Institute of
System Security

Software-Entwicklungspraktikum

OfficeMania SEC0

Software-Entwicklungspraktikum (SEP)
Sommersemester 2021

Technischer Entwurf

Auftraggeber
Technische Universität Braunschweig
Institut für Systemsicherheit
Prof. Dr. Konrad Rieck
Mühlenpfordtstraße 23
38106 Braunschweig

Betreuer: Erwin Quiring, Alexander Warnecke, Jonas Möller

Auftragnehmer:

Name	E-Mail-Adresse
Michael Goslar	m.goslar@tu-bs.de
Paul Hagedorn	paul.hagedorn@tu-bs.de
Johan Kolms	j.kolms@tu-bs.de
Eva Nortmann	e.nortmann@tu-bs.de
Joel Schaub	j.schaub@tu-bs.de
Han Thang	h.thang@tu-bs.de
Fabian Vizi	f.vizi@tu-bs.de
Lukas Wieland	l.wieland@tu-bs.de
Giulia Woywod	g.woywod@tu-bs.de

Braunschweig, 30. Juni 2021

Bearbeiterübersicht

Kapitel	Autoren	Kommentare
1	Michael	...
1.1	Michael	...
2	Lukas	...
2.1	Lukas	...
3	Eva, Joel, Han	...
3.1	Eva, Joel, Han	...
3.2	Eva, Joel, Han	...
3.3	Eva, Joel, Han	...
4	Eva, Joel, Han	...
5	Fabian, Michael	...
5.1	Fabian, Michael	...
5.2	Fabian, Michael	...
5.3	Fabian, Michael	...
5.4	Fabian, Michael	...
6	Giulia	...
7	Paul	...
8	Eva, Joel, Han	...
9	Johan	...
9.1	Johan	...
9.2	Johan	...
9.3	Johan	...
10	...	...

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Unsere Webapplikation *„Officemania“* implementiert eine virtuelle Büroumgebung, welche im Rahmen des Softwareentwicklungspraktikums 2021 erstellt wird.

Der technische Entwurf besteht zu Teilen aus Kapiteln des Fachentwurfs und wird durch weitere Kapitel zum Entwurf des Projektes ergänzt.
Zu Anfang gibt es nochml ein grobes Gesammtbild über die Funktionen der Anwendung. In den folgenden Kapiteln werden die einzelnen Anforderungen des Projektes die Komponenten, sowie Schnittstellen zwsichen den Komponenten genauer betrachtet. Anhand von verschiedenen Diagrammen und erklärenden Texten werden so die Funktionweisen und deren Umsetzung aufgezeigt.
In den hinteren Kapiteln werden auch die verschieden Bibliotheken, welche benutzt wurden aufgezeigt und des Weiteren erklärt, warum in unserem Projekt keine Daten permanent gespeichert werden.
Zuletzt werden nochmal Änderungen zum Fachentwurf beschrieben und die Umsetzung der Kriterien aus dem Plichtenheft und deren Umsetzung genauer betrachtet.

Im nachfolgenden Text wird nocheinmal die Funktionalität des Projekts erklärt. Hierfür wurde in Kapitel 1 ein Statechart angefertigt, welches einen Überblick über die Funktionsweisen gibt. Im Unterkapitel 1.1 gibt es noch ein Aktivitätsdiagramm, dass einen komplizierten Zustand aus dem Statechart genauer erläutert.

Um unsere Software zu starten ruft man die Seite *„officemania.de“* mit einem Webaufruf auf. Als erstes wird überprüft, ob schon Daten im Local Storage vorhanden sind. War der Nutzer vorher schon mal auf der Seite wurden Daten im Local Storage gespeichert, welche den Namen und den Skin des Spielers sichern. Sind solche Daten im Local Storage vorhanden, werden die genannten Daten übernommen. Ist dies nicht der Fall, wird man aufgefordert einen Namen einzugeben.
Parallel wird im Hintergrund die Map berechnet und nach dem Setzten des Namens dann gezeichnet.
Als nächstes wird überprüft, ob sich Spieler in der Nähe befinden. Ist dass der Fall, wird sofort der Videochat mit den Spielern in der Nähe gestartet. Jetzt befindet sich der Spieler in einem

Zustand, in dem er bereit für Interaktionen mit dem User ist. Ton und Video ist standardmäßig angeschaltet, wobei die Bildschirmübertragung zu Beginn immer ausgeschaltet ist. Per Tastendruck können die Zustände von angeschaltet zu ausgeschaltet und andersherum gewechselt werden.
Im Hintergrund wird durchgehend getestet, ob Spieler in der Nähe sind und ob somit ein Videochat gestartet wird.
Ein Spieler kann sich jederzeit bewegen, mit Objekten interagieren, den Skin wechseln und seinen Namen ändern.
Möchte man das Programm beenden, kann man die Anwendung schließen, indem man zum Beispiel den Tab bzw. Browser schließt. Dadurch gelangt man im Statechart in den Endzustand.

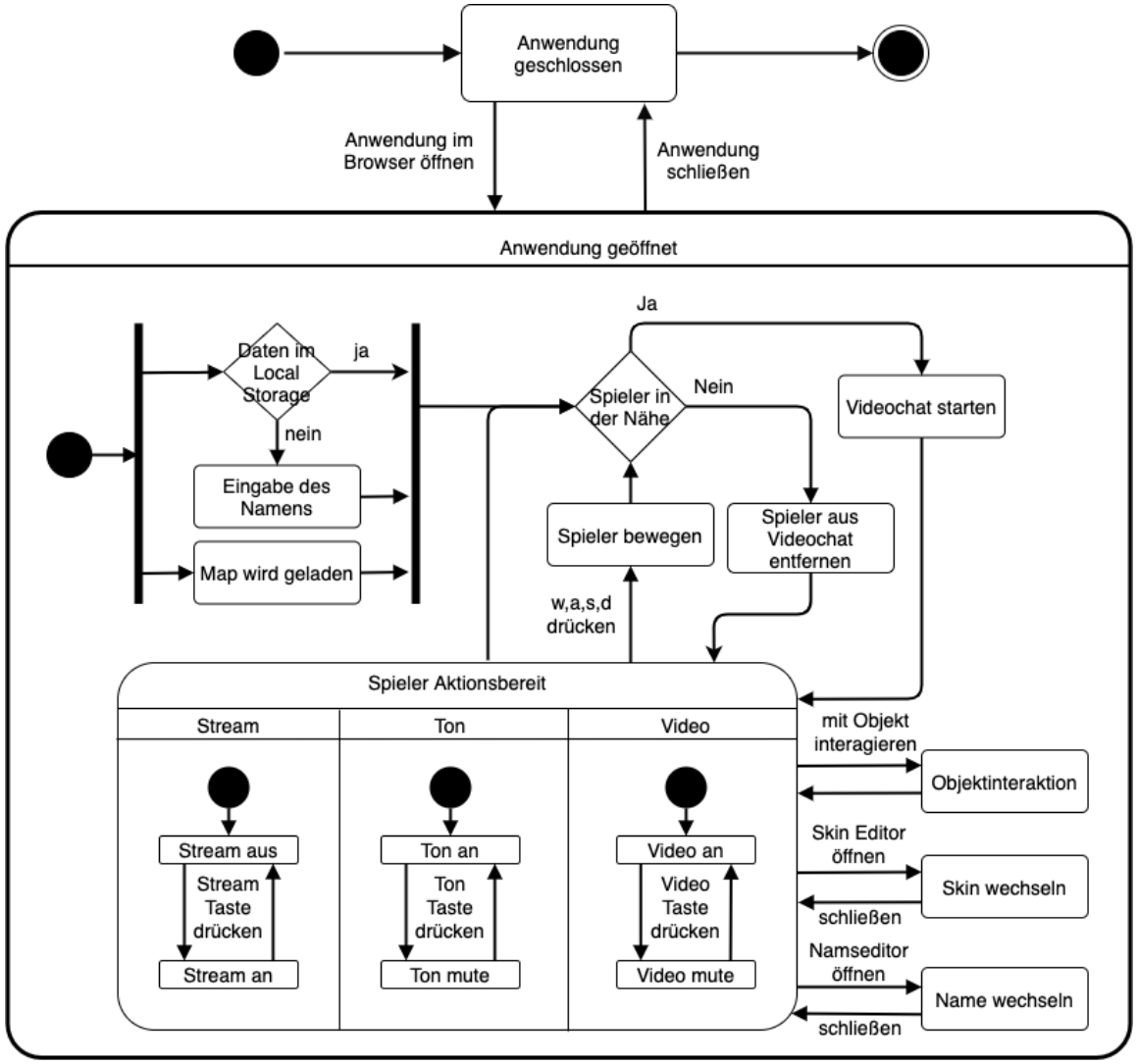

Abbildung 1.1: Systemablauf

1.1 Projektdetails

In diesem Abschnitt wird ein bestimmter Zustand aus dem Statechart genauer betrachtet und fachlich anhand eines Textes und dem zugehörigen Aktivitätsdiagramm erläutert.

1.1.1 Spielerbewegung

Eine Spielerbewegung wird durch das drücken einer der Tasten W, A, S oder D gestartet. Die vier Tasten stehen hierbei für die vier Richtungen oben, unten, rechts und links. In Abhängigkeit der Richtung wird nun ein ensprechendes Signal an den Server übergeben, welcher dadurch die neue Position und Richtung des Spielers erhält.
Der Client holt sich nun die neue Koordinate, um die Map entsprechend der neuen Spielerposition zu zeichnen.
Zuletzt wird der Charakter des Spielers neu gezeichnet, welcher immer in der Mitte des Bildschirm gezeichnet wird. Um die Blickrichtng des Spielers richtig zu zeichnen wird diese vom Server abgefragt.
Beim Übergang von der alten Position zur Neuen wird eine Animation gezeichnet, welche durch die Blickrichtung bestimmt wird.

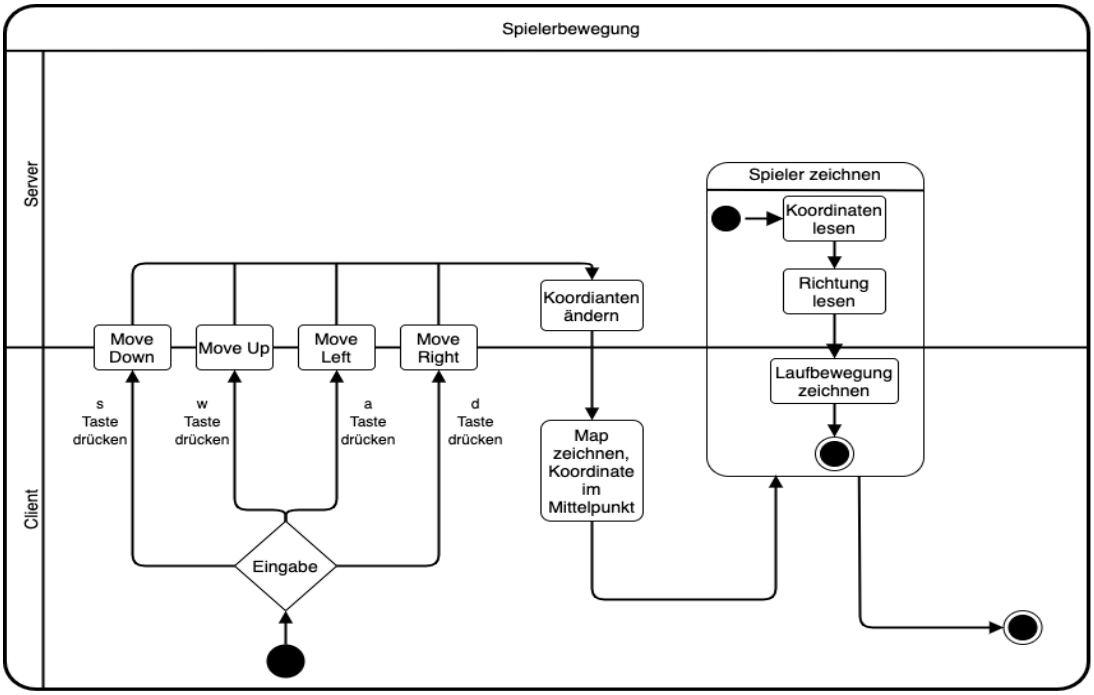

Abbildung 1.2: Spieler bewegen

2 Analyse der Produktfunktionen

Dieses Kapitel wird grundlegend aus dem Fachentwurf übernommen. Mögliche Änderungen der Analyse sind allerdings mit inbegriffen. Das Kapitel befasst sich mit der Analyse und graphischen Darstellung der Produktfunktionen aus dem Pflichtenheft. Hierbei werden Sequenzdiagramme als Graphiken verwendet. Diese sollen beteiligte Komponenten, sowie deren Interaktionen und Abhängigkeiten von relevanten Funktionalitäten darstellen.

