HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Kai Stüdemann

Konzeption und prototypische Implementierung einer VoIP Lösung für videospielbasierte virtuelle Filmdrehs

2014

Fakultät **Mathematik/Naturwissenschaften/Informatik**

BACHELORARBEIT

Konzeption und prototypische Implementierung einer VoIP Lösung für videospielbasierte virtuelle Filmdrehs

Autor:
Kai Stüdemann

Studiengang:
Informatik

Seminargruppe:
IF09w1-B

Erstprüfer:
Prof. Dr.-Ing. Mario Geißler

Zweitprüfer:
Herr Enrico Pisko, M. Sc.

Mittweida, Februar 2014

Bibliografische Angaben

Stüdemann, Kai: Konzeption und prototypische Implementierung einer VoIP Lösung für videospielbasierte virtuelle Filmdrehs, 24 Seiten, 4 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Mathematik/Naturwissenschaften/Informatik

Bachelorarbeit, 2014

Referat

Diese Arbeit beschäftigt sich mit der Integration einer VoIP-Lösung in ein Videospiel, welches für die Erstellung von Animationsfilmen genutzt werden soll. Hierzu wurde eine geeignete Software-Lösung ausgewählt. Der Client der gewählten Lösung wurde in das Spiel eingebunden und ein Konzept für eine serverseitig, automatisierte Verwaltung von Nutzern entwickelt und umgesetzt. Außerdem wurde die positionsabhängige Audio-Ausgabe angepasst und das Audio-Signal genutzt, um das Gesicht eines Spielcharakters beim Sprechen zu animieren.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Abkürzungsverzeichnis

ACL	Access Control List
API	Application Programming Interface
CELP	Code excited linear prediction
CELT	Constrained Energy Lapped Transform
DLL	Dynamic-link Library
GSM	Global System for Mobile Communications
GUID	Globally unique identifier
HRTF	Head Related Translation Function
IETF	Internet Engineering Task Force
PC	personal computer
TCP	Transfer Control Protocol
UDK	Unreal Development Kit
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol

1 Einleitung

In heutigen Online-Multiplayer-Spielen ist die Kommunikation mittels eines Voice-Chats ein wichtiger Bestandteil. Hierdurch können Spieler sich untereinander mittels Sprache austauschen, um Spielelemente zu kommentieren oder die Vorgehensweise zu koordinieren. Diese Form der Verständigung ist jedoch vom eigentlichen Spielgeschehen getrennt und kann deshalb sogar mittels getrennter Infrastruktur realisiert werden. In den letzten Jahren zeichnet sich eine neue Entwicklung ab, in der sich die Form von Online-Multiplayer-Spielen weiterentwickelt. Spiele wie Everquest 2[1] und Urban Legend[2] präsentieren Multiplayer-Szenarien in denen die Spieler und ihre Spielcharaktere zu virtuellen Darstellern werden, die durch ihre Interaktion Szenen erzeugen. Damit Spieler sich schauspielerisch in der virtuellen Welt ausdrücken können, mussten die Aktionsmöglichkeiten der Spielcharakter vergrößert werden. Bei Urban Legend geschah dies durch Komponenten, mit denen Spieler die Mimik und Gestik ihrer Charaktere steuern können. Eine Komponente die bisher fehlte, war die Nutzung der Sprache, als Mittel des schauspielerischen Ausdrucks. Hierzu muss eine Lösung entwickelt werden, welche die Sprachübertragung, nicht getrennt von der Spielwelt realisiert, sondern sie zum Teil der jeweils sprechenden Spielfigur werden lässt.

1.1 Zielstellung

Das Ziel dieser Arbeit ist die Integration einer VoIP-Lösung in ein Online-Multiplayer-Videospiel, welches als Bühne für die Erstellung von Animationsfilmen dienen soll. Hierzu soll eine geeignete, bereits existierende Software ausgewählt werden, welche anschließend für die Nutzung mit dem Videospiel angepasst wird. Bei der Integration ist zu beachten, dass sich das Gesprochene in die Spielwelt einfügt. Hierfür ist es notwendig, dass die Audio-Ausgabe abhängig von der räumlichen Beziehung zwischen dem Charakter eines sprechenden und dem des zuhörenden Spielers erfolgt. Weiterhin müssen die Sprachdaten genutzt werden, um das Gesicht des Spielcharakters eines sprechenden Spielers zu animieren. Dies ist notwendig um den anderen Spielern einen klaren Indikator darüber zu geben, welcher Spieler gerade spricht und die Sprachkommunikation auch visuell in der Spielwelt zu verankern.

[1] vgl. [1]
[2] vgl. [4]

2 Grundlagen

2.1 Gamecast

Gamecast ist eine Forschungsgruppe der Hochschule Mittweida. Deren Ziel die Generierung von Filmen auf Basis von Multiplayer-Videospielen ist. Zur Erreichung dieses Ziel wurde im Rahmen der Forschungsarbeit ein Protyp entwickelt. Außerdem wurde ein Spiel produziert, mit welchem das Konzept und der Prototyp getestet werden konnten.

2.1.1 Urban Legend

Urban Legend ist ein Videospiel, dass die im Rahmen von Gamecast erforschten Techniken anschaulich präsentiert[3]. Das Spiel bietet ein futuristisches Szenario, in welchem ein totalitärer Überwachungsstaat seine Bürger mit unterschwelligen Botschaften in den Medien ruhig stellt. Auf der Gegenseite des Staates steht eine Gruppe von Rebellen[4]. In der Spielmechanik schlägt sich dies in der Form nieder, dass ein Spieler die Möglichkeit hat sich entweder den Rebellen oder den Streitkräften der Regierung anzuschließen, wobei sich letztere in Polizisten und Techniker unterteilen. In einer normalen Spielpartie haben dann die Rebellen das Ziel möglichst viele WLAN-Hotspots zu hacken. Während die Polizisten mit dem Schutz der Hotspots und die Techniker mit der Reparatur von gehackten Hotspots beauftragt sind. Weiterhin verfügt Urban Legend über eine Komponente welche das Verhalten eines Spielers beobachtet und ihm basierend auf seinen Präferenzen zusätzliche Aufgaben zuteilt. So würde ein Rebellenspieler der direkte Konfrontation mit Polizisten sucht, indem er diese blendet, actionreichere Aufgaben erhalten, als ein Spieler der als Rebell versucht möglichst unentdeckt Hotspots zu hacken.

2.2 UDK

Zur Entwicklung von Urban Legend wurde das Unreal Development Kit, kurz UDK, verwendet. Das UDK ist eine kostenlose Version der Unreal Engine 3. Die Entwicklung von Spielen mit dem UDK erfolgt größtenteils mit UnrealScript einer Skript-Sprache, deren Syntax an C oder Java erinnert. Die Besonderheit bei der Arbeit mit UnrealScript ist, dass die Spielfunktionalität bereits in einer Reihe von Grundklassen implementiert ist. Eigene Spielkonzepte werden dabei implementiert, indem eigene Klassen von den Grundklassen abgeleitet und Funktionen entsprechend erweitert oder überschrieben

[3] vgl. [4]
[4] vgl. [3]

werden. Um logisch zusammengehörende Funktionen zusammenzufassen werden diese häufig in eine eigene Klasse in der Hierarchie der am Ende verwendeten Klasse eingefügt.

