

# Entwicklung eines Gamification-basierten Unterstützungs- und Motivationsgeräts zur Rehabilitation von Schlaganfall-Patienten

Einreichung zum  
**ALBERT-SCHWEITZER-SCHULPREIS 2019**

Lukas Rost

Fachbetreuer: Johannes Süpke  
Seminarfachbetreuer: Dr. Marion Moor  
Außenbetreuer: Hannes Weichel

8. März 2019  
Erfurt

## **Inhaltsverzeichnis**

<b>1 Einleitung</b>	<b>2</b>
<b>2 Die Erkrankung Schlaganfall und geeignete Therapiemethoden</b>	<b>3</b>
2.1 Schlaganfall als Krankheitsbild . . . . .	3
2.2 Rehabilitation und Effektivität von Bewegungsübungen . . . . .	4
<b>3 Umgesetzte Konzepte</b>	<b>5</b>
3.1 Gamification . . . . .	5
3.1.1 Grundlegende Mechanismen der Gamification . . . . .	5
3.1.2 Beispiele für die motivationssteigernde Wirkung . . . . .	6
3.2 Biofeedback . . . . .	7
<b>4 Schaltung und Implementierung des Mikrocontroller-Systems</b>	<b>8</b>
4.1 Aufbau der Schaltung . . . . .	8
4.2 Entwicklung des Programms auf dem Mikrocontroller . . . . .	8
<b>5 Entwicklung der Begleitapp für Android</b>	<b>10</b>
5.1 Grundlegender Aufbau und Konzept der App . . . . .	10
5.2 Kommunikation mit dem Mikrocontroller . . . . .	11
5.3 Konzept und programmiertechnische Umsetzung des Minispiele . . . . .	11
5.4 Umsetzung der Gamification in der App . . . . .	13
5.5 Funktionsweise des Benachrichtigungssystems . . . . .	15
<b>6 Zusammenfassung</b>	<b>16</b>
<b>7 Literatur- und Quellenverzeichnis</b>	<b>17</b>
<b>8 Anhang</b>	<b>20</b>

# 1 Einleitung

„[Der Schlaganfall ist] nach Herzerkrankungen und Krebsleiden [...] die dritthäufigste Todesursache in Deutschland.“<sup>13</sup> So gab es im Jahr 2015 rund 40000 Todesfälle durch einen Schlaganfall oder seine Folgen.<sup>30</sup> Doch viele der Betroffenen überleben glücklicherweise, tragen dabei jedoch Langzeitbehinderungen davon. So ist er sogar die häufigste Ursache für diese.

Für die davon betroffenen Patienten bedeutet das langwierige Bewegungsübungen, bei denen der Erfolg noch nicht einmal sicher ist. In Studien wurde beispielsweise festgestellt, dass nur 5 Prozent der Patienten nach einer solchen Therapie wieder in der Lage waren, ihre Arme und Hände uneingeschränkt einzusetzen, während bei 20 Prozent der Patienten keine Hand- und Armfunktion zurückkehrte.<sup>27</sup>

Um eine solche Rehabilitation so erfolgreich wie möglich zu gestalten, existieren verschiedene Methoden, die in dieser Arbeit auch kurz vorgestellt werden sollen. Ihnen ist jedoch eines gemeinsam: Sie legen wenig Wert darauf, den Patienten zu unterstützen, zu motivieren und möglicherweise den Heilungsprozess durch solche positiven Wirkungen zu vereinfachen und zu beschleunigen. Dabei gibt es Methoden, die in dieser Hinsicht vielversprechend klingen, zum Beispiel die Gamification, auch Spielifizierung genannt. Bei dieser werden spieltypische Elemente in fremde Kontexte wie hier eine Therapie integriert, wodurch motivierende Wirkungen erzielt werden können.

Ausgehend von dieser Überlegung soll in dieser Arbeit ein Gerät entwickelt werden, das bei der Therapie einer Armlähmung unterstützend wirken kann. Es soll Patienten motivieren, unterstützen und gegebenenfalls Hinweise bezüglich des Behandlungsfortschritts geben. Das Gerät benutzt dazu einen Elektromyografie-Sensor, der mittels Oberflächenelektroden die Kontraktion der Armmuskeln messen kann. Über Bluetooth-Funk werden diese Sensordaten an ein Smartphone übertragen, das der Patient mit einer zugehörigen Begleitapp benutzen kann.

Diese App nutzt Methoden der Gamification wie Erfahrungspunkte oder Aufgaben („Quests“), um die Motivation des Patienten bei solch monotonen Übungen zu steigern. Bezogen auf den Aufbau soll sie ähnliche wie eine konventionelle Fitness-App die Möglichkeit bieten, Übungen durchzuführen und auszuwerten. Nicht zuletzt soll sie es auch erlauben, ein Minispiel zu spielen, welches durch Armbewegungen gesteuert werden kann.

Um die Motivation des Patienten über längere Zeit zu erhalten, sollen Benachrichtigungen den Nutzer, falls gewünscht, täglich an seine Übungen erinnern. Doch da die beste technische Lösung ohne die Einbindung in eine medizinisch anerkannte Therapie sinnlos ist, soll diese Arbeit auch auf Möglichkeiten dazu eingehen.

Besonders danken möchte ich meinem Fachbetreuer Herrn Johannes Süpke, der mich im Prozess der Erstellung der Arbeit sowohl bei inhaltlichen und fachlichen Fragen als auch bei der Bereitstellung der nötigen Hardware unterstützte. Weiterhin gilt mein Dank meiner Seminarfachbetreuerin Frau Dr. Marion Moor für die Unterstützung in formalen, rhetorischen und sprachlichen Fragen.

Außerdem möchte ich meinem Außenbetreuer Herrn Hannes Weichel danken, der mich mit wichtigen Vorschlägen zu Thema und Konzept der Arbeit sowie mit der Bereitstellung eines Mikrocontrollers und weiterer Materialien unterstützte. Nicht unerwähnt bleiben sollen jedoch auch Herr Frank Paulig und das SFZ Erfurt sowie Herr Udo Weitz und der Sponsorpool Thüringen des Wettbewerbs „Jugend forscht“, die die nötigen finanziellen Mittel für die Beschaffung noch nicht vorhandener Hardware bereitstellten.

## 2 Die Erkrankung Schlaganfall und geeignete Therapiemethoden

### 2.1 Schlaganfall als Krankheitsbild

Beim Schlaganfall, auch *Apoplexia cerebri* genannt, handelt es sich allgemein um eine „plötzliche Durchblutungsstörung im Gehirn.“<sup>21</sup> Durch diese kommt es zu einem regionalen Mangel an Sauerstoff und Nährstoffen, welcher zu einem Absterben von Gehirngewebe führt.

Es existieren zwei mögliche Ursachen: Der ischämische oder Hirninfarkt tritt bei 80 bis 85 % der Fälle auf. In diesem Fall ergibt sich eine mangelnde Durchblutung aufgrund von Gefäßverschlüssen, auch unter dem Begriff Arteriosklerose oder Thrombose zusammengefasst. Folgend kann es dabei zusätzlich auch zu einem Schlaganfall der zweiten Art kommen, dem sogenannten hämorrhagischen Infarkt bzw. der Hirnblutung, die 10 bis 15 % der Fälle zugrundeliegt. Dieser wird durch geplatzte und eingerissene Gefäße verursacht, aus denen Blut ins Hirngewebe austritt. Dieses schädigt durch eine verminderte Sauerstoffversorgung, seinen Druck sowie seine neurotoxische Wirkung das Gehirn. Ein solcher Infarkt kann seinerseits wiederum eine Ischämie verursachen.<sup>13</sup>

Im Vorfeld eines Schlaganfalls treten oft transitorisch-ischämische Attacken, also vorübergehende neurologische Ausfälle, auf. Symptome eines Schlaganfalls reichen von halbseitigen Körperlähmungen über Sprachstörungen und eingeschränktes Sprachverständnis (motorische und sensorische Aphasie) bis zu Sehstörungen und Gleichgewichtsproblemen. Auch Verwirrtheit, Übelkeit und Kopfschmerzen können auftreten.<sup>13</sup>

Zur Erkennung von Schlaganfällen wird meist der FAST-Test benutzt, der fachsprachlich unter dem Begriff „Cincinnati Prehospital Stroke Scale“ bekannt ist.<sup>19</sup> Dieser besteht aus folgenden Punkten:

1. Face/Gesicht: Person kann nur mit einer Gesichtshälfte lächeln
2. Arms/Arme: Unfähigkeit, beide Arme mit nach oben geöffneten Handflächen nach vorne zu strecken
3. Speech/Sprache: undeutliche Aussprache, kann nicht sprechen, versteht nichts mehr

Sollten diese Punkte zutreffen, zählt der vierte Punkt: **T**ime (Zeit). Der Rettungsdienst sollte umgehend verständigt werden, um den Schaden zu begrenzen, denn „Zeit ist Hirn“.<sup>13†</sup> Die Diagnostik erfolgt in einer geeigneten Klinik mit „Stroke Unit“. Dabei werden meist Verfahren wie Computertomografie und Magnetresonanztomografie genutzt. Die Schwere des Schlaganfalls wird mit Scoresystemen wie der „National Institutes of Health Stroke Scale“ beurteilt.

Risikofaktoren umfassen unter anderem Bluthochdruck, Rauchen, Diabetes sowie Übergewicht und Bewegungsmangel. Dementsprechend wirkt eine gesunde Lebensweise präventiv. Dazu gehören gesunde Ernährung und regelmäßige Bewegung sowie ausreichende Flüssigkeitsaufnahme, aber auch Stressvermeidung.<sup>13</sup> Es existieren jedoch auch andere, nicht beeinflussbare Risikofaktoren wie Alter, Blutgruppe und genetische Veranlagung.

Zu den Basismaßnahmen bei der Therapie gehört als erstes die Stabilisierung der Vitalfunktionen wie Blutdruck, Puls und Körpertemperatur. Auch sollte der Patient mit erhöhtem Oberkörper gelagert werden. Diese Maßnahmen stabilisieren den Patienten und dienen der Verhinderung eines weiteren Schlaganfalls. Weiterhin kann bei einem ischämischen Infarkt bis zu drei Stunden nach Auftreten des Schlaganfalls abhängig von der Größe des Infarkts eine intravenöse Thrombolyse-Therapie durchgeführt werden, um eventuell verschlossene Blutgefäße wieder zu öffnen. Bei Hirnblutungen dagegen sind operative Behandlungen sinnvoll, beispielsweise zur Hirndruck-Entlastung.<sup>13</sup>

---

<sup>†</sup>siehe Abb. 2, S. 20

## **2.2 Rehabilitation und Effektivität von Bewegungsübungen**

Armlähmungen zählen zu den häufigsten Folgen einer Hirnschädigung, wie sie durch einen Schlaganfall hervorgerufen wird.<sup>7</sup> Aber auch Lähmungen anderer Körperteile oder Sprachstörungen wie Aphasie können auftreten.<sup>19</sup> Meist betrifft eine solche Hirnschädigung nur eine Gehirnhälfte, sodass es nur auf einer Körperseite zu Lähmungen kommt.

Die meisten Rehabilitationsmaßnahmen dienen dazu, die Körperwahrnehmung des Patienten zu fördern und verlorene Fähigkeiten zu kompensieren. Dabei sind Ansätze wie die „Constraint-Induced Movement Therapy“ vielversprechend. Bei dieser auch als „Taubische Bewegungsinduktion“ bekannten Methode wird der gesunde Arm täglich über längere Zeit immobilisiert und der Betroffene somit gezwungen, die erkrankte Hand zu benutzen.<sup>19</sup> So kann ein „erlernter Nichtgebrauch“ verhindert werden.<sup>27</sup>

Ein weiteres bekanntes Rehabilitationskonzept ist das Bobath-Konzept, welches annimmt, dass gesunde Hirnregionen die Aufgaben geschädigter Hirnregionen übernehmen können. Durch das Konzept soll eine entsprechende Vernetzung innerhalb des Gehirns gefördert und die vom Schlaganfall betroffene Körperseite wieder in Bewegungen einbezogen werden.<sup>23</sup> Solche Ansätze erfordern interdisziplinäre Zusammenarbeit. So können beispielsweise mit Physiotherapeuten Gangmuster eingeübt werden, während Ergotherapeuten an der Wiederherstellung der sensomotorischen Fähigkeiten arbeiten und Logopäden mit Sprachtherapie die Aphasie behandeln.<sup>19</sup>

Armlähmungen als solches zeigen sich unter anderem in einer stark beeinträchtigten willentlichen Bewegungsfähigkeit, aber auch durch „erhöhte Muskelanspannung (‘Spastik’) mit einer Fehlstellung des Armes in Ruhe“ sowie der Schwierigkeit, den Arm passiv zu bewegen.<sup>7</sup> Zunächst sollten Armlähmungen medizinisch beurteilt werden, wozu spezielle Verfahren wie der Fugl-Meyer-Test existieren. Auf diese soll hier jedoch nicht genauer eingegangen werden.