2.1 Analyse von Funktionalität F10: Anwendungsaufruf

Die Funktion <F10>, die in Abbildung 2.1 mit einem Sequenzdiagramm beschrieben wird, beschreibt das Starten der Anwendung. Durch das Aufrufen des Links wird die Anwendung gestartet. Diese lässt die Karte von der GraphicsComponent zeichnen und den Spieler von der PhysicsComponent erstellen und zeichnen. Diese Zeichnungen werden im Browser dargestellt und so dem Nutzer indirekt wiedergegeben.

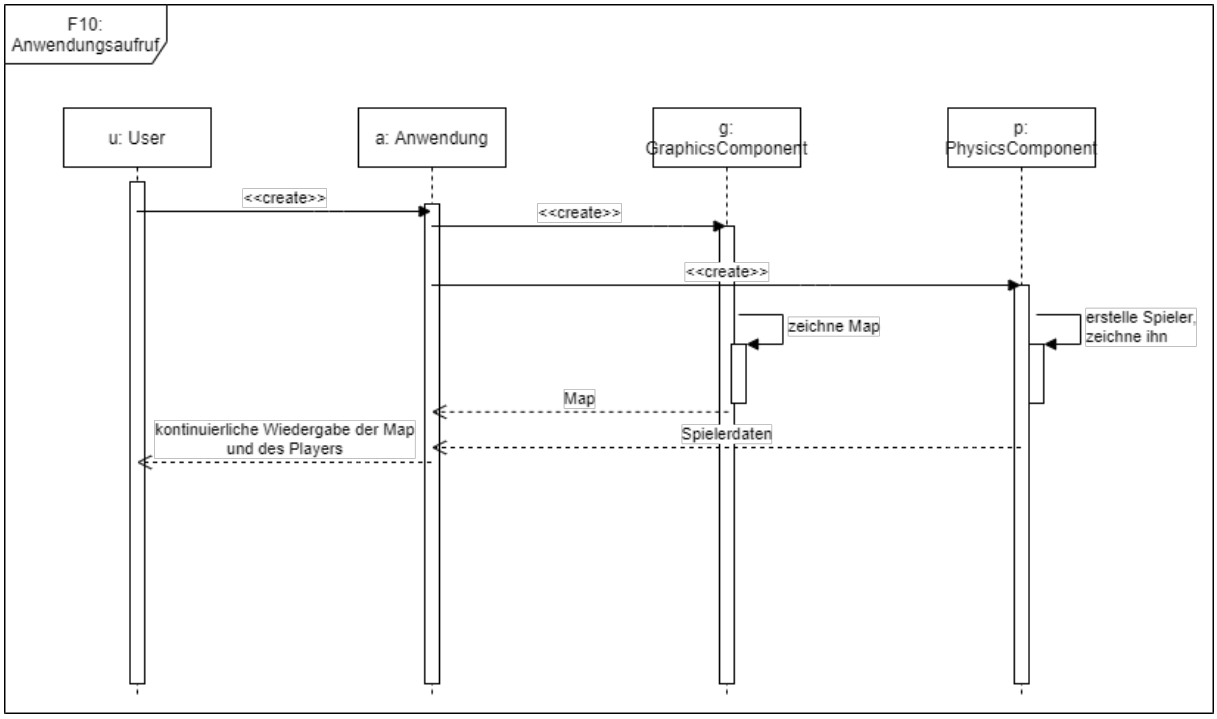

Abbildung 2.1: Anwendungsaufruf

2.2 Analyse von Funktionalität F20: Kartenzeichnung

Die Funktion <F20>, die in Abbildung 2.2 mit einem Sequenzdiagramm beschrieben wird, beschreibt die Kartenzeichnung. Dabei gibt der User der Anwendung (durch eine Bewegung von sich selbst oder jemand Anderen ausgelöst) den Impuls, die Map erneut zu zeichnen. Die Anwendung gibt diesen Impuls an die GraphicsComponent und die Physics Component weiter, welche die Map und die andern Spieler darstellen. Auch hier wird alles wieder im Browser dargestellt.

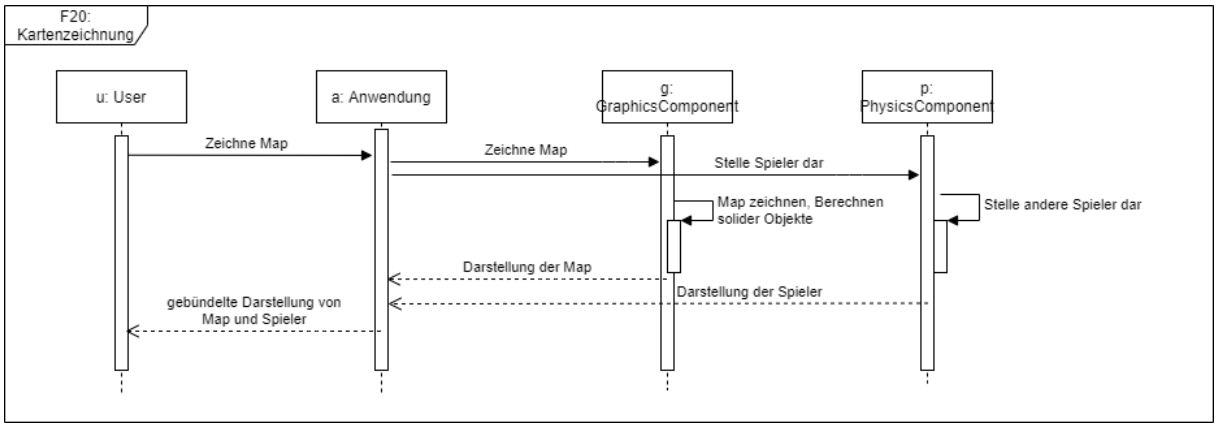

Abbildung 2.2: Kartenzeichnung

2.3 Analyse von Funktionalität F30: Spielerbewegung

Die Funktion <F30>, die in Abbildung 2.3 mit einem Sequenzdiagramm beschrieben wird, beschreibt die Spielerbewegug. Der Nutzer nutzt die Tasten W, A, S und D, um den Character zu bewegen. Die Anwendung nimmt dies wahr und leitet die Informationen über den Tastendruck an die PhysicsComponent weiter. Diese berechnet die neue Spielerposition, startet eine Laufanimation und zeichnet diese auf ein Canvas. Die berechnete Spielerpodition bekommt die Anwendung wieder und lässt damit die GraphicsComponent einen neuen Ausschnitt der Map zeichnen. Am Ende wird alles wieder im Browser dargestellt und so dem User wiedergegeben.

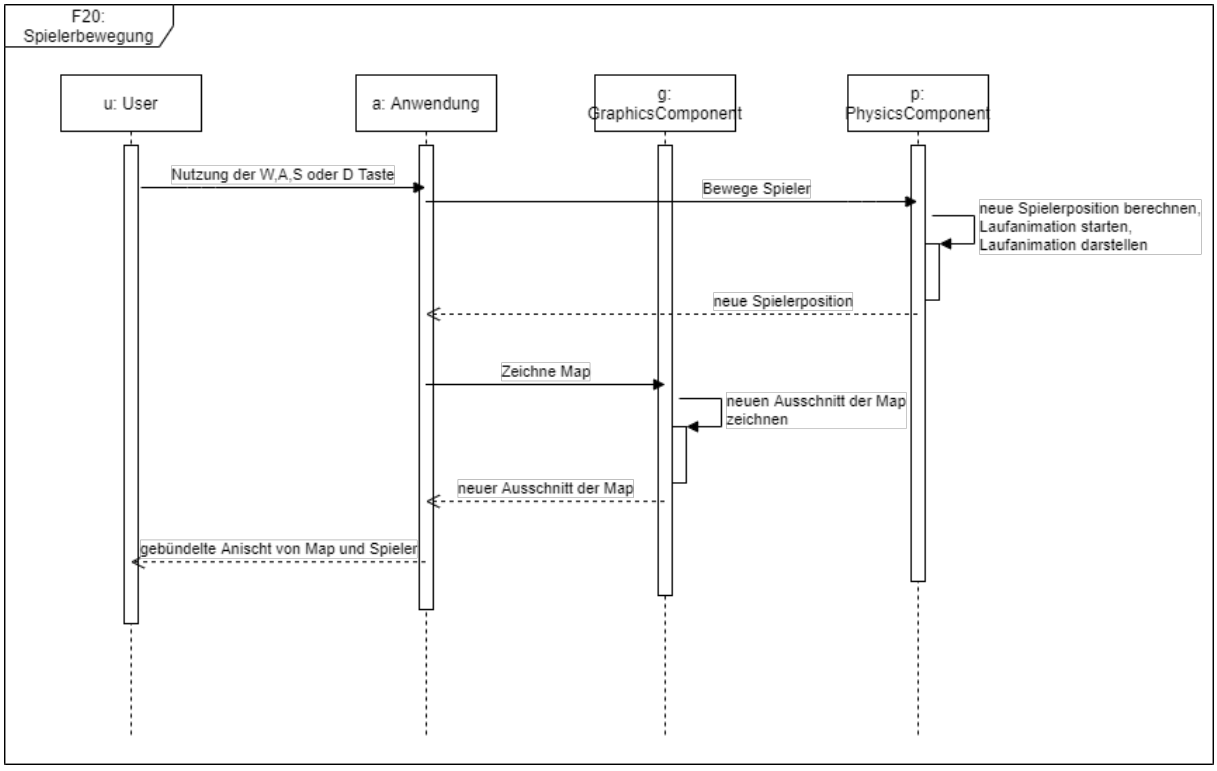

Abbildung 2.3: Spielerbewegung

2.4 Analyse von Funktionalität F40: Start Videokonferenz

Die Funktion <F40>, die in Abbildung 2.4 mit einem Sequenzdiagramm beschrieben wird, beschreibt das Starten einer Videokonferenz. Die Anwendung übermittelt kontinuierlich Audio und Video an den Jitsi-Server und dieser übermittelt kontinuierlich Audio und Video der anderen User. Der OfficeMania-Server stellt die Positionen der anderen Spieler zur Verfügung. Die Anwendung berechnet so den Abstand zu den anderen. Ist der Abstand klein genug, werden dem User Video und Audio der entsprechenden anderen User eingeblendet.