Die wichtigsten Grundklassen in UnrealScript sind:

- `Object` ist die Basisklasse in UnrealScript.
- `Actor` repräsentiert Objekte welche eine Position und Ausrichtung in der Spielwelt besitzen und Event-Funktionen definieren, welche bei bestimmten Ereignissen von der Engine aufgerufen werden. Hierzu zählt das Tick-Ereignis, welches periodisch, mehrfach in der Sekunde ausgeführt wird.
- `GameInfo` ist von Actor abgeleitet. Sie ist die Hauptklasse eines Spiels. Sie ist für die Initialisierung und Deinitialisierung der Spielwelt und der zu einem Spieler gehörenden Klassen (im Normalfall ein Pawn und ein PlayerController) verantwortlich
- `Pawn` ist von Actor abgeleitet. Sie repräsentiert einen Spielcharakter
- `Controller` ist von Actor abgeleitet. Sie ist für die Kontrolle eines Pawns verantwortlich.
- `PlayerController` ist von Controller abgeleitet. Stellt die Schnittstelle des Spielers zum Spiel dar. Sie wertet Benutzereingaben aus und leitet Informationen an die grafische Oberfläche weiter.

Weiterhin ist es auch möglich C++- bzw. C#-Code als DLL in den UDK-Prozess zu laden, indem an das Ende einer Klassendeklaration das Schlüsselwort DLLBind mit dem Namen der DLL in Klammern gestellt wird. Die durch die DLL exportierten Funktionen müssen dann noch mit dem Modifizierer dllimport final im Körper der Klasse deklariert werden um in UnrealScript zur Verfügung zu stehen[5].

2.3 Softwaretechnisches Vorgehensmodell

Bei der Entwicklung wurden Prinzipien aus Scrum verwendet. Da die Entwicklung von einer Einzelperson durchgeführt wurde, waren viele Scrum-Techniken nicht anwendbar. Dennoch war die iterative Entwicklung in Sprints, besonders bei der Integration des Clients, günstig um funktional abgeschlossene Zwischenschritte zu erreichen. Auch wenn erst mit dem letzten Sprint ein Ergebnis erreicht wurde, welches den Anforderungen entsprach.

[5] vgl. [5]

2.4 Anforderungen an die Anwendung

Funktionsanforderungen, die sich direkt aus der Zielstellung in 1.2 ergeben, sind die Beachtung der räumlichen Beziehung bei der Audio-Ausgabe und die Anwendung der Sprachdaten zur Animation des Gesichtes eines Charakters. Weiterhin sollte sich die integrierte VoIP-Lösung möglichst nahtlos in Urban Legend einfügen.

Für die Nutzung mit Urban Legend ergeben sich zusätzliche Anforderungen. Hierzu zählt, dass ein Spieler, beim gedrückt Halten einer Taste, zu allen anderen Spielern seiner Fraktion, ohne Berücksichtigung der Entfernung, sprechen können soll. Dies soll im Spiel wird im Spiel als “Funkruf” zwischen den Mitgliedern einer Fraktion verwendet. Zusätzlich erfolgt die Verwaltung der Spiele-Server-Instanzen von Urban Legend automatisch, indem bei Bedarf neue Instanzen erzeugt bzw. bei Beendigung einer Spielpartie die entsprechende Instanz beendet wird. Daher muss die Trennung der Sprachkommunikation für die einzelnen Spiele-Server-Instanzen auch automatisiert erfolgen.

3 Vorbetrachtung

3.1 Außenohrübertragungsfunktion

Eine Aussenohrübertragungsfunktion (engl. Head Related Transfer Function, HRTF), beschreibt für einen Punkt im Raum wie sich eine Schallwelle durch die Form des Kopfes bezüglich ihrer Laufzeit, Amplitude und Phase für beide Ohren ändert. Dies kann dazu genutzt werden um für ein beliebiges Audiosignal, bei der Wiedergabe über Stereo-Kopfhörer, den Eindruck zu erwecken, es käme aus der Richtung des durch die HRTF definierten Punktes.[6]

3.2 Videospielorientiertes VoIP

Für die Nutzung von VoIP in einem Multiplayer-Videospiel ergeben sich andere Anforderungen, als für die Anwendungsszenarien, welche von den klassischen VoIP-Protokollen wie H.323 oder SIP abgedeckt werden. Letztere bilden durch Funktionen, wie zum Beispiel Rufumleitung oder Anklopfen, die Funktion eines klassischen ISDN-Netzes nach. Bei der Nutzung mit einem Videospiel, nimmt an einer Sitzung häufig eine größere Anzahl von Nutzern teil. Um die Kommunikation zwischen diesen zu regeln, kommt bei den drei für den Einsatz mit Videospielen verbreitetsten VoIP-Lösungen[7] eine kanalbasierte, dynamische Nutzerverwaltung zum Einsatz.

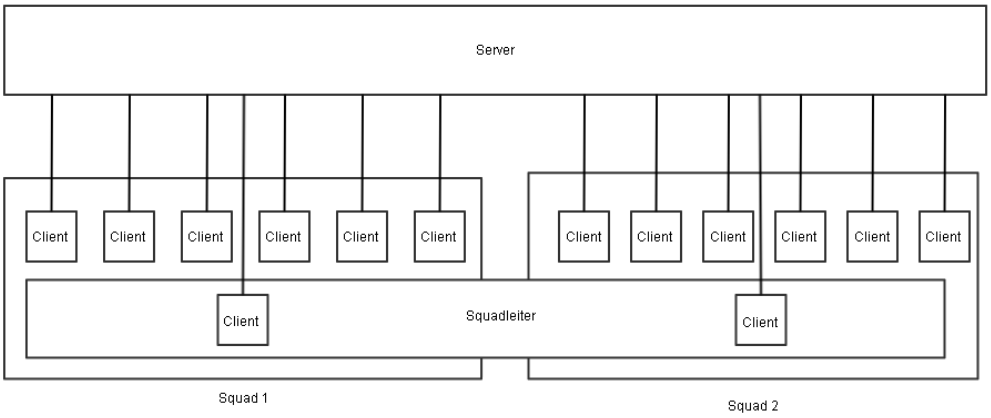

Abbildung 3.1: Beispiel für eine VoiP-Konfiguration für ein Multiplayer-Shooter-Spiel

Ein Beispiel, dafür wie dies in einem Multiplayer-Shooter genutzt werden kann, ist in

[6] vgl. [13] S. 425ff
[7] siehe 3.3

Abbildung 3.1 zu sehen. Hierbei ist ein Team in zwei Squads unterteilt, welche getrennt voneinander agieren. Die normalen Spieler müssen nur mit den Mitgliedern ihres Squads kommunizieren. Lediglich die beiden Squadleiter müssen miteinander reden können. Nach Beendigung einer Spielrunde könnten die Squad-Mitglieder neu aufgeteilt werden. Ein weiteres Feature für die Nutzung mit Videospielen ist die Übertragung und Anwendung von Positionsdaten zur räumlichen Abbildung des Sprachsignals. Aus diesem Grund verwenden Lösungen, welche für den Einsatz im Zusammenhang mit Videospielen konzipiert wurden, eigene Protokolle, welche diese Funktionalitäten unterstützen.

