

## Titelblatt

### **Abstract**

- Ergebnisse aus den Spieletests, der Entwicklung und der Literaturrecherche zusammengefasst erläutern.
- Erst am Schluss schreiben
- Wichtig für den ersten Eindruck

# Contents

<b>1</b>	<b>Einleitung</b>	<b>2</b>
<b>2</b>	<b>Theoretischer Teil</b>	<b>3</b>
<b>3</b>	<b>Praktischer Teil</b>	<b>4</b>
3.1	Videospiel Palim-Palim . . . . .	4
3.1.1	Gamedesign . . . . .	4
3.1.2	Implementation . . . . .	4
3.1.3	WebRTC . . . . .	5
3.2	Durchführung Spieletests . . . . .	8
<b>4</b>	<b>Schlussfolgerung</b>	<b>9</b>
<b>5</b>	<b>Literaturverzeichnis</b>	<b>10</b>
<b>A</b>	<b>Ehrlichkeitserklärung</b>	<b>11</b>
<b>B</b>	<b>Testprotokolle, weitere Spielkonzepte...</b>	<b>12</b>

# Chapter 1

## Einleitung

### Teil 1

- Beschreibung des Videospiels Palim-Palim (inkl. Screenshot)
- Aufstellung der Forschungsfragen und den Erkenntnissen

### Teil 2

- Ausgangslage inkl. Forschungsstand
- Relevanz der Problemstellung

### Teil 3

- Grobe Beschreibung der angewendeten Methodik
  - Spielentwicklung (Architektur, Technologien)
  - Spieltests (Methoden)

### Teil 4

- Aufbau des Dokuments und Überleitung in den theoretischen Teil

# Chapter 2

## Theoretischer Teil

- Beschreibung des Umfelds / Anwendungsdomäne (betagte Personen und Kinder)
- Intergenerationelles Spielen
  - Forschungsstand (Welche Methoden/Ansätze werden angewendet?)
  - Bisherige Erkenntnisse zu intergenerationellem Spielen
- Videochats in Videospielen
  - Forschungsstand (Welche Methoden/Ansätze werden angewendet?)
  - Bisherige Erkenntnisse zu Videochats in Videospielen
- Lücken der bisherigen Forschung
- Aufstellung der Forschungsfragen
- In Palim-Palim verwendete Methoden
  - Methode der Spieletest
  - Methode der Spieletest-Auswertung

# Chapter 3

## Praktischer Teil

### 3.1 Videospiel Palim-Palim

#### 3.1.1 Gamedesign

- Auswahl Spielkonzept
- Spielkonzept
- Gameplay-Loops
- Spielvarianten
- Design und Usability

#### 3.1.2 Implementation

- Technologien
- Architektur
- Sicherheit
- Testing
- Deployment
- Betrieb

### 3.1.3 WebRTC

WebRTC [<https://webrtc.org/>] ist ein Open-Source-Projekt, das die Echtzeitkommunikation von Audio, Video und Daten in Web- und nativen Anwendungen ermöglicht. Die Technologie ist in allen modernen Browsern sowie auf nativen Clients für alle wichtigen Plattformen verfügbar. Dabei wird zwischen zwei Browsern eine Peer-To-Peer-Verbindung aufgebaut, worüber die Daten gestreamt werden.

#### Signaling

WebRTC verwendet die `RTCPeerConnection` [<https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection>] JavaScript API, um Streaming-Daten zwischen Browsern zu kommunizieren. Zusätzlich wird aber auch einen Mechanismus benötigt, um die Kommunikation zu koordinieren und Kontrollnachrichten zu senden. Dieser Prozess wird als *Signaling* bezeichnet. Signaling-Methoden und -protokolle sind von WebRTC nicht spezifiziert und können je nach Anwendungsfall entsprechend gewählt werden.

In diesem Projekt wurde `Socket.IO` für die Signalisierung verwendet, da es sich mit seinem integrierten Room-Konzept für eine Video-Chat-App anbietet. Es gibt aber viele Alternativen, welche sich teilweise besser für produktive Applikationen eignen (<https://bloggeek.me/siganling-protocol-webrtc/>). `PalimPalim` verwendet eine Node.js Server-Applikation (implementiert in `server.js`), welche als Signaling-Server fungiert. Der Server hat dabei folgende zwei Aufgaben.

Erstens fungiert er als Message Relay. Dies wird benötigt, um verschiedene Informationen von und zu den Clients zu senden, damit diese untereinander eine WebRTC-Peer-Verbindung aufbauen können:

```
1 socket.on('message', function (message) {
2   log('Got message: ', message);
3   socket.broadcast.emit('message', message);
4 });
```

Zweitens verwaltet er alle WebRTC-Videochat-'Räume':

```
1 if (numClients === 0) {
2   socket.join(room);
3   socket.emit('created', room, socket.id);
4 } else if (numClients === 1) {
5   socket.join(room);
6   socket.emit('joined', room, socket.id);
7   io.sockets.in(room).emit('ready');
8 } else { // max two clients
9   socket.emit('full', room);
```

Wie im Code ersichtlich ist, erlaubt unsere Applikation maximal zwei Peers in einem Raum.

Zusätzlich werden in unserer Applikation die Positionen der Gameobjekte über den Server synchronisiert (TODO).

