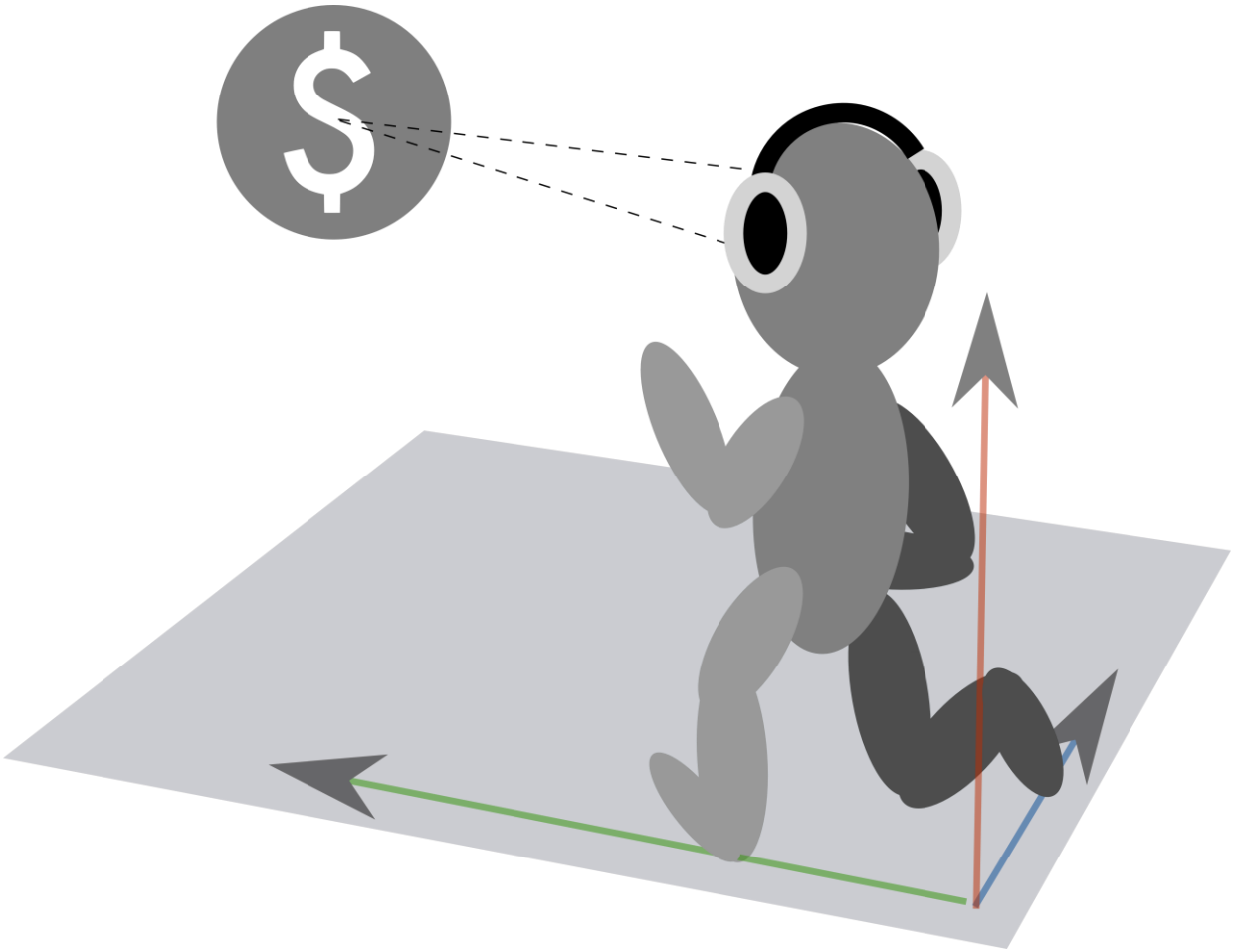
CHALMERS



Sound-guided running

Using GPS and binaural audio in an exercise application

Bachelor of Science Thesis in Computer Science and Engineering

MARCUS BERNHARD
DANIEL JOHANSSON
JOAKIM JOHANSSON
LINUS KARLSSON
ANTON PALMQVIST

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2014

Sound-guided running
Using GPS and binaural audio in an exercise application

Marcus Bernhard
Daniel Johansson
Joakim Johansson
Linus Karlsson
Anton Palmqvist

Examiner: Jan Skansholm

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover: Illustration of a user of the developed application navigating toward a virtual coin using their headphones as explained in subsection 3.6.1 Coin collector.]