3.3 Verwendete Codecs

Speex

Speex ist ein patent- und lizenzgebührenfreier Audio-Codec, der für die Kodierung von Sprache optimiert ist. Er unterstützt Abtastraten bis 32 kHz und variable Bitraten welche im Bereich von 2.15 kbps bis 44 kbps liegen können. Für die Kodierung wird Code Excited Linear Prediction (CELP) verwendet. In die Ermittlung der variablen Bitrate ist außerdem eine Sprachpausenerkennung (VAD) integriert. Diese kann im Zusammenhang mit mit der ebenfalls unterstützten discontinuous transmission (DTX) genutzt werden, um in Sprachpausen keine Daten übertragen zu müssen. Durch die bei Speex verwendete Größe für Audio-Frames und einen zusätzlichen Lookahead erzeugt die Kodierung mit Speex eine Latenz von 30 ms bei der Schmalband (8 kHz)- und 34 ms bei der Breitbandkommunikation (16 kHz). Speex enthält darüber hinaus einen Präprozessor, welcher zur Rauschunterdrückung und automatischen Verstärkungsregelung genutzt werden kann und eine bessere Sprachpausenerkennung enthält als der eigentliche Codec[8].

CELT

CELT steht für Constrained Energy Lapped Transform und beschreibt einen ebenfalls patent- und lizenzgebührenfreier Audio-Codec. Der, im Gegensatz zu Speex, aber auch für Musik geeignet ist. Er unterstützt Abtastraten bis 48 kHz und konstante Bitraten zwischen 32 und mehr als 128 kbps. Außerdem unterstützt er Packet loss concealment, was es erlaubt den Verlust einzelner Pakete zu überbrücken. Das bedeutendste Feature von CELT ist die niedrige Latenz bei der Kodierung, welche, abhängig von der Konfiguration, durchschnittlich 5 bis 22.5 ms beträgt[9].

[8] vgl. [17]
[9] vgl. [18]

Opus

Opus ist ein patent- und lizenzgebührenfreier Audio-Codec. Er wurde als RFC 6716 von der IETF standardisiert. Er besitzt 2 Schichten. Eine davon basiert auf CELT, wodurch Opus auch alle Features von CELT besitzt. Die zweite Schicht basiert auf SILK einem von Skype entwickelten Codec, zur Kodierung von Sprachdaten. Dieser unterstützt Abtastraten bis 16 kHz und variable Bitraten mit mindestens 6 kbps. SILK verursacht bei der Kodierung eine Latenz von 15 bis 65 ms. Opus besitzt außerdem einen Hybrid-Modus, bei dem die Frequenzanteile unter 8 kHz mit SILK und die darüber mit CELT kodiert werden[10]. Er hat sich in Hörtests, bei der Nutzung vergleichbarer Datenraten, Speex als klanglich überlegen gezeigt[11].

3.4 Auswahl einer geeigneten Software

Im Folgenden werden die drei für den Einsatz mit Videospielen verbreitetsten VoIP-Lösungen verglichen. Alle drei bieten sowohl eine Client-, als auch eine Server-Anwendung.

3.4.1 Ventrilo

Ventrilo wird von Flagship Industries als Freeware angeboten. Wobei der kostenlose Server auf die gleichzeitige Nutzung mit acht Clients beschränkt ist. Es bietet zwar eine kanalbasierte Nutzerverwaltung, allerdings keine positionsabhängige Audio-Ausgabe. Es unterstützt neben Speex, auch den aus dem Mobilfunkbereich stammenden Audio-Codec GSM 06.10. Da keine Möglichkeit Ventrilo in eine andere Anwendung einzubinden existiert, ist es als Grundlage für die weitere Entwicklung ungeeignet.

3.4.2 TeamSpeak 3

TeamSpeak wird von der TeamSpeak Systems GmbH entwickelt und vertrieben. TeamSpeak wird sowohl als eigenständige Server- und Client-Anwendung angeboten, als auch in Form eines SDK, welches es ermöglicht TeamSpeak in eigene Anwendungen zu integrieren. Für die Nutzung eines Servers mit mehr als 32 Clients ist eine Lizenz erforderlich, welche für nichtkommerzielle Projekte aber kostenlos erlangt werden kann[12]. Es verfügt sowohl über eine kanalbasierte Nutzerverwaltung, als auch eine positionsabhängige Audio-Ausgabe. Die eigenständigen TeamSpeak Anwendungen unterstützen Opus, CELT und Speex. Während mit dem SDK lediglich Speex genutzt werden kann.

[10] vgl. [15]
[11] vgl. [16]
[12] vgl. [11]

3.4.3 Mumble

Mumble ist quelloffen und unter der BSD-Lizenz verfügbar.[13] Es verfügt sowohl über eine kanalbasierte Nutzerverwaltung, als auch eine positionsabhängige Audio-Ausgabe[14]. Zu den unterstützten Codecs zählen neben Opus, auch Speex und CELT. Unabhängig vom gewählten Codec wird immer der Speex Präprozessor verwendet. Von Mumble ist, mit libmumble, lediglich eine Implementation des Protokolls als einbettbare Bibliothek verfügbar. Da der Quellcode von Mumble aber frei verfügbar ist, kann auch die Desktop-Client-Anwendung für die Integration in eigene Anwendungen angepasst werden.

3.4.4 Gewählte Software

Laut der Dokumentation des TeamSpeak-SDKs besteht keine Möglichkeit die positionsabhängige Audio-Ausgabe zu deaktivieren[15]. Dies würde die Umsetzung, “Funkrufe” in Urban Legend, unmöglich machen. Die Beschränkung auf Speex ist ein weiterer Nachteil von TeamSpeak. Daher wurde Mumble für die Umsetzung von VoIP in Urban Legend gewählt, auch wenn der Client erst für die Einbettung in eine andere Software angepasst werden muss.

[13] vgl. [10]
[14] vgl. [9]
[15] vgl. [12]

4 Umsetzung

4.1 Integration des Clients

Das Ziel bei diesem Schritt war es den Mumble-Client möglichst nahtlos in Urban Legend einzubetten. Hierzu war es zunächst notwendig festzulegen in welcher Weise Mumble in Urban Legend eingebunden werden soll.

Als eigenständiger Prozess

Es könnte eine DLL erstellt werden, welche den Mumble-Client als eigenständigen Prozess startet. Die Kommunikation zwischen diesem und Urban Legend könnte dann, mittels einer beliebigen Interprozesskommunikationsmethode, über die DLL erfolgen oder Mumble könnte, über TCP, direkt mit dem UDK kommunizieren, wie dies schon bei anderen Komponenten des Gamecast-System, wie z.B. der Mimikerkennung geschehen ist.

Ein Problem dieses Ansatzes ist das sogenannten ducking in Windows 7. Mit der Veröffentlichung von Windows Vista führte Microsoft eine neue Low-Level-Audio-API ein, welche als Grundlage für alle High-Level-Audio-APIs wie DirectSound und die Windows multimedia waveXxx and mixerXxx functions dient[16]. Diese definiert Rollen denen jeweils ein Audio-Aus- und Eingabegerät zugewiesen sein muss. Von besonderem Interesse ist hierbei die mit Windows 7 eingeführte Rolle **eCommunications**, welche dem Gerät zugewiesen ist das in den Windows Sound-Einstellungen als Standardkommunikationsgerät bezeichnet wird[17]. Wird sowohl das Ein- als auch das Ausgabegerät mit der Rolle **eCommunications** von einer Anwendung angesprochen, erkennt Windows dies und löst das sogenannte ducking aus. Dies bedeutet das die Lautstärke für andere Anwendungen um, ein in den Sound-Einstellungen festlegbares, Niveau abgesenkt wird. Alternativ kann eine Anwendung auch entscheiden nicht das Standard ducking-Verhalten zu verwenden und eigene Funktionen zur Behandlung von ducking-Ereignisse zu definieren[18].