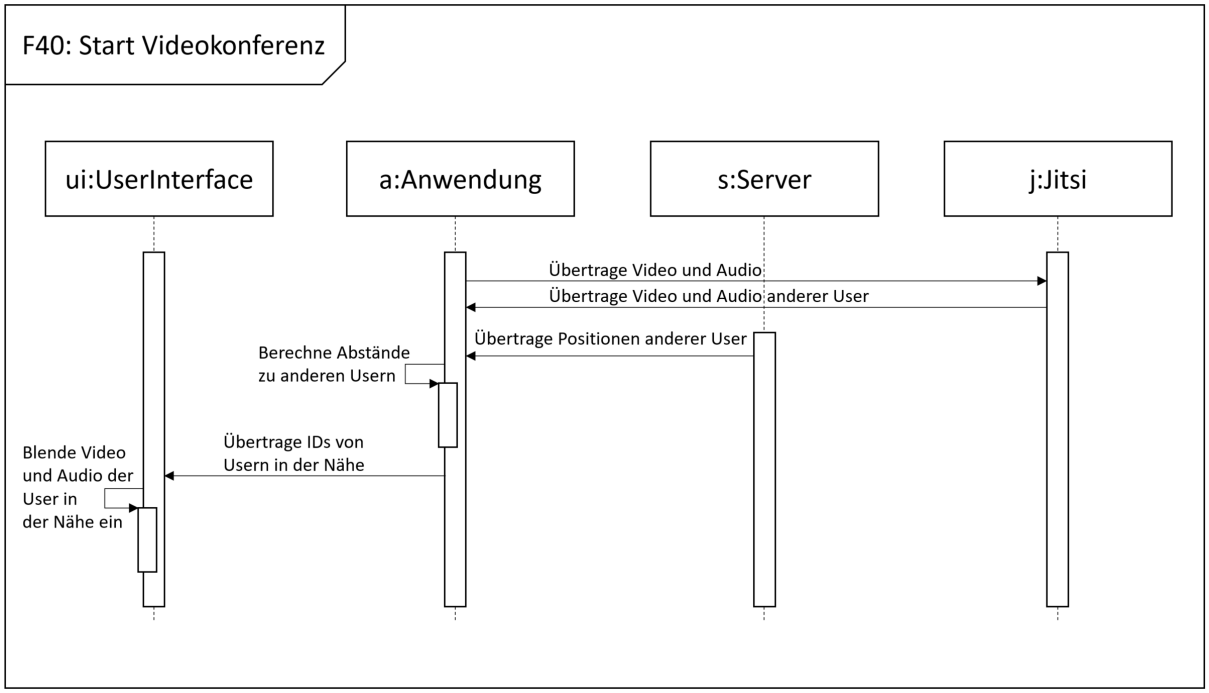

Abbildung 2.4: Start Videokonferenz

2.5 Analyse von Funktionalität F50: Stop Videokonferenz

Die Funktion <F50>, die in Abbildung 2.8 mit einem Sequenzdiagramm beschrieben wird, beschreibt das Beenden einer Videokonferenz. Die Anwendung übermittelt kontinuierlich Audio und Video an den Jitsi-Server und dieser übermittelt kontinuierlich Audio und Video der anderen User. Der OfficeMania-Server stellt die Positionen der anderen Spieler zur Verfügung. Die Anwendung berechnet so den Abstand zu den anderen. Wird der Abstand zu groß, werden Video und Audio der entsprechenden anderen User ausgeblendet.

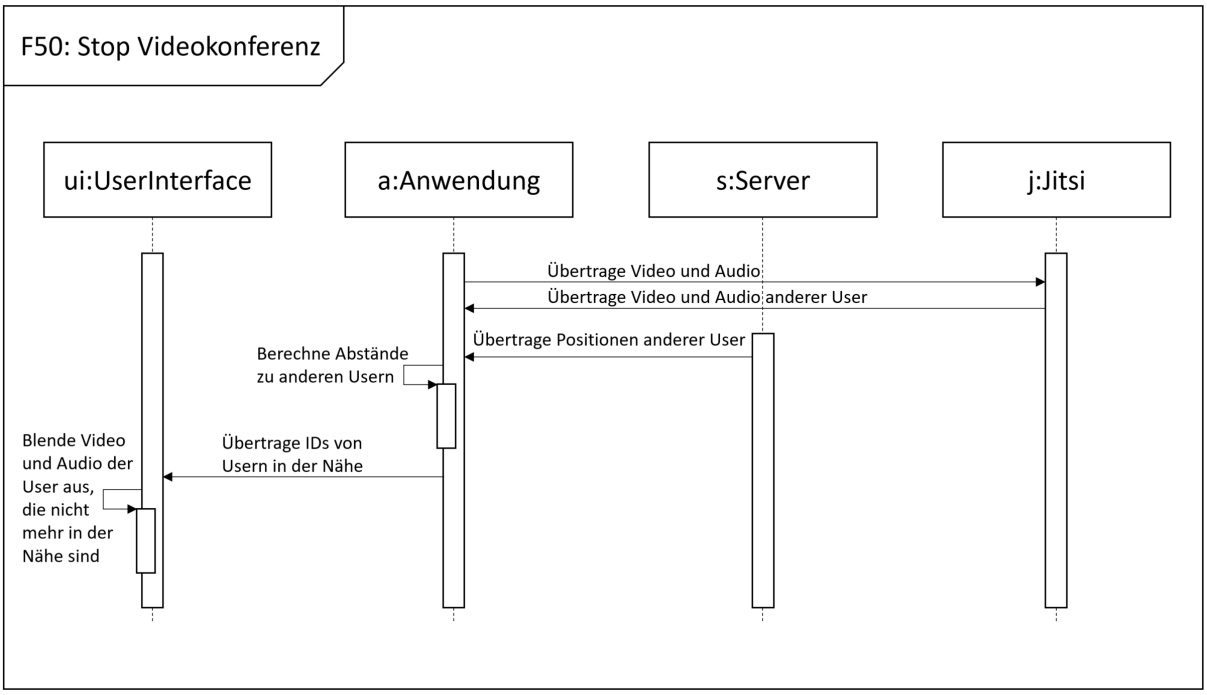

Abbildung 2.5: Stop Videokonferenz

2.6 Analyse von Funktionalität F60: Anzeige aktueller Spielerzahl

Die Funktion <F60>, die in Abbildung 2.6 mit einem Sequenzdiagramm beschrieben wird, beschreibt die Anzeige der aktuellen Spielerzahl. Wenn ein Nutzer die Anwendung startet, wird der Server durch die Anwendung von der Spielerkreation informiert. Anschließend speichert die Anwendung den Spieler in der Spielerliste. Bei Beendigung der Anwendung wird der Server wieder informiert und der Spieler wird aus der Liste gelöscht. Bei jedem Anwendungsstart und -ende wird die aktuelle Spieleranzahl anhand der eingetragenen Spieler ermittelt und ausgegeben.

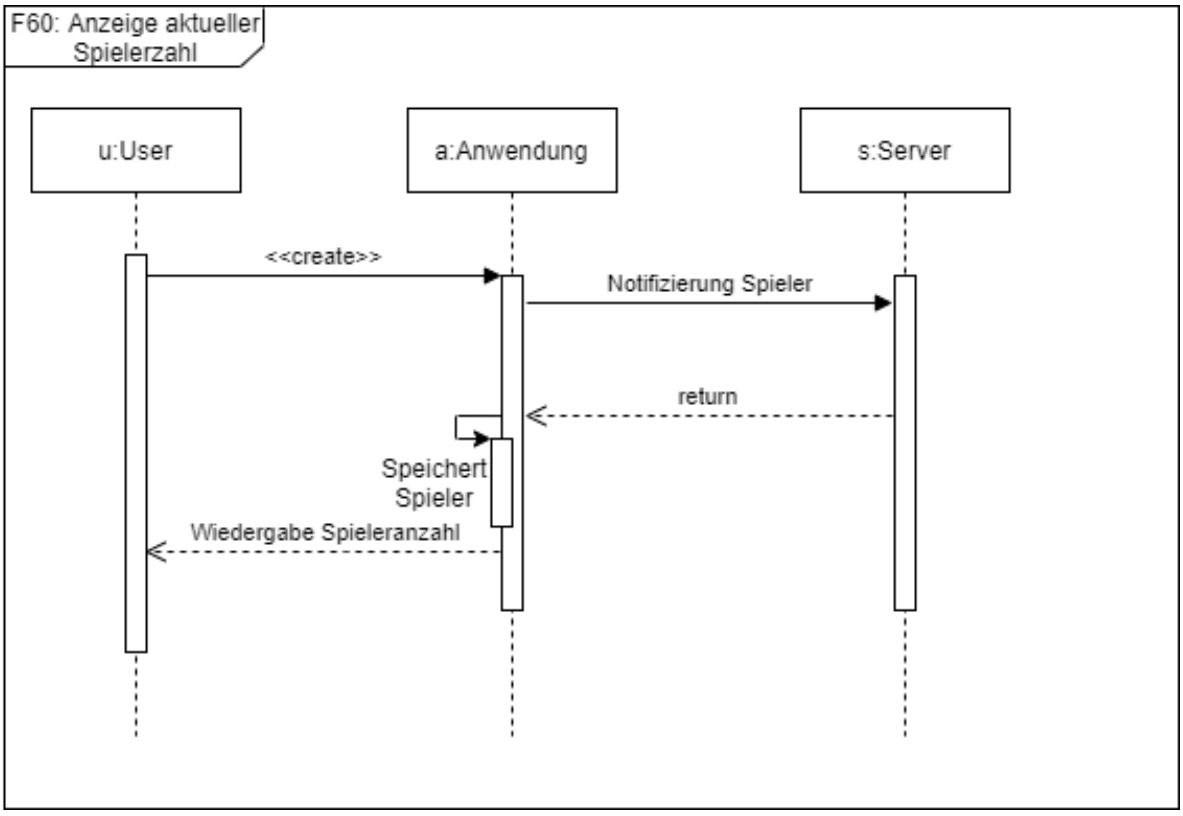

Abbildung 2.6: Anzeige aktueller Spielerzahl

2.7 Analyse von Funktionalität F70: Namensbestimmung

Die Funktion <F70>, die in Abbildung 2.7 mit einem Sequenzdiagramm beschrieben wird, beschreibt die Namensbestimmung. Wenn der Nutzer die Option zur Namensbestimmung ausführt, wird von der Anwendung ein Textfeld für den einzugebenden Namen geöffnet. Nach Eingabe und Bestätigung des gewünschten Namens wird dieser anschließend im Server für den Spieler gespeichert. Die PhysicsComponent der Anwendung hat Zugriff auf den neuen Namen. Der neue Name wird fortan vom UserInterface bei allen Nutzern wiedergegeben.

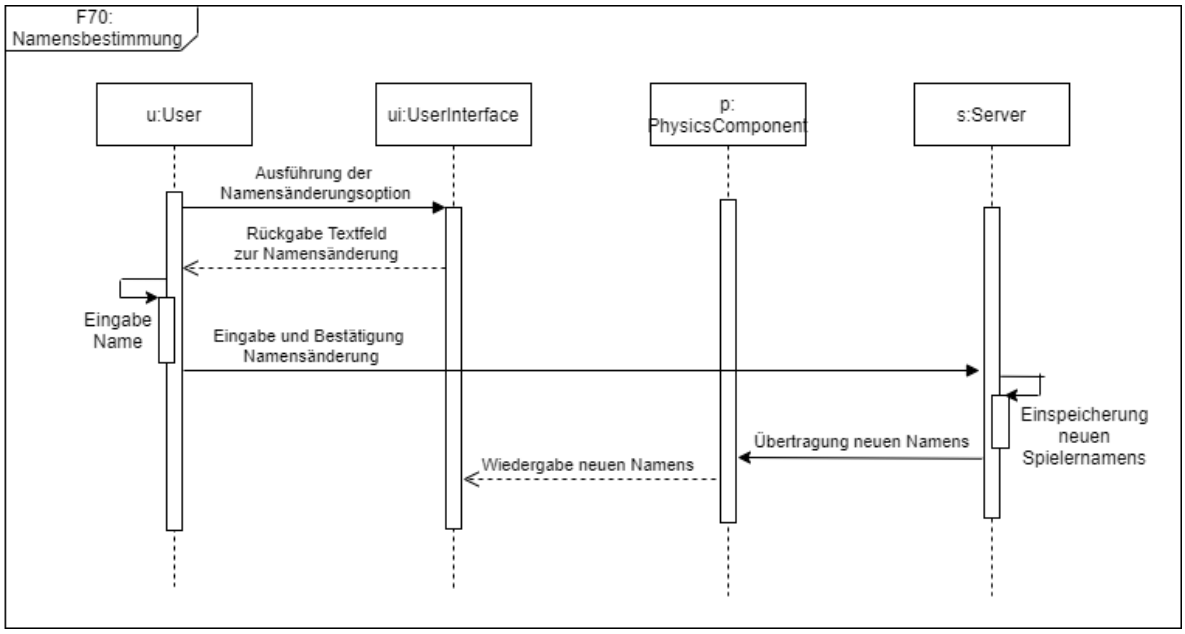

Abbildung 2.7: Namensbestimmung

2.8 Analyse von Funktionalität F80: Spielerinteraktion

Die Funktion <F80>, die in Abbildung 2.8 mit einem Sequenzdiagramm beschrieben wird, beschreibt die Spielerinteraktion. Wenn der Spieler sich an einem interagierbaren Objekt befindet und die Option zur Interaktion ausführt, startet die Spielerinteraktion. Hierbei führt die Anwendung bei dem relevanten Objekt im Server eine Statusänderung durch. Je nach Objekt und Interaktion, sowie der folgenden Serveranpassung, führt die PhysicsComponent die Interaktion durch, welche visuell vom UserInterface dargestellt wird.

