID_KUNI))

                # Load room_kuni for kuni and spawn kuni at position (1, 1)
                #
                self.message_for_host.event_list.extend(self._load_room(ID_KUNI,
                                                                         "room_kuni",
                                                                         (1, 1)))

                # Delete pending movements if present
                #
                if ID_KUNI in self.tries_to_move_dict.keys():

                    del self.tries_to_move_dict[ID_KUNI]

            elif (ID_KUNI in self.host.room_by_client.keys()
               and self.host.room_by_client[ID_KUNI].identifier == "room_kuni"
               and ID_KUNI in self.host.room_by_client[ID_KUNI].entity_locations.keys()):

                # (7, 4 'room_kuni') -> Teleports Kuni to 'room_entry')
                #
                if self.host.room_by_client[ID_KUNI].entity_locations[ID_KUNI] == (7, 4):

                    # Delete client
                    #
                    self.message_for_host.event_list.append(fabula.DeleteEvent(ID_KUNI))

                    # Load room_entry for kuni and spawn kuni at position (1, 3)
                    #
                    self.message_for_host.event_list.extend(self._load_room(ID_KUNI,
                                                                             "room_entry",
                                                                             (1, 3)))

                    # Delete pending movements if present
                    #
                    if ID_KUNI in self.tries_to_move_dict.keys():

                        del self.tries_to_move_dict[ID_KUNI]

                # (1, 0 'room_kuni') -> Teleports Kuni to 'room_default')
                #
                elif self.host.room_by_client[ID_KUNI].entity_locations[ID_KUNI] == (1, 0):

                    # Delete client
                    #
                    self.message_for_host.event_list.append(fabula.DeleteEvent(ID_KUNI))

                    # Load default room for kuni and spawn kuni at position (5, 0)
                    #
                    self.message_for_host.event_list.extend(self._load_room(ID_KUNI,
                                                                             "default",
                                                                             (5, 0)))

                    # Delete pending movements if present
                    #
                    if ID_KUNI in self.tries_to_move_dict.keys():

                        del self.tries_to_move_dict[ID_KUNI]

            # (1, 4, 'room_entry') -> Teleports Kuni to room_kuni
            #
            elif (ID_KUNI in self.host.room_by_client.keys()
               and self.host.room_by_client[ID_KUNI].identifier == "room_entry"
               and ID_KUNI in self.host.room_by_client[ID_KUNI].entity_locations.keys()
               and self.host.room_by_client[ID_KUNI].entity_locations[ID_KUNI] == (1, 4)):

                # Delete client
                #
                self.message_for_host.event_list.append(fabula.DeleteEvent(ID_KUNI))

                # Load room_kuni for kuni and spawn kuni at position (6, 4)
                #
                self.message_for_host.event_list.extend(self._load_room(ID_KUNI,
                                                                         "room_kuni",
                                                                         (6, 4)))
```

```
375
376             # Delete pending movements if present
377             #
378             if ID_KUNI in self.tries_to_move_dict.keys():
379
380                 del self.tries_to_move_dict[ID_KUNI]
381
382         # Cassandra teleporting
383         # (6, 3, 'default') -> Teleports Cassandra to room_kuni
384         #
385         if (ID_CASSANDRA in self.host.room_by_client.keys()
386            and self.host.room_by_client[ID_CASSANDRA].identifier == "default"
387            and ID_CASSANDRA in self.host.room_by_client[ID_CASSANDRA].entity_locations.keys()
388            and self.host.room_by_client[ID_CASSANDRA].entity_locations[ID_CASSANDRA] == (6, 3)):
389
390             if ('key' in self.host.rack.owner_dict.keys()
391                 and self.host.rack.owner_dict['key'] == ID_CASSANDRA):
392
393                 # Kuni needs key to leave the room, Cassandra is not allowed to leave
394                 #
395                 self.message_for_host.event_list.append(fabula.PerceptionEvent(ID_CASSANDRA,
396                                                                                'Du bist zu schwer um mit dem '
397                                                                                'Schlüssel über den Fluss zu fliegen.'))
398                 self.process_TriesToMoveEvent(fabula.TriesToMoveEvent(ID_CASSANDRA,
399                                                                       (6, 2)))
400
401             else:
402
403                 self._remove_sentence(ID_CASSANDRA, 'Meine Flügel sind wieder heil, ich kann fliegen!')
404
405                 # Delete client
406                 #
407                 self.message_for_host.event_list.append(fabula.DeleteEvent(ID_CASSANDRA))
408
409                 # Load room_cassandra for Cassandra and spawn her at position (1, 2)
410                 #
411                 self.message_for_host.event_list.extend(self._load_room(ID_CASSANDRA,
412                                                                         "room_cassandra",
413                                                                         (1, 2)))
414
415                 # Delete pending movements if present
416                 #
417                 if ID_CASSANDRA in self.tries_to_move_dict.keys():
418
419                     del self.tries_to_move_dict[ID_CASSANDRA]
420
421         elif (ID_CASSANDRA in self.host.room_by_client.keys()
422            and self.host.room_by_client[ID_CASSANDRA].identifier == "room_cassandra"
423            and ID_CASSANDRA in self.host.room_by_client[ID_CASSANDRA].entity_locations.keys()):
424
425             # (7, 3 'room_cassandra') -> Teleports Cassandra to 'room_entry')
426             #
427             if self.host.room_by_client[ID_CASSANDRA].entity_locations[ID_CASSANDRA] == (7, 3):
428
429                 # Delete client
430                 #
431                 self.message_for_host.event_list.append(fabula.DeleteEvent(ID_CASSANDRA))
432
433                 # Load room_entry for Cassandra and spawn her at position (5, 3)
434                 #
435                 self.message_for_host.event_list.extend(self._load_room(ID_CASSANDRA,
436                                                                         "room_entry",
437                                                                         (5, 3)))
438
439                 # Delete pending movements if present
440                 #
441                 if ID_CASSANDRA in self.tries_to_move_dict.keys():
442
443                     del self.tries_to_move_dict[ID_CASSANDRA]
444
445             # (0, 2 'room_cassandra') -> Teleports Cassandra to 'room_default')
446             #
447             elif self.host.room_by_client[ID_CASSANDRA].entity_locations[ID_CASSANDRA] == (0, 2):
448
449                 # Delete client
450                 #
```

```
451                 self.message_for_host.event_list.append(fabula.DeleteEvent(ID_CASSANDRA))
452
453                 # Load default room for Cassandra and spawn her at position (5, 3)
454                 #
455                 self.message_for_host.event_list.extend(self._load_room(ID_CASSANDRA,
456                                                                         "default",
457                                                                         (5, 3)))
458
459                 # Delete pending movements if present
460                 #
461                 if ID_CASSANDRA in self.tries_to_move_dict.keys():
462
463                     del self.tries_to_move_dict[ID_CASSANDRA]
464
465
466         # (5, 4, 'room_entry') -> Teleports Cassandra to room_cassandra
467         #
468         elif (ID_CASSANDRA in self.host.room_by_client.keys()
469            and self.host.room_by_client[ID_CASSANDRA].identifier == "room_entry"
470            and ID_CASSANDRA in self.host.room_by_client[ID_CASSANDRA].entity_locations.keys()
471            and self.host.room_by_client[ID_CASSANDRA].entity_locations[ID_CASSANDRA] == (5, 4)):
472
473             # Delete client
474             #
475             self.message_for_host.event_list.append(fabula.DeleteEvent(ID_CASSANDRA))
476
477             # Load room_cassandra for Cassandra and spawn her at position (6, 3)
478             #
479             self.message_for_host.event_list.extend(self._load_room(ID_CASSANDRA,
480                                                                     "room_cassandra",
481                                                                     (6, 3)))
482
483             # Delete pending movements if present
484             #
485             if ID_CASSANDRA in self.tries_to_move_dict.keys():
486
487                 del self.tries_to_move_dict[ID_CASSANDRA]
488
489         return self.message_for_host
490
491     def process_TriesToPickUpEvent(self, event):
492         """Call base, but also respond().
493         """
494
495         # First call base...
496         #
497         fabula.plugins.serverside.DefaultGame.process_TriesToPickUpEvent(self,
498                                                                          event)
499
500         # Problem with Fabula: Unfortunately, the process_... handlers do not
501         # set a return value. So check whether the most recent event is not
502         # AttemptFailed.
503         #
504         if not isinstance(self.message_for_host.event_list[-1], fabula.AttemptFailedEvent):
505
506             self.respond(event)
507
508         return
509
510     def process_TriesToDropEvent(self, event):
511         """Copied from DefaultGame.
512            Changed to disable non item-targets. Game Logic depends on dropping
513            items only on other items. Think Monkey Island.
514            If the target is an Entity, forward to respond().
515            If the target is a position, send a PerceptionEvent.
516         """
517
518         room = None
519
520         for current_room in self.host.room_by_id.values():
521
522             if event.identifier in current_room.entity_dict.keys():
523
524                 room = current_room
525
526         # Restrict drops to tiles right next to the player.
```

```
            #
            player_location = room.entity_locations[event.identifier]

            surrounding_positions = fabula.surrounding_positions(player_location)

            # Include own position to allow drop on self
            #
            surrounding_positions += [player_location]

            # Server.process_TriesToDropEvent() has already done some checks,
            # so we can be sure that target_identifier is either a valid
            # coordinate tuple or an instance of fabula.Entity.
            #
            if event.target_identifier in room.entity_dict.keys():

                fabula.LOGGER.debug("target '{}' is an entity identifier".format(event.target_identifier))

                if room.entity_locations[event.target_identifier] in surrounding_positions:

                    fabula.LOGGER.info("'{}' dropped on Entity '{}',"
                                       " forwarding to respond()".format(event.item_identifier,
                                                                         event.target_identifier))
                    self.respond(event)

                else:

                    self.message_for_host.event_list.append(fabula.AttemptFailedEvent(event.identifier))

                    msg = "AttemptFailed: drop of '{}' on '{}' at {} not next to player: {}, nor in rack"

                    fabula.LOGGER.info(msg.format(event.item_identifier,
                                                  event.target_identifier,
                                                  room.entity_locations[event.target_identifier],
                                                  surrounding_positions))

            elif event.target_identifier in self.host.rack.owner_dict.keys():

                fabula.LOGGER.info("'{}' dropped on Entity in rack '{}',"
                                   "forwarding to respond()".format(event.item_identifier,
                                                                    event.target_identifier))
                self.respond(event)

            else:
                fabula.LOGGER.debug("target '{}' is not an entity identifier".format(event.target_identifier[:2]))

                self.message_for_host.event_list.append(fabula.PerceptionEvent(event.identifier,
                                                                               'Das möchte ich hier nicht ablegen.'))

                fabula.LOGGER.info("PerceptionEvent: drop of '{}' on "
                                   "position {} is not desired".format(event.item_identifier,
                                                                       event.target_identifier[:2]))

            return

        def _remove_sentence(self, id, sentence):
            """Auxiliary method to remove a sentence from clients sentences collection in a save way.
            """

            if id == ID_KUNI:

                if sentence in self.sentences_kuni:

                    self.sentences_kuni.remove(sentence)

            elif id == ID_CASSANDRA:

                if sentence in self.sentences_cassandra:

                    self.sentences_cassandra.remove(sentence)

            return

        def _append_sentence(self, id, sentence):
            """Auxiliary method to append a sentence on clients sentences collection only if sentences does not already exists.
            """
```

```
603             if id == ID_KUNI:
604
605                 if sentence not in self.sentences_kuni:
606
607                     self.sentences_kuni.append(sentence)
608
609             elif id == ID_CASSANDRA:
610
611                 if sentence not in self.sentences_cassandra:
612
613                     self.sentences_cassandra.append(sentence)
614
615             return
616
617         def _respond_room_default(self, event, room):
618             """Handles game logic for default room.
619             """
620
621             event_list = []
622             messages = []
623
624             # identifier ID_KUNI
625             #
626             if event == fabula.SaysEvent(identifier=ID_KUNI,
627                                          text='Hallo, ich bin der Gnom Kuni.'):
628
629                 self._remove_sentence(ID_KUNI, event.text)
630                 self._append_sentence(ID_KUNI, 'Es muss irgendwo einen Eingang zum alten Gnomenweg geben.')
631                 self._remove_sentence(ID_CASSANDRA, 'Mit wem habe ich die Ehre?')
632                 self._append_sentence(ID_CASSANDRA, 'Die Brücke, die über den Fluss in Richtung Zauberwald führt, ist kaputt.')
633
634                 event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
635                                                 text='Sehr erfreut!')]
636
637             elif event == fabula.SaysEvent(identifier=ID_KUNI,
638                                            text='Wer bist du?'):
639
640                 self._remove_sentence(ID_KUNI, event.text)
641                 self._append_sentence(ID_KUNI, 'Ich suche den Weg zum Zauberwald!')
642                 self._remove_sentence(ID_CASSANDRA, 'Guten Tag, ich bin die Fee Cassandra.')
643                 self._append_sentence(ID_CASSANDRA, 'Ich bin auf einer Reise, die mich durch den Zauberwald führt!')
644
645                 event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
646                                                 text='Guten Tag, ich bin die Fee Cassandra.')]
647
648             elif event == fabula.SaysEvent(identifier=ID_KUNI,
649                                            text='Ich suche den Weg zum Zauberwald!'):
650
651                 event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
652                                                 text='Ich ebenfalls.')]
653
654             elif (event == fabula.SaysEvent(identifier=ID_KUNI,
655                                             text='Es muss irgendwo einen Eingang zum alten Gnomenweg geben.')):
656
657                 self._remove_sentence(ID_KUNI, event.text)
658
659                 event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
660                                                 text='Vielleicht finden wir ja den Eingang!')]
661
662             elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
663                                             text='Die Brücke, die über den Fluss in Richtung Zauberwald führt, ist kaputt.')):
664
665                 self._remove_sentence(ID_CASSANDRA, event.text)
666
667                 event_list += [fabula.SaysEvent(identifier=ID_KUNI,
668                                                 text='Hm - kannst du nicht über den Fluss fliegen? Du bist doch eine Fee.')]
669
670                 messages += [fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
671                                                               text='Mein Flügel muss geklebt werden '
672                                                                    'dann könnte ich wieder fliegen.')]),
673                              fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
674                                                               text='Hm - könntest du mich mitnehmen?')]),
675                              fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
676                                                               text='Leider können meine Flügel kaum mein eigenes Gewicht tragen.')])]
677
678             elif (event == fabula.SaysEvent(identifier=ID_KUNI,
```

```
                                    text='Die Tür zum Gnomenweg ist offen! Da gehts zum Zauberwald lang!')):

            self._remove_sentence(ID_KUNI, event.text)

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Mein Flügel muss geklebt werden dann könnte ich über den Fluss fliegen.')]

            messages += [fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                          text='Ich pass da nicht durch')]),
                         fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
                                                          text='Dann flieg doch beim Schild über den Fluss.')])]

        elif (event == fabula.SaysEvent(identifier=ID_KUNI,
                                        text='Deine Flügel sind ganz, du kannst wieder fliegen!')):

            self._remove_sentence(ID_KUNI, event.text)

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Tausend Dank!')]

        elif (event == fabula.SaysEvent(identifier=ID_KUNI,
                                        text='Hinter der Spinne ist doch irgendwas!')
              or event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                           text='Die Spinne verbirgt etwas!')):

            self._remove_sentence(ID_KUNI, 'Hinter der Spinne ist doch irgendwas!')
            self._remove_sentence(ID_CASSANDRA, 'Die Spinne verbirgt etwas!')

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Vielleicht kann ich sie weglocken!')]

            messages += [fabula.Message([fabula.TriesToMoveEvent(identifier=ID_CASSANDRA,
                                                                 target_identifier=(1, 4))]),
                         fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                          text='Oh ein einzelner Spinnweben')]),
                         fabula.Message([fabula.TriesToPickUpEvent(identifier=ID_CASSANDRA,
                                                                   target_identifier='cobweb')])]

        elif (event == fabula.SaysEvent(identifier=ID_KUNI,
                                        text='Ich hab den Eingang zum Gnomenweg gefunden! Aber er ist verschlossen.')):

            self._remove_sentence(ID_KUNI, event.text)

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Vielleicht liegt der Schlüssel hier rum?')]

        elif event == fabula.SaysEvent(identifier=ID_KUNI,
                                       text='Ich hab den Eingang zum Gnomenweg gefunden! Aber er ist verschlossen.'):

            self._remove_sentence(ID_KUNI, event.text)

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Vielleicht liegt der Schlüssel hier irgendwo rum?')]

        elif event == fabula.SaysEvent(identifier=ID_KUNI,
                                       text='Auf dem Schild an der Brücke steht Überfliegen erlaubt!'):

            self._remove_sentence(ID_KUNI, event.text)

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Mit reparierten Flügeln kann ich da also rüber fliegen')]

        elif (event == fabula.SaysEvent(identifier=ID_KUNI,
                                        text='Ich habe einen sehr klebrigen Spinnweb dabei.')):

            self._remove_sentence(ID_KUNI, event.text)

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Vielleicht kann ich damit meinen Flügel kleben?')]

        elif (event == fabula.SaysEvent(identifier=ID_KUNI,
                                        text='Rück mal ein Stück!')):

            event_list += [fabula.SaysEvent('spider', 'Raaar! schirr!')]

            messages += [fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
```

```
                                                text='Öhm nichts für ungut, ich komm später wieder!')])]

        elif (event == fabula.SaysEvent(identifier=ID_KUNI,
                                        text='Öhm, kannst du mal beiseite treten?')):

            event_list += [fabula.SaysEvent('spider', 'Raaar! Ich beiß dich gleich!')]

            messages += [fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
                                                text='Ähm, ich geh dann mal!')])]

        elif (event == fabula.SaysEvent(identifier=ID_KUNI,
                                        text='Geh mir aus dem Weg!')):

            event_list += [fabula.SaysEvent('spider', 'Du wärst ein guter Happen für zwischendurch!')]

            messages += [fabula.Message([fabula.SaysEvent(ID_KUNI, 'Mit der Spinne ist wohl nicht gut Kirschen essen!')])]

        # Events Kuni for target spider
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                                 target_identifier='spider')):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Woah, das ist aber eine große Spinne!')]

            if room.entity_locations['spider'] == (1, 0):

                messages += [fabula.Message([fabula.PerceptionEvent(identifier=ID_KUNI,
                                                                    perception='Hinter der Spinne ist doch irgendwas!')])]

                self._append_sentence(ID_KUNI, 'Hinter der Spinne ist doch irgendwas!')

        elif (event == fabula.TriesToTalkToEvent(identifier=ID_KUNI,
                                                 target_identifier='spider')):

            event_list += [fabula.CanSpeakEvent(identifier=ID_KUNI,
                                                sentences=['Öhm, kannst du mal beiseite treten?',
                                                           'Rück mal ein Stück!',
                                                           'Geh mir aus dem Weg!'])]

        # Events Kuni for target door
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                                 target_identifier='door')):

            self._remove_sentence(ID_KUNI, 'Es muss irgendwo einen Eingang zum alten Gnomenweg geben.')
            self._append_sentence(ID_KUNI, 'Ich hab den Eingang zum Gnomenweg gefunden! Aber er ist verschlossen.')

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Das müsste die Tür zum Gnomenweg sein! '
                                                             'Mist, sie ist verschlossen.')]