UDK besitzt keine eigene Behandlung von ducking-Ereignissen. Wenn Mumble als eigener Prozess ausgeführt wird, würde dies auf PCs mit Windows 7 dazu führen, dass die Lautstärke der Spiel-Sounds abgesenkt wird sobald Mumble aktiviert wird. Dieses Verhalten entspricht nicht der gewünschten Spielerfahrung. Zwar lässt sich ducking systemweit deaktivieren, allerdings kann vom Nutzer nicht erwartet werden für die Nutzung

[16] vgl. [6]
[17] vgl. [7]
[18] vgl. [8]

von Urban Legend in die Systemeinstellungen einzugreifen. Daher ist es nötig, dass die Audioein- und ausgabe von Mumble über den UDK-Prozess erfolgt.

Als eigenständiger Prozess mit ausgelagerter Audioein- und ausgabe

Eine weitere Möglichkeit wäre es wie zuvor den Client als eigenständigen Prozess über eine DLL auszuführen, aber zusätzlich auch die Audioein- und ausgabe in die DLL zu verlegen. Die Audiodaten könnten dann zum Beispiel über einen Ringpuffer in einem shared memory segment übertragen werden. Diese Lösung hätte die Komplexität, bezüglich der Synchronisation zwischen den beiden Prozessen, deutlich erhöht.

Als Thread im UDK-Prozess

Der dritte mögliche Ansatz ist die vollständig Verlagerung des Mumble-Clients in eine DLL. Welche diesen als Thread im UDK-Prozess ausführt.

Die letztgenannte Option ist die geeignetste. Da die Umgehung des ducking Verhaltens als funktionsentscheidend anzusehen ist und der erhöhten Komplexität durch die ausgelagerter Audioein- und ausgabe kein wirklicher Mehrwert gegenüber steht.

Das Ziel der ersten Iteration war es die Erstellung eines minimalen konsolenbasierten Mumble-Clients. Dieser soll sich beim Start mit einer fest programmierten Serveradresse verbinden und die Sprachkommunikation über diesen erlaubt. Hierzu mussten die für den vorgesehenen Anwendungsfall nicht mehr benötigten Klassen entfernt werden. Daher musste bestimmt werden welche Klassen weiterhin benötigt werden. Diese werden in der folgenden Übersicht dargestellt:

- `ACL` Beschreibt die Access Control List, die Rechteverwaltung von Mumble.
- `AudioInput` Basisklasse für die Implementierung einer Audioeingangsschnittstelle.
- `AudioOutput` Basisklasse für die Implementierung einer Audioeingangsschnittstelle.
- `AudioOutputUser` Basisklasse für die Implementierung von Klassen, welche Audio-Daten dekodieren und über AudioOutput ausgeben.
- `AudioOutputSpeech` Implementierung von AudioOutputUser für Audio-Frames, welche aus Sprachpaketen stammen.
- `Cert` Die Verwaltung von SSL-Zertifikaten.
- `Channel` Repräsentiert einen Kanal (siehe nächster Abschnitt).
- `ClientUser` Wird zur Verwaltung von Daten anderer User benötigt.
- `Connection` Diese Klasse stellt die low level Komponente der Netzwerkanbindung dar. Sie sendet und empfängt Netzwerkpakete, über Socket-Verbindungen.
- `CryptState` Ist für die Ver- und Entschlüsselung von Paketen verantwortlich.

- `DXAudioInput / DXAudioOutput` Implementierung von AudioInput bzw. AudioOutput für die DirectSound Audio API.
- `Global` Container für globale Variablen.
- `ServerHandler` Die Hauptschnittstelle bei der Kommunikation mit dem Server. Ist für den Empfang und das Versenden von Protokollnachrichten verantwortlich.
- `User` Repräsentiert einen Nutzer.

Die nicht mehr benötigten Klassen umfassen hauptsächlich die grafische Oberfläche, aber auch z.B. die Erkennung von Tastenkombinationen oder eine Komponente welche sich mittels API hooking zwischen 3D-Anwendung und die verwendete Render-API schalten kann um ein eigenes Overlay über die Grafik der Anwendung zu zeichnen. Die Klasse `MainWindow`, welche das Hauptfenster des Clients repräsentiert, enthält Funktionen für die Verbindung und Trennung zum Server. Diese wurden in eine neue Klasse verlegt.

In der zweiten Iteration entstand auf Grundlage des minimalen Mumble-Clients eine DLL welche die folgenden Funktionen exportiert.

- `void start(wchar_t* server, wchar_t* channelpath, wchar_t* username, wchar_t* password)`: Mit dieser Funktion wird der Client initialisiert, eine Verbindung mit dem Server hergestellt und der übergebene Kanal betreten.
- `void quit()`: Trennt die Verbindung zum Server und deinitialisiert die Komponenten des Clients.

Auf der Seite des UDK wird die Einbindung über die Klassen `GC_VoiceChat` und `GC_PlayerControlller_VoiceChat` realisiert. `GC_VoiceChat` ist eine einfache Wrapper-Klasse welche die DLL einbindet und die exportierten Funktionen in UnrealScript zugänglich macht. `GC_PlayerControlller_VoiceChat` befindet sich in der Vererbungshierarchie des für Urban Legend verwendeten PlayerControllers. Er besitzt ein `GC_VoiceChat`-Objekt. Sobald er einen Kanalpfad, Benutzernamen und das zugehörige Passwort erhält startet er den Client.

Die vierte Iteration hatte den Ersatz der Konfigurationsdialoge zum Ziel. Einige davon, wie die Belegung von Tastenbefehlen, werden für den vorgesehenen Anwendungsfall nicht benötigt. Für diese muss dementsprechend kein Ersatz geschaffen werden. Andere Funktionen deren Konfigurierbarkeit entfallen kann sind jene, die bei Urban Legend mit festen Werten verwendet werden. Hierzu zählt die Parametrisierung der ortsabhängigen Audio-Ausgabe. Eine Vielzahl der verfügbaren Optionen sind aber auch für die Nutzung in Urban Legend von Bedeutung. Der Autor dieser Arbeit hatte bereits ein Starterprogramm für Urban Legend entwickelt. Dieses bot bereits verschiedene Einstellungsmöglichkeiten für die Grafik des Spiel. Es war daher sinnvoll die Optionen für Mumble in dieses Programm zu integrieren. Da das Starterprogramm wxWidgets zur Darstellung der Oberfläche nutzt musste der Dialog neu erstellt werden. Die Funktio-

nalität zur Messung der Lautstärke bzw. des Signal-Rausch-Abstands konnte hingegen größtenteils übernommen werden.