Department of Computer Science and Engineering
Göteborg, Sweden June 2014

Abstract

This report describes the development of an exercise application for Android smartphones, called *Run For Life*. It combines the sensors found in present day smartphones with spatial audio feedback, creating an exercise application with virtual sound sources. The idea of combining spatial audio with smartphone sensors is new and the research is limited. By using the GPS sensor to calculate both the user's position and orientation, the project attempts to combine the spatial audio achievements made by Lawitzki (2012) with the running game elements of applications like *Zombies, Run!*. The audio was implemented using general, non-individualized *head related transfer functions*. An attempt to use the gyroscope sensor to calculate the orientation of a user standing still was made, often resulting in a confusing sound.

The application's main mode is game inspired. By providing a randomized running route and audio feedback to guide the user though its checkpoints, the application aims to enhance regular exercise. Run For Life also presents the opportunity for the user to view statistics of previous running sessions. A lot of time was put into designing the user interface in a way that makes the application appealing.

The report concludes that it is hard to make the perception of an angle realistic to the user because of the difficulty in sensing a user orientation. GPS technology is still too inaccurate to create a perfect perception of the sound direction by itself. Finally, determining a user's orientation is not reliable while the sensor is not at a fixed position relative to the user ears; a head-mounted sensor device would probably avoid most of these problems and provide a more realistic perception.

Sammanfattning

Denna rapport beskriver utvecklingen av en träningsapplikation för smartphones med operativsystemet Android, kallad *Run For Life*. Run For Life kombinerar sensorerna som finns i dagens smartphones med ljud i 3D. Följande resulterar i en träningsapplikation med inslag av spelmoment och virtuella ljudkällor. Idén med att kombinera 3D-ljud med smartphonesensorer är ny och forskningen inom området är hittills mycket begränsad.

Projektet kombinerar resultaten inom 3D-ljud som gjorts av Lawitzki (2012) med spelelement från applikationer som *Zombies, Run!*. Mobiltelefonens GPS-sensor används för att beräkna användarens position och färdriktning, medan ljudet implementerades med hjälp av så kallade *head related transfer functions*. För att kunna få en riktning även när användaren är stillastående gjordes ett försök att använda gyroskopsensorn, men ljudet blev då ofta mer förvirrande än när bara GPS-sensorn användes.

Applikationen innehåller spelmoment och hoppas kunna uppmuntra användarna till en mer regelbunden träning. Detta görs genom att generera en slumpmässig löprunda innehållande kontroller (virtuella ljudkällor) och med hjälp av ljud vägleda användaren i rätt riktning mot dessa kontroller. Run For Life gör det också möjligt för användaren att se historik om sina tidigare löprundor. Mycket tid har lagts på att utforma ett användargränssnittet som ska vara tilltalande och användarvänligt.

I rapporten dras slutsatsen att det kan vara svårt för användaren att uppfatta riktningen av ett ljud. Detta då GPS-sensorn i nuläget ej är tillräckligt exakt för att utan kombination med andra sensorer erhålla en bra färdrikting. Slutligen är det svårt att avgöra hur användaren är riktad vid ett stillastående läge, då telefonens placeringen på användaren är okänd. Om en huvudmonterad enhet används bör problemen med den okända placeringen undvikas och riktningen till ljudkällan upplevas mer exakt.

Acknowledgments

We would like to kindly thank Prof. Graham Kemp for his valuable guidance while writing this report. Without any external stakeholders in the project, his competent advices has been even more appreciated.

We would also like to thank Arne Linde and Jan Skansholm, for being the examiners of this course and for giving us approval to realize this project.

Furthermore, we would like to thank our friends and families for helping us test our application, as well as for their support.