        # Events Kuni for target key
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                                 target_identifier='key')):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Ein goldener, kleiner alter Schlüssel.')]

        elif (event == fabula.TriesToPickUpEvent(identifier=ID_KUNI,
                                                 target_identifier='key')):

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Ich hab einen Schlüssel gefunden!')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                               item_identifier='key',
                                               target_identifier='door')):

            self._remove_sentence(ID_CASSANDRA, 'Ich glaube, ich hab den Eingang zum Gnomenweg gefunden.')
            self._remove_sentence(ID_KUNI, 'Ich hab den Eingang zum Gnomenweg gefunden! Aber er ist verschlossen.')

            self._append_sentence(ID_KUNI, 'Die Tür zum Gnomenweg ist offen! Da gehts zum Zauberwald lang!')

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
```

```
831                                             perception='Der Eingang zum Gnomenweg ist offen.')]
832
833             messages += [fabula.Message([fabula.SpawnEvent(fabula.Entity('door_unlocked',
834                                                          fabula.ITEM,
835                                                          False,
836                                                          False,
837                                                          {'image/png': fabula.Asset(uri='1x1-empty.png',
838                                                                                     data=None),
839                                                           'audio/ogg': fabula.Asset(uri='door_unlocked.ogg',
840                                                                                     data=None),
841                                                           'text/plain': fabula.Asset(uri='door_unlocked.txt',
842                                                                                      data=None)}),
843                                                          room.entity_locations['door'] + (room.identifier,))]),
844                         fabula.Message([fabula.DropsEvent(ID_KUNI,
845                                                          self.host.rack.entity_dict['key'],
846                                                          room.entity_locations['door'])]),
847                         fabula.Message([fabula.DeleteEvent(identifier='key')]),
848                         fabula.Message([fabula.DeleteEvent(identifier='door')])]
849
850         elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
851                                                item_identifier='key',
852                                                target_identifier=ID_CASSANDRA)):
853
854             fabula.PerceptionEvent(identifier=ID_KUNI,
855                                    perception='Cassandra hat den Schlüssel gesehen.')
856
857             messages += [fabula.Message([fabula.PerceptionEvent(identifier=ID_CASSANDRA,
858                                    perception='Kuni hat einen Schlüssel.')])]
859
860         elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
861                                                item_identifier='key',
862                                                target_identifier='spider')):
863
864             event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
865                                                   perception='Warum sollte ich der fiesen Spinne den Schlüssel zurückgeben?')]
866
867         elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
868                                                item_identifier='key',
869                                                target_identifier='sign')):
870
871             event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
872                                                   perception='Wo soll ich da den Schlüssel reinstecken?')]
873
874         elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
875                                                item_identifier='key',
876                                                target_identifier='cobweb')):
877
878             event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
879                                                   perception='Hm, das macht keinen Sinn.')]
880
881         elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
882                                                item_identifier='key',
883                                                target_identifier='bridge')):
884
885             event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
886                                                   perception='Der Abflugsort ist schon offen.')]
887
888         # Events Kuni for target cobweb
889         #
890         elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
891                                                  target_identifier='cobweb')):
892
893             event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
894                                                   perception='Ein klebriger Spinnenweb, könnte nützlich sein.')]
895
896         elif (event == fabula.TriesToPickUpEvent(identifier=ID_KUNI,
897                                                  target_identifier='cobweb')):
898
899             self._remove_sentence(ID_KUNI, 'Hinter der Spinne ist doch irgendwas!')
900             self._remove_sentence(ID_CASSANDRA, 'Die Spinne verbirgt etwas!')
901             self._append_sentence(ID_KUNI, 'Ich habe einen sehr klebrigen Spinnweb dabei.')
902
903             event_list += [fabula.SaysEvent(identifier=ID_KUNI,
904                                             text='Hey, die Spinne geht weg!')]
905
906             self.process_TriesToMoveEvent(fabula.TriesToMoveEvent(identifier='spider',
```

```
907                                                     target_identifier=(0, 4)))
908
909         elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
910                                                item_identifier='cobweb',
911                                                target_identifier='door_unlocked')):
912
913             event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
914                                                   perception='Warum soll ich mir denn meinen eigenen Weg zu kleben?')]
915
916         elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
917                                                item_identifier='cobweb',
918                                                target_identifier=ID_CASSANDRA)):
919
920             self._remove_sentence(ID_CASSANDRA, 'Laut dem Schild an der Brücke ist das Überfliegen erlaubt!')
921             self._remove_sentence(ID_KUNI, 'Auf dem Schild an der Brücke steht Überfliegen erlaubt!')
922             self._remove_sentence(ID_KUNI, 'Ich habe einen sehr klebrigen Spinnweb dabei.')
923             self._remove_sentence(ID_CASSANDRA, 'Ich habe einen sehr feinen klebrigen Spinnweb dabei.')
924             self._append_sentence(ID_KUNI, 'Deine Flügel sind ganz, du kannst wieder fliegen!')
925
926             event_list += [fabula.SaysEvent(identifier=ID_KUNI,
927                                             text='Deine Flügel sind wieder ganz, du kannst wieder fliegen!')]
928
929             if 'cobweb' not in room.entity_dict.keys():
930
931                 messages += [fabula.Message([fabula.DropsEvent(ID_KUNI,
932                                                                self.host.rack.entity_dict['cobweb'],
933                                                                room.entity_locations[ID_CASSANDRA])])]
934
935             if room.entity_locations['spider'] == (1, 0):
936
937                 self.process_TriesToMoveEvent(fabula.TriesToMoveEvent(identifier='spider',
938                                                                        arget_identifier=(0, 4)))
939                 self._remove_sentence(ID_KUNI, 'Hinter der Spinne ist doch irgendwas!')
940                 self._remove_sentence(ID_CASSANDRA, 'Die Spinne verbirgt etwas!')
941
942             messages += [fabula.Message([fabula.DeleteEvent(identifier='cobweb')]),
943                          fabula.Message([fabula.SpawnEvent(fabula.Entity('bridge',
944                                                                          fabula.ITEM,
945                                                                          False,
946                                                                          False,
947                                                                          {'image/png': fabula.Asset(uri='1x1-empty.png',
948                                                                                                     data=None),
949                                                                           'audio/ogg': fabula.Asset(uri='teleport.ogg',
950                                                                                                     data=None),
951                                                                           'text/plain': fabula.Asset(uri='bridge.txt',
952                                                                                                      data=None)}),
953                                                                          room.entity_locations['sign'] + (room.identifier,))]),
954                          fabula.Message([fabula.DeleteEvent(identifier='sign')])]
955
956         elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
957                                                item_identifier='cobweb',
958                                                target_identifier=ID_KUNI)):
959
960             event_list += [fabula.SaysEvent(identifier=ID_KUNI,
961                                             text='Ihh, sind die klebrig - damit könnte man bestimmt gut Dinge kleben.')]
962
963         elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
964                                                item_identifier='cobweb',
965                                                target_identifier='spider')):
966
967             event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
968                                                   perception='Ganz schön klebrig, '
969                                                              'vielleicht kann ich es besser woanders benutzen.')]
970
971         elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
972                                                item_identifier='cobweb',
973                                                target_identifier='sign')):
974
975             event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
976                                                   perception='Ich möchte den Spinnweb nicht an das Schild kleben.')]
977
978         elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
979                                                item_identifier='cobweb',
980                                                target_identifier='door')):
981
982             event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
```

```
                                            perception='Warum soll ich Spinnweben an diese Tür kleben? '
                                                       'Davon geht sie ja doch nicht auf.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                               item_identifier='cobweb',
                                               target_identifier='key')):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Der Schlüssel muss nicht geklebt werden.')]

        # Events Kuni for target door_unlocked
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                                 target_identifier='door_unlocked')):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Hier gehts für mich weiter Richtung Zauberwald.')]

        # Events Kuni for target sign
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                                 target_identifier='sign')):

            self._append_sentence(ID_KUNI, 'Auf dem Schild an der Brücke steht Überfliegen erlaubt!')

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Brücke gesperrt! Betreten verboten, Überfliegen erlaubt!')]

        # Events Kuni for bridge
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                                 target_identifier='bridge')):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Abflugsort über die Brücke. Leider kann ich nicht fliegen.')]

        # identifier ID_CASSANDRA
        #
        elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Guten Tag, ich bin die Fee Cassandra.')):

            self._remove_sentence(ID_CASSANDRA, 'Guten Tag, ich bin die Fee Cassandra.')
            self._append_sentence(ID_CASSANDRA, 'Ich bin auf einer Reise, die mich durch den Zauberwald führt!')
            self._remove_sentence(ID_KUNI, 'Wer bist du?')
            self._append_sentence(ID_KUNI, 'Ich suche den Weg zum Zauberwald!')

            event_list += [fabula.SaysEvent(identifier=ID_KUNI, text='Angenehm!')]

        elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA, text='Mit wem habe ich die Ehre?')):

            self._remove_sentence(ID_CASSANDRA, 'Mit wem habe ich die Ehre?')
            self._append_sentence(ID_CASSANDRA, 'Die Brücke, die über den Fluss in Richtung Zauberwald führt, ist kaputt.')
            self._remove_sentence(ID_KUNI, 'Hallo, ich bin der Gnom Kuni.')
            self._append_sentence(ID_KUNI, 'Es muss irgendwo einen Eingang zum alten Gnomenweg geben.')

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Hallo, ich bin der Gnom Kuni.')]

        elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Ich bin auf einer Reise, die mich durch den Zauberwald führt!')):

            event_list += [fabula.SaysEvent(identifier=ID_KUNI, text='Ich auch.')]

        elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Ich habe einen sehr feinen klebrigen Spinnweb dabei.')):

            self._remove_sentence(ID_CASSANDRA, event.text)

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Vielleicht kannst du damit deinen Flügel kleben?')]

        elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Meine Flügel sind wieder heil, ich kann fliegen!')):

            self._remove_sentence(ID_CASSANDRA, event.text)

```

```
            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Juhu!')]

        elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Die Tür zum Gnomenweg ist offen! Aber da pass ich nicht durch.')):

            self._remove_sentence(ID_CASSANDRA, event.text)

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Dann flieg doch beim Schild über den Fluss.')]

        elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Ich glaube, ich hab den Eingang zum Gnomenweg gefunden.')):

            self._remove_sentence(ID_CASSANDRA, event.text)

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Ich wusste doch, dass der hier irgendwo sein muss!')]

        elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Laut dem Schild an der Brücke ist das Überfliegen erlaubt!')):

            self._remove_sentence(ID_CASSANDRA, event.text)

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Könnt ich fliegen würd ich da so oder so über den Fluss fliegen!')]

        elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Entschuldigung, könntest du mal einen Schritt zur Seite gehen?')):

            event_list += [fabula.SaysEvent('spider', 'Rrrrrr! Ckkck!')]

            messages += [fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                          text='Oh, Entschuldigung ich wollte nicht stören!')])]

        elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Könntest du mich bitte mal vorbei lassen?')):

            event_list += [fabula.SaysEvent('spider', 'Du wärst aber eine leckere vollwertige Mahlzeit!')]

            messages += [fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                          text='Schnell weg!')])]

        elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Würde es dir etwas ausmachen, mich mal kurz durch zu lassen?')):

            event_list += [fabula.SaysEvent(identifier='spider',
                                            text='Würde es dir was ausmachen, wenn ich dich verspeise?')]

            messages += [fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                          text='Schon, ich geh mal lieber!')])]

        # Events Cassandra for target spider
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                                 target_identifier='spider')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                  perception='Huch, das ist aber eine Riesenspinne!')]

            if room.entity_locations['spider'] == (1, 0):

                self._append_sentence(ID_CASSANDRA, 'Die Spinne verbirgt etwas!')

                messages += [fabula.Message([fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                                    perception='Die Spinne verbirgt etwas!')])]

        elif (event == fabula.TriesToTalkToEvent(identifier=ID_CASSANDRA,
                                                 target_identifier='spider')):

            event_list += [fabula.CanSpeakEvent(identifier=ID_CASSANDRA,
                                                sentences=['Entschuldigung, könntest du mal einen Schritt zur Seite gehen?',
                                                           'Könntest du mich bitte mal vorbei lassen?',
                                                           'Würde es dir etwas ausmachen, mich mal kurz durch zu lassen?'])]

        # Events Cassandra for target door
```

```
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                                 target_identifier='door')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                  perception='Eine kleine Tür, mit einem altem goldenen Schloss!')]
            self._append_sentence(ID_CASSANDRA, 'Ich glaube, ich hab den Eingang zum Gnomenweg gefunden.')

        # Events Cassandra for target key
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                                 target_identifier='key')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                  perception='Ein güldener Schlüssel! Etwas alt, aber hübsch.')]

        elif (event == fabula.TriesToPickUpEvent(identifier=ID_CASSANDRA,
                                                 target_identifier='key')):

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Ich fand einen güldenen Schlüssel und trage ihn bei mir.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                               item_identifier='key',
                                               target_identifier='door')):

            self._remove_sentence(ID_KUNI, 'Ich hab den Eingang zum Gnomenweg gefunden! Aber er ist verschlossen.')
            self._remove_sentence(ID_CASSANDRA, 'Ich glaube, ich hab den Eingang zum Gnomenweg gefunden.')

            self._append_sentence(ID_CASSANDRA, 'Die Tür zum Gnomenweg ist offen! Aber da pass ich nicht durch.')

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Der Eingang zum Gnomenweg ist offen.')]

            messages += [fabula.Message([fabula.SpawnEvent(fabula.Entity('door_unlocked',
                                                                         fabula.ITEM,
                                                                         False,
                                                                         False,
                                                                         {'image/png': fabula.Asset(uri='1x1-empty.png',
                                                                                                    data=None),
                                                                          'audio/ogg': fabula.Asset(uri='door_unlocked.ogg',
                                                                                                    data=None),
                                                                          'text/plain': fabula.Asset(uri='door_unlocked.txt',
                                                                                                     data=None)}),
                                                                         room.entity_locations['door'] + (room.identifier,))]),
                                        fabula.Message([fabula.DropsEvent(ID_CASSANDRA,
                                                                          self.host.rack.entity_dict['key'],
                                                                          room.entity_locations['door'])]),
                                        fabula.Message([fabula.DeleteEvent(identifier='key')]),
                                        fabula.Message([fabula.DeleteEvent(identifier='door')])]

        elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                               item_identifier='key',
                                               target_identifier=ID_KUNI)):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                  perception='Kuni hat den Schlüssel gesehen.')]

            messages += [fabula.Message([fabula.PerceptionEvent(identifier=ID_KUNI,
                                                                perception='Cassandra hat einen Schlüssel.')])]

        elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                               item_identifier='key',
                                               target_identifier='spider')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                  perception='Ich traue mich nicht, der Spinne den Schlüssel zurück zu geben.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                               item_identifier='key',
                                               target_identifier='sign')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                  perception='Ein Schild hat kein Schloss.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
```

```
                                        item_identifier='key',
                                        target_identifier='cobweb')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                          perception='Das lass ich lieber.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                        item_identifier='key',
                                        target_identifier='bridge')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                          perception='Der Abflugsort ist schon offen.')]

        # Events Cassandra for target cobweb
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                          target_identifier='cobweb')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                          perception='Ein sehr klebriger Spinnweb, '
                                                     'damit könnte man bestimmt gut etwas kleben.')]

        elif (event == fabula.TriesToPickUpEvent(identifier=ID_CASSANDRA,
                                          target_identifier='cobweb')):

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Hey, die Spinne bewegt sich!')]

            self._remove_sentence(ID_KUNI, 'Hinter der Spinne ist doch irgendwas!')
            self._remove_sentence(ID_CASSANDRA, 'Die Spinne verbirgt etwas!')
            self._append_sentence(ID_CASSANDRA, 'Ich habe einen sehr feinen klebrigen Spinnweb dabei.')

            self.process_TriesToMoveEvent(fabula.TriesToMoveEvent(identifier='spider',
                                                                   target_identifier=(0, 4)))


        elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                        item_identifier='cobweb',
                                        target_identifier='door_unlocked')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                          perception='Ich bin froh, dass die Tür offen ist, '
                                                     'die möchte ich nicht wieder zu kleben.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                        item_identifier='cobweb',
                                        target_identifier=ID_CASSANDRA)):

            self._remove_sentence(ID_CASSANDRA, 'Laut dem Schild an der Brücke ist das Überfliegen erlaubt!')
            self._remove_sentence(ID_KUNI, 'Auf dem Schild an der Brücke steht Überfliegen erlaubt!')
            self._remove_sentence(ID_KUNI, 'Hinter der Spinne ist doch irgendwas!')
            self._remove_sentence(ID_CASSANDRA, 'Die Spinne verbirgt etwas!')
            self._remove_sentence(ID_KUNI, 'Ich habe einen sehr klebrigen Spinnweb dabei.')
            self._remove_sentence(ID_CASSANDRA, 'Ich habe einen sehr feinen klebrigen Spinnweb dabei.')
            self._append_sentence(ID_CASSANDRA, 'Meine Flügel sind wieder heil, ich kann fliegen!')

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                        text='Meine Flügel sind wieder ganz, ich kann fliegen!')]

            if 'cobweb' not in room.entity_dict.keys():

                messages += [fabula.Message([fabula.DropsEvent(ID_CASSANDRA,
                                                               self.host.rack.entity_dict['cobweb'],
                                                               room.entity_locations[ID_CASSANDRA])])]

            if room.entity_locations['spider'] == (1, 0):

                self.process_TriesToMoveEvent(fabula.TriesToMoveEvent(identifier='spider',
                                                                       target_identifier=(0, 4)))

            messages += [fabula.Message([fabula.DeleteEvent(identifier='cobweb')]),
                         fabula.Message([fabula.SpawnEvent(fabula.Entity('bridge',
                                                         fabula.ITEM,
                                                         False,
                                                         False,
                                                         {'image/png': fabula.Asset(uri='1x1-empty.png',
```

```
                                                                    data=None),
                                                 'audio/ogg': fabula.Asset(uri='teleport.ogg',
                                                                           data=None),
                                                 'text/plain': fabula.Asset(uri='bridge.txt',
                                                                            data=None)}),
                                               room.entity_locations['sign'] + (room.identifier,))]),
                        fabula.Message([fabula.DeleteEvent(identifier='sign')])]

    elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                           item_identifier='cobweb',
                                           target_identifier=ID_KUNI)):


        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Kuni guckt irritiert.')]

        messages += [fabula.Message([fabula.PerceptionEvent(identifier=ID_KUNI,
                                                            perception='Cassandra kommt dir mit Spinnweben '
                                                                       'gefährlich nah.')]),
                     fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
                                                      text='Ihh, was machst du? Das ist klebrig!')])]

    elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                           item_identifier='cobweb',
                                           target_identifier='spider')):

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Ganz schön klebrig, '
                                                         'vielleicht kann ich es besser woanders benutzen.')]

    elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                           item_identifier='cobweb',
                                           target_identifier='sign')):

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Ich möchte den Spinnweb nicht an das Schild kleben.')]

    elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                           item_identifier='cobweb',
                                           target_identifier='door')):

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Warum soll ich Spinnweben an diese Tür kleben?')]

    elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                           item_identifier='cobweb',
                                           target_identifier='key')):

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Der Schlüssel muss nicht geklebt werden.')]

    # Events Cassandra for target door_unlocked
    #
    elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                             target_identifier='door_unlocked')):

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Die Tür ist offen, aber ich pass da nicht durch.')]

    # Events Cassandra for target sign
    #
    elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                             target_identifier='sign')):

        self._append_sentence(ID_CASSANDRA, 'Laut dem Schild an der Brücke ist das Überfliegen erlaubt!')