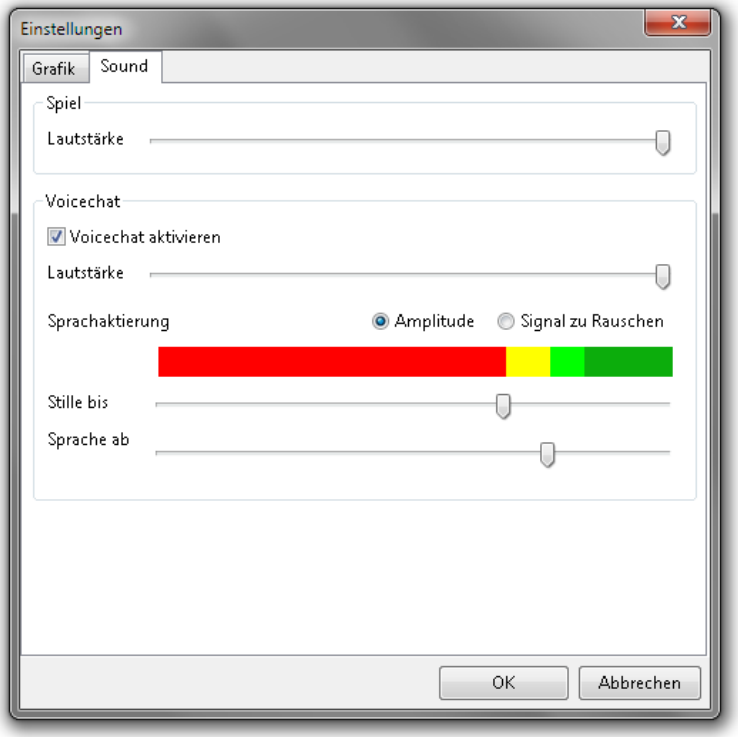

Abbildung 4.1: Urban Legend Konfigurationsdialog

4.2 Nutzerverwaltung auf dem Server

Auf einem Server sollen mehrere Instanzen des Spiele-Servers gleichzeitig laufen können. Diese sollen zur Sprachkommunikation aber alle die selbe Murmur[19]-Instanz nutzen. Dies kann mit der in Mumble enthaltenen Funktion einen Server in Kanäle aufzuteilen realisiert werden. Kanäle bieten eine sehr flexible Möglichkeit zu konfigurieren ob Nutzer einander hören können. Die Kanäle verfügen außerdem über ein, Access Control List (ACL) genanntes, Rechteverwaltungssystem. Rechte werden für Benutzergruppen vergeben wobei jeder Kanal eigene Gruppen definiert oder die Gruppen eines, in der Hierarchie über ihm gelegen, Kanals erbt. Es werden folgende Rechte unterschieden:[20]

- **Write** Gibt dem Nutzer Vollzugriff auf den Kanal. Ein Nutzer mit diesem Recht kann die Rechte des Kanals bearbeiten, den Kanal löschen und verfügt implizit über alle anderen Rechte.

[19] Murmur ist die Server-Anwendung von Mumble.
[20] vgl. [20]

- **Traverse** Ermöglicht einem Nutzer sich auf einem Unterkanal anzumelden, sofern er in diesem über die entsprechenden Rechte verfügt.
- **Enter** Erlaubt es einem Nutzer einem Kanal beizutreten.
- **Speak** Ein Nutzer mit diesem Recht kann von den Nutzern in diesem Kanal gehört werden, wenn er diesem beigetreten ist.
- **Mute** / **Deafen** Erlaubt es einem Nutzer andere Nutzer stumm bzw. taub zu setzen.
- **Move** / **Kick** Berechtigt einen Nutzer dazu andere Nutzer in einen anderen Kanal zu verschieben bzw. sie vom Server zu werfen.
- **Make Channel** Dieses Recht erlaubt es einem Nutzer neue Unterkanäle zu diesem Kanal anzulegen.
- **Make Temporary Channel** Ähnlich wie Make Channel, allerdings können nur temporäre Kanäle erstellt werden. Diese werden gelöscht, sobald der letzte Nutzer sie verlässt.
- **Link Channel** Gibt dem Nutzer das Recht Kanäle miteinander zu verbinden. Wenn zwei Kanäle miteinander verbunden sind, werden die Sprachpakete der Nutzer des einen Kanals auch in den mit ihm verbundenen Kanal übertragen, solange die Nutzer das Speak Recht im verbundenen Kanal besitzen.
- **AltSpeak** Erlaubt es dem Nutzer bei Betätigung einer Taste in diesen Kanal zu sprechen, auch wenn er kein Mitglied dieses Kanals ist und der Kanal nicht mit dem Kanal, in dem sich der Nutzer gerade aufhält, verbunden ist.

Um den oben beschriebenen Fall und die “Funkrufe” in Urban Legend umzusetzen wurde die folgende Kanalstruktur entworfen:

- **Root** Dieser Kanal existiert auf dem Mumble Server grundsätzlich. Alle Nutzer erhalten für diesen Kanal lediglich das Traverse Recht.

  - Dieser Kanal gehört zu einer spezifischen Spiele-Server-Instanz. Der Name des Kanals wird wie im nächsten Abschnitt beschrieben zufällig generiert. Für diesen Kanal erhalten nur die zum Spiel gehörenden Spieler das Traverse Recht.

    * **Police** Dieser Kanal ist für die Polizisten einer spezifischen Spiele-Server-Instanz gedacht. Diese erhalten hier die Rechte Traverse, Enter und Speak. Die Rebellen erhalten das Speak Recht für diesen Kanal, da beide Kanäle miteinander verbunden werden, sodass die Polizisten die Rebellen hören können.
    * **Rebels** Analog zu Police mit dem Unterschied, dass dieser Kanal für die Rebellen gedacht ist und die Polizisten das Speak Recht erhalten.

Für die Automatisierung der Nutzerverwaltung bietet sich die in den Murmur integrierte Ice-Konfigurationsschnittstelle an. Ice ist eine von der Firma ZeroC entwickelte Remote

Procedure Call Bibliothek. Diese erlaubt es über ein in einer Metadatei beschriebenes Interface auf Proxy-Objekte welche dieses implementieren zuzugreifen. Der Zugriff kann dabei lokal zwischen unterschiedlichen Prozessen oder über das Netzwerk erfolgen. Listing 4.1 zeigt beispielhaft wie Ice genutzt werden kann um einen neuen Kanal auf einem lokalen Mumble-Server anzulegen.

```
communicator = Ice::initialize();
//Fordert ein Proxy-Objekt der Klasse Meta einer Murmur-Instanz,
    welche auf dem lokalen PC läuft und Port 6502 für Ice nutzt, an.
    Über Meta kann auf die virtuellen Server, welche auf dem Murmur
    laufen, zugegriffen werden.
MetaPrx meta = MetaPrx::checkedCast(communicator->stringToProxy("Meta:
    tcp -h localhost -p 6502"));
ServerList list = meta->getAllServers();
//In unserer Konfiguration läuft immer nur ein virtueller Server,
    welcher den Index 0 besitzt.
ServerPrx server = list[0];
//Legt einen neuen Kanal unter dem Root-Kanal an. Der Root-Kanal hat
    immer den Index 0.
int gameChannelId = server->addChannel("newChannel", 0);
communicator->destroy();
```

Listing 4.1: Einrichten eines neuen Kanals auf einem Mumble Server mittels Ice

Um diese anzusprechen muss ein Ice Client für die Integration in den UDK Server geschaffen werden. Hierfür wird erneut auf die DLL-Bind Mechanik zurückgegriffen. Die entwickelte DLL exportiert die folgenden Funktionen:

- `void createChannel(wchar_t* channelname)`: Generiert eine GUID als Name für einen neuen Kanal. Anschließend prüft sie ob auf dem Server bereits ein Kanal mit diesem Namen existiert. Wenn er noch nicht existiert wird ein neuer Kanal mit beiden Unterkanälen erstellt und der Name in der Variable channelName zurückgegeben.
- `void registerUser(wchar_t* channelname, wchar_t* faction, wchar_t* username, wchar_t* password)`: Generiert zwei GUIDs als Name und Passwort für einen neuen Nutzer. Anschließend prüft sie ob auf dem Server bereits ein Nutzer mit diesem Namen existiert. Wenn er noch nicht existiert wird ein neuer Nutzer erstellt und in die Gruppen mit den benötigten Rechten eingetragen. In den Variablen username und password wird der Benutzername bzw. das Passwort des neuen Nutzers zurückgegeben.
- `void removeUser(wchar_t* username)`: Entfernt den Nutzer mit dem übergebenen Benutzername vom Server.
- `void removeChannel(wchar_t* channelname)`: Entfernt den Kanal mit dem übergebenen Name vom Server.

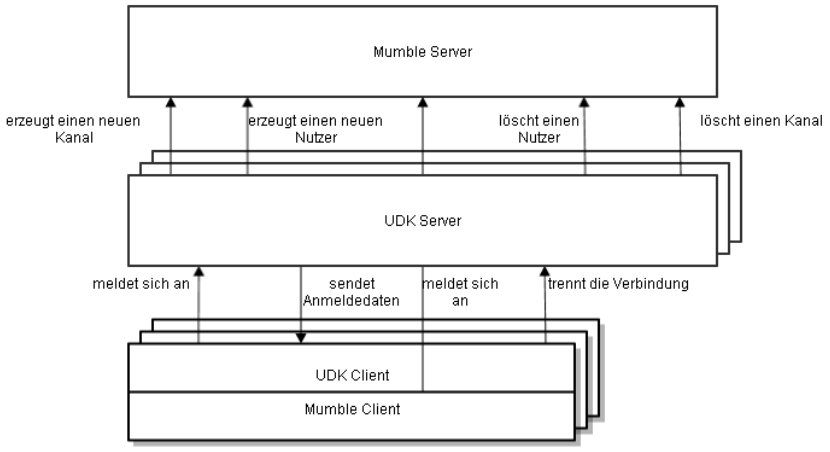

Abbildung 4.2: Ablauf der Kommunikation zwischen dem UDK und Mumble

In UnrealScript wird diese DLL von der Klasse `GC_VoiceChat_Server` gewrappt. Die Klasse `GC_GameInfo` ist die GameInfo Klasse von Urban Legend. Sie besitzt die Funktion `PostBeginPlay` welche beim Start eines Servers aufgerufen wird, nachdem dieser vollständig vom UDK initialisiert wurde. In ihr wird `GC_VoiceChat_Server` instanziiert und createChannel aufgerufen. `GC_PlayerController_VoiceChat` erbt die Funktion `ServerSetNewPlayerName` von `GC_PlayerController_Input`. Diese Funktion wird aufgerufen sobald ein Spieler die Charakterauswahl verlässt und dem eigentlichen Spiel beitritt. Sie eignet sich daher dafür registerUser aufzurufen. Der so erhaltene Benutzername mit Passwort und der Kanalpfad wird anschließend auf den Client übertragen. Dort werden diese Daten genutzt um start aufzurufen und eine Verbindung zum Mumble Server herzustellen. Die Funktion Logout der Klasse GC_GameInfo wird aufgerufen wenn ein Spieler das Spiel verlässt. Daher wird hier removeUser für den entsprechenden Spieler ausgeführt. Nach Beendigung einer Spielpartie wird die Funktion closeServer der Klasse `GC_Game_Manager` aufgerufen. `removeChannel` wird daher an dieser Stelle ausgeführt. Weiterhin wird hier über die eventuell noch im Spiel befindlichen PlayerController iteriert und `removeUser` auf diese angewendet. Da es passieren kann das z.B. bei einem Verbindungsabbruch Logout für einen Spieler nicht ausgeführt wurde.

Um die “Funkrufe” in Urban Legend, zu realisieren wurde eine Änderung am Protokoll vorgenommen. Das erste Byte im Header der Pakete für Sprachdaten codiert in den letzten fünf Bit das Ziel eines Pakets. Hierbei steht der Wert “0” für die normale Weiterleitung an alle Nutzer im selben und verbundenen Kanälen. Während “31” den Server anweist das Paket an den Absender zurückzusenden (Loopback). Der Wert “1” sorgt dafür, dass das Paket nicht an Nutzer in verbundenen Kanälen gesendet wird. Die Werte “2” bis “30” geben an das ein Paket nur an einen bestimmten Kanal oder Nutzer weitergeleitet werden soll, welcher zuvor über eine VoiceTarget Nachricht registriert wurde

(Flüstern)[21]. “Flüstern” eignet sich nicht für die entfernungunabhängige Audio-Ausgabe, da ein Spieler der zu seiner Fraktion spricht immer noch ortsabhängig von den Spielern der anderen Fraktion gehört werden soll. Es wird auch sonst für Urban Legend nicht benötigt. Daher wurde die Zielangabe auf vier Bit eingeschränkt, wobei nun “15” für den Loopback und “2” bis “14” für die Angabe eines “Flüster”-Zieles genutzt werden können. Das so freigewordene Bit konnte nun genutzt werden um Sprachpakete zu markieren, für welche die Wiedergabe für Nutzer des selben Kanals entfernungunabhängig erfolgen soll. Die Client-DLL wurde um die Funktion `void setSpeakDirectlyToChannel(bool state)` erweitert, mit welcher gesteuert werden kann, ob das Bit für versendete Sprachpakete gesetzt werden soll.

[21] vgl. [21]

5 Integration ins Spiel

5.1 Räumliche Abbildung

Die positionsabhängige Audio-Ausgabe von Mumble basiert auf einer entfernungsabhängigen Dämpfung der Lautstärke. Diese kann mit der Distanz, bis zu welcher keine Dämpfung erfolgen soll und der, ab welcher die Lautstärke ihren Minimalwert erreicht, und einer minimalen und maximalen Lautstärke parametrisiert werden. Diese Werte wurden so festgelegt, dass die Lautstärke bis zu einer Entfernung von 7 Metern 100 Prozent beträgt. Bei größeren Entfernungen fällt Sie stetig, bis sie schließlich bei 20 Metern 0 Prozent erreicht. Diese Entfernungswerte wurden, zusammen mit der Projektleitung von GameCast empirisch ermittelt. Ein Vorteil der aus der festen Parametrisierung entsteht ist, dass der Server, basierend darauf ob sich ein Spieler in der Hörweite eines sprechenden Spielers, entscheiden kann ob er ein Paket an ersteren weiterleiten soll. Hierdurch kann die genutzte Bandbreite reduziert werden.

