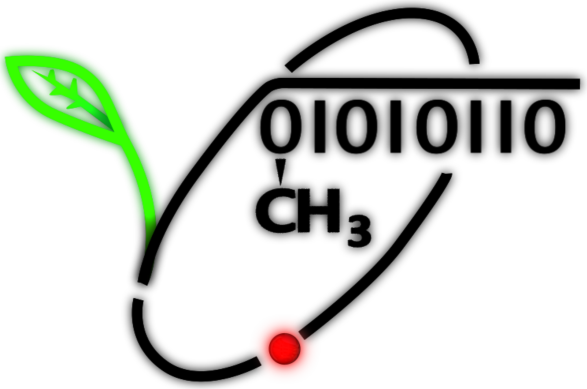

Entwicklung eines Gamification-Unterstützungs- und Motivationsgeräts zur Rehabilitation von Schlaganfall-Patienten

Seminarfacharbeit der Klassenstufe 11/12 (Schuljahre 2017-2019)
am
Albert-Schweitzer-Gymnasium Erfurt, Spezialschulteil

Lukas Rost

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

20. Dezember 2018
Erfurt

Inhaltsverzeichnis

1 Einleitung **2**

2 Die Erkrankung Schlaganfall und geeignete Therapiemethoden **3**
2.1 Schlaganfall als Krankheitsbild 3
2.2 Rehabilitation und Effektivität von Bewegungsübungen 4

3 Umgesetzte Konzepte **5**
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

4 Schaltung und Implementierung des Mikrocontroller-Systems **8**
4.1 Aufbau der Schaltung 8
4.2 Entwicklung des Programms auf dem Mikrocontroller 8

5 Entwicklung der Begleitapp für Android **10**
5.1 Grundlegender Aufbau und Konzept der App 10
5.2 Kommunikation mit dem Mikrocontroller 11
5.3 Konzept und programmiertechnische Umsetzung des Minispiels 11
5.4 Umsetzung der Gamification in der App 12
5.5 Funktionsweise des Benachrichtigungssystems 12

6 Zusammenfassung **14**

7 Literatur- und Quellenverzeichnis **15**

8 Anhang **18**

1 Einleitung

'Die Inzidenz des Schlaganfalls beträgt in Deutschland ca. 180/100.000. Nach Herzerkrankungen und Krebsleiden ist der Schlaganfall die dritthäufigste Todesursache in Deutschland und die häufigste Ursache für Langzeitbehinderung.' (ApoFlex) –> noch ändern

39499 Todesfälle in Deutschland durch Schlaganfall und Folgen 2015 (Destatis)

Große Studien zeigen, dass lediglich 5% der Patienten ihre Arme und Hände wieder uneingeschränkt einsetzen konnten und dass in 20% keinerlei Arm-Handfunktion zurückkehrte. Hingegen werden etwa 75% der hemiparetischen Patienten - selbständig oder mit Hilfe - gehfähig. 25% bleiben auf den Rollstuhl angewiesen oder sind bettlägrig. (RehabNelles)

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.“[21] 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.[13]

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.[13]

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

1. **F**ace/Gesicht: Person kann nur mit einer Gesichtshälfte lächeln
2. **A**rms/Arme: Unfähigkeit, beide Arme mit nach oben geöffneten Handflächen nach vorne zu strecken
3. **S**peech/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 “.[13†] 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.[13] 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.[13]

[†] siehe Abb. 2

2.2 Rehabilitation und Effektivität von Bewegungsübungen

Armlähmungen zählen zu den häufigsten Folgen einer Hirnschädigung, wie sie durch einen Schlaganfall hervorgerufen wird.[7] Aber auch Lähmungen anderer Körperteile oder Aphasie können auftreten.[19] 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 „Taubsche Bewegungsinduktion“ bekannten Methode wird der gesunde Arm täglich über längere Zeit immobilisiert und der Betroffene somit gezwungen, die erkrankte Hand zu benutzen.[19] So kann ein „erlernter Nichtgebrauch“ verhindert werden.[27]

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.[23] 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.[19]

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.[7] 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.[7]

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.[7] Das Arm-Fähigkeits-Training dagegen schult verschiedene Formen von Geschicklichkeit und wird bei leichten Lähmungen angewendet.[7]Beide Formen dieses Trainings zeigen einen positiven Effekt.[27] 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.[7]

Doch auch ein Technikeinsatz kann bei einer solchen Therapie erfolgen. So existiert beispielsweise die Armrobot-Methode, bei der ein Roboter nicht selbständig ausführbare Bewegungen mechanisch unterstützt. In Bezug auf die Effektivität wird diese positiv beurteilt.[7] 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.[7] Die Therapie wurde neutral beurteilt.[27]

Ausgehend von diesen Informationen entschied ich mich dazu, 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“[25] 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.[25]

Indem spieletypische 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.[20] Es kommt zu einer „Actio-et-Reactio“-Erfahrung.[25] Diese Gestaltung und die entsprechende motivationsssteigernde Wirkung lassen sich mithilfe von psychologischen Theorien erklären, auf welche hier jedoch nicht genauer eingegangen werden soll. Gamification an sich 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.[20]