Gothenburg, May 16, 2014

Marcus Bernhard
Daniel Johansson
Joakim Johansson
Linus Karlsson
Anton Palmqvist

Contents

1. Introduction

In recent years, smartphones with *GPS (Global Positioning System)* sensors have become increasingly common, encouraging the development of applications using it to track the user's position and movement (M2 Communications 2012). One such example is *Endomondo (Endomondo Sports Tracker)*, launched in 2008 (Endomondo 2014a). The application constantly tracks the position of the user as they are moving and uses the information to calculate statistics such as speed and distance (Endomondo 2014b).

Zombies, Run!, released in 2012, is another exercise application focusing on making running more enjoyable by introducing game elements (Six to Start 2014). It uses non-binaural auditory cues to create the illusion of the user being chased by zombies. These cues are independent of the user's position, meaning that the route of which the user is running is unimportant to the game. When increasing the speed, thus running faster, the user runs away from the zombies, resulting in the sound effects (imitating approaching zombies) being silenced.

Binaural sound was introduced in computer games already in the nineties, used by games such as *Unreal Tournament* (Brieger and Göthner 2011). In these games, however, the sound positioning is based on a virtual character - not the actual user. A study in 2012 examines the possibilities of tracking the user's head movements using a smartphone in combination with binaural audio (Lawitzki 2012). By virtually position a sound in headphones, head movements create an effect similar to what humans would hear in real life.

This report aims to further investigate the possibilities of using the smartphone and its sensors together with binaural audio to create an exercising application with game elements; by combining GPS technology and virtual sound sources the user can navigate using the sound appearing from their headphones.

1.1. Problem assignment

1. Translate location data obtained with sensors into navigation instructions with sound.
2. Register running activity and present its statistics.
3. Motivate users to exercise by implementing game elements.

4. Create a graphical user interface that is straightforward and easy to understand.

1.2. Purpose

The purpose of this bachelor project is to design and implement an Android application combining GPS technology and binaural audio to guide the user without them having to interact with the display of the smartphone. Based on the user's direction and location, the sound is modulated and panorated spatially, creating the effect of the sound appearing from a certain angle.

By introducing exciting elements to classic exercise, the application hopes to encourage physical activity and thus prevent sedentary amongst its users. Apart from making running into an enjoyable experience, the application will register running activity and present statistics. To enhance the usability of the application it will have a straightforward graphical user interface that is easy to understand.

1.3. Method

The development process can be divided into two major phases - the first one being the research necessary to understand the subject. In the second phase, the theoretical framework obtained was used to build the application.

An agile development method was used by having daily meetings and working together most of the time. This way problems were found at an early stage and could be solved directly. The tools used during the development are described below.

1.3.1. Development environment and version control

Run for Life was developed using an Integrated Development Environment (IDE) called Eclipse, together with the Android Development Tools (ADT) plugin, providing the tools necessary for creating an Android applications (Android Developers 2014b). It provides graphical tools for designing the user interface and for debugging, among other things.

To simplify the collaboration process, the application was developed using the revision control system Git, hosted on a free Github account. Git makes it possible for groups of people to edit the same documents simultaneously without the work getting overwritten. To host projects on Github for free, its content needs to be visible for others to read (GitHub 2014).

1.3.2. Libraries

In order for the application to integrate with Google Maps, the Google Play services library had to be included in the project. It is developed by Google and contains features such as maps, Google+ and analytics.

Android's Software Development Kit (SDK) does not currently provide any operations necessary to implement binaural audio. By using the Android's Native Development Kit (NDK) in combination with OpenAL Soft, it is possible to access low-level functions to position and move audio in three dimensions.

The application has been developed for devices using platform version 4.0 and above, corresponding to an API level of 14. It offers many functionalities and is compatible with the majority of the devices on the market today, see Table 1.1. Statistics from the beginning of May 2014 shows that platform version 4.0 will cover more than 80% of the devices (Android Developers 2014k).

