Bachelorarbeit

Erstellung eines browserbasierten Webprogramms zur Moderation des Gesellschaftsspiels "Mord In Palermo"

Zur Erlangung des Grades Bachelor of Engineering

Vorgelegt von

Martin Hennings

Marcel Haupt

Prüfer

Prof. Dr.-Ing. Jörg J. Buchholz

Prof. Dr. Volker Paelke

September 2018

Inhaltsverzeichnis

1 Einleitung

Da heutzutage fast jeder ein Smartphone besitzt, liegt es auf der Hand klassische Gesellschaftsspiele mit Appunterstützung auszustatten. Man stelle sich beispielsweise “Monopoly” oder “Spiel des Lebens” vor, bei dem das Geld per App überwiesen werden kann. Oder aber man erweitert die Realität mit Spielinhalten, ganz im Sinne von Augmented Reality.[1]

Für das Spiel ”Mord in Palermo” haben wir eine praktische Appunterstützung vermisst, bei der Abstimmungen und Spielverlauf nicht mehr durch eine weitere Person, die nicht am Spiel teilnehmen kann, durchgeführt werden muss, sondern alles bequem mit dem Smartphone gemacht werden kann. Unsere Recherche ergab, dass zwar Variationen des Spiels als Browsergame existieren, diese jedoch nicht mit dem Gesellschaftsspiel in der Realität kompatibel waren, da sie als reines Browsergame konzipiert waren. Eine anwendungsfreundliche Appunterstützung für Mord in Palermo ließ sich dagegen in Apple’s Appstore finden. Die App “Werewolf” moderiert das Spiel, indem das Smartphone herum gereicht wird und jeder Spieler nacheinander seine rollenspezifischen Abstimmungen tätigt. Der Moderator des Spiels wird dadurch zwar ersetzt, jedoch wird die Anonymität der einzelnen Rollen unserer Meinung nach gefährdet, wenn beispielsweise der Mörder oder hier der Wolf eine Abstimmung mehr tätigt als andere und das Smartphone deswegen etwas länger in der Hand hält. Außerdem fanden wir diese Lösung äußerst unpraktisch, da eine Spielrunde bei vielen Teilnehmern durch das Umherreichen des Smartphones oft mehrere Minuten dauern kann. Es musste also eine Spielunterstützung her, die nicht nur die Aufgaben des Moderators übernimmt, sondern auch praktisch ausgelegt ist, damit der Spielfluss auch bei unbegrenzt vielen Spielern nicht gestört wird, und außerdem das zwischenmenschliche Spielerlebnis durch Zusatzfunktionen, wie z. B. einen Messenger zum Schreiben geheimer Nachrichten steigert.

[1] Wikipedia: Augmented Reality, Stand: 05. Aug. 2018, URL: `https://en.wikipedia.org/wiki/Augmented_reality`

2 Mord In Palermo

2.1 Das Spiel

"Mord In Palermo" ist ein Gesellschaftsspiel und wurde abgleitet vom bekannten Spiel "Mafia".[2] Zu Beginn des Spiels wird jedem Spieler seine geheime Rolle zugewiesen. Er spielt nun entweder auf der Seite der Bürger oder der Mörder. Die Mörder wollen unentdeckt bleiben und geben sich als Bürger aus. In jeder Runde bestimmen sie dabei einen Bürger, den sie "ermorden." Das Ziel der Bürger ist es, die Mörder unter sich ausfindig zu machen und zu entlarven. Die Mörder versuchen die Bürger dabei auf eine falsche Fährte zu führen. In jeder Runde stimmen die Spieler dann darüber ab, wer als Mörder infrage kommt und bestimmen einen Spieler, der "erhängt" werden soll.

Unter den Bürgern gibt es solche mit besonderen Fähigkeiten, wie den Arzt oder den Spion. Diese Fähigkeiten können dazu genutzt werden, die Mörder ausfindig zu machen oder ihren "Mordanschlag" abzuwehren. Da diese Fähigkeiten den Mördern gefährlich werden können, müssen diese Spieler sehr vorsichtig mit ihrer wahren Identität umgehen.

2.2 Die Rollen

2.2.1 Spion

Als Spion ist man der Sherlock Holmes unter den Bürgern. Er hat die besondere Fähigkeit, jede Runde die wahre Identität eines von ihm ausgewählten Spielers zu erfahren. Die Mörder haben es als Erstes auf ihn abgesehen, denn er kann sie am ehesten entlarven.

2.2.2 Arzt

Der Arzt hat die Fähigkeit einen Mordanschlag abzuwehren. Dazu wählt er jede Runde einen Mitspieler aus und macht ihn dadurch immun gegen einen Mordversuch. Dabei muss der Arzt seinen Instinkt nutzen, denn er weiß nicht, wen die Mörder ausgewählt haben. Seine Hauptaufgabe ist es, den Spion am Leben zu halten, damit dieser die Mörder entlarven kann.

2.2.3 Mörder

Die Mörder müssen zusammenhalten. Sie kennen die Identität der anderen Mörder und können sich im Geheimen absprechen. Sie müssen sich nach

[2]Wikipedia: Mafia (Gesellschaftsspiel), Stand: 28. August 2018, URL: `https://de.wikipedia.org/wiki/Mafia_(Gesellschaftsspiel)`

außen als Bürger ausgeben, um am Ende der Runde nicht gehängt zu werden. Ihr Ziel ist es, die Bürger zu töten bevor sie selber entlarvt und gehängt werden. Ihre ersten Zielpersonen sollten der Spion und die Ärzte sein, denn mit ihren besonderen Fähigkeiten können sie den Mördern gefährlich werden. Achtung! Die Mörder müssen sich darüber einig sein, wen sie umbringen, denn sie können nur einen Mitspieler pro Runde ermorden.

2.2.4 Bürger

Die normalen Bürger haben keine besonderen Fähigkeiten. Sie müssen auf ihren Instinkt vertrauen und aktiv in den Rundendiskussionen zwischen Freund und Feind unterscheiden lernen. Sie stimmen in jeder Runde für einen Spieler ab, den sie für einen Mörder halten. "Gehängt" wird schließlich der Spieler mit den meisten Stimmen.

2.3 Klassischer Spielverlauf

Zu Beginn jedes Spiels wird ein Spielleiter ausgewählt. Dieser bestimmt die Rollen der einzelnen Spieler, indem entweder Zettel gezogen werden oder alle Spieler die Augen schließen und der Spielleiter die Mörder, Ärzte und Detektive nacheinander antippt.

Das Spiel beginnt mit einer Diskussionsrunde. Die Mörder versuchen sich als Bürger auszugeben und die anderen Spieler auf eine falsche Spur zu führen. Die Ärzte, Bürger und Spione versuchen durch Verschwörungstheorien und hitzige Diskussionen die Mörder ausfindig zu machen. Am Ende jeder Runde kommt es zur Abstimmung. Es wird ein Spieler ausgewählt, der von den anderen erhängt wird und somit aus dem Spiel ausscheidet.

Anschließend beginnt die Nacht. Der Spielleiter bittet alle Mitspieler, die Augen zu schließen. Er lässt die Mörder erwachen. Die Mörder verständigen sich mit Handzeichen, um sich auf ein Opfer zu einigen. Der Spielleiter merkt sich die Entscheidung der Mörder. Die Mörder schließen die Augen und der Spielleiter bittet alle Ärzte, die Augen zu öffnen. Jeder Arzt entscheidet sich individuell, wen er retten möchte. Anschließend darf der Spion die Augen öffnen und eine Person auswählen. Ist die ausgewählte Person ein Mörder, so nickt der Spielleiter. Jetzt beginnt der Tag, und alle Spieler dürfen wieder ihre Augen öffnen. Der Spielleiter gibt nun bekannt, welcher Spieler durch die Mörder getötet wurde. Danach beginnt die Diskussion erneut.

Es gibt zwei Möglichkeiten zur Beendigung des Spiels. Die Bürger haben gewonnen, wenn alle Mörder erhängt worden sind. Hingegen haben die Mörder gewonnen, wenn genauso viele Bürger wie Mörder übrig sind.

2.4 Unterschiede im Spielverlauf der browserbasierten Webanwendung

In der von uns, im Rahmen der Bachelorarbeit, entwickelten Version des Spiels wird der Spielleiter und damit die Nacht ersetzt. Spieler tätigen während der Diskussionsrunde ihre Abstimmungen und Mörder, Detektive und Ärzte können ihre rollen-spezifischen Abstimmungen zeitgleich mit den anderen tätigen. Jeder Spieler wird darüber informiert, wer gegen ihn abgestimmt hat und alle Spieler sehen in Echtzeit welcher Spieler bereits wie viele Stimmen hat. Die Abstimmungen finden alle gleichzeitig und nicht nacheinander statt, da Mörder, Ärzte und Spion zwei Abstimmungen zu machen haben und niemand anhand der Häufigkeit der Benutzung des Smartphones daraus schließen können sollte, wer eine geheime Rolle innehat.

Am Ende der Runde wird dem Spieler, wie im klassischen Spiel auch, das Ergebnis der Abstimmung auf dem Bildschirm mitgeteilt. Gerade bei Spielen mit sehr vielen Mitspielern wird hier weniger Zeit als im klassischen Spiel aufgewendet, um die Abstimmungen zu tätigen und auszuwerten.

Auch wenn "Mord in Palermo" ein Face-To-Face Gesellschaftsspiel ist und eigentlich keine Nachrichtenfunktion benötigt, eröffnet der Messenger eine weitere Kommunikationsebene im Spiel. Dadurch, dass man private Nachrichten an Mitspieler schicken kann, hat man nun die Möglichkeit sich geheim mit einem oder mehreren Spielern abzusprechen.

Den wohl größten Nutzen daraus ziehen die Mörder, denn sie werden zu Beginn jedes Spiels einem "Komplizen"-Chat hinzugefügt, der dazu animieren soll, sich abzusprechen.

Mörder können nun Strategien und Theorien entwickeln, ohne dass andere Mitspieler davon etwas mitbekommen. Aber auch die Bürger können sich die Chatfunktion zu Nutze machen, indem beispielsweise der Detektiv Gruppenchats mit denjenigen Spielern erstellt, die er bereits überprüft hat und denen er vertraut, um Einfluss auf das Ergebnis der Abstimmung zu nehmen. Natürlich kann diese Funktion wiederum von einem ausgefuchsten Mörder ausgenutzt werden, um die Bürger in die Irre zu führen. Es ist also ein Spiel mit doppeltem Boden!

3 Aufgabenstellung

Im Rahmen unserer Bachelorarbeit wollen wir ein Programm erstellen, mit dem der Klassiker "Mord In Palermo" gespielt werden kann. Da heutzutage die große Mehrheit der Menschen über ein mobiles Gerät verfügt, soll jeder Spieler mit seinem eigenen Smartphone am Spiel teilnehmen können. Über unser Programm sollen spielentscheidende Abstimmungen getätigt, private und öffentliche Nachrichten geschickt und Sonderaktionen getätigt werden können. Die Software ersetzt damit den Spielleiter, der normalerweise nicht regulär mitspielen kann.

Folgenden Funktionsumfang soll die Software erfüllen:

- Eine Person startet das Spiel (der "Host") und lädt die Mitspieler über einen Einladecode oder einen Link ein
- Die Mitspieler geben den Code ein oder klicken auf den Link und treten automatisch dem Spiel bei
- Sobald alle im Spiel sind, kann der Host das Spiel starten
- Einstellungen können vor dem Start vom Host vorgenommen werden
- Die Software verteilt automatisch die Rollen
- Abstimmungen können über die Benutzeroberfläche erfolgen
- Man soll sehen können, welche Mitspieler wie viele Stimmen bekommen haben
- Mörder, Ärzte und Spione haben eine zusätzliche Funktion für rollenspezifische Abstimmungen
- Detektive haben die Möglichkeit, jede Runde die Rolle eines Spielers zu erfahren
- Senden von Nachrichten an alle Teilnehmer, an Rollenpartner oder beliebige Mitspieler

Die Benutzeroberfläche soll folgende Vorgaben erfüllen:

- Erstellung der Benutzeroberfläche mit HTML, CSS und Javascript

- Optimierung der Benutzeroberfläche für mobile Geräte
- Darstellung der Spielerliste zum Abstimmen
- Zusätzliche Funktion zum Abstimmen für Mörder und Ärzte
- Nachrichtenfenster zum Abschicken von Nachrichten
- Ansprechendes, modernes Design
- Datenabgleich mit dem Server über Javascript

Für den Server möchten wir folgenden Funktionsumfang haben:

- Erstellen von neuen Spielen
- Verwalten von laufenden Spielen
- Datenabgleich mit den einzelnen Spielern
- Einhaltung der Datenschutzverordnungen
- Sichere Verbindung durch HTTPS

4 Arbeitsumgebung

Es stellt sich zunächst die Frage, mit welcher Programmiersprache und unter welcher Umgebung wir das Spiel erstellen wollen. Zur Wahl standen die native Mobilapp oder die flexible Webapplikation.

Mobilapps haben grundsätzlich mehr Zugriff auf hardwarenahe Ressourcen. Sie laufen in der Regel schneller, können auf Sensoren zugreifen und können im Hintergrund laufen, auch wenn der Bildschirm gerade gesperrt ist. Im Gegenzug verlangen sie größeren Aufwand beim Programmieren. Es müssen für Android, iOS und Windows Apps unterschiedlichen Programmiersprachen verwendet werden.

Seit kurzem gibt es zusätzlich die sogenannten hybriden Apps[3]. Sie werden mit HTML, CSS und Javascript geschrieben und können mit Hilfe eines Compilers in Apps für verschiedene Mobilgeräte übersetzt werden. Hierbei

[3]Wikipedia: Mobile_App, Stand: 26. Juli 2018, URL: `https://de.wikipedia.org/wiki/Mobile_App#Hybrid-Apps`

ist sogar der Zugriff auf Sensoren des Mobilgeräts wie GPS, Beschleunigungsmesser oder Kamera möglich.

Für unsere Software ist lediglich die Eingabe von Text und das Klicken auf Elemente notwendig. Wegen geringeren Aufwands und durch den großen Vorteil, dass eine Webapplikation auf allen erdenklichen Geräten mit Webbrowser läuft, entschieden wir uns für eine Webapplikation.

Statische Webseiten schreibt man für gewöhnlich mit HTML und CSS. HTML ist für die semantische Gliederung der Webseite verantwortlich, während CSS für das Aussehen verantwortlich ist. Für das Zeichnen von dynamischen Inhalten und für die Kommunikation mit dem Server verwenden wir zusätzlich Javascript.

Unseren Server bestücken wir mit dem Webserver Apache2, dem Datenbankserver MariaDB und der PHP Skriptumgebung. Unsere Skripte für den Server können wir damit in PHP schreiben, der meistbenutzten Sprache für Webprogrammierung. Jegliche Daten zu Spielen und Spielern, sowie die Nachrichten können in unserer Datenbank bequem über SQL gespeichert werden. Ereignisse und Informationen können über das Internet gesendet werden und dynamisch mittels Javascript auf der Webseite gezeichnet werden.

Zur Datenübertragung bei Webapplikationen gibt es prinzipiell zwei Möglichkeiten: Der unidirektionale Transfer von Informationen über HTTP Abfragen (die 'Pull Methode') oder eine bidirektionale TCP Verbindung durch Websockets (die 'Push-Pull Methode').

4.1 Push-Pull Methode

Bei neuen Daten kann der Server dem Benutzer direkt Nachrichten schicken, ohne auf dessen Reaktion warten zu müssen. Damit können Daten viel schneller übertragen werden. Außerdem kann man mit dieser Methode Datenvolumen einsparen, da kein ständiges Nachfragen des Benutzers nach neuen Daten notwendig ist.

Der große Nachteil von Push Benachrichtigungen ist der große Programmieraufwand und das Benötigen von bestimmten Berechtigungen auf der Seite des Mobilgeräts. Außerdem unterstützen nicht alle Browser diese Funktion.

4.2 Pull Methode

Bei dieser Methode werden Daten nur vom Benutzer abgerufen. Die Benutzerseite muss somit in regelmäßigen Abständen den Server nach neuen Informationen fragen und der Server kann nicht mehr eigenständig dem Benutzer

Informationen schicken. Der große Vorteil dieser Methode ist die einfache Umsetzung.

Datenanfragen können bequem über einen HTTP-Request erfolgen und benötigen keine aufwendige Programmierung. Da unser Spiel sowieso nur geringe Datenmengen verschicken muss und beim Spielen Zuhause häufig WLAN verfügbar ist, entschieden wir uns für die Pullmethode.

Natürlich ist es weder notwendig noch effizient, das komplette Spiel als Datenkonstrukt bei jedem Pull zu verschicken. Es lag daher auf der Hand, lediglich einmal am Anfang alle vorhandenen Daten zu senden und danach nur noch die neusten Daten zu verschicken. Dafür braucht es aber einen speziellen Aufbau.