Dieser postulierte auch, dass die Wirkung von Anreizen sich bei verschiedenen Menschen unterscheide. Er entwarf deshalb ein Spielertypen-Koordinatensystem, in dem er Menschen in vier Kategorien verortete, wobei ein Mensch mehreren Kategorien zugleich angehören kann.[17] 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.[17] 10 Prozent aller Menschen werden hauptsächlich diesem Typ zugeordnet.[1]
- die Geselligen/Socializer, denen Kontakte und Interaktionen mit Mitspielern, wie sie auch in klassischen Gesellschaftsspielen vorkommen, besonders wichtig sind.[17] 75 Prozent aller Menschen sind vor allem Socializer.[1]
- die Forscher/Explorer, deren Ziel es ist, möglichst viel zu entdecken und erkunden.[17] Dieser Typ ist bei ca. 10 Prozent aller Menschen am meisten ausgeprägt.[1]
- die Killer, welche Wettbewerb, Wettkampf und Konflikt lieben. Damit sie motiviert sind, müssen andere verlieren und ihnen Respekt erweisen.[17] Dieser Typ macht 5 Prozent aller Menschen aus.[1]

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.[25] Außerdem sollten immer neue Herausforderungen gewährleistet sein.[31]

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.[1] 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.[1]

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).[1] 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-Spielens. Mit diesen können Aktionen des Spielers bewertet werden. Ein Highscore erlaubt einfache Vergleiche durch z.B. eine metrische Punkteskala.[25] Highscores können auch suggestiv eingesetzt werden, indem sie den Spieler möglichst in der Mitte der Liste zeigen und somit gleichzeitig belohnen und motivieren.[1]
- 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.[25] Motivierend sind sie außerdem, da Menschen gerne sammeln. Sie sollten jedoch nicht zu viel eingesetzt werden.[1]
- 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.[1]
- eine Fortschrittsanzeige, die eine dynamische Visualisierung des bisherigen Erfolgs und noch zu erledigender Aufgaben erlaubt.[25]
- gegebenenfalls ein Epic Meaning, also die Arbeit an etwas Erstrebenswerten und an sinnvollen Zielen.[25]

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. [§] Im Ergebnis wurde diese Treppe sogar öfter als die danebenliegende Rolltreppe benutzt.[20]

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.[20] Der Spieledesigner Kevin Richardson schlug für dieses Experiment eine Radarfallenlotterie vor, bei der alle, die die Geschwindigkeitsbegrenzung einhielten, an einem Gewinnspiel um die Strafen der Raser teilnahmen. Als dieser Vorschlag umgesetzt wurde, sank die Geschwindigkeit an der kontrollierten Stelle um 20 %[1]

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.[28] Unternehmen versuchen ebenfalls, durch Gamification geeignete Mitarbeiter zu finden. So entwarf der Bayer-Konzern eine Karriere-App

[§] siehe Abb. 4

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.[26] 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.[1]

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, zu gamifizieren. Im Ergebnis arbeiteten die Kommissionierer schneller und zufriedener als ohne Gamification.[31]

Bei einer amerikanischen Studie, die sich auf den Erfolg von Fitnessmaßnahmen mit und ohne Wearable 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.[5]

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.[18] 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 Rückkoppelung 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.[18]

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.[18] 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 entschied ich mich jedoch für die Messung von Muskelpotentialen per Elektromyografie, 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.[18]

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**[29] , 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**[14] , welcher die vom EMG-Sensor gelieferten Daten an den Bluetooth-Chip weitergibt. Es handelt sich hierbei um einen 8-Bit-Mikrocontroller[15] , 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.[8] 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**[24] , 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. 20) 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 eigenen 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[14] 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[14] 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_putch()`. 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_putch()` 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,[16] weshalb die eigentliche Spannung aus der Gleichung $U = \frac{x}{1024} \cdot U_{ref}$ gewonnen wird.* Die dabei entstandene Gleitkommazahl wird mithilfe der Bibliotheksfunktion `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.