Für die Erzeugung von Raumklang fällt die Dämpfung der Lautstärke für die Lautsprecher, welche sich in der Richtung befinden, aus der im Spiel der Ton kommen müsste weniger stark aus als auf der abgewandten Seite. Daraus ergibt sich, dass die Anzahl der Richtungen, denen ein Tonsignal zugeordnet werden, kann der Anzahl verwendeten Lautsprecher, entspricht. Bei der Nutzung mit Lautsprecherkonfigurationen mit fünf oder sieben Lautsprechern, erlaubt dies eine gute Lokalisierbarkeit der Schallquelle. Allerdings empfiehlt sich für die Nutzung von Urban Legend der Gebrauch eines Headset, um Rückkopplungen zu vermeiden. Auf dem Markt sind auch Headsets erhältlich, bei denen in jeder Kopfhörermuschel mehrere Lautsprecher verbaut sind, welche ausgerichtet sind um Raumklang zu erzeugen. Diese sind aber deutlich teurer als Headsets, welche lediglich zur Stereofonie fähig sind. Diese haben bei der oben beschriebenen Vorgehensweise das Problem, dass sich für eine virtuelle Tonquelle nur feststellen lässt, ob sich diese links oder rechts neben, nicht aber ob sie sich vor oder hinter, dem Spieler befindet. Um auch bei der Nutzung eines Stereo-Headset Schallquellen lokalisierbar zu machen können HRTFs verwendet werden. Die quelloffene Codec-Sammlung ffdshow[22] verfügt über eine Komponente mit HRTFs für die Lautsprecherpositionen eines 5.1-Surround-Lautsprecher-Systeme. Dies kann in Mumble integriert werden, indem die Audioausgabe für 5 Lautsprecher initialisiert wird. Nach der Abmischung des Sounds auf 5 Kanäle durch die Mumble-Komponente, kann dieser durch den aus ffdshow übernommenen Code für Stereo-Kopfhörer zur Ausgabe vorbereitet werden.

[22] [23]

5.2 Abbildung auf den Spielcharakter

Ein Kernpunkt des Gamecast Systems ist die Abbildung des Gesichtsausdrucks eines Spielers auf seine Spielfigur. Dies wird durch die SHORE-Engine des Frauenhofer Instituts für integrierte Schaltungen realisiert. Diese erlaubt es über eine Webcam die Mimik für wütend, glücklich, traurig und überrascht und ob Augen und Mund geöffnet oder geschlossen sind zu erkennen. Die so gewonnen Daten werden anschließend, durch Morph Targets, auf das Gesicht des Spielcharakters übertragen. Morph Targets erlauben es ein 3D-Modell zu verformen, indem sie für bestimmte Vertices (die Eckpunkte der Dreiecksflächen, aus denen ein 3D-Modell zusammengesetzt ist) eine Verschiebung definieren, welche durch Multiplikation mit einem Skalar in ihrer Ausprägung variiert werden kann[23].

Nun soll zusätzlich auch das Sprachsignal genutzt werden, um auf das Gesicht eines Spielcharakter abgebildet werden. Wichtig ist hierbei, dass die Mundform mit dem gerade Gesprochenen korreliert. Hieraus ergeben sich die Anforderungen einer möglichst geringen Latenz und eine Einhaltung der zeitlichen Relationen. Für die Latenz hat die International Telecommunication Union 1998 eine Empfehlung veröffentlicht nach der eine Verzögerung des Bildsignals zum Audiosignal ab 45 ms wahrnehmbar ist und ab 90 ms als störend empfunden wird[24].

Die realistischsten Ergebnisse würde ein Verfahren erreichen bei dem Phoneme erkannt werden. Ein Ansatz hierfür wurde im Rahmen von Gamecast bereits von Herrn Thomas Puschmann in einer Diplomarbeit untersucht. Die in dieser Arbeit präsentierte Lösung nutzt die Windows Speech API um ein gesprochenes Wort zu erkennen und zerlegt dieses anschließend in die Phoneme, welche notwendig sind um es auszusprechen. Diese Methode liefert deutlich genauere Ergebnisse, als dies bei der direkten Erkennung von Phonemen der Fall wäre. Dies begründet sich darin, dass Spracherkennung stark kontextbasiert ist[25]. Dieser Ansatz weist allerdings eine Reihe von Problemen auf, die ihn für den Einsatz während des Spiels ungeeignet macht. Am schwerwiegendsten ist dabei, dass ein Wort erst vollständig ausgesprochen sein muss, bevor es erkannt werden kann. Dies führt dazu, dass die Latenz sich aus der Zeit die benötigt wird ein Wort auszusprechen und der für die Erkennung notwendigen Zeit zusammensetzt. Was bereits bei einsilbigen Wörtern zu Latenzen von mehr als 200 ms führt, welche den Grenzwert von 90 ms deutlich überschreiten. Weiterhin macht es die variable Latenz bei der Erkennung eines Wortes und die Ausgabe der Phoneme ohne zeitlichen Bezug auf ihre Position im Sprachsignal praktisch unmöglich Gesprochenes und Mundbewegung zu synchronisieren. Zusätzlich erfordert die Windows Speech API mindestens eine etwa zehnminütige Trainingsphase bei der der Nutzer einen vorgegebenen Text vorlesen muss, um die Stimme einer Person richtig zu erkennen. Darüber hinaus besitzen die in

[23] vgl. [22]
[24] vgl. [19]
[25] vgl. [24]

Urban Legend verwendeten Charaktermodelle keine Morph-Targets für Phoneme.

Es musste also ein alternativer Ansatz gefunden werden. Das Problem kann deutlich vereinfacht werden, indem auf die Nutzung von Phonemen verzichtet wird. Ein sprechender Charakter kann dann durch das Öffnen und Schließen des Mund animiert werden. Durch die SHORE-Engine kann zwar die Mundöffnung des Spielers ermittelt werden, allerdings ist diese beim Sprechen zu schwach ausgeprägt um richtig erkannt zu werden.

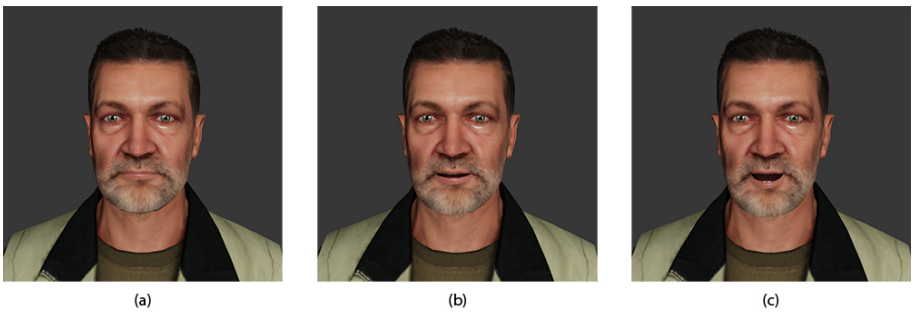

Abbildung 5.1: Verschiedene Ausprägungen des “Mund geöffnet”-Morph Targets beim Sprechen: a) stumm b) leises sprechen c) lautes sprechen

Daher wurde eine Lösung umgesetzt welche die Sprachdaten nutzt um den Grad der Mundöffnung zu bestimmen. Hierzu wurde eine Funktion implementiert welche von `AudioInput` und `AudioOutput` aufgerufen wird, nachdem ein neues Audio-Frame über das Mikrofon eingegangen bzw. ein eingegangenes Frame dekodiert wurde. Für dieses wird die durchschnittliche Lautstärke berechnet. Die DLL wurde um die Funktion `float getVolume(wchar_t username)` erweitert, um vom UDK auf die ermittelten Werte zugreifen zu können. Sie wird ebenfalls im Tick von `GC_PlayerController_VoiceChat` aufgerufen, welcher über alle für den Spieler sichtbaren Pawns iteriert und für diese die Lautstärkewerte abfragt und lokal anwendet.