## STUN und TURN

WebRTC ist grundsätzlich so konzipiert, dass es Peer-to-Peer funktioniert. Benutzer können sich also auf dem direktesten Weg verbinden. Die Technologie ist jedoch darauf ausgelegt, mit realen Netzwerken zurechtzukommen: Client-Anwendungen müssen NAT-Gateways und Firewalls überwinden, und Peer-to-Peer-Netzwerke benötigen Fallbacks, falls die direkte Verbindung ausfällt. Als Teil dieses Prozesses verwenden WebRTC-APIs sogenannte *STUN-Server*, um die IP-Adresse ihres Computers zu ermitteln, und *TURN-Server* (Traversal Using Relay NAT), welche als Relay-Server fungieren, falls die Peer-to-Peer-Kommunikation fehlschlägt. [<https://webrtc.org/getting-started/turn-server>]

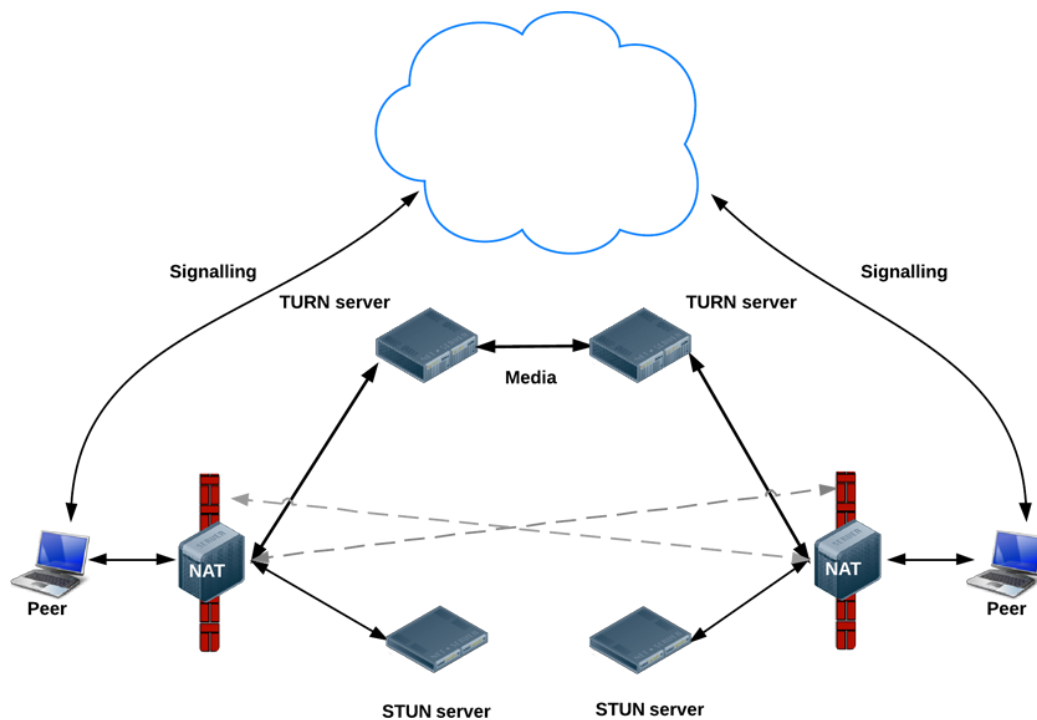


Figure 1: STUN, TURN, und Signalisierung in WebRTC. Quelle: <https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/>.



Palim-Palim verwendet als STUN-Server öffentlich verfügbare Server von Google. Als TURN UDP- und TCP-Fallback-Verbindungen werden die Konfigurationen aus dem WebRTC-Sample-Projekt übernommen. Diese Server werden gratis angeboten und können mit Tools wie Google's Trickle ICE sample [<https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/>] auf ihre Verfügbarkeit getestet werden [<https://bloggeek.me/webrtc-turn/>]. Die STUN- und TURN-Server-Adressen müssen bei dem Aufbau der Peer-Connection vom Client an den Signaling-Server übermittelt werden. Dazu übermittelt jeder Palim-Palim-Client beim Verbindungsaufbau folgende pcConfig-Werte an den Server:

```
1      var pcConfig = {
2      'iceServers': [
3      {
4      'urls': 'stun:stun.l.google.com:19302'
5      },
6      {
7      'urls': 'turn:192.158.29.39:3478?transport=udp',
8      'credential': 'JZE0Et2V3Qb0y27GRntt2u2PAYA=',
9      'username': '28224511:1379330808'
10     },
11     {
12     'urls': 'turn:192.158.29.39:3478?transport=tcp',
13     'credential': 'JZE0Et2V3Qb0y27GRntt2u2PAYA=',
14     'username': '28224511:1379330808'
15     }
16   ]
17 };
```

In etwa 82 Prozent der Fälle ist die Peer-To-Peer-Verbindung stabil genug, und die TURN-Server werden überhaupt nicht benötigt. Jedoch müssen sie trotzdem bei jeder Verbinung angegeben werden. TODO more infos <https://www.callstats.io/blog/2017/10/26/webrtc-product-turn-server>

## Sicherheit

Verschlüsselung ist für alle WebRTC-Komponenten obligatorisch, und seine JavaScript-APIs können nur von sicheren Quellen (HTTPS oder localhost) aus verwendet werden. Die Signalisierungsmechanismen sind allerdings nicht in den WebRTC-Standards definiert. Hier liegt es an den Entwicklern, sichere Protokolle zu verwenden. TODO;Für unser Projekt noch definieren und ausformulieren;

## **3.2 Durchführung Spieletests**

- Organisation der Spieletests (Aufbau, Ablauf, Testpersonen, Testszenarien, Testumgebung)
- Beobachtungen
- Ergebnisse
- Beantwortung der aufgestellten Forschungsfragen

# Chapter 4

## Schlussfolgerung

- Zusammenfassung des Erreichten / Zielerreichung
- Zentrale Erkenntnisse
- Reflektion
- Mögliche Weiterentwicklungen (bezogen auf die Software)
- Weiterführende Forschung

## Chapter 5

### Literaturverzeichnis

# Appendix A

## Ehrlichkeitserklärung

## Appendix B

Testprotokolle, weitere  
Spielkonzepte...