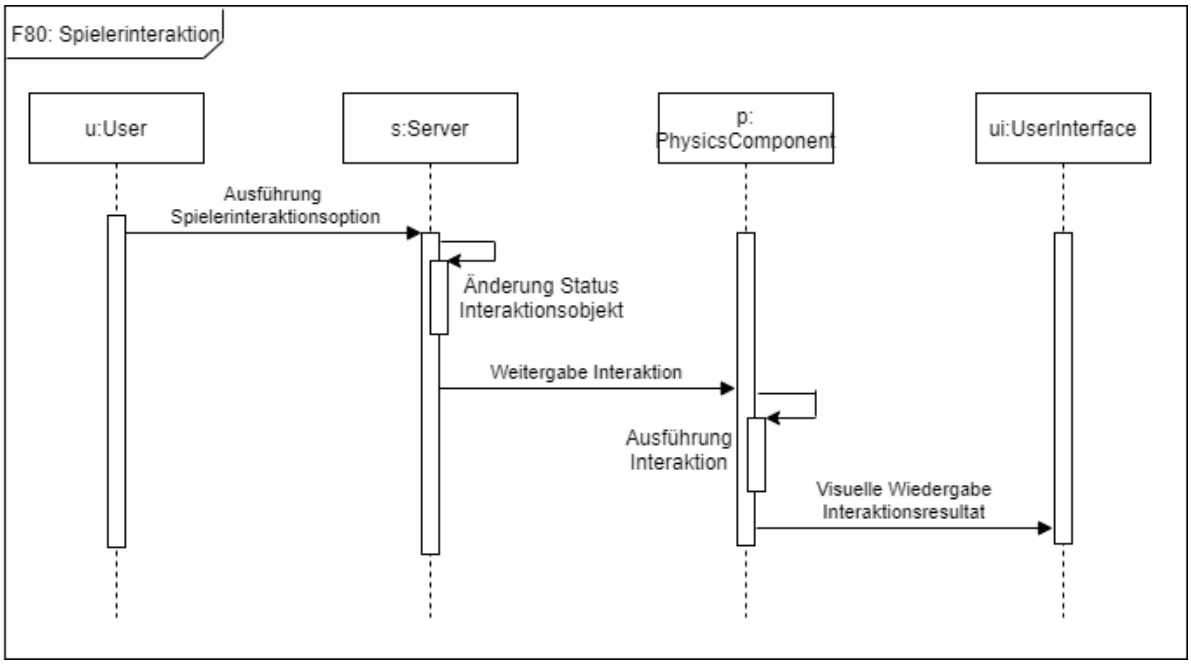

Abbildung 2.8: Spielerinteraktion

3 Resultierende Softwarearchitektur

In diesem Kapitel behandeln wir die Zusammenhänge der Komponenten unseres Projekts.

3.1 Komponentenspezifikation

In diesem Abschnitt betrachten wir die Komponenten einzeln und gehen näher auf ihre Funktionsweise und ihre Einsatzbereiche ein.

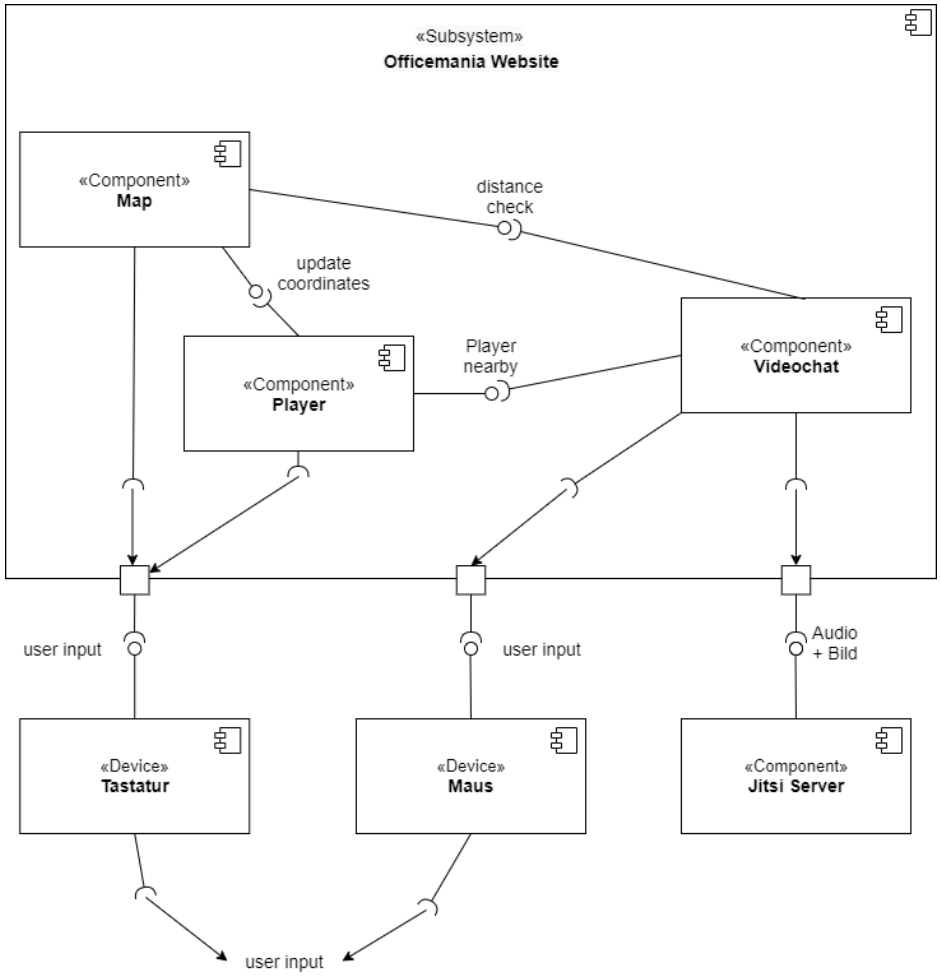

Abbildung 3.1: Komponentendiagramm.

Komponente $\langle C10 \rangle$: Map
Die Aufgabe der Map Komponente ist es, die Büroumgebung möglichst wahrheitsgemäß in 2D zu generieren. Ebenfalls stellt die Map Objekte bereit, wie z.B. Easter Eggs, mit denen der Player interagieren kann.

Komponente $\langle C20 \rangle$: Player
Die Player Komponente wird von dem User gesteuert und hat die Aufgabe den User in der virtuellen Büroumgebung als Figur darzustellen. Darüber hinaus werden die anderen Player im Verhältnis zu der eigenen Position abgebildet.
Außerdem ist eine weitere Aufgabe der Komponente, User-Interaktionen, wie z.B. Avatarskin-wechsel, auszuführen, wenn es von dem User per Tastatureingabe gefordert wird.

Komponente $\langle C30 \rangle$: Videochat
Die Videochat Komponente hat als Aufgabe Spieler, die in einer bestimmten Distanz zueinander sind mit einem Videochat zu verbinden, sodass diese darüber kommunizieren können.

Komponente $\langle C40 \rangle$: Jitsi Server
Der Jitsi-Server stellt dem Videochat die Bildschirmübertragung, Audio- und Videospur zur Verfügung, sodass ein virtueller Videochat-Raum erstellt werden kann.

3.2 Schnittstellenspezifikation

Schnittstelle $\langle I10 \rangle$: distance check

Operation	**Beschreibung**
nearbyPlayerCheck(players: PlayerRecord, ourPlayer: Player)	Die Funktion bekommt als Parameter eine Liste aller Player und sortiert diese in Spieler, die in der Nähe sind und Spieler, die nicht in der Nähe sind. Außerdem überprüft sie, ob die Spieler sich im gleichen Raum befinden.

Schnittstelle $\langle I20 \rangle$: update coordinates

Operation	**Beschreibung**
updatePosition(player: Player, room: Room)	In der Funktion werden die Daten des Players auf dem Client aktualisiert.
updateOwnPosition(player: Player, room: Room, collisionInfo: solidInfo[][])	Die Funktion überprüft, ob in der Richtung, in die der Spieler sich bewegen möchte eine Wand ist. Ist das der Fall, dreht der Spieler sich in die entsprechende Richtung und bleibt stehen. Ist das nicht der Fall, dreht der Spieler sich in die Richtung und geht in die Richtung.
convertMapData(mapdata:string, room: Room, canvas: HTMLCanvasElement)	Die Funktion berechnet von allen Layern der Map die kleinste und größte x- und y-Koordinate. Außerdem werden alle Layer und alle Tilesets, ebenso wie Canvas, Auflösung und Texturen zurückgegeben.

Schnittstelle $\langle I30 \rangle$: Player nearby

Operation	**Beschreibung**
nearbyPlayerCheck(players: PlayerRecord, ourPlayer: Player)	Die Funktion bekommt als Parameter eine Liste aller Player und sortiert diese in Spieler, die in der Nähe sind und Spieler, die nicht in der Nähe sind.

Schnittstelle $\langle I40 \rangle$: user input

Operation	Beschreibung
addEventListener(type: string, listener: EventListenerOrEventListenerObject, options?: boolean)	Die Funktion bekommt einen Input vom User. Das kann zum Beispiel ein Mausklick, eine Tastatureingabe oder ein Event vom Jitsi-Server sein. Daraufhin wird der Eventlistener aufgerufen.
toggleMuteByType(type: string): boolean	Die Funktion schaltet Audio, Video oder Stream, je nachdem welcher Typ angegeben wird, an bzw. aus.
toggleSharing(done: (enabled: boolean) => void)	Die Funktion startet bzw. beendet die Bildschirmfreigabe.
updateUsers(players: PlayerRecord)	Die Funktion listet alle Namen der Player auf, die aktuell online sind.

Schnittstelle $\langle I50 \rangle$: Audio + Bild

Operation	Beschreibung
onConnectionSuccess()	Die Funktion initialisiert eine Jitsi-Konferenz, der der User beitritt.
onDisconnected()	Die Funktion entfernt den User aus der Jitsi-Konferenz, wenn der User mit dem Server disconnected.

3.3 Protokolle für die Benutzung der Komponenten

Im folgenden Abschnitt werden die Komponenten aus Abschnitt 3.1 und deren Verwendung anhand von Protokoll-Statecharts näher betrachtet.
Dabei gehen wir auch darauf ein, inwiefern die Komponenten wiederverwendet werden können, wenn das sinnvoll erscheint.

3.3.1 Map

Die Map wird nach dem Programmstart komplett geladen und gezeichnet. Dabei ist jedoch nur ein kleiner Abschnitt für den Nutzer sichtbar.
Bewegt sich der Charakter des Nutzers auf dem Feld nun weiter so werden die neuen Koordinaten geladen und der Abschnitt verschoben. Dadurch behält der Nutzer die gleiche Bildschirmgröße und sieht einen Abschnitt der Map.
Des Weiteren wird die Map aktualisiert, wenn der Charakter mit Türen interagiert. Die Tür ist hierbei immer geschlossen. Wenn der Spieler nun nah genug an der Tür steht wird diese animiert, dass sie sich öffnet. Sobald der Spieler sich wieder von der Tür entfernt, so wird das Schließen der Tür animiert.