Es existieren verschiedene Therapiemethoden mit und ohne Technikeinsatz. Das bilaterale Training beispielsweise besteht darin, dass mit beiden Armen gleichzeitig symmetrische Bewegungen ausgeführt werden, sodass in beiden Armen eine gleichmäßige Bewegungsfähigkeit hergestellt wird. Es wird in verschiedenen Studien neutral bis positiv beurteilt.<sup>7</sup>

Das schädigungsorientierte Training zielt darauf ab, spezifische Behinderungen bei alltäglichen Tätigkeiten zu beheben. Es existieren zwei Formen. Das Arm-Basis-Training beübt alle Bewegungsmöglichkeiten des Arms, also Schulter, Ellenbogen, Handgelenk und Finger. Es ist dabei auf Patienten mit schweren Lähmungen ausgelegt.<sup>7</sup> Das Arm-Fähigkeits-Training dagegen schult verschiedene Formen von Geschicklichkeit und wird bei leichten Lähmungen angewendet.<sup>7</sup> Beide Formen dieses Trainings zeigen einen positiven Effekt.<sup>27</sup> Das aufgabenorientierte Training stellt eine weitere Trainingsform dar, bei der über Bewegungsaufgaben aus dem Alltag die funktionellen Fähigkeiten des Arms wiederhergestellt werden sollen. Es wird neutral beurteilt.<sup>7</sup>

Doch auch ein Technikeinsatz kann bei einer solchen Therapie erfolgen. So existiert beispielsweise die Armrobot-Methode, bei der ein Roboter nicht selbstständig ausführbare Bewegungen mechanisch unterstützt. In Bezug auf die Effektivität wird diese positiv beurteilt.<sup>7</sup> Auch die neuromuskuläre Elektrostimulation ist eine mögliche Therapie, bei der ein Gerät per Elektromyographie Bewegungsversuche des Muskels erkennt und diesen daraufhin elektrisch stimuliert, was zu einer großen Bewegung führt.<sup>7</sup> Die Therapie wurde neutral beurteilt.<sup>27</sup>

Ausgehend von diesen Informationen erschien es sinnvoll, das Gerät als Unterstützung für Arm-Basis- und Arm-Fähigkeits-Training zu konzipieren. Im Zuge einer Erweiterung könnten auch Ansätze wie Armrobot und Elektrostimulation einbezogen werden.

### 3 Umgesetzte Konzepte

#### 3.1 Gamification

##### 3.1.1 Grundlegende Mechanismen der Gamification

Es existieren verschiedene Definitionen für Gamification. Beispielhaft sei hier die Definition nach Breuer zitiert, die besagt, dass es sich bei Gamification um die „Verwendung von spieltypischen Mechaniken außerhalb reiner Spiele, mit dem Ziel, das Verhalten von Menschen zu beeinflussen“<sup>25</sup> handelt. Entsprechend werden Spielkonzepte verwendet, um die Nutzungsmotivation zu steigern und die Nutzer dazu zu bewegen, mehr oder länger mit einem Produkt zu arbeiten als ohne Gamification. Diese Technik wird innerhalb eines Produktes dazu verwendet, dessen Nutzung zu proklamieren.<sup>25</sup>

Indem spieltypische Merkmale außerhalb spielerischer Zusammenhänge verwendet werden, nutzt man den menschlichen Spieltrieb aus. Es werden positive Anreize gesetzt, um Menschen zu einem bestimmten Verhalten anzuregen, während der Nutzer ebenfalls vorhandene negative Anreize wie eine Bestrafung vermeiden will (operante Konditionierung).<sup>20</sup> Es kommt zu einer „Actio-et-Reactio“-Erfahrung.<sup>25</sup> Diese Gestaltung und die entsprechende motivationsssteigernde Wirkung lassen sich mithilfe von psychologischen Theorien erklären, auf welche hier jedoch nicht genauer eingegangen werden soll. Gamification ist an sich ein relativ neues Phänomen und wurde auch erst in letzter Zeit umgesetzt. So wurde der Begriff erst 1978 durch Richard Bartle, einen britischen Informatiker und Computerspiel-Pionier, geschaffen.<sup>20</sup>

Dieser postulierte auch, dass die Wirkung von Anreizen sich bei verschiedenen Menschen unterscheide. Er entwarf deshalb ein Spielertypen-Koordinatensystem, indem er Menschen in vier Kategorien einteilte, wobei ein Mensch mehreren Kategorien zugleich angehören kann.<sup>17</sup> Im Einzelnen beschrieb er folgende Typen, von denen in einem gelungenen Spiel oder bei einer gelungenen Gamification möglichst viele eine passende Motivation finden sollten:

- die Erfolgstypen/Achiever, welche im Spiel nach konkreten Maßstäben möglichst viel erreichen wollen.<sup>17</sup> 10 Prozent aller Menschen werden hauptsächlich diesem Typ zugeordnet.<sup>1</sup>
- die Geselligen/Socializer, denen Kontakte und Interaktionen mit Mitspielern, wie sie auch in klassischen Gesellschaftsspielen vorkommen, besonders wichtig sind.<sup>17</sup> 75 Prozent aller Menschen sind vor allem Socializer.<sup>1</sup>
- die Forscher/Explorer, deren Ziel es ist, möglichst viel zu entdecken und erkunden.<sup>17</sup> Dieser Typ ist bei ca. 10 Prozent aller Menschen am meisten ausgeprägt.<sup>1</sup>
- die Killer, welche Wettbewerb, Wettkampf und Konflikt lieben. Damit sie motiviert sind, müssen andere verlieren und ihnen Respekt erweisen.<sup>17</sup> Dieser Typ macht 5 Prozent aller Menschen aus.<sup>1</sup>

Um Gamification mit Erfolg anwenden zu können, ist es wichtig, die Kriterien und Mechanismen zu kennen, die ein erfolgreiches Spiel hervorbringen. Wie also kann Gamification angewendet werden, um langweilige, frustrierende, monotone oder unbeliebte Tätigkeiten, wie die dieser Arbeit zugrundeliegende Armlähmungs-Therapie, einfacher und motivierender zu gestalten? Zunächst sollten klare Ziele und Regeln aufgestellt werden, anhand derer der Nutzer weiß, welche Rückmeldung er auf eine bestimmte Aktion bekommt. Diese sogenannte Resultatstransparenz führt zu einer gesteigerten Handlungsmotivation.<sup>25</sup> Außerdem sollten immer neue Herausforderungen gewährleistet sein.<sup>31</sup>

Eine motivierende Wirkung erzielt man auch, indem man den Spieler immer oder möglichst leicht gewinnen lässt. Der Spieler sollte beim Spielen in einen „Flow“ kommen, der durch eine starke Fokussierung auf das Spiel gekennzeichnet ist.<sup>1</sup> Dafür darf ein Spiel weder zu einfach

noch zu schwierig sein. Eine Belohnung des Spielers in einem festgelegten Intervall führt zu einer schnellen Abnahme der Motivation. Stattdessen sollte man auf die sogenannte operante Konditionierung setzen, indem man in einem variablen Intervall und in variabler Menge belohnt.<sup>1</sup>

Ein Spiel kann nach dem MDA-Modell durch drei Bestandteile charaktisiert werden: Mechanics (Spielkomponenten und -funktionen), Dynamics (Interaktion zwischen Spieler und Spiel) und Aesthetics (Emotionen, die beim Spieler erzeugt werden).<sup>1</sup> Von diesen kann der Entwickler nur die Mechanics mithilfe spieltypischer Mechanismen beeinflussen. Zu diesen zählen unter anderem:

- ein Punktesystem gemeinsam mit einer einsehbaren Rangliste (Highscore) bei Mehrspieler-Spielen. Mit diesen können Aktionen des Spielers bewertet werden. Ein Highscore erlaubt einfache Vergleiche durch z.B. eine metrische Punkteskala.<sup>25</sup> Highscores können auch suggestiv eingesetzt werden, indem sie den Spieler möglichst in der Mitte der Liste zeigen und somit gleichzeitig belohnen und motivieren.<sup>1</sup>
- ein sichtbarer Status durch Titel oder Badges. Diese repräsentieren nach außen, dass der Spieler ein bestimmtes Ziel erreicht hat und bieten somit Vergleichs- und Wettbewerbsmöglichkeiten.<sup>25</sup> Motivierend sind sie außerdem, da Menschen gerne sammeln. Sie sollten jedoch nicht zu viel eingesetzt werden.<sup>1</sup>
- entdeckbare Aufgaben (Quests), die dem Spiel ein Ziel und Struktur geben und dafür sorgen, dass das Spiel das Interesse des Spielers behält. Wenn Spieler zur Lösung einer Aufgabe mit Mitspielern zusammenarbeiten müssen, kann dies sozial sehr stark motivieren.<sup>1</sup>
- eine Fortschrittsanzeige, die eine dynamische Visualisierung des bisherigen Erfolgs und noch zu erledigender Aufgaben erlaubt.<sup>25</sup>
- gegebenenfalls ein Epic Meaning, also die Arbeit an etwas Erstrebenswertem und an sinnvollen Zielen.<sup>25</sup>

### 3.1.2 Beispiele für die motivationssteigernde Wirkung

Gamification kann erstaunliche Effekte erzeugen. Exemplarisch deutlich wurde das bei einer Reihe von Experimenten der schwedischen Werbeagentur DDB unter dem Namen „The Fun Theory“. Bei einem dieser Experimente wurde eine U-Bahn-Treppe so umgebaut, dass sie wie eine Klaviertastatur bei Bedienung mit den Füßen Töne erzeugte. <sup>§</sup> Im Ergebnis wurde diese Treppe sogar öfter als die danebenliegende Rolltreppe benutzt.<sup>20</sup>

Auch ein Flaschencontainer, der bunt blinkte, wenn Flaschen in die richtige Öffnung geworfen wurden, und der „tiefste Mülleimer der Welt“, der beim Hineinwerfen von Müll einen Pfiff und einen Aufprall ertönen ließ, stellen erfolgreiche Beispiele dar. Im genannten Mülleimer beispielsweise erhöhte sich die Müllmenge drastisch.<sup>20</sup> Der Spieldesigner Kevin Richardson schlug für dieses Experiment eine Radarfallenlotterie vor, bei der alle, die die Geschwindigkeitsbegrenzung einhielten, an einem Gewinnspiel um die Strafzahlungen der Raser teilnahmen. Als dieser Vorschlag umgesetzt wurde, sank die Geschwindigkeit an der kontrollierten Stelle um 20 %<sup>1</sup>.

Im wissenschaftlichen Umfeld kann Gamification ebenfalls die Motivation steigern und so bessere Ergebnisse erzielen. So gelang es beispielsweise einigen Gamern, mithilfe eines Tetris-Nachbaus innerhalb von zehn Tagen die Proteinstruktur des AIDS-Virus zu entschlüsseln, was die Wissenschaftler hinter dem Projekt 15 Jahre gekostet hätte.<sup>28</sup> Unternehmen versuchen ebenfalls, durch Gamification geeignete Mitarbeiter zu finden. So entwarf der Bayer-Konzern eine Karriere-App

---

<sup>§</sup>siehe Abb. 3, S. 20

im Stil von Quizshows und die Management-Simulation „BIMS Online“, die dort nun zur Rekrutierung geeigneter Fachkräfte dienen.

Selbst eine Universität hat Gamification schon ausprobiert: An der Indiana University wurde zeitweise nach einem Experience-Point-System statt nach Noten bewertet.<sup>26</sup> Ein weiteres Beispiel für gelungene Gamification findet sich bei Frage-Antwort-Websites wie Quora, Stack Exchange oder Stack Overflow, bei welchen andere Nutzer entsprechende Punkte für gute Antworten auf bestimmte Fragen verleihen können. Außerdem kann ein Fragesteller die beste Antwort auf seine Frage markieren. All dies führt dazu, dass die Qualität der Antworten meist konstant hoch ist.<sup>1</sup>

Die bei solchen Experimenten gewonnenen Beobachtungen legen zumindest empirisch nahe, dass Gamification eine motivationssteigernde Wirkung besitzt. Unter Wissenschaftlern ist dieses Thema jedoch strittig. Einige Studien belegen diese These, andere widersprechen ihr. So wurde in einem Experiment der TU München versucht, das Berufsfeld der Kommissionierung, welches noch relativ wenig automatisiert ist, spielerischer zu gestalten. Im Ergebnis arbeiteten die Kommissionierer schneller und zufriedener als ohne Gamification.<sup>31</sup>

Bei einer amerikanischen Studie, die sich auf den Erfolg von Fitnessmaßnahmen mit und ohne Wearable (z.B. Smartwatches oder Fitnesstracker) fokussierte, kam man jedoch zu gegenteiligen Ergebnissen. Obwohl Wearables ein klassisches Beispiel für Gamification darstellen, war keine signifikant höhere Gewichtsabnahme zu messen. Die mit Wearable ausgestatteten Probanden verloren im Vergleich zur Kontrollgruppe über den Zeitraum von 24 Monaten meist sogar weniger Gewicht.<sup>5</sup>

### 3.2 Biofeedback

In unserem Körper kommt es durch Regulationsvorgänge ständig zu Veränderungen von messbaren Zustandsgrößen bei biologischen Vorgängen und Körperfunktionen. Diese sind der unmittelbaren Sinneswahrnehmung, also dem Bewusstsein, nicht zugänglich. Sie können jedoch mit technischen und elektronischen Hilfsmitteln beobachtbar gemacht werden, was fachsprachlich unter dem Begriff Biofeedback bekannt ist.<sup>18</sup> Oft wird dieses Verfahren zur Rehabilitation von erlahmten Muskeln, wie z.B. bei einer Armlähmung, eingesetzt.