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Brücke gesperrt! Betreten verboten, Überfliegen erlaubt')]

    # Events Cassandra for bridge
    #
    elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                             target_identifier='bridge')):

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Abflugsort über die Brücke.')]
```

```
1363
1364            self.queue_messages(*messages)
1365
1366            return event_list
1367
1368        def _respond_room_cassandra(self, event, room):
1369            """Handles game logic for room cassandra.
1370            """
1371
1372            event_list = []
1373            messages = []
1374
1375            # TriesToLookAtEvent
1376            #
1377            if (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
1378                                                   target_identifier='box')):
1379
1380                event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
1381                                                      perception='Eine Kiste, leider lässt sie sich nicht öffnen.')]
1382
1383            # TriesToPickUpEvent
1384            #
1385            elif (event == fabula.TriesToPickUpEvent(identifier=ID_CASSANDRA,
1386                                                     target_identifier='lute_broken')):
1387
1388                self._append_sentence(ID_CASSANDRA, "Ich habe eine Laute. Leider ist sie kaputt.")
1389
1390                event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
1391                                                      perception='Vielleicht findet sich noch etwas um die Laute zu reparieren.')]
1392
1393            elif (event == fabula.TriesToPickUpEvent(identifier=ID_CASSANDRA,
1394                                                     target_identifier='cake')):
1395
1396                self._append_sentence(ID_CASSANDRA, "Willst du ein Stück Kuchen?")
1397
1398                event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
1399                                                      perception='Ich pack das Kuchenstück doch mal lieber ein, '
1400                                                                 'für die lange Reise.')]
1401
1402            elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
1403                                                   item_identifier='pry',
1404                                                   target_identifier='cake')):
1405
1406                event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
1407                                                      perception='Das Kuchenstück enthält schon genug Eisen.')]
1408
1409            elif event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
1410                                                  item_identifier='cake',
1411                                                  target_identifier=ID_CASSANDRA):
1412
1413                event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
1414                                                      perception='Ich bin doch auf Diät!')]
1415
1416            elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
1417                                                   item_identifier='cake',
1418                                                   target_identifier='pry')):
1419
1420                event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
1421                                                      perception='Mit Lebensmitteln spielt man nicht!')]
1422
1423            elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
1424                                                   item_identifier='pry',
1425                                                   target_identifier='teleport_default')
1426                  or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
1427                                                      item_identifier='pry',
1428                                                      target_identifier='teleport_entry')
1429                  or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
1430                                                      item_identifier='cake',
1431                                                      target_identifier='teleport_default')
1432                  or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
1433                                                      item_identifier='cake',
1434                                                      target_identifier='teleport_entry')
1435                  or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
1436                                                      item_identifier='lute_broken',
1437                                                      target_identifier='teleport_default')
1438                  or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
```

```
                                                item_identifier='lute_broken',
                                                target_identifier='teleport_entry')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                perception='Das möchte ich hier nicht ablegen.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                               item_identifier='pry',
                                               target_identifier='box')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                perception='Die Kiste ist aufgebrochen!')]

            messages += [fabula.Message([fabula.DropsEvent(ID_CASSANDRA,
                                                           self.host.rack.entity_dict['pry'],
                                                           room.entity_locations['box'])]),
                         fabula.Message([fabula.DeleteEvent(identifier='pry')]),
                         fabula.Message([fabula.SpawnEvent(fabula.Entity('lute_broken',
                                                         fabula.ITEM,
                                                         True,
                                                         True,
                                                         {'image/png': fabula.Asset(uri='lute_broken.png',
                                                                                    data=None),
                                                          'audio/ogg': fabula.Asset(uri='lute_broken.ogg',
                                                                                    data=None),
                                                          'text/plain': fabula.Asset(uri='lute_broken.txt',
                                                                                     data=None)}),
                                                 room.entity_locations['box'] + (room.identifier,))]),
                         fabula.Message([fabula.DeleteEvent(identifier='box')])]

        # Events Cassandra for teleport entry
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                                 target_identifier='teleport_entry')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                perception='Hier geht es weiter Richtung Zauberwald.')]

        # Events Cassandra for teleport_default
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                                 target_identifier='teleport_default')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                perception='Hier geht es zurück auf die andere Seite des Flusses.')]

        self.queue_messages(*messages)

        return event_list

    def _respond_room_kuni(self, event, room):
        """Handles game logic for room kuni.
        """

        event_list = []
        messages = []

        if (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                               target_identifier='elf')):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                perception='Ein Harfe-spielender Elf. '
                                                           'Der sieht ein bisschen unglücklich aus.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                               item_identifier='goblet',
                                               target_identifier='elf')):

            event_list += [fabula.SaysEvent(identifier='elf',
                                            text='Was soll ich mit einem leeren Kelch?')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                               item_identifier='dew',
                                               target_identifier='elf')):

            event_list += [fabula.SaysEvent(identifier='elf',
```

```
                                        text='Hach, ich habs doch schon schwer genug.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                               item_identifier='goblet',
                                               target_identifier='teleport_default')
              or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                  item_identifier='goblet',
                                                  target_identifier='teleport_entry')
              or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                  item_identifier='goblet_filled',
                                                  target_identifier='teleport_default')
              or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                  item_identifier='goblet_filled',
                                                  target_identifier='teleport_entry')
              or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                  item_identifier='dew',
                                                  target_identifier='teleport_default')
              or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                  item_identifier='dew',
                                                  target_identifier='teleport_entry')
              or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                  item_identifier='string_harp',
                                                  target_identifier='teleport_default')
              or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                  item_identifier='string_harp',
                                                  target_identifier='teleport_entry')):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Das möchte ich hier nicht ablegen.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                               item_identifier='goblet_filled',
                                               target_identifier='elf')):

            event_list += [fabula.DropsEvent(ID_KUNI,
                                             self.host.rack.entity_dict['goblet_filled'],
                                             room.entity_locations['elf'])]

            messages += [fabula.Message([fabula.SaysEvent(identifier='elf',
                                                          text='Oh du bist so gut zu mir, dafür schenke ich dir diese Saite!')]),
                         fabula.Message([fabula.SpawnEvent(fabula.Entity('string_harp',
                                                                         fabula.ITEM,
                                                                         True,
                                                                         True,
                                                                         {'image/png': fabula.Asset(uri='string_harp.png',
                                                                                                    data=None),
                                                                          'audio/ogg': fabula.Asset(uri='string_harp.ogg',
                                                                                                    data=None),
                                                                          'text/plain': fabula.Asset(uri='string_harp.txt',
                                                                                                     data=None)}),
                                                                       room.entity_locations['elf'] + (room.identifier,))]),
                         fabula.Message([fabula.DeleteEvent('goblet_filled')]),
                         fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
                                                          text='Ähm...vielen Dank!')]),
                         fabula.Message([fabula.PicksUpEvent(identifier=ID_KUNI,
                                                             item_identifier='string_harp')])]

            self._append_sentence(ID_KUNI, 'Ich habe eine Saite von einer Harfe.')

        # Events Kuni for teleport entry
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                                 target_identifier='teleport_entry')):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Hier geht es weiter Richtung Zauberwald.')]

        # Events Kuni for teleport_default
        #
        elif (event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                                 target_identifier='teleport_default')):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Hier geht es zurück auf den Gnomenweg.')]

        elif (event == fabula.TriesToTalkToEvent(identifier=ID_KUNI,
```

```
                                                target_identifier='elf')):

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text=' Was machst du hier?')]

            messages += [fabula.Message([fabula.SaysEvent(identifier='elf',
                                                          text='Ich hab Harfe-Spielen-Schicht und das noch fünf Stunden lang.')]),
                         fabula.Message([fabula.SaysEvent(identifier='elf',
                                                          text='Dabei bin ich sooo durstig!')])]

        # WOULD BE NICE: silly conversation between Kuni and elf

        self.queue_messages(*messages)

        return event_list

    def _respond_room_entry(self, event, room):
        """Handles game logic for room entry.
        """

        event_list = []
        messages = []

        if event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                              target_identifier='teleport_room_kuni'):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Hier gehts zurück, wo ich hergekommen bin!')]

        elif event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                                target_identifier='teleport_room_cassandra'):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Weg für Gnome verboten!')]

        elif event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                                target_identifier='teleport_room_cassandra'):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                  perception='Dieser Weg führt zurück zum Fluss.')]

        elif event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                                target_identifier='teleport_room_kuni'):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                  perception='Dieser Weg ist nur für sehr kleine Wesen gemacht.')]

        elif event == fabula.TriesToTalkToEvent(identifier=ID_KUNI,
                                                target_identifier='guardian'):

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Hey. Kannst du mich mal durch lassen?')]

            messages += [fabula.Message([fabula.SaysEvent('guardian',
                                                          'Wenn du 5 Goldtaler locker machen kannst.')]),
                         fabula.Message([fabula.SaysEvent(ID_KUNI,
                                                          '5 Taler? Das sind ja Preise!')]),
                         fabula.Message([fabula.SaysEvent('guardian',
                                                          'Wenn du nicht zahlen kannst, kommst du auch nicht durch.')])]

        elif event == fabula.TriesToTalkToEvent(identifier=ID_CASSANDRA,
                                                target_identifier='guardian'):

            event_list += [fabula.SaysEvent(ID_CASSANDRA,
                                            'Ich bin Cassandra die Fee und würde gerne durch den Zauberwald reisen!')]

            messages += [fabula.Message([fabula.SaysEvent('guardian',
                                                          'Wegezoll auch für Feen! 5 Goldtaler bitte.')]),
                         fabula.Message([fabula.SaysEvent(ID_CASSANDRA,
                                                          'Ich habe leider kein Gold.')]),
                         fabula.Message([fabula.SaysEvent('guardian',
                                                          'Tja, dann darfst du nicht passieren.')])]

        elif event == fabula.SaysEvent(identifier=ID_KUNI,
                                       text='Hallo, ich bin der Gnom Kuni.'):
```

```
            self._remove_sentence(ID_KUNI, event.text)
            self._remove_sentence(ID_CASSANDRA, 'Mit wem habe ich die Ehre?')

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Sehr erfreut!')]

        elif event == fabula.SaysEvent(identifier=ID_KUNI,
                                       text='Wer bist du?'):

            self._remove_sentence(ID_KUNI, event.text)
            self._remove_sentence(ID_CASSANDRA, 'Guten Tag, ich bin die Fee Cassandra.')

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Guten Tag, ich bin die Fee Cassandra.')]

        elif event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                       text='Guten Tag, ich bin die Fee Cassandra.'):

            self._remove_sentence(ID_CASSANDRA, 'Guten Tag, ich bin die Fee Cassandra.')
            self._remove_sentence(ID_KUNI, 'Wer bist du?')

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Angenehm!')]

        elif event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                       text='Mit wem habe ich die Ehre?'):

            self._remove_sentence(ID_CASSANDRA, 'Mit wem habe ich die Ehre?')
            self._remove_sentence(ID_KUNI, 'Hallo, ich bin der Gnom Kuni.')

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Hallo, ich bin der Gnom Kuni.')]

        elif event == fabula.SaysEvent(identifier=ID_KUNI,
                                       text='Ich suche den Weg zum Zauberwald!'):

            event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                            text='Ich ebenfalls.')]

        elif event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                       text='Ich bin auf einer Reise, die mich durch den Zauberwald führt!'):

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Ich auch.')]

        elif event == fabula.SaysEvent(identifier=ID_CASSANDRA,
                                       text='Willst du ein Stück Kuchen?'):

            self._remove_sentence(ID_CASSANDRA, event.text)

            event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                            text='Nein Danke!')]

        elif (event == fabula.SaysEvent(identifier=ID_KUNI,
                                        text='Ich habe eine Saite von einer Harfe.')
              and not self.lute_is_repaired):

            if 'lute_broken' not in self.host.rack.entity_dict.keys():

                event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                text='Vielleicht kann man die noch gebrauchen.')]

            else:

                event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                text='Ich habe ein Laute ohne Saiten. '
                                                     'Vielleicht können wir sie damit reparieren.')]

                self.lute_is_repaired = True

                self._remove_sentence(ID_CASSANDRA, 'Ich habe eine Laute. Leider ist sie kaputt.')
                self._remove_sentence(ID_KUNI, event.text)

                event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                text='Ich habe ein Laute ohne Saiten. '
                                                     'Vielleicht können wir sie damit reparieren.')]
```

```
1743
1744
1745                 messages += [fabula.Message([fabula.DropsEvent(ID_KUNI,
1746                                                self.host.rack.entity_dict['string_harp'],
1747                                                room.entity_locations[ID_KUNI])]),
1748                              fabula.Message([fabula.DropsEvent(ID_CASSANDRA,
1749                                                self.host.rack.entity_dict['lute_broken'],
1750                                                room.entity_locations[ID_KUNI])]),
1751                              fabula.Message([fabula.DeleteEvent(identifier='string_harp')]),
1752                              fabula.Message([fabula.SpawnEvent(fabula.Entity('lute',
1753                                                                fabula.ITEM,
1754                                                                True,
1755                                                                True,
1756                                                                {'image/png': fabula.Asset(uri='lute.png',
1757                                                                                           data=None),
1758                                                                 'audio/ogg': fabula.Asset(uri='lute.ogg',
1759                                                                                           data=None),
1760                                                                 'text/plain': fabula.Asset(uri='lute.txt',
1761                                                                                            data=None)}),
1762                                                room.entity_locations[ID_KUNI] + (room.identifier,))]),
1763                              fabula.Message([fabula.DeleteEvent(identifier='lute_broken')]),
1764                              fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
1765                                                text='Es hat funktioniert! '
1766                                                     'Jetzt haben wir eine spielbare Laute.')]),
1767                              fabula.Message([fabula.PicksUpEvent(identifier=ID_KUNI,
1768                                                item_identifier='lute')]),
1769                              fabula.Message([fabula.PerceptionEvent(identifier=ID_CASSANDRA,
1770                                                perception='Kuni hat die Laute eingesteckt.')])]
1771
1772         elif (event == fabula.SaysEvent(identifier=ID_CASSANDRA,
1773                                         text='Ich habe eine Laute. Leider ist sie kaputt.')
1774               and not self.lute_is_repaired):
1775
1776             if 'string_harp' not in self.host.rack.entity_dict.keys():
1777
1778                 event_list += [fabula.SaysEvent(identifier=ID_KUNI,
1779                                                 text='Vielleicht kann man die noch reparieren.')]
1780
1781             else:
1782
1783                 self.lute_is_repaired = True
1784
1785                 self._remove_sentence(ID_CASSANDRA, event.text)
1786                 self._remove_sentence(ID_KUNI, 'Ich habe eine Saite von einer Harfe.')
1787
1788                 event_list += [fabula.SaysEvent(identifier=ID_KUNI,
1789                                                 text='Ich habe ein Harfensaite. Vielleicht können wir sie damit reparieren')]
1790
1791                 messages += [fabula.Message([fabula.DropsEvent(ID_KUNI,
1792                                                self.host.rack.entity_dict['string_harp'],
1793                                                room.entity_locations[ID_CASSANDRA])]),
1794                              fabula.Message([fabula.DropsEvent(ID_CASSANDRA,
1795                                                self.host.rack.entity_dict['lute_broken'],
1796                                                room.entity_locations[ID_CASSANDRA])]),
1797                              fabula.Message([fabula.DeleteEvent(identifier='string_harp')]),
1798                              fabula.Message([fabula.SpawnEvent(fabula.Entity('lute',
1799                                                                fabula.ITEM,
1800                                                                True,
1801                                                                True,
1802                                                                {'image/png': fabula.Asset(uri='lute.png',
1803                                                                                           data=None),
1804                                                                 'audio/ogg': fabula.Asset(uri='lute.ogg',
1805                                                                                           data=None),
1806                                                                 'text/plain': fabula.Asset(uri='lute.txt',
1807                                                                                            data=None)}),
1808                                                room.entity_locations[ID_CASSANDRA] + (room.identifier,))]),
1809                              fabula.Message([fabula.DeleteEvent(identifier='lute_broken')]),
1810                              fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
1811                                                text='Es hat funktioniert! '
1812                                                     'Jetzt haben wir eine spielbare Laute.')]),
1813                              fabula.Message([fabula.PicksUpEvent(identifier=ID_CASSANDRA,
1814                                                item_identifier='lute')]),
1815                              fabula.Message([fabula.PerceptionEvent(identifier=ID_KUNI,
1816                                                perception='Cassandra hat die Laute eingesteckt.')])]
1817
1818
```

```
        elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                               item_identifier='lute_broken',
                                               target_identifier=ID_KUNI)
              and not self.lute_is_repaired):

            if 'string_harp' not in self.host.rack.entity_dict.keys():

                event_list += [fabula.SaysEvent(identifier=ID_KUNI,
                                                text='Ich besitze nichts, womit man die kaputte Laute reparieren könnte.')]
            else:

                self.lute_is_repaired = True

                self._remove_sentence(ID_CASSANDRA, 'Ich habe eine Laute. Leider ist sie kaputt.')
                self._remove_sentence(ID_KUNI, 'Ich habe eine Saite von einer Harfe.')

                event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA, perception='Kuni nimmt die kaputte Laute.')]

                messages += [fabula.Message([fabula.PerceptionEvent(identifier=ID_KUNI,
                                                                    perception='Cassandra gibt dir eine kaputte Laute.')]),
                             fabula.Message([fabula.DropsEvent(ID_CASSANDRA,
                                                               self.host.rack.entity_dict['lute_broken'],
                                                               room.entity_locations[ID_KUNI])]),
                             fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
                                                              text='Moment mal, ich hab da eine Idee!')]),
                             fabula.Message([fabula.DropsEvent(ID_KUNI,
                                                               self.host.rack.entity_dict['string_harp'],
                                                               room.entity_locations[ID_KUNI])]),
                             fabula.Message([fabula.DeleteEvent(identifier='string_harp')]),
                             fabula.Message([fabula.DeleteEvent(identifier='lute_broken')]),
                             fabula.Message([fabula.SpawnEvent(fabula.Entity('lute',
                                                                             fabula.ITEM,
                                                                             True,
                                                                             True,
                                                                             {'image/png': fabula.Asset(uri='lute.png',
                                                                                                        data=None),
                                                                              'audio/ogg': fabula.Asset(uri='lute.ogg',
                                                                                                        data=None),
                                                                              'text/plain': fabula.Asset(uri='lute.txt',
                                                                                                         data=None)}),
                                                               room.entity_locations[ID_KUNI] + (room.identifier,))]),
                             fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
                                                              text='Es hat funktioniert! Zusammen ergeben Harfensaite '
                                                                   'und kaputte Laute ein spielbares Instrument.')]),
                             fabula.Message([fabula.PicksUpEvent(identifier=ID_KUNI,
                                                                 item_identifier='lute')]),
                             fabula.Message([fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                                    perception='Kuni hat die Laute eingesteckt.')])]

        elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                               item_identifier='string_harp',
                                               target_identifier=ID_CASSANDRA)
              and not self.lute_is_repaired):

            if 'lute_broken' not in self.host.rack.entity_dict.keys():

                event_list += [fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                text='Ich besitze nichts, wofür man eine Saite gebrauchen könnte.')]
            else:

                self.lute_is_repaired = True

                self._remove_sentence(ID_CASSANDRA, 'Ich habe eine Laute. Leider ist sie kaputt.')
                self._remove_sentence(ID_KUNI, 'Ich habe eine Saite von einer Harfe.')

                event_list += [fabula.PerceptionEvent(identifier=ID_KUNI, perception='Cassandra nimmt die Seite der Harfe.')]

                messages += [fabula.Message([fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                                    perception='Kuni gibt dir die Saite einer Harfe.')]),
                             fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                              text='Moment mal, ich hab da eine Idee!')]),
                             fabula.Message([fabula.DropsEvent(ID_KUNI,
                                                               self.host.rack.entity_dict['string_harp'],
                                                               room.entity_locations[ID_CASSANDRA])]),
                             fabula.Message([fabula.DropsEvent(ID_CASSANDRA,
                                                               self.host.rack.entity_dict['lute_broken'],
```

