Dokumentation "FINd die Werwölfe"

Alexandra Koch, Gina Seckendorf, Jonathan Kloss

January 9, 2017

Contents

1 Projektidee **3**
1.1 Spielbeschreibung 3
1.1.1 Vorbereitung 3
1.1.2 Nachtphase 3
1.1.3 Tagphase 4
1.1.4 Ende des Spiels 5
1.2 Idee: Umsetzung als Android-App 5

2 Umsetzung **6**
2.1 Projektplanung 6
2.1.1 Meilensteine 6
2.1.2 Projektplan 7
2.1.3 Burn-Down-Chart 7
2.2 Projektdurchführung 7
2.2.1 Klassendiagramm 7
2.2.2 Beschreibung der Klassen 7
2.2.3 Datenbankschema 18

1 Projektidee

1.1 Spielbeschreibung

Als Vorlage für die App dient das Spiel "Die Werwölfe von Düsterwald" von Philippe des Phalliéres und Hervé Marly. Thematisch geht es darum, dass das kleine Dörfchen Düsterwald von Werwölfen heimgesucht wird. Die Gruppe der Bürger versucht die Wölfe, die sich als Bürger getarnt haben, zu entlarven. Dagegen versuchen die Wölfe als einzige zu überleben und Widersacher auszuschalten.

1.1.1 Vorbereitung

Der Spielleiter mischt alle Charakterkarten und teilt an jeden Spieler verdeckt eine davon aus. Die Spieler schauen sich ihre Karte an und erkennen nun, ob sie einen Werwolf, einen einfachen Dorfbewohner oder eine Sonderrolle verkörpern. Danach ruft der Spielleiter zur ersten Nacht aus und das eigentliche Spiel kann beginnen.

1.1.2 Nachtphase

In der Nachtphase schließen alle Spieler die Augen. Der Spielleiter ruft die handelnden Charaktere einzeln auf. Sie öffnen ihre Augen und führen ihre Aktion aus.

Der *Dieb* ist der erste, der im Spiel erwacht. Wird mit Dieb gespielt, werden zwei Karten mehr ausgeteilt. Der Dieb darf diese ansehen und seine Karte gegen eine der beiden übrig gebliebenen Karten austauschen. Er hat ab jetzt also eine neue Rolle. Möchte er nicht tauschen, ist er für den Rest des Spiels einfacher Dorfbewohner.

Amor erwacht nur einmal in der allerersten Nacht, um zwei Spieler seiner Wahl miteinander zu verkuppeln (eventuell auch sich selbst). Danach schläft er wieder ein. Anschließend berührt der Spielleiter die beiden Verliebten an der Schulter, sodass diese kurz erwachen können und wissen, wer der jeweilige Partner ist. Die Verliebten haben im Laufe des Spiels die Aufgabe, den Partner zu beschützen, denn wenn einer der beiden stirbt, macht es ihm der Partner trauernd nach; sie dürfen nie gegeneinander stimmen.

Werden die *Werwölfe* vom Spielleiter aufgerufen, wachen sie auf und

erkennen sich gegenseitig. Je nach Spielerzahl gibt es zwei bis vier Wölfe. Die Wölfe einigen sich durch Gesten auf ein Opfer und schlafen dann wieder ein. Der Spielleiter merkt sich das Opfer der Werwölfe.

Das kleine *Mädchen* darf nachts in der Werwolf−Phase heimlich blinzeln, um so die Werwölfe zu erkennen. Die Werwölfe ihrerseits hingegen achten natürlich darauf, das Mädchen dabei zu ertappen, es besteht also beim Blinzeln ein gewisses Risiko.

Die *Seherin* erwacht in der Nacht alleine und zeigt auf einen Spieler. Der Spielleiter zeigt der Seherin nun die entsprechende Charakter-Karte der Person. Die Seherin weiß dadurch mehr als die übrigen Dorfbewohner, muss aber mit ihrem Wissen sorgfältig umgehen, um nicht von den Werwölfen enttarnt zu werden.

Die *Hexe* erwacht immer nachdem die Werwölfe ihr Opfer ausgesucht haben. Sie hat im Verlauf des gesamten Spiels einen Gift- und einen Heiltrank. Der Spielleiter zeigt auf die Person, die von den Werwölfen als Mordopfer gewählt wurde und die Hexe kann diese mit ihrem Heiltrank heilen (auch sich selbst), so dass es am nächsten Morgen keinen Toten gibt. Sie kann aber auch den Gifttrank auf einen anderen Spieler anwenden − dann gibt es mehrere Tote.

Scheidet der *Jäger* aus dem Spiel aus, feuert er in seinem letzten Atemzug noch einen Schuss ab, mit dem er einen Spieler seiner Wahl mit in den Tod reißt, d.h. er bestimmt einen Spieler, der mit ihm aus dem Spiel ausscheidet.