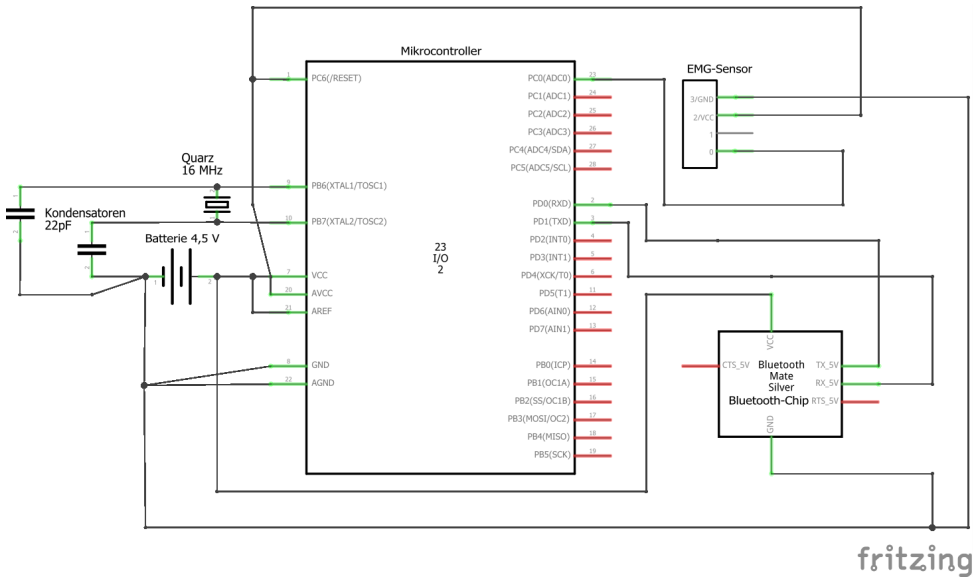

Abbildung 1: Der Schaltplan des Mikrocontrollers

*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 entschied ich mich, das weitverbreitete Betriebssystem Android zu verwenden und die App zu diesem (ab Android 6.0) kompatibel zu machen. 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[6] 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`[6] bereit.

Die einzelnen Funktionen 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.[9]

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.[9]

Ist dies geschehen, kann mit dem Einlesen der per Bluetooth 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 10 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 Median 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 Median 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 Minispiels

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 Minispiels wurde die Möglichkeit des Android-SDKs genutzt, eigene Oberflächenelemente (sogenannte *Views*[11]) 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.[11] 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.[11] 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 prozedural zufällig (bezüglich ihrer Höhe) generierten Berge.
- `onTouchEvent()`: Diese Methode verarbeitet Touch-Eingaben bzw. Klicks.[12] Android bietet dazu eine sogenannten `GestureDetector` an, der einfache und komplizierte Gesten erkennen kann. In diesem Fall muss jedoch nur das Tippen auf den Bildschirm erkannt werden 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`).[11] 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.[11]

5.4 Umsetzung der Gamification in der App

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.[10] Das Erstellen des Kanals erfolgt bei jedem Aufruf der Startseite, aber nach dem ersten Mal bleibt dies wirkungslos, sodass nur genau ein Kanal erstellt wird.

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`.[22] 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 reagieren.[6] 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.‡

‡Die Implementierung des Benachrichtigungssystems basiert teilweise auf einer in[22] vorgestellten Lösung.

6 Zusammenfassung

Ausblick: "Die Rehabilitation [ist] ein großes Thema, also das Bewegungstraining bei Schlaganfallpatienten [...]."(Sami Haddadin in Interview zu Robotern) (Src:CTHaddadin)

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, 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, 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, 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

[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

[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

[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.soziotech.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/

[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

[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

[Zugriff am 31.12.2017, 13:37 Uhr]