6 Fazit

6.1 Ergebnis

In dieser Arbeit wurden für die Nutzung mit Videospielen entworfene VoIP-Lösungen verglichen. Von diesen wurde Mumble gewählt, um in Urban Legend integriert zu werden. Es wurde gezeigt wie Mumble in ein UDK-Spiel integriert werden kann. Zusätzlich wurde ein Konzept für die Automatisierung der Nutzerverwaltung auf dem Server entworfen und implementiert. Die eingebaute positionsabhängige Audio-Ausgabe wurde für Urban Legend angepasst. Außerdem wurde eine Lösung gefunden die Sprachdaten in Echtzeit für die Animation des Gesichtes zu nutzen. Die im Rahmen dieser Bachelorarbeit integrierten Funktionalitäten ermöglichen eine positionsabhängige und dem Spieler zuordenbare Kommunikation mittels VoIP. Das Ziel dieser Bachelorarbeit ist damit in technischer Hinsicht erreicht.

6.2 Ausblick

Für eine realistischere Abbildung der Mundbewegungen auf einen Spielcharakter, muss eine Möglichkeit gefunden die Mundform aus dem Tonsignal zu erlangen. Weiterhin sollten die gewonnen Daten in das Logging-System von Gamecast eingebunden werden. Hierfür wäre zu untersuchen ob, das Mitschneiden von Audiodaten oder die Speicherung einer durch Spracherkennung erstellten Transkription, welche anschließend für eine professionelle Nachvertonung genutzt werden kann, zu diesem Zweck besser geeignet ist. Hierfür ist außerdem eine Klärung der rechtlichen Situation in Bezug auf die Erstellung eines Mittschnitts erforderlich.

Literaturverzeichnis

[1] URL: <https://www.soe.com/soemote/guide/index.vm>, verfügbar am 13.02.2014

[2] URL: <http://www.gamecast-tv.com/index.php?id=171>, verfügbar am 13.02.2014

[3] URL: <http://www.zeit.de/digital/games/2013-05/urban-legend-game-test/komplettansicht>, verfügbar am 13.02.2014

[4] URL: <http://www.gamecast-tv.com/index.php?id=172&L=>, verfügbar am 13.02.2014

[5] URL: <http://udn.epicgames.com/Three/DLLBind.html>, verfügbar am 13.02.2014

[6] URL: <http://msdn.microsoft.com/en-us/library/windows/desktop/dd370802%28v=vs.85%29.aspx>, verfügbar am 13.02.2014

[7] URL: <http://msdn.microsoft.com/en-us/library/windows/desktop/dd370821%28v=vs.85%29.aspx>, verfügbar am 13.02.2014

[8] URL: <http://msdn.microsoft.com/en-us/library/windows/desktop/dd316773%28v=vs.85%29.aspx>, verfügbar am 13.02.2014

[9] URL: <https://github.com/mumble-voip/mumble/blob/master/LICENSE>, verfügbar am 13.02.2014

[10] URL: <https://github.com/mumble-voip/mumble/blob/master/LICENSE>, verfügbar am 13.02.2014

[11] URL: <http://sales.teamspeakusa.com/pricing.php>, verfügbar am 13.02.2014

[12] TeamSpeak 3 Client SDK Developer Manual, 2012, enthalten im TeamSpeak 3 SDK

[13] Lerch, Reinhard ; Sessler, Gerhard ; Wolf, Dietrich; Technische Akustik: Grundlagen und Anwendungen, 2009, Springer-Verlag

[14] Der Scrum Guide, Ken Schwaber, Jeff Sutherland, Juli 2013: <https://www.

scrum.org/Portals/0/Documents/ScrumGuides/2013/Scrum-Guide-DE.pdf>, verfügbar am 13.02.2014

[15] Valin, JM., Vos, K., and T. Terriberry, Definition of the Opus Audio Codec, RFC 6716, September 2012: <http://www.rfc-editor.org/rfc/rfc6716.txt>, verfügbar am 13.02.2014

[16] URL: <http://tools.ietf.org/html/draft-ietf-codec-results-00>, verfügbar am 13.02.2014

[17] URL: <http://www.speex.org/docs/manual/speex-manual/node4.html>, verfügbar am 13.02.2014

[18] URL: <http://git.xiph.org/?p=celt.git;a=blob;f=README;hb=HEAD>, verfügbar am 13.02.2014

[19] URL: <http://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1359-0-199802-S!!PDF-E.pdf>, verfügbar am 13.02.2014

[20] URL: <http://mumble.sourceforge.net/ACL_and_Groups>, verfügbar am 13.02.2014

[21] URL: <https://github.com/mumble-voip/mumble/blob/master/doc/mumble-protocol.pdf?raw=true>, verfügbar am 13.02.2014

[22] URL: <https://udn.epicgames.com/Three/MorphTargets.html>, verfügbar am 13.02.2014

[23] URL: <http://ffdshow-tryout.sourceforge.net/>, verfügbar am 13.02.2014

[24] Thomas Puschmann, Implementierung eines Spracherkennungssystems zur lippensynchronen Animation von Charakteren - 2011 - Mittweida, Hochschule Mittweida, Fakultät Elektro- und Informationstechnik, Unveröffentlichte Quelle

[25] Lazarus, Hans ; Sust, Charlotte ; Steckel, Rita ; Kulka, Marko ; Kurtz, Patrick; Akustische Grundlagen sprachlicher Kommunikation, 2009, Springer-Verlag

Glossar

Code excited linear prediction ist ein Verfahren zur Audiodatenkompression.

Constrained Energy Lapped Transform ist ein patent- und lizenzgebührenfreier Audio-Codec.

DirectSound eine API für die Ausgabe von Sound, welche in Microsofts DirectX enthalten ist.

HRTF ist eine Funktion mit der die Ausgabe über Stereo-Kopfhörer auf einen Punkt im Raum abgebildet werden kann.

Morph Target ist eine Verschiebung von Vertices, die es erlaubt ein 3D-Modell zu verformen.

Mumble ist eine freie, für die Nutzung mit Videospielen optimierte VoIP-Software.

Murmur ist die Server Anwendung von Mumble..

Opus ist ein patent- und lizenzgebührenfreier Audio-Codec.

Phonem ist eine nicht segmentierbare Lauteinheiten, welche einen Bedeutungsunterschied beim Verstehen von Sprache ausmacht vgl. [25] S. 81f.

SILK ist ein von Skype entwickelter Audio-Codec für die Kompression von Sprache.

Speex ist ein patent- und lizenzgebührenfreier Audio-Codec für die Kompression von Sprache.

TeamSpeak ist eine für die Nutzung mit Videospielen optimierte VoIP-Software.

UDK ist eine kostenlos verfügbare Version der Unreal Engine 3.

Ventrilo ist eine für die Nutzung mit Videospielen optimierte VoIP-Software.

VoIP ist die Übertragung von Sprachdaten über ein Internet Protocol Netzwerk.

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 13.02.2014