1.1.3 Tagphase

Am Tag wachen alle Spieler auf. Das Opfer der Werwölfe wird verkündet, es dreht seine Karte um, gilt als tot und scheidet aus der Runde aus, d. h., er darf keinen Kommentar zum Spiel mehr abgeben. Nun diskutieren die Dorfbewohner, wer von ihnen ein Werwolf sein könnte. Diese Diskussionsphase ist das eigentliche Herzstück des Spiels.

Am Ende des Tages gibt es eine sogenannte Abstimmung durch das Dorfgericht, wobei auf Kommando des Spielleiters jeder, außer den ausgeschiedenen Personen, mit dem Finger auf eine für ihn verdächtige Person deutet. Wer die meisten Stimmen erhält, scheidet aus. Bei Gleichstand gibt es eine Stichwahl, bei erneutem Patt entscheidet ein zu Spielbeginn gewählter

Hauptmann. Den verbleibenden Spielern wird die Charakterrolle des ausgeschiedenen Spielers bekanntgeben. Nach dem Tag wird es wieder Nacht und der Zyklus beginnt von vorn.

1.1.4 Ende des Spiels

Das Spiel endet, sobald entweder alle Werwölfe oder alle Bürger tot sind. Das Ziel der Werwölfe ist es, alle Bürger auszulöschen, während die Dorfbewohner den Wölfen den Garaus machen wollen. Lediglich wenn das Liebespaar aus einem Werwolf und einem Dorfbewohner besteht, können diese beiden Spieler nur dann gewinnen, wenn außer ihnen niemand überlebt.

1.2 Idee: Umsetzung als Android-App

Die App versucht das Spiel so gut wie möglich digital umzusetzen. Die Spielkarten werden durch ein Android−Gerät ersetzt und ein Spielleiter ist nicht mehr notwendig. Alle notwendigen Daten werden in einer Datenbank gespeichert. So kann jedes Gerät jederzeit darauf zugreifen. Der Spielablauf soll nun wie folgt aussehen: Jeder Spieler benötigt zum Spielen sein Smartphone mit der "FINd die Werwölfe"−App mit einen Spieler−Account. Ein Spieler kann nun ein neues Spiel erstellen. Dadurch wird ein QR-Code erstellt, welchen die anderen Spieler scannen können, um diesem Spiel beizutreten. Die Rollen werden den Spielern automatisch und zufällig zugewiesen. Sobald alle Spieler bereit sind, kann das Spiel beginnen. Wie im Kartenspiel haben die Spieler die Augen während der Nachtphase geschlossen. Audio−Ausgaben leiten die Spieler durch das Spiel und fordern die Spieler zum Öffnen/Schließen ihrer Augen auf. Diese werden vom "Spielleiter"−Gerät (*Gerät des Spielers, der das Spiel erstellt hat*) abgespielt. In der Nacht erwachen nach und nach die Sonderrollen und führen ihre Aktionen auf dem Gerät aus. Am Tag sehen dann alle Spieler auf ihren Geräten welche Opfer es in der Nacht gegeben hat. Danach werden den Spielern Buttons von allen Spielern angezeigt. Aus diesen können sie dann das Opfer des Tages wählen. Die Geräte stehen in ständiger Verbindung zur Datenbank, um zu wissen in welcher Phase sich das Spiel befindet und wann das Ende des Spiels erreicht ist.

2 Umsetzung

2.1 Projektplanung

Zur Implementierung der App wurde Android-Studio verwendet. [Datenbank auf Server] ...