Table 1.1.: Visualization of the percentage of platform versions used on devices linked to Google Play. The compatibility column indicates the percentage of devices supported if the corresponding API level is chosen. The compatibility was calculated by summing the usage for each succeeding API level.

Data collected from Android Developers (2014k).

API level	Usage	Compatibility
8	1.0%	100%
10	16.2%	99%
13	0.1%	82.8%
15	13.4%	82.7%
16	33.5%	69.3%
17	18.8%	35.8%
18	8.5%	17%
19	8.5%	8.5%

1.3.3. Programming language

Developing an application using the ADT plugin requires most of the application to be written in Java (Android Developers 2014b). To define the visual structure of the user interface, the markup language XML was used. The database was created and managed using the programming language SQL.

1.3.4. Interaction design process

To design a usable application, the interaction design process was applied (Rogers 2013). This process consists of establishing requirements, designing alternatives and

creating prototypes and also evaluating, which was done with user evaluations on test persons.

1.4. Limitations

This project will primarily focus on implementing the functionality needed in order to achieve the purpose. The application was not designed to optimize other factors such as energy efficiency and security.

1.5. Report Outline

The following section describes the overall layout of the report.

Theoretical Framework (chapter 2) covers the theory behind the most important parts of the application: sound, sensors and how a random location can be generated by using predefined algorithms. Furthermore, the basic structure of databases, Android applications and graphical design patterns are explained

The Implementation (chapter 3) describes how the different parts in (chapter 2) was implemented into the application. It also covers the findings obtained from the user evaluation.

The last parts of the report consist of the result (chapter 4), discussion (chapter 5) and the conclusion (chapter 6). It begins by describing the end product, that is the application. Thereafter, the result along with the entire design and implementation process is discussed. Future work is also covered in this section. Lastly, in the conclusion, the projects major findings are reflected.

2. Theoretical Framework: Sound, sensors and design

The following chapter is a compilation of the theory that supported the development of the application. It describes how the sound is interpreted by humans as well as how some of the sensors in a smart phone work. Furthermore, database structure and various design patterns concerning graphical user interfaces are described.

2.1. Sound: localization and interaction

The functionality and usability of the application depend on how the user perceives and interprets the audio coming from the device. If a user were to become confused by the sound, they would probably not be able to follow its instructions, hence defeating the purpose of the application. It is therefore vital to have an audio core that is easy to understand and behaves as expected.

2.1.1. Sound localization

There are several auditory cues for humans to determine where a sound source is located. Two of these cues are the interaural level difference (ILD) and the interaural time difference (ITD) (Algazi and Duda 2011). ILD stands for the difference in volume between the closest ear and the ear being the farthest from a sound. ITD, on the other hand, is the time between the sound reaching each ear. Both are results of the head and torso shadowing the sound waves (Palomäki et al. 2011).

When the sound reaches the pinna, located in the outer ear, it is filtered even further (Algazi and Duda 2011). Along with the directional characteristics of the torso and head, the pinna helps us determine where a specific sound source is located. Its filtering properties (caused by the shape) even make it possible for us to differentiate between sounds appearing from the front and from above, as can be seen on Fig. 2.1. Naturally, the properties of the pinna are of great value when trying to recreate a natural environment using only two speakers. Such environments can be very useful when simulating virtual realities, like in 3D games and architecture (Kleiner 2011).

To create virtual audio environments, one can use so called head-related transfer functions (HRTFs) (Grubesa and Jauk 2004). They aim to describe how a sound