Die Map-Komponente kann für Drittanbieter wiederverwendet werden, um zum Beispiel ein „2D-Spiel“ zu programmieren. Das ist sinnvoller, als eine neue Map zu erstellen, da die Map durch weitere Texturen angepasst werden kann und die Grundfunktionen, wie dass die Wände solide sind, bereits funktionieren.
Es müsste also nur das Design angepasst werden, was weniger Aufwand ist, als eine ganz neue Map zu implementieren.

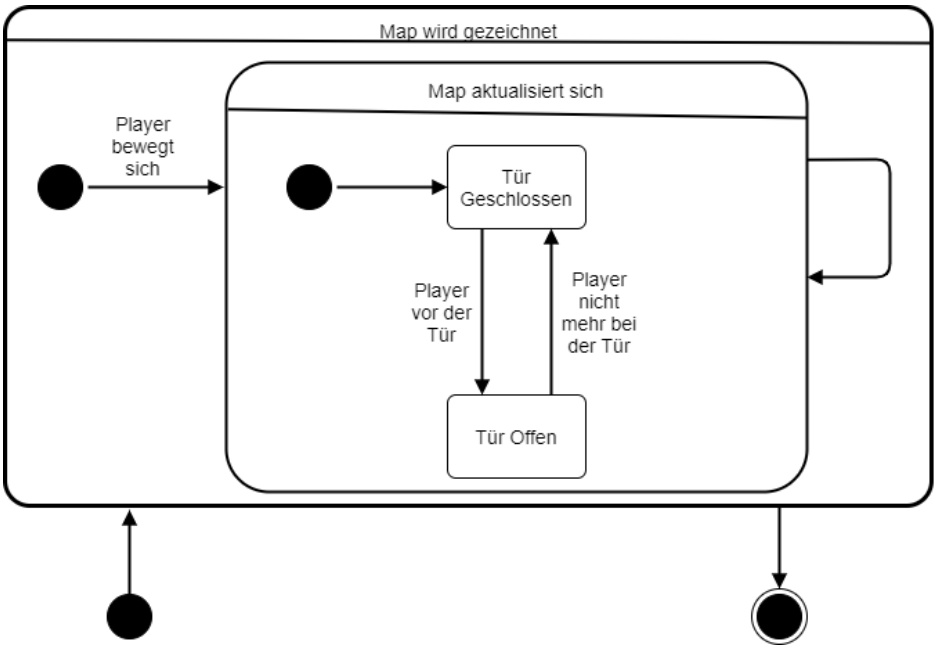

Abbildung 3.2: Statechart Map

3.3.2 Player

Die Player-Komponente zeichnet nach dem Starten des Programms zunächst den virtuellen Charakter des Nutzers, welcher still auf einer Stelle steht. Von dieser Ausgangsposition kann der Nutzer per Eingabe über die Tastatur die Interaktion des Charakters bestimmen.
Mit halten der Tasten W, A, S oder D kann der Nutzer den Charakter auf der Map enweder nach vorne, rechts, hinten oder links bewegen. Lässt er die Tasten wieder los so befindet sich der Charakter wieder im Stillstand.
Mit der Taste C kann ein Skin-Wechsel-Menü aufgerufen werden, in dem der Nutzer seinen Skin ändern und dann bestätigen kann. Des Weiteren kann jederzeit durch Betätigen der Taste R der Spielername geändert werden.
Ebenfalls gibt es die Möglichkeit, zu Objekten zu gehen und mit diesen zu interargieren.

Die Player-Komponente kann, ebenso wie die Map, für andere „2D-Spiele“ wiederverwendet werden, da die Figur animiert ist und in alle Richtungen laufen kann.
Dies kann nicht nur in einer Büroumgebung verwendet werden, sondern auch in beliebigen anderen Umgebungen.
Außerdem können für den Charakter weitere Charkterdesigns von Drittanbietern geladen werden. Dadurch wird es möglich sein, dass der Charakter ein beliebiges neues Aussehen hat, sodass er nicht nur in eine Bürowelt passt, sondern auch für andere Projekte geeignet ist, wo es Charakter gibt, die sich bewegen können.

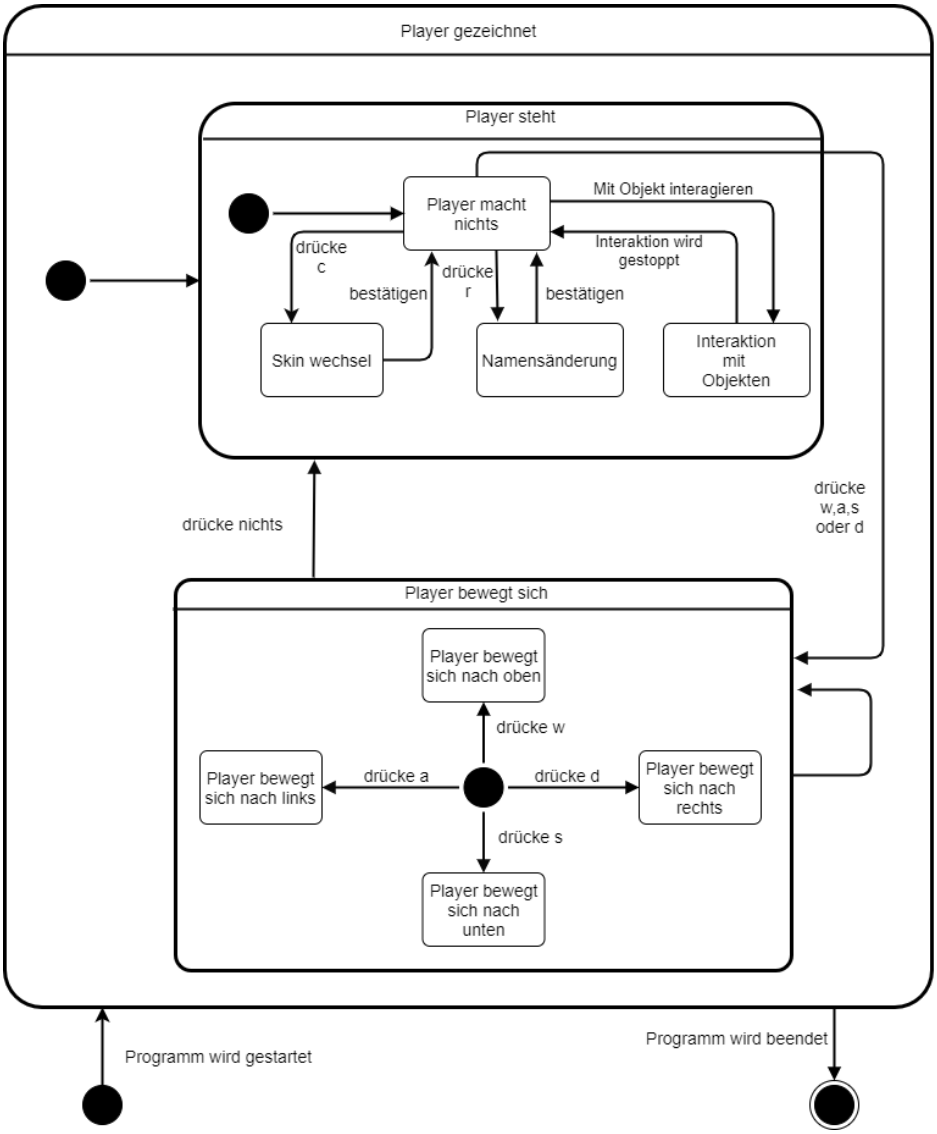

Abbildung 3.3: Statechart Player

3.3.3 Videochat

Die Videochat-Komponente wird nach dem Starten des Jitsi-Server für das Programm bereit gestellt. Hierbei sind am Start immer Audio und Video des Nutzers an und der Stream aus. Durch das Betätigen des Audio-, Video- oder Stream-Buttons mit der Maus kann der Nutzer variieren welche der drei Elemente er an haben möchte oder sogar ganz aus haben möchte. Hierbei ist zu beachten, dass der Nutzer nur das Video an haben kann, wenn der Stream aus ist und andersherum, der Stream kann nur an sein, wenn das Video aus ist. Des Weiteren besitzt die Komponente eine Nähe-Erkennung, welche andere virtuelle Charaktere in dem Umfeld des Nutzer-Charakters sucht. Befinden sich nun andere Charaktere in Reichweite, so werden die Videochats der Nutzer automatisch verbunden. Bewegen sich die Charaktere der Nutzer nun

voneinander weg und befinden sich dann außerhalb der Reichweite, so wird die Verbindung des Videochats zwischen den Nutzern getrennt. Der Zugriff auf Mute, Video und Stream Button ist hierbei immer gewährleistet. Es ist somit egal, ob die Nutzer miteinander verbunden sind oder nicht.

Allgemein kann man die Videochat-Komponente in jeglichen Projekten wiederverwenden, wo das Aufrufen von Audio, Video und Stream gefordert ist. Ebenfalls kann man je nach Belieben auch die Nähe-Erkennunng der Komponente verwenden. Diese Komponente kann der Drittanbieter dann je nach Wunsch weiterentwickeln. Sei es mit einem Chatfenster zum schreiben oder anderen Ideen.

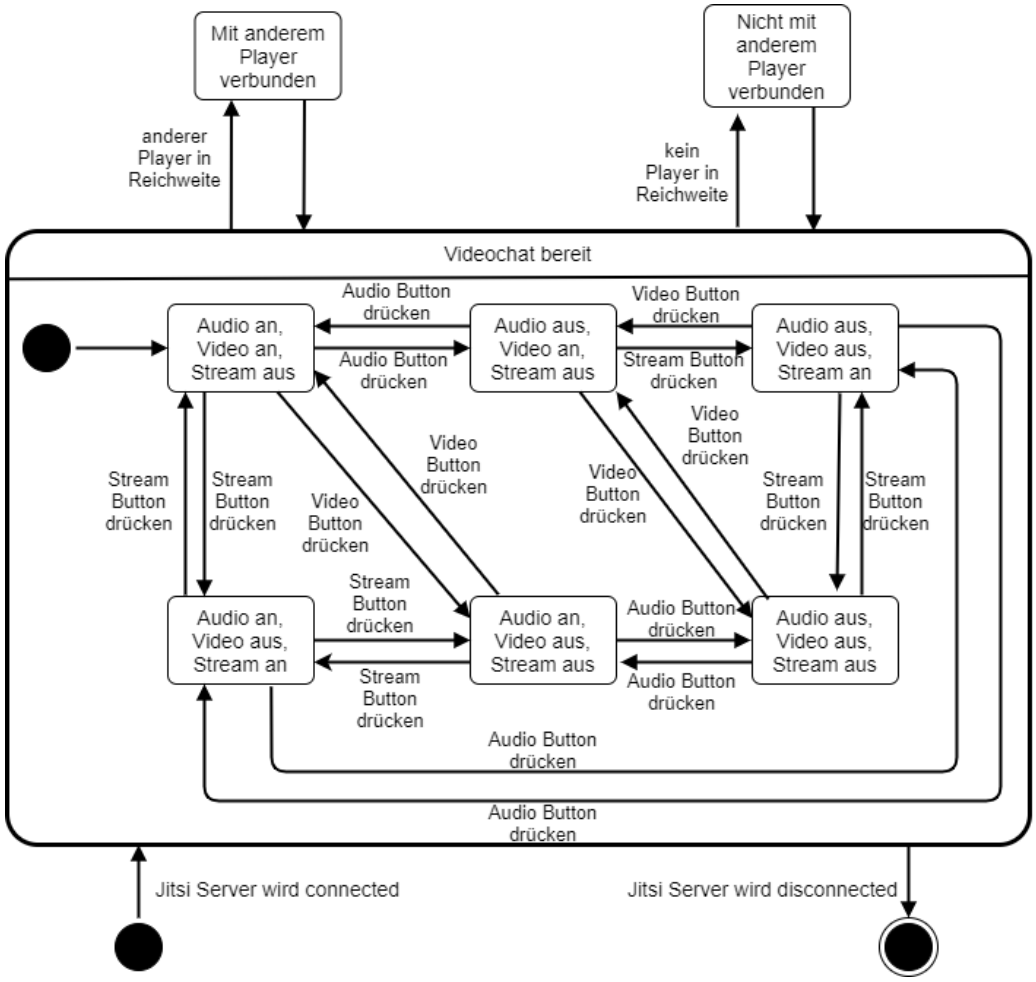

Abbildung 3.4: Statechart Video

3.3.4 Jitsi Server

Die genaue Funktionsweise des Jitsi-Servers ist eine Black-Box, da wir ihn nicht programmiert haben, sondern ihn nur für unseren Videochat verwenden.
Dabei stellt der Jitsi-Server die Videochat-Funktion bereit. Das bedeutet, er ermöglicht es uns, Audio und Video zu übertragen und über den Server live mit den anderen im Videochat zu teilen, sodass man sich sehen und hören kann.