[Abb. 4] 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

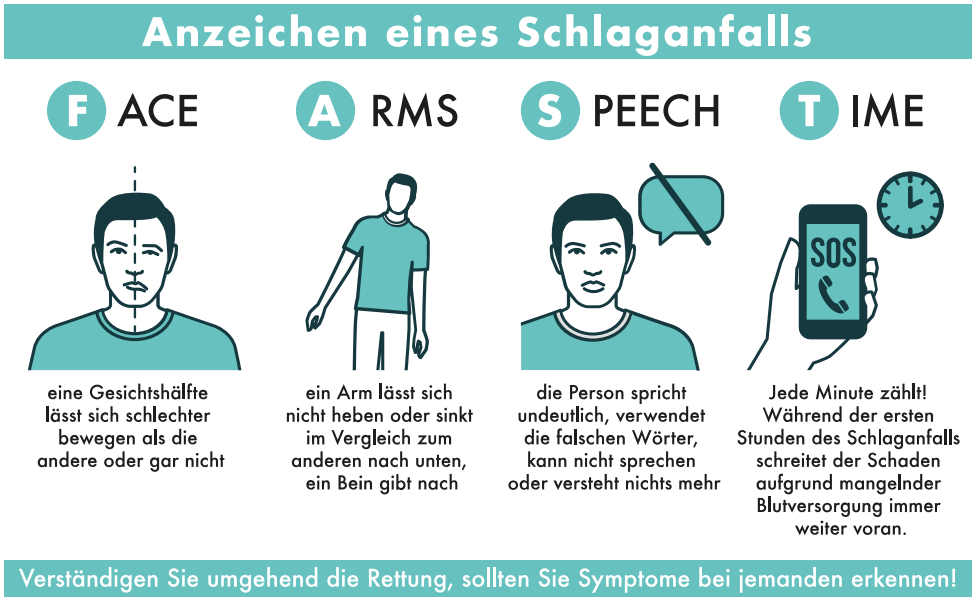

Abbildung 2: Schaubild zum FAST-Test

3.1 Gamification

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

4.1 Aufbau der Schaltung

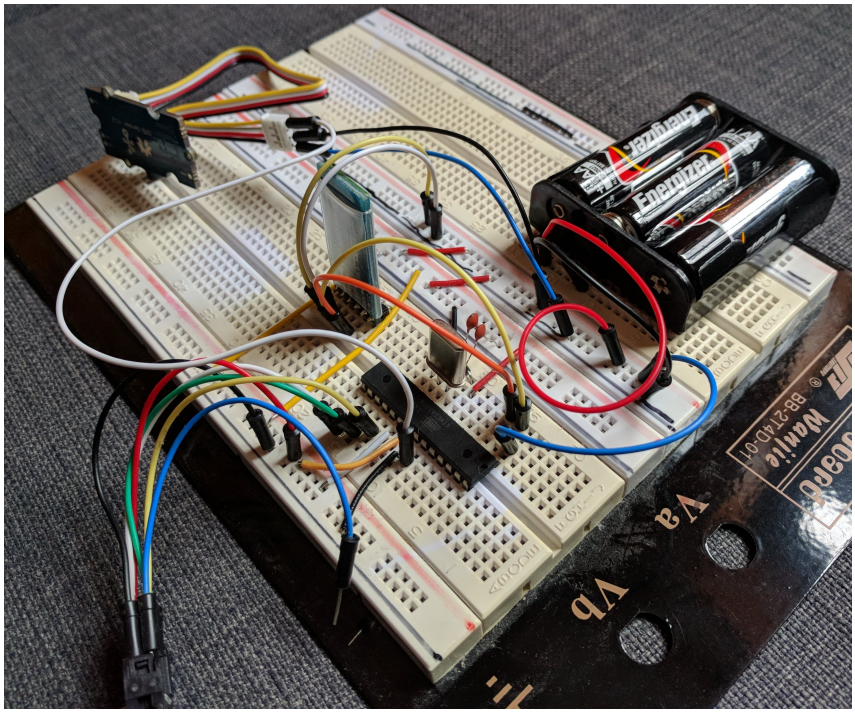

Abbildung 4: Foto der aufgebauten Schaltung auf einem Breadboard

4.2 Entwicklung des Programms auf dem Mikrocontroller

```
#include <avr/io.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>

#define F_CPU 8000000ul
#include <util/delay.h>
#define BAUD 9600ul
#define UBBRVAL 51

void init(void);
void u_putch(uint8_t x);
void u_puts(char *s);
void sendCurrentVoltage(void);
void delay(uint16_t millisec);

int main(void) {
        init();
        while(1) {
                sendCurrentVoltage();
                delay(500);
        }
        return 0;
}

void init(void) {
        ADMUX = 0x00;
        ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1);
        ADCSRA |= (1<<ADSC);
        while (ADCSRA & (1<<ADSC));
```

```
        (void) ADC;

        UBRR0H = UBBRVAL >> 8;
        UBRR0L = UBBRVAL & 0xFF;
        UCSR0B |= (1 << TXEN0) | (1 << RXEN0);
        UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);
        (void) UDR0;
}

void u_putch (uint8_t x) {
        while(!(UCSR0A & (1 << UDRE0)));
        UDR0 = x;
}

void u_puts (char *s) {
        while (*s) {
                u_putch(*s++);
        }
}

void sendCurrentVoltage(void) {
        ADCSRA |= (1 << ADSC);
        while (ADCSRA & (1 << ADSC));
        uint16_t out = ADC;
        double voltage = (out/1024.0) * 4.5;

        char str[10];
        sprintf(str, "%.3f \r\n", voltage);
        u_puts(str);
}

void delay(uint16_t millisec) {
        while(millisec--) {
                _delay_ms(1);
        }
}
```

Quellcode 1: Das Programm für den Mikrocontroller

5.2 Kommunikation mit dem Mikrocontroller

```
package de.lukasrost.apoplexy

import android.bluetooth.BluetoothDevice
import android.bluetooth.BluetoothSocket
import java.nio.charset.StandardCharsets
import java.util.*

// Steuerung der Bluetooth-Verbindung
class BluetoothNoService {
    // Bluetooth-Gerät und Socket
    private lateinit var device: BluetoothDevice
    private var bluetoothSocket: BluetoothSocket? = null

    // Queue der Messwerte
    private val btQueue = mutableListOf<Double>()

    // Thread zum Updaten der Queue
    private var keepRunning = false
```

```
19      private val updateQueueRunnable = Runnable {
20          var read = 0
21          val data = ByteArray(1024)
22
23          // wenn verbunden
24          if (bluetoothSocket != null) {
25
26              // solange nicht gestoppt und Datenempfang vorhanden
27              while (this@BluetoothNoService.keepRunning &&
                ↪  ((bluetoothSocket!!.inputStream.read(data).let { read =
                ↪  it; it != -1 }))) {
28
29                  // Einlesen der Bluetooth-Daten
30                  val readdata = Arrays.copyOf(data, read)
31                  val value = String(readdata, StandardCharsets.UTF_8)
32
33                  // Daten der Queue hinzufügen
34                  for (number in value.split("\r\n")) {
35                      if (btQueue.size == 10) {
36                          btQueue.removeAt(0)
37                      }
38                      val num = number.toDoubleOrNull()
39                      num?.let { if( num < 1.5) btQueue.add(1.5) else
                        ↪  btQueue.add(it) } // nur Werte > 1.5
40                  }
41              }
42          }
43      }
44
45      // Verbindung beginnen
46      fun establishConnection(device: BluetoothDevice){
47          this.device = device
48      }
49
50      // mit Bluetooth-Gerät verbinden und Thread starten
51      fun startReading(){
52          bluetoothSocket = device.createRfcommSocketToServiceRecord(UUID.
            ↪  fromString("00001101-0000-1000-8000-00805F9B34FB"))
53          bluetoothSocket?.connect()
54          keepRunning = true
55          Thread(updateQueueRunnable).start()
56      }
57
58      // Verbindung beenden, Thread stoppen
59      fun stopReading(){
60          keepRunning = false
61          Thread.sleep(1000)
62          bluetoothSocket?.close()
63      }
64
65      // aktuellen Durchschnittswert der Queue in Prozent des Maximalwerts
        ↪  berechnen
66      fun getCurrentValuePercent() : Float = 100 * ((btQueue.median() -
        ↪  MIN_VOLTAGE_EMG) / (MAX_VOLTAGE_EMG - MIN_VOLTAGE_EMG)).toFloat()
67  }
```