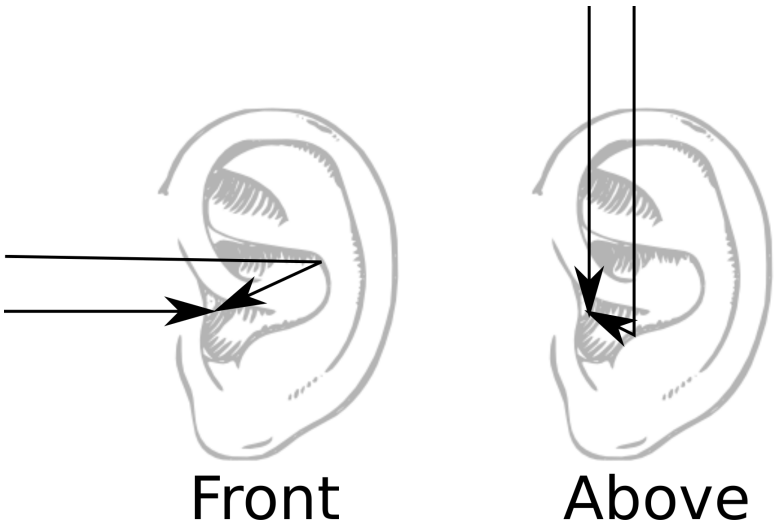


Figure 2.1.: The pinna filters the sound differently depending on its location.

signal is filtered before reaching the eardrum - as a result of the reflection caused by the pinna, torso and head - and hence being perceived by us humans. With HRTFs, it is possible to create the illusion of a sound signal being heard in an anechoic chamber using stereo headphones (Algazi and Duda 2011). That, however, assumes individualized HRTF measurements. Since the pinna is shaped differently on each person, the result might be more or less convincing depending on how well the measurements are made (Algazi and Duda 2011). Ways to capture data in the best way possible, hence overcoming the measuring problems, are currently being developed by Spatially Oriented Format for Acoustics (SOFA 2013). SOFA is a file format in which acoustic data is stored, eliminating the problem with researchers storing their data in different ways.

At present, very few studies seem to have been conducted on sound localization through spatial audio. A study made on four sighted individuals showed that the average error in localizing sound was 8° (Barreto 2009). Under the same conditions, four blind individuals showed an average error of about 4°. The test was done by placing speakers around the subject at ear level, hence not covering elevation localization. Sounds coming from above and below our ears proves to be more difficult to localize, judging by the result of a study made in 2009 at the Florida International University (Barreto 2009). Experiments were made by playing short sounds spatialized using both individualized and generic HRTFs, with the virtual sound's position only moving in the vertical plane. The study shows that the accuracy of the elevation perception decreases as the virtual sound is moved further up and down.

A general problem with non-individualized HRTFs is that, due to different shapes of the pinna, confusion whether a sound is appearing from the front or back might appear (So et al. 2010).

2.1.2. Human-Computer Interaction using sound

Kramer (1994) mentions several reasons to why using audio feedback can be advantageous over visual feedback. Firstly, it reduces the demands on visual attention. An issue with today's smartphones is that it is hard to pay attention to the screen while being on the move. By giving information to the user through sound, the need to look at a screen would be eliminated, enabling the user to focus on their surroundings. Another advantage of using audio feedback is that it grabs the user's attention (Kramer 1994). A person can easily choose not to see something; on the contrary, it is difficult to choose not to hear something.

When it comes to providing the user with continuous information, nonspeech sounds seem to have some advantages over speech (Sears and Jacko 2007). Just as with textual feedback, speech lacks expressive capability, meaning that describing something fairly simple might take many words. By using nonspeech sounds, information can be presented both at a faster rate and through shorter messages; instead of having something described with words, the user can instead recall the meaning of a sound. Continuous sounds often gradually fade into the background until the sound changes or stops playing - often referred to as habitutation. Because of the large dynamic range of speech, habituation cannot easily be achieved using it, hence adding to the annoyance of audio feedback previously mentioned. Overall, nonspeech sounds have been proven to be a better choice when providing the user with continuous information (Sears and Jacko 2007).

Studies of environmental noises and speech show that excessive sound volume is the primary reason for annoyance to the user (Sears and Jacko 2007); a loud sound is attention-grabbing even though its intended message is unimportant. To avoid annoyance, it is therefore suggested to avoid using sound volume as a cue in applications. Instead, the authors of The Human-Computer Interaction Handbook (2007) recommends using pitch and rhythm as primary cues, as changes in stimuli are easier detected by the human auditory system.