2.1.1 Meilensteine

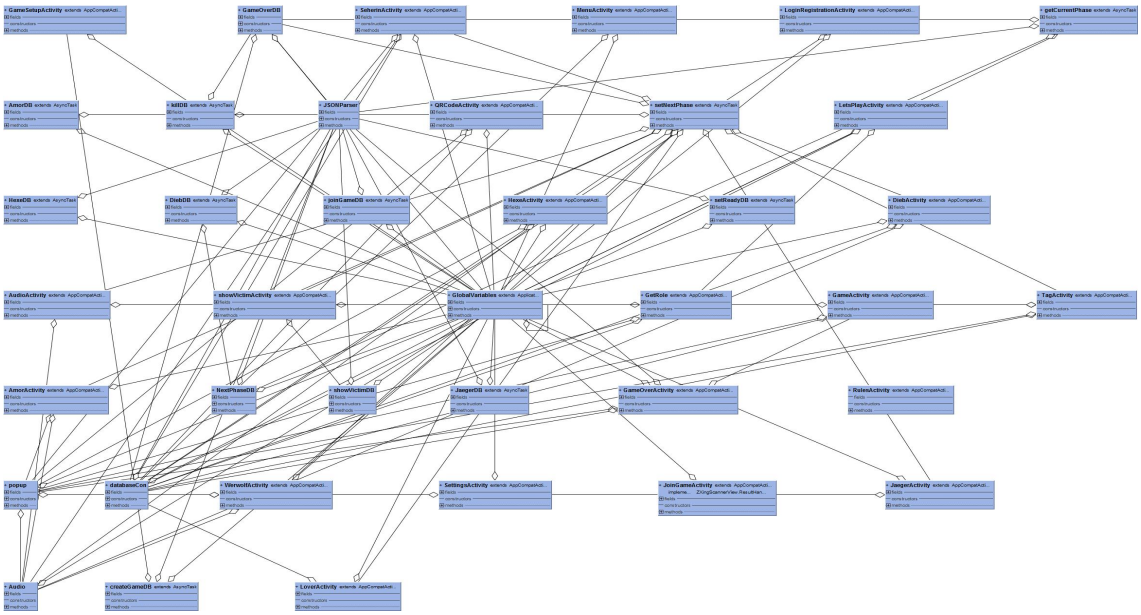
1. **Aufgabenplanung**
   →
2. **Modellierung der App**
   →
3. **Recherche**
   → Informieren über die Techniken und Werkzeuge zur Umsetzung der App
4. **Implementierung**
   → Umsetzung aller Funktionen in Android−Studio
5. **Datenbank**
   → Implementierung der php-Files zur Kommunikation zwischen Datenbank und Android−Geräten
6. **Design**
   → optische und akustische Gestaltung der App
7. **Dokumentation**
   → Erklärung für Entwickler und Benutzer wie die App funktioniert
8. **Testen**
9. **Abschlusspräsentation**

2.1.2 Projektplan

2.1.3 Burn-Down-Chart

2.2 Projektdurchführung

2.2.1 Klassendiagramm



2.2.2 Beschreibung der Klassen

AmorActivity. Zu Beginn der Activity wird über das Spielleiter-Gerät ein Audio abgespielt, welches Amor auffordert die Augen zu öffnen. Diesem wird nun auf dem Display die Spieler anzeigt, welche mit Hilfe der Methode createObjects der Klasse *GameActivity* erstellt werden. Allen anderen Spielern wird ein schwarzer Bildschirm angezeigt (activity_wait layout). Amor kann nun zwei Spieler (auch sich selbst) wählen. Seine Wahl bestätigt er mit dem "Bestätigen"− Button. Damit wird die Klasse *AmorDB* aufgerufen und die gewählten Spieler werden in der Datenbank mit ihrem jeweiligen Liebespartner abgespeichert (setLovers()). Über ein PopUp der Klasse *popup.java* wird Amor noch einmal das Liebespaar angezeigt. Mit dem Klick auf OK wird die nächste Phase aufgerufen (new setNextPhase().execute("audio");).

AudioActivity. Diese Activity wird nach jeder Phase aufgerufen (new

setNextPhase().execute("audio");), um dem Spieler zu sagen, dass dieser wieder einschlafen/die Augen schließen soll. Sie zeigt einen schwarzen Bildschirm und lässt vom Spielleiter-Gerät das entsprechende "Einschlaf-Audio" abspielen. Dazu ruft sie die Klasse *Audio* auf.

DiebActivity. Aus dieser Activity werden zuerst die beiden übrig gebliebenen Rollen aus der Datenbank geladen. Diese werden zusammen mit dem entsprechenden Bild und der Beschreibung auf zwei Buttons dargestellt. Der Dieb bekommt zwei Rollen angezeigt. Diese werden mit Hilfe der Methode *DiebGetRole* der Klasse *databaseCon* aus der Datenbank abgefragt. Diese Rollen werden mit Hilfe der Rollenkarten auf dem Display des Diebs dargestellt. Wählt er eine aus, so wird ihm die Rollenbeschreibung in einem PopUpChoice−Fenster angezeigt, welches mit Hilfe der Klasse *popup* erstellt wurde und er kann wählen, ob er sich für oder gegen diese Rolle entscheiden will. Sind beide wählbaren Rollen Sonderrollen, so bekommt der Dieb auch die Möglichkeit durch einen Extrabutton, dass er ein einfacher Dorfbewohner bleiben kann. Ist eine mögliche Wahl ein Werwolf, so hat der Dieb keine andere Wahl, als ein Werwolf zu werden. Es erscheint somit, bevor der Dieb eine Wahl treffen kann, ein PopUp−Fenster der Klasse *popup*, welches ihm mitteilt, dass er von nun an als Werwolf weiter spielen wird. Hat der Dieb seine Wahl getroffen oder das "Werwolf−Info−PopUp" geschlossen, so wird die *DiebDB.java* ausgeführt. Nachdem sich der Spieler seine Entscheidung getroffen hat, wird die Auswahl in den globalen Variablen sowie in der Datenbank gespeichert. Anschließend wird die nächste Phase wird aufgerufen.