Da solche Körperfunktionen dem Bewusstsein normalerweise nicht zugänglich sind, können sie auch nicht beeinflusst werden. Durch Biofeedback wird genau das jedoch möglich. Durch die Visualisierung der Messwerte für den Patienten kommt es zu einer Rückkopplung und dieser kann Kontrolle über die Körperfunktion ausüben. Weiterhin können über die operante Konditionierung ganze Reiz-Reaktions-Muster erlernt werden. Insgesamt wird eine Bewusstseinsschärfung für die eigenen inneren Zustände erreicht und die Einflussnahme auf das Nervensystem somit vereinfacht.<sup>18</sup>

Die Messwerte können nicht nur visualisiert werden, sondern werden bei manchen Anwendungen auch als Töne dargestellt. Die oft kleinen und tragbaren Messgeräte, die beim Biofeedback benutzt werden, messen meist nichtinvasiv.<sup>18</sup> Ihre Messergebnisse werden dann in einem Analog-Digital-Wandler konvertiert, gemittelt und verstärkt. Anschließend werden sie per kabelloser Bluetooth-Übertragung auf ein zur Darstellung geeignetes Gerät übertragen.

Grundsätzlich können per Biofeedback viele Größen gemessen werden, wie beispielsweise Atem, Blutdruck, Blutwerte, der Hautwiderstand oder Gehirnströme. Im hier beschriebenen Anwendungsfall ist jedoch die Messung von Muskelpotentialen per Elektromyografie am sinnvollsten, da mit dieser die Beweglichkeit des Arms am besten erfasst werden kann. In vielen Fällen erzielt Biofeedback positive Ergebnisse, sodass es in manchen Fällen sogar eine Alternative zu Medikamenten sein kann.<sup>18</sup>

## 4 Schaltung und Implementierung des Mikrocontroller-Systems

### 4.1 Aufbau der Schaltung

Das Mikrocontroller-System, welches für das Aufnehmen und Übertragen der Messdaten zuständig ist, besteht im Wesentlichen aus vier Bestandteilen, die miteinander verbunden sind:

1. Dem **Grove EMG-Sensor**<sup>29</sup>, der per Elektromyografie Messungen über die Stärke der Armmuskel-Kontraktionen erhebt. Dabei wird die elektrische Muskelaktivität anhand von Potentialänderungen auf der Haut mithilfe von drei Oberflächenelektroden gemessen. Diese Signale werden durch den Sensor verstärkt und gefiltert. Zuletzt gibt dieser eine Spannung im Bereich von 1,5 bis 3,3 Volt aus, wobei eine höhere Spannung höhere Muskelaktivität bedeutet. Dabei wird eine Versorgungsspannung von 3,3 bis 5 Volt benötigt.
2. Dem Mikrocontroller **Atmel ATmega 88PA**<sup>14</sup>, welcher die vom EMG-Sensor gelieferten Daten an den Bluetooth-Chip weitergibt. Es handelt sich hierbei um einen 8-Bit-Mikrocontroller<sup>15</sup>, d.h. es können pro Takt maximal 8 Bit verarbeitet werden. Dieser folgt der sogenannten RISC-Architektur, welche einen reduzierten Befehlssatz im Vergleich zu Standardcomputern besitzt, dafür aber schneller arbeitet. Damit eignet er sich gut für den beabsichtigten Einsatzzweck, bei dem Daten möglichst schnell auf das Mobilgerät übertragen werden sollen.  
Weiterhin setzt dieser Controllertyp die Harvard-Struktur um.<sup>8</sup> Es existieren also getrennte Speicher- und Adressbereiche für Befehle und Daten. Die peripheren Schnittstellen können über Portadressen angesprochen werden. Zu diesen Schnittstellen zählen zwei für die beabsichtigte Anwendung nötige, denn es werden sowohl eine UART-Schnittstelle zur Kommunikation mit dem Bluetooth-Chip als auch ein Analog-Digital-Wandler zum Einlesen der Ausgangsspannung des EMG-Sensors angeboten. Das Programm kann bei diesem Controller über ein Entwicklungsgerät in den Programmspeicher (Flash) geladen werden.
3. Dem Bluetooth-Chip **HC-05**<sup>24</sup>, welcher eine Möglichkeit zur drahtlosen Kommunikation mit dem Android-Mobilgerät über Funk nach dem Bluetooth-Standard bereitstellt. Dabei sind alle zur Kommunikation nötigen Bestandteile auf einem Chip integriert. Der HC-05 wird über die UART-Schnittstelle (Universal Asynchronous Receiver Transmitter) angesprochen, wobei diese eine digitale serielle Schnittstelle zur Datenübertragung realisiert. Der Bluetooth-Chip kommuniziert standardmäßig mit 9600 Baud.
4. Einem **Quarzoszillatoren** mit eingebautem Schwingquarz, in diesem Fall mit der Frequenz 8 MHz, welcher einen genauen Systemtakt für den Mikrocontroller liefert. Dieser wird benötigt, um eine möglichst störungsfreie UART-Kommunikation realisieren zu können. Der Oszillator muss in den Einstellungen (Fuses) des Mikrocontrollers als Taktquelle ausgewählt (CKSEL-Bit) und die interne Teilung des Taktes durch 8 abgeschaltet (CKDIV8-Bit) werden.

Diese Bestandteile wurden nach dem auf der folgenden Seite sichtbaren Schaltplan (Abb. 1) verbunden.

### 4.2 Entwicklung des Programms auf dem Mikrocontroller

Das Programm (siehe Quellcode 1, S. 22) wurde in der Programmiersprache C geschrieben und besteht aus sechs Funktionen, die jedoch auf einige in den C-Standardbibliotheken vorhandene Funktionen und Definitionen für den C-Präprozessor zurückgreifen. Zusätzlich tätigt das Programm auch einige eigene Definitionen für den Präprozessor. Alle Funktionen außer der Main-Funktion, mit der das Programm startet, werden dem Compiler zunächst als Funktionsprototypen mithilfe ihrer Signaturen bekanntgemacht und erst nach der Main-Funktion definiert. Im einzelnen sind folgende Funktionen definiert:

- `init()`. Diese Funktion initialisiert die einzelnen Funktionen des Controllers, indem sie die entsprechenden Einstellungsregister<sup>14</sup> setzt. Der Analog-Digital-Wandler wird aktiviert und dessen interner Taktteiler gesetzt. Die Sende- und die Empfangsfunktion der UART-Schnittstelle werden ebenfalls aktiviert und deren Baudratenfaktor, der angibt, wie schnell kommuniziert werden soll, gesetzt. Der Baudratenfaktor<sup>14</sup> berechnet sich aus (gerundet):

$$UBBR = \frac{f_{CPU}}{16 \cdot r_{BAUD}} - 1$$

Anschließend werden beide Schnittstellen zum ersten Mal ausgelesen, um ihre Funktionsfähigkeit sicherzustellen.

- `u_putchar()`. Hier werden einzelne Zeichen (*Char*) über die UART-Verbindung übertragen. Dabei muss gewartet werden, bis diese Verbindung freigegeben wird.
- `u_puts()`. Diese Funktion überträgt Zeichenketten (*String*) über UART. Hierzu wird über die einzelnen Zeichen mithilfe der in C integrierten Zeigerarithmetik iteriert und jeweils `u_putchar()` aufgerufen.
- `delay()` lässt den Mikrocontroller für die angegebene Zahl an Sekunden warten. Dazu wird die in der Bibliothek `util/delay.h` definierte Funktion `_delay_ms()` benutzt.
- `sendCurrentVoltage()`. In dieser Funktion wird eine Analog-Digital-Wandlung gestartet und nach deren Abschluss das Ergebnis ausgelesen. Dabei handelt es sich um einen relativen Wert im Vergleich zur Referenzspannung,<sup>16</sup> weshalb die eigentliche Spannung aus der Gleichung  $U = \frac{x}{1024} \cdot U_{ref}$  gewonnen wird.\* Die dabei entstandene Gleitkommazahl wird mithilfe der Bibliotheksfunction `sprintf()` in einen String geschrieben, der per `u_puts()` übertragen werden kann.
- `main()`. In dieser Funktion wird zuerst `init()` aufgerufen. Anschließend wird in einer Endlosschleife im Abstand einer halben Sekunde `sendCurrentVoltage()` gestartet und die Spannung an das Mobilgerät übertragen. Es muss sich hier um eine Endlosschleife handeln, da der Mikrocontroller nach der Ausführung von `main()` in ein undefiniertes Verhalten übergeht, aus dem er nur durch Neustart befreit werden kann.

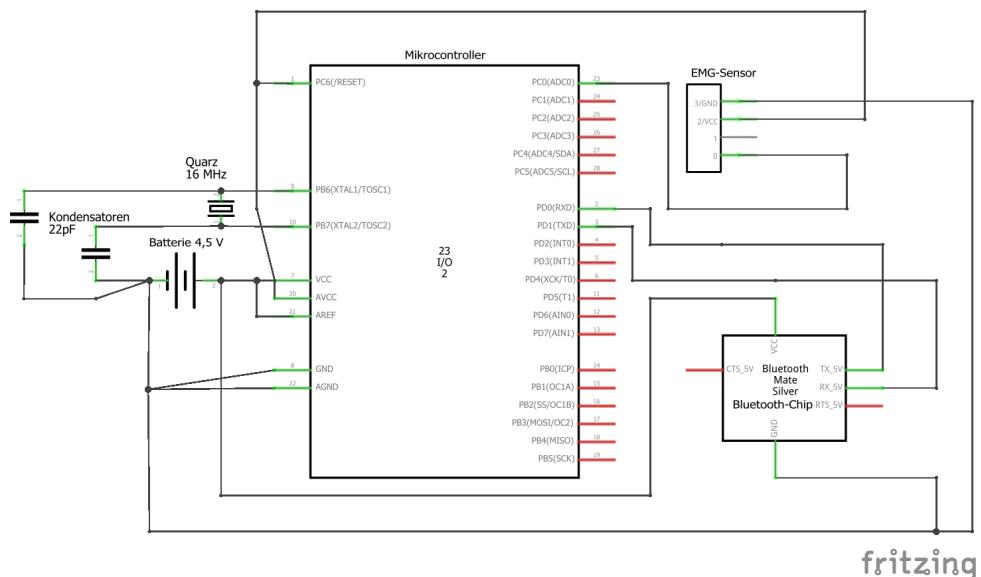


Abbildung 1: Der Schaltplan des Mikrocontrollers. Einige Verbesserungen, um Rauschen aus dem Signal zu entfernen, sind der Einfachheit halber nicht dargestellt.

\*Hierbei ergibt sich jedoch insoweit ein Problem, als dass die Versorgungsspannung bei Batteriebetrieb schwanken kann.

## 5 Entwicklung der Begleitapp für Android

### 5.1 Grundlegender Aufbau und Konzept der App

Um die vom EMG-Sensor gelieferten Daten zu verarbeiten und dem Benutzer bzw. Patienten anschaulich darzustellen, erschien es sinnvoll, eine App für Mobilgeräte zu programmieren. Hierbei fiel die Entscheidung, das weitverbreitete Betriebssystem Android zu verwenden und die App zu diesem (ab Android 6.0) kompatibel zu machen, wobei zur Umsetzung die Programmiersprache Kotlin<sup>4</sup> verwendet wurde. Diese App sollte folgende grundlegende Funktionen enthalten:

- **Durchführung von Bewegungsübungen:** Es sollten einfache Bewegungsübungen möglich sein, bei denen sowohl der aktuelle Messwert als auch die Messwerte im zeitlichen Verlauf angezeigt werden sollten. Dabei sollte auch ein Vergleich mit früheren Übungen möglich sein.
- **Ein durch Bewegungsübungen steuerbares Minispiel:** Die App sollte ein durch den Benutzer mithilfe von Bewegungen steuerbares Minispiel enthalten. Dieses sollte möglichst einfach verständlich sein.
- **Ein verbindendes Gamification-System:** Sowohl bei normalen Übungen als auch beim Minispiel sollten Gamification-Elemente eingebracht werden. So sollte das Sammeln von Erfahrungspunkten (Experience Points, XP) möglich sein und es sollten für besondere Leistungen sogenannte Badges (Abzeichen) vergeben werden können. Dabei sollten die zu erbringenden Leistungen für die Badges in Form von Quests bzw. Aufgaben vorher für den Benutzer sichtbar sein.
- **Erinnerungen an die Übungen:** Die App sollte den Benutzer zu von ihm festgelegten Zeiten durch z.B. eine Benachrichtigung an die Durchführung seiner Übungen erinnern.

Ausgehend davon bot sich eine Gliederung in insgesamt vier für den Anwender sichtbare und miteinander verknüpfte Bildschirmseiten (bei Android Activities<sup>6</sup> genannt) an. Dies sind:

- Eine **Startseite**, die die bisher erreichten Gamification-Erfolge zusammenfasst und kurze Informationstexte sowie Verknüpfungen zu den Übungsmöglichkeiten anbietet. Damit soll der Einstieg möglichst einfach gestaltet werden. Um die einzelnen Bereiche klar voneinander zu unterscheiden, kommt hier das einer Karteikarte ähnelnde Oberflächenelement CardView zum Einsatz.
- Eine **Übungsseite**, auf der man Übungen durchführen kann. Dabei zeigt ein tachoähnliches Oberflächenelement den aktuell festgestellten Messwert an, während mithilfe eines Diagramms die Messwerte während der gesamten Übung dargestellt werden. Diese Übung kann durch den Benutzer beliebig gestartet und beendet werden, während diese Seite auch eine Funktion zur Ansicht der aktuell verfügbaren Quests bereitstellt.
- Eine Seite für das **Minispiel**. Da dieses in einem späteren Abschnitt noch genauer beschrieben wird, soll hier nicht darauf eingegangen werden.
- Eine **Einstellungsseite**. Hier können Einstellungen getroffen werden, die für die restlichen Teile der App von Bedeutung sind. So kann man hier den eigenen Namen einstellen, die Gamification-Datenbank importieren und exportieren und die Übungserinnerungen konfigurieren. Für solche Seiten stellt das Android-SDK die Klasse PreferenceFragment<sup>6</sup> bereit.