Der Browser sollte dafür mit Javascript alle wichtigen Daten des Spiels speichern. Sobald neue Daten hinzukommen, kann er die alten Daten mit den neuen ergänzen und verschiedene Aktionen ausführen, um die Daten zu verarbeiten und visuell darzustellen. Außerdem muss der Server wissen, was genau die neuesten Daten sind. Um zu wissen, wie alt Daten sind, versahen wir jede Datenbanktabelle mit einem Zeitstempel. Ändert sich so beispielsweise der Name eines Spielers, so wird der Zeitstempel ”modified_at” dieses Spielers auf die aktuelle Zeit gesetzt. Somit können alle Spalten aus der Datenbank geholt werden, die neuer sind als der Zeitpunkt der letzten Abfrage. Hierbei gibt es jedoch ein Problem.

Zeitstempel besitzen immer eine gewisse Auflösung. In unserem Fall (da wir die UNIX Zeit nutzen) ist das eine Sekunde. In dieser Sekunde können in der Informatik sehr viele Dinge gleichzeitig passieren.

Nehmen wir ein Beispiel an, in dem am Anfang der Sekunde 100 eine Pullabfrage ausgeführt und kurz danach am Ende der gleichen Sekunde eine Variable geändert wird. Der Zeitstempel der letzten Pullabfrage ist somit 100 (lastpull = 100). Ebenso ist ”modified_at” dieser Variable 100. Holen wir in der nachfolgenden Pullabfrage alle neuen Daten, deren Zeit größer als 100 ist (modified_at > lastpull), bekommen wir die eben genannte Variablenänderung nicht mit. “Größer als” können wir also nicht nehmen, da sonst einige Daten nicht gesendet werden. ”Größer gleich” würde zwar in diesem Beispiel funktionieren, macht aber in einem anderen Beispiel ebenfalls Probleme.

Nehmen wir dafür ein zweites Beispiel an, in dem am Anfang der Sekunde 100 eine Variable geändert wird und kurz danach eine Pullabfrage stattfindet. ”modified_at” und ”lastpull” sind somit beide wieder 100. Der Unterschied ist jedoch, dass der erste Pull bereits die Variablenänderung verschickt. Folgt jetzt der nächste Pull und werden alle Daten genommen, bei denen ”modi-

fied_at" größer gleich "lastpull" sind, wird die Änderung erneut verschickt.

Natürlich ist ein doppeltes Verschicken nicht so schlimm, als wenn Daten gar nicht gesendet werden. Trotzdem muss der Empfänger mit doppelten Daten umgehen und nicht blind den gleichen Spieler zweimal hinzufügen.

Ein weiteres Beispiel, bei dem das doppelte Empfangen von Daten ein Problem darstellt, ist der Messenger. Um sicherzustellen, dass ein neu erstellter Chat nicht zweimal vom Empfänger gespeichert und gezeichnet wird, muss überprüft werden, ob dieser Chat bereits vorhanden ist.

Die empfangenen Chat-Daten werden in der game.js-Datei mit den bereits vorhandenen Chat-Daten verglichen.

```
//get chat data
if (data.data.chat.length > 0)
{
  //check if chat appears two times
  for (var x = 0; x < data.data.chat.length; x++)
  {
        var chatDouble = false;

    for (var j = 0; j < GAME.chats.length; j++)
    {
      //if chat already exists, replace relevant information
      if (data.data.chat[x].id == GAME.chats[j].id)
      {
      chatDouble = true;
      GAME.chats[j].chat_member = data.data.chat[x].chat_member;
      GAME.chats[j].modified_at = data.data.chat[x].modified_at;
      GAME.chats[j].name = data.data.chat[x].name;
      GAME.chats[j].type = data.data.chat[x].type;

      break;
      }//if
     }//for y
```

Anhand der Chat-ID kann man erkennen, ob ein bestimmter Chat bereits im lokalen Speicher vorhanden ist. In diesem Fall werden relevante Daten, wie Chatmitglieder, Chattitel, Art des Chats und die Timestamp mit den neuen Daten vom Server überschrieben, um immer die aktuellsten Informationen über diesen Chat zu speichern.

Bleibt die Variable "chatDouble" hingegen "false", existiert der empfan-

gene Chat noch nicht im lokalen Speicher. In diesem Fall wird mit diesen Daten ein neues Element im Array ”GAME.chats” erstellt. Damit wird der Chat dem lokalen Speicher hinzugefügt.

```
// If chat doesn't exist yet, create new chat object
  if (!chatDouble)
  {
      GAME.chats.push(data.data.chat[x]);
      var li = GAME.chats.length-1;
      GAME.chats[li].messages = [];
      GAME.chats[li].last_message_time = GAME.chats[li].modified_at;
      GAME.chats[li].last_time_checked = GAME.chats[li].modified_at;
  }//if
}//for x
```

Das gleiche Problem tritt bei Nachrichten auf. Um doppelte Nachrichten zu vermeiden, muss überprüft werden, ob die empfangene Nachricht bereits im lokalen Speicher existiert. Für jeden existierenden Chat wird zunächst geschaut, ob es dort neue Nachrichten gibt.

```
//get message data
if (data.data.message.length > 0)
{
  for (var i = 0; i < GAME.chats.length; i++)
  {
    //if message doesn't exist, create empty array to avoid errors
    if (GAME.chats[i].messages == null)
    {
      GAME.chats[i].messages = [];
    }
    //select the corresponding chat for the new message
    for (var x = 0; x < data.data.message.length; x++)
    {
      if (GAME.chats[i].id != data.data.message[x].chat_id)
      {
        continue;
      }
```

Um doppelte Nachrichten zu vermeiden, überprüfen wir die ID’s der letzten

fünf Nachrichten aus diesem Chat (sofern dieser Chat bereits Nachrichten enthält).

```
var msgDouble = false;
var startIndex = GAME.chats[i].messages.length -5;

//check for double messages
if (startIndex < 0) startIndex = 0;
  for (var j = startIndex; j < GAME.chats[i].messages.length; j++)
  {
    if (data.data.message[x].id == GAME.chats[i].messages[j].id)
    {
      msgDouble = true;
      break;
    }
```

Wurde keine doppelte Nachricht gefunden, bleibt die Variable ”msgDouble = false” und die Nachricht wird gespeichert. Außerdem erhält der betroffene Chat eine ”last_message_time”, die für den Zähler der noch ungelesenen Nachrichten von Bedeutung ist.

```
  }//for j
  //only if message doesn't already exist, push it into the object
  //and create a new last message time stamp
  if (!msgDouble)
  {
    GAME.chats[i].messages.push(data.data.message[x]);
    GAME.chats[i].last_message_time = data.data.message[x].time;
   }
 }//for x
}//for i
```

Durch diese Verfahren verhindert man effektiv das doppelte Speichern und Darstellen von doppelt versendeten Daten.

5 Benutzerinterface

Das perfekte Benutzerinterface für unser Spiel muss einerseits auf mobile Endgeräte optimiert, andererseits sehr intuitiv zu bedienen sein. Alle Buttons müssen zudem groß genug für eine Toucheingabe sein. Um das Rad nicht neu erfinden zu müssen, entschieden wir uns für einige Bibliotheken, die uns grundlegende Arbeiten abnehmen. Das hat den großen Vorteil, dass wir mehr Gedanken und Arbeit in das eigentliche Problem und die Software stecken können und uns nicht unverhältnismäßig lang am Design einzelner Schaltflächen und Elemente aufhalten müssen.

Als erstes entschieden wir uns für eine Bibliothek namens Bootstrap. Bootstrap ist eine HTML und CSS Bibliothek, die vor allem wegen ihrer dynamischen Gitter, vorgefertigten Komponenten, wie Buttons, Labels und Popups sowie vielen nützlichen CSS Klassen, gerne von Webdesignern benutzt wird. Mit Bootstrap ist es sehr leicht, für Mobilgeräte optimierte Webseiten zu erstellen. Das durchdachte Gittersystem von Bootstrap ist in der Lage, die Webseite automatisch an die Bildschirmbreite anzupassen und einzelne Elemente zu skalieren oder deren Anordnung zu verschieben.

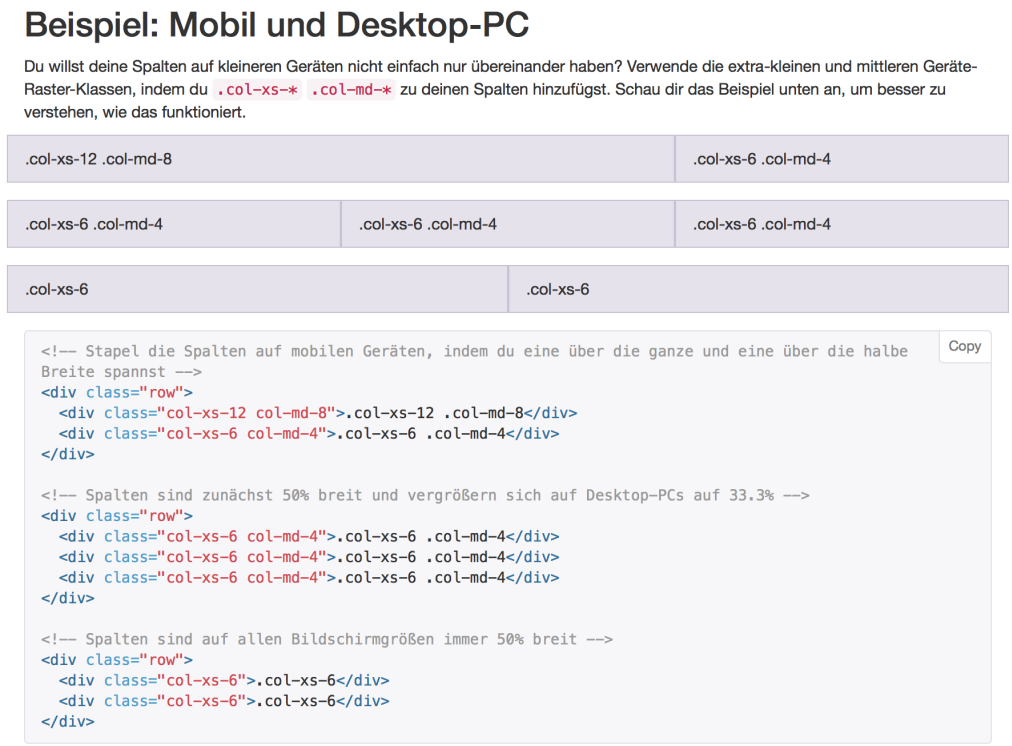

Abbildung 1: Ein Beispiel für das Gittersystem von Bootstrap. Die Spalten werden je nach Gerätebreite unterschiedlich dargestellt.
[4]

Mobile Benutzerinterfaces lassen nicht viel Spielraum für erklärenden Text übrig. Besonders deshalb war eine intuitive Gestaltung sehr wichtig. Für diesen Zweck sind Symbole sehr gut geeignet. Beispielsweise ist es leicht zu verstehen, dass sich hinter dem Briefumschlag eine Nachrichtenfunktion versteckt. Symbole, die der Benutzer wiedererkennt, können Textbeschreibungen von Schaltflächen ersetzen und somit wertvollen Platz auf dem Bildschirm einsparen. Eine sehr umfangreiche Bibliothek, die eine Vielzahl an hilfreichen Symbolen zur Verfügung stellt ist "Fontawesome". Fontawesome kann mittels HTML, CSS und JS Vektorgrafiken an beliebige Stellen in die Webseite einfügen.

[4]Bootstrap: CSS, Stand: 16. Aug. 2018, URL: `http://holdirbootstrap.de/css/#grid`

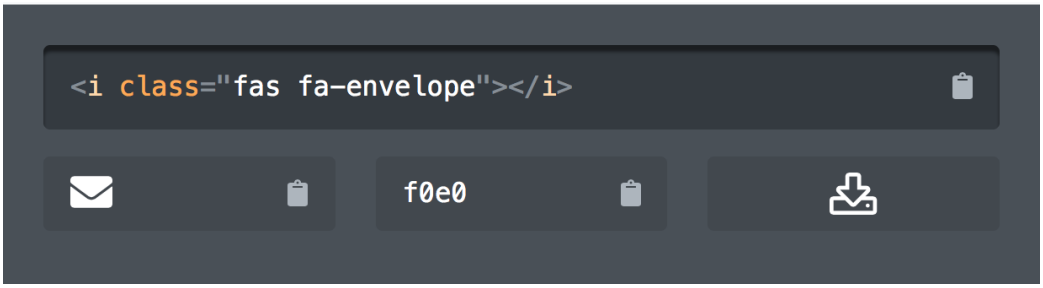

Abbildung 2: Ein Beispiel für Fontawesome. Hier sieht man den nötigen HTML Code und das Icon wie es im Dokument steht.
[5]

Um aktuelle Daten vom Server zu laden und unser Benutzerinterface auch dementsprechend in Echtzeit verändern zu können, war uns schon am Anfang klar, dass wir sehr viel Programmcode in Javascript schreiben müssen. Eines der hilfreichsten Tools um mit Javascript und HTML arbeiten zu können ist jQuery. Diese Bibliothek wurde entworfen, um die Manipulation von HTML einfacher zu gestalten. Ebenso befinden sich einige Funktionen in jQuery, die das Senden und Verarbeiten von HTTP Anfragen einfacher machen.

[5]Fontawesome: Envelope, Stand: 16. Aug 2018, URL: `https://fontawesome.com/icons/envelope?style=solid`

A Brief Look

DOM Traversal and Manipulation

Get the `<button>` element with the class 'continue' and change its HTML to 'Next Step...'

```
$( "button.continue" ).html( "Next Step..." )
```

Event Handling

Show the `#banner-message` element that is hidden with `display:none` in its CSS when any button in `#button-container` is clicked.

```
var hiddenBox = $( "#banner-message" );
$( "#button-container button" ).on( "click", function( event ) {
  hiddenBox.show();
});
```

Ajax

Call a local script on the server `/api/getWeather` with the query parameter `zipcode=97201` and replace the element `#weather-temp`'s html with the returned text.

```
$.ajax({
  url: "/api/getWeather",
  data: {
    zipcode: 97201
  },
  success: function( result ) {
    $( "#weather-temp" ).html( "<strong>" + result + "</strong> degrees" );
  }
});
```

Abbildung 3: Ein Beispiel für die Anwendung von jQuery.
[6]

Zu guter Letzt fanden wir die MMenu Bibliothek im Internet, die das Erstellen von App-ähnlichen Menüs und Oberflächen sehr einfach macht. Einerseits benutzen wir MMenu für die Erstellung des Menüs für unsere Startseite. Andererseits benutzen wir die Bibliothek, um eine App-ähnliche Struktur für das Spiel zu erstellen.

[6] jQuery: Startseite, Stand: 16. Aug 2018, URL: `https://jquery.com`

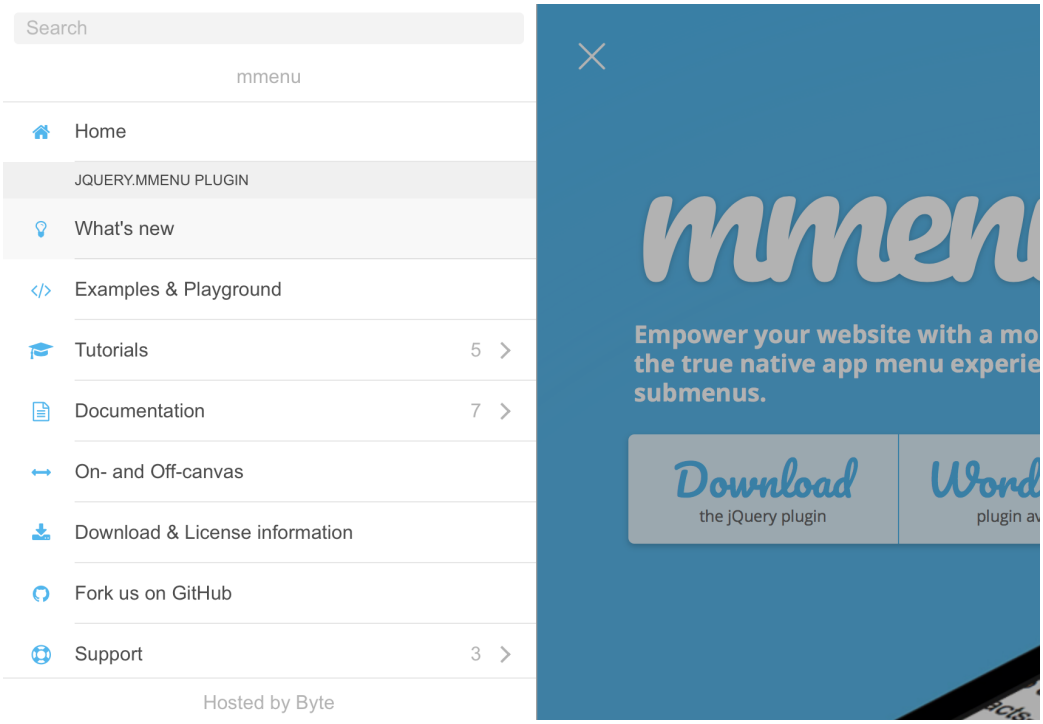

Abbildung 4: Ein Beispiel für die Anwendung von MMenu.
[7]

5.1 Startseite

Die Startseite ist die erste Seite, die ein Besucher sieht. Hier kann man natürlich per Schaltfläche ein neues Spiel erstellen oder einem vorhandenen Spiel beitreten. Hier war es uns besonders wichtig, dass die Funktionsweise zum Erstellen und Beitreten auf den ersten Blick ersichtlich ist. Hierzu verwendeten wir eine Schaltfläche und ein Eingabefeld. Ist das Eingabefeld leer, so steht "Neues Spiel" auf der Schaltfläche. Sobald jemand einen Spielcode eingibt, ändert sich der Text auf der Schaltfläche zu "Spiel beitreten". Zusätzlich steht noch "Spielcode Eingeben" auf dem Eingabefeld, damit dem Nutzer sofort klar wird, wozu dieses Feld benutzt wird.