HexeActivity. Zuerst bekommt die Hexe angezeigt, wer das Opfer der Werwölfe ist. Sollte sie noch einen Heiltrank zur Verfügung haben, hat sie die Wahl, ob sie das Opfer retten möchte. Dies wird mithilfe der *databaseCon* in der Datenbank gespeichert. Sollte sie noch einen Gifttrank zur Verfügung haben, wird gefragt, ob sie diesen verwenden möchte. Wenn sie sich dafür entscheidet, darf sie einen Spieler auswählen, der sterben soll. Andernfalls wird die nächste Phase aufgerufen. Auch diese Wahl wird mithilfe der *databaseCon* in der Datenbank gespeichert.

JaegerActivity. Diese Activity wird aufgerufen, sobald der Jaeger gestorben ist. Über ein PopUp wird er aufgefordert einen Spieler auszuwählen, der mit ihm sterben soll. zur Auswahl werden ihm alle Spieler angezeigt. Sobald er ein Wahl getroffen hat und diese mit OK bestätigt, wird die *JaegerDB* aufgerufen. Allen anderen Spielern wird derweil ein Infotext angezeigt und

mit "timerHandler.postDelayed(timerRunnable, 2000);" warten sie auf die nächste Phase.

LoverActivity. Alle Spieler werden durch ein Audio aufgefordert die Augen zu öffnen, um auf ihrem Display zu sehen, ob sie in eine andere Person verliebt sind. Das Audio fordert sie auch auf wieder einzuschlafen. Im Anschluss wird die nächste Phase aufgerufen.

SeherinActivity. Die Seherin wird durch ein PopUp−Info−Fenster der Klasse *popup* aufgefordert, einen Spieler zu wählen, dessen Gesinnung er erfahren möchte. Die Wahl erfolgt über durch die Methode *createObjects* der Klasse *GameActivity* erstellte Spieler−Icons. Wählt der Spieler einen anderen Spieler aus, so erscheint ein PopUp−Info−Fenster der Klasse *popup*, welches dem Spieler mitteilt, ob der von ihm gewählte Spieler gut (Dorfbewohner oder Sonderrolle) oder böse (Werwolf) ist. Diese Informationen werden mit Hilfe der Methode *getIdentity*, welche die Methode Seherin der Klasse *databaseCon* aufruft, erhalten. Bestätigt der Spieler die Informationen mit Betätigung des "OK"−Buttons des PopUp−Info−Fensters, so wird *setNextPhase* aufgerufen. Anschließend wird die nächste Phase aufgerufen.

showVictimActivity. In dieser Activity werden die Opfer angezeigt. Mögliche Gründe dafür sind die Abstimmung der Dorfbewohner am Tag, die Wahl der Werwölfe in der Nacht, der Gifttrank der Hexe, der Schuss des Jägers oder ein gestorbener Liebender. Nach einer gewissen Zeit wird automatisch die nächste Phase aufgerufen.

TagActivity. Am Tag erwachen alle Spieler, die noch am Leben sind. Sie können ihre Stimme für denjenigen abgeben, den sie töten möchten. Ihre Auswahl wird in der Datenbank gespeichert. Anschließend warten das Gerät auf ein Ergebnis der Abstimmung. Das Gerät des Spielleiters kontrolliert dabei, ob alle Spieler ihre Stimme abgegeben haben. Sollte das der Fall sein, wird der Spieler mit den meisten Stimmen als *victimDor* in der Datenbank gespeichert

WerwolfActivity. In dieser Phase erwachen die Werwölfe und wählen per Klick auf einen Button ihr Opfer. Die Auswahl wird in der Datenbank vermerkt. Das Gerät des Spielleiters kontrolliert regelmäßig, ob alle Wölfe ihre Stimme abgegeben haben. Sollte dies der Fall sein, wird der Spieler mit den meisten Stimmen als *opferWer* in der Datenbank gespeichert. Anschließend wird die nächste Phase aufgerufen.

GameActivity. Diese Activity ist die Grundlage für alle kommenden Spielphasen. Die *createObjects*−Methode ist für die Erstellung aller benötigten Darstellungselemente zuständig. Das Display wird zunächst in vier Layouts geteilt, die nach und nach (je nach Anzahl der Spieler) mit Player−Buttons befüllt werden. Durch einen langen Klick, kann das dazugehörige Spieler−Bild angezeigt werden. Spieler−Buttons von toten Spielern sind nicht anwählbar. Die Methode *playerSelected* wird von einem Button bei einem aufgerufen und ist dafür zuständig die aktuelle Auswahl visuell darzustellen. Nach dem erstellen der benötigten Elemente wird *getCurrentPhase* aufgerufen. Diese Methode überprüft, ob eine neue Phase aktiv geworden ist und ruft die dementsprechende Activity auf. Das Spielleitergerät kündigt die jeweils folgende Phase an. Zu Beginn einer "Rollen"−Activity wird kontrolliert, ob der entsprechende Spieler erwachen soll oder nicht. Die Displays der Spieler mit den entsprechenden Rollen gehen an und Aktionen können ausgeführt werden. Sollte eine einzeln aufgerufene Rolle bereits gestorben sein, passiert nichts und nach einer gewissen Zeit wird automatisch in die nächste Phase geschaltet.