```
                                                    room.entity_locations[ID_CASSANDRA])]),
                        fabula.Message([fabula.DeleteEvent(identifier='string_harp')]),
                        fabula.Message([fabula.SpawnEvent(fabula.Entity('lute',
                                                                        fabula.ITEM,
                                                                        True,
                                                                        True,
                                                                        {'image/png': fabula.Asset(uri='lute.png',
                                                                                                   data=None),
                                                                         'audio/ogg': fabula.Asset(uri='lute.ogg',
                                                                                                   data=None),
                                                                         'text/plain': fabula.Asset(uri='lute.txt',
                                                                                                    data=None)}),
                                                    room.entity_locations[ID_CASSANDRA] + (room.identifier,))]),
                        fabula.Message([fabula.DeleteEvent(identifier='lute_broken')]),
                        fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                         text='Es hat funktioniert jetzt haben wir eine '
                                                              'spielbare Laute.')]),
                        fabula.Message([fabula.PicksUpEvent(identifier=ID_CASSANDRA,
                                                            item_identifier='lute')]),
                        fabula.Message([fabula.PerceptionEvent(identifier=ID_KUNI,
                                                               perception='Cassandra hat die Laute eingesteckt.')])]

    elif event == fabula.TriesToLookAtEvent(identifier=ID_KUNI,
                                            target_identifier='guardian'):

        event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                              perception='Ein ziemlich großer, grobschlächtiger Kerl, '
                                                         'der den Eingang zum Zauberwald bewacht.')]

    elif event == fabula.TriesToLookAtEvent(identifier=ID_CASSANDRA,
                                            target_identifier='guardian'):

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Der Eingang zum Zauberwald wird von einem Wächter '
                                                         'bewacht, der sieht ganz schön ungemütlich aus.')]

    elif event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                          item_identifier='cake',
                                          target_identifier=ID_KUNI):

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Du reichst Kuni das Kuchenstück.')]

        messages += [fabula.Message([fabula.PerceptionEvent(identifier=ID_KUNI,
                                                            perception='Cassandra will dir ein Stück Kuchen geben.')]),
                     fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
                                                      text='Nein Danke Cassandra, ich mag nicht so viel Kuchen.')])]

    elif event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                          item_identifier='cake',
                                          target_identifier=ID_CASSANDRA):

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Ich bin doch auf Diät!')]

    elif event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                          item_identifier='cake',
                                          target_identifier='guardian'):

        if ID_KUNI in room.entity_locations.keys():

            self._remove_sentence(ID_CASSANDRA, 'Willst du ein Stück Kuchen?')

            event_list += [fabula.DropsEvent(ID_CASSANDRA,
                                             self.host.rack.entity_dict['cake'],
                                             room.entity_locations['guardian'])]

            messages += [fabula.Message([fabula.SaysEvent(identifier='guardian',
                                                          text='Ach das ist ja nett! Ich teile den Kuchen.')]),
                         fabula.Message([fabula.SaysEvent(identifier='guardian',
                                                          text='Mampf, mampf...')]),
                         fabula.Message([fabula.DeleteEvent(identifier='cake')]),
                         fabula.Message([fabula.SaysEvent(identifier='guardian',
                                                          text='Wisst ihr, ich war ja nicht immer Wächter des Waldweges.')]),
                         fabula.Message([fabula.SaysEvent(identifier='guardian',
                                                          text='Ich war leidenschaftlicher Minnesänger.')]),
```

```
                            fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                             text='Oh wirklich?')]),
                            fabula.Message([fabula.SaysEvent(identifier='guardian',
                                                             text='Ja! Und dann haben mich Räuber überfallen, mir alles Gold '
                                                                  'abgeknüpft und meine Laute geklaut!')]),
                            fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                             text='Oh nein, das tut mir aber leid!')]),
                            fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
                                                             text='Lässt du uns jetzt durch?')]),
                            fabula.Message([fabula.SaysEvent(identifier='guardian',
                                                             text='Nein, meine Aufgabe ist es den Weg zu bewachen.')]),
                            fabula.Message([fabula.SaysEvent(identifier='guardian',
                                                             text='Auch wenn ich mich sofort auf und davon machen würde, wenn '
                                                                  'ich wieder eine Laute hätte.')])]

        else:

            event_list += [fabula.SaysEvent(identifier='guardian',
                                            text='Oh lecker, aber das können wir nicht zu zweit essen!')]

    elif event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                          item_identifier='string_harp',
                                          target_identifier=ID_KUNI):

        event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                              perception='Was fang ich damit nur an?')]

    elif event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                          item_identifier='string_harp',
                                          target_identifier='guardian'):

        event_list += [fabula.SaysEvent(identifier='guardian',
                                        text='Was soll ich mit einer Saite ohne Klangkörper?')]

    elif event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                          item_identifier='lute_broken',
                                          target_identifier=ID_CASSANDRA):

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Was fang ich damit nur an?')]

    elif event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                          item_identifier='lute_broken',
                                          target_identifier='guardian'):

        event_list += [fabula.SaysEvent(identifier='guardian',
                                        text='Was soll ich mit einem Klangkörper ohne Saite?')]

    elif event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                          item_identifier='lute',
                                          target_identifier=ID_KUNI):

        event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                              perception='Kuni schüttelt den Kopf.')]

        messages += [fabula.Message([fabula.PerceptionEvent(identifier=ID_KUNI,
                                                            perception='Cassandra möchte dir die Laute überreichen.')]),
                     fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
                                                      text='Ach, trag du die ruhig mal.')])]

    elif event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                          item_identifier='lute',
                                          target_identifier=ID_CASSANDRA):

        event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                              perception='Cassandra schüttelt den Kopf.')]

        messages += [fabula.Message([fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                            perception='Kuni möchte dir die '
                                                                       'Laute überreichen.')]),
                     fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
                                                      text='Ich kann leider keine Saiteninstrumente spielen.')])]

    elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                           item_identifier='pry',
                                           target_identifier='teleport_room_cassandra')
```

```
                or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                                    item_identifier='pry',
                                                    target_identifier='teleport_room_kuni')
                or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                                    item_identifier='cake',
                                                    target_identifier='teleport_room_cassandra')
                or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                                    item_identifier='cake',
                                                    target_identifier='teleport_room_kuni')
                or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                                    item_identifier='lute_broken',
                                                    target_identifier='teleport_room_cassandra')
                or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                                    item_identifier='lute_broken',
                                                    target_identifier='teleport_room_kuni')
                or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                                    item_identifier='lute_broken',
                                                    target_identifier='teleport_room_cassandra')
                or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                                    item_identifier='lute_broken',
                                                    target_identifier='teleport_room_kuni')
                or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                                    item_identifier='lute',
                                                    target_identifier='teleport_room_kuni')
                or event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                                    item_identifier='lute',
                                                    target_identifier='teleport_room_cassandra')):

            event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
                                                  perception='Das möchte ich hier nicht ablegen.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                               item_identifier='string_harp',
                                               target_identifier='teleport_room_cassandra')
                or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                    item_identifier='string_harp',
                                                    target_identifier='teleport_room_kuni')
                or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                    item_identifier='lute',
                                                    target_identifier='teleport_room_kuni')
                or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                    item_identifier='lute',
                                                    target_identifier='teleport_room_cassandra')
                or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                    item_identifier='goblet',
                                                    target_identifier='teleport_room_cassandra')
                or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                    item_identifier='goblet',
                                                    target_identifier='teleport_room_kuni')
                or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                    item_identifier='goblet_filled',
                                                    target_identifier='teleport_room_cassandra')
                or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                    item_identifier='goblet_filled',
                                                    target_identifier='teleport_room_kuni')
                or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                    item_identifier='dew',
                                                    target_identifier='teleport_room_cassandra')
                or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                    item_identifier='dew',
                                                    target_identifier='teleport_room_kuni')
                or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                    item_identifier='string_harp',
                                                    target_identifier='teleport_room_cassandra')
                or event == fabula.TriesToDropEvent(identifier=ID_KUNI,
                                                    item_identifier='string_harp',
                                                    target_identifier='teleport_room_kuni')):

            event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
                                                  perception='Das möchte ich hier nicht ablegen.')]

        elif (event == fabula.TriesToDropEvent(identifier=ID_CASSANDRA,
                                               item_identifier='lute',
                                               target_identifier='guardian')):

            if ID_KUNI in room.entity_locations.keys():
```

```
2123
2124                event_list += [fabula.SaysEvent(identifier='guardian',
2125                                                text='Oh ihr seid die besten, ich kann endlich wieder muszierend umherziehen!')]
2126
2127                # The guardian runs off to find his luck in music making
2128                #
2129                messages += [fabula.Message([fabula.DropsEvent(ID_CASSANDRA,
2130                                                               self.host.rack.entity_dict['lute'],
2131                                                               room.entity_locations['guardian'])]),
2132                             fabula.Message([fabula.DeleteEvent('lute')]),
2133                             fabula.Message([fabula.MovesToEvent(identifier='guardian',
2134                                                                 location=(2, 2))]),
2135                             fabula.Message([fabula.MovesToEvent(identifier='guardian',
2136                                                                 location=(1, 2))]),
2137                             fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
2138                                                              text='Juhu der Weg ist frei!'),
2139                                             fabula.MovesToEvent(identifier='guardian',
2140                                                                 location=(0, 2))]),
2141                             fabula.Message([fabula.DeleteEvent('guardian')]),
2142                             fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
2143                                                              text='Wollen wir dann gemeinsam weiter ziehen?')]),
2144                             fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
2145                                                              text='Sehr gerne! Das hat ja bisher '
2146                                                                   'auch ausgezeichnet geklappt!')]),
2147                             fabula.Message([fabula.PerceptionEvent(identifier=ID_KUNI,
2148                                                                    perception='ENDE...')]),
2149                             fabula.Message([fabula.PerceptionEvent(identifier=ID_CASSANDRA,
2150                                                                    perception='ENDE...')])]
2151
2152            else: event_list += [fabula.PerceptionEvent(identifier=ID_CASSANDRA,
2153                                                        perception='Kuni möchte das bestimmt mit erleben...')]
2154
2155        elif event == fabula.TriesToDropEvent(identifier=ID_KUNI,
2156                                              item_identifier='lute',
2157                                              target_identifier='guardian'):
2158
2159            if ID_CASSANDRA in room.entity_locations.keys():
2160
2161                event_list += [fabula.SaysEvent(identifier='guardian',
2162                                                text='Oh ihr seid die besten, ich kann endlich wieder muszierend umherziehen!')]
2163
2164                # The guardian runs off to find his luck in music making
2165                #
2166                messages += [fabula.Message([fabula.DropsEvent(ID_KUNI,
2167                                                               self.host.rack.entity_dict['lute'],
2168                                                               room.entity_locations['guardian'])]),
2169                             fabula.Message([fabula.DeleteEvent('lute')]),
2170                             fabula.Message([fabula.MovesToEvent(identifier='guardian',
2171                                                                 location=(2, 2))]),
2172                             fabula.Message([fabula.MovesToEvent(identifier='guardian',
2173                                                                 location=(1, 2))]),
2174                             fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
2175                                                              text='Juhu der Weg ist frei!'),
2176                                             fabula.MovesToEvent(identifier='guardian',
2177                                                                 location=(0, 2))]),
2178                             fabula.Message([fabula.DeleteEvent('guardian')]),
2179                             fabula.Message([fabula.SaysEvent(identifier=ID_CASSANDRA,
2180                                                              text='Wollen wir dann gemeinsam weiter ziehen?')]),
2181                             fabula.Message([fabula.SaysEvent(identifier=ID_KUNI,
2182                                                              text='Sehr gerne! Das hat ja bisher auch ausgezeichnet '
2183                                                                   'geklappt!')]),
2184                             fabula.Message([fabula.PerceptionEvent(identifier=ID_KUNI,
2185                                                                    perception='ENDE...')]),
2186                             fabula.Message([fabula.PerceptionEvent(identifier=ID_CASSANDRA,
2187                                                                    perception='ENDE...')])]
2188
2189            else: event_list += [fabula.PerceptionEvent(identifier=ID_KUNI,
2190                                                        perception='Cassandra möchte bestimmt dabei sein...')]
2191
2192        self.queue_messages(*messages)
2193
2194        return event_list
2195
2196    def _load_room(self, client_id, room_name, location):
2197        """Load room for client with client_id corresponding to room_name.
2198
```

```
            location is a (x, y) tuple of where to spawn the Entity when the room
            already exists.

            Returns a list of events establishing the according room.
        """
        # Add EnterRoomEvent in front
        #
        event_list = [fabula.EnterRoomEvent(client_id, room_name)]

        # Does the room already exist?
        #
        if room_name in self.host.room_by_id.keys():

            fabula.LOGGER.debug("Sending existing room")

            event_list.extend(self.host._generate_room_rack_events(room_name))

            # In that case, we have to spawn the Entity manually.

            # The Entity should still exist while we're here.
            #
            entity = self.host.room_by_client[client_id].entity_dict[client_id]

            location = location + (room_name,)

            fabula.LOGGER.debug("Spawning '{}' at '{}'".format(client_id, location))

            event_list.append(fabula.SpawnEvent(entity, location))

        else:

            fabula.LOGGER.debug("Room does not yet exist, attempting to load from file")

            roomfile_name = room_name + ".floorplan"

            event_list.extend(fabula.plugins.serverside.load_room_from_file(roomfile_name,
                                                                            complete=False))

            # Has anything been added?
            #
            if len(event_list) == 1:

                # TODO: jump back to the level selection screen / room
                #
                fabula.LOGGER.critical("error opening file '{}'".format(roomfile_name))

                # TODO: exit properly
                #
                raise SystemExit

            # Remove all players that do not match client id
            #
            event_list = self._filter_spawn_events(event_list, client_id)

        # All events there, by one way or another.
        #
        event_list.append(fabula.RoomCompleteEvent())

        fabula.LOGGER.info("Room loaded.")

        return event_list

class ZWPygameUI(fabula.plugins.pygameui.PygameUserInterface):
    """A subclass of the default Pygame User Interface to implement custom behavior.
    """

    def get_connection_details(self, prompt_connector=True):
        """Copied from pygameui.get_connection_deatils and modified to display two buttons to start game as kuni or cassandra.
        """

        fabula.LOGGER.debug("called")

        # Display the splash screen
        #
        try:
            file = self.assets.fetch("splash.png")
```

```
2275
2276                surface = pygame.image.load(file)
2277
2278                file.close()
2279
2280                # Convert to internal format suitable for blitting.
2281                # Not using convert_alpha(), no RGBA support for splash.
2282                #
2283                surface = surface.convert()
2284
2285                fabula.LOGGER.debug("displaying splash.png")
2286
2287                self.window.image = surface
2288
2289            except:
2290                fabula.LOGGER.warning("splash.png not found, not displaying splash screen")
2291
2292            self.display_single_frame()
2293
2294            # Use a container so we can access it from an ad-hoc function.
2295            # That's Python.
2296            #
2297            container_dict = {"login_name" : None,
2298                              "connector" : None}
2299
2300            def login(buttonplane):
2301
2302                # Construct tuple with default IP
2303                #
2304                container_dict["connector"] = "127.0.0.1"
2305
2306                buttonplane.parent.destroy()
2307
2308                return
2309
2310            def login_kuni_callback(buttonplane):
2311
2312                container_dict["login_name"] = ID_KUNI
2313
2314                login(buttonplane)
2315
2316            def login_cassandra_callback(buttonplane):
2317
2318                container_dict["login_name"] = ID_CASSANDRA
2319
2320                login(buttonplane)
2321
2322            container = PLANES.gui.tmb.TMBContainer("get_connection_details",
2323                                                    PLANES.gui.tmb.C_256_STYLE,
2324                                                    padding=5)
2325
2326            # Login buttons (label, width, callback[, style])
2327            #
2328            button_cassandra = PLANES.gui.lmr.LMRButton("Als Fee Cassandra spielen",
2329                                                        150,
2330                                                        login_cassandra_callback)
2331            container.sub(button_cassandra)
2332
2333            button_kuni = PLANES.gui.lmr.LMRButton("Als Gnom Kuni spielen",
2334                                                   150,
2335                                                   login_kuni_callback)
2336            container.sub(button_kuni)
2337
2338            # Optional connector prompt
2339            #
2340            if prompt_connector:
2341
2342                container.sub(PLANES.gui.Label("connector_caption",
2343                                               "Server IP",
2344                                               pygame.Rect((0, 0),
2345                                                           (150, 30)),
2346                                               background_color=(128, 128, 128, 0)))
2347
2348                container.sub(PLANES.gui.TextBox("connector",
2349                                                 pygame.Rect((0, 0),
2350                                                             (150, 30)),
```

```
                                            return_callback=None))

            # Provide a default
            #
            container.connector.text = "127.0.0.1"

            # Upon clicked, register the TextBox for keyboard input
            #
            container.connector.left_click_callback = lambda plane : self.window.key_sensitive(plane)

        container.rect.center = self.window.rect.center

        self.window.sub(container)

        while container_dict["login_name"] is None:

            self.display_single_frame()

            self.collect_player_input()

            if self.exit_requested:

                # Return anything to get out of the loop
                #
                container_dict["login_name"] = "exit requested"

        # We are done with the splash screen if there was one. Now make sure
        # that there is no image for the window Plane left that would invisibly
        # be blitted below room and inventory planes.
        #
        self.window.image = self.window.display

        # Return connector (ip, port) using the default Fabula
        # port 0xfab == 4011.
        #
        return (container_dict["login_name"],
                (container_dict["connector"], 4011))

class ZWAudioUI(fabula.plugins.audioui.AudioUserInterface):
    """A subclass of the default AudioUserInterface to implement custom behavior.
    """

    VOICE_SPECIFICATIONS = [str(b'german'),
                            str(b'German'),
                            str(b'de'),
                            str(b'De'),
                            str(b'DE'),
                            'german',
                            'German',
                            'de',
                            'De',
                            'DE']

    def __init__(self, assets, framerate, host):

        # Call base class
        #
        fabula.plugins.audioui.AudioUserInterface.__init__(self, assets, framerate, host)

        # Set speech rate of TTS
        #
        rate = 100
        print("__init__(): rate == {}".format(rate))
        self.tts_engine.setProperty('rate', rate)

        # Overwrite english default passive event messages
        #
        self.in_inventory_msg = "{} ins Inventar aufgenommen"
        self.taker_in_room_msg = "{} ist im Raum"
        self.taker_left_room_msg = "{} hat den Raum verlassen"

        return

    def get_connection_details(self, prompt_connector=True):
        """Read out an introductory text and make an AudioMenuList as main Menu.
           On the main menu the player can choose to start game as kuni or
```