Activities werden von Android intern auf dem sogenannten *Back Stack* verwaltet, der basierend auf den abstrakten Datentyp Stack ein Zurückkehren zur vorherigen Activity erlaubt. Allgemein ist beim Arbeiten mit Activities das Lebenszyklus-Modell zu beachten, das beschreibt, auf welche Weise Activities gestartet, pausiert und gestoppt werden können.<sup>6</sup> Die einzelnen Bestandteile der App sollen im folgenden näher erläutert werden. Screenshots der jeweiligen Activities befinden sich im Anhang.

## 5.2 Kommunikation mit dem Mikrocontroller

Wie bereits erläutert, sendet der Mikrocontroller die Messdaten über eine Bluetooth-Verbindung. Um nun mit diesem kommunizieren zu können, muss die Android-App eine solche Verbindung implementieren. Glücklicherweise enthält das Android-SDK (Software Development Kit) bereits eine Softwarebibliothek, die genau dies vereinfacht.<sup>96</sup>

Um eine Verbindung herzustellen, müssen zunächst einige Schritte durchlaufen werden, die nur in den Activities durchgeführt werden können. In dieser App benötigen zwei Activities Bluetooth-Zugriff: die Übungsseite und die Minispiel-Seite. Beide müssen unabhängig voneinander den Code zum Auffinden des zu verbindenden Bluetooth-Geräts implementieren. Dabei müssen beispielsweise die nötigen Berechtigungen überprüft, der Bluetooth-Adapter eingeschaltet und, sofern noch nicht geschehen, das Bluetooth-Gerät gekoppelt werden.<sup>96</sup>

Ist dies geschehen, kann mit dem Einlesen der per Bluetooth über das Protokoll RFCOMM (Radio Frequency Communication)<sup>6</sup> eintreffenden Werte begonnen werden. Diese Funktionalität ist in der App in der Klasse `BluetoothNoService` gekapselt. Um kurzzeitige Schwankungen der Messwerte auszugleichen, bietet es sich an, diese in einer *Queue* zwischenzulagern. Eine Queue oder Warteschlange ist eine in der Informatik häufig eingesetzte Datenstruktur, die nach dem First In - First Out - Prinzip (FIFO) arbeitet, d.h. das Objekt, welches als erstes der Warteschlange hinzugefügt wurde, verlässt sie auch als erstes wieder. Für diesen Zweck eignet sie sich sehr gut, da im Laufe der Zeit immer wieder neue Messwerte hinzukommen, während ältere entfernt werden müssen. Damit keine zu alten Werte verwendet werden, ist es sinnvoll, die Länge der Queue auf 4 Werte zu beschränken.

Zum Hinzufügen der Werte ist es sinnvoll, einen Thread zu implementieren, d.h. eine Funktion, die parallel zum übrigen Programm abläuft. Dieser Thread kann der Queue dann beständig neue Werte hinzufügen, sobald diese eintreffen.

Um nun zu jedem Zeitpunkt einen Durchschnittswert aus der Queue berechnen zu können, bietet sich der Mittelwert der Werte an. Da die Werte jedoch als Spannung abgelegt sind und der EMG-Sensor Werte zwischen  $U_{min} = 1.5V$  und  $U_{max} = 3.3V$  ausgibt, ist es sinnvoll, dem Nutzer die Werte in Prozent des Maximalwertes zu präsentieren. Ein solcher Wert lässt sich mit folgender Gleichung berechnen (wobei  $x$  der Mittelwert der Messwerte ist):

$$p = 100 * \frac{x - U_{min}}{U_{max} - U_{min}}$$

Dieser Wert kann nun an die Benutzeroberfläche zur weiteren Darstellung übergeben werden.

## 5.3 Konzept und programmiertechnische Umsetzung des Minispiele

Das Minispiel soll es erlauben, ein virtuelles Flugzeug über eine Gebirgslandschaft mit Bergen unterschiedlicher Höhe zu steuern. Dabei bestimmt die gemessene Muskelaktivität die Höhe des Flugzeugs. Je höher die Aktivität, desto höher fliegt das Flugzeug. Das Ziel ist es dabei, das Flugzeug möglichst lange fliegen zu lassen, ohne gegen einen Berg zu stoßen. Dieses Spielprinzip ähnelt teilweise dem erfolgreichen Smartphone-Spiel „Flappy Bird“. Es wäre wünschenswert, wenn es gelänge, an den Erfolg des genannten Spielprinzips anzuknüpfen.

Zur Umsetzung des Minispiele wurde die Möglichkeit des Android-SDKs genutzt, eigene Oberflächenelemente (sogenannte *Views*<sup>11</sup>) zu erstellen. Eine solche View beinhaltet in diesem Fall das Spiel und übernimmt dessen Darstellung. Das Android-System fordert dabei, dass man bestimmte, für die eigene View spezifische Methoden überschreibt. Dies sind im einzelnen:

- `onDraw()`: Hier wird der sichtbare Teil der View gezeichnet.<sup>116</sup> Was gezeichnet wird, hängt von der boolesche Objektvariable `inGame` der View ab, welche den Spielzustand

(innerhalb oder außerhalb des Spiels) speichert. Befindet man sich außerhalb des Spiels, wird ein Text angezeigt, der darauf hinweist, dass durch einen Klick das Spiel gestartet werden kann. Innerhalb des Spiels müssen mehr Elemente gezeichnet werden. Zunächst jedoch wird der aktuelle Messwert des `BluetoothNoService` übernommen.

Nun werden anhand dieses Messwerts die Koordinaten der oberen linken Ecke des Flugzeugs bestimmt. Damit kann festgestellt werden, ob das Flugzeug gegen den Berg direkt vor ihm gestoßen ist. In diesem Fall ist das Spiel verloren. Ein entsprechender Infotext wird angezeigt und das Spiel durch die `handleTap()`-Methode beendet.

Ansonsten können der blaue Hintergrund, die grünen Berge sowie das als Icon vorliegende Flugzeug gezeichnet werden.

- `onSizeChanged()` und `onMeasure()`: Diese Methoden werden aufgerufen, wenn das System beispielsweise bei einer Drehung des Bildschirms eine Größenänderung der View anfordert.<sup>11</sup> Entsprechend wird dann eine neue Höhe und Breite festgelegt. Daraufhin können alle von diesen abhängigen Teile der View neu generiert werden. Hier sind dies der Hintergrund und die zufällig (bezüglich ihrer Höhe) generierten Berge.
- `onTouchEvent()`: Diese Methode verarbeitet Touch-Eingaben bzw. Klicks.<sup>12</sup> Android bietet dazu einen sogenannten `GestureDetector` an, der einfache und komplizierte Gesten erkennen kann. In diesem Fall muss jedoch nur das Tippen auf den Bildschirm erkannt und entsprechend das Spiel gestartet oder gestoppt werden. Dies übernimmt die Methode `handleTap()`.

`handleTap()` führt nun einige Schritte aus, die für das Starten bzw. Stoppen des Spiels nötig sind. Die bereits erwähnte `inGame`-Variable wird nun negiert, so dass der Spielzustand wechselt. Befindet man sich danach innerhalb des Spiels, so müssen nur die Höhen der Berge neu generiert werden. Dazu benutzt man einen einfachen Zufallsgenerator.\*\*

Befindet man sich jedoch anschließend außerhalb des Spiels, so sind mehr Schritte durchzuführen. Die Liste der Berge sowie die Liste der Bluetooth-Messwerte müssen zurückgesetzt werden. Davor jedoch sollten entsprechende Gamification-Bewertungsfunktionen aufgerufen werden, die feststellen, wie viele XP beziehungsweise welche Badges der Nutzer für diese Leistung erhält. Die dazu nötigen Funktionen werden im nächsten Abschnitt genauer beschrieben. Nach den jeweiligen Änderungen wird die View natürlich neu gezeichnet.

Das Android-Grafik-Framework unterscheidet beim Zeichnen im Übrigen dazwischen, was gezeichnet wird (bestimmt von der Klasse `Canvas`) und wie es gezeichnet wird (geregelt durch die Klasse `Paint`).<sup>11</sup> Ein solches `Paint`-Objekt bestimmt dabei unter anderem Farbe, Stil und Schrift, die auf ein Objekt angewendet werden. Durch diese Teilung ist es möglich, `Paint`-Objekte schon vor der Benutzung zu erstellen und anschließend wiederzuverwenden. Da Views oft neu gezeichnet werden, kann dadurch die Performance verbessert und die Benutzeroberfläche flüssiger werden.<sup>11</sup>

Das fertige Spiel lässt sich nun folgendermaßen steuern: Zu Beginn erscheint der Text „Um das Spiel zu beginnen, berühre den Bildschirm.“ Tippt man auf diesen, erscheint die Oberfläche des Spiels. Das Flugzeug kann auf dieser nun ausschließlich durch Muskelaktivität und nicht durch Touch-Gesten oder ähnliches gesteuert werden. Stößt man nun gegen einen Berg oder tippt man erneut auf den Bildschirm, wird das Spiel beendet, wobei im ersten Fall ein zusätzlicher Hinweis („Du bist gegen einen Berg gestoßen. Du verlierst!“) erscheint. Anschließend wird die für das

---

\*\*Zunächst befinden sich jedoch über den Bereich einer Bildschirmbreite keine Berge, um den Spielstart zu vereinfachen.

Spiel zu vergebende XP-Punktzahl berechnet und angezeigt sowie auf eventuell abgeschlossene Quests überprüft.

## 5.4 Umsetzung der Gamification in der App

Im Bereich der Gamification fiel die Entscheidung, ein (Erfahrungs-)Punktesystem und Quests bzw. Badges einzuführen. Für jede durchgeführte Übung bzw. für jedes Spielen des Minispiels werden hier abhängig von den aufgenommenen Messwerten Punkte verteilt. Dabei gelten für die Verteilung folgende Kriterien:

- Bei einer **Übung** wird eine Liste  $L$  der prozentualen Messwerte erstellt. Aus dieser kann dann die Punktzahl  $P$  berechnet werden:

$$P = \frac{\min(L) + \text{avg}(L) + \max(L)}{3}$$

Dabei berechnet  $\min(L)$  den Minimalwert der Liste,  $\text{avg}(L)$  den Durchschnittswert der Liste und  $\max(L)$  den Maximalwert der Liste.

- Beim **Minispiel** gelten die Regeln für eine Übung grundsätzlich weiter. Jedoch werden als Bonus zusätzlich  $2 \cdot d$  XP vergeben, wobei  $d$  die Anzahl der aufgenommenen Messwerte bis zum Absturz des Flugzeugs darstellt.

Natürlich wird der Nutzer auch jeweils über die vergebene Punktzahl mithilfe einer im unteren Bildschirmteil eingeblendeten Toast-Benachrichtigung informiert.

Bei den bereits erwähnten Quests erschien es sinnvoll, verschiedene Kriterien bzw. Anforderungen für die Fertigstellung der Quest vorauszusetzen. Um diese bei den einzelnen Quests flexibel regulieren zu können, kann man solche Daten in einer lokalen Datenbank speichern, auf welche die App dann zugreifen kann. Android bietet dafür die sehr kompakte Datenbankbibliothek *SQLite* an, welche ein relationales Datenbanksystem implementiert und über die standardisierte Datenbanksprache *SQL* angesprochen werden kann.<sup>6</sup> Besonders wichtig ist hierbei, dass keine Server-Software benötigt wird, sondern die Datenbank einfach in einer einzelnen Textdatei gespeichert werden kann, was im Kontext einer Android-App vieles vereinfacht.<sup>6</sup>

Die Datenbanktabelle besteht nun aus den folgenden Spalten, welche eine einzelne Quest kennzeichnen:

Spaltenname	Spaltentyp	Beschreibung
<code>-id</code>	INT	Die eindeutige Identifikationsnummer der Quest, über die sie aufgerufen und verändert werden kann.
<code>title</code>	VARCHAR(50)	Der Titel der Quest. Dieser kann bis zu 50 Zeichen lang sein.
<code>description</code>	VARCHAR(150)	Eine Beschreibung der Quest, die bis zu 150 Zeichen lang sein kann. Sie sollte möglichst die in den restlichen Spalten definierten Ziele zum Erfüllen der Quest zusammenfassen.
<code>icon</code>	INT	Ein Zahl zwischen 1 und 5, die für die Anzeige der Quest eines von 5 vordefinierten Symbolen auswählt.
<code>requiredXP</code>	INT	Gibt den XP-Stand an, der zur Anzeige der Quest nötig ist.

Spaltenname	Spaltentyp	Beschreibung
finishedXP	INT	Gibt den XP-Stand an, der zum Beenden der Quest notwendig ist.
earnedXP	INT	Gibt die Anzahl an XP an, die der Nutzer für die Beendigung der Quest erhält.
overPercentage	INT	Gibt einen Messwert an, über dem man für eine bestimmte Zeit sein muss, um die Quest zu beenden.
timeOverPercentage	INT	Gibt die Zeitspanne an, während der man über dem genannten Messwert sein muss.
minimumPercentage	INT	Gibt eine Prozentzahl bzw. einen Messwert an, der mindestens einmal während einer Übung erreicht werden muss, um die Quest zu beenden.
isCompleted	BOOL	Gibt an, ob die Quest bereits beendet und damit in ein Badge umgewandelt wurde.