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

5.3 Konzept und programmiertechnische Umsetzung des Minispiels

```
1  package de.lukasrost.apoplexy.game
2
3  import android.annotation.SuppressLint
4  import android.content.Context
5  import android.graphics.*
6  import android.support.v4.app.FragmentActivity
7  import android.util.AttributeSet
8  import android.view.GestureDetector
9  import android.view.MotionEvent
10 import android.view.View
11 import de.lukasrost.apoplexy.BluetoothNoService
12 import de.lukasrost.apoplexy.R
13 import de.lukasrost.apoplexy.helpers.GamificationGraderHelper
14 import java.util.*
15
16 // eigene View, beinhaltet das Minispiel
17 class PlaneGameView : View {
18     // Konstruktoren: zeichnet sich nicht selbst
19     constructor(context : Context) : super(context){
20         setWillNotDraw(false)
21     }
22     constructor(context: Context, attributeSet: AttributeSet) :
       ↪ super(context,attributeSet){
23         setWillNotDraw(false)
24     }
25
26     // Bluetooth- und Gamification-Variablen
27     private lateinit var bluetoothNoService : BluetoothNoService
28     private var bluetoothPercList = mutableListOf<Float>()
29     private var inGame = false
30     private val graderHelper = GamificationGraderHelper(context)
31
32     // Höhe und Breite der View
33     private var effWidth = width - (paddingLeft + paddingRight)
34     private var effHeight = height - (paddingTop + paddingBottom)
35
36     // Zufällige Berge als Queue
37     private val random = Random()
38     private var randomHills = mutableListOf<Int>()
39
40     // Gesten-Detektor für Klicks
41     private val gestureListener =  object
       ↪ : GestureDetector.SimpleOnGestureListener(){
42         override fun onDown(e: MotionEvent?): Boolean {
43             return true
44         }
45     }
46     private val gestureDetector = GestureDetector(context,gestureListener)
47
48     // Texte
49     private val pausedText = "Um das Spiel zu beginnen, berühre den
       ↪ Bildschirm."
50     private val lostText = "Du bist gegen einen Berg gestoßen. Du
       ↪ verlierst!"
51
52     // Hintergrund-Rechteck
53     private var backgroundRect =
       ↪ Rect(paddingLeft,paddingTop,effWidth,effHeight)
```

```
54      // Farben, Stile, Textgrößen
55      private val backgroundPaint = Paint(0).apply {
56          color = Color.BLUE
57          style = Paint.Style.FILL
58      }
59      private val hillPaint = Paint(Paint.ANTI_ALIAS_FLAG).apply {
60          color = Color.GREEN
61          style = Paint.Style.FILL
62      }
63      private val textPaint = Paint(Paint.ANTI_ALIAS_FLAG).apply {
64          style = Paint.Style.FILL
65          textSize = 60f
66          color = Color.BLACK
67      }
68      private val planePaint = Paint(Paint.ANTI_ALIAS_FLAG)
69
70      // Flugzeug-Bild (nur halb so groß wie Originalbild)
71      private var opts = BitmapFactory.Options().apply {
72          inSampleSize = 2
73      }
74      private val planeBitmap =
        ↪ BitmapFactory.decodeResource(context.resources,
        ↪ R.drawable.ic_airplane,opts)
75
76      // Zeichnen der View
77      override fun onDraw(canvas: Canvas?) {
78          super.onDraw(canvas)
79          if(inGame) {
80              // im Spiel
81              canvas?.apply {
82                  bluetoothPercList.add(bluetoothNoService.⌟
                    ↪ getCurrentValuePercent())
83                  // Verschiebung des Flugzeugs nach oben
84                  val verschieb = 20
85                  // obere linke Ecke des Flugzeugs durch  *Magie* bestimmen
86                  val top = paddingTop + effHeight - ( effHeight *
                    ↪ (bluetoothPercList[bluetoothPercList.size-1] +
                    ↪ verschieb) / 100 )
87                  val left = paddingLeft + effWidth/2 - planeBitmap.width
88
89                  // Kollision mit Hügel vor Flugzeug -> verloren
90                  if(top + planeBitmap.height >= paddingTop + effHeight -
                    ↪ randomHills[randomHills.size/2]){
91                      drawText(lostText,50f,(effHeight/2).⌟
                        ↪ toFloat(),textPaint)
92                      Thread.sleep(2500)
93                      // Spiel beenden
94                      handleTap()
95                  } else {
96                      // Spiellandschaft (Flugzeug, Himmel, Berge) zeichnen
97                      drawRect(backgroundRect, backgroundPaint)
98                      drawHills(this)
99                      drawBitmap(planeBitmap, left.toFloat(), top,
                        ↪ planePaint)
100                 }
101             }
102         } else {