4 Verteilungsentwurf

In diesem Kapitel behandeln wir die Anwendungsverteilung von OfficeMania.
OfficeMania kann mit einem PC in einem Browser, wie z.B Chrome, mit der Adresse www.officemania.de aufgerufen werden. Dies erfordert jedoch einen PC mit konstanter Internetverbindung. Hierbei hat der Nutzer im Optimalfall auch ein funktionierendes Mikrofon, einen funktionierden Lautsprecher und eine funktionsfähige Webcam. Diese Elemente werden benötigt um den einwandfreien Videochat zu gewährleisten.
Die Web-Anwendung OfficeMania beinhaltet grundlegend das „index.html“, das „main-bundle.js“ und das „vendor-bundle.js“. Das „index.html“ ist dabei die Haupt-HTML-Seite, auf der alles dargestellt wird. Das „main-bundle.js“ bezieht hierbei Daten vom Jitsi-Server, welcher hierbei für die Audio und Video Übertragung zuständig ist und welcher dafür sorgt, dass der Nutzer live mit anderen Nutzern im Videochat kommunizieren kann.
Das „vendor-bundle.js“ erhält über TCP/IP, Daten vom Colyseus Framework Server, um die Funktion des Multiplayers über Internet zu ermöglichen.
Ebenfalls bezieht die Web-Anwendung Daten vom „JQuery“, welches für die einfache Anwendung des JavaScript auf die Webseite zuständig ist.

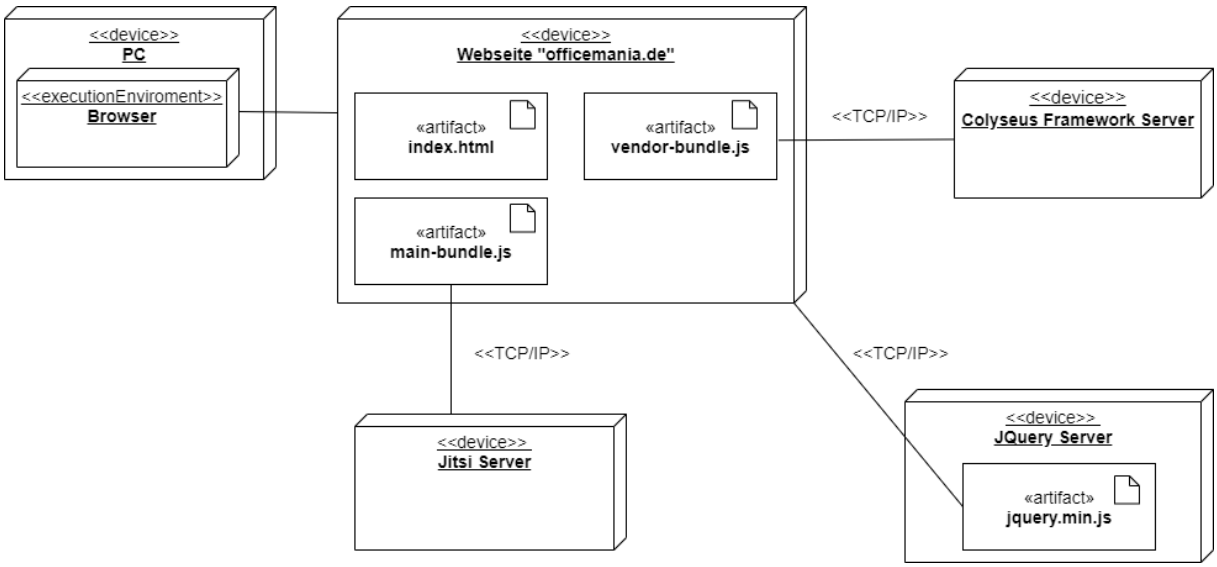

Abbildung 4.1: Verteilungsdiagramm

5 Implementierungsentwurf

Die Abbildung 5.1 zeigt das Klassendiagramm der Komponente Map.

5.1 Implementierung von Komponente $\langle C10 \rangle$: Map:

Die Komponente Map betsteht aus mehreren Klassen und Funktionen zum lesen und zeichnen der Map ausgehend von einer JSON-Datei. Die Klassen in der Komponente dienen zum abspeichern der Daten aus der JSON-Datei. Im wesentlichen enthält die Klasse MapInfo alle wichtigen Infos über die Map. Layer, Chunk und Tileset dienen dabei als Hilfsklassen zur besseren Übersichtlichkeit. SaveArray wird beim füllen von MapInfo gebraucht und SolidInfo enthält daten über die Kollision und soll später auch Infos über Raumstrukturen enthalten. Funktionen und Klassen der Map-Komponente werden in der Main Methode benutzt, Klassen wie SolidInfo zusätlich auch in der Playerkomponente und Videokomponente.

5.1.1 Paket-/Klassendiagramm

Die Abbildung 5.1 zeigt das Klassendiagramm der Komponente Map.

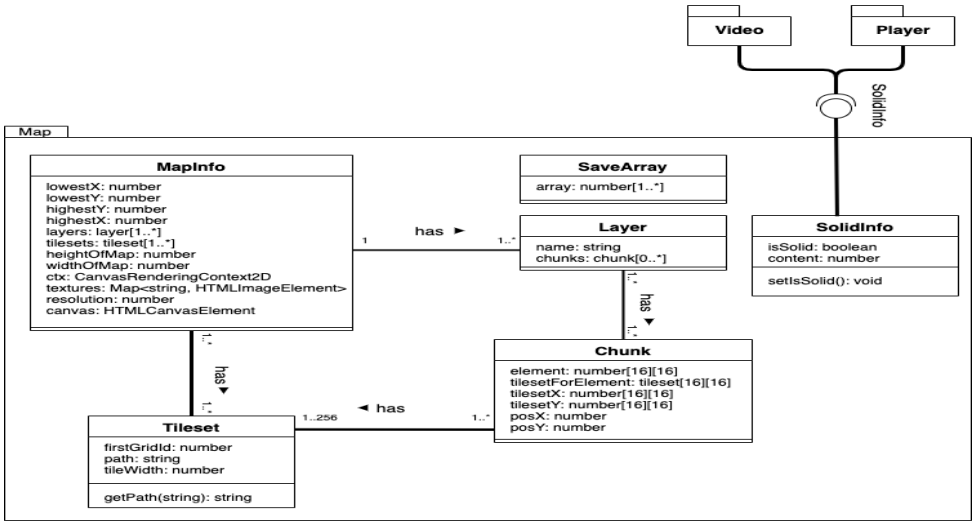

Abbildung 5.1: Packet-/Klassendiagramm für die Map-Komponente $\langle C10 \rangle$

5.1.2 Erläuterung

Hier werden Aufgaben, Attribute, Operationen und Kommunikationspartner der Klasse $\langle C10 \rangle$ veranschaulicht.

Tileset$\langle CL10 \rangle$

Aufgabe

Verwaltung der Infos über die Texturedateien.

Attribute

firstGridId: ID der Texture in der linken oberen Ecke der Texturedatei.

path: Dateipfad der Texturedatei.

tileWidth: Breite einer Texture in Pixeln.

Operationen

getPath(string): Durchsucht Abgespeicherte Pfade nach dem gegebenen String und gibt dann den gefundenen Pfad zurück. Wird im Constructor benuntzt, um das Attribut path zu füllen.

Kommunikationspartner

Keine

Chunk$\langle CL20 \rangle$

Aufgabe

Verwaltung der Texturen in einem 16x16 Feld.

Attribute

element: Speichert Nummern der Texturen ab, welche später dann in die entsprechenden Texturen übersetzt wird.

tilesetForElement: Speichert die entsprechende Texturedatei zu jeder Nummer aus element.

tilesetX: Speichert einen normierten Wert für die x-Koordinate zum zeichenen der Map ab für alle Elemente aus element.

tilesetY: Speichert einen normierten Wert für die y-Koordinate zum zeichenen der Map ab für alle Elemente aus element.

posX: Die x-Koordiante von element[0][0].

posY: Die y-Koordiante von element[0][0].

Operationen

Keine

Kommunikationspartner

Keine

Layer⟨$CL30$⟩

Aufgabe

Verwaltung der einzelnen Ebenen der Map.

Attribute

name: Name der Ebene.

chunks: Enthält alle Chunks die Felder aus der Ebene enthalten.

Operationen

Keine

Kommunikationspartner

Keine

SaveArray⟨$CL40$⟩

Aufgabe

Zwischenspeicherung eines Arrays.

Attribute

array: Speichert einen Numberarray.

Operationen

Keine

Kommunikationspartner

Keine

MapInfo⟨$CL50$⟩

Aufgabe

Verwaltung der Map.

Attribute

lowestX: Niedrigste x-Koordinate.

lowestY: Niedrigste y-Koordinate.

highestX: Größte x-Koordinate.

highestY: Größte y-Koordinate.

layers: Enthält alle Ebenen der Map.

tilesets: Enthält alle Tilesets welche in der Map benutzt werden.

heightOfMap: Höhe der Map.

widthOfMap: Breite der Map.

ctx: CanvasElement auf dem gezeichnet wird.

textures: Enthält ein Key-Value-Paar von Strings und den dazugehörigen Texturedateien.

resolution: Auflösung der einzelnen Felder.

canvas: Canvas der Map.

Operationen

Keine

Kommunikationspartner

Player

SolidInfo$\langle CL60 \rangle$

Aufgabe

Verwaltung der Infos über Kollision und der Raumstruktur.

Attribute

isSolid: Ist wahr, wenn die Koordinate Kollisonsfähig sein soll.

content: Nummer zum identifizieren von Räumen. Gleiche Nummern gruppieren Räume.

Operationen

setIsSolid(): setzt isSolid auf wahr.

Kommunikationspartner

Keine

5.2 Implementierung von Komponente $\langle C20 \rangle$: Player:

Die Komponente Player besteht nur aus einem Interface, Player. Dieses Interface wird benutzt, um Player-Objekte direkt durch den Zugriff auf deren Attribute zu manipulieren. Dies wird durch die Anwendung von Eventlistenern garantiert, welche auf User-Input, aber auch auf bestimmte Aufrufe aus anderen Klassen warten.
Das Player-Interface wird sowohl von der Map-, als auch von der Videochatkomponente benutzt.

5.2.1 Paket-/Klassendiagramm

Die Abbildung 5.2 zeigt das Klassendiagramm der Komponente Player.