In allen INT-Spalten außer `_id` und `icon` kann auch eine 0 stehen. Dies bedeutet dann, dass die entsprechende Anforderung bei dieser Quest nicht benötigt wird.

Nun ist es relativ einfach, die Quests und Badges in der App anzuwenden. Nach jeder Übung und jedem Spiel kann nun anhand der in der Tabelle festgelegten Kriterien überprüft werden, ob eine der Quests beendet wurde. Dazu müssen drei Bedingungen erfüllt sein: die `minimumPercentage`-Bedingung (der Wert wurde mindestens einmal erreicht), die `overPercentage`-Bedingung (der Wert wurde für die angegebene Zeitspanne überschritten) und ein entsprechender XP-Stand.

Ist dies der Fall, kann die Quest durch eine Datenbankabfrage auf „erledigt“ gesetzt werden. Die durch die Quest verdienten XP-Punkte können dem Nutzer nun gutgeschrieben werden. Gleichzeitig wird mithilfe eines `DialogFragments` dem Benutzer ein Dialog angezeigt, der ihn über die abgeschlossenen Quests und die dafür erhaltenen Erfahrungspunkte informiert.

Natürlich ist es dem Nutzer auch möglich, sich die erhaltenen Badges bzw. die zur Verfügung stehenden Quests anzuzeigen. Im ersten Fall wird diese Funktion durch einen Button auf der Startseite aufgerufen, im letzteren Fall durch einen Button auf der Übungsseite. In beiden Fällen wird ein `Dialog` mit einer `ListView` angezeigt, die Elemente auflisten kann. Hier sind dies die Quests bzw. Badges, welche über einen `CursorAdapter` einfach aus der Datenbank abgerufen werden können.

In Hinsicht auf die Umsetzung der Gamification ist jedoch auch zu betrachten, inwieweit die App der Gamification-Theorie folgt, um daraus möglicherweise Rückschlüsse auf die Wirksamkeit zu ziehen. Bezüglich der Spielertypen ist die App beispielsweise eher für Forscher und Erfolgstypen ausgelegt, da sich die Gamification-Elemente aktuell darauf fokussieren, den Nutzer Erfolge erreichen zu lassen. Nutzer dagegen, die im Bartle-Test eher als Gesellige oder Killer eingeordnet werden, haben wenig Möglichkeiten, mit anderen zu interagieren und mit ihnen in Wettbewerb zu treten, da eine Mehrspielerfunktion fehlt, welche den zeitlichen Rahmen der Arbeit gesprengt hätte. Die Einbindung einer solchen Funktion wäre jedoch eine Erweiterungsmöglichkeit.

Jedoch ist die für Gamification wichtige Handlungstransparenz gegeben, da der Nutzer sich seinen XP-Stand und die verfügbaren Quests einfach ansehen kann. Neue Herausforderungen sind trotzdem nur begrenzt gegeben, da das Hinzufügen von Quests in der aktuellen Version noch schwierig ist. Dies ließe sich durch eine Updatefunktion über das Internet beheben. Das

Minispiel ist weder zu einfach noch zu schwer, da es sich am bekannten Prinzip des Spiels *Flappy Bird* orientiert. Das ermöglicht es dem Nutzer, in einen Flow zu kommen.

Auch das Prinzip der operanten Konditionierung wurde umgesetzt, denn die Vergabe von Erfahrungspunkten und Badges erfolgt abhängig von den Messwerten immer unterschiedlich. Die Belohnung in einem variablen Intervall ist dabei jedoch nicht möglich, da die Vergabe nur jeweils nach einer Übung oder einem Spiel erfolgen kann.

Im Bereich der spieltypischen Mechanismen wurden einige umgesetzt. So enthält die App, wie bereits beschrieben, ein Erfahrungspunkte-System. Ein Highscore jedoch ist aufgrund der fehlenden Mehrspieler-Möglichkeiten nicht vorhanden. Hingegen ist ein System aus Quests und Badges vorhanden, die so verknüpft sind, dass aus einer abgeschlossenen Quest ein Badge erzeugt wird. Damit existiert ein sichtbarer Status, der der App ein Ziel und eine Struktur gibt und den Fortschritt des Nutzers symbolisiert. Ein sinnvolles Ziel bzw. Epic Meaning ist hier in jedem Fall vorhanden, denn der Benutzer möchte seine Armlähmung bekämpfen.

## 5.5 Funktionsweise des Benachrichtigungssystems

Das Benachrichtigungssystem besteht aus einer Reihe von Funktionen, die vorrangig durch die Einstellungsseite gesteuert werden. Die erste dieser Funktionen erledigt eine seit Android 8 für das Senden von Benachrichtigungen nötige Maßnahme, indem sie einen Benachrichtigungskanal einrichtet. Mithilfe dessen ist es dem Benutzer möglich, die Benachrichtigungen über die Systemeinstellungen zu unterdrücken.<sup>10</sup> Tut er dies nicht, können die Benachrichtigungen über die Einstellungsseite der App reguliert werden.

Eine weitere Funktion ist für das Setzen einer Erinnerung zuständig. Sie liest aus den Einstellungen (sogenannte SharedPreferences) aus, ob und wann erinnert werden soll. Nachdem alle bisherigen Erinnerungen gelöscht wurden, kann eine neue gesetzt werden. Dazu verwendet die App den System-Service AlarmManager.<sup>22</sup> Sobald die gewünschte Zeit erreicht wurde, ruft dieser einen BroadcastReceiver auf. Solche BroadcastReceiver sind ein fester Bestandteil des Android-SDK und können auf vielfältige Meldungen durch das Android-System, welche auf Systemereignisse hinweisen, reagieren.<sup>6</sup> In diesem Fall wird hier eine Methode zum Senden der Benachrichtigung aufgerufen.

Wird eine solche Benachrichtigung gesendet, ist sie systemweit für den Nutzer sichtbar und erinnert ihn daran, seine Übungen durchzuführen. Bei Klick auf die Benachrichtigung öffnet sich die Startseite der App.

Schließlich lassen sich, wie bereits erwähnt, sämtliche im AlarmManager gespeicherten Erinnerungen auch wieder löschen, wovon entsprechend der Einstellungen Gebrauch gemacht wird.<sup>‡</sup>

---

<sup>‡</sup>Die Implementierung des Benachrichtigungssystems in NotificationScheduler.kt basiert teilweise auf einer in<sup>22</sup> vorgestellten Lösung.

## 6 Zusammenfassung

Das im Rahmen dieser Arbeit entstandene Gerät mitsamt der App soll Patienten bei der Therapie von Armlähmungen unterstützen und motivieren. Deshalb waren die Möglichkeit zur Durchführung von Bewegungsübungen und ein durch solche steuerbares Minispiel Bestandteile der Zielstellung.

Um diese in eine App zu integrieren und diese möglichst motivierend zu gestalten, kam das Prinzip der Gamification zum Einsatz. Dabei werden spieltypische Elemente wie Ranglisten, Erfahrungspunkte und Aufgaben außerhalb spielerischer Zusammenhänge angewendet. Mithilfe von Gamification lässt sich das Verhalten von Menschen beeinflussen, in diesem Fall etwa, um die Motivation der Patienten zu erhöhen. Damit Gamification erfolgreich ist, sollte man verschiedene Kriterien beachten, die im Idealfall ein erfolgreiches Spiel hervorbringen. Dazu zählt beispielsweise das Spielertypen-Koordinatensystem nach Bartle.

Gamification kann, wie das Experiment „The Fun Theory“ zeigte, motivationssteigernde Wirkung haben, auch wenn dies wissenschaftlich umstritten ist. Weiterhin wird in der App das Konzept des Biofeedbacks umgesetzt, das Körperfunktionen mithilfe von technischen Mitteln für das Bewusstsein beobachtbar macht.

Zur Therapie von Schlaganfällen existieren verschiedene Methoden. Nach ersten Maßnahmen wie der Stabilisierung der Vitalfunktionen und einer Thrombolyse-Therapie müssen die aufgetretenen Folgen behandelt werden. Dazu existieren verschiedene Therapiekonzepte wie das Bobath-Konzept, die versuchen, verlorene Fähigkeiten zu kompensieren. Dabei werden Methoden wie das Arm-Basis- und das Arm-Fähigkeits-Training angewendet, die als Grundlage für das in dieser Arbeit entwickelte Gerät dienen.

Das Gerät selbst besteht aus einer Schaltung. Diese enthält einen Mikrocontroller, einen Muskelsensor (EMG) und einen Bluetooth-Chip. Dabei nimmt der Mikrocontroller die durch den Sensor gelieferten Messdaten auf und sendet sie an ein in der Nähe befindliches Smartphone. Dieser Ablauf wurde mithilfe eines Programms in C realisiert.

Auf diesem Smartphone befindet sich eine App für das Betriebssystem Android, die mit dem Gerät über Bluetooth kommuniziert und die Messwerte auswertet. Dabei erlaubt sie es, Bewegungsübungen durchzuführen, während diese grafisch dargestellt werden können. Ein Minispiel, welches als Android-View umgesetzt wurde und ähnlich wie das bekannte Spiel *Flappy Bird* funktioniert, ist ebenfalls vorhanden. Beide Bereiche werden über ein Gamification-System verbunden, das aus je nach Leistung verteilten Erfahrungspunkten und Abzeichen bzw. Aufgaben besteht. Diese Aufgaben werden in einer SQL-Tabelle gespeichert. Weiterhin kann die App mithilfe von Benachrichtigungen an Übungen erinnern.

Es bestehen einige Weiterführungsmöglichkeiten. So wäre es zum Beispiel möglich, Mehrspielerfunktionen hinzuzufügen, die dem Spiel eine soziale Komponente geben würden. Sollte das Gerät tatsächlich eingesetzt werden, so wäre es sinnvoll, zusammen mit Ärzten ein tragfähiges Therapiekonzept zu entwickeln. Auch könnte man mithilfe professioneller Spieldesigner die Gamification ausbauen. Ein genauerer Sensor sowie eine Übersicht über den Therapieerfolg für den Arzt wären in dieser Hinsicht ebenfalls mögliche Erweiterungen.

Abschließend lässt sich sagen, dass das hier vorliegende Gerät samt der App bei entsprechender Entwicklung einiges Potential birgt. So meinte sogar der Robotiker Sami Haddadin in einem Interview zu Robotern: „Die Rehabilitation [ist] ein großes Thema, also das Bewegungstraining bei Schlaganfallpatienten [...]“<sup>3</sup> Ein solches Gerät könnte dabei vielleicht schon in naher Zukunft eine wichtige Rolle spielen.

## 7 Literatur- und Quellenverzeichnis

### Literaturquellen

- [1] Cunningham, Christopher; Zichermann, Gabe: *Gamification by Design - Implementing Game Mechanics in Web and Mobile Apps*, 1. Auflage, Sebastopol, O'Reilly Verlag, 2011, online verfügbar unter [https://doc.lagout.org/programmation/GameDesign/GamificationbyDesign-Zichermann,Cunningham-O'Reilly\(2011\)/GamificationbyDesign-Zichermann,Cunningham-O'Reilly\(2011\).pdf](https://doc.lagout.org/programmation/GameDesign/GamificationbyDesign-Zichermann,Cunningham-O'Reilly(2011)/GamificationbyDesign-Zichermann,Cunningham-O'Reilly(2011).pdf)  
[Zugriff am 30.1.2018, 18:20 Uhr]
- [2] Gargenta, Marko: *Einführung in die Android-Entwicklung*, 1. Auflage, Köln, O'Reilly Verlag, 2011
- [3] Grävemeyer, Arne: *Ära starker Bots mit zarten Fingern*, Roboterforscher Haddadin erwartet Wandel in Industrie und Haushalt, in: c't 11/2018, S. 68 - 69
- [4] Isakova, Svetlana; Jemerov, Dmitry: *Kotlin in Action*, 1. Auflage, Shelter Island, Manning Publications, 2017
- [5] Jakicic, John M. et al.: *Effect of Wearable Technology Combined With a Lifestyle Intervention on Long-term Weight Loss*, The IDEA Randomized Clinical Trial, In: Journal of the American Medical Association, 316(11)/2016, S. 1161 - 1171, online verfügbar unter <https://jamanetwork.com/journals/jama/articlepdf/2553448/joi160104.pdf>  
[Zugriff am 27.12.2017, 15:37 Uhr]
- [6] Künneth, Thomas: *Android 8 - Das Praxisbuch für Java-Entwickler*, 5. aktualisierte Auflage, Bonn, Rheinwerk Verlag, 2018
- [7] Platz, Thomas; Roschka, Sybille: *Rehabilitative Therapie bei Armlähmungen nach einem Schlaganfall*, Patientenversion der Leitlinie der Deutschen Gesellschaft für Neurorehabilitation, Bad Honnef, Hippocampus Verlag, 2011, online verfügbar unter [http://www.kompetenznetz-schlaganfall.de/fileadmin/download/Arm-Reha/Leitlinie\\_Therapie\\_Armlaehmung\\_220911-verlinkt.pdf](http://www.kompetenznetz-schlaganfall.de/fileadmin/download/Arm-Reha/Leitlinie_Therapie_Armlaehmung_220911-verlinkt.pdf)  
[Zugriff am 27.12.2017, 16:04 Uhr]
- [8] Schmitt, Günter: *Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie*, Programmierung in Assembler und C - Schaltungen und Anwendungen, 4. Auflage, München, Oldenbourg Verlag, 2008