```
2427                cassandra and enter an IP address. The default IP address is
2428                127.0.0.1.
2429            """
2430
2431            fabula.LOGGER.debug("called")
2432
2433            self.connection_details_complete = False
2434
2435            # Use a container so we can access it from an ad-hoc function.
2436            # That's Python.
2437            #
2438            container_dict = {'login_name' : 'player',
2439                              'connector' : '127.0.0.1'}
2440
2441            self.tts_engine.say("Willkommen bei dem Abenteuerspiel Zauberwald! "
2442                                "Wähle mit den Pfeiltasten hoch und runter aus, ob du die Fee Cassandra oder "
2443                                "den Gnom Kuni spielen möchtest. Bestätige deine Auswahl mit der Pfeiltaste rechts.", 'intro')
2444            self.tts_engine.runAndWait()
2445
2446            # Connection details menu
2447            #
2448            connection_details_entries = ['Als Cassandra spielen',
2449                                          'Als Kuni spielen']
2450
2451            # Optional connector prompt
2452            #
2453            if prompt_connector:
2454
2455                connection_details_entries.append('Ei Pi Adresse')
2456                connection_details_entries.append('Eine neue Ei Pi Adresse eingeben')
2457
2458                container_dict['connector'] = '127.0.0.1'
2459
2460                audio_text_field_ip_address = AudioTextField('Ei Pi Adresse',
2461                                                             self.tts_engine)
2462
2463            connection_details_entries.append('Beenden')
2464
2465            def callback_on_entry_selected(connection_details_menu):
2466                """Callback function for entry selected in self.connection_details_menu.
2467                """
2468
2469                selected_entry = connection_details_menu.list[connection_details_menu.list_index]
2470
2471                fabula.LOGGER.debug("Entry selected: {}".format(selected_entry))
2472
2473                if selected_entry == 'Als Cassandra spielen':
2474
2475                    self.tts_engine.say("Spiel startet als Cassandra")
2476                    self.tts_engine.runAndWait()
2477                    container_dict['login_name'] = 'cassandra'
2478                    self.connection_details_complete = True
2479
2480                elif selected_entry == 'Als Kuni spielen':
2481
2482                    self.tts_engine.say("Spiel startet als Kuni")
2483                    self.tts_engine.runAndWait()
2484                    container_dict['login_name'] = 'kuni'
2485                    self.connection_details_complete = True
2486
2487                elif selected_entry == 'Ei Pi Adresse':
2488
2489                    self.tts_engine.say(container_dict['connector'])
2490                    self.tts_engine.runAndWait()
2491
2492                elif selected_entry == 'Eine neue Ei Pi Adresse eingeben':
2493
2494                    audio_text_field_ip_address.focus_audio_text_field()
2495
2496                    container_dict['connector'] = audio_text_field_ip_address.process_key_input()
2497
2498                elif selected_entry == 'Beenden':
2499
2500                    self.connection_details_complete = True
2501                    container_dict['login_name'] = 'exit requested'
2502
```

```
2503                return
2504
2505            # Sound to be played when entering menu
2506            #
2507            file = self.assets.fetch('fabula_main_menu.ogg')
2508            main_menu_sound = pygame.mixer.Sound(file)
2509
2510            self.connection_details_menu = TextMenuList(connection_details_entries,
2511                                                        main_menu_sound,
2512                                                        None,
2513                                                        self.channel_system,
2514                                                        callback_on_entry_selected,
2515                                                        None,
2516                                                        self.tts_engine)
2517
2518            # Transition to connection-details-input-state
2519            #
2520            self._push_input_state_on_stack(self.INPUT_STATE_CONNECTION_DETAILS)
2521
2522            while not self.connection_details_complete:
2523
2524                self.collect_player_input()
2525
2526            return (container_dict['login_name'],
2527                    (container_dict['connector'], 4011))
2528
2529        def _set_up_help_menu(self):
2530            """Sets up the help menu list.
2531            """
2532
2533            # Help Menu for Zauberwald controls
2534            #
2535            help_menu_entries_values = {'Taste q' : 'Inventar öffnen',
2536                                        'Taste e' : 'Interaktionsmenü öffnen',
2537                                        'Taste x' : 'Spielerposition vorlesen',
2538                                        'Taste y': 'Liste mit allen Objekten im Raum',
2539                                        'Bewegung': 'w a s d Tasten',
2540                                        'Taste w' : 'Um ein Feld nach oben bewegen',
2541                                        'Taste a' : 'Um ein Feld nach links bewegen',
2542                                        'Taste s' : 'Um ein Feld nach unten bewegen',
2543                                        'Taste d' : 'Um ein Feld nach rechts bewegen'}
2544
2545            help_menu_entries = ['Taste q',
2546                                 'Taste e',
2547                                 'Taste x',
2548                                 'Taste y',
2549                                 'Bewegung',
2550                                 'Taste w',
2551                                 'Taste a',
2552                                 'Taste s',
2553                                 'Taste d']
2554
2555            def callback_on_help_entry_selected(help_menu):
2556
2557                selected_entry = help_menu.list[help_menu.list_index]
2558
2559                # Let the TTS engine read the help information at current index
2560                #
2561                help_menu.tts_engine.say(help_menu_entries_values[selected_entry])
2562                help_menu.tts_engine.runAndWait()
2563
2564                return
2565
2566            def callback_on_help_menu_exit(help_menu):
2567
2568                # Going back to previous input state
2569                #
2570                self._pop_input_state_from_stack()
2571
2572                return
2573
2574            # Sound to be played when help menu is opened
2575            #
2576            file = self.assets.fetch('help_menu.ogg')
2577            help_menu_sound = pygame.mixer.Sound(file)
2578
```

```
2579            help_menu = TextMenuList(help_menu_entries,
2580                                     help_menu_sound,
2581                                     None,
2582                                     self.channel_system,
2583                                     callback_on_help_entry_selected,
2584                                     callback_on_help_menu_exit,
2585                                     self.tts_engine)
2586
2587            return help_menu
```

C.2 start_zauberwald_server.py

```
1   import sys
2
3   # Add current and parent directory. One of them is supposed to contain the fabula
4   # package.
5   #
6   sys.path.append("../")
7   sys.path.append("./")
8
9   import fabula.plugins.serverside
10  import fabula.run
11  import fabula.interfaces.python_tcp
12  from optparse import OptionParser
13
14  from zauberwald import ZWServerPlugin
15
16  def main():
17      """Run the zauberwald server. Can optionally accept an ip address from the
18         command line.
19      """
20
21      parser = OptionParser()
22      parser.add_option("-a", dest="ip_address", default="127.0.0.1",
23                        help="IP address of the server (default is 127.0.0.1)")
24
25      (options, args) = parser.parse_args()
26      print(options.ip_address)
27
28      # Setting up server interface
29      #
30      server_interface = fabula.interfaces.python_tcp.TCPServerInterface()
31
32      app_server = fabula.run.App(timeout=0)
33      app_server.server_plugin_class = ZWServerPlugin
34      app_server.run_server(30, server_interface, 0.8,  options.ip_address, True)
35
36      return
37
38  if __name__ == "__main__":
39
40      main()
```

C.3 start_zauberwald_pygame_client.py

```
import sys

# Add current and parent directory. One of them is supposed to contain the fabula
# package.
#
sys.path.append("../")
sys.path.append("./")

import fabula.plugins.serverside
import fabula.run
import fabula.interfaces.python_tcp

from zauberwald import ZWUI

def main():
    """Run a zauberwald pygame client.
    """

    # Setting up client interface
    #
    client_interface_player = fabula.interfaces.python_tcp.TCPClientInterface()

    app_client = fabula.run.App(timeout=0)
    app_client.user_interface_class = ZWUI
    app_client.run_client(30, client_interface_player)

    return

if __name__ == "__main__":

    main()
```

C.4 start_zauberwald_audio_client.py

```
import sys

# Add current and parent directory. One of them is supposed to contain the fabula
# package.
#
sys.path.append("../")
sys.path.append("./")

import fabula.plugins.serverside
import fabula.run
import fabula.interfaces.python_tcp

from zauberwald import ZWAudioUI

def main():
    """Run a zauberwald audio client.
    """

    # Setting up client interface
    #
    client_interface_player = fabula.interfaces.python_tcp.TCPClientInterface()

    app_client = fabula.run.App(timeout=0)
    app_client.user_interface_class = ZWAudioUI
    app_client.run_client(30, client_interface_player)

    return

if __name__ == "__main__":

    main()
```

C.5 audioui.py

```
"""Audio Fabula User Interface using Pygame and Pyttsx

   Copyright 2013 Linda Kerkhoff
"""

# This file is part of Fabula.
#
# Fabula is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# Fabula is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with Fabula.  If not, see <http://www.gnu.org/licenses/>.

# work started on 12. Sep 2013

import sys

import fabula.plugins.ui
import pygame
import datetime
import os
import unicodedata
import pyttsx
# For cx_Freeze
import pyttsx.drivers.sapi5

# Hardwired screen size.
#
SCREENSIZE = (400, 300)

ENTRY_UP_KEYS = [pygame.K_UP]
ENTRY_DOWN_KEYS = [pygame.K_DOWN]
SELECT_ENTRY_KEYS = [pygame.K_RIGHT, pygame.K_RETURN]
EXIT_AUDIO_MENU_LIST_KEYS = [pygame.K_LEFT]
ENTER_KEYS = [pygame.K_RETURN]

class AudioMenuList():
    """An AudioMenuList is an acoustic widget for a list menu. It can be opened
       and while it is opened scrolling with key-events through its entries is
       enabled as well as select an entry or exit the menu by pressing a key.

       A callback method can be provided to handle the selection of an entry.
       A second callback method is called if the list is exited.

       If an AudioMenuList with an empty list is opened its empty_sound is
       played and the callback_on_exit method is called.

       AudioMenuList.list
           A list containing all entries of the menu.

       AudioMenuList.list_index
           Current index to the list of the AudioMenuList. Initially set to
           None.

       AudioMenuList.menu_sound
           Supposed to be a pygame.mixer.Sound and is played as title when the
           list opens.

       AudioMenuList.empty_sound
           Supposed to be a pygame.mixer.Sound and is played when the list opens
           and has no entries.

       AudioMenuList.channel
           Supposed to be a pygame.mixer.channel and is used to play back all
           sounds of the list.

       AudioMenuList.callback_on_entry_selected
```

```
            Callback function to handle the selection of an entry.

        AudioMenuList.callback_on_exit
            Callback function to handle exiting the list without a previous
            selection.

        To get acoustic feedback a specific subclass of AudioMenuList is
        needed to provide a way of creating an acoustic version of all the list
        entries.
    """

    def __init__(self, list, menu_sound, empty_sound, channel, callback_on_entry_selected, callback_on_exit):
        """Initialize with given parameters and the list_index set to None.
        """
        self.list = list
        self.list_index = None
        self.menu_sound = menu_sound
        self.empty_sound = empty_sound
        self.channel = channel
        self.callback_on_entry_selected = callback_on_entry_selected
        self.callback_on_exit = callback_on_exit
        return

    def try_to_open_menu_list(self):
        """Tries to open the AudioMenuList. If the list is empty it plays the
           empty_sound, calls callback_on_exit and returns False.
           If the list is not empty it calls open_menu_list and returns True.
        """

        has_entries = False

        # If the list is empty just leave the menu and play the empty sound
        #
        if len(self.list) <= 0:

            if self.empty_sound is not None:

                # Play back menu lists empty sound
                #
                self.channel.play(self.empty_sound)

                # Wait till sound is complete
                #
                while self.channel.get_busy():
                    pass

            if self.callback_on_exit is not None:

                self.callback_on_exit(self)

        else:

            has_entries = True
            self.open_menu_list()

        return has_entries

    def open_menu_list(self):
        """Opens the AudioMenuList by playing its menu_sound as title and sets the index to the first entry.
        """

        if self.menu_sound is not None:

            # Play back menu lists name
            #
            self.channel.play(self.menu_sound)

            # Wait till sound is complete
            #
            while self.channel.get_busy():
                pass

        self.list_index = 0

        return
```

```
151         def process_key_input(self, key):
152             """Processes key input and performs the corresponding action on the AudioMenuList.
153                Actions are the selection of an entry, scrolling entries up and down
154                and exiting the list.
155             """
156 
157             # Scrolling through entries with up and down arrow keys
158             #
159             if key in ENTRY_DOWN_KEYS or key in ENTRY_UP_KEYS:
160 
161                 self.scroll_list(key)
162 
163             # Call callback function for entry selected
164             #
165             elif (key in SELECT_ENTRY_KEYS
166                   and self.callback_on_entry_selected is not None):
167 
168                 self.callback_on_entry_selected(self)
169 
170             # Call callback function for exit menu list
171             #
172             elif (key in EXIT_AUDIO_MENU_LIST_KEYS
173                   and self.callback_on_exit is not None):
174 
175                 self.callback_on_exit(self)
176 
177             return
178 
179         def scroll_list(self, key):
180             """Scrolls entries by decreasing or increasing the list_index.
181             """
182 
183             # Scrolling through entries with up and down arrow keys
184             #
185             if key in ENTRY_UP_KEYS:
186 
187                 self.list_index -= 1
188 
189             elif key in ENTRY_DOWN_KEYS:
190 
191                 self.list_index += 1
192 
193             # Keep the index in bounds of length of list
194             #
195             self.list_index %= len(self.list)
196 
197             return
198 
199     class SoundMenuList(AudioMenuList):
200         """A SoundMenuList is a specific AudioMenuList whose list is supposed to contain pygame.mixer.Sounds.
201         """
202 
203         def __init__(self, list, menu_sound, empty_sound, channel, callback_on_entry_selected, callback_on_exit):
204             """Initialize with given parameters.
205             """
206 
207             # Call base class
208             #
209             AudioMenuList.__init__(self,
210                                    list,
211                                    menu_sound,
212                                    empty_sound,
213                                    channel,
214                                    callback_on_entry_selected,
215                                    callback_on_exit)
216             return
217 
218         def open_menu_list(self):
219             """Opens the SoundMenuList and plays the first sound at index 0.
220             """
221 
222             # Call base class
223             #
224             AudioMenuList.open_menu_list(self)
225 
226             # Play back first entry
```

```
            #
            self.channel.play(self.list[self.list_index])

            return

        def scroll_list(self, key):
            """Scrolls entries by decreasing or increasing the list_index and plays the sound in the list at the current index.
            """

            # Call base class
            #
            AudioMenuList.scroll_list(self, key)

            # Play back current entry
            #
            self.channel.play(self.list[self.list_index])

            return

    class EntityMenuList(AudioMenuList):
        """An EntityMenuList is a specific AudioMenuList whose list is supposed to contain fabula.Entity information.
           This information is supposed to be a tuple of Entity, and
           Entity.identifier or a location.

           It can be used to provide entity selection for fabula.Events, for which
           the second value of the tuple is used.

           When scrolling the menu the Entity's identifier or the text/plain asset
           of the Entity is read out by a text-to-speech engine.

           Additional attributes:

           EntityMenuList.tts_engine
               A pyttsx-engine to transform text to speech.
        """

        # Parameter list must contain a tuple of (Entity, Entity.identifier) if the Entity is in the rack
        # or (entity, location) if the Entity is in the room
        def __init__(self, list, menu_sound, empty_sound, channel, callback_on_entry_selected, callback_on_exit, tts_engine):
            """Initialize with given parameters.
            """

            # Call base class
            #
            AudioMenuList.__init__(self, list, menu_sound, empty_sound, channel, callback_on_entry_selected, callback_on_exit)
            self.tts_engine = tts_engine

            return

        def open_menu_list(self):
            """Opens the EntityMenuList and reads out the identifier or text/plain asset of the first Entity in the list.
            """

            # Call base class
            #
            AudioMenuList.open_menu_list(self)

            # Entries are a tuple like (Entity, <Entity.identifier|location>)
            #
            entity = self.list[self.list_index][0]

            entity_text = entity.identifier

            # Check if entity provides a name
            #
            if ('text/plain' in entity.assets.keys()
                and entity.assets['text/plain'].data is not None):

                entity_text = entity.assets['text/plain'].data[0]

            # Let the TTS engine read the entry at index
            #
            self.tts_engine.say(entity_text)
            self.tts_engine.runAndWait()

            return
```

```
303
304         def scroll_list(self, key):
305             """Scrolls entries by decreasing or increasing the list_index and reads out the currently selected Entity.
306                The identifier or text/plain asset of the selected Entity is read
307                out.
308             """
309
310             # Call base class
311             #
312             AudioMenuList.scroll_list(self, key)
313
314             entity = self.list[self.list_index][0]
315
316             entity_text = entity.identifier
317
318             # Check if entity provides a name
319             #
320             if ('text/plain' in entity.assets.keys()
321                 and entity.assets['text/plain'].data is not None):
322
323                 entity_text = entity.assets['text/plain'].data[0]
324
325             # Let the TTS engine read the entry at index
326             #
327             self.tts_engine.say(entity_text)
328             self.tts_engine.runAndWait()
329
330             return
331
332     class TextMenuList(AudioMenuList):
333         """A TextMenuList is a specific AudioMenuList whose list is supposed to contain strings.
334
335            When scrolling the menu the string entry is read out by a text-to-speech
336            engine.
337
338            Additional attributes:
339
340            TextMenuList.tts_engine
341                A pyttsx-engine to transform text to speech.
342         """
343
344         def __init__(self, list, menu_sound, empty_sound, channel, callback_on_entry_selected, callback_on_exit, tts_engine):
345             """Initialize with given parameters.
346             """
347
348             # Call base class
349             #
350             AudioMenuList.__init__(self, list, menu_sound, empty_sound, channel, callback_on_entry_selected, callback_on_exit)
351             self.tts_engine = tts_engine
352
353             return
354
355         def open_menu_list(self):
356             """Opens the TextMenuList and reads out the first entry in the list.
357             """
358
359             # Call base class
360             #
361             AudioMenuList.open_menu_list(self)
362
363             # Let the TTS engine read the current entry
364             #
365             self.tts_engine.say(self.list[self.list_index])
366             self.tts_engine.runAndWait()
367
368             return
369
370         def scroll_list(self, key):
371             """Scrolls entries by decreasing or increasing the list_index and reads out the the current entry of the list.
372             """
373
374             # Call base class
375             #
376             AudioMenuList.scroll_list(self, key)
377
378             # Let the TTS engine read the current entry
```

```
            #
            self.tts_engine.say(self.list[self.list_index])
            self.tts_engine.runAndWait()

            return

    class AudioTextField():
        """An AudioTextField is an acoustic widget for a text input field. It
           collects key input and its text-to-speech engine reads out the entered
           key. The input ends if one of the defined ENTER_KEYS is pressed and the
           text-to-speech engine reads out the complete collected input string. When
           an AudioTextField gains focus its label is read out.

           Unfortunately the text-to-speech engine is not able to read single
           periods. But this may be considered as normal text-to-speech behavior.

           AudioTextField.label
               Label of the text field. Read out if the field gains focus.

           AudioTextField.tts_engine
               A pyttsx-engine to transform text to speech.

           AudioTextField.input_text
               A string containing the current input string. When a valid key is
               pressed it will be appended to input_text. By default empty string.

        """

        def __init__(self, label, tts_engine, input_text=""):
            """Initialize with given parameters.
            """

            self.label = label
            self.tts_engine = tts_engine
            self.input_text = input_text

            return

        def focus_audio_text_field(self):
            """Empties the content of the AudioTextField an reads out the label.
            """

            self.input_text = ''
            self.tts_engine.say(self.label)
            self.tts_engine.runAndWait()

            return

        def process_key_input(self):
            """Processes key input while active. Any Unicode symbols of category
               letter, number, punctuation, symbols and separators will be appended
               to input_text. Backspace key will delete the last entered symbol and
               any of the defined ENTER_KEYS will end the input processing and read
               out the entered input_text and return it.

               Returns content of AudioTextField.input_text
            """

            gets_input = True

            while gets_input:

                events = pygame.event.get()

                for event in events:

                    if event.type == pygame.KEYDOWN:

                        if event.key == pygame.K_BACKSPACE:

                            self.input_text = self.input_text[0:-1]

                        elif event.key in ENTER_KEYS:

                            gets_input = False
```

```
                    # Copied from planes.gui.keydown(self, keydown_event)
                    # We can not use Python 3's str.isprintable() for Python 2 compatibility
                    # reasons. As a workaround, we check the Unicode category of the input.
                    # See
                    # http://www.unicode.org/Public/5.1.0/ucd/UCD.html#General_Category_Values
                    #
                    elif (len(event.unicode)
                          and unicodedata.category(event.unicode)[0] in "LNPSZ"):

                        self.tts_engine.say(event.unicode)
                        self.tts_engine.runAndWait()
                        self.input_text = self.input_text + event.unicode

        self.tts_engine.say(self.input_text)
        self.tts_engine.runAndWait()

        return self.input_text

class AudioEntity(fabula.Entity):
    """Pygame-aware subclass of Entity to be used in AudioUserInterface.