Außerdem gibt es Seiten, auf denen wir die Spielregeln erklären und natürlich ein Impressum mit der Datenschutzerklärung. Zudem gibt es eine Seite, auf der wir uns persönlich vorstellen.

[7] MMenu: Startseite, Stand: 16. Aug 2018, URL: `http://mmenu.frebsite.nl`

5.2 Einstellungen

Auf der Einstellungsseite befindet sich die Steuerung des Spiels. Ganz oben kann man den Einladecode erkennen, der von anderen Spielern benötigt wird, um dem Spiel beizutreten. Darunter befindet sich eine Schaltfläche, um das Spiel zu verlassen. Hier können auch die Anzahl der Spezialrollen, sowie der Rundentimer eingestellt werden. Der Admin kann auf dieser Seite aber auch Spieler aus dem Spiel kicken.

Diese Seite ist sehr wichtig, um dem Host mehr Kontrolle über das Spielgeschehen zu geben, gegebenenfalls Leute rauszuwerfen und natürlich, um die Parameter einzustellen.

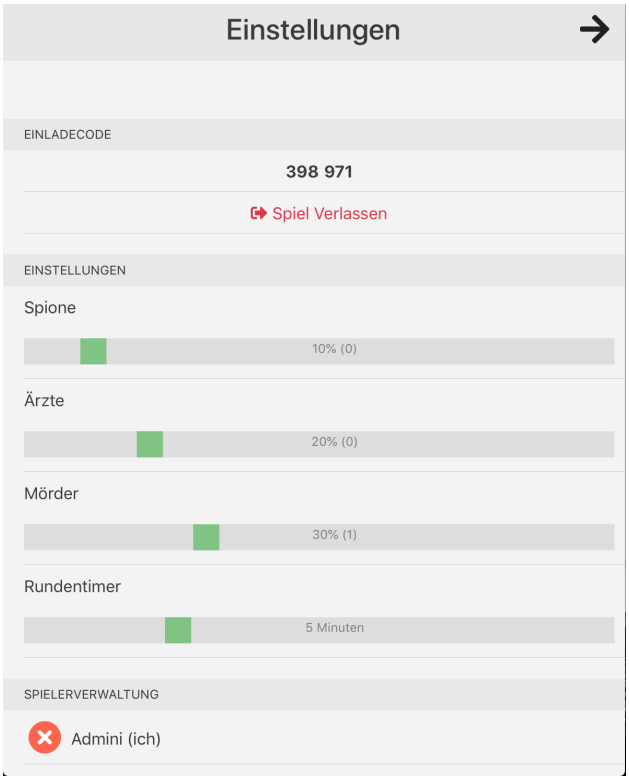

Abbildung 5: Die verschiedenen Einstellungsmöglichkeiten

5.3 Dashboard

Das Dashboard ist das Herzstück des Spiels. Ganz oben sieht man seinen Spielernamen, die verbleibende Rundenzeit und seine eigene Rolle. Darunter befindet sich ein Bereich, auf dem verschiedene Meldungen und Warnungen angezeigt werden können. Zuletzt sieht man die Liste der Mitspieler. Hier

kann man seine Stimme als Bürger oder die spezielle Stimme seiner Rolle abgeben. Auch kann hier per Knopfdruck auf einen privaten Chat gewechselt werden.

Mit der Schaltfläche ganz links kann der Spion Rollen von ausgespähten Mitspielern sehen. Auch der Mörder kann sehen, wer die anderen Mörder sind.

Tote Mitspieler sind grau hinterlegt und haben ein Totenkreuz neben ihrem Namen. Ist man selbst gestorben, so wird ein entsprechender Hinweis angezeigt. Ebenfalls erscheint ein Totenkreuz neben dem eigenen Namen.

Normalerweise lässt sich die eigene Rolle nicht sofort erkennen. Somit ist es auch nicht schlimm, wenn einmal ein Mitspieler auf den fremden Bildschirm schielt. Um die eigene Rolle anzuzeigen und um für seine Rolle die Spezialabstimmung zu tätigen, kann man natürlich auf die "Rollen"-Schaltfläche drücken. Damit werden alle speziellen Schaltflächen und Infos für einen kurzen Augenblick eingeblendet.

Abbildung 6: Das Dashboard im Grundzustand. Unten sieht man seine Mitspieler. Oben kann man seine eigene Rolle und die verbleibende Rundenzeit erkennen.

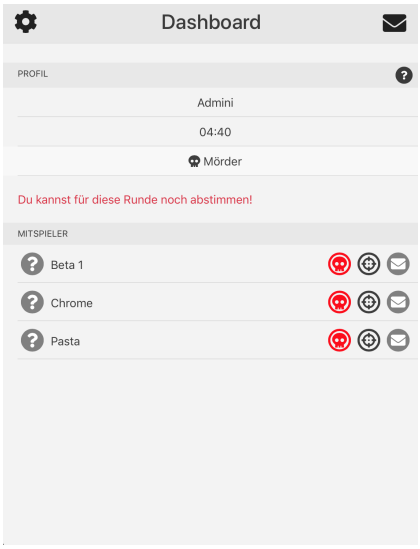

Abbildung 7: Beim Klick auf ”Rolle” erscheinen Interaktionsmöglichkeiten je nach eigener Rolle. Als Mörder kann man nun einen Spieler auswählen, der beim Rundenende getötet werden soll.

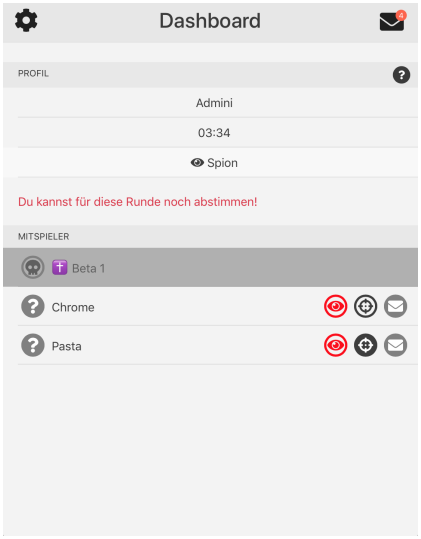

Abbildung 8: Als Spion hat man die Möglichkeit, seine Mitspieler auszuspähen. Nach Ende der Runde erfährt man dann die Rolle des ausgewählten Mitspielers durch das entsprechende Icon links vom Spielernamen. In diesem Fall war der Mitspieler ein Mörder.

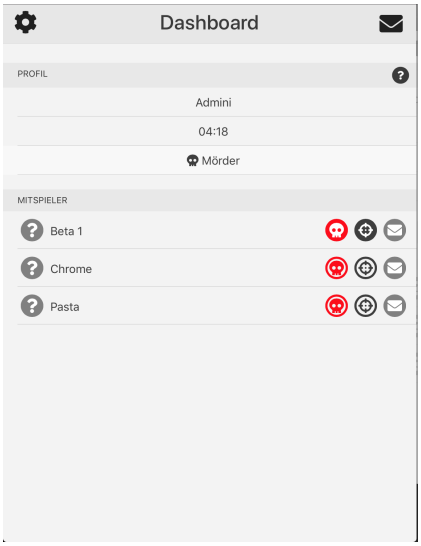

Abbildung 9: Getätigte Abstimmungen werden durch ausgefüllte Buttons gekennzeichnet. Die Meldung ”Du kannst noch abstimmen” verschwindet.

5.4 Messages

Gelangt man vom Dashboard auf die Messages-Seite, erhält man eine Übersicht aller Chats, in denen man sich befindet. Die Chatbuttons sind aufgelistet und leiten den Spieler beim Anklicken zum gewünschten Chatfenster weiter.

Die Reihenfolge der Auflistung entspricht der Aktualität des jeweiligen Chats. Wurde ein Chat neu erstellt oder bekommt man eine neue Nachricht, so rutscht dieser Chat nach oben in die Liste. Damit der Benutzer sieht, wie viele Nachrichten er in welchem Chat noch nicht gelesen hat, wird auf jedem Chatbutton die Anzahl der noch ungelesenen Nachrichten dargestellt.

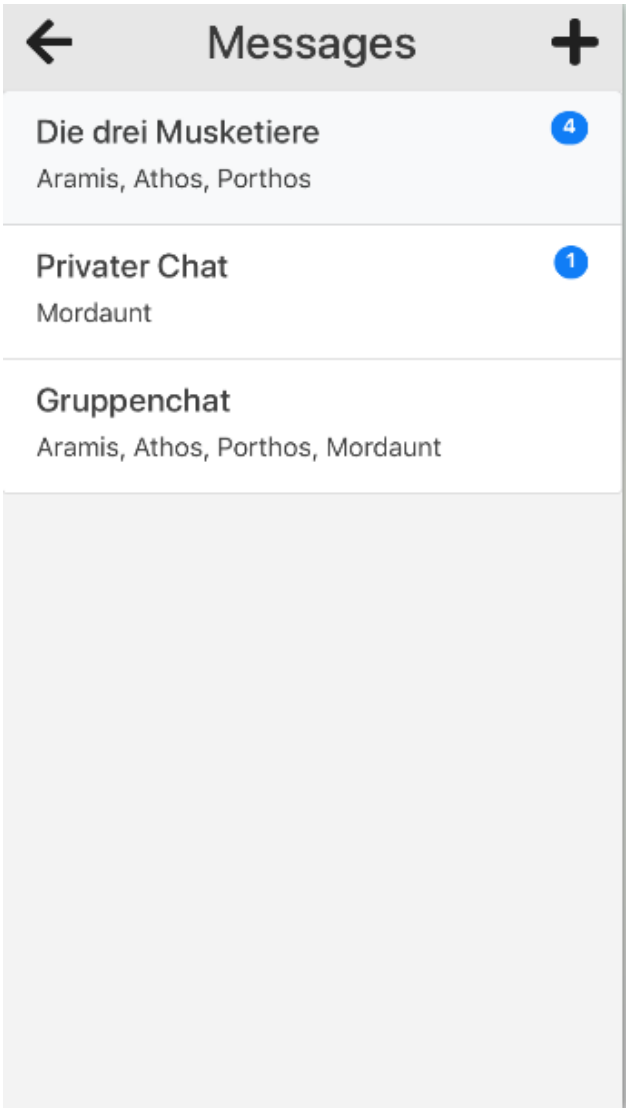

Abbildung 10: Das Messages-Fenster. Der aktuellste Chat steht ganz oben in der Liste und die Anzahl der noch ungelesenen Nachrichten wird rechts in einer blauen Pille dargestellt.

Möchte der Nutzer einen neuen privaten Chat oder einen weiteren Gruppenchat erstellen, drückt er beinahe selbsterklärend auf das Plus in der oberen rechten Ecke im Messages-Fenster.

So öffnet sich ein Eingabefenster, in dem der Nutzer einen Namen für den Chat eingeben kann und in einer Liste auswählen kann, welcher Mitspieler dem neuen Chat hinzugefügt werden soll.

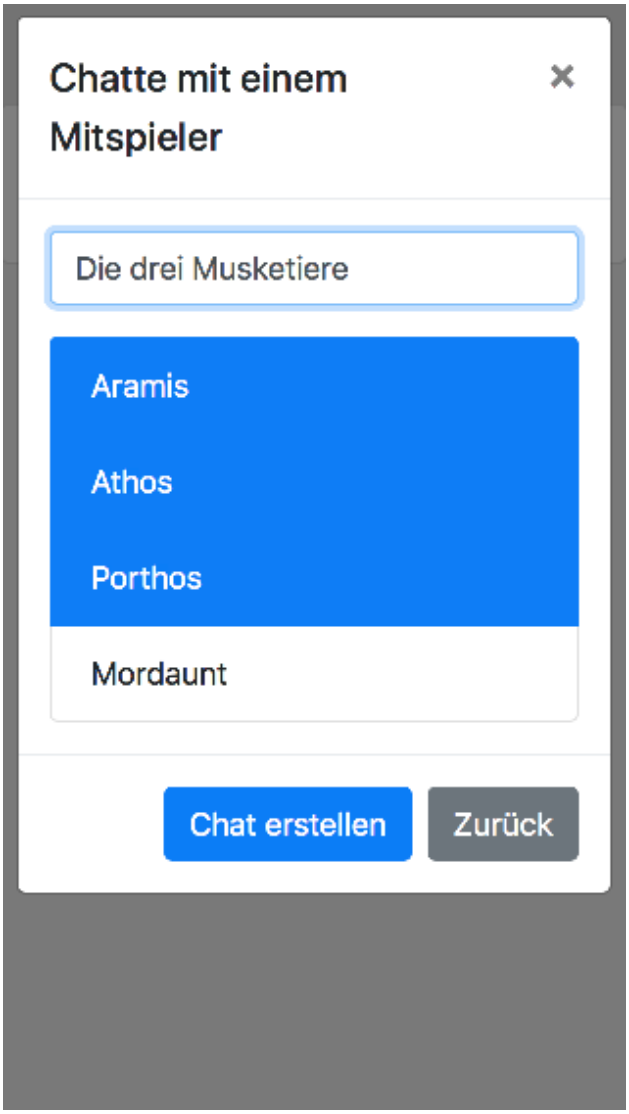

Abbildung 11: Das Chat-erstellen Fenster. Spielernamen lassen sich an- und abwählen, um sie dem Chat hinzuzufügen.

5.5 Chat

Wählt man auf der Messages-Seite einen Chat aus, wird man auf die Nachrichtenseite weitergeleitet. Hier werden alle Nachrichten des ausgewählten Chats gezeichnet. Um eine Nachricht zu versenden, gibt man sie im Textfeld ein und drückt anschließend auf den ”Senden” Button.

Eigene Nachrichten werden in einer blauen Blase dargestellt, während Nachrichten von Mitspielern grau unterlegt sind und der Spielername dabei

steht. Mit dem Pfeil in der oberen linken Ecke gelangt man zurück zum Messages-Fenster.

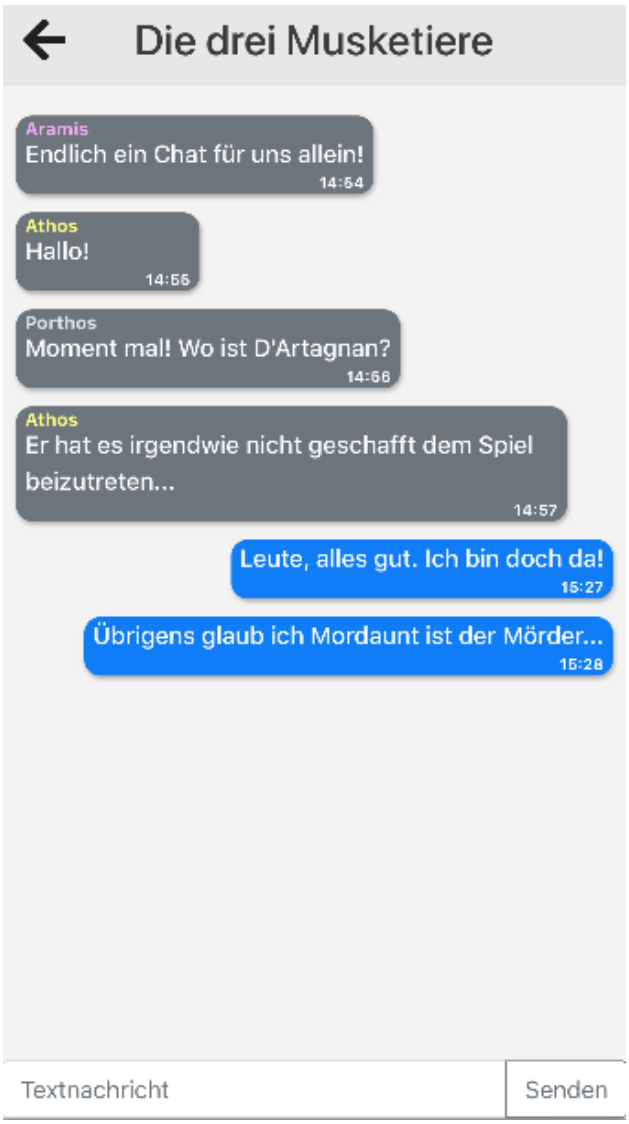

Abbildung 12: Der Chat. Verschiedene Spieler erhalten zufällige Farben um die visuelle Unterscheidung zu ermöglichen.