### Internetquellen

- [9] Android Developer Team: *Bluetooth overview*, <https://developer.android.com/guide/topics/connectivity/bluetooth>  
[Zugriff am 11.11.2018, 15:36 Uhr]
- [10] Android Developer Team: *Create a Notification*, <https://developer.android.com/training/notify-user/build-notification>  
[Zugriff am 11.11.2018, 15:38 Uhr]
- [11] Android Developer Team: *Custom Drawing*, <https://developer.android.com/training/custom-views/custom-drawing>  
[Zugriff am 11.11.2018, 15:40 Uhr]

- [12] Android Developer Team: *Making the View Interactive* , <https://developer.android.com/training/custom-views/making-interactive>  
 [Zugriff am 11.11.2018, 15:43 Uhr]
- [13] Antwerpes, Frank et al.: *Schlaganfall*, <http://flexikon.doccheck.com/de/Schlaganfall>  
 [Zugriff am 31.12.2017, 12:35 Uhr]
- [14] Atmel Corporation: *ATmega48A/PA/88A/PA/168A/PA/328/P*, Atmel 8-Bit Microcontroller with 4/8/16/32 KBytes In-System Programmable Flash, Datasheet, [http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P\\_datasheet\\_Complete.pdf](http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf)  
 [Zugriff am 27.12.2017, 11:46 Uhr]
- [15] Atmel Corporation: *ATmega48PA/88PA/168PA*, 8-bit AVR Microcontrollers, Datasheet Complete, [http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42734-8-bit-AVR-Microcontroller-ATmega48PA-88PA-168PA\\_Datasheet.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42734-8-bit-AVR-Microcontroller-ATmega48PA-88PA-168PA_Datasheet.pdf)  
 [Zugriff am 27.12.2017, 11:48 Uhr]
- [16] Autorengemeinschaft: *AVR-GCC-Tutorial*, <https://www.mikrocontroller.net/articles/AVR-GCC-Tutorial>  
 [Zugriff am 27.12.2017, 12:10 Uhr]
- [17] Autorengemeinschaft: *Bartle-Test*, <https://de.wikipedia.org/wiki/Bartle-Test>  
 [Zugriff am 4.4.2018, 14:11 Uhr]
- [18] Autorengemeinschaft: *Biofeedback*, <https://de.wikipedia.org/wiki/Biofeedback>  
 [Zugriff am 5.4.2018, 16:25 Uhr]
- [19] Autorengemeinschaft: *Schlaganfall*, <https://de.wikipedia.org/wiki/Schlaganfall>  
 [Zugriff am 31.12.2017, 13:01 Uhr]
- [20] Drescher, Frank: *Spiele und Spielzeug - Gamification*, [https://www.planet-wissen.de/gesellschaft/spiele\\_und\\_spielzeug/gamification/index.html](https://www.planet-wissen.de/gesellschaft/spiele_und_spielzeug/gamification/index.html)  
 [Zugriff am 30.3.2018, 10:12 Uhr]
- [21] Feichter, Martina: *Schlaganfall*, <https://www.netdoktor.de/krankheiten/schlaganfall/>  
 [Zugriff am 31.12.2017, 13:25 Uhr]
- [22] Fernando, Jaison: *How to schedule notifications using AlarmManager?*, <https://droidmentor.com/schedule-notifications-using-alarmmanager/>  
 [Zugriff am 11.11.2018, 15:49 Uhr]
- [23] Freyer, Timo; Mörkl, Sabrina; Ostendorf, Norbert: *Bobath-Konzept*, <http://flexikon.doccheck.com/de/Bobath-Konzept>  
 [Zugriff am 3.1.2018, 17:42 Uhr]

- [24] ITead Studio: *HC-05, Bluetooth to Serial Port Module*, <http://www.electronicaestudio.com/docs/istd016A.pdf>  
 [Zugriff am 27.12.2017, 11:13 Uhr]
- [25] Koch, Michael; Ott, Florian: *Gamification – Steigerung der Nutzungsmotivation durch Spielkonzepte*, Projekt mit der Forschungsgruppe Kooperationssysteme an der Universität der Bundeswehr München, <http://www.sozio.tech.org/gamification-steigerung-der-nutzungsmotivation-durch-spielkonzepte/>  
 [Zugriff am 30.1.2018, 18:03 Uhr]
- [26] Parrish, Kevin: Professor Uses RPG-like Exp Rather Than Grades, <https://www.tomsguide.com/us/Experience-Points-XP-Indiana-University-news-6183.html>  
 [Zugriff am 5.4.2018, 15:32 Uhr]
- [27] Nelles, Gereon et al.: *Motorische Rehabilitation nach Schlaganfall*, <http://www.friedehorst.de/nrz/rehabilitation.pdf?m=1140520827>  
 [Zugriff am 27.12.2017, 16:16 Uhr]
- [28] Reinhardt, Anja: *Motivation und Manipulation im Alltag*, Spieltheorie „Gamification“, <http://www.deutschlandfunk.de/spieltheorie-gamification-motivation-und-manipulation-im.724.de.html>  
 [Zugriff am 5.4.2018, 15:59]
- [29] Seeed Technology Co. Ltd.: *Grove - EMG Detector*, [http://wiki.seeed.cc/Grove-EMG\\_Detector/](http://wiki.seeed.cc/Grove-EMG_Detector/)  
 [Zugriff am 30.12.2017, 14:03 Uhr]
- [30] Statistisches Bundesamt: *Ergebnisse der Todesursachenstatistik für Deutschland 2015*, ausführliche 4-stellige ICD10-Klassifikation, [https://www.destatis.de/DE/Publikationen/Thematisch/Gesundheit/Todesursachen/Todesursachenstatistik5232101157015.xlsx?\\_\\_blob=publicationFile](https://www.destatis.de/DE/Publikationen/Thematisch/Gesundheit/Todesursachen/Todesursachenstatistik5232101157015.xlsx?__blob=publicationFile)  
 [Zugriff am 2.1.2018, 11:16 Uhr]
- [31] W wie Wissen: *Gamification - Wie Spielen den Alltag interessanter macht*, Ausschnitt aus der gleichnamigen Sendung in Das Erste am 19.12.2015 (16.00 Uhr), <http://www.ardmediathek.de/tv/W-wie-Wissen/Gamification-Wie-Spielen-den-Alltag-in/Das-Erste/Video?bcastId=427262&documentId=32368232>  
 [Zugriff am 4.4.2018, 14:32 Uhr]

## Bildquellen

- [Abb. 2] [http://www.volkskrankheiten.at/images/1237/widgets/Schlaganfall-\(2\).svg](http://www.volkskrankheiten.at/images/1237/widgets/Schlaganfall-(2).svg)  
 [Zugriff am 31.12.2017, 13:37 Uhr]
- [Abb. 3] [https://scontent-frx5-1.xx.fbcdn.net/v/t1.0-9/225807\\_10150179024982659\\_311531\\_n.jpg?\\_nc\\_cat=0&oh=986b13c0619d22e40dc9de883a6ed930&oe=5B5CEE4C](https://scontent-frx5-1.xx.fbcdn.net/v/t1.0-9/225807_10150179024982659_311531_n.jpg?_nc_cat=0&oh=986b13c0619d22e40dc9de883a6ed930&oe=5B5CEE4C)  
 [Zugriff am 17.5.2018, 15:56 Uhr]

## 8 Anhang

### 2.1 Schlaganfall als Krankheitsbild

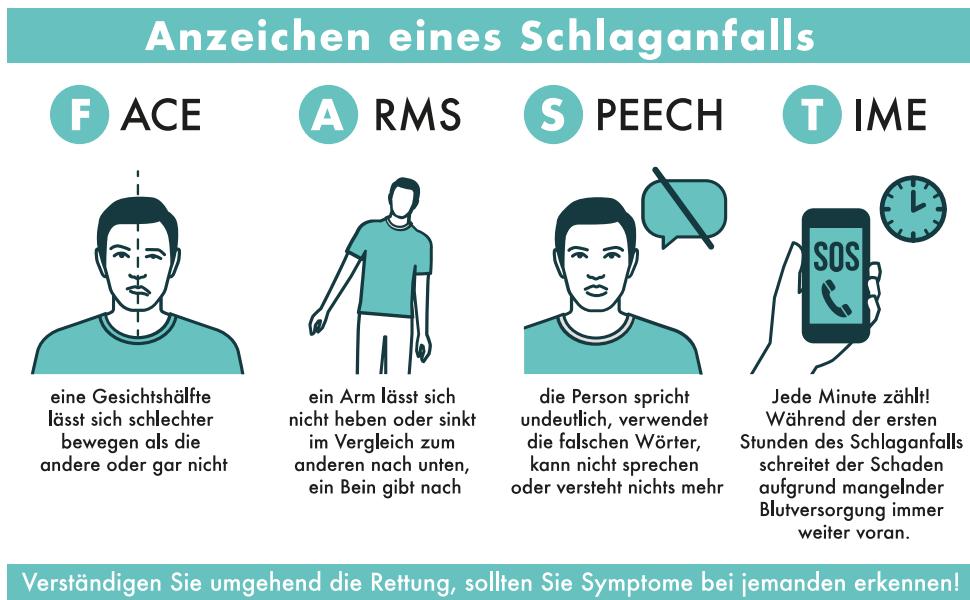


Abbildung 2: Schaubild zum FAST-Test

### 3.1 Gamification



Abbildung 3: Die Klavier-Treppe aus dem Projekt *The Fun Theory*

## 4.1 Aufbau der Schaltung

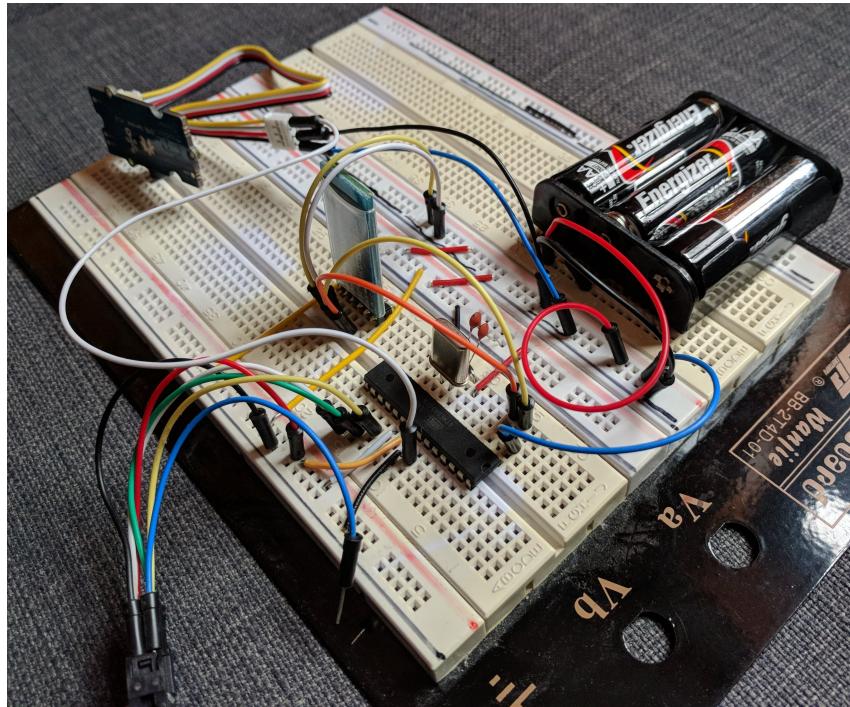


Abbildung 4: Foto der aufgebauten Schaltung auf einem Breadboard

## 4.2 Entwicklung des Programms auf dem Mikrocontroller

```
1 #include <avr/io.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdint.h>
5
6 #define F_CPU 8000000ul
7 #include <util/delay.h>
8 #define BAUD 9600ul
9 #define UBBRVAL 51
10
11 void init(void);
12 void u_putchar(uint8_t x);
13 void u_puts(char *s);
14 void sendCurrentVoltage(void);
15 void delay(uint16_t millisec);
16
17 int main(void) {
18     init();
19     while(1) {
20         sendCurrentVoltage();
21         delay(500);
22     }
23     return 0;
24 }
25
26 void init(void) {
27     ADMUX = 0x00;
28     ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1);
29     ADCSRA |= (1<<ADSC);
30     while (ADCSRA & (1<<ADSC));
```

```

31     (void) ADC;
32
33     UBRR0H = UBBRVAL >> 8;
34     UBRR0L = UBBRVAL & 0xFF;
35     UCSR0B |= (1 << TXEN0) | (1 << RXEN0);
36     UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);
37     (void) UDR0;
38 }
39
40 void u_putchar (uint8_t x) {
41     while (!(UCSR0A & (1 << UDRE0)));
42     UDR0 = x;
43 }
44
45 void u_puts (char *s) {
46     while (*s) {
47         u_putchar (*s++);
48     }
49 }
50
51 void sendCurrentVoltage (void) {
52     ADCSRA |= (1 << ADSC);
53     while (ADCSRA & (1 << ADSC));
54     uint16_t out = ADC;
55     double voltage = (out/1024.0) * 4.5;
56
57     char str[10];
58     sprintf(str, "% .3f \r\n", voltage);
59     u_puts(str);
60 }
61
62 void delay (uint16_t millisec) {
63     while (millisec--) {
64         _delay_ms(1);
65     }
66 }
```