       AudioEntity.assets["sound/ogg"].data is supposed to be an instance of
       pygame.mixer.Sound and contain movement sound e.g. step sounds.

       If the AudioEntity is moving it plays its moving sound.
    """

    def process_MovesToEvent(self, event):
        """Plays back the movement sounds of the AudioEntity.
        """

        if (event.identifier is self.identifier
            and self.assets['audio/ogg'].data is not None
            and 'steps' in self.assets['audio/ogg'].uri):

            # Play footsteps
            #
            self.assets['audio/ogg'].data.play()

        return

class AudioUserInterface(fabula.plugins.ui.UserInterface):
    """This is an auditive implementation of an UserInterface for the Fabula Client.
       It uses Pygame to process input and a text-to-speech engine to read.
       The interaction concept is based on Audio Widgets which represent the
       user interface and give acoustic feedback on player input.

       Entities in the room are playing their sounds if they are in the
       surrounding positions of the PLAYER of this user interface.

       Additional attributes:

       AudioUserInterface.input_dict['keyboard'] = True
           AudioUserInterface used keyboard events to collect player input.

       AudioUserInterface.input_state_stack
           A list representing the stack of input states. The current state is
           the topmost. INPUT_STATE_IN_ROOM is the default in-game input state,
           therefore it is always present as the bottommost state on the stack.
           Whenever the input state is switched to in room, the stack is reset
           to contain only INPUT_STATE_IN_ROOM.

       AudioUserInterface.tts_engine
           Text-to-speech engine, by default None.

       AudioUserInterface.surrounding_position_channels
           Saves for each of the surrounding positions and the player position
           itself a tuple if a pygame.mixer.channel, a volume for left and a
           volume for right speaker to play back entities of surrounding fields
           with a specific spatial volume (result of different volumes for
           left and right speaker).

       AudioUserInterface.channel_system
           pygame.mixer.channel to play back menus and system events such as
           loading_sound.
```

```
AudioUserInterface.channel_ambience
    pygame.mixer.channel to play back the ambience sounds of a room.

AudioUserInterface.room_ambience_sound
    pygame.mixer.Sound to be played when entering in room input state.

AudioUserInterface.loading_sound
    pygame.mixer.Sound to be played when a room is loading.

AudioUserInterface.loading_complete_sound
    pygame.mixer.Sound to be played when loading a room is complete.

AudioUserInterface.attempt_failed_sound
    pygame.mixer.Sound to be played if an AttemptFailedEvent is sent to
    this user interface.

AudioUserInterface.connection_details_complete
    Flag to set if connection details are complete.

AudioUserInterface.interaction_sound_menu
    SoundMenuList for interaction menu. List contains a
    pygame.mixer.Sound for every interaction.

AudioUserInterface.select_item_menu
    EntityMenuList for item selection for a 'use' interaction. List
    contains tuples of (Entity, <Entity.identifier|location>). The
    second value depends on the origin of the Entity. If from the rack,
    Entity.identifier is present. If from the room, a location is
    present.

AudioUserInterface.select_target_menu
    EntityMenuList for item selection for a 'lockAt', 'pickUp', 'talkTo'
    or 'use' interaction. List contains tuples of
    (Entity, <Entity.identifier|location>). The second value depends on
    the origin of the Entity. If from the rack, Entity.identifier is
    present. If from the room, a location is present.

AudioUserInterface.inventory_menu
    EntityMenuList for all items in the inventory. List contains tuples
    of (Entity, Entity.identifier).

AudioUserInterface.select_sentence_menu
    TextMenuList for all sentences if user interface has to process a
    CanSpeakEvent. List contains all possible sentences.

AudioUserInterface.see_room_menu
    EntityMenuList for all entities in the room. List contains tuples
    of (Entity, location).

AudioUserInterface.help_menu
    TextMenuList for keyboard layout help menu. List contains keys and
    on selection the corresponding function is read out.

AudioUserInterface.taker_in_room_dict
    A dict to store the Entity objects of the other players that are in
    the same room as the client. Keys are supposed to be
    Entity.identifier.

AudioUserInterface.in_inventory_msg
    A feedback message given to the user when an item is added to the
    inventory. Initially it is '{} in inventory' where the braces are
    replaced by the items name, or identifier if it does not provide one.

AudioUserInterface.taker_in__room_msg
    A feedback message given to the user when another player already is
    in or entered the room. Initially it is '{} is in the room' where the
    braces are replaced by the players name.

AudioUserInterface.taker_left_room_msg
    A feedback message given to the user when another player left the
    room. Initially it is '{} left the room' where the braces are
    replaced by the players name.

The following variables map the different input states of the client to
numeric values. To check in collect_player_input which menu the input
should be delegated to.
```

```
607
608             AudioUserInterface.INPUT_STATE_IN_ROOM
609                 To collect input when player is in room.
610
611             AudioUserInterface.INPUT_STATE_INTERACTION
612                 To collect input when player is in interaction menu.
613
614             AudioUserInterface.INPUT_STATE_SELECT_TARGET
615                 To collect input when player is in select target menu.
616
617             AudioUserInterface.INPUT_STATE_SELECT_ITEM
618                 To collect input when player is in select item menu.
619
620             AudioUserInterface.INPUT_STATE_SEE_RACK
621                 Enables to scroll the rack to hear what it contains.
622
623             AudioUserInterface.INPUT_STATE_CAN_SPEEK
624                 Enables to select a sentence to say.
625
626             AudioUserInterface.INPUT_STATE_SEE_ROOM
627                 Enables to select entries in the room.
628
629             AudioUserInterface.INPUT_STATE_CONNECTION_DETAILS
630                 Enables to select entries for connection detail input.
631
632             AudioUserInterface.INPUT_STATE_HELP_MENU
633                 Enables to select entries of the help menu.
634
635             The following attributes map an interaction to numeric values to decide
636             which interaction is selected. Used in interaction_sound_menu and
637             select_target_menu to sent the event corresponding to the current
638             interaction.
639
640             AudioUserInterface.ATTEMPT_LOOK_AT
641                 For a look at interaction.
642
643             AudioUserInterface.ATTEMPT_TALK_TO
644                 For a talk to interaction.
645
646             AudioUserInterface.ATTEMPT_PICK_UP
647                 For a pick up interaction.
648
649             AudioUserInterface.ATTEMPT_USE
650                 For an use interaction.
651
652             AudioUserInterface.CANCEL
653                 For cancel the interaction menu.
654         """
655
656         VOICE_SPECIFICATIONS = [str(b'english'),
657                                 str(b'English'),
658                                 str(b'en'),
659                                 str(b'En'),
660                                 str(b'EN'),
661                                 'english',
662                                 'en',
663                                 'English',
664                                 'En',
665                                 'EN']
666
667         # 'Constants' for the different input states:
668         #
669         INPUT_STATE_IN_ROOM = 0
670         INPUT_STATE_INTERACTION = 1
671         INPUT_STATE_SELECT_TARGET = 2
672         INPUT_STATE_SELECT_ITEM = 3
673         INPUT_STATE_SEE_RACK = 4
674         INPUT_STATE_CAN_SPEEK = 5
675         INPUT_STATE_SEE_ROOM = 6
676         INPUT_STATE_CONNECTION_DETAILS = 7
677         INPUT_STATE_HELP_MENU = 8
678
679         # 'Constants' for the different attempt actions:
680         #
681         ATTEMPT_LOOK_AT = 0
682         ATTEMPT_TALK_TO = 1
```

```
683         ATTEMPT_PICK_UP = 2
684         ATTEMPT_USE = 3
685         ATTEMPT_MANIPULATE = 4
686         CANCEL = 5
687 
688         def __init__(self, assets, framerate, host):
689             """Initializes the AudioUserInterface.
690 
691                assets must be an instance of fabula.Assets or a subclass.
692             """
693 
694             # Call base class
695             #
696             fabula.plugins.ui.UserInterface.__init__(self,
697                                                      assets,
698                                                      framerate,
699                                                      host)
700 
701             fabula.LOGGER.debug("called")
702 
703             self.input_dict['keyboard'] = True
704 
705             fabula.LOGGER.debug("initializing pygame")
706 
707             pygame.init()
708             pygame.mixer.init()
709             pygame.mixer.set_num_channels(11)
710 
711             # Center window. Hint from the pygame-users mailing list.
712             #
713             os.environ['SDL_VIDEO_CENTERED'] = '1'
714 
715             # Open a window to collect pygame events and add a caption.
716             #
717             pygame.display.set_mode(SCREENSIZE)
718             pygame.display.set_caption("fabula audio client")
719 
720             # Initialize input state stack
721             #
722             self.input_state_stack = []
723 
724 
725             # Set up text-to-speech engine
726             #
727             self.tts_engine = pyttsx.init()
728 
729             # Variable for the main voice
730             #
731             main_voice_id = None
732 
733             # Setting up voices for text-to-speech engine
734             #
735             for voice in self.tts_engine.getProperty('voices'):
736 
737                 # Voices can have different names on different platforms
738                 #
739                 for specification in self.VOICE_SPECIFICATIONS:
740 
741                     if specification in str(voice.name):
742 
743                         main_voice_id = voice.id
744 
745             if main_voice_id:
746 
747                 self.tts_engine.setProperty('voice', main_voice_id)
748 
749             # Callback methods which can be bound to the events of the engine via
750             # engine.connect('engine_event', method_example) where definition of
751             # method_example would be def method_example(param):
752             def tts_started(name):
753                 """Callback function for TTS engine starting an utterance.
754                 """
755 
756                 fabula.LOGGER.debug("TTS: started utterance '{}'".format(name))
757 
758                 return
```

```
759
760             def tts_finished(completed, name):
761                 """Callback function for TTS engine finished an utterance.
762                 """
763
764                 fabula.LOGGER.debug("TTS: finished utterance")
765
766                 return
767
768             self.tts_engine.connect('started-utterance', tts_started)
769             self.tts_engine.connect('finished-utterance', tts_finished)
770
771             fabula.LOGGER.debug("Audio engine based on pyttsx added")
772
773
774             # Set up sounds and menus
775
776             # Channels: Give every channel an unique ID
777             #
778             # One channel for each surrounding position including client position.
779             # Used to play sounds for items in the surrounding positions - for
780             # example when moving.
781             #
782             self.surrounding_position_channels = [(pygame.mixer.Channel(0), 0.7, 0.5),
783                                                   (pygame.mixer.Channel(1), 0.8, 0.8),
784                                                   (pygame.mixer.Channel(2), 0.5, 0.7),
785                                                   (pygame.mixer.Channel(3), 0.0, 0.8),
786                                                   (pygame.mixer.Channel(4), 0.3, 0.5),
787                                                   (pygame.mixer.Channel(5), 0.4, 0.4),
788                                                   (pygame.mixer.Channel(6), 0.5, 0.3),
789                                                   (pygame.mixer.Channel(7), 0.8, 0.0),
790                                                   (pygame.mixer.Channel(8), 1, 1)]
791
792             # Channel to play back menus and events
793             #
794             self.channel_system = pygame.mixer.Channel(9)
795
796             # Channel to play back the ambience sounds of a room
797             #
798             self.channel_ambience = pygame.mixer.Channel(10)
799
800             # Sound to be played when entering in room input state
801             #
802             file = self.assets.fetch('ambience_room.ogg')
803             self.room_ambience_sound = pygame.mixer.Sound(file)
804             self.room_ambience_sound.set_volume(0.4)
805
806             # Sound to be played when a room is loading
807             #
808             file = self.assets.fetch('loading.ogg')
809             self.loading_sound = pygame.mixer.Sound(file)
810
811             # Sound to be played when loading a room is complete
812             #
813             file = self.assets.fetch('loading_complete.ogg')
814             self.loading_complete_sound = pygame.mixer.Sound(file)
815
816             # Sound to be played if an attempt failed event is sent by the server
817             #
818             file = self.assets.fetch('attempt_failed.ogg')
819             self.attempt_failed_sound = pygame.mixer.Sound(file)
820
821             # Flag to set if connection details are complete
822             #
823             self.connection_details_complete = False
824
825             # Interaction menu - to chose an interaction type
826             #
827             self.interaction_sound_menu = self._set_up_interaction_menu()
828
829             # Select item menu - to select an item for interaction use
830             #
831             self.select_item_menu = self._set_up_select_item_menu()
832
833             # Select target menu - to select a target for an interaction
834             #
```

```
835             self.select_target_menu = self._set_up_select_target_menu()
836
837             # Inventory menu - to read out and look at all inventory (rack) items
838             #
839             self.inventory_menu = self._set_up_inventory_menu()
840
841             # Can speak menu - to select a sentence to speak
842             #
843             self.select_sentence_menu = self._set_up_can_speak_menu()
844
845             # See room menu - get spatial information about all entities in the room
846             #
847             self.see_room_menu = self._set_up_see_room_menu()
848
849             # Help menu - to receive help and select help options
850             #
851             self.help_menu = self._set_up_help_menu()
852
853             # Initialize empty dict for other players in the same room
854             #
855             self.taker_in_room_dict = {}
856
857             # Initialize passive event messages
858             #
859             self.in_inventory_msg = "{} in inventory"
860             self.taker_in_room_msg = "{} is in the room"
861             self.taker_left_room_msg = "{} left the room"
862
863             fabula.LOGGER.debug("complete")
864
865             return
866
867         def get_connection_details(self, prompt_connector=True):
868             """Initially, the client Interface is not connected to the Server. This
869                method should prompt the user for connection details.
870
871                It must return a tuple (identifier, connector), where
872                connector will be used for Interface.connect(connector) and
873                identifier will be used to send InitEvent(identifier) to the server.
874
875                If prompt_connector is False, the dialog will not ask for a connector
876                and use None instead.
877             """
878
879             fabula.LOGGER.debug("called")
880
881             # Use a container so we can access it from an ad-hoc function.
882             # That's Python.
883             #
884             container_dict = {'login_name' : 'player',
885                               'connector' : None}
886
887             # Connection details menu
888             #
889             connection_details_entries = ["Login name",
890                                           "Enter new login"]
891
892             # Optional connector prompt
893             #
894             if prompt_connector:
895
896                 connection_details_entries.append("IP address")
897                 connection_details_entries.append("Enter new IP address")
898
899                 # Set default IP address
900                 #
901                 container_dict['connector'] = '127.0.0.1'
902
903                 audio_text_field_ip_address = AudioTextField('Enter IP address',
904                                                              self.tts_engine)
905
906             connection_details_entries.append("Start game")
907             connection_details_entries.append("Exit")
908
909             audio_text_field_login = AudioTextField('Enter login',
910                                                     self.tts_engine)
```

```
911
912
913             def callback_on_entry_selected(connection_details_menu):
914                 """Callback function for entry selected in self.connection_details_menu.
915                    connection_details_menu is supposed to be the TextMenuList for
916                    the connection details.
917                 """
918
919                 selected_entry = connection_details_menu.list[connection_details_menu.list_index]
920
921                 fabula.LOGGER.debug("Entry selected: {}".format(selected_entry))
922
923                 if selected_entry == "Login name":
924
925                     self.tts_engine.say(container_dict['login_name'])
926                     self.tts_engine.runAndWait()
927
928                 elif selected_entry == "Enter new login":
929
930                     audio_text_field_login.focus_audio_text_field()
931
932                     container_dict['login_name'] = audio_text_field_login.process_key_input()
933
934                 elif selected_entry == "IP address":
935
936                     self.tts_engine.say(container_dict['connector'])
937                     self.tts_engine.runAndWait()
938
939                 elif selected_entry == "Enter new IP address":
940
941                     audio_text_field_ip_address.focus_audio_text_field()
942
943                     container_dict['connector'] = audio_text_field_ip_address.process_key_input()
944
945                 elif selected_entry == "Start game":
946
947                     self.connection_details_complete = True
948
949                 elif selected_entry == "Exit":
950
951                     callback_on_menu_exit(connection_details_menu)
952
953                 return
954
955             # Sound to be played when entering menu
956             #
957             file = self.assets.fetch('fabula_main_menu.ogg')
958             main_menu_sound = pygame.mixer.Sound(file)
959
960             self.connection_details_menu = TextMenuList(connection_details_entries,
961                                                         main_menu_sound,
962                                                         None,
963                                                         self.channel_system,
964                                                         callback_on_entry_selected,
965                                                         None,
966                                                         self.tts_engine)
967
968             # Switch to connection-details-input-state
969             #
970             self._push_input_state_on_stack(self.INPUT_STATE_CONNECTION_DETAILS)
971
972             while not self.connection_details_complete:
973
974                 self.collect_player_input()
975
976             return (container_dict['login_name'],
977                     (container_dict['connector'], 4011))
978
979         def display_single_frame(self):
980             """Pumps the Pygame event queue.
981             """
982
983             # Call base class
984             #
985             fabula.plugins.ui.UserInterface.display_single_frame(self)
986
```

```
987             # Pump the Pygame Event Queue
988             #
989             pygame.event.pump()
990 
991             return
992 
993         def collect_player_input(self):
994             """Gather Pygame events, scan for QUIT or special keys.
995             """
996 
997             # Handle events
998             #
999             events = pygame.event.get()
1000 
1001            for event in events:
1002 
1003                current_state = self.input_state_stack[-1]
1004 
1005                if (self.host.room is not None
1006                    and event.type == pygame.KEYDOWN):
1007 
1008                    # Let the position be read out when key pressed
1009                    #
1010                    if event.key == pygame.K_x:
1011 
1012                        position = self.host.room.entity_locations[self.host.client_id]
1013 
1014                        # Let the TTS engine read out the current client position
1015                        #
1016                        self.tts_engine.say(position, 'position')
1017                        self.tts_engine.runAndWait()
1018 
1019                    # Open list with all objects in room
1020                    #
1021                    elif event.key == pygame.K_z: # English keyboard layout for "y"
1022 
1023                        # Transition to see-room-input-state
1024                        #
1025                        self._push_input_state_on_stack(self.INPUT_STATE_SEE_ROOM)
1026 
1027                    elif event.key == pygame.K_F1:
1028 
1029                        # Transition to help-menu-input-state
1030                        #
1031                        self._push_input_state_on_stack(self.INPUT_STATE_HELP_MENU)
1032 
1033                    # Default input state, player is in room and can move or start
1034                    # to interact or open the rack
1035                    #
1036                    if current_state == self.INPUT_STATE_IN_ROOM:
1037 
1038                        # Check if collected input for movement
1039                        #
1040                        if (event.key in (pygame.K_w, pygame.K_a, pygame.K_s, pygame.K_d)):
1041 
1042                            surrounding_positions = fabula.surrounding_positions(self.host.room.entity_locations[self.host.client_id])
1043 
1044                            fabula.LOGGER.debug("got key '{}' from user, returning TriesToMoveToEvent".format(event.key))
1045 
1046                            moves_event = fabula.TriesToMoveEvent(self.host.client_id,
1047                                                                  {pygame.K_w : surrounding_positions[1],
1048                                                                   pygame.K_d : surrounding_positions[3],
1049                                                                   pygame.K_s : surrounding_positions[5],
1050                                                                   pygame.K_a : surrounding_positions[7]
1051                                                                   }[event.key])
1052 
1053                            self.message_for_host.event_list.append(moves_event)
1054 
1055                        # Open rack
1056                        #
1057                        elif(event.key == pygame.K_q):
1058 
1059                            # Transition to see-rack-input-state
1060                            #
1061                            self._push_input_state_on_stack(self.INPUT_STATE_SEE_RACK)
1062 
```