5.6 Javascript Dateien und Funktionen

5.6.1 game.js

In dieser Datei befindet sich die Game Klasse. Diese Klasse wird hauptsächlich für die Steuerung des Spielablaufes verwendet und stellt die Datenübertragung vom und zum Server sicher. Zu Beginn wird ein Intervall eingerichtet, dass jede Sekunde einen Pullrequest an den Server schickt. Die ankommenden Daten werden anschließend verarbeitet und die notwendigen Funktionen zum Rendern der Informationen werden ausgeführt, die sich in den weiteren Dateien befinden.

Ebenfalls werden in der Game Klasse wichtige Daten gespeichert. Hier finden sich Arrays, in denen die Spieler oder die Chats gespeichert werden und viele Variablen, die den Spielstatus und die Einstellungen speichern. Jede Javascript Funktion kann auf diese Klasse und ihre Variablen zugreifen. Damit stellt sie die wichtigste Klasse im Spiel dar.

```
function Game()
{
  this.id;
  this.player = [];
  this.player_self;
  this.is_started;
  this.code;
  this.spione;
  this.aerzte;
  this.moerder;

  this.chats = [];

  this.votes = [];
  this.role_votes = [];

  this.lastupdate;
  this.interval_handler;

// [...]
```

Das Speichern der Spieler und Chats ist wichtig, da vom Server immer nur die neusten Spieler und Informationen übertragen werden. Tritt beispielsweise ein neuer Spieler bei, bekommen alle bisherigen Spieler beim nächsten Pullrequest nur den neuesten Spieler als Datenpaket gesendet. Es wird dann anhand einer ID überprüft, ob dieser Spieler im Datenpaket bereits im Array

GAME.player vorhanden ist. Ist das nicht der Fall, wird dem Array ein neues Element hinzugefügt (wie in unserem Beispiel, dem neu beigetretenen Spieler). Andernfalls wird das alte Element durch das neue ersetzt (dies kommt vor, wenn ein Spieler z.B. seinen Namen ändert). Diese Überprüfung wird für Spieler sowie für Chats gleichermaßen durchgeführt.

Für den Fall, dass ein Spieler das Spiel verlässt, kommen wir jedoch mit diesem Aufbau nicht weit. Über diesen Spieler würden zwar keine weiteren Informationen vom Server an die Spieler gesendet werden, jedoch befände er sich weiterhin im GAME.player Array aller Browser. Um dieses Problem zu lösen, schickt der Server unter gewissen Umständen eine komplette Liste aller aktuellen Spieler-IDs an alle Spieler. Damit ist es möglich, alte Spieler, die bereits das Spiel verlassen haben, aus dem GAME.player Array zu entfernen. Diese komplette Liste wird jedes Mal geschickt, wenn sich die Anzahl der Spieler oder andere Variablen des Spiels und der Datenbanktabelle 'game' ändern. Oder anders gesagt: wenn die Datenbankspalte 'modified_at' der Tabelle 'game' neuer ist als der Zeitpunkt des letzten Pullrequests.

Im folgenden Beispiel wird eine komplette Spielerliste verarbeitet. Zunächst bekommen alle lokal bereits existierenden Spieler die Variable $updated = false$. Sollte ein Spieler aus der Liste auch lokal existieren, so wird $updated = true$ gesetzt. Neue Spieler werden mit Namen und ID hinzugefügt. Am Ende werden alle Spieler, bei denen immer noch $updated = false$ ist, gelöscht.

```
/**
 * takes the player list from the game
 * checks if player left the game and removes this player from this.player
 * also sees if a new player joined the game and adds it to this.player
 * @param  {array} gplist the player list from data.game.player
 * @return {void}
 */
this.parseGamePlayerList = function(gplist)
{
  //marks every existing player as outdated
  for (var j=0; j < this.player.length; j++) {
    this.player[j].updated = false;
  }

  //goes through every received player
  for(var i=0; i < gplist.length; i++)
  {
    //searches for existing player
    //marks updated to true - player still exists
    var playerfound = false
```

```
    for (var j=0; j < this.player.length; j++) {
      if (this.player[j].id == gplist[i].id) {
        this.player[j].updated = true;
        playerfound = true;
      }
    }

    //player is new - add it
    if (!playerfound) {
      var ni = this.player.length;
      //console.log("Add Player" + ni);
      this.player[ni] = Object.create(gplist[i]);
      this.player[ni].updated = true;
      this.player[ni].color = getRandColor(400);
      //console.log(this.player[ni]);
    }

    //playerfound = true
    //--> player already exists
    //--> and the player will be updated later
  }

  //deletes outdated player
  for (var j=this.player.length-1; j>=0; j--) {
    if (!this.player[j].updated) {
      //console.log("Delete Player" + j);
      //console.log(this.player[j]);
      this.player.splice(j, 1);
    }
  }
}
```

In folgendem Beispiel kann man sehen, wie die alten Spielerdaten durch neue (gerade vom Server empfangene Daten) ersetzt werden:

```
/**
 * updates name, id, alive and is_admin from a player
 * takes only player who have been recently modified
 * @param  {array} plist the data.player array
 * @return {void}
 */
this.updatePlayerInfo = function(plist)
{

  //loops through every updated player
  for(var i=0; i < plist.length; i++)
  {
    //loops through existing player
    for(var j=0; j < this.player.length; j++)
    {
      //player found (i and j pointing to the same player id)
      if (plist[i].id == this.player[j].id)
      {

        //update information
        this.player[j].name = plist[i].name;
        this.player[j].is_admin = plist[i].is_admin;
        this.player[j].alive = plist[i].alive;

        //player is self
        if (this.player_self.id == this.player[j].id)
        {
          //take data but keep assigned role
          var ownRole = this.player_self.role;
          this.player_self = Object.create(this.player[j]);
          this.player_self.role = ownRole;

          //check for player name
          if (this.player_self.name.length <= 0)
          {
            this.askForName();
          }
        }

      }
```

```
    }
  }
}
```

Nachdem neue Daten empfangen und in den dafür vorgesehenen Variablen, Arrays und Objekten gespeichert wurden, werden die jeweiligen Funktionen in den einzelnen Javascript Dateien ausgeführt, die für die Darstellung verantwortlich sind.

5.6.2 game_settings.js

Diese Datei steuert das Verhalten und die Darstellung auf der Einstellungsseite. Es handelt sich hierbei nicht um eine separate Klasse, jedoch befinden sich hier viele Funktionen, die dazu verwendet werden die Informationen vom Server in Form von Schiebereglern, Tabellen und Listen darzustellen. Sollte der Benutzer die Schieberegler verstellen, werden die neuen Informationen an den Server zur Übernahme gesendet. So können Einstellungen vorgenommen werden, die später beim Starten des Spiels berücksichtigt werden.

5.6.3 game_dashboard.js

Diese Datei steuert das Verhalten und die Darstellung auf dem Dashboard. Es werden persönliche Informationen sowie die Spielerliste vom Server geladen und anschließend auf dem Bildschirm dargestellt. Dafür gibt es eine Funktion, die den HTML Code für die Liste der Mitspieler generiert und anschließend an vorgegebener Stelle einfügt. Alle Eingaben des Benutzers wie zum Beispiel das Abstimmen werden ebenfalls verarbeitet und an den Server weitergeleitet. Es befinden sich zusätzlich einige Funktionen zur Steuerung der Animationen des Dashboards in dieser Datei. Genau wie die vorherige Datei handelt es sich um eine Funktionssammlung.

5.6.4 game_messages.js

Diese Datei steuert das Verhalten und die Darstellung der Chat Buttons im Messenger Fenster. Hier kann der Benutzer Gruppen- oder private Chats mit anderen Nutzern seiner Wahl erstellen und durch Anwählen des Chat Buttons in den gewünschten Chat gelangen. Für die Erstellung eines neuen Chats klickt der Nutzer das Plus in der oberen rechten Ecke des Bildschirms an, was zunächst eine Messagebox mit einem Eingabefeld für den gewünschten Chatnamen und der Spielerliste zum Auswählen aufruft.

```
//This function toggles the playername button
function clickOnNameForChat(data)
{
  var pid = data.dataset.pid;

  $("#messageBox .modal-body .list-group .
  list-group-item[data-pid=" + pid + "]").toggleClass("active");
}

//This function opens the messagebox when clicking on '+'
$("#messages #menu-icon-right").click(function() {
  //Code the Inside of the messageBox with html for selecting a player
  //to chat with
```

Die Mitspielerliste, aus der der Nutzer die Teilnehmer auswählen kann, wird in einer For-Schleife generiert und die Player-ID dem HTML-Element dieses Buttons angehängt.

```
  //Each player namefield is being created in a for-loop which
  //attaches the player-id to the corresponding button.
  var spielerliste = '<div class="input-group mb-3">';
  spielerliste += '<input id="chatNameInputForChat" type="text"
  class="form-control" placeholder="Wie soll der Chat heissen?"
  aria-label="Username" aria-describedby="basic-addon1">';
  spielerliste += '</div>';
  spielerliste += '<ul class="list-group">';
  for (var i = 0; i < GAME.player.length; i++)
  {
    var pname = 'Unbekannt';
    if (GAME.player[i].name.length > 0)
    {
      pname = GAME.player[i].name;
    }
    if (GAME.player[i].id == GAME.player_self.id)
    {
      continue;
    }
    spielerliste += '<li data-pid="' + GAME.player[i].id + '"
    class="list-group-item" onclick="clickOnNameForChat(this)">
    ' + pname + '</li>';
  }
  spielerliste += '</ul>';
```

Mit dieser Liste kann ein Popup für den Nutzer erstellt werden. Für das Popup verwenden wir das "Modal" Objekt von Bootstrap[8]. Für die Benutzung im Messages-Fenster wird das Modal entsprechend mit der Spielerliste in HTML gefüllt.

```
  //Create the Message Box for the user
  $("#messageBox").modal();
  $("#messageBox␣.modal-title").html("Chatte␣mit␣einem␣Mitspieler");
  $("#messageBox␣.modal-footer␣.btn.btn-primary").html("Chat␣erstellen");
  $("#messageBox␣.modal-footer␣.btn.btn-secondary").html("Zurueck");
  $("#messageBox␣.modal-body").html(spielerliste);
```

Der Nutzer wählt, wie in Abbildung 11 dargestellt, die Spieler aus, die dem Chat beitreten sollen. Klickt der Nutzer dann auf "Chat erstellen", werden die Player ID's der aktiven Buttons in einer Array gespeichert und zusammen mit einem "request" und dem gewünschten Chattitel an den Server geschickt.

```
/**
 * When opening the messageBox in the messenger this
 * msgbox_callback function will be activated.
 * By pressing the 'Chat erstellen'-button this function will
 * create an array containing all player id's attached to
 * the activated buttons and send it to the server.
 */
msgbox_callback = function()
{
  var player_array = [];
  var chatTitleInput = $("#messageBox␣#chatNameInputForChat").val();
    if (chatTitleInput < 1) {
      chatTitleInput = 'Privater␣Chat';
      }

  $("#messageBox␣.modal-body␣.list-group␣.active").each(function()
   {
     var pid = $(this).attr('data-pid');
     player_array.push(pid);
     console.log(pid);
    });
```

[8]Bootstrap: Modal, Stand: 28. Aug. 2018, URL: https://getbootstrap.com/docs/4.0/components/modal/

Der ”request” ”createPrivateChat” erlaubt dabei dem Spieler, anders als beim Gruppenchat oder dem ”Mörder”-Chat, die Teilnehmer und den Chatnamen selbst zu wählen. Ein entsprechender Eintrag soll in der Datenbank gemacht werden, anschließend kann das Modal wieder verschwinden.

```
 /**
   * Request to server for creating a chat with
   * chosen players and chosen chatTitle
   */
  webhookCall("./webhook.php",
    {request: 'createPrivateChat', player_ids: player_array,
    chatTitle: chatTitleInput},
    function(data) {},
    function(data) {
      alert(data.error);
    });

  $("#messageBox").modal('hide'); //Close msgBox after creating the chat

} //Closes the msgbox_callback function
```

Damit für den Nutzer der aktuellste Chat immer ganz oben in der Liste erscheint, wird die Zeit der letzten Nachricht in einer Variable innerhalb des Chat Objekts gespeichert. Ein neuer Chat, der noch keine Nachrichten beinhaltet, speichert in diese Variable den Zeitpunkt der Erstellung des Chats. So können die Chats nach dieser Variablen sortiert werden.

```
var chatDataSort = chatData.sort(function (a, b) {
    if (a.last_message_time > b.last_message_time) {
      return -1;
    }
    if (a.last_message_time < b.last_message_time) {
      return 1;
    }
    // a has to be equal b
    return 0;
  });
```

Um in das richtige Chatfenster zu gelangen, enthält das HTML-Element jedes Chatbuttons die Chat ID und den Chatnamen als Attribut. Die Funktion ”pickThisChatWindow()”, die beim Onclick-Event ausgeführt wird, greift darauf zurück und sorgt in ”chat_bubbles.js” dafür, dass der richtige Chat dargestellt wird.

```
function renderChatButton(chatTitle, chatID, chatMember_string,
badgePill_num)
{
  if (badgePill_num > 0){
    var pillBadge = '<h5 class="badge badge-primary badge-pill">
    ' + badgePill_num + '</h5> </div>';
  }
  else {
    var pillBadge = '</div>';
  }

  var chatButton = '<a href=#chat-window
  onclick="pickThisChatWindow(this)" data-title ="'+ chatTitle +'"
  data-cid="'+ chatID +'"
  class="list-group-item list-group-item-action
  flex-column align-items-start">';
  chatButton += '<div class="d-flex w-100 justify-content-between">';
  chatButton += '<h5 class="mb-1">' + chatTitle + '</h5>';
  chatButton += pillBadge;
  chatButton += '<p class="mb-1">' + chatMember_string + '</p>';
  chatButton += '</a>';

  return chatButton;
}
```

Zusätzlich besitzt jeder Chatbutton auf der rechten Seite eine kleine Anzeige, die die Anzahl der noch ungelesenen Nachrichten in diesem Chat verrät. Dazu wird im Array "GAME.chats" die Zeit, in der man das Chatfenster verlassen hat, in eine Variable gespeichert und für jeden Chat in einer For-Schleife mit dem Erstellungszeitpunkt der Nachrichten verglichen.

```
function pillNumCounter(chatData)
{
  var pillNum = 0;

  for (var i = 0; i < chatData.messages.length; i++)
  {
    if (chatData.last_time_checked > chatData.messages[i].time){
      continue;
    }
    pillNum = pillNum + 1;
  }

  return pillNum;
}
```

Ein weiterer Zähler bestimmt auf die gleiche Weise den Wert aller noch ungelesenen Nachrichten, um ihn im Dashboard darstellen zu können. Da man im Dashboard auf der Spielerliste auch erkennen können soll, welcher Spieler einem wie viele neue Nachrichten geschickt hat, bestimmt ein weiterer Zähler die ungelesenen Nachrichten bezogen auf jeden Mitspieler.

Drückt man im Dashboard auf der Spielerliste auf den Briefumschlag neben einem Spielernamen, überprüft die Funktion "createPrivateChatFromDashboard" zunächst, ob es bereits einen privaten Chat mit diesem Spieler gibt. Bedingung für die Suchfunktion ist, dass der Chat zwei Teilnehmer hat.

```
//This function will be activated from the Dashboard
function createPrivateChatFromDashboard(player_id)
{
  var exists = false;
  var chat_id;
  var chat_title;
  var run_again_counter = 0;
//Look if there is already a chat with this player
  for (var i = 0; i < GAME.chats.length; i++)
  {
    //The chat we are looking for is a private chat
    // and should not have more than 2 members
    if (GAME.chats[i].chat_member.length > 2) continue;

    for (var x = 0; x < GAME.chats[i].chat_member.length; x++)
    {
      //Does the chat already exist set variable 'exists'=true
      //and continue to the next step
      if (player_id == GAME.chats[i].chat_member[x]) {
        exists = true;
        chat_id = GAME.chats[i].id;
        chat_title = GAME.chats[i].name;

        break;
      }
    }//for x
    if (exists){
      break;
    }
  }//for i
```

Existiert der Chat bereits, sind die Chatdaten vorhanden und der Nutzer kann direkt zum Chat weitergeleitet werden.

```
  //Go direct to chatwindow if chat exists
  if (exists)
  {
    var data = {dataset: {cid: chat_id, title: chat_title}};
    pickThisChatWindow(data);
    var api = $("#menu").data( "mmenu" );
    api.openPanel( $("#chat-window") );
  }
```

Existiert der Chat jedoch noch nicht, wird der Request zum Erstellen des Chats mithilfe der ”webhookCall” Funktion an den Server geschickt.

```
  //If not we first need to create a new private chat with webhookCall
  else
  {
    webhookCall("./webhook.php",
      {request: 'createPrivateChat',
      player_ids: [player_id], chatTitle: 'Privater Chat'},
      function(data) {},
      function(data) {
        alert(data.error);
      });
      //This variable will be needed in game.js
      joinNextCreatedChat = true;
          //In the mean time user will be directed to messages window
      var api = $("#menu").data( "mmenu" );
      api.openPanel( $("#messages") );
  }
}
```

Weil es einen kurzen Moment braucht, bis der Chat in der Datenbank eingetragen wurde und diese Information erst beim nächsten Pull beim Nutzer ankommt, kann der Nutzer hier nicht sofort zum Chat weitergeleitet werden. Aus diesem Grund wird die Variable ”joinNextCreatedChat” gleich ”true” gesetzt, was in ”game.js” erst beim nächsten Pull das Öffnen des Chatfensters auslöst, wenn die Chatdaten vom Server angekommen sind. Damit der unwissende Nutzer nicht das Gefühl hat, seine Eingabe sei ignoriert worden und denkt, er müsse deswegen den Button mehrmals drücken, wird er zunächst zum Messages-Fenster weitergeleitet solange der Chat noch nicht erstellt wurde. Das soll ihm das Gefühl vermitteln, es passiere etwas.