While depending on audio feedback rather than visual enables the user to focus more on its surroundings, it has its disadvantages - one of the biggest being the low resolution (Kramer 1994). It is difficult to perceive small differences in sounds, such as changes in sound intensity and panoration. Furthermore, the attention-grabbing property previously mentioned might lead to annoyance amongst some users (Kramer 1994). According to Spears and Jacko (2007), there are two types of sounds: information and noise - the former helping us and the latter derailing us (Sears and Jacko 2007). Naturally, a sound that one person interprets as informative might be noise to another; designing the right sounds to use in an application is a process that should not be overlooked.

2.2. Android sensors: Differences and comparison

In order to correctly play the sound, it is important to determine the direction in which the user is facing; the *user direction*. Since the orientation of the phone relative to the user is unknown (a user can carry a phone in many ways), the problem of acquiring a user direction is not elementary. However, this report investigates whether a sufficiently accurate user direction can be calculated using a combination of multiple sensors.

2.2.1. GPS

GPS (Global Positioning System) is used to determine the global coordinates of a device in latitudes and longitudes through the use of satellites (National Coordination Office for Space-Based Positioning, Navigation, and Timing 2014). The 24 satellites that are being used for this are situated in six different orbits circling the Earth in a manner that assures visibility of at least five of them no matter of time and location on the Earth (Bajaj and Lalinda Ranaweera 2002).

To establish locations of devices, the satellites continuously send out *PNCs (Pseudorandom Number Codes)* (Bajaj and Lalinda Ranaweera 2002). As illustrated in Fig. 2.2, each PNC consists of information of the satellite's location and time at the moment of transmission. When acquiring a PNC, its travel time ($\triangle t$) is calculated by the GPS receiver by subtracting the departure time ($t_{departure}$) from the arrival time ($t_{arrival}$) as in Formula 2.2. A PNC travels at the speed of light (c), which in combination with the travel time ($\triangle t$) is enough information to calculate the distance ($\triangle x$) between the satellite and receiver (see Formula 2.1).

Figure 2.2.: *Pseudorandom Number Codes* are sent from satellites and picked up by GPS receivers. A PNC contains information about the satellite's location and the time at the moment of transmission.

$$\triangle x = c \cdot \triangle t \tag{2.1}$$

$$\triangle x = c \cdot (t_{arrival} - t_{departure}) \tag{2.2}$$

Location wise, this distance does however only determine that the GPS receiver is located somewhere at the surface on a sphere with the radius being the distance ($\triangle x$) and the center being the satellite. To acquire the GPS receiver's location on the sphere, the method *trilateration* as shown in Fig. 2.3 (Bajaj and Lalinda Ranaweera 2002) is used.

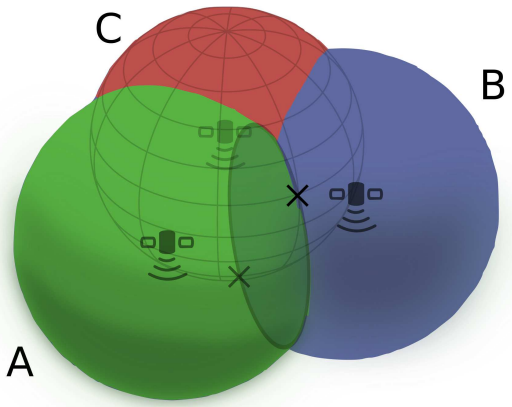


Figure 2.3.: Location finding using trilateration.

In the context of GPS technology, trilateration is a way to narrow down the possible locations by determining where spheres intersect (Bajaj and Lalinda Ranaweera 2002). All visible satellites each provide one sphere of possible locations. Two satellites will narrow down the possible location to a circle as shown at the intersection of Satellite A's and Satellite B's spheres in Fig. 2.3. Furthermore the third satellite, Satellite C, will narrow down this circle into two points. Normally, this information is enough since often only one of the two provided points is located on the surface of the Earth. It is however useful to use more satellites to acquire a confirmed location. Also, another satellite might be used to synchronize the GPS receiver's internal clock. When a receiver has a fixed position it can be used along with another satellites position to determine their intermediate distance ($\triangle x$), which in combination with the speed of light will give the travel time of a PNC between them. By adding the calculated travel time to the departure time of the PNC the correct current time in the GPS receiver is acquired. This is good practice since the satellite's clock, that is being synchronized against, is atomic and thereby very accurate.