Quellcode 1: Das Programm für den Mikrocontroller

## 5.1 Grundlegender Aufbau und Konzept der App



Abbildung 5: Die Startseite der App

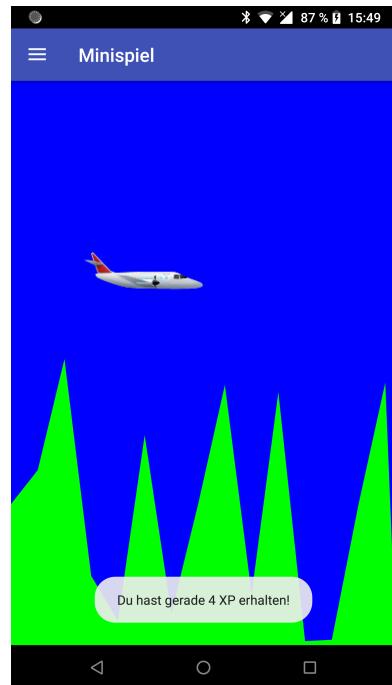


Abbildung 7: Das Minispiel

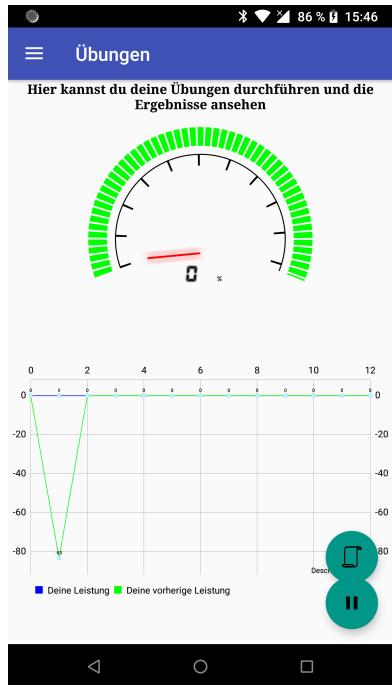


Abbildung 6: Die Übungsseite der App

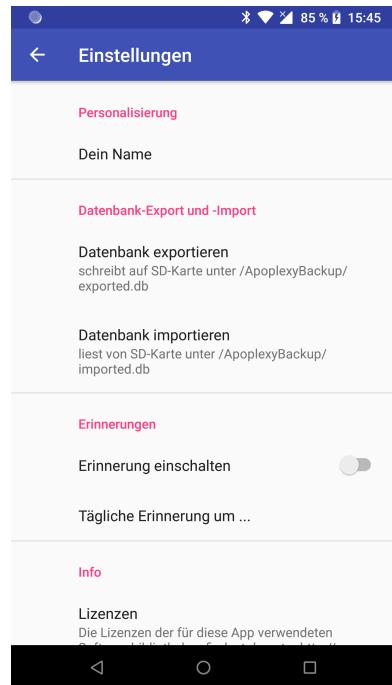


Abbildung 8: Die Einstellungsseite der App

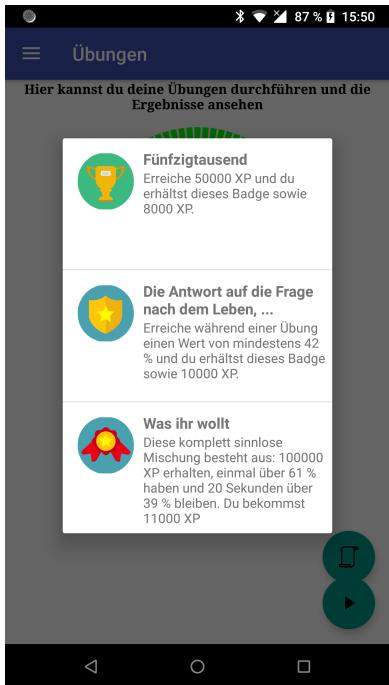


Abbildung 9: Die Liste der verfügbaren Quests

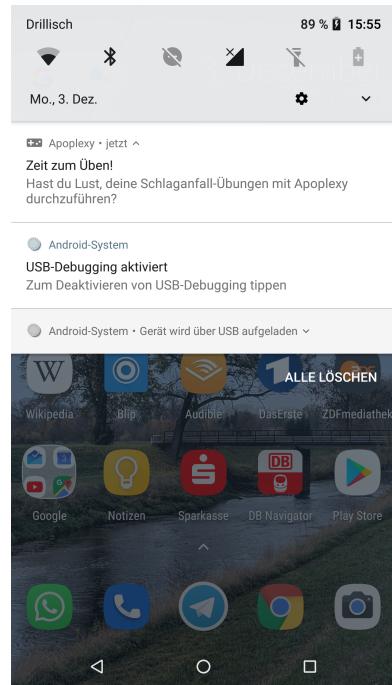


Abbildung 10: Eine Übungsbenachrichtigung

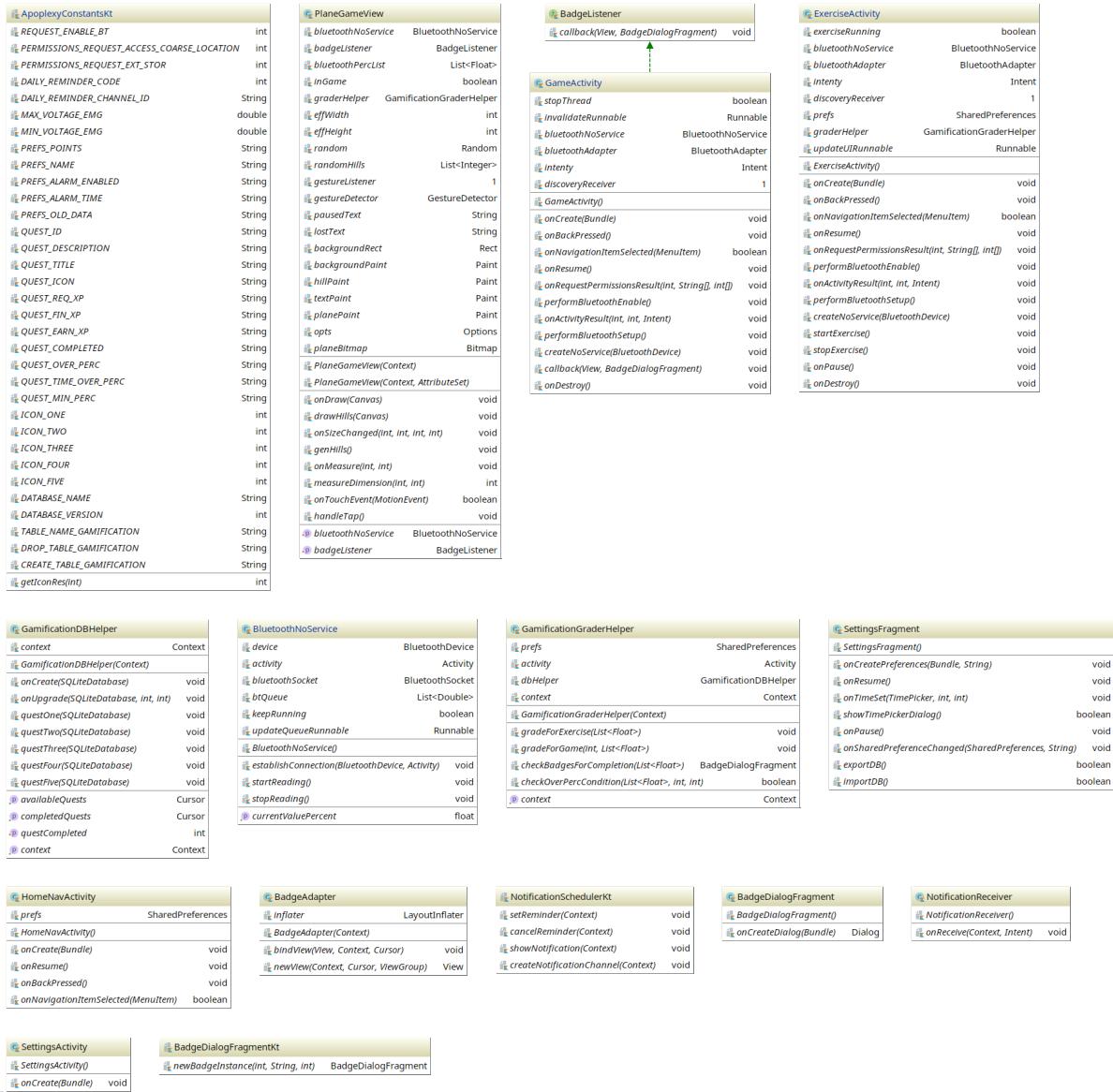


Abbildung 11: UML-Diagramm der App

## 5.2 Kommunikation mit dem Mikrocontroller

```

1 package de.lukasrost.apoplexy
2
3 import android.app.Activity
4 import android.bluetooth.BluetoothDevice
5 import android.bluetooth.BluetoothSocket
6 import android.widget.Toast
7 import java.nio.charset.StandardCharsets
8 import java.util.*
9
10 // Steuerung der Bluetooth-Verbindung
11 class BluetoothNoService {
12     // Bluetooth-Gerät und Socket
13     private lateinit var device: BluetoothDevice
14     private lateinit var activity: Activity
15     private var bluetoothSocket: BluetoothSocket? = null
16
17     // Queue der Messwerte

```

```

18  private val btQueue = mutableListOf<Double>()
19
20  // Thread zum Updaten der Queue
21  private var keepRunning = false
22  private val updateQueueRunnable = Runnable {
23      var read = 0
24      val data = ByteArray(1024)
25
26      // wenn verbunden
27      if (bluetoothSocket != null) {
28
29          // solange nicht gestoppt und Datenempfang vorhanden
30          while (this@BluetoothNoService.keepRunning && bluetoothSocket
31              → != null && ((bluetoothSocket!!.inputStream.read(data).let
32              → { read = it; it != -1 }))) {
33
34              // Einlesen der Bluetooth-Daten
35              val readdata = Arrays.copyOf(data, read)
36              val value = String(readdata, StandardCharsets.UTF_8)
37
38              // Daten der Queue hinzufügen
39              for (number in value.split("\r\n")) {
40                  if (btQueue.size == 4) {
41                      btQueue.removeAt(0)
42                  }
43                  val num = number.toDoubleOrNull()
44                  num?.let { if( num < 1.5) btQueue.add(1.5) else
45                      → btQueue.add(it) } // nur Werte > 1.5
46              }
47          }
48      }
49      activity.runOnUiThread {
50          → Toast.makeText(activity, "Bluetooth-Verbindung beendet!", 
51          → Toast.LENGTH_LONG).show() }
52  }
53
54
55  // Verbindung beginnen
56  fun establishConnection(device: BluetoothDevice, activity: Activity) {
57      this.device = device
58      this.activity = activity
59  }
60
61
62  // mit Bluetooth-Gerät verbinden und Thread starten
63  fun startReading() {
64      try {
65          bluetoothSocket =
66              → device.createRfcommSocketToServiceRecord(UUID.|
67              → fromString("00001101-0000-1000-8000-00805F9B34FB"))
68          bluetoothSocket?.connect()
69          keepRunning = true
70          Thread(updateQueueRunnable).start()
71          activity.runOnUiThread { Toast.makeText(activity, "Erfolgreich
72              → verbunden!", Toast.LENGTH_LONG).show() }
73      } catch (e: Exception) {
74          activity.runOnUiThread { Toast.makeText(activity, "Fehler beim
75              → Verbinden über Bluetooth!", Toast.LENGTH_LONG).show() }
76      }
77  }

```

```

68     // Verbindung beenden, Thread stoppen
69     fun stopReading() {
70         keepRunning = false
71         Thread.sleep(1000)
72         bluetoothSocket?.close()
73     }
74
75     // aktuellen Durchschnittswert der Queue in Prozent des Maximalwerts
76     // berechnen
77     fun getCurrentValuePercent() : Float = 100 * ((btQueue.average() -
78             MIN_VOLTAGE_EMG) / (MAX_VOLTAGE_EMG - MIN_VOLTAGE_EMG)).toFloat()
79 }
```