5.6.5 chat_bubbles.js

Diese Datei enthält die Funktionen, um den Nutzer in den richtigen Chat weiterzuleiten und die Nachrichten aus diesem darzustellen. Eine Überschrift und eine Chat-ID werden benötigt, um das Chatfenster und die Sprechblasen korrekt zu zeichnen. Beide Attribute werden dem HTML-Element des Chat-

buttons im Messenger Fenster angeheftet. Wählt man einen Chat im Messenger Fenster aus, setzt die Funktion "pickThisChatWindow" den angehefteten Chattitel als Überschrift in die Kopfzeile und lässt die Funktion "renderChatWindowPack" die Nachrichten in der korrekten Reihenfolge zeichnen. Nun muss noch sichergestellt werden, dass die eigenen Nachrichten vom Nutzer den richtigen Chat erreichen. Dafür wird die Chat-ID dem HTML-Element des "Senden"-Buttons angeheftet. Um direkt die neueste Nachricht vor sich zu haben, wird die Seite für den Nutzer so weit es geht nach unten gescrollt.

```
function pickThisChatWindow(data)
{
  var chatID = data.dataset.cid;
  var chatTitle = data.dataset.title;
  //add this ChatID to the HTML Element of the send-button
  $("#chat-send-button").attr("data-cid", chatID);
  //now render the chatTitle
  var titleHtml = '<h1>'+ chatTitle +'</h1>';
  $("#chatTitlePageName").html(titleHtml);
 //now render all messages
  renderChatWindowPack(GAME.chats);
  setTimeout(scrollToBottom, 50);
}
```

Um die Nachrichten korrekt darzustellen, sucht die Funktion "renderChatWindowPack" im Javascript Array "GAME.chats" nach der gewünschten Chat-ID.

```
function renderChatWindowPack(chats)
{
  var bubbleString = "";
  var chatID = $("#chat-send-button").attr('data-cid');

  if (chatID == 0){
    return;}

  var messageString = [];
  //Select complete message-data from this chat
  for (var x = 0; x < chats.length; x++)
  {
    if (chats[x].id == chatID)
    {
      messageString = chats[x].messages;
```

```
      break;
    }

  }
//String containing chatbubble HTML elements
  bubbleString += renderMessages(messageString);

  //render all chat bubbles
  $("#chat-text-container").html(bubbleString);
}
```

Alle Nachrichten-Daten, die sich in diesem Chat befinden, werden an die Funktion ”renderMessages” weitergereicht, die für jede einzelne Nachricht wiederum die Funktion ”renderChatbubble” aufruft und anschließend jedes HTML Element der Nachrichtenblase hintereinander aufreiht. Die Funktion ”renderChatbubble” erstellt dabei das HTML Element einer einzigen Nachrichtenblase. Damit die Nachrichtenblase das Design eines seriösen Messengers bekommt, entschieden wir uns, die Uhrzeit mit abzubilden. Dazu muss zunächst die Unixzeit in die klassische Schreibweise umgewandelt werden.

```
function renderChatbubble(singleMessageData)
{
  var date = new Date(singleMessageData.time * 1000);
  var hours = date.getHours();
  var minutes_pure = date.getMinutes();

  var minutes = minutes_pure;
  if (minutes_pure < 10)
  {
    var minutes = "0" + minutes_pure.toString();
  }

  var time = hours +':'+ minutes;
```

Die klassische Schreibweise mit Stunden und Minuten wird in der Variablen ”time” gespeichert.

Als nächstes muss überprüft werden, ob der Nutzer selbst Urheber dieser Nachricht ist. Ist das der Fall, wird die Nachricht in einer blauen Blase auf der rechten Hälfte des Bildschirms gezeichnet.

```
  var chatBubble = "";
  var message = singleMessageData.message;
  var sender_id = singleMessageData.sender;

//creat HTML Element for users chatbubble
if (sender_id == GAME.player_self.id)
  {
      chatBubble += '<div class="chat-container-bubble-r">';
      chatBubble +=   '<div class="chat-bubble chat-bubble-blue">';
      chatBubble +=     '<span class="chat-bubble-text">'+ message
      +'</span>';
      chatBubble +=     '<span class="chat-bubble-time">'+ time
      +'</span>';
      chatBubble +=   '</div>';
      chatBubble += '</div>';
  }
```

Kommt die Nachricht dagegen von einem Mitspieler, erscheint der Spielername in seiner individuellen Farbe innerhalb der Blase und diese wird dann in der linken Hälfte des Bildschirms in grauer Farbe gezeichnet. Für den Fall, dass der Sendername leer ist weil z.B. ein Mitspieler das Spiel verlassen hat, heißt der Sender "Unbekannt".

```
else
  {
    var senderName = "Unbekannt";
    var senderColor = "#ffffff";

    //Search senderName and senderColor for sender chatbubble
    for (var i = 0; i < GAME.player.length; i++)
    {
      if (GAME.player[i].id == sender_id) {
        senderName = GAME.player[i].name;
        senderColor = GAME.player[i].color;
        break;
      }
    }
    //create HTML Element for sender chatbubble
    [...]
   }
```

Um eine Nachricht zu versenden, gibt der Nutzer, wie von anderen Messenger-Apps gewohnt, seine Nachricht in das Eingabefeld ein und drückt auf 'Senden'. Der "request" "sendMessage" wird nun mit der eingegebenen Nachricht und der Chat-ID in der "webhook.php" weiterbearbeitet, um in die Datenbank eingetragen zu werden.

```
function submitChatInputField()
{
//if input is empty, nothing shell be rendered
    var text_me = $("#chat-window␣#chat-input-field").val();
      if (text_me < 1) {return;}
//Chat ID is the 'data-cid' attribute of the send-button
    var chatID = $("#chat-send-button").attr('data-cid');
//enter new message in the database
    webhookCall("./webhook.php",
      {request: 'sendMessage', newMessage: text_me, chat_ID: chatID},
      function(data) {},
      function(data) {
        alert(data.error);
      });
  //reset value in the input-field
  $("#chat-window␣#chat-input-field").val([]);
}
```

Dargestellt wird die Nachricht jedoch erst, wenn sie mit dem neuen Pull im "GAME.chats" Array erscheint.

Verlässt man das Chatfenster über den Pfeil oben links im Fenster, wird die Zeit in einer Variablen innerhalb der "Chat"-Array gespeichert, um beim Zeichnen der Chatbuttons im Messenger-Fenster die Anzahl der ungelesenen Nachrichten zu aktualisieren. Es sollen nur Nachrichten gezählt werden, die nach dieser Zeit erstellt wurden. Weil die Zeitmarkierung der einzelnen Nachrichten beim Eintrag in die Datenbank erstellt werden und die Zeitmarkierung unserer Variablen der Serverzeit beim letzten Update entspricht, kann es passieren, dass diese um ein paar Zehntelsekunden hinterher hängt. Ist das der Fall, wird diese Nachricht vom Zähler als "neu" angesehen, wenn man schnell genug zurück zum Messenger-Fenster wechselt, nachdem eine neue Nachricht gerendert wurde. Um das zu vermeiden, machen wir die Zeitmarkierung unserer Variablen minimal größer, sodass sie in jedem Fall "neuer" ist als die Zeitmarkierung der letzten Nachricht.

```
//go back to messenger window
$("#chat-window #menu-icon-left").click(function() {

  var chatID = $("#chat-send-button").attr('data-cid');
//reset 'last_time_checked' to reset pillnumber
  for (var i = 0; i < GAME.chats.length; i++)
  {
    if (GAME.chats[i].id != chatID) {
      continue;
    }
    GAME.chats[i].last_time_checked = GAME.lastupdate + 1;
  }
  //render chatbuttons
  renderChatButtonWindow(GAME.chats);
  dashbRenderMessagePill();
  drawPlayerMessageCounter();
});
```

5.6.6 palermotools.js

Bei dieser Datei handelt es sich wieder um viele Hilfsfunktionen, die in anderen Dateien und Klassen benötigt werden. Unter anderem finden sich hier Funktionen zum Generieren einer zufälligen Farbe, Funktionen um Cookies zu lesen und zu schreiben, Zeiten in ein lesbares Format umzuwandeln und natürlich eine Funktion, um Daten an den Server zu schicken und bei einem Fehler auf bestimmte Art und Weise zu reagieren. Diese Funktion wird jedes Mal aufgerufen, wenn Datenbankeinträge von der Seite des Nutzers gemacht werden müssen.

```
function webhookCall(url, senddata, success, failure)
{
  //ajax request
  var ajaxh = $.post(url, senddata)
  .done(function (data)
  {
    if (data.status == 'success') {
      success(data);
    } else {
      failure(data);
    }
  })
```

```
  .fail(function (data)
  {
    alert("Verbindungsfehler!␣Bitte␣versuche␣es␣spaeter␣erneut!");
    console.log(data);
    failure();
  });
}
```

5.6.7 site.js

In dieser Datei befinden sich die Skripte, die auf der Startseite benötigt werden. Hier handelt es sich hauptsächlich um Funktionen, die das Formular und die Schaltfläche auf der Startseite steuern. Zuletzt haben wir dort einen Sicherungsmechanismus eingefügt, der den Benutzer, der eigentlich aktiv im Spiel eingeloggt ist, wieder zurück leitet, für den Fall, dass er aus Versehen auf die Zurück-Schaltfläche seines Handys gekommen ist. Unsere Tests haben ergeben, dass das versehentliche Tippen auf die Zurück Taste gerade bei Android Nutzern häufiger vorkommt.

```
function timeoutFunction()
{
  if (getCookie('pcode'))
  {
    window.location.href = "./game.php";
  }
}
```

6 Server

Wie bereits im Kapitel Arbeitsumgebung erwähnt, benutzen wir den Apache Webserver, die MariaDB als SQL Datenbankserver und PHP als Skriptsprache, um den Server unseres Spiels zu programmieren. Der Server gilt als zentrales Gehirn des Spiels. Hier werden alle wichtigen Entscheidungen und Aktionen gefällt, überprüft und ausgeführt. Hier liegen auch die HTML und Javascript Dateien bereit für einen Webbrowser zum Herunterladen. Um diese beiden Aufgaben getrennt voneinander auszuführen, entschieden wir uns den Server funktional in zwei Dateien zu trennen: die index.php und die webhook.php.

Die "index.php" ist verantwortlich für das Laden und Darstellen des HTML, CSS und Javascript Codes. Dateien mit diesem Namen werden standardmäßig geladen, wenn ein Besucher nur die URL ohne Dateinamen eingibt (z. B.: https://palermo.mh.net).

Die "webhook.php" wird lediglich intern von Javascript aufgerufen. Zu dieser Datei können alle Anfragen, Aufträge und neuen Infos gesendet werden. Wird diese Datei beispielsweise mit dem POST-Parameter "request" = "pull" aufgerufen, bekommt man alle neuesten Informationen des aktuellen Spiels. Die webhook.php gibt eine Antwort im sogenannten JSON-Format heraus. JSON steht für "Javascript Object Notation" und kann, wie der Name bereits erahnen lässt, mit Javascript eingelesen werden, um ein Objekt zu erstellen. Bei diesem Objekt handelt es sich um eine beliebig konstruierbare Datenstruktur mit der sich viele verschiedene Daten einfach speichern und darauf zugreifen lassen. Diese Objekte können als Text im sogenannten JSON-Format einfach verschickt und auf der Gegenseite eingelesen werden.

6.1 Requests

Requests können vom Benutzer jederzeit an den Server über HTTP gesendet werden. Die Parameter müssen mit POST übermittelt werden. Der Parameter "request" enthält die Art des Requests. Ebenso können je nach Request zusätzliche Parameter erforderlich sein. Folgende Requests haben wir implementiert:

6.1.1 newgame

Der "newgame" Request erstellt ein neues Spiel und tritt automatisch dem erstellten Spiel bei.

6.1.2 joingame

Mit dem “joingame” Request kann einem bereits laufendem Spiel beigetreten werden. Es wird per Parameter der Einladecode benötigt, um das Spiel aus der Datenbank zu finden.

6.1.3 leavegame

Mit dem “leavegame” Request kann das Spiel verlassen werden. Der Spieler wird dem Spiel entfernt und alle Nachweise werden aus der Datenbank entfernt.

6.1.4 pull

Der Pullrequest. Hiermit werden neue Daten vom Server geladen. Über den Parameter “lastpull” wird der Zeitpunkt der letzten Abfrage mitgeteilt. Gleichzeitig gibt es in fast jeder Datenbanktabelle die Spalte ‘modified_at’, die den Zeitpunkt der letzten Änderung speichert. Somit muss der Server nur die neuesten Ereignisse mitschicken, indem er alle Daten aus der Datenbank holt, bei denen ‘modified_at’ größer gleich ‘lastpull’ ist.

6.1.5 vote

Mit diesem Request können Abstimmungen dem Server mitgeteilt werden.

6.1.6 kick

Mit dem “kick” Request kann der Host einen Spieler vom Spiel kicken.

6.1.7 savesettings

Mit dem “savesettings” Request können Einstellungen vom Host an den Server zum Speichern übertragen werden.

6.1.8 startgame

Mit diesem Request kann der Host das Spiel starten, nachdem alle Spieler beigetreten sind.

6.1.9 sendname

Mit diesem Request können Spieler ihre Namen festlegen. Es ist nicht möglich, seinen Namen nachträglich zu ändern.

6.1.10 createPrivateChat

Mit diesem Request kann ein privater Chat mit ausgewählten Mitspielern und einem selbst bestimmten Namen erstellt werden.

6.1.11 sendMessage

Mit diesem Request kann eine Nachricht innerhalb eines bestimmten Chats verschickt werden.

6.2 Wiedererkennung von Benutzern

Stellen wir uns einmal einen Benutzer vor, der einem neuen Spiel beigetreten ist. Nun lädt er die Seite neu. Wie erkennt der Server jetzt, um welchen Spieler es sich handelt, um die Benutzeroberfläche neu aufzubauen? Dazu muss der Server in irgendeiner Weise dem Besucher einen Spieler zuordnen können. Beispielsweise kann man die ID des Spielers in einem Cookie speichern und diesen beim erneuten Besuchen der Seite wieder auslesen. Grundsätzlich gibt es zum Speichern von benutzerspezifischen Daten zwei Möglichkeiten: Cookies und Sessions.

Sessions sind von kurzlebiger Dauer und Bestandteil von PHP. Beim ersten Besuch wird ein zufallsgenerierter Code als Cookie im Browser des Besuchers gespeichert. Mit diesem Code (der sogenannten Session ID) kann PHP den Benutzer eindeutig identifizieren. Auf dem Server lassen sich nun Daten in einer Art temporären Datei zu diesem Benutzer speichern. Die Benutzung von Sessions ist generell sehr einfach. Jedoch haben sie den Nachteil, dass die Daten meistens verloren gehen, sobald der Benutzer seinen Browser schließt und wieder öffnet.

Eine permanentere Art der Speicherung muss gefunden werden. Wir möchten, dass der Benutzer sich auch nach mehreren Tagen noch ins Spiel einloggen kann. Daher haben wir uns entschieden, eine ID des Spielers als Cookie zu speichern. Cookies haben jedoch den Nachteil, dass der Benutzer die gespeicherten Daten direkt einsehen und manipulieren kann. Würden wir lediglich die Spieler ID speichern, wäre der Benutzer in der Lage diese ID beliebig zu verändern und sich somit als anderer Benutzer auszugeben. Dieses Verfahren ist auch bekannt als Session Hijacking[9].

Um das Spiel abzusichern entschieden wir uns dazu, neben der ID jedem Benutzer einen eindeutigen zufällig generierten 18 stelligen Code zuzuweisen.