```
                    # Start interaction input state
                    #
                    elif(event.key == pygame.K_e):

                        # Transition to interaction-menu-input-state
                        #
                        self._push_input_state_on_stack(self.INPUT_STATE_INTERACTION)

                elif current_state == self.INPUT_STATE_INTERACTION:

                    # Scroll ways of interaction, select one or go back
                    #
                    self.interaction_sound_menu.process_key_input(event.key)

                elif current_state == self.INPUT_STATE_SELECT_ITEM:

                    # Scroll possible items to be used to drop on a target,
                    # select one or go back
                    #
                    self.select_item_menu.process_key_input(event.key)

                elif current_state == self.INPUT_STATE_SELECT_TARGET:

                    # Scroll possible targets, select one or go back
                    #
                    self.select_target_menu.process_key_input(event.key)

                elif current_state == self.INPUT_STATE_SEE_RACK:

                    # Scroll the entities in the rack, select one or go back
                    #
                    self.inventory_menu.process_key_input(event.key)

                elif current_state == self.INPUT_STATE_CAN_SPEEK:

                    # Scroll possible sentences, select a sentence to be spoken
                    #
                    self.select_sentence_menu.process_key_input(event.key)

                elif current_state == self.INPUT_STATE_SEE_ROOM:

                    # Scroll the entities in the room, select one or go back
                    #
                    self.see_room_menu.process_key_input(event.key)

                elif current_state == self.INPUT_STATE_HELP_MENU:

                    # Scroll help menu options, select one or go back
                    #
                    self.help_menu.process_key_input(event.key)

            if (event.type == pygame.KEYDOWN
                and current_state == self.INPUT_STATE_CONNECTION_DETAILS):

                    # Scroll connection detail options or select one
                    #
                    self.connection_details_menu.process_key_input(event.key)

            if event.type == pygame.QUIT or (event.type == pygame.KEYDOWN
                                             and event.key == pygame.K_q
                                             and pygame.key.get_mods() & pygame.KMOD_CTRL):

                fabula.LOGGER.info("exit request from user")
                self.exit_requested = True

                # Quit pygame here, though there is still shutdown work to be
                # done in Client and ClientInterface.
                #
                pygame.quit()

        return

    def process_EnterRoomEvent(self, event):
        """Play loading_sound.
        """
```

```
1139            fabula.LOGGER.info("entering room: {}".format(event.room_identifier))
1140
1141            fabula.LOGGER.info("freezing")
1142            self.freeze = True
1143
1144            # Pause room ambience sound if playing
1145            #
1146            if self.channel_ambience.get_busy():
1147
1148                self.channel_ambience.pause()
1149
1150            # Play back loading sound
1151            #
1152            self.channel_system.play(self.loading_sound)
1153
1154            return
1155
1156        def process_RoomCompleteEvent(self, event):
1157            """Play loading_complete_sound and change input state to INPUT_STATE_IN_ROOM.
1158            """
1159
1160            fabula.LOGGER.info("unfreezing")
1161            self.freeze = False
1162
1163            self.channel_system.queue(self.loading_complete_sound)
1164
1165            # Wait for loading sound and loading complete sound to be played
1166            # completely
1167            #
1168            while (self.channel_system.get_busy()
1169                   or self.channel_system.get_queue() is not None):
1170                pass
1171
1172            # process_ChangeMapElementEvent and process_SpawnEvent should have
1173            # set up everything by now
1174            #
1175            # Switch to in-room-input-state to collect the right input
1176            #
1177            self._push_input_state_on_stack(self.INPUT_STATE_IN_ROOM)
1178
1179            return
1180
1181        def process_CanSpeakEvent(self, event):
1182            """Opens a select sentence menu with the events sentences.
1183            """
1184
1185            # If it's not for us, ignore.
1186            #
1187            if event.identifier == self.host.client_id and event.sentences:
1188
1189                # Set up sentence menu and switch to can-speak-input-state
1190                #
1191                self.select_sentence_menu.list = event.sentences
1192                self._push_input_state_on_stack(self.INPUT_STATE_CAN_SPEEK)
1193
1194            return
1195
1196        def process_SpawnEvent(self, event):
1197            """Entity.asset points to the sound or the text asset of the Entity.
1198
1199               If the asset file name contains 'steps', this method will treat
1200               the sound as a step sound. See the AudioEntity documentation for
1201               details.
1202
1203               If the spawned entity is another PLAYER, it will be added to the
1204               self.taker_in_room_dict and its name will be announced.
1205            """
1206
1207            if event.entity.user_interface is None:
1208
1209                # Call base class
1210                #
1211                fabula.plugins.ui.UserInterface.process_SpawnEvent(self, event)
1212
1213            # Set up entity's sound asset
1214            #
```

```
1215            if 'audio/ogg' in event.entity.assets.keys():
1216
1217                # This is possibly a respawn from Rack, and the entity already has
1218                # an asset.
1219                #
1220                if event.entity.assets['audio/ogg'].data is not None:
1221
1222                    msg = "Entity '{}' already has an audio asset"
1223                    fabula.LOGGER.info(msg.format(event.entity.identifier))
1224
1225                else:
1226
1227                    # Entity has an audio file attached, fetch the asset
1228                    #
1229                    try:
1230
1231                        # Get a file-like object from asset manager
1232                        #
1233                        file = self.assets.fetch(event.entity.assets['audio/ogg'].uri)
1234
1235                    except:
1236
1237                        self.display_asset_exception(event.entity.assets['audio/ogg'].uri)
1238
1239                    # Replace with pygame.mixer.Sound from file
1240                    #
1241                    sound = pygame.mixer.Sound(file)
1242
1243                    file.close()
1244
1245                    # Finally attach the Sound as the Entity's asset
1246                    #
1247                    event.entity.assets['audio/ogg'].data = sound
1248
1249                # Entity has a step sound which consists of a left and a right step.
1250                # Special handling for this 'sound animation' is done in the
1251                # AudioEntity class so change the entity's class to support steps.
1252                #
1253                if 'steps' in event.entity.assets['audio/ogg'].uri:
1254
1255                    # The Entity must be able to react to events. These reactions
1256                    # are Pygame specific and are thus not covered in the basic
1257                    # fabula.Entity class. Thus, we exploit a Python feature here
1258                    # and change the entity's class at runtime to a class that
1259                    # supports Pygame.
1260                    #
1261                    if (event.entity.__class__ is AudioEntity
1262                        or AudioEntity in event.entity.__class__.__bases__):
1263
1264                        # AudioEntity already in use? Great!
1265                        #
1266                        pass
1267
1268                    elif event.entity.__class__ is fabula.Entity:
1269
1270                        # Oh well. Just swap.
1271                        #
1272                        fabula.LOGGER.debug("changing class of '{}' from {} to {}".format(event.entity.identifier,
1273                                                                                          event.entity.__class__,
1274                                                                                          AudioEntity))
1275
1276                        event.entity.__class__ = AudioEntity
1277
1278                    else:
1279                        # To preserve the class, create a new class that inherits from
1280                        # the current one and from PygameEntity.
1281                        #
1282                        class ExtendedEntity(event.entity.__class__, AudioEntity):
1283
1284                            pass
1285
1286                        fabula.LOGGER.debug("changing class of '{}' from {} to bases {}".format(event.entity.identifier,
1287                                                                                                event.entity.__class__,
1288                                                                                                ExtendedEntity.__bases__))
1289
1290                        event.entity.__class__ = ExtendedEntity
```

```
1291
1292                    # Since we do not call AudioEntity.__init__() to prevent messing
1293                    # up already present data, we add required attributes
1294                    # TODO: it's easy to miss that when changing the PygameEntity.__init__()
1295                    #
1296                    # Steps are played every movement therefore they play at a low
1297                    # volume
1298                    #
1299                    event.entity.assets['audio/ogg'].data.set_volume(0.3)
1300
1301            # Set up entity's text asset
1302            #
1303            if 'text/plain' in event.entity.assets.keys():
1304
1305                # This is possibly a respawn from Rack, and the entity already has
1306                # an asset.
1307                #
1308                if event.entity.assets['text/plain'].data is not None:
1309
1310                    msg = "Entity '{}' already has an text asset"
1311                    fabula.LOGGER.info(msg.format(event.entity.identifier))
1312
1313                else:
1314
1315                    # Entity has no text file attached so get a 'text/plain'
1316                    # representation for it
1317                    #
1318                    try:
1319
1320                        # Get a file-like object from asset manager
1321                        #
1322                        file = self.assets.fetch(event.entity.assets['text/plain'].uri, mode='t')
1323
1324                    except:
1325
1326                        self.display_asset_exception(event.entity.assets['text/plain'].uri)
1327
1328                    text = file.readlines()
1329
1330                    file.close()
1331
1332                    # Finally attach the text as the entity's asset
1333                    #
1334                    event.entity.assets['text/plain'].data = text
1335
1336            # Play sound if spawned entity is close to client
1337            #
1338            self._play_sound_surrounding_entity(event.entity)
1339
1340            # If another player entity was spawned (i.e. is in the room), play its
1341            # name and store its entity in self.taker_in_room_dict
1342            #
1343            if (event.entity.entity_type == fabula.PLAYER and
1344                event.entity.identifier is not self.host.client_id):
1345
1346                # Store other players entity
1347                #
1348                self.taker_in_room_dict[event.entity.identifier] = event.entity
1349
1350                player_name = event.entity.identifier
1351
1352                # Check if player provides a name
1353                #
1354                if ('text/plain' in event.entity.assets.keys()
1355                    and event.entity.assets['text/plain'].data is not None):
1356
1357                    player_name = event.entity.assets['text/plain'].data[0]
1358
1359                taker_in_room_event = fabula.PerceptionEvent(self.host.client_id,
1360                                                             self.taker_in_room_msg.format(player_name))
1361
1362                self.process_PerceptionEvent(taker_in_room_event)
1363
1364            return
1365
1366        def process_DeleteEvent(self, event):
```

```
            """If the deleted entity is another PLAYER, it will be announced that the player left the room.
               Its Entity will be removed from the self.taker_in_room_dict.
            """

            # Play sound if other player entity was deleted (left the room) and
            # delete it from self.taker_in_room_dict
            #
            if event.identifier in self.taker_in_room_dict:

                entity = self.taker_in_room_dict[event.identifier]

                player_name = entity.identifier

                # Check if player provides a name
                #
                if ('text/plain' in entity.assets.keys()
                    and entity.assets['text/plain'].data is not None):

                    player_name = entity.assets['text/plain'].data[0]

                taker_left_room_event = fabula.PerceptionEvent(self.host.client_id,
                                                               self.taker_left_room_msg.format(player_name))

                self.process_PerceptionEvent(taker_left_room_event)

                # Remove other players entity
                #
                del self.taker_in_room_dict[event.identifier]

            return

        def process_ChangeMapElementEvent(self, event):
            """Fetch asset and register/replace tile.
            """

            fabula.LOGGER.debug("called")

            tile_from_list = None

            # The Client should have added the Tile to self.host.room.tile_list
            #
            for tile in self.host.room.tile_list:

                # This should succeed only once
                #
                if tile == event.tile:

                    # Tiles may compare equal, but they may not refer to the same
                    # instance, so we use tile from tile_list.
                    #
                    fabula.LOGGER.debug("found event.tile in self.host.room.tile_list")
                    tile_from_list = tile

            if tile_from_list is None:

                fabula.LOGGER.error("could not find tile {} in tile_list of room '{}'".format(event.tile, self.host.room.identifier))
                raise RuntimeError("could not find tile {} in tile_list of room '{}'".format(event.tile, self.host.room.identifier))

            if ('audio/ogg' in tile_from_list.assets.keys()
                and tile_from_list.assets['audio/ogg'].data is not None):

                fabula.LOGGER.debug('tile already has an audio asset: {}'.format(tile_from_list))

            elif 'audio/ogg' in event.tile.assets.keys():

                # Assets are entirely up to the UserInterface, so we fetch
                # the asset here
                #
                fabula.LOGGER.debug('no asset for {}, attempting to fetch'.format(tile_from_list))

                # Entity has an audio file attached, fetch the asset
                #
                try:

                    # Get a file-like object from asset manager
                    #
```

```
                    file = self.assets.fetch(tile_from_list.assets['audio/ogg'].uri)

                except:

                    self.display_asset_exception(event.entity.assets['audio/ogg'].uri)

                # Replace with pygame.mixer.Sound from file
                #
                sound = pygame.mixer.Sound(file)

                file.close()

                # Finally attach the Sound as the Tile's asset, update sound
                # regardless whether the Tile existed or not
                #
                tile_from_list.assets['audio/ogg'].data = sound

                fabula.LOGGER.debug("changing sound for tile at {0} to {1}".format(str(event.location),
                                                                                   tile_from_list.assets['audio/ogg'].data))

            if ('text/plain' in tile_from_list.assets.keys()
                and tile_from_list.assets['text/plain'].data is not None):

                fabula.LOGGER.debug("tile already has an text asset: {}".format(tile_from_list))

            elif 'text/plain' in event.tile.assets.keys():

                # Assets are entirely up to the UserInterface, so we fetch
                # the asset here
                #
                fabula.LOGGER.debug("no asset for {}, attempting to fetch".format(tile_from_list))

                # Entity has no audio file attached so get a 'text/plain'
                # representation for it
                #
                try:

                    # Get a file-like object from asset manager
                    #
                    file = self.assets.fetch(tile_from_list.assets['text/plain'].uri, mode='t')

                except:

                    self.display_asset_exception(event.entity.assets['text/plain'].uri)

                text = file.readlines()

                file.close()

                # Finally attach the Text as the Tile's asset, update text
                # regardless whether the Tile existed or not
                #
                tile_from_list.assets['text/plain'].data = text

                fabula.LOGGER.debug("changing text for tile at {0} to {1}".format(str(event.location),
                                                                                  tile_from_list.assets['text/plain'].data))
            return

        def process_PerceptionEvent(self, event):
            """Read out the perception with the text-to-speech engine.
            """

            fabula.LOGGER.debug("called")

            # If it's not for us, ignore.
            #
            if event.identifier == self.host.client_id:

                # Let the TTS engine read the perception text
                #
                self.tts_engine.say(event.perception)
                self.tts_engine.runAndWait()

            else:

                fabula.LOGGER.warning("perception for '{}', not read out".format(event.identifier))
```

```
1519
1520            return
1521
1522        def process_AttemptFailedEvent(self, event):
1523            """Play back attempt failed sound.
1524            """
1525
1526            fabula.LOGGER.debug("called")
1527
1528            # If it's not for us, ignore.
1529            #
1530            if event.identifier == self.host.client_id:
1531
1532                # Acoustic feedback for attempt failed
1533                #
1534                self.channel_system.play(self.attempt_failed_sound)
1535
1536            return
1537
1538        def process_MovesToEvent(self, event):
1539            """Call the base class and make Entity's surrounding the new player position audible.
1540            """
1541
1542            fabula.LOGGER.debug("called")
1543
1544            # To listen to the new surrounding entities, fade the old ones out
1545            #
1546            self._fadeout_sound_surrounding_entities()
1547
1548            # Collect entities from surrounding_positions
1549            #
1550            surrounding_positions = fabula.surrounding_positions(self.host.room.entity_locations[self.host.client_id])
1551            for entity in self.host.room.entity_dict.values():
1552
1553                # Delegate play sound
1554                #
1555                self._play_sound_surrounding_entity(entity)
1556
1557            # Call base class, this will forward the Event to the affected Entity.
1558            #
1559            fabula.plugins.ui.UserInterface.process_MovesToEvent(self, event)
1560
1561            return
1562
1563        def process_PicksUpEvent(self, event):
1564            """Check the affected Entity, and give feedback that item is now in the inventory.
1565            """
1566
1567            # The Client has already put the item from Client.room to Client.rack.
1568            # We have to inform the client that the item is now in the inventory
1569            #
1570            if (event.item_identifier in self.host.rack.owner_dict.keys()
1571                and self.host.rack.owner_dict[event.item_identifier] == self.host.client_id):
1572
1573                item_name = event.item_identifier
1574
1575                item = self.host.rack.entity_dict[event.item_identifier]
1576
1577                # Check if Entity provides a name
1578                #
1579                if ('text/plain' in item.assets.keys()
1580                    and item.assets['text/plain'].data is not None):
1581
1582                    item_name = item.assets['text/plain'].data[0]
1583
1584                picked_up_perception = self.in_inventory_msg.format(item_name)
1585                perception_event = fabula.PerceptionEvent(identifier=self.host.client_id,
1586                                                          perception=picked_up_perception)
1587
1588                self.process_PerceptionEvent(perception_event)
1589
1590            # Call base class to notify the Entity
1591            #
1592            fabula.plugins.ui.UserInterface.process_PicksUpEvent(self, event)
1593
1594        def process_SaysEvent(self, event):
```

```
        """Call the base class, then present the text to the user and wait some time.
        """

        fabula.LOGGER.debug("called")

        # Call base class, this will forward the event to the Entity.
        #
        fabula.plugins.ui.UserInterface.process_SaysEvent(self, event)

        entity_id = event.identifier

        if event.identifier in self.host.room.entity_dict.keys():

            entity = self.host.room.entity_dict[event.identifier]

            # Check if Entity provides a name
            #
            if ('text/plain' in entity.assets.keys()
                and entity.assets['text/plain'].data is not None):

                entity_id = entity.assets['text/plain'].data[0]

        # Let the TTS engine read the sentence, lead by the speakers name, the
        # colon results in a small pause between identifier and text
        #
        entity_id += ": "
        self.tts_engine.say(entity_id, '{}'.format(entity_id))
        self.tts_engine.say(event.text, 'SaysEvent')
        self.tts_engine.runAndWait()

        return

    def _set_up_interaction_menu(self):
        """Sets up the interaction menu list.
        """

        def callback_on_interaction_selected(interaction_sound_menu):

            # attempt_look_at, attempt_talk_to, attempt_pick_up and
            # attempt_manipulate
            #
            if interaction_sound_menu.list_index in (self.ATTEMPT_LOOK_AT,
                                                     self.ATTEMPT_TALK_TO,
                                                     self.ATTEMPT_PICK_UP,
                                                     self.ATTEMPT_MANIPULATE):

                # Transition to select-target-input-state
                #
                self._push_input_state_on_stack(self.INPUT_STATE_SELECT_TARGET)

            # attempt_use
            #
            elif interaction_sound_menu.list_index == self.ATTEMPT_USE:

                # Transition to select-item-input-state
                #
                self._push_input_state_on_stack(self.INPUT_STATE_SELECT_ITEM)

            # cancel
            #
            elif interaction_sound_menu.list_index == self.CANCEL:

                callback_on_interaction_menu_exit(interaction_sound_menu)

            return

        def callback_on_interaction_menu_exit(interaction_sound_menu):

            # Going back to previous input state
            #
            self._pop_input_state_from_stack()

            return

        # Load the standard attempt interaction sounds
        #
```

```
        interaction_sounds = []

        # Pay Attention to order of index, must be same as in
        #
        # ATTEMPT_LOOK_AT = 0
        # ATTEMPT_TALK_TO = 1
        # ATTEMPT_PICK_UP = 2
        # ATTEMPT_USE = 3
        # ATTEMPT_MANIPULATE = 4
        # CANCEL = 5
        #
        for name in ('attempt_look_at',
                     'attempt_talk_to',
                     'attempt_pick_up',
                     'attempt_use',
                     'attempt_manipulate',
                     'cancel'):