GameOverActivity. Sobald eine Bedingung für das Ende des Spiels erfüllt ist (siehe Punkt 1 "Ende des Spiels") wird diese Activity aufgerufen. Über ein PopUp wird jedem Spieler angezeigt, wer gewonnen hat (Text abhängig von Rolle und Gewinner−Team). Mit dem Klick auf den ENDE−Button gelangt man zurück ins Menü. Dabei werden alle Spiel−spezifischen Daten aus der Datenbank gelöscht.

GameSetupActivity. Diese Activity lässt den Spieler die Einstellungen für das zu erstellende Spiel treffen. Es gibt einen *NumberPicker* für die Auswahl der Anzahl der Spieler. Außerdem existiert ein *Spinner*, der automatisch die benötigte Anzahl an Werwölfen mithilfe der Funktion *setRecommendedNumberOfWer(int players)* berechnet. Es steht dem Spieler frei die Anzahl im Nachhinein zu verändern. Im Folgenden können die Extrarollen an− bzw. abgewählt werden. Bei jeder Änderung erfolgt ein Aufruf der Funktion *calculateGame*. Diese berechnet anhand der Anzahl der Werwölfe und der Anzahl der Extrarollen die benötigte Anzahl an Dorfbewohnern für das Spiel und setzt diese automatisch. Alle teilnehmenden Rollen werden in das *cards*−Array geschrieben, welches anschließend gemischt wird und das *cardsShuffled*−Array entsteht. Die für das Spiel benötigten Phasen werden gesammelt und die *createGameDB()* wird ausgeführt. Die Phasen sowie die

Rollen werden in die Datenbank geschrieben. Nach erfolgreicher Erstellung des Spiels in der Datenbank wird die *QRCodeActivity* aufgerufen. Das Gerät des Spieler, der das Spiel erstellt hat, wird zum "Spielleiter"−Gerät.

GetRole. Mit dem Aufruf dieser Activity wird die der Spielerrolle entsprechenden Karte geladen und kann durch eine Betätigung des Kartensymbols angezeigt werden. Bei einer weiteren Betätigung dieses wird dem Spieler die seiner Rolle entsprechenden Rollenbeschreibung angezeigt. Berührt er die Karte erneut, so wird wieder nur die Kartenrückseite angezeigt (die Rolle bleibt verborgen). Ist der Spieler bereit, so betätigt er des "Bereit"−Button. Sobald alle Spieler bereit sind, wird die *LetsPlayActivity* aufgerufen.

JoinGameActivity. In dieser Activity kann der Spieler einen, auf einem anderen Handy erzeugten, QR-Code scannen. Dieser codiert die entsprechende *GameId* für das Spiel, dem beigetreten werden soll. Mithilfe dieser Information wird in der Datenbank der Spieler dem Spiel hinzugefügt. Anschließend wird *GetRole* aufgerufen.

LetsPlayActivity. Diese Activity dient als Überleitung zum eigentlichen Spiel. Hierbei wird ein Audio von dem Spielleitergerät abgespielt, welches eine kurze Einleitung gibt und die Spieler auffordert die Augen zu schließen. Außerdem werden durch die Aufrufe der Methoden *getPlayerIDs*, *getPlayerNames* und *getImagesAsString* der Klasse *databaseCon* die im Spiel befindlichen *PlayerIDs* und zugehörigen Namen sowie Bilder aus der Datenbank geladen und anschließend global gespeichert. Somit stehen diese zu späteren Zeitpunkten zur Verfügung und müssen nicht bei Gebrauch von der Datenbank abgefragt werden. Anschließend wird die *GameActivity* aufgerufen.

LoginRegistrationActivity. Jeder neue Spieler muss sich einen Account erstellen. Dazu wird ein neuer Eintrag in der Datenbank mit *playerID*, *name*, *username*, *password* und *image* angelegt. Die Registrierung erfolgt mithilfe der Klasse *databaseCon* über die Methode *registration*. Bereits registrierte User können sich einfach einloggen. Der Benutzername und das Passwort werden mithilfe der Methode *login* verifiziert. Anschließend wird die *MenuActivity* gestartet.

MenuActivity. In dieser Activity kann sich der Spieler entscheiden, ob er ein neues Spiel starten möchte, welches die *GameSetupActivity* startet. Andernfalls kann er einem Spiel beitreten, mittels der *JoinGameActivity*, oder er öffnet die Einstellungen, wobei die *SettingsActivity* geöffnet wird.