[9]Wikipedia: Session Hijacking, Stand: 05. Aug. 2018, URL: `https://en.wikipedia.org/wiki/Session_hijacking`

Diesen Code können wir mit gutem Gewissen in den Cookies speichern, denn er lässt sich deutlich schwerer erraten als eine fortlaufende ID.

```
/**
 * creates a new, empty player handle
 * @return Player returnes the new player
 */
public static function createNewPlayer() {
  $player = new Player();
  $player->code = self::getNewPlayerCode();
  return $player;
}


/**
 * Generates a new unused player code
 * @return string player code
 */
private static function getNewPlayerCode() {
  global $_CONFIG;

  $player_code = "";

  do {
    $player_code = Toolbelt::genCode($_CONFIG['PLAYERCODE_LENGTH']);
    $result = palermoDb::get()->query("SELECT␣id␣FROM␣player
␣␣␣␣WHERE␣code='$player_code'");
  } while($result->num_rows > 0);

  return $player_code;
}
```

Wir richteten zudem eine Weiterleitung von der Startseite zur Spielseite ein, falls der Benutzer bereits einen Player Code in seinen Cookies besitzt. Somit kann der Benutzer gleich weiter spielen.

```
//if player cookie available -> Redirect
if (array_key_exists("pcode", $_COOKIE)
&& !array_key_exists('p', $_GET)){
  header('location:␣./game.php');
  die();
```

```
}
```

6.3 PHP Klassen und Dateien

Der Quellcode des Servers ist auf viele verschiedene Klassen aufgeteilt. In diesem Kapitel möchten wir die PHP Klassen und Dateien vorstellen und ihre Funktionsweise und Aufgaben näher beschreiben.

6.3.1 webhook.php

Die webhook.php Datei wird immer aufgerufen, um Daten zwischen Benutzer und Server auszutauschen. Wie bereits erwähnt, können ihr verschiedene Anfragen (die sogenannten Requests) geschickt werden, um bestimmte Dinge zu tun. Die am häufigsten verwendete Anfrage ist der Pull Request. In dieser Datei werden auch alle weiteren Dateien des Servers eingebunden. Sie fungiert somit als zentrale Datei des Servers.

6.3.2 Game

Die Game Klasse steuert das Hauptspielgeschehen. Über diese Klasse kann man ein neues Spiel erstellen, ein Spiel löschen, Spieler hinzufügen und entfernen, Runden starten und beenden sowie viele andere Dinge. Hier wird auch überprüft, wann eine Runde zu Ende ist oder wann das Spiel vorbei ist und wer gewonnen hat.

Wird das Spiel erstellt, befindet es sich zunächst noch in einem Pausenmodus. Es können jetzt alle Spieler über den Einladecode beitreten und der Host kann Einstellungen zum Spiel anpassen. Sind alle bereit, kann der Host mit einem Klick auf die Schaltfläche auf dem Dashboard das Spiel starten. Solange das Spiel läuft, können keine Spieler mehr beitreten und die Einstellungen können nicht mehr geändert werden. Nun bekommt jeder Spieler eine Rolle zugewiesen und die Spieler können anfangen abzustimmen und zu diskutieren.

Die Runde kann auf zwei Arten beendet werden. Entweder ist die Zeit abgelaufen (sofern eine Zeitbegrenzung eingestellt wurde), oder alle Spieler haben abgestimmt. Beim Ende der Runde wird zunächst festgestellt, welchen Spieler die Bürger erhängen wollten. Dieser Spieler wird dann durch das Setzen von ‘is_alive’ auf null (0) getötet. Anschließend wird die Entscheidung der Mörder und Ärzte festgestellt. Sollten die Ärzte das Opfer der Mörder nicht gerettet haben, wird der betroffene Spieler ebenfalls getötet. Zuletzt werden den Spionen durch einen Eintrag in der ‘spion_data’ Tabelle die Rolle eines Mitspielers mitgeteilt. Damit beginnt dann die neue Runde.

Die Rundenzeit wird dazu zurückgesetzt und die Spieler können wieder abstimmen. Zusätzlich wird eine Public Message erstellt, um jeden Spieler über den Ausgang der Runde zu informieren.

```
private function endRound()
{
  // [...]
  $killedby_buerger = -1;
  $killedby_moerder = -1;

  //buerger vote
  $result_buerger = palermoDb::get()->query("SELECT␣vote_id,␣COUNT(*)
␣␣FROM␣vote␣WHERE␣game_id='" . $this->id . "'␣AND␣type=0
␣␣␣␣GROUP␣BY␣vote_id␣ORDER␣BY␣COUNT(*)␣DESC");

  if ($obj = $result_buerger->fetch_object())
  {
    $killedby_buerger = $obj->vote_id;
    Player::killPlayerById($killedby_buerger);
  }

  //get moerder votes
  $result_moerder = palermoDb::get()->query("SELECT␣vote_id,␣COUNT(*)
␣␣FROM␣vote␣WHERE␣game_id='" . $this->id . "'␣AND␣type=2
␣␣␣␣GROUP␣BY␣vote_id␣ORDER␣BY␣COUNT(*)␣DESC");

  if ($obj = $result_moerder->fetch_object())
  {
    $killedby_moerder = $obj->vote_id;
  }

  //moerder victim already killed by player?
  if ($killedby_moerder == $killedby_buerger)
  {
    $killedby_moerder = -3;
  }



  //get arzt votes - prevent moerder kill
  if ($killedby_moerder > 0)
  {
```

```
    $arzt_votes = palermoDb::get()->query("SELECT␣player_id,␣vote_id
␣␣␣␣␣␣FROM␣vote␣WHERE␣game_id='" . $this->id . "'␣AND␣type=3");

    while($obj = $arzt_votes->fetch_object())
    {
      if ($obj->vote_id == $killedby_moerder)
      {
        //prevent moerder kill
        $killedby_moerder = -2;
        break;
      }
    }
  }

  //kill moerder victim
  if ($killedby_moerder > 0)
  {
    Player::killPlayerById($killedby_moerder);
  }


  //get spion votes
  $spion_votes = palermoDb::get()->query("SELECT␣player_id,␣vote_id
␣␣␣␣FROM␣vote
␣␣␣␣WHERE␣game_id='" . $this->id . "'␣AND␣type=1");

  while($svote = $spion_votes->fetch_object())
  {
    $spied_person = Player::getFromId($svote->vote_id);
    palermoDb::get()->query("INSERT␣INTO␣spion_data
␣␣␣␣␣␣SET
␣␣␣␣␣␣game_id='" . $this->id . "',
␣␣␣␣␣␣player_id='" . $svote->player_id . "',
␣␣␣␣␣␣data_id='" . $spied_person->getId() . "',
␣␣␣␣␣␣role='" . $spied_person->getRole() . "',
␣␣␣␣␣␣created_at='" . time() . "'");
  }

  //delete all old votes
  palermoDb::get()->query("DELETE␣FROM␣vote
␣␣WHERE␣game_id='" . $this->id . "'");
```

```
  //prepare sendback message
  $myObj = new stdClass();
  $myObj->killedby_buerger = $killedby_buerger;
  $myObj->killedby_moerder = $killedby_moerder;
  $myObj->gameover = false;


  //check if game is over
  $gameover_var = $this->gameOverChecker();
  if ($gameover_var != 0)
  {
      //GAME IS OVER
      // [...]
      return;
  }


  //insert round end message
  PublicMessage::createMessage($this->id, "endround", $myObj);


  //free table
  $this->round_started_at = time();
  $this->modified_at = $this->round_started_at;
  palermoDb::get()->query("UPDATE␣game
␣␣␣␣SET␣round_end_block=0,␣round_started_at=
␣␣␣␣'" . $this->round_started_at . "',
␣␣␣␣modified_at='" . $this->modified_at . "'
␣␣␣␣WHERE␣id='" . $this->id . "'");

}
```

Sobald kein Mörder mehr vorhanden ist oder die Bürger in der Unterzahl sind, endet das Spiel an dieser Stelle. Die Spieler bekommen eine Information, welche Gruppe gewonnen hat. Außerdem wechselt das Spiel wieder in den Pausenmodus. Damit können wieder neue Spieler beitreten und Einstellungen geändert werden.

```
//check if game is over
$gameover_var = $this->gameOverChecker();
if ($gameover_var != 0)
{
  //GAME IS OVER
  $myObj->gameover = true;
  $myObj->gameover_var = $gameover_var;

  //insert game over round end message
  PublicMessage::createMessage($this->id, "endround", $myObj);

  //free table and set is_started to zero
  $this->modified_at = time();
  palermoDb::get()->query("UPDATE␣game
␣␣␣␣SET␣round_end_block=0,␣round_started_at='0',␣is_started='0',
␣␣␣␣modified_at='" . $this->modified_at . "'
␣␣␣␣WHERE␣id='" . $this->id . "'");

  return;
}
```

6.3.3 Player

Diese Klasse steuert Spieler. Man kann Spieler erstellen, löschen, Namen verändern und viele andere Dinge machen.

Unter anderem kann hier die Zeichenfolge generiert werden, die später als Cookie im Browser des Benutzers diesen als Spieler identifizieren kann. Dazu wird innerhalb einer Schleife eine Zeichenfolge generiert und geprüft, ob dieser Code bereits in der Datenbank vorhanden ist. Ist die Zeichenfolge einzigartig, so kann der Spieler mit diesem Code erstellt werden. Zuletzt wird dieser Code beim Benutzer in einem Cookie gespeichert.

```
/**
 * Generates a new unused player code
 * @return string player code
 */
private static function getNewPlayerCode() {
  global $_CONFIG;

  $player_code = "";

  do {
    $player_code = Toolbelt::genCode($_CONFIG['PLAYERCODE_LENGTH']);
    $result = palermoDb::get()->query("SELECT␣id␣FROM␣player␣WHERE
␣␣␣␣code='$player_code'");
  } while($result->num_rows > 0);

  return $player_code;
}
```

Um den Namen des Spielers verändern zu können, gibt es ebenfalls eine Methode in dieser Klasse. Zuerst wird überprüft, ob der Spieler bereits einen Namen gewählt hat, denn ein nachträgliches Ändern ist nicht zulässig und würde die anderen Mitspieler verwirren. Dann wird der Spielername auf seine Zusammensetzung untersucht. Zulässig sind nur alphanumerische Zeichen mit einer Länge von 2 bis 20 Stellen. Ab dem zweiten Zeichen sind Leerzeichen ebenfalls erlaubt. Erfüllt der Name alle Voraussetzungen, wird er in der Datenbank gespeichert und steht somit allen anderen Spielern bei der nächsten Datenabfrage ebenfalls zur Verfügung.

```
/**
 * sets the player name or throws an error on failure
 * @param string $name the new player name
 */
public function setName($name)
{
  $re = '/^(?=.{2,20}$)(?![\s])(?!.*[\s]{2})[a-zäöüA-ZÄÖÜ0-9\s]+$/m';

  if (strlen($this->name) > 0)
  {
    throw new Exception("Du kannst deinen Namen nicht mehr ändern!");
  }

  $array = array();
  if (!preg_match($re, $name, $array))
  {
    throw new Exception("Der Name hat nicht die erforderliche Struktur");
  }

  //just safety
  $name = htmlspecialchars($name);
  $time = time();

  $prep = palermoDb::get()->prepare("UPDATE player SET name=?,
  modified_at=? WHERE id=?");
  $prep->bind_param("sii", $name, $time, $this->id);
  $prep->execute();
  $prep->close();
}
```

6.3.4 Chat

Die Chat Klasse enthält alle Funktionen zum Erstellen der verschiedenen Chats sowie das Hinzufügen von Spielern in einen Chat. Beim Erstellen der Chats wird zwischen drei Arten von Chats unterschieden. Der Private Chat wird vom Nutzer erstellt und er kann den Titel sowie die Teilnehmer selbst bestimmen. Der Gruppenchat wird erstellt, sobald das Spiel erstellt wurde und fügt jeden Spieler als Teilnehmer hinzu, sobald er dem Spiel beitritt.

Und schließlich der Chat der Mörder, der jede Spielrunde neu erstellt wird und automatisch die Mörder hinzufügt. War ein Spieler in der letzten Runde ein Mörder und ist es in der neuen Runde dann nicht mehr, wird er in

der Datenbank aus dem Wolfchat entfernt, damit er keine Nachrichten mehr empfangen oder senden kann. Zu guter Letzt enthält diese Klasse eine Funktion zur Abfrage der vorhandenen Chats, die bei jedem Pull ausgeführt wird. Um den Datenverkehr effizient zu gestalten, werden auch hier nur die neuen Daten gespeichert und an den Nutzer geschickt. Um nur Daten von einem Chat zu erhalten, in dem der Nutzer auch Mitglied ist, wird die Schnittmenge der Listen "chat_member" und "chat" gebildet, in der die Player-ID des Nutzers vorhanden ist.

```
public static function getNewChatInfo($player_id, $lastpull)
  {
    $return = array();

    $result = palermoDb::get()->query("SELECT␣*␣FROM
␣␣␣␣chat_member,␣chat␣WHERE
␣␣␣␣chat_member.player_id='" . $player_id . "'␣AND
␣␣␣␣chat_member.chat_id␣=␣chat.id
␣␣␣␣AND␣modified_at>='" . $lastpull . "'");
    [...]
    }
```

6.3.5 Message

Die Message Klasse enthält zwei Funktionen. Eine zum Eintragen einer neuen Nachricht in die Datenbank und eine zum Abrufen neuer Nachrichten.

Um sicherzustellen, dass eine Nachricht dem richtigen Chat zugeordnet wird, werden Chat-ID, Game-ID und Player-ID des Senders ebenfalls in der Datenbank gespeichert.

Beim Abfragen der Datenbank wird schließlich überprüft, ob der Nutzer am Chat teilnimmt und ob die Nachrichten neuer sind als der Zeitpunkt der letzten Datenbankabfrage. So bekommt der Nutzer nur Nachrichten, die er noch nicht gespeichert hat und der Datenverkehr zwischen Server und Browser wird gering gehalten.

```
public static function getNewMessageInfo($player_id, $lastpull)
{
  $return = array();

  $result = palermoDb::get()->query("SELECT␣message.id,
␣␣chat_member.chat_id,␣message.player_id,␣message.message,␣message.time
␣␣␣␣FROM␣message,␣chat_member,␣chat␣WHERE
␣␣␣␣message.chat_id␣=␣chat.id␣AND␣chat_member.player_id=
␣␣␣␣'" . $player_id . "'␣AND␣chat_member.chat_id␣=␣chat.id
␣␣␣␣AND␣message.time>='" . $lastpull . "'");
[...]
}
```

6.3.6 Vote

Über die Vote Klasse kann man alle erdenklichen Abstimmungen tätigen. Ebenfalls beinhaltet sie eine sehr hilfreiche Funktion um zu erkennen, ob alle Spieler abgestimmt haben. Trotzdem ist die Klasse sehr klein gehalten.

6.3.7 PublicMessage

Diese Klasse ermöglicht es, öffentliche Nachrichten zu verschicken. Diese Funktion wird zurzeit nur für das Rundenende genutzt. Es ist jedoch denkbar, das Spiel später mit gewissen Spezialevents zu erweitern und diese Klasse dazu zu nutzen, alle Spieler zu informieren. Eine Art der Public Message wird durch den Nachrichtentyp definiert. “endround” ist beispielsweise die Nachricht am Ende der Runde. Bei der Nachricht selbst handelt es sich um ein JSON Objekt, das für den Nachrichtentyp benötigte Variablen speichern kann. Zum Darstellen einer Textnachricht werden beispielsweise die Attribute ‘title’ und ‘body’ benötigt.

6.3.8 Templates

Diese statische Klasse stellt einige Hilfsmethoden zur Verfügung, um das Darstellen von HTML innerhalb von PHP Dateien zu erleichtern. Letztendlich beinhaltet sie nur eine Funktion die natürlich auch in die Toolbelt Klasse übernommen werden könnte.

6.3.9 Database

Die Database Klasse ist eine öffentliche statische Klasse, die eine Datenbankverbindung aufbaut und für jegliche Aktionen zur Verfügung stellt. Zugriff auf das SQL Objekt hat man durch die statische Methode "get()". Die Verbindung wird beim ersten Aufruf von "get()" aufgebaut und bleibt bis zum Ende bestehen. Im Anschluss wird die Verbindung sauber geschlossen.

```
/**
 * This constructor establishes a database connection
 */
private function __construct() {
  global $_CONFIG;
  $this->mysqli = new mysqli($_CONFIG['DB_HOST'], $_CONFIG['DB_USER'],
  $_CONFIG['DB_PASS'], $_CONFIG['DB_DB']);
  $this->mysqli->set_charset("latin1");
}

// [...]

/**
 * This static function returns a mysqli object. If no connection has been
 * established yet, the function calls the constructor.
 * @return mysqli the mysqli object
 */
public static function get() {
  if (self::$self == null) {
    self::$self = new palermoDb();
  }

  return self::$self->mysqli;
}
```

6.3.10 Toolbelt

Bei dieser Klasse handelt es sich um unseren Werkzeuggürtel. Sie beinhaltet viele verschiedene Funktionen, die wir verteilt überall benötigen und in keine andere Klasse hinein passen. Beispielsweise kann man mit einer Methode einen zufälligen Zeichencode generieren oder in einem anderen Beispiel ein Objekt einer Klasse bekommen und im Fehlerfall automatisch eine Fehlermeldung ausgeben.