Quellcode 2: Die BluetoothNoService-Klasse mit der Bluetooth-Funktionalität

### 5.3 Konzept und programmiertechnische Umsetzung des Minispiele

```

1 package de.lukasrost.apoplexy.game
2
3 import android.annotation.SuppressLint
4 import android.content.Context
5 import android.graphics.*
6 import android.util.AttributeSet
7 import android.view.GestureDetector
8 import android.view.MotionEvent
9 import android.view.View
10 import de.lukasrost.apoplexy.BluetoothNoService
11 import de.lukasrost.apoplexy.R
12 import de.lukasrost.apoplexy.helpers.GamificationGraderHelper
13 import java.util.*
14 import de.lukasrost.apoplexy.BadgeListener
15
16
17     // eigene View, beinhaltet das Minispiel
18 class PlaneGameView : View {
19     // Konstruktoren: zeichnet sich nicht selbst
20     constructor(context : Context) : super(context) {
21         setWillNotDraw(false)
22     }
23     constructor(context: Context, attributeSet: AttributeSet) :
24         super(context, attributeSet) {
25         setWillNotDraw(false)
26     }
27
28     // Bluetooth- und Gamification-Variablen
29     private lateinit var bluetoothNoService : BluetoothNoService
30     private lateinit var badgeListener: BadgeListener
31     private var bluetoothPercList = mutableListOf<Float>()
32     private var inGame = false
33     private val graderHelper = GamificationGraderHelper(context)
34
35     // Höhe und Breite der View
36     private var effWidth = width - (paddingLeft + paddingRight)
37     private var effHeight = height - (paddingTop + paddingBottom)
38
39     // Zufällige Berge als Queue
40     private val random = Random()
41     private var randomHills = mutableListOf<Int>()
```

```

41 // Gesten-Detektor für Klicks
42 private val gestureListener = object
43     → : GestureDetector.SimpleOnGestureListener() {
44         override fun onDown(e: MotionEvent?): Boolean {
45             return true
46         }
47     }
48 private val gestureDetector = GestureDetector(context,gestureListener)
49
50 // Texte
51 private val pausedText = "Um das Spiel zu beginnen, berühre den
52     → Bildschirm."
53 private val lostText = "Du bist gegen einen Berg gestoßen. Du
54     → verlierst!"
55
56 // Hintergrund-Rechteck
57 private var backgroundRect =
58     → Rect(paddingLeft,paddingTop,effWidth,effHeight)
59 // Farben, Stile, Textgrößen
60 private val backgroundPaint = Paint().apply {
61     color = Color.BLUE
62     style = Paint.Style.FILL
63 }
64 private val hillPaint = Paint(Paint.ANTI_ALIAS_FLAG).apply {
65     color = Color.GREEN
66     style = Paint.Style.FILL
67 }
68 private val textPaint = Paint(Paint.ANTI_ALIAS_FLAG).apply {
69     style = Paint.Style.FILL
70     textSize = 60f
71     color = Color.BLACK
72 }
73 private val planePaint = Paint(Paint.ANTI_ALIAS_FLAG)
74
75 // Flugzeug-Bild (nur halb so groß wie Originalbild)
76 private var opts = BitmapFactory.Options().apply {
77     inSampleSize = 2
78 }
79 private val planeBitmap =
80     → BitmapFactory.decodeResource(context.resources,
81     → R.drawable.ic_airplane,opts)
82
83 // Zeichnen der View
84 override fun onDraw(canvas: Canvas?) {
85     super.onDraw(canvas)
86     if(inGame) {
87         // im Spiel
88         canvas?.apply {
89             bluetoothPercList.add(bluetoothNoService.]
90             → getCurrentValuePercent())
91             // Verschiebung des Flugzeugs nach oben
92             val verschieb = 70
93             // obere linke Ecke des Flugzeugs durch *Magie* bestimmen
94             val top = paddingTop + effHeight - ( effHeight *
95                 → (bluetoothPercList[bluetoothPercList.size-1] +
96                 → verschieb) / 100 )
97             val left = paddingLeft + effWidth/2 - planeBitmap.width
98         }
99     }
100 }
```

```

91         // Kollision mit Hügel vor Flugzeug -> verloren
92         if(top + planeBitmap.height >= paddingTop + effHeight -
93             ↳ randomHills[randomHills.size/2]){
94             drawText(lostText,50f,(effHeight/2).]
95             ↳ toFloat(),textPaint)
96             Thread.sleep(2500)
97             // Spiel beenden
98             handleTap()
99         } else {
100            // Spiellandschaft (Flugzeug, Himmel, Berge) zeichnen
101            drawRect(backgroundRect, backgroundPaint)
102            drawHills(this)
103            drawBitmap(planeBitmap, left.toFloat(), top,
104             ↳ planePaint)
105        }
106    }
107
108    else {
109        // außerhalb des Spiels -> Text anzeigen
110        canvas?.apply {
111            drawText(pausedText,50f,(effHeight/2).toFloat(),textPaint)
112        }
113    }
114
115    // Berge zeichen
116    private fun drawHills(canvas: Canvas){
117        // Höhe des nächsten Bergs zufällig bestimmen
118        val k = random.nextInt(effHeight) - effHeight / 5
119        randomHills.add(if (k < 0) effHeight / 4 else k)
120        randomHills.removeAt(0)
121
122        // Berge im Abstand von 100 Pixel zeichnen
123        val p = Path()
124        canvas.apply {
125            var offset = 0
126            for (hill in randomHills){
127                p.lineTo((paddingLeft+offset).toFloat(),(paddingTop+
128                  ↳ effHeight - hill).toFloat())
129                offset += 100
130            }
131            p.lineTo(paddingLeft +
132             ↳ effWidth).toFloat(),(paddingTop+effHeight).toFloat())
133            p.lineTo(paddingLeft.toFloat(),(paddingTop+effHeight).]
134            ↳ toFloat())
135            p.close()
136            drawPath(p,hillPaint)
137        }
138    }
139
140    // Bildschirmgröße verändert -> abhängige Werte anpassen
141    override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {
142        effWidth = width - (paddingLeft + paddingRight)
143        effHeight = height - (paddingTop + paddingBottom)
144        backgroundRect = Rect(paddingLeft,paddingTop,effWidth,effHeight)
145        genHills()
146        super.onSizeChanged(w, h, oldw, oldh)
147    }
148
149    // neue initiale Berge-Liste mit Null-Höhe generieren

```

```

144     private fun genHills() {
145         randomHills = mutableListOf<Int>().apply {
146             for (i in 0..effWidth/100) {
147                 add(0)
148             }
149         }
150     }
151
152     // diese und nächste Funktion setzen Höhe und Breite der View bei
→ Anforderung durch System
153     override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int)
154     → {
155
155         val desiredWidth = suggestedMinimumWidth + paddingLeft +
156             → paddingRight
156         val desiredHeight = suggestedMinimumHeight + paddingTop +
157             → paddingBottom
158
158         setMeasuredDimension(measureDimension(desiredWidth,
159             → widthMeasureSpec),
160                     measureDimension(desiredHeight, heightMeasureSpec))
161     }
162
162     private fun measureDimension(desiredSize: Int, measureSpec: Int): Int
163     → {
164         var result: Int
165         val specMode = View.MeasureSpec.getMode(measureSpec)
166         val specSize = View.MeasureSpec.getSize(measureSpec)
167
167         if (specMode == View.MeasureSpec.EXACTLY) {
168             result = specSize
169         } else {
170             result = desiredSize
171             if (specMode == View.MeasureSpec.AT_MOST) {
172                 result = Math.min(result, specSize)
173             }
174         }
175         return result
176     }
177
178     // bei Klick Spiel starten bzw. beenden
179     @SuppressLint("ClickableViewAccessibility")
180     override fun onTouchEvent(event: MotionEvent?): Boolean {
181         return gestureDetector.onTouchEvent(event).let {
182             if(it) {
183                 handleTap()
184                 true
185             } else false
186         }
187     }
188
189     // Spiel starten bzw. beenden
190     private fun handleTap(){
191         inGame = !inGame
192         if(!inGame) {
193             // Beenden
194             // Gamification durchführen
195             graderHelper.gradeForGame(bluetoothPercList.size,
196                 bluetoothPercList)

```

```

196     val fragment =
197         → graderHelper.checkBadgesForCompletion(bluetoothPercList)
198     badgeListener.callback(this,fragment)
199
200     // Bluetooth- und Berge-Listen leeren
201     bluetoothPercList = mutableListOf()
202     randomHills = mutableListOf()
203 } else {
204     // Starten -> Berge generieren
205     genHills()
206 }
207 // View neu zeichnen
208 invalidate()
209
210 // Bluetooth-Service setzen
211 fun setBluetoothNoService(bb : BluetoothNoService){
212     bluetoothNoService = bb
213     bluetoothNoService.startReading()
214 }
215
216 fun setBadgeListener(bl: BadgeListener){
217     badgeListener = bl
218 }
219 }
```

Quellcode 3: Die PlaneGameView-Klasse mit der Implementierung des Minispiele

## 5.4 Umsetzung der Gamification in der App

```

1 package de.lukasrost.apoplexy.helpers
2
3 import android.app.Activity
4 import android.content.Context
5 import android.preference.PreferenceManager
6 import android.widget.Toast
7 import de.lukasrost.apoplexy.*
8 import de.lukasrost.apoplexy.badges.*
9 import kotlin.math.roundToInt
10
11 // Bewertungshelfer für Gamification-Funktionen
12 class GamificationGraderHelper(val context: Context) {
13     private val prefs =
14         → PreferenceManager.getDefaultSharedPreferences(context)
15     private val activity = context as Activity
16     private lateinit var dbHelper : GamificationDBHelper
17
18     // XP für Übungen vergeben
19     fun gradeForExercise(data : MutableList<Float>){
20         // Minimal-, Maximal- und Durchschnittswert der Liste bestimmen
21         val min = data.min()?:0f
22         val max = data.max()?:0f
23         val avg = if (data.average().isNaN()) 0.0 else data.average()
24
25         // Bewertungsfunktion
26         val points = ((avg + min + max) / 3 )
27         val pointsInt = if(!points.isNaN() && points.roundToInt() > 0)
28             → points.roundToInt() else 0
29     }
30 }
```

```

27
28     // neue XP zu bisherigen XP hinzufügen
29     val pointsBefore = prefs.getInt(PREFS_POINTS, 0)
30     prefs.edit().putInt(PREFS_POINTS, pointsBefore + pointsInt).apply()
31     activity.runOnUiThread { Toast.makeText(context, "Du hast gerade
32         → $pointsInt XP erhalten!", Toast.LENGTH_LONG).show() }
33
34     // XP für Minispiel vergeben
35     fun gradeForGame(distanceUntilCrash: Int, data: MutableList<Float>) {
36         // Minimal-, Maximal- und Durchschnittswert der Liste bestimmen
37         val min = data.min():0f
38         val max = data.max():0f
39         val avg = if (data.average().isNaN()) 0.0 else data.average()
40
41         // Bewertungsfunktion
42         val points = ((avg + min + max) / 3) + distanceUntilCrash * 2
43         val pointsInt = if (!points.isNaN() && points.toInt() > 0)
44             → points.toInt() else 0
45
46         // neue XP zu bisherigen XP hinzufügen
47         val pointsBefore = prefs.getInt(PREFS_POINTS, 0)
48         prefs.edit().putInt(PREFS_POINTS, pointsBefore + pointsInt).apply()
49         activity.runOnUiThread { Toast.makeText(context, "Du hast gerade
50             → $pointsInt XP erhalten!", Toast.LENGTH_LONG).show() }
51
52     // Freischaltung von Badges überprüfen
53     fun checkBadgesForCompletion(data: MutableList<Float>) :
54         BadgeDialogFragment?{
55         dbHelper = GamificationDBHelper(context)
56         val cursor = dbHelper.getAvailableQuests()
57         var shouldShowDialog = false
58         var icon = 0
59         var title = ""
60         var earnedXP = 0
61
62         // durch alle verfügbaren Quests iterieren
63         while (cursor.moveToNext()) {
64             val neededXP =
65                 → cursor.getInt(cursor.getColumnIndex(QUEST_FIN_XP))
66             val minPerc =
67                 → cursor.getInt(cursor.getColumnIndex(QUEST_MIN_PERC))
68             val isMinPercCompleted = data.any { it >= minPerc.toFloat() }
69             val isOverPercCompleted = checkOverPercCondition(data, cursor.]
70                 → getInt(cursor.getColumnIndex(QUEST_OVER_PERC)), cursor.]
71                 → getInt(cursor.getColumnIndex(QUEST_TIME_OVER_PERC)))
72
73             // spezifische Bedingungen überprüfen
74             if (neededXP <= prefs.getInt(PREFS_POINTS, 0) &&
75                 isMinPercCompleted && isOverPercCompleted) {
76
77                 // Quest fertig -> Badge freigeschaltet
78                 dbHelper.setQuestCompleted(cursor.getInt(cursor.]
79                     → getColumnIndex(QUEST_ID)))
80
81                 // entsprechend XP vergeben
82                 val pointsBefore = prefs.getInt(PREFS_POINTS, 0)

```

```

75     prefs.edit().putInt(PREFS_POINTS, pointsBefore + cursor.↓
    ↵     getInt(cursor.getColumnIndex(QUEST_EARN_XP))).apply()
76
77     // Dialog soll angezeigt werden
78     icon = cursor.getInt(cursor.getColumnIndex(QUEST_ICON))
79     title += "\\" + ↓
    ↵     cursor.getString(cursor.getColumnIndex(QUEST_TITLE)) + ↓
    ↵     "\\", "↓
80     earnedXP += ↓
    ↵     cursor.getInt(cursor.getColumnIndex(QUEST_EARN_XP))
81     shouldShowDialog = true
82   }
83 }
84 cursor.close()
85 dbHelper.close()
86
87 // Dialog an Aufrufer zurückgeben
88 if (shouldShowDialog){
89   title = title.substring(0,title.length-2)
90   return newBadgeInstance(icon, title, earnedXP)
91 }
92 return null
93 }
94
95 // Überprüfen der Bedingung, dass man bestimmte Zeit über bestimmtem
    ↵ Prozentwert war
96 private fun checkOverPercCondition(data: MutableList<Float>, overPerc
    ↵ :Int, timeOverPerc: Int): Boolean{
97   var count = 0
98   for (el in data){
99     if (el >= overPerc) {
100       count++
101     } else {
102       count = 0
103     }
104     if (count >= timeOverPerc){
105       return true
106     }
107   }
108   return false
109 }
110 }
```

Quellcode 4: Die GamificationGraderHelper-Klasse mit den Gamification-Bewertungsfunktionen

## **Digitale Version**