103             // außerhalb des Spiels -> Text anzeigen
104             canvas?.apply {
```

```
105                 drawText(pausedText,50f,(effHeight/2).toFloat(),textPaint)
106             }
107         }
108     }
109
110     // Berge zeichen
111     private fun drawHills(canvas: Canvas){
112         // Höhe des nächsten Bergs zufällig bestimmen
113         val k = random.nextInt( effHeight) - effHeight / 5
114         randomHills.add(if (k < 0) effHeight / 4 else k)
115         randomHills.removeAt(0)
116
117         // Berge im Abstand von 100 Pixel zeichnen
118         val p = Path()
119         canvas.apply {
120             var offset = 0
121             for (hill in randomHills){
122                 p.lineTo((paddingLeft+offset).toFloat(),(paddingTop+
                    ↪  effHeight - hill).toFloat())
123                 offset += 100
124             }
125             p.lineTo((paddingLeft +
                ↪  effWidth).toFloat(),(paddingTop+effHeight).toFloat())
126             p.lineTo(paddingLeft.toFloat(),(paddingTop+effHeight).
                ↪  toFloat())
127             p.close()
128             drawPath(p,hillPaint)
129         }
130     }
131
132     // Bildschirmgröße verändert -> abhängige Werte anpassen
133     override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {
134         effWidth = width - (paddingLeft + paddingRight)
135         effHeight = height - (paddingTop + paddingBottom)
136         backgroundRect = Rect(paddingLeft,paddingTop,effWidth,effHeight)
137         genRandomHills()
138         super.onSizeChanged(w, h, oldw, oldh)
139     }
140
141     // neue initiale Berge-Liste zufällig generieren
142     private fun genRandomHills(){
143         randomHills = mutableListOf<Int>().apply {
144             for (i in 0..effWidth/100){
145                 val k = random.nextInt( effHeight) - effHeight / 5
146                 add(if (k < 0) effHeight / 4 else k)
147             }
148         }
149     }
150
151     // diese und nächste Funktion setzen Höhe und Breite der View bei
        ↪  Anforderung durch System
152     override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int)
        ↪  {
153
154         val desiredWidth = suggestedMinimumWidth + paddingLeft +
            ↪  paddingRight
155         val desiredHeight = suggestedMinimumHeight + paddingTop +
            ↪  paddingBottom
156
```

```
157         setMeasuredDimension(measureDimension(desiredWidth,
            ↪ widthMeasureSpec),
158                 measureDimension(desiredHeight, heightMeasureSpec))
159     }
160
161     private fun measureDimension(desiredSize: Int, measureSpec: Int): Int
            ↪ {
162         var result: Int
163         val specMode = View.MeasureSpec.getMode(measureSpec)
164         val specSize = View.MeasureSpec.getSize(measureSpec)
165
166         if (specMode == View.MeasureSpec.EXACTLY) {
167             result = specSize
168         } else {
169             result = desiredSize
170             if (specMode == View.MeasureSpec.AT_MOST) {
171                 result = Math.min(result, specSize)
172             }
173         }
174         return result
175     }
176
177     // bei Klick Spiel starten bzw. beenden
178     @SuppressLint("ClickableViewAccessibility")
179     override fun onTouchEvent(event: MotionEvent?): Boolean {
180         return gestureDetector.onTouchEvent(event).let {
181             if(it){
182                 handleTap()
183                 true
184             } else false
185         }
186     }
187
188     // Spiel starten bzw. beenden
189     private fun handleTap(){
190         inGame = !inGame
191         if(!inGame) {
192             // Beenden
193             // Gamification durchführen
194             graderHelper.gradeForGame(bluetoothPercList.size,
                ↪ bluetoothPercList)
195             val fragment =
                ↪ graderHelper.checkBadgesForCompletion(bluetoothPercList)
196             val activity = context as FragmentActivity
197             activity.runOnUiThread { fragment?.show(activity.」
                ↪ supportFragmentManager,"badgecompleted")
                ↪ }
198
199             // Bluetooth- und Berge-Listen leeren
200             bluetoothPercList = mutableListOf()
201             randomHills = mutableListOf()
202         } else {
203             // Starten -> Berge generieren
204             genRandomHills()
205         }
206         // View neu zeichnen
207         invalidate()
208     }
209
```

```
210     // Bluetooth-Service setzen
211     fun setBluetoothNoService(bb : BluetoothNoService){
212         bluetoothNoService = bb
213         bluetoothNoService.startReading()
214     }
215 }
```