6.3.11 config.php

Dies ist keine richtige Klasse. Es handelt sich hierbei um eine Konfigurationsdatei, die wichtige Einstellungen speichert. Unter anderem stehen hier Benutzername und Passwort für die Datenbankverbindung. Ebenfalls können hier URL zum Hauptpfad und gewisse Standardeinstellungen des Spiels vorgefunden werden.

6.4 Datenbank

Die Datenbank ist ein wesentlicher Bestandteil des Programms. Hier werden wichtige Daten zum Spiel, den Spielern und Nachrichten gespeichert. Wenn zum Beispiel ein Benutzer seinen Webbrowser schließt und erneut öffnet, sind alle gespeicherten Daten verloren. Die Datenbank dient somit als sicherer Speicher, damit wichtige Spieldaten nicht verloren gehen können. In diesem Kapitel möchten wir so den Aufbau der Datenbank erklären.

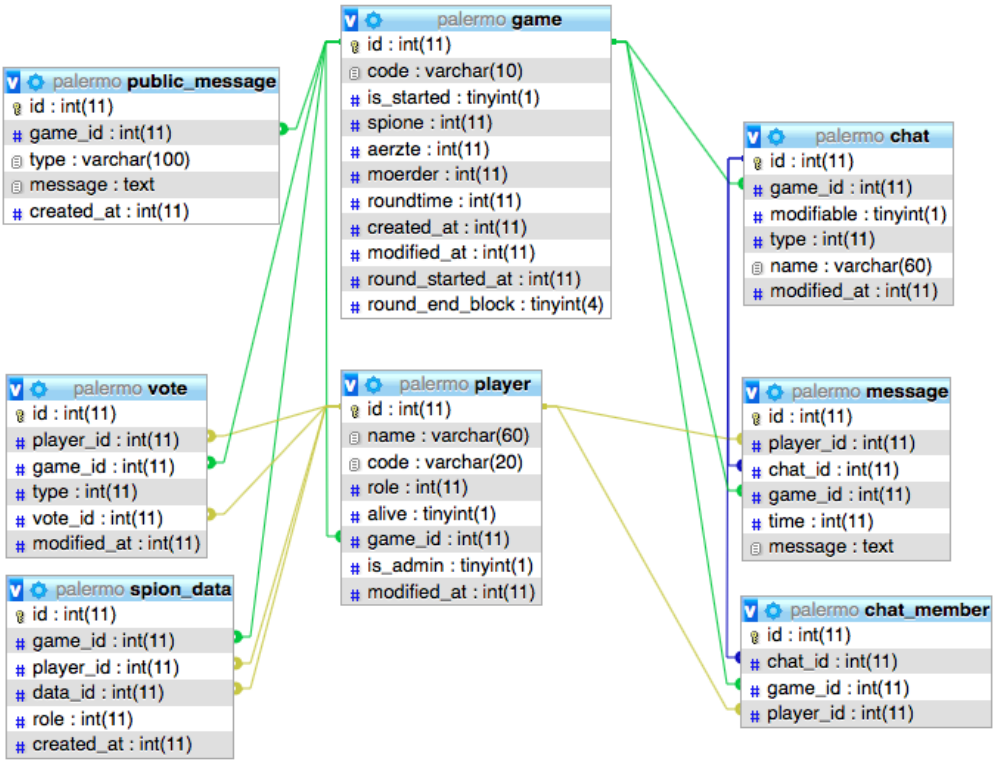

Abbildung 13: Grafische Darstellung der Datenbank für ”Mord in Palermo”

6.4.1 game Tabelle

In dieser Tabelle werden die Einstellungen und der Status des Spiels gespeichert. Es gibt Spalten für die Anzahl an Ärzten, Mördern und Spionen und die Rundenzeit. Ebenfalls wird gespeichert, ob das Spiel gerade läuft oder pausiert.

6.4.2 player Tabelle

In dieser Tabelle werden die einzelnen Spieler gespeichert. Um die Spieler dem Spiel zuordnen zu können, sind sie über die Spalte "game_id" mit der Tabelle game verknüpft.

6.4.3 chat Tabelle

In dieser Tabelle werden die Chats gespeichert. Um sie dem Spiel zuordnen zu können, existiert die Spalte "game_id".

6.4.4 chat_member Tabelle

Diese Tabelle enthält die Mitglieder eines Chats. Die Tabelle ist mit den Tabellen chats und player mit den Spalten player_id und chat_id verknüpft. Zur einfacheren Nutzung gibt es ebenfalls die Spalte game_id.

6.4.5 message Tabelle

Diese Tabelle beinhaltet die Nachrichten, die über die Chatfunktion verschickt werden.

6.4.6 vote Tabelle

Diese Tabelle beinhaltet die Abstimmungen, die die Spieler tätigen. Somit kann nachvollzogen werden, wer bereits abgestimmt hat und wer noch abstimmen muss. Die Tabelle ist mit den Tabellen player und game über die Spalten player_id und game_id verknüpft.

6.4.7 spion_data Tabelle

Diese Tabelle beinhaltet Rolleninformationen, die Spielern über ihre Mitspieler mitgeteilt werden. Späht beispielsweise ein Spion einen Mitspieler aus, so wird seine Rolle und seine ID für den Spion in dieser Tabelle gespeichert. Anschließend werden dem Spion diese Daten beim nächsten Pull übermittelt. Somit weiß der Spion dann, welche Rolle sein Mitspieler hat. Diese Tabelle

wird auch bei den Mördern benutzt, damit einem Mörder angezeigt wird, wer seine Partner sind.

6.4.8 public_message Tabelle

Diese Tabelle wird benutzt, um öffentliche, also für alle sichtbare Nachrichten zu verschicken. Sie wird hauptsächlich dazu verwendet, die Nachricht für das Rundenende zu verschicken. Sie kann aber auch zukünftig für vielfältige Zwecke genutzt werden.

6.5 Installationsskript

Um das Programm schnell und einfach (und auch für Laien) auf einem Server installieren zu können, haben wir ein Installationsskript geschrieben. Denn wir finden es immer schön, neue Software schnell und unkompliziert installieren zu können, ohne lange in einer Dokumentation lesen zu müssen. Aus Sicherheitsgründen funktioniert es nur, wenn die class/config.php Datei nicht vorhanden ist (sonst könnte jeder Besucher einfach die Konfigurationsdatei mit seinen Einstellungen überschreiben). Sofern die Software noch nicht installiert wurde, wird der Administrator sogar automatisch zum Installationsskript weitergeleitet.

Beim Installationsvorgang hat der Administrator die Möglichkeit, wichtige Einstellungen wie URL und Benutzerdaten für die Datenbankverbindung einzugeben. Beim Klick auf Start werden zunächst alle eingegebenen Daten überprüft. Tritt ein Fehler auf (zum Beispiel beim Erstellen einer Testdatenbanktabelle) wird der Installationsvorgang abgebrochen und der Administrator aufgefordert, die Einstellungen zu überprüfen. Danach wird die Konfigurationsdatei class/config.php erstellt und mit den gewählten Einstellungen beschrieben. Anschließend werden die Datenbanktabellen erstellt. Danach ist die Installation abgeschlossen und die Software kann verwendet werden. Für doppelte Sicherheit kann der Administrator den Installationsordner löschen.

6.6 Sicherheit gegen Cyberangriffe

Um zu vermeiden, dass wir eines Tages plötzlich die Kontrolle über unseren Server verlieren, wollten wir von Anfang an eine hohe Priorität auf Sicherheit legen. Als Leitfaden warfen wir einen Blick in die OWASP (Open Web Application Security Project), die jedes Jahr die Top 10 der Sicherheitslücken in Webapplikationen veröffentlicht[10]. Seit mehreren Jahren halten “Injections”

[10]OWASP: Top 10-2017, Stand: 16. Aug. 2018, URL: `https://www.owasp.org/index.php/Top_10-2017_Top_10`

den ersten Platz. Unter einer Injection versteht man eine Einführung von fremden Code in eine Webseite. Da wir häufig SQL verwendeten, um Daten in die Datenbank zu schreiben, war unser Code potentiell besonders anfällig für SQL Injections. Daneben haben Benutzer die Möglichkeit, eigenen Text an andere Benutzer zu schicken. Hierbei könnte man durch Codeeinschlüsse potenziell XSS Attacken ausführen. Im Folgenden erklären wir die beiden Angriffsmethoden und welche Maßnahmen wir dagegen unternahmen.

6.6.1 SQL Injections

SQL Injections sind Codeeinschlüsse in normale SQL Abfragen, die das gewünschte Verhalten der Abfrage verändern oder sogar fremde Abfragen einschleusen.

```
$query = "SELECT * FROM player WHERE id=$player_id;";

$player_id = 3;

echo $query;
// Ausgabe: SELECT * FROM player WHERE id=3;

$player_id = "3; DROP TABLE game;";

echo $query;
// Ausgabe: SELECT * FROM player WHERE id=3; DROP TABLE game;
```

In diesem Beispiel wird durch eine SQL Injection eine ganze Tabelle gelöscht. Weiterhin lassen sich durch SQL Injections beliebige Daten anzeigen, eigene Daten einschleusen oder beliebige Daten löschen. Da uns das natürlich nie passieren sollte, haben wir besondere Vorkehrungen getroffen. Jedes Mal, wenn der Benutzer einen Parameter wählen kann, der anschließend für eine SQL Abfrage gebraucht wird, benutzten wir die “prepare” Funktion von php-mysqli. Dadurch wird die reine SQL Abfrage mit Platzhalter anstatt Variablen eingefügt. Anschließend werden die Parameter getrennt von der Abfrage eingelesen. Dadurch werden SQL Injections unmöglich. Da das ganze etwas mehr Code in Anspruch nimmt, verwenden wir diese Art der Abfrage nur, wenn der Benutzer direkt oder indirekt Einfluss auf einen Parameter in der Abfrage hat. Sobald wir jedoch Abfragen ausführen, die nur Parameter benötigen die entweder von uns selbst generiert oder bereits aus der Datenbank stammen, benutzen wir die klassische Abfragevariante.

Geschützte Abfragevariante:

```
$prep = $mysqli->prepare("SELECT * FROM player WHERE id=?");

$prep->bind_param("i", $player_id);

$prep->execute();
```

Klassische Abfragevariante:

```
$mysqli->query("SELECT * FROM player WHERE id=$player_id");
```

6.6.2 XSS Angriffe

XSS (Cross Site Scripting) ist ein Angriff, bei dem fremder HTML Code zeitweise oder permanent auf einer fremden Webseite eingeschleust wird. Man hat somit die Möglichkeit, das Aussehen der Webseite komplett anzupassen und sogar schädlichen Javascript Code einzufügen. Fügt man beispielsweise folgenden Code als Text in das Chatfenster ein, wird ein Popup dargestellt und alle Cookieinformationen werden entblößt:

```
alert(document.cookie);
```

Alternativ könnte man über AJAX jegliche Informationen aus dem Zwischenspeicher an einen fremden Server schicken. Mit dieser Sicherheitslücke könnte man ebenfalls einen Cheat bauen, der fremden Javascript Code bei allen Mitspielern einfügt, der dann die Rolle und eventuell sogar geheime Nachrichten an einen fremden Server schickt auf den der Angreifer Zugriff hat. Glücklicherweise gibt es eine einfache Gegenmaßnahme.

Damit unser Chat nicht Opfer einer XSS Attacke wird und möglicherweise Cookies oder andere wertvolle Informationen der Benutzer ausgespäht werden, haben wir eine einfache aber effektive Abwehrmaßnahme eingefügt. Durch die PHP Funktion “htmlspecialchars”, werden alle HTML Sonderzeichen in Text umgewandelt. Beispielsweise “<b>” wird somit nicht mehr als HTML Tag, sondern als Text interpretiert. Somit nimmt man dem Angreifer die Möglichkeit, HTML Tags einzuschleusen und verhindert XSS angriffe.

7 Testspiele

7.1 Erster Test 05.08.2018

Als die erste Version auf dem Server installiert wurde und der wesentliche Teil des Spiels funktionierte, entschieden wir uns, den ersten Testlauf mit einer Gruppe E-Commerce Studenten zu starten. Das Spiel war nicht allgemein bekannt und es bedurfte zunächst einer Erklärung der Regeln. Hinzu kam, dass die Benutzeroberfläche, anders als wir erwartet hatten, nicht selbsterklärend zu sein schien. Nachdem alles ausführlich erklärt wurde, konnte die erste Runde starten. Es stellte sich bald heraus, dass das nächste Problem sein würde, die Runde zu beenden. Da Detektive, Spione und Mörder zwei Abstimmungen zu tätigen haben statt nur einer und die Runde erst vorbei sein würde, wenn alle Abstimmungen getätigt wurden, suchten nun alle Mitspieler gemeinsam nach der einen Person, die vergessen hatte abzustimmen. So dauerte es eine ganze Weile, bis der eine Mörder gefunden wurde, dessen Abstimmung noch fehlte. In der nächsten Runde wurde besagter Mörder schließlich von den Bürgern sofort gehängt.

Als Fazit haben wir uns einige Verbesserungen überlegt. Wir möchten die Spielregeln gerade beim ersten Spiel leicht abrufbar machen. Über einen Button soll ein interaktives Tutorial gestartet werden, das den Aufbau der Benutzeroberfläche erklärt. Zudem soll der Spieler darauf hingewiesen werden, dass er noch Abstimmungen zu tätigen hat. Damit die Runde nicht zu lange dauert, soll ein Timer eingebaut werden, der die maximale Rundendauer begrenzt. Außerdem soll die Nachricht am Ende der Runde aussagekräftiger werden. Es soll zum Beispiel ersichtlich sein, wenn die Ärzte ein Opfer der Mörder gerettet haben, oder dass das Opfer der Mörder vorher bereits durch die Bürger erhängt wurde. Zuletzt soll auch ersichtlich sein, wie viele und wer gegen einen selbst abgestimmt hat. Dadurch kann es zu witzigen Situationen und Konflikten kommen, die das Spiel bereichern.

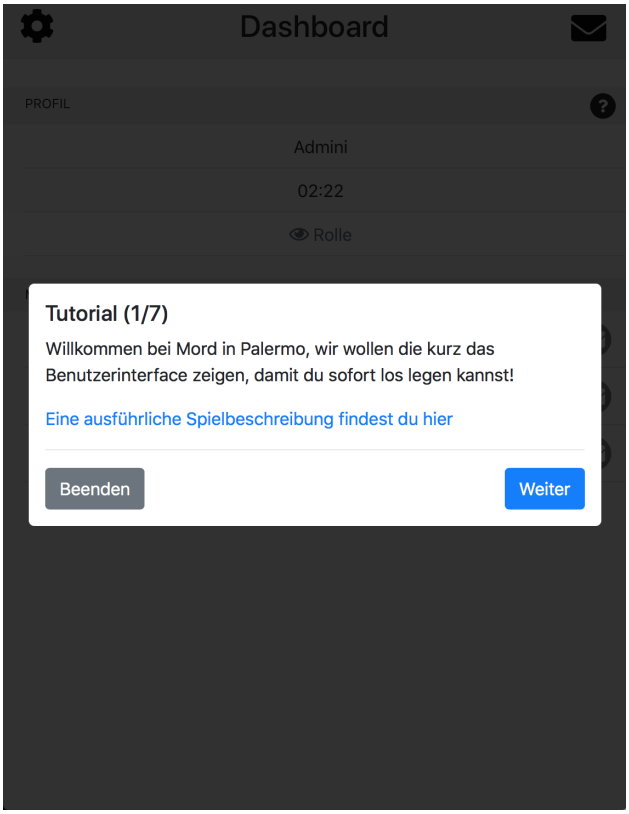

Abbildung 14: Interaktive Einführung in das Benutzerinterface

7.2 Zweiter Test 15.08.2018

Als wir das Gefühl hatten, wir seien mit allem fertig und auch unsere Testspiele ohne Probleme liefen, luden wir einige Mitstudenten für ein weiteres Testspiel ein.

Nach kurzer Erklärung und Beantwortung aller Fragen haben unsere Probanden das Userinterface und die Benutzung schnell verstanden. Jedoch kamen sie aus Gewohnheit immer wieder auf den Zurück Button ihres Handys und flogen damit aus dem Spiel. Da es sich bei unserer App um eine sogenannte Single Site Application handelt, landet man beim Betätigen der Zurücktaste wieder auf der Startseite. Das Gerät lädt die letzte besuchte Seite komplett aus dem Cache und die eigentliche Weiterleitung funktionierte damit nicht (Spieler werden von der Startseite zum Spiel weitergeleitet, wenn in den Cookies ein Player Code gespeichert ist). Wir lösten das Problem mit Javascript. Mittels einer Endlosschleife wird alle 50 Millisekunden überprüft, ob der Player Code Cookie vorhanden ist und gegebenenfalls wird der Spieler auf die Spielseite weitergeleitet. Drückt der Spieler nun versehentlich auf die

Zurücktaste, wird er nun einfach wieder zurück auf das Spiel geleitet.