            # Load sound file and save it as pygame.mixer.Sound:
            #
            file = self.assets.fetch(name + '.ogg')
            sound = pygame.mixer.Sound(file)
            sound.set_volume(1)

            interaction_sounds.append(sound)

        # Sound to be played when interaction menu is opened
        #
        file = self.assets.fetch('interaction_menu.ogg')
        menu_sound = pygame.mixer.Sound(file)

        interaction_sound_menu = SoundMenuList(interaction_sounds,
                                               menu_sound,
                                               self.attempt_failed_sound,
                                               self.channel_system,
                                               callback_on_interaction_selected,
                                               callback_on_interaction_menu_exit)

        return interaction_sound_menu

    def _set_up_select_item_menu(self):
        """Sets up the item selection menu list.
        """

        def callback_on_item_selected(select_item_menu):

            # Transition to select-target-input-state
            #
            self._push_input_state_on_stack(self.INPUT_STATE_SELECT_TARGET)

            return

        def callback_on_item_menu_exit(select_item_menu):

            # Going back to previous input state
            #
            self._pop_input_state_from_stack()

            return

        # Sound to be played when item selection menu is opened
        #
        file = self.assets.fetch('select_item_menu.ogg')
        select_item_menu_sound = pygame.mixer.Sound(file)

        # Sound to be played when no items present to be selected
        #
        file = self.assets.fetch('empty_item_menu.ogg')
        empty_item_menu_sound = pygame.mixer.Sound(file)

        select_item_menu = EntityMenuList([],
                                          select_item_menu_sound,
                                          empty_item_menu_sound,
                                          self.channel_system,
                                          callback_on_item_selected,
                                          callback_on_item_menu_exit,
```

```
                                        self.tts_engine)

        return select_item_menu

    def _set_up_select_target_menu(self):
        """Sets up the target selection menu list.
        """

        def callback_on_target_selected(select_target_menu):

            # Selects target for the event and appends the resulting event to
            # messages for the server. Remember selected target is a tuple of
            # (Entity, <Entity.identifier|location>) and we need the second
            # value to create an event.
            #
            selected_target = select_target_menu.list[select_target_menu.list_index][1]

            # attempt_look_at
            #
            if self.interaction_sound_menu.list_index == self.ATTEMPT_LOOK_AT:

                interaction_event = fabula.TriesToLookAtEvent(self.host.client_id,
                                                              selected_target)

                self.message_for_host.event_list.append(interaction_event)

            # attempt_talk_to
            #
            elif self.interaction_sound_menu.list_index == self.ATTEMPT_TALK_TO:

                interaction_event = fabula.TriesToTalkToEvent(self.host.client_id,
                                                              selected_target)

                self.message_for_host.event_list.append(interaction_event)

            # attempt_pick_up
            #
            elif self.interaction_sound_menu.list_index == self.ATTEMPT_PICK_UP:

                interaction_event = fabula.TriesToPickUpEvent(self.host.client_id,
                                                              selected_target)

                self.message_for_host.event_list.append(interaction_event)

            # attempt_manipulate
            #
            elif self.interaction_sound_menu.list_index == self.ATTEMPT_MANIPULATE:

                interaction_event = fabula.TriesToManipulateEvent(self.host.client_id,
                                                                  selected_target)

                self.message_for_host.event_list.append(interaction_event)

            # attempt_use
            #
            elif self.interaction_sound_menu.list_index == self.ATTEMPT_USE:

                # Remember selected item is a tuple of
                # (Entity, <Entity.identifier|location>) and we need the second
                # value to create an event              #
                selected_item = self.select_item_menu.list[self.select_item_menu.list_index][1]

                interaction_event = fabula.TriesToDropEvent(self.host.client_id,
                                                            selected_item,
                                                            selected_target)

                self.message_for_host.event_list.append(interaction_event)

            #  After interaction return to in-room-input-state
            #
            self._push_input_state_on_stack(self.INPUT_STATE_IN_ROOM)

            return

        def callback_on_target_menu_exit(select_target_menu):
```

```
            # Going back to previous input state
            #
            self._pop_input_state_from_stack()

            return

        # Sound to be played when target selection menu is opened
        #
        file = self.assets.fetch('select_target_menu.ogg')
        select_target_menu_sound = pygame.mixer.Sound(file)

        # Sound to be played when no targets present to be selected
        #
        file = self.assets.fetch('empty_target_menu.ogg')
        empty_target_menu_sound = pygame.mixer.Sound(file)

        select_target_menu = EntityMenuList([],
                                            select_target_menu_sound,
                                            empty_target_menu_sound,
                                            self.channel_system,
                                            callback_on_target_selected,
                                            callback_on_target_menu_exit,
                                            self.tts_engine)

        return select_target_menu

    def _set_up_inventory_menu(self):
        """Sets up the inventory menu list.
        """

        def callback_on_inventory_item_selected(inventory_menu):

            # Remember selected item is a tuple of
            # (Entity, <Entity.identifier|location>) and we need the second
            # value to create an event.
            #
            selected_item = inventory_menu.list[inventory_menu.list_index][1]

            # Attempt to look at selected item
            #
            look_at_event = fabula.TriesToLookAtEvent(self.host.client_id,
                                                      selected_item)

            self.message_for_host.event_list.append(look_at_event)

            return

        def callback_on_inventory_menu_exit(inventory_menu):

            # Going back to previous input state
            #
            self._pop_input_state_from_stack()

            return

        # Sound to be played when inventory is opened
        #
        file = self.assets.fetch('inventory.ogg')
        inventory_sound = pygame.mixer.Sound(file)

        # Sound to be played when inventory is empty
        #
        file = self.assets.fetch('empty_inventory_menu.ogg')
        empty_inventory_menu_sound = pygame.mixer.Sound(file)

        inventory_menu = EntityMenuList([],
                                        inventory_sound,
                                        empty_inventory_menu_sound,
                                        self.channel_system,
                                        callback_on_inventory_item_selected,
                                        callback_on_inventory_menu_exit,
                                        self.tts_engine)

        return inventory_menu

    def _set_up_can_speak_menu(self):
```

```
            """Sets up the can speak menu list.
            """

            def callback_on_sentence_selected(select_sentence_menu):

                # Send an event to the server with the selected sentence
                #
                selected_sentence = select_sentence_menu.list[select_sentence_menu.list_index]

                says_event = fabula.SaysEvent(self.host.client_id, selected_sentence)

                self.message_for_host.event_list.append(says_event)

                # Return to previous input state
                #
                self._pop_input_state_from_stack()

                return

            # Sound to be played when a CanSpeak Event is processed and a sentence
            # should be selected
            #
            file = self.assets.fetch('select_sentence.ogg')
            select_sentence_sound = pygame.mixer.Sound(file)

            # Sound to be played when a CanSpeak Event is processed and no sentences
            # are provided
            #
            file = self.assets.fetch('empty_sentences_menu.ogg')
            empty_sentences_menu_sound = pygame.mixer.Sound(file)

            select_sentence_menu = TextMenuList([],
                                                select_sentence_sound,
                                                empty_sentences_menu_sound,
                                                self.channel_system,
                                                callback_on_sentence_selected,
                                                None,
                                                self.tts_engine)

            return select_sentence_menu

        def _set_up_see_room_menu(self):
            """Sets up the see room entities menu list.
            """

            def callback_on_item_in_room_selected(see_room_menu):

                # Reads out the selected item's location, selected item is a tuple
                # of (Entity, <Entity.identifier|location>) and we need the second
                # value to create an event.
                #
                selected_item_id = see_room_menu.list[see_room_menu.list_index][1]

                if selected_item_id in self.host.room.entity_locations.keys():

                    position = self.host.room.entity_locations[selected_item_id]

                    # Let the TTS engine read the position of the entity at current
                    # index
                    #
                    self.tts_engine.say(position)
                    self.tts_engine.runAndWait()

                return

            def callback_on_see_room_menu_exit(see_room_menu):

                # Going back to previous input state
                #
                self._pop_input_state_from_stack()

                return

            # Sound to be played when see room entities menu is opened
            #
            file = self.assets.fetch('entities_in_room.ogg')
```

```
            entities_in_room_sound = pygame.mixer.Sound(file)

            # Sound to be played when there are no entities in the room
            #
            file = self.assets.fetch('empty_entities_in_room.ogg')
            empty_entities_in_room_sound = pygame.mixer.Sound(file)

            see_room_menu = EntityMenuList([],
                                           entities_in_room_sound,
                                           empty_entities_in_room_sound,
                                           self.channel_system,
                                           callback_on_item_in_room_selected,
                                           callback_on_see_room_menu_exit,
                                           self.tts_engine)

            return see_room_menu

        def _set_up_help_menu(self):
            """Sets up the help menu list.
            """

            def callback_on_help_entry_selected(help_menu):

                selected_entry = help_menu.list[help_menu.list_index]

                # Let the TTS engine read the help information at current index
                #
                help_menu.tts_engine.say(help_menu_entries_values[selected_entry])
                help_menu.tts_engine.runAndWait()

                return

            def callback_on_help_menu_exit(help_menu):

                # Going back to previous input state
                #
                self._pop_input_state_from_stack()

                return

            # Default help menu (for adventure games)
            #
            help_menu_entries_values = {'key q' : 'open inventory',
                                        'key e' : 'open interaction menu',
                                        'key x' : 'read out player position',
                                        'key y': 'list all items in room',
                                        'movement': 'w a s d keys',
                                        'key w' : 'move one field up',
                                        'key a' : 'move one field to the left',
                                        'key s': 'move one field down',
                                        'key d' : 'move one field to the right'}

            # Keys for help menu entries, notice that the order of this list defines
            # the order in which the entries are sorted in the list
            #
            help_menu_entries = ['key q',
                                 'key e',
                                 'key x',
                                 'key y',
                                 'movement',
                                 'key w',
                                 'key a',
                                 'key s',
                                 'key d']

            # Sound to be played when help menu is opened
            #
            file = self.assets.fetch('help_menu.ogg')
            help_menu_sound = pygame.mixer.Sound(file)

            help_menu = TextMenuList(help_menu_entries,
                                     help_menu_sound,
                                     None,
                                     self.channel_system,
                                     callback_on_help_entry_selected,
                                     callback_on_help_menu_exit,
```

```
                                  self.tts_engine)

        return help_menu

    def _play_sound_surrounding_entity(self, entity):
        """Plays back the audio assets of the given entity if it is on a position close enough to the client.
           The stereo volumes of the sound will be adjusted to simulate a
           spatial impression.
        """

        # Get client position and positions close to it
        #
        client_position = self.host.room.entity_locations[self.host.client_id]
        surrounding_positions = fabula.surrounding_positions(client_position)

        surrounding_positions.append(client_position)

        # Get location of given entity
        #
        location = self.host.room.entity_locations[entity.identifier]

        # If given entity close enough, play its sound
        #
        if (location in surrounding_positions
            and entity.identifier is not self.host.client_id
            and 'audio/ogg' in entity.assets.keys()
            and entity.assets['audio/ogg'].data is not None):

            fabula.LOGGER.debug("Playing Sound of: {}".format(entity.identifier))

            # Channel consist of a mixer.channel, a value for left volume and
            # a value for right volume.
            #
            channel = {surrounding_positions[0] : self.surrounding_position_channels[0],
                       surrounding_positions[1] : self.surrounding_position_channels[1],
                       surrounding_positions[2] : self.surrounding_position_channels[2],
                       surrounding_positions[3] : self.surrounding_position_channels[3],
                       surrounding_positions[4] : self.surrounding_position_channels[4],
                       surrounding_positions[5] : self.surrounding_position_channels[5],
                       surrounding_positions[6] : self.surrounding_position_channels[6],
                       surrounding_positions[7] : self.surrounding_position_channels[7],
                       surrounding_positions[8] : self.surrounding_position_channels[8]
                      }[location]

            channel[0].play(entity.assets['audio/ogg'].data)
            channel[0].set_volume(channel[1], channel[2])

        return

    def _fadeout_sound_surrounding_entities(self):
        """Fades out all sounds that are playing from surrounding Entities.
        """

        for channel in self.surrounding_position_channels:

            # Fade out every channel which could play back sounds of
            # surrounding entities
            #
            channel[0].fadeout(300)

        return

    def _push_input_state_on_stack(self, state):
        """Pushes the given input state on the stack and performs a state transition to it.
        """

        # Handling for states which are already on the stack
        #
        if (len(self.input_state_stack) is 0
            or (len(self.input_state_stack) > 0
                and state is not self.input_state_stack[-1])):

            self.input_state_stack.append(state)

        self._transition_to_input_state()
```

```
            return

        def _pop_input_state_from_stack(self):
            """Pops the topmost input state from the stack and performs a transition to the previous state.
            """

            if len(self.input_state_stack) > 0:

                self.input_state_stack.pop()
                self._transition_to_input_state()

            else:

                self._push_input_state_on_stack(self.INPUT_STATE_IN_ROOM)

            return

        def _transition_to_input_state(self):
            """Performs the state transition from the former input state to the topmost input state.
               Before calling this method the new state has to be pushed on the
               stack.
            """

            audio_menu_list = None
            current_state = self.input_state_stack[-1]

            if current_state is self.INPUT_STATE_IN_ROOM:

                # Empty stack from old history whenever in room
                self.input_state_stack = [self.INPUT_STATE_IN_ROOM]

                # Play ambience sound for room if not already playing
                #
                if self.channel_ambience.get_busy():

                    self.channel_ambience.unpause()

                else:

                    self.channel_ambience.play(self.room_ambience_sound, -1)

            elif current_state is self.INPUT_STATE_INTERACTION:

                audio_menu_list = self.interaction_sound_menu

            elif current_state is self.INPUT_STATE_SELECT_TARGET:

                # Get the current attempt type
                #
                attempt_type = self.interaction_sound_menu.list_index

                # Set up list with possible targets
                #
                self.select_target_menu.list = self._get_possible_targets(attempt_type)
                audio_menu_list = self.select_target_menu

            elif current_state is self.INPUT_STATE_SELECT_ITEM:

                # Set up list with possible items
                #
                self.select_item_menu.list = self._get_possible_items()
                audio_menu_list = self.select_item_menu

            elif current_state is self.INPUT_STATE_CAN_SPEEK:

                audio_menu_list = self.select_sentence_menu

            elif current_state is self.INPUT_STATE_SEE_RACK:

                # Gather all rack items
                #
                self.inventory_menu.list = self._get_rack_items()
                audio_menu_list = self.inventory_menu

            elif current_state is self.INPUT_STATE_SEE_ROOM:

```

```
            # Set up list with all entities in room
            #
            self.see_room_menu.list = self._get_room_entities()
            audio_menu_list = self.see_room_menu

        elif current_state is self.INPUT_STATE_HELP_MENU:

            audio_menu_list = self.help_menu

        elif current_state is self.INPUT_STATE_CONNECTION_DETAILS:

            audio_menu_list = self.connection_details_menu


        if audio_menu_list is not None:

            if audio_menu_list.try_to_open_menu_list():

                # Pause ambient sound and fade out sounds of surrounding
                # objects while in menu
                #
                self.channel_ambience.pause()
                self._fadeout_sound_surrounding_entities()

        return

    def _get_possible_items(self, attempt_type=ATTEMPT_USE):
        """Returns the possible items for the given interaction type.
        """

        items = []

        surrounding_positions = fabula.surrounding_positions(self.host.room.entity_locations[self.host.client_id])

        # Add entities from surrounding_positions
        #
        for entity in self.host.room.entity_dict.values():

            if (self.host.room.entity_locations[entity.identifier] in surrounding_positions
                and entity.entity_type not in (fabula.PLAYER, fabula.NPC)
                and entity.mobile):

                items.append((entity, entity.identifier))

        # Add entities from rack
        #
        for entity in self.host.rack.entity_dict.values():

            if (entity.identifier in self.host.rack.owner_dict.keys()
                and self.host.rack.owner_dict[entity.identifier] == self.host.client_id):

                items.append((entity, entity.identifier))

        return items

    def _get_possible_targets(self, attempt_type):
        """Returns the possible targets for the given interaction type.
        """

        targets = []

        client_position = self.host.room.entity_locations[self.host.client_id]
        surrounding_positions = fabula.surrounding_positions(client_position)
        surrounding_positions.append(client_position)

        # Add entities from room
        #
        for entity in self.host.room.entity_dict.values():

            location = self.host.room.entity_locations[entity.identifier]

            if attempt_type is self.ATTEMPT_TALK_TO:

                if (entity.entity_type is not fabula.ITEM
                    and entity.identifier is not self.host.client_id):
```

```
                    targets.append((entity, location))

            elif attempt_type is self.ATTEMPT_LOOK_AT:

                if entity.identifier is not self.host.client_id:

                    targets.append((entity, location))

            # Add only entities from surrounding positions for pick up and use
            #
            if location in surrounding_positions:

                if attempt_type is self.ATTEMPT_PICK_UP:

                    # Add only mobile entities of type item for pickup
                    #
                    if entity.entity_type is fabula.ITEM and entity.mobile:

                        targets.append((entity, location))

                # Add entities for use
                #
                elif attempt_type is self.ATTEMPT_USE:

                    targets.append((entity, location))

                # Add items for manipulate
                #
                elif attempt_type is self.ATTEMPT_MANIPULATE:

                    # Add only entities of type item for manipulate
                    #
                    if entity.entity_type is fabula.ITEM:

                        targets.append((entity, location))

        # Add entities from rack for look at, use and manipulate
        #
        if attempt_type in (self.ATTEMPT_LOOK_AT, self.ATTEMPT_USE, self.ATTEMPT_MANIPULATE):

            # Check all entities in rack
            #
            for entity in self.host.rack.entity_dict.values():

                # Check if the entity is owned by someone and if the client is
                # that someone
                #
                if (entity.identifier in self.host.rack.owner_dict.keys()
                    and self.host.rack.owner_dict[entity.identifier] is self.host.client_id):

                    targets.append((entity, entity.identifier))

        # Remove item for use from the target list to prevent using it with
        # itself
        #
        if attempt_type == self.ATTEMPT_USE:

            selected_item = self.select_item_menu.list[self.select_item_menu.list_index][0]

            # If selected item is as well in target list remove it because an item
            # can not be used with itself
            #
            for target in targets:

                if target[0] is selected_item:
                    targets.remove(target)

        return targets

    def _get_rack_items(self):
        """Returns the items in the rack (inventory).
        """

        items = []

        for entity in self.host.rack.entity_dict.values():
```

```
                # Check if the entity is owned by someone and if client is
                # that someone
                #
                if (entity.identifier in self.host.rack.owner_dict.keys()
                    and self.host.rack.owner_dict[entity.identifier] is self.host.client_id):

                    items.append((entity, entity.identifier))

            return items

        def _get_room_entities(self):
            """Returns all entities in the room except for the client.
            """

            items = []

            for entity in self.host.room.entity_dict.values():

                if entity.identifier is not self.host.client_id:

                    items.append((entity, entity.identifier))

            return items
```

D Datenträger

Der Arbeit ist eine CD beigelegt, auf dem sich eine unter Windows ausführbare Version von Zauberwald befindet. Diese spiegelt nicht den aktuellsten Stand der Engine wieder, der aus dem angehängetem Quellcode entnommen werden kann. Außerdem befindet sich auf dem Datenträger eine digitale Version dieser Arbeit.

Eidesstattliche Erklärung

Hiermit versichere ich die Arbeit selbstständig angefertigt zu haben und keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel verwendet zu haben.

_____

Datum und Unterschrift des Verfassers

Diese Masterarbeit ist ein Prüfungsdokument. Eine Verwendung zu einem anderen Zweck ist nur mit dem Einverständnis von Verfasser und Prüfern erlaubt.