Quellcode 3: Die `PlaneGameView`-Klasse mit der Implementierung des Minispiels

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 =
       ↪  PreferenceManager.getDefaultSharedPreferences(context)
14     private val activity = context as Activity
15     private lateinit var dbHelper : GamificationDBHelper
16
17     // XP für Übungen vergeben
18     fun gradeForExercise(data : MutableList<Float>){
19         // Minimal-, Maximal- und Durchschnittswert der Liste bestimmen
20         val min = data.min()?:0f
21         val max = data.max()?:0f
22         val avg = if (data.average().isNaN()) 0.0 else data.average()
23
24         // Bewertungsfunktion
25         val points = ((avg + min + max) / 3 )
26         val pointsInt = if(!points.isNaN() && points.roundToInt() > 0)
           ↪  points.roundToInt() else 0
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
           ↪  $pointsInt XP erhalten!",Toast.LENGTH_LONG).show() }
32     }
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.roundToInt() > 0)
           ↪  points.roundToInt() else 0
```

```
44
45          // neue XP zu bisherigen XP hinzufügen
46          val pointsBefore = prefs.getInt(PREFS_POINTS,0)
47          prefs.edit().putInt(PREFS_POINTS,pointsBefore + pointsInt).apply()
48          activity.runOnUiThread { Toast.makeText(context,"Du hast gerade
             ↪  $pointsInt XP erhalten!",Toast.LENGTH_LONG).show() }
49      }
50
51      // Freischaltung von Badges überprüfen
52      fun checkBadgesForCompletion(data: MutableList<Float>) :
         ↪  BadgeDialogFragment?{
53          dbHelper = GamificationDBHelper(context)
54          val cursor = dbHelper.getAvailableQuests()
55          var shouldShowDialog = false
56          var icon = 0
57          var title = ""
58          var earnedXP = 0
59
60          // durch alle vefügbaren Quests iterieren
61          while (cursor.moveToNext()){
62              val neededXP =
                 ↪  cursor.getInt(cursor.getColumnIndex(QUEST_FIN_XP))
63              val minPerc =
                 ↪  cursor.getInt(cursor.getColumnIndex(QUEST_MIN_PERC))
64              val isMinPercCompleted = data.any { it >= minPerc.toFloat() }
65              val isOverPercCompleted = checkOverPercCondition(data,cursor.
                 ↪  getInt(cursor.getColumnIndex(QUEST_OVER_PERC)),cursor.
                 ↪  getInt(cursor.getColumnIndex(QUEST_TIME_OVER_PERC)))
66
67              // spezifische Bedingungen überprüfen
68              if (neededXP <= prefs.getInt(PREFS_POINTS,0) &&
                 ↪  isMinPercCompleted && isOverPercCompleted){
69
70                  // Quest fertig -> Badge freigeschaltet
71                  dbHelper.setQuestCompleted(cursor.getInt(cursor.
                     ↪  getColumnIndex(QUEST_ID)))
72
73                  // entsprechend XP vergeben
74                  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 BadgeDialogFragment().setDialogInformation(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