Im späteren Spielverlauf merkte ein Proband, dass der Chat keine Emojis erkennt und verschicken kann. Nach kurzer Recherche stellte sich heraus, dass die Datenbank die Codierung Latin-1 verwendet. PHP versucht jedoch Daten im UTF-8 Format zu schreiben und zu lesen. Dadurch funktionierten zwar die normalen Sonderzeichen, jedoch machten Emojis Probleme. Nachdem dann die Codierung von PHP ebenfalls auf Latin-1 umgestellt wurde, funktionierten Emojis einwandfrei.

Ein weiterer Proband bemerkte, dass sich keine Namen mit Ä, Ö oder Ü wählen ließen. Hier passten wir die regulären Ausdrücke an, die die Namen filtern und lösten damit das Problem. Ein weiteres Problem war, dass sich das Tutorial nicht scrollen ließ. Dadurch konnte man auch nicht auf den "Weiter" oder "Beenden" Button drücken. Das lösten wir damit, dass wir die längsten Tutorials in zwei Teile teilten und damit keine Notwendigkeit mehr für scrollen gegeben war. Zuletzt wurde noch die Nachricht am Rundenende verbessert. Gibt es keine Opfer der Mörder, so erscheint auch keine Nachricht mehr.

Es fiel uns außerdem auf, dass der Chat für geheime Nachrichten nur sparsam genutzt wurde. Das lag vor allem daran, dass man sich während des Spiels auf dem Dashboard bei der Spielerliste weiter unten aufhält, um seine Abstimmungen zu tätigen und gar nicht mitbekommt, wenn oben rechts im Bild eine neue Nachricht angezeigt wird. Daher entschieden wir uns, die neuen Nachrichten auch in der Spielerliste neben dem entsprechenden Namen anzuzeigen.

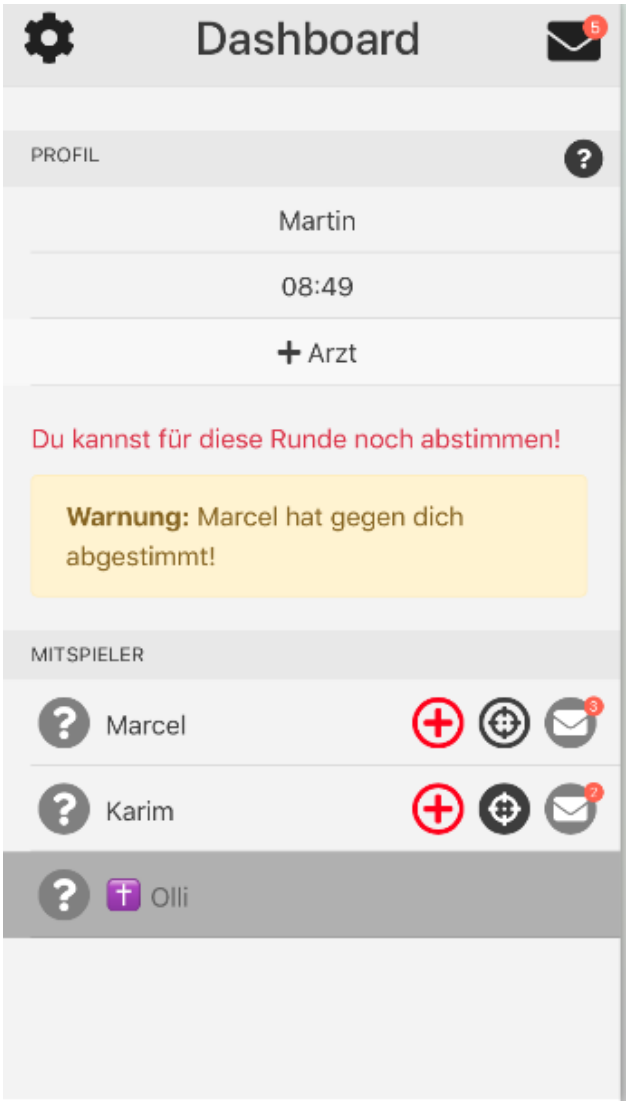

Abbildung 15: Dashboard mit Rundentimer, Benachrichtigungen die das Spiel betreffen und Zähler für Nachrichten von anderen Spielern

7.3 Rezeption

Mit großer Freude fanden wir heraus, dass unsere App mit Begeisterung aufgenommen wurde. Nach einer kurzen Einfindungsphase haben alle Mitspieler die App verstanden und konnten alle Funktionen nutzen.

Die Rundendauer wurde zeitweise als zu schnell empfunden und den Spielern hat die erzählte Geschichte des Moderators gefehlt. Stellte man die Rundenzeit auf unter drei Minuten, so minimierten sich Diskussionen und es wurde fast nur noch kommentarlos getippt und auf die Wahlen der anderen reagiert. Hat man beispielsweise für eine beliebige Person gestimmt, war der Reiz für diese Person groß, diese Stimme mit einer Stimme gegen die eigene Person zu stimmen. Durch das leichte Ändern der eigenen Stimme kam es vermehrt zu "Rudelverhalten" unter den Probanden. Das heißt: Hatte ein Spieler geringfügig mehr Stimmen als andere, kam es häufiger vor, dass die anderen auch für ihn abstimmten, aus Angst, man könne sonst noch für sie abstimmen. Auch wenn dieses Verhalten die ersten paar Male einen gewissen Unterhaltungswert besitzt, verfehlt es doch leider den eigentlichen Sinn des Spiels. War der Rundentimer dagegen ausreichend lang eingestellt, hat das dynamische Abstimmverhalten, das durch unsere App gefördert wird, auch positive Nebeneffekte gezeigt. Man konnte im Laufe der Diskussion in Echtzeit beobachten, wie viele Mitspieler man mit seinen Argumenten erreicht hatte und ob einige Mitspieler (vielleicht waren es ja die Mörder) sich einfach nicht von den Argumenten überzeugen lassen wollten. Dadurch, dass das Wahlergebnis sich noch in letzter Sekunde ändern konnte, wurden die Diskussionen mit abnehmender Zeit zunehmend temperamentvoller.

Der Chat konnte leider sein volles Potenzial noch nicht ausschöpfen. Wenn jemand etwas zu sagen hatte, sagte er es gerade heraus in die Runde und der Gruppenchat wurde eher zum Spaß genutzt. Der Mörderchat dagegen erwies sich für geheime Besprechungen als sehr nützlich. Aber auch der selbst erstellte Gruppenchat von einem sehr findigen Spion hat in einer Situation strategischen Nutzen bewiesen.

8 Abschließender Vergleich zu verschiedenen Varianten von "Mord in Palermo"

Wie in der Einleitung dieser Arbeit bereits kurz erwähnt, sind wir nicht die ersten, die an einer Spielerweiterung für die klassische Variante von "Mord in Palermo" gearbeitet haben. Eine sehr weit verbreitete Variante des Spiels ist das im Jahr 2001 erschienene Kartenspiel "Die Werwölfe von Düsterwald"[11], das zunächst als reines Kartenspiel ohne Software veröffentlicht wurde, aber mit der App "Werwölfe Vollmondnacht"[12] in Apple's App Store ein Kartenspiel mit App-Erweiterung und somit ein mit unserer Arbeit vergleichbares Produkt darstellt.

Eine andere Variante des "Werwolf"-Spiels, für das keine Karten mehr benötigt werden, ist die App "Werwolf"[13], die den kompletten Kartensatz und den Spielleiter ersetzt. Die App wird auf einem Gerät verwendet und benötigt keine Internetverbindung.

Eine reine Onlineversion von "Mord in Palermo" unter anderem Namen ist das Spiel "Town of Salem"[14], das als reines Browsergame konzipiert wurde und unserer Arbeit damit am nächsten kommt.

Die drei eben vorgestellten Spiele sind jeweils stellvertretende Beispiele für drei mögliche Varianten, wie man das klassische "Mord in Palermo"-Spiel mithilfe von Software erweitern kann. Im Folgenden möchten wir unsere Arbeit mit den oben genannten Stellvertretern der drei Kategorien "Kartenspiel mit App-Erweiterung", "Reine App ohne Karten auf nur einem Gerät" und "Reines Browsergame" vergleichen und die Vor- und Nachteile erörtern.

8.1 Kartenspiel mit App-Erweiterung

Die App "Werwölfe Vollmondnacht" aus dem App Store ist nur in Kombination mit dem Kartenspiel von Ravensburger[15] spielbar. Die App ersetzt den Spielleiter und eine Stimme begleitet den Spielverlauf mit einer Erzählung. Es gibt über 40 verschiedene Rollen und jede Runde werden besondere

[11] Wikipedia: Die Werwölfe von Düsterwald, Stand: 28. Aug. 2018, URL: https://de.wikipedia.org/wiki/Die_Werwölfe_von_Düsterwald

[12] App Store: Werwölfe Vollmondnacht, Stand: 28. Aug. 2018, URL: https://itunes.apple.com/de/app/werwölfe-vollmondnacht/id1015385157?mt=8

[13] Google Play Store: Werwolf, Stand: 28. Aug. 2018, URL: https://play.google.com/store/apps/details?id=org.faudroids.werewolf&hl=de

[14] Town of Salem, Stand: 28. Aug. 2018, URL: http://www.blankmediagames.com

[15] Ravensburger: Werwölfe Vollmondnacht, Stand: 28. Aug. 2018, URL: https://ravensburger.de/produkte/spiele/kartenspiele/werwoelfe-vollmondnacht-26703/index.html

Situationen erstellt, die den Spielverlauf beeinflussen und spannend halten.

Eine Erzählerstimme und eine Hintergrundgeschichte hat unser Programm leider nicht. Hier muss ein Teilnehmer sich noch immer dazu erbarmen, seine eigene Stimme zur Verfügung zu stellen und sich die Story selbst ausdenken. Tatsächlich machen auch die vielen verschiedenen Rollen, die man bei "Werwölfe Vollmondnacht" spielen kann, das Spiel etwas interessanter als unsere weniger fhantasievolle Version mit nur vier spielbaren Rollen und vergleichbar linearem Spielverlauf.

Der große Nachteil ist jedoch, dass man "Werwölfe Vollmondnacht" ausschließlich zusammen mit den Karten spielen kann und die App nichts weiter als die Moderation übernimmt. Bei sehr vielen Mitspielern wird das Spiel leider sehr unübersichtlich und dauert am Ende jeder Runde, wie in der klassischen Version von "Mord in Palermo", sehr lange bis alle Abstimmungen getätigt wurden. Solange man keinen Messenger wie z. B. "Whatsapp"[16] benutzt, was während des Spiels wiederum sehr auffällig ist, hat man auch keine Möglichkeit geheime Absprachen mit anderen Spielern zu halten.

8.2 Reine App ohne Karten auf nur einem Gerät

Ein gutes Beispiel für die Entwicklung weg von den Karten in Richtung reine Softwarelösung stellt die "Google Play Store"-App mit dem Namen "Werwolf" dar. Sie bietet eine vergleichbare Rollenvielfalt, wie das Ravensburger Kartenspiel, und man benötigt nur ein Gerät, um das Spiel zu spielen. Die Rollen werden den Spielern in einer bestimmten Reihenfolge zugewiesen und das Smartphone muss dann in genau dieser Reihenfolge herumgereicht werden, damit jeder Spieler seinen Turnus korrekt spielen kann.

Unserer Meinung nach ist die Benutzung dieser App eine gute Lösung, wenn man keine Internetverbindung zur Verfügung hat und nicht im Besitz der Ravensburger Spielkarten ist. Ansonsten hebt sie sich vom klassischen "Mord in Palermo"-Spiel nur dadurch ab, dass man keinen Spielleiter mehr benötigt und eine größere Rollenauswahl hat. Mit zunehmender Teilnehmerzahl wird die Rundendauer immer länger bis der Spielspaß schließlich darunter leidet. Dadurch, dass unsere Version auf dem Smartphone eines jeden Teilnehmers gespielt wird und die Abstimmungen alle während der Diskussionsphase gleichzeitig getätigt werden können, ist die Rundendauer unabhängig von der Teilnehmerzahl. Es können theoretisch beliebig viele Teilnehmer das Spiel gleichzeitig Spielen, ohne dass das Spielerlebnis darunter leidet, sondern wahrscheinlich im Gegenteil sogar verbessert wird.

Außerdem fehlt auch hier bei "Werwolf" noch die Möglichkeit, geheime

[16] Whatsapp, Stand: 28. Aug. 2018, URL: `https://www.whatsapp.com`

Absprachen mit Mitspielern zu halten, was unser Programm von den beiden letzten Beispielen deutlich abhebt.

8.3 Reines Browsergame

Eine gute Umsetzung vom klassischen "Mord in Palermo" in ein reines Browsergame mit großer Rollenauswahl, spannender Story und ansprechendem Design ist das Browsergame "Town of Salem".

Der Spielablauf ist aufgebaut wie im klassischen Spiel mit einer "Tagphase", in der die Teilnehmer in einem Gruppenchat diskutieren und einen Spieler wählen, der gehängt werden soll und anschließend mit einer "Nachtphase", in der jeder seine rollen-spezifischen Fähigkeiten einsetzen kann. Es besteht außerdem die Möglichkeit, private Nachrichten an Mitspieler zu schicken, sodass geheime Absprachen möglich sind. Der Host hat die Möglichkeit einzustellen, welche Rollen im Spiel vorkommen sollen und wie viele Teilnehmer das Spiel haben soll. Während des Spiels kann der Teilnehmer seine Rollenbeschreibung einsehen, sodass trotz großer Rollenvielfalt jeder weiß, was er zu tun hat.

Das Browsergame ist insofern mit unserer Version vergleichbar, als es ein Browser-basiertes Spiel ist, dem mehrere Teilnehmer von ihren Computern aus beitreten können, der Spielleiter ersetzt wird, eine große Anzahl an Spielern gleichzeitig teilnehmen kann, ohne dass das Spielerlebnis darunter leidet und ein Chat mit der Möglichkeit der geheimen Absprache gegeben ist. Nutzt man "Town of Salem" allerdings als Gesellschaftsspiel mit Softwareunterstützung, wie unser Spiel konzipiert wurde, stellt sich schnell heraus, dass das detailreiche Design und der große Funktionsumfang "Town of Salem" zwar zu einem sehr ansprechenden Browsergame machen, jedoch als Gesellschaftsspiel zu unübersichtlich werden lässt und statt verbale Diskussionen anzuheizen, allgemein eher als störend empfunden wird, weil die komplette Aufmerksamkeit auf die Bedienung der Software gerichtet wird.

Unsere "Palermo"-Applikation hat ein schlichtes, praktisches und dabei modernes Design, das erlaubt, die wichtigsten Funktionen auf einen Blick zu finden und diese zu benutzen, ohne dass die verbalen Diskussionen dafür abnehmen. Und dadurch, dass die Rolle automatisch auf dem Dashboard versteckt wird, kann niemand durch Schielen auf den Bildschirm die Rolle seines Mitspielers erfahren. Unser Programm ist durch die Benutzung von Bootstrap auch auf mobilen Endgeräten problemlos nutzbar, während "Town of Salem" durch die Benutzung von "Adobe Flash Player" auf iOS-Geräten nur bedingt nutzbar ist.

9 Fazit

Da im klassischen Spiel die erzählerische Art einen deutlich höheren Wert erhält als in unserer Spielvariante, kann unser Programm nach aktuellem Stand das klassische Spiel nicht vollständig ersetzen. Es stellt vielmehr eine schnelle und spaßige Alternative dar, die vor allem bei großer Teilnehmerzahl gut genutzt werden kann, um die Abstimmungen und den Spielverlauf zu vereinfachen und zu beschleunigen. Um das vollwertige Spielerlebnis zu garantieren, müsste das Programm um eine variierende Hintergrundgeschichte ergänzt werden. Man könnte mehrere Erzählstränge bilden und mit einer KI dem Spielverlauf anpassen, sodass der Spielspaß auch nach mehreren Spielen noch nicht abflaut.

Eine weitere interessante Möglichkeit sich von allen übrigen "Mord in Palermo"-Varianten abzugrenzen, wäre die Erstellung eines Chatbots[17], der im Spiel teilnimmt und sich je nach Rolle unterschiedlich verhält. Damit würde zwar das Ziel unserer Arbeit - eine Softwareunterstützung für ein Gesellschaftsspiel zu erstellen - entarten, es wäre aber dennoch ein interessantes Thema.

Abschließend lässt sich zusammenfassen, dass unser Programm zwar Potenzial hat erweitert zu werden, die an ihn gestellten Anforderungen aber zufriedenstellend erfüllt und eine praktische Applikation für das Gesellschaftsspiel "Mord in Palermo" darstellt, wie sie es so, aufgrund der in Kapitel 8 diskutierten Alleinstellungsmerkmale, noch nicht gibt.

10 Quellcode

Der Quellcode unserer Software kann über GitHub unter folgender URL eingesehen werden: `https://github.com/cbacon93/palermo`

[17] Chatbot, Stand: 28. Aug. 2018, URL: `https://de.wikipedia.org/wiki/Chatbot`