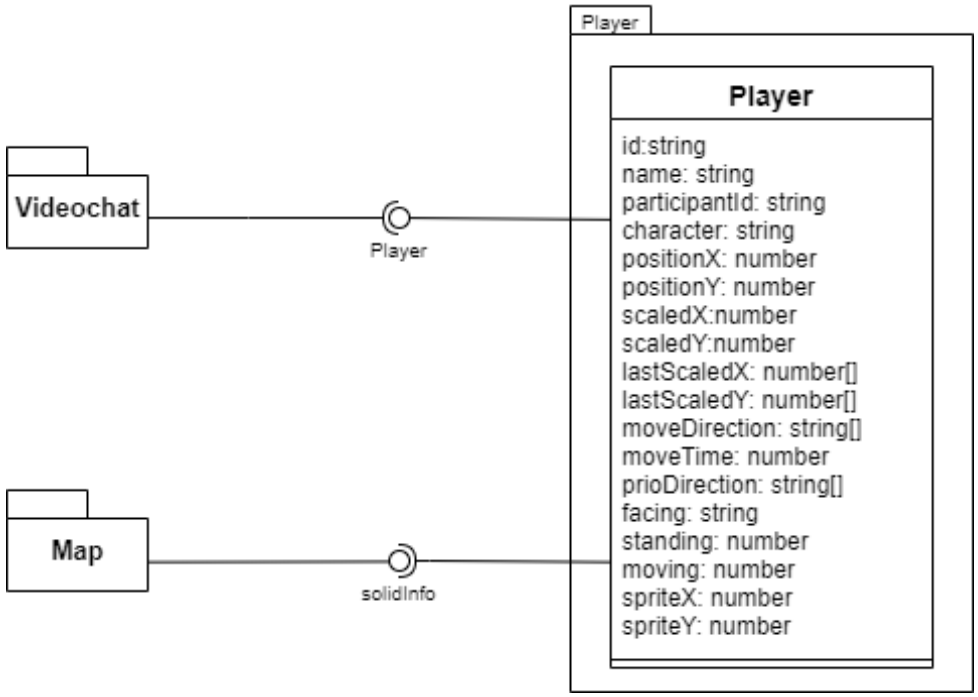

Abbildung 5.2: Packet-/Klassendiagramm für die Player-Komponente $\langle C20 \rangle$

5.2.2 Erläuterung

Hier werden Aufgaben, Attribute, Operationen und Kommunikationspartner der Klasse $\langle CL10 \rangle$ veranschaulicht.
Player$\langle CL10 \rangle$

Aufgabe

Speichern des Playerstadiums

Attribute

id: string: ID des Players in dem Raum
name: string: Name des Players
participantId: string: ID des Players in der Jitsi Konferenz
character: string: Name der Textur des Players
positionX: number: X-Koordinate des Players auf der Map
positionY: number: Y-Koordinate des Players auf der Map
scaledX: number: den Schrittgrößen nach Skalierte X-Koordinate des Players auf der Map
scaledY: number: den Schrittgrößen nach Skalierte Y-Koordinate des Players auf der Map
lastScaledX: number[]: Speicher der letzten 5 X-Koordinaten des Players
lastScaledY: number[]: Speicher der letzten 5 Y-Koordinaten des Players
moveDirection: string: jetztige Bewegungsrichtung des Players (auch keine möglich)
moveTime: number: Dauer des jetztigen Bewegungsinputs
prioDirection: string[]: letzte Bewegungsrichtungen
facing: string: zugewandte Richtung
standing: number: Zeit, seit dem der Player steht
moving: number: Zeit, set dem der Player sich bewegt
spriteX: number: X-Koordinate der Playertextur
spriteY: number: Y-Koordinate der Playertextur

Operationen

keine

Kommunikationspartner

Map, Videochat

5.3 Implementierung von Komponente $\langle C30 \rangle$: Videochat:

Die Komponente $\langle C30 \rangle$ Videochat ist für die Verwaltung der verschidenen Video, Audio und Textelemente verantwortlich. Sie kümmert sich um das Stummschalten von den verschiedenen Audio- und Videotracks, welches automatisch und auch bei Userinput geschieht.
Videochat besteht aus 3 Klassen, dem User, dem VideoContainer und dem SelfUser, der von dem User erbt. Jedes Objekt der Klasse User benutzt einen VideoContainer. Der SelfUser ist ein Spezialfall des Users, es ist die Userinstanz des Clients.
Die Komponente Videochat implementiert und arbeitet mit Funktionen aus der Jitsi-Bibliothek, um den Video- und Audiochat zu ermöglichen.

Außerdem wird in die Klassen Player und MapInfo aus den Komponenten Player und Map zugegriffen, um die nähebasierte Funktion und Konferenzräume zu ermöglichen, sowie das Verbinden durch solide Wände zu verhindern.

5.3.1 Paket-/Klassendiagramm

Die Abbildung 5.3 zeigt das Klassendiagramm der Komponente Videochat.

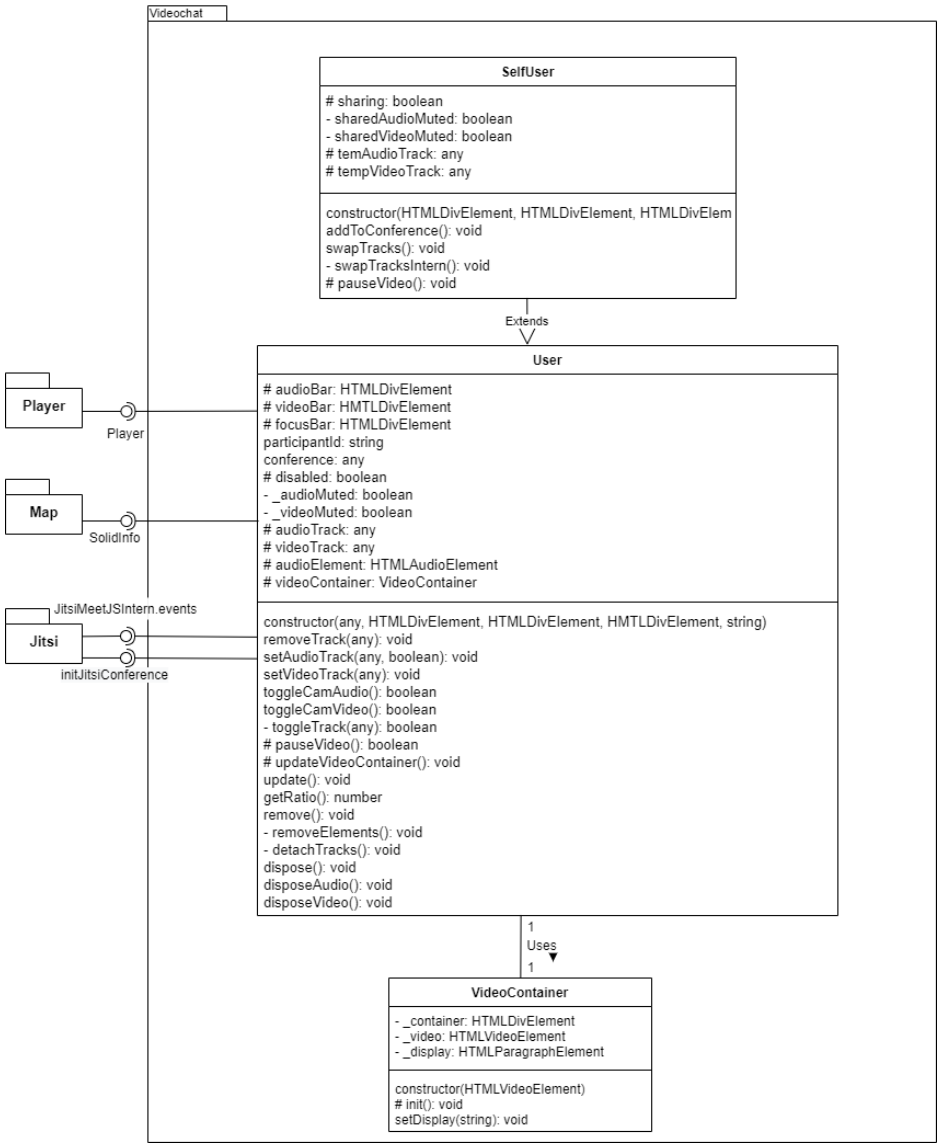

Abbildung 5.3: Packet-/Klassendiagramm für die Videochat-Komponente $\langle C30 \rangle$

5.3.2 Erläuterung

Hier werden Aufgaben, Attribute, Operationen und Kommunikationspartner der Klassen $\langle CL10 \rangle$, $\langle CL20 \rangle$ und $\langle CL30 \rangle$ veranschaulicht.

VideoContainer$\langle CL10 \rangle$

Aufgabe

Einbinden der Video und Textelemente in ein Paket, welches als Block manipulierbar ist.

Das Textelement bildet den Names des Users ab

Attribute

_container HTMLDivElement: tatsächlicher Container, zu dem das Videoelement und das Textelement gebunden werden
_video: HTMLVideoElement: das Videoelement, welches in dem Container gezeigt wird
_display: HTMLParagraphElement: Textelement, welches in dem Container gezeigt wird

Operationen

init(): void: initialisiert den VideoContainer mit dem Video und dem Textelement mit einem Standardnamen
setDisplay(string): void: setzt den innerText des _displayElementes auf den übergebenen string

Kommunikationspartner

keine

User⟨$CL20$⟩

Aufgabe

Verwaltung je eines VideoContainer- und Audioelements einens Users in der Konferenz

Attribute

audioBar: HTMLDivElement: Referenz zu dem audioBar-Element, welches alle audioTracks enthält
videoBar: HMTLDivElement: Referenz zu dem videoBar-Element, welches alle videoTracks enthält
focusBar: HTMLDivElement: Referenz zu dem focusBar-Element, je einen ausgewählten videoTrack enthält
participantId: string: ID des Users in der Jitsi-Session
conference: any: Referenz zu der Jitsi-Konferenz
disabled: boolean: speichert, ob der User sein Audio und Video am teilen ist
_audioMuted: boolean: speichert, ob der User sein Audio stummgeschaltet hat
_videoMuted: boolean: speichert, ob der User sein Video ausgeschaltet hat
audioTrack: any: speichert den audioTrack des Users
videoTrack: any: speichert den videoTrack des Users
audioElement: HTMLAudioElement: speichert das tatsächliche Audio Element als HTMLAudioElement

videoContainer: VideoContainer: speichert das VideoContainer-Objekt, welches den Namen und den Videostream des Users enthält

Operationen

removeTrack(any): void: entfernt den track, der übergeben wurde, von dem Element
setAudioTrack(any, boolean): void: erstellt ein neues oder ersetzt ein existierendes mit dem übergebenen Audioelement
setVideoTrack(any): void: setzt den Videotrack auf den, der der Funktion als Parameter übergeben wurde
toggleCamAudio(): boolean: stellt den Audiotrack von diesem User aus oder an
toggleCamVideo(): boolean: stellt den Videotrack von diesem User aus oder an
toggleTrack(any): boolean: stellt den ausgewählten Track aus oder an
pauseVideo(): boolean: gibt false zurück
updateVideoContainer(): void: aktualisiert alle Attribute des VideoContainers
update(): void: aktualisiert alle Attribute des Users
getRatio(): number: rechnet das Seitenverhältnis des Videos aus, gibt es als number:1 zurück
remove(): void: entfernt alle Audio und Videoelemente, außerdem deren Tracks mithilfe removeElements und detachTracks
removeElements(): void: entfernt alle Audio und Videoelemente
detachTracks(): void: entfernt alle Audio und Videotracks
dispose(): void: ruft disposeAudio und disposeVideo auf
disposeAudio(): void: entfernt alle Audioelemente und -tracks
disposeVideo(): void: entfernt alle Videoelemente und -tracks

Kommunikationspartner

Map, Player, Jitsi

Selfuser$\langle CL30 \rangle$

Aufgabe

Verwaltung der Video- und Audioelemente des Selfusers, hat auch die Funktionen des Users

Attribute

sharing: boolean: speichert, ob das Sharing an ist
sharedAudioMuted: boolean: speichert, ob das Videosharing an ist
sharedVideoMuted: boolean: speichert, ob das Audiosharing an ist
tempAudioTrack: any: speichert einen temporären Audiotrack zwischen

tempVideoTrack: any: speichert einen temporären Videotrack zwischen

Operationen

addToConference(): void: fügt die dem Konstruktor übergebenen Audio und Videotracks zur Konferenz hinzu
swapTracks(): void: stellt Muted- und Sharingparameter des tempVideoTracks auf die des exsitierenden, ruft dann swapTracksIntern auf
swapTracksIntern(): void: ersetzt den exisitierenden videoTrack des Objektes mit dem tempVideoTrack, und ruft update auf
pauseVideo(): boolean: gibt den sharing Parameter des Objektes zurück

Kommunikationspartner

keine

5.4 Implementierung von Komponente $\langle C40 \rangle$: <Jitsi>:

Der Jitsi service wird als Black Box behandelt, er stellt die Videochat-Funtion bereit. Er wird wie eine Bibliothek implementiert.