Ebenfalls möglich ist das Betrachten der Regeln *Rules*.

QRCodeActivity. Diese Activity erzeugt aus der *gameID* des erstellten Spiels einen QR-Code der von anderen Spielern gescannt werden muss, um dem Spiel beizutreten. Erst, wenn alle Spieler dem Spiel beigetreten sind, wird die *LetsPlayActivity* auf dem Spielleitergerät aufgerufen. Wird die "Zurücktaste" in dieser Activity betätigt, erscheint ein PopUp−Fenster (Aufruf *popup*) bei welchem bestätigt werden muss, dass ins Menü zurückgekehrt werden soll. Wird diese Bestätigung vorgenommen, so wird die Methode *resetOneGame* der Klasse *databaseCon* aufgerufen. Im Zuge dessen wird das bereits erstellte Spiel wieder aus der Datenbank gelöscht (aus den Tabellen "player_game", "_GAME" und "_PHASES") und die *MenuActivity* wird aufgerufen.

RulesActivity. Diese Activity zeigt die Spielregeln und Rollenbeschreibungen an.

SettingsActivity. In der *SettingsActivity* hat der Spieler die Möglichkeit sein Bild zu ändern. Die Änderung erfolgt mittels der Funktion *setImage* der Klasse *databaseCon*. Außerdem kann er seinen Account löschen.

Classes Audio. Diese Klasse kümmert sich um das Abspielen des richtigen Audios zur richtigen Zeit.

databaseCon. Diese Klasse enthält verschiedene Funktionen, die für die Kommunikation bzw. den Datenaustausch mit der Datenbank zuständig sind.

GlobalVariables. Die Klasse *GlobalVariables* enthält diverse globale Variablen, sowie deren *Getter* und *Setter*, um eine Parameterübergabe zwischen Activitys und java−Klassen zu vereinfachen.

Global variables: Game

ownPlayerID
→ speichert eigene PlayerID (für Login, Rollenabfrage, Spiel)

ownRole
→ speichert eigene Rolle

gameID
→ speichert aktuelle GameID

numPlayers
→ speichert Spieleranzahl des aktuellen Spiels

numPlayersAlive
→ speichert aktuelle Anzahl an "lebenden" Spielern des aktuellen Spiels

PlayerIDs
→ Array aller PlayerIDs des Spiels

PlayerNames
→ Array aller Spielernamen des Spiels

cards
→ Array der gemischten Rollenkarten des aktuellen Spiels --→ wird nur vom Spielleiter gebraucht --→ nötig zur Spielerstellung (Einfügen des Spiels in die Datenbank)

currentlySelectedPlayer
→ Button des aktuell ausgewählten Spielers

Phases
→ Array, das alle Spiel−relevanten Phasen enthält --→ wird nur vom Spielleiter gebraucht --→ nötig zur Spielerstellung

currentPhase
→ String, der aktuelle Phase wiedergibt --→ überwiegend zur Aktionsentscheidung in der popup-Klasse nötig

nextPhase
→ String, der nächste Phase wiedergibt --→ nötig für Audios

currentContext
→ speichert aktuelle Activity --→ ermöglicht den Aufruf einer Activity aus einer java-Klasse heraus

sharedPrefContext
→ speichert Activity, in der sharedPreferences angewandt wird (nötig für Logout)

winner
→ String, der den Gewinner enthält

spielleiter

→ Boolean, welcher angibt, ob das Gerät Spielleitergerät ist, oder nicht --→ nötig zum Abspielen der Audios Aufruf der killDB

images

→ String−Array, welches die Base64−Strings der Bilder der Spieler des aktuellen Spiels speichert.

Global variables: Dieb

DiebChoosen

→ Boolean, welcher angibt, ob der Dieb als Sonderrolle ausgewählt wurde --→ denn wenn ja, müssen zwei weitere Karten ins *cards*−Array aufgenommen werden + wenn der Dieb ausgewählt wurde und es nur einen Werwolf gibt, darf dieser beim Mischen nicht eine der beiden letzten Karten werden (*GameSetupActivity* --→ *calculateGame* + in der *QRCodeActivity* wird ermittelt, wie viele Spieler dem Spiel bereits beigetreten sind (mit der Hilfe der Methode *getPlayerInGame* der Klasse *databaseCon*) somit muss bekannt sein, dass die letzten beiden Einträge frei bleiben werden.

Global variables: Amor

lover1

→ speichert den Button des Spielers, welchen Amor als lover1 ausgewählt hat

lover2

→ speichert den Button des Spielers, welchen Amor als lover2 ausgewählt hat

OK

→ Button, welcher von Amor betätigt wird, wenn er seine Wahl getroffen hat -¿ global, da der Button erst auswählbar wird, wenn Amor genau zwei Spieler ausgewählt hat. Die auf die Button angewendeten Methoden finden allerdings in der GameActivity statt.