5.5 Funktionsweise des Benachrichtigungssystems

```
1  package de.lukasrost.apoplexy.notifications
2
3  import android.content.Context
4  import android.preference.PreferenceManager
5  import java.util.*
6  import android.content.pm.PackageManager
7  import android.content.ComponentName
8  import android.app.AlarmManager
9  import android.content.Context.ALARM_SERVICE
10 import android.app.PendingIntent
11 import android.content.Intent
12 import android.support.v4.app.NotificationCompat
13 import android.app.NotificationManager
14 import android.support.v4.content.ContextCompat.getSystemService
15 import android.app.NotificationChannel
16 import android.os.Build
17 import android.support.v4.app.NotificationManagerCompat
18 import de.lukasrost.apoplexy.*
19
20 // Funktionen zur Benachrichtigungsverwaltung
21
22 // Benachrichtigungszeitpunkt neu setzen
23 fun setReminder(ctx: Context){
24     val prefs = PreferenceManager.getDefaultSharedPreferences(ctx)
25
26     // wenn Benachrichtigungen aktiviert
27     if (prefs.getBoolean(PREFS_ALARM_ENABLED,false)) {
28
```

```
29          // Zeit aus Einstellungen lesen
30          val time = prefs.getString(PREFS_ALARM_TIME,
            ↪  "00:00")?.split(":")!!
31          val hour = time[0].toInt()
32          val minute = time[1].toInt()
33          val calendar = Calendar.getInstance()
34          val setcalendar = Calendar.getInstance()
35
36          // Zeit setzen
37          setcalendar.set(Calendar.HOUR_OF_DAY, hour)
38          setcalendar.set(Calendar.MINUTE, minute)
39          setcalendar.set(Calendar.SECOND, 0)
40
41          // bisherige Reminder löschen
42          cancelReminder(ctx)
43
44          // wenn Zeit heute schon vergangen -> für morgen setzen
45          if (setcalendar.before(calendar)) {
46              setcalendar.add(Calendar.DATE, 1)
47          }
48
49          // Receiver des Alarms setzen
50          val receiver = ComponentName(ctx,
            ↪  NotificationReceiver::class.java)
51          ctx.packageManager.setComponentEnabledSetting(receiver,
52                  PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
53                  PackageManager.DONT_KILL_APP)
54          val intent1 = Intent(ctx, NotificationReceiver::class.java)
55          val pendingIntent = PendingIntent.getBroadcast(ctx,
56                  DAILY_REMINDER_CODE, intent1,
57                  PendingIntent.FLAG_UPDATE_CURRENT)
58
59          // Benachrichtigung dem AlarmManager übergeben
60          val am = ctx.getSystemService(ALARM_SERVICE) as AlarmManager
61          am.setInexactRepeating(AlarmManager.RTC_WAKEUP,
            ↪  setcalendar.timeInMillis,
62                  AlarmManager.INTERVAL_DAY, pendingIntent)
63      } else {
64          // wenn Benachrichtigung deaktiviert -> Erinnerung löschen
65          cancelReminder(ctx)
66      }
67  }
68
69  // Alarm des AlarmManagers mithilfe von *Magie* löschen
70  fun cancelReminder(context: Context){
71      val receiver = ComponentName(context,
        ↪  NotificationReceiver::class.java)
72      val pm = context.packageManager
73      pm.setComponentEnabledSetting(receiver,
74              PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
75              PackageManager.DONT_KILL_APP)
76
77      val intent1 = Intent(context, NotificationReceiver::class.java)
78      val pendingIntent = PendingIntent.getBroadcast(context,
79              DAILY_REMINDER_CODE, intent1,
                ↪  PendingIntent.FLAG_UPDATE_CURRENT)
80      val am = context.getSystemService(ALARM_SERVICE) as AlarmManager
81      am.cancel(pendingIntent)
82      pendingIntent.cancel()
```

```
83  }
84
85  // Benachrichtigung anzeigen
86  fun showNotification(context: Context){
87
88      // Activity, die geöffnet werden soll
89      val intent = Intent(context, HomeNavActivity::class.java)
90      intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or
        ↪  Intent.FLAG_ACTIVITY_CLEAR_TASK
91      val pendingIntent = PendingIntent.getActivity(context, 0, intent, 0)
92
93      // Aussehen der Benachrichtigung
94      val builder = NotificationCompat.Builder(context,
        ↪  DAILY_REMINDER_CHANNEL_ID)
95              .setSmallIcon(R.drawable.ic_minigame_icon)
96              .setContentTitle("Zeit zum Üben!")
97              .setContentText("Hast du Lust, deine Schlaganfall-Übungen mit
                ↪  Apoplexy durchzuführen?")
98              .setStyle(NotificationCompat.BigTextStyle()
99                      .bigText("Hast du Lust, deine Schlaganfall-Übungen mit
                        ↪  Apoplexy durchzuführen?"))
100             .setPriority(NotificationCompat.PRIORITY_DEFAULT)
101             // Set the intent that will fire when the user taps the
                ↪  notification
102             .setContentIntent(pendingIntent)
103             .setAutoCancel(true)
104
105     // Benachrichtigung senden
106     val notificationManager = NotificationManagerCompat.from(context)
107     notificationManager.notify(DAILY_REMINDER_CODE,builder.build())
108 }
109
110 // Benachrichtigungskanal auf unterstützten Geräten erstellen
111 fun createNotificationChannel(context: Context) {
112     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
113         val name = "Erinnerungen"
114         val description = "Apoplexys Übungserinnerungen"
115         val importance = NotificationManager.IMPORTANCE_DEFAULT
116         val channel = NotificationChannel(DAILY_REMINDER_CHANNEL_ID, name,
            ↪  importance)
117         channel.description = description
118
119         val notificationManager =
            ↪  getSystemService(context,NotificationManager::class.java)
120         notificationManager!!.createNotificationChannel(channel)
121     }
122 }
```

Quellcode 5: Der `NotificationScheduler` zur Verwaltung der Benachrichtigungen

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich meine Seminarfacharbeit „Entwicklung eines Gamification-Unterstützungs- und Motivationsgeräts zur Rehabilitation von Schlaganfall-Patienten“ selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel verfasst habe. Außerdem erkläre ich, dass ich alle wörtlich oder sinngemäß aus fremden Werken entnommene Stellen als solche kenntlich gemacht habe.

Die vorliegende Arbeit entspricht in ihrer Vollständigkeit der auf der gegebenenfalls beigelegten DVD gespeicherten digitalen Version.

Erfurt, 20. Dezember 2018

_____

Lukas Rost

Digitale Version