5.4.1 Paket-/Klassendiagramm

In Abbildung 5.4 wird das Packetdiagramm der Black Box Jitsi gezeigt, und deren benutzte Interfaces.

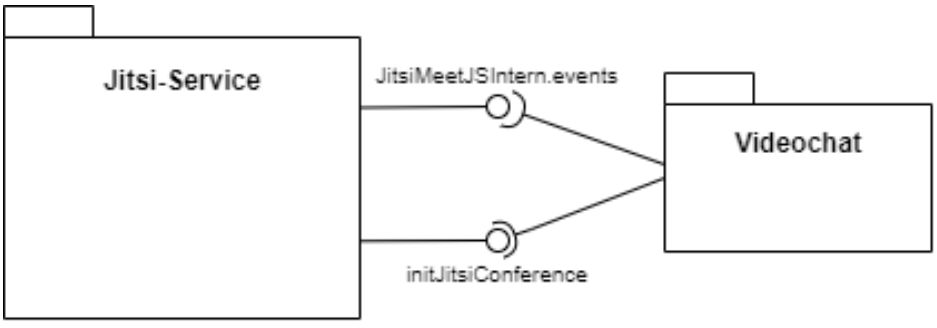

Abbildung 5.4: Packet-/Klassendiagramm für die Jitsi Komponente $\langle C40 \rangle$

5.4.2 Erläuterung

Jitsi hat für Kommunikation zu anderen Komponenten eine JitsiMeetJSIntern.events Funktion, die über Eventlistener von der Videochat-Komponente beobachtet wird. Hingegen wird die Jitsi-Komponente über eine bereitgestellte Klasse von Videochat manipuliert.

6 Datenmodell

Office Mania speichert keine Daten dauerhaft. Es gibt keine Anmeldung, denn die Anwendung soll für jeden möglichen Nutzer schnell zugänglich sein. Somit werden auch keine Daten hierfür gespeichert. Um die Nutzer dennoch nach dem Schließen und wieder Öffnen des Browsers wiederzuerkennen und eine neue Charaktererstellung zu verhindern, werden Daten im Local Storage beim Client gespeichert. Die Position auf der Map wird dabei nicht mitgespeichert. Diese Daten haben aber auch keinen dauerhaften Character, da der Nutzer selber sie schnell wieder löschen kann ohne, dass die Anwendung daran beteiligt ist. Die Map ist als Konfigurationsdatei zu betrachten und gehört damit zum Programm.
Während der Laufzeit gespeicherte Daten sind Informationen über Türen und die Positionen der Nutzer. Informationen über Türen werden gespeichert, um festzuhalten, ob eine Tür abgeschlossen oder offen ist. Das passiert solange die Anwendung auf dem Server läuft. Falls die Anwendung aber auf dem Server geschlossen wird, sind Informationen über Türen nicht relevant und werden daher nicht weiter gespeichert.
Die Position der Nutzer wird ebenfalls nicht dauerhaft gespeichert, um zu verhindern, dass Nutzer beim Einloggen ungewollt in zum Beispiel eine Besprechung gelangen. Daher gibt es einen festen Ort auf der Map, an den der Nutzer beim Starten des Clients gelangt. Das dauerhafte Speichern der Position wird somit nicht benötigt.

Bei der Interaktion mit Objekten kann es möglich sein, dass andere Anwendungen geöffnet werden, welche Daten speichern. Dabei handelt es sich um beispielsweise Bearbeitungsprogramme für Dateien. Diese Programme gehören aber nicht zu Office Mania.

7 Konfiguration

Als grundlegende Konfiguration reicht ein Server aus, auf dem eine aktuelle Version der Docker Engine läuft, die die amd64 Architektur und Linux Docker Container unterstützt.
Das Projekt ist deshalb ziemlich plattformunabhängig, was die Konfiguration des Servers angeht, da die Konfiguration des Projekts innerhalb des Docker Images und der Umgebungsvariablen beim Starten vorgenommen werden kann.
Die explizit benötigte Konfigurationsdatei `docker-compose.yml` ist bereits in unserem Git Repository enthalten, welches man auf den Server herunterladen sollte.

In dieser Konfigurationsdatei gibt es mehrere Parameter, die angepasst werden können.

- **build** Sollte nicht direkt angepasst werden, sondern auskommentiert werden, wenn **image** benutzt wird. Dies baut das Docker Image aus den lokalen Sources.
- **image** Gibt ein Image an, welches schon gebaut wurde, um das Projekt direkt zu starten (Nicht zusammen mit **build** benutzen!).
- **hostname** Hostname des Docker Containers, in dem das Projekt läuft.
- **container_name** Name des Docker Containers, in dem das Projekt läuft.
- **ports** Portweiterleitung vom Server zum Docker Container, in dem das Projekt läuft.
- **restart** Hiermit wird festgelegt, wann oder ob der Docker Container neu gestartet werden soll, falls er gestoppt wurde (z.B. durch einen Absturz).

Jedoch ist eine weitere explizite Nennung bestimmter Konfigurationsdateien für den Server nicht ohne weiteres möglich, da weitere Dinge wie bestimmte Portfreigaben, Firewalleinstellungen etc. abhängig vom jeweiligen Betriebssystem und verwendeter Software auf einem Server ist.
Außerdem überschreitet die benötigte Einrichtung von HTTPS für Verbindungen zu dem Server dieses Projekt, da dies abhängig von den Wünschen bzw. Vorgaben des Kunden ist, ob er z.B. eigene Zertifikate benutzt etc.

8 Änderungen gegenüber Fachentwurf

In diesem Kapitel fassen wir alle Änderungen gegenüber dem Fachentwurf zusammen bezüglich der Themen, die dort schon behandelt wurden.

Beim Systemablauf hat sich verändert, dass nicht Cookies, sondern Local Storage verwendet wird, um den Namen und den Skin zu speichern.

Die Analyse der Produktfunktionen hat sich im Vergleich zum Fachentwurf nicht verändert, da keine neuen Komponenten hinzugekommen sind, sondern die Komponenten bei dem Kapitel zur Softwarearchitektur nur anders bezeichnet werden, da es dort auch um die Schnittstellen geht und daher der Fokus anders liegt.

Da wir keine dauerhaft zu speichernden Daten haben, hat sich das Datenmodell im Vergleich zum Fachentwurf auch nicht verändert. Wir beschreiben hier lediglich genauer, warum wir keine Daten dauerhaft speichern.

Die Konfiguration nennt jetzt expliziter Konfigurationsdateien und anpassbare Parameter.

9 Erfüllung der Kriterien

Im folgenden wird beschrieben, wie die im Pflichtenheft genannten Kriterien erfüllt werden und auf was besonderes Augenmerk gelegt wird.

9.1 Musskriterien

Die folgenden Kriterien sind unabdingbar und müssen durch das Produkt erfüllt werden:

RM1 Die Darstellung der Karte wird durch Komponente C10, das Bewegen des Spielers durch Komponente C20 umgesetzt.
RM2 wird durch die Map-Komponente C10 umgesetzt.
RM3 Diese Funktionalität ist von allen Komponenten gewährleistet.
RM4 wird durch die Player-Komponente C20 realisiert.
RM5 Die Komponente C30 berechnet die hier notwendigen Parameter und startet die Videokonferenz über die Komponente C40.
RM6 Die Komponente C30 berechnet die hier notwendigen Parameter und beendet die Videokonferenz über die Komponente C40.
RM7 wird durch die Komponente C30 umgesetzt. Hierbei werden die Player IDs der Komponente C40 genutzt.
RM8 Die Map-Komponente C10 erfüllt diese Anforderung. Durch solide Objekte kann somit der Player der Komponente C20 nicht hindurch laufen.
RM9 Die Komponenten C10 und C20 arbeiten zusammen an dieser Darstellung.
RM10 Diese Anforderung wird durch ein effizientes Zusammenarbeiten aller Komponenten erreicht, kann aber auch von der Hardware des OfficeMania-Servers abhängig sein.
RM11 wird durch Komponente C20 erfüllt.

9.2 Sollkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

RS1 Diese Funktion soll von der Komponente C10 erfüllt werden, ist aber noch nicht implementiert.
RS2 Diese Funktion soll von der Komponente C10 erfüllt werden, ist aber noch nicht implementiert.
RS3 Diese Funktion soll von der Komponente C20 durch die Nutzung vom Local Storage erfüllt werden.
RS4 wird von der Map-Komponente C10 unterstützt. Alle anderen Komponenten arbeiten nach einem Austausch ebenfalls reibungslos.
RS5 Diese Funktion soll von der Komponente C10 erfüllt werden, ist aber noch nicht implementiert.
RS6 Diese Anforderung wird noch nicht erfüllt.
RS7 Diese Anforderung wird noch nicht erfüllt.
RS8 Diese Anforderung wird noch nicht erfüllt.

9.3 Kannkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

RC1 Diese Funktion kann von der Komponente C10 erfüllt werden, ist aber noch nicht implementiert.
RC2 Diese Funktion kann von der Komponente C10 erfüllt werden, ist aber noch nicht implementiert.
RC3 Diese Funktion kann von der Komponente C10 erfüllt werden, ist aber noch nicht implementiert.
RC4 Diese Funktion kann von der Komponente C20 erfüllt werden, ist aber noch nicht implementiert.
RC5 Diese Funktion kann von der Komponente C20 erfüllt werden, ist aber noch nicht implementiert.
RC6 Diese Funktion kann von den Komponenten C10 und C20 erfüllt werden, ist aber noch nicht implementiert.
RC7 Diese Funktion kann von der Komponente C40 erfüllt werden, ist aber noch nicht implementiert.
RC8 Diese Funktion kann von der Komponente C10 erfüllt werden, ist aber noch nicht implementiert.
RC9 Diese Funktion kann von der Komponente C10 erfüllt werden, ist aber noch nicht implementiert.

10 Glossar

Hier werden Fachbegriffe erklärt.
Avatar: Ein Avatar ist für einen Nutzer ein virtueller Charakter für eine virtuellen Welt.

Canvas: Ein Canvas ist ein HTML Element, auf welchem sich graphische Elemente darstellen lassen.

Client: Clients sind Programme oder Computer, die auf Server zugreifen können und von denen Daten oder Dienste abrufen.

Colyseus Framework: Das Colyseus Framework ist eine Sammlung von Funktionen, die der Entwicklung von Multiplayer-Anwendungen mit Node.js dient.

Cookies: Cookies sind kleine Textdateien, die auf dem Rechner gespeichert werden, was beim Surfen im Internet von einer Webseite über dem Webbrowser passiert. Wird die Seite noch mal besucht, wird der Cookie wieder aufgerufen.

Jitsi: Ist eine freie Open-Source-Videoconferencing-Software.

JQuery: Ist eine JavaScript Bibliothek, die dafür sorgt, JavaScript einfacher auf Webseiten zu verwenden. JQuery ermöglicht es hierbei dem Nutzer normale Aufgaben, für die man sonst mehrere Zeilen Code braucht, in einer Zeile Code zusammenzufassen.

Local Storage: Der Local Storage ist ein Teil vom Web Storage, was eine Technik zum Speichern von Daten in Browsern ist. Im Local Storage können Schlüssel Wert Paare lokal im Browser abgelegt werden, welche nicht gelöscht werden, nachdem die Seite oder der Browser geschlossen wurde.

Statechart: Statechart beschreibt das Antwortverhalten eines Objekts.