Global variables: Seherin

PopUpSeherinIdentity

→ AlertDialog, welcher das PopUp−Info−Fenster darstellt, welches die Gesinnung eines Spielers zeigt. Da die Fenster nur in der *On-Create*−Methode einer Activity erstellt werden können, es in diesem Fall jedoch in der *getIdentity*−Methode der *SeherinActivity* erst den entsprechenden Text erhält, muss es global definiert werden.

Global variables: Jäger

JaegerDies

→ Boolean, welcher zu Beginn eines jeden Spiels auf "false" gesetzt wird. Er zeigt an, ob der Jäger ein Opfer der Nacht des Tages war --→ ist dies der Fall, so wird in der *showVictimActivity* noch nicht die *killDB* nach Ablaufen der für die Anzeige vorgesehenen Zeit aufgerufen, sondern erst die *JaegerActivity* aufgerufen. Des weiteren wird die Variable erneut in der *showVictimActivity* gebraucht, um die Opfer der Nacht des Tages und dasdie Opfer des Jägers entsprechend anzeigen zu können (es muss bekannt sein, welche Textfelder angezeigt werden sollen − so gibt es eines speziell für dasdie Jägeropfer)

victimJaeger

→ Boolean, der das Opfer des Jägers beinhaltet. Wird genutzt, damit die entsprechende Aktion nur ausgeführt wird, wenn der Jäger ein Opfer gewählt hat.

JSONParser.

popup.

Database AmorDB. Diese Klasse wird als AsyncTask ausgeführt. Dabei wird eine Datenbankaktualisierung mit Hilfe des PHP−Files *setLovers*, der Methode "POST" und den *params* lover1(*params*[0]), lover2(*params*[1]) durchgeführt. Im Zuge dessen werden in der "player_game"−Datenbanktabelle bei den Spielern, deren playerIDs lover1 bzw. lover2 entsprechen, in der Spalte "lover" die PlayerID des jeweils andere eingetragen.

createGameDB. Mithilfe des "create_new_game.php" erstellt diese Klasse ein neues Spiel in der Datenbank. Dabei fügt sie ein neues Spiel in die

_GAME−Tabelle ein und erstellt dazu Einträge in der player_game−Tabelle [2.2.3 Fig. 2]. Des weiteren werden die Phasen des Spiels angelegt.

DiebDB. Diese Klasse wird als AsyncTask ausgeführt. Dabei wird ein http-request mit den Parametern gameID, playerID, der neuen Rolle (newRole --→ aus params[0]), der Rolle, die nicht gewählt wurde (notChoosen --→ aus params[1]]) und, wenn der Dieb ein Dorfbewohner bleiben möchte, die zweite Rolle, die nicht gewählt wurde (nothingChoosen --→ aus params[2]), der Methode "POST" und unter der Nutzung des PHP−Files "changeRole.php" an die Datenbank geschickt. Im Zuge dessen wird die Rolle des Diebs entsprechend seiner Wahl geädert, die überflüssigen Rollen werden aus der "player_game"−Tabelle gelöscht und, wenn nötig, wird die "_PHASES"−Datenbanktabelle geändert. Dies geschieht, wenn der Dieb sich gegen eine Sonderrolle entscheidet. Sollte dies der Fall sein, so wird die entsprechende Phase gelöscht. Nach Abschluss der Datenbankkommunikation wird die Klasse *setNextPhase* aufgerufen.

GameOverDB. Diese Klasse überprüft regelmäßig, ob eine Bedingung für das Ende des Spiels erfüllt ist. Wenn ja, ändert sie dementsprechend die globale Variable *winner*.

getCurrentPhase. Diese Klasse wird als AsyncTask ausgeführt. Jedes Gerät, das gerade nicht "aktiv" ist, d.h. die Geräte der Spieler die ihre Augen geschlossen haben, erfragen periodisch mithilfe dieser Klasse die aktuelle Phase. Die aktuelle Phase wird einmal global und einmal in der Datenbank gespeichert. Unterscheiden sich diese Phasen, weiß das Gerät, dass sich die Phase geändert hat und schaltet zur nächsten Activity Phase weiter.

HexeDB Wenn die Hexe sich dazu entschließt einen ihrer Tränke zu verwenden, wird dies durch diese Klasse in der Datenbank gespeichert *updateHexe.php*.

JaegerDB. Diese Klasse speichert das Opfer des Jägers in der Datenbank. (*setVictims.php*)

joinGameDB. Sobald ein Spieler den QR-Code gescannt hat, wird er mithilfe dieser Klasse bei der passenden gameID in die player_game−Tabelle eingefügt (*insert_player.php*).

killDB. In dieser Klasse wird der *alive*−Status der aktuellen Opfer auf 0

gesetzt und die *victim*−Einträge in der _GAME−Tabelle werden auf NULL (bzw. 0) zurückgesetzt. (*changeAlive.php*).

NextPhaseDB. Diese Klasse holt sich aus der Datenbank zu einer bestimmten Phase den *nextPhase*−Eintrag und ändert dementsprechend die globale Variable *nextPhase*.

setNextPhase. Am Ende jeder Phase wird vom gerade "aktiven" Gerät die Klasse *setNextPhase* aufgerufen. Diese ändert die aktuelle Phase in der Datenbank.

setReadyDB. Diese Klasse ändert den *ready*−Eintrag eines Spielers in der Datenbank (in der player_game−Tabelle), sobald dieser in der *GetRole* auf den *bereit*−Button gedrückt hat.

showVictimDB.

php-Files

2.2.3 Datenbankschema

Beschreibung der Tabellen in Fig.1:

player_game. Diese Tabelle ordnet einem Spieler ein Spiel zu und speichert, welche Rolle dieser verkörpert (*role*). Der *ready*−Wert ist für den Beginn des Spiels gedacht. Sobald alle Spieler dem Spiel beigetreten und bereit sind (*ready == 1*) beginnt das Spiel. Bei den Abstimmungen wird in der *numOfVotes* gespeichert, wie oft für einen Spieler abgestimmt wurde.

$\_GAME$. In dieser Tabelle werden Spiel-spezifische Informationen gespeichert. Sie enthält den Tränke-Vorrat der Hexe (*poison, heal*) und die aktuell zum Tode verurteilten (*victimDor, victimWer, ...*).

$\_PHASES$. Hier werden zu jedem Spiel die notwendigen Spielphasen gespeichert. Abhängig von der Auswahl der Rollen werden diese zu Beginn des Spiels festgelegt. Die Tabelle zeigt die jeweiligen Phasen (*phases*) mit ihren darauf folgenden Phasen (*nextPhase*). Der Wert *currentPhase* gibt an, in welcher Phase sich ein Spiel befindet.

$\_PLAYER$. Diese Tabelle enthält alle Spieler-Accounts. Der *login*-Wert gibt an, ob ein Spieler auf einem Gerät angemeldet ist. So wird sicher gestellt, dass jeder Spieler nur auf einem Gerät angemeldet sein kann.

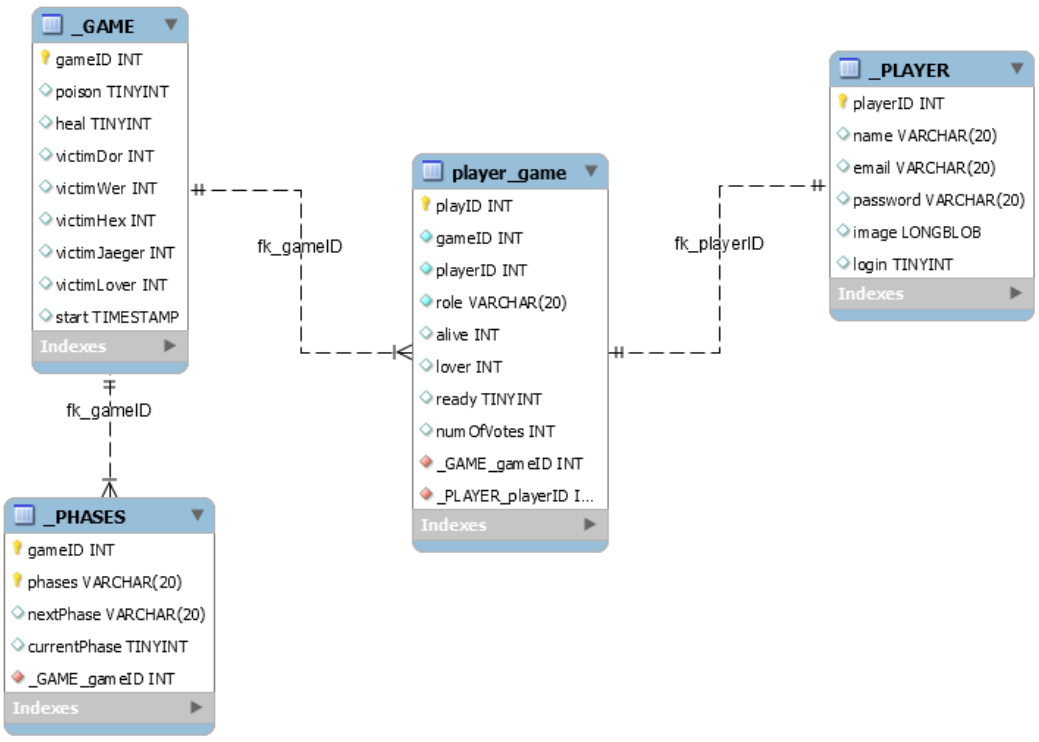


Figure 1: Datenbankschema