In this project, the GPS technology is mainly utilized to calculate the user position relative to the virtual sound sources. However, by keeping track of previous GPS locations, it is possible to acquire a user bearing. This bearing is the direction in which the user has been traveling during the most recent sensor updates and has

the great advantage of not being dependent on how the user carries the device at all (Android Developers 2014h). The disadvantage is however that the user has to be in motion for the bearing to update correctly. If the user is standing still or turning around rapidly the bearing will not change.

There are several formats that can be used to represent a location on the earth. This project is using the geographic coordinate system with longitude and latitude (Ordnance Survey - the national mapping agency of Great Britain 2002). As seen in the Fig. 2.4 the latitude, φ, is the angle between the equator and the poles. It is zero at the equator, 90° N at the north pole and 90º S at the south pole. The longitude, λ, is the angle east or west of the reference meridian. The reference meridian goes from one pole to the other and passes through the Royal Observatory, Greenwich in England. This Greenwich meridian has a longitude value of 0° and the antipodal meridian has the values of 180° W and 180° E.

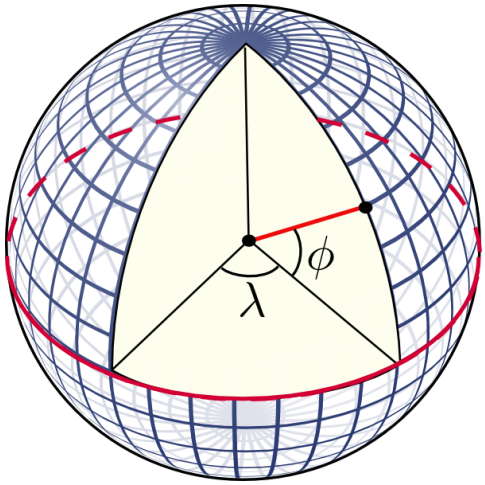


Figure 2.4.: Figure showing definition of latitude (φ) and longitude (λ) on a sphere (Mercator 2010).

2.2.2. Accelerometer

The accelerometer gives the linear acceleration of the device in meters per square second in the device's coordinate system (Android Developers 2014i). As illustrated in Fig. 2.5, the accelerometer gives values as a 3-dimensional vector $a = \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix}$, where a_1 is the rightward movement speed, a_2 is the forward moving speed and a_3 is the upward moving speed. Although the coordinate system is based on the device, a gravitational acceleration is always measured toward the ground. This makes it possible to convert measurements in the coordinate system of the device into a global coordinate system. For example, these values can be used together with magnetic

field sensor to obtain device orientation (basically a compass), or together with the gyroscope to obtain device rotation in a global coordinate system.

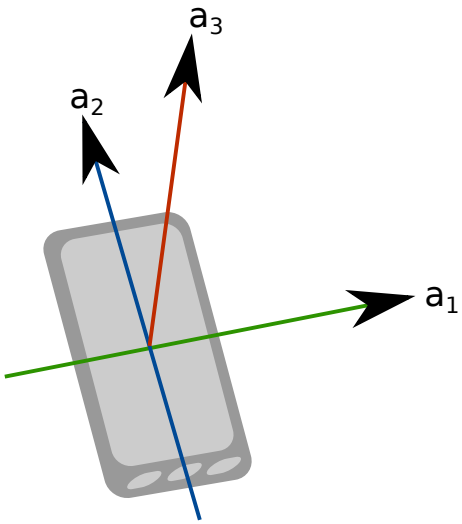


Figure 2.5.: Coordinate system and measured vector of the accelerometer. The accelerometer measures linear acceleration on the x-, y-, and z-axis in meters per square second.

2.2.3. Magnetic field

The magnetic field sensor determines current ambient magnetic field strength in micro teslas as a 3-dimensional vector (Android Developers 2014i). As mentioned above, it can be used together with the accelerometer to obtain device orientation.

2.2.4. Gyroscope

Gives the rotational speed of the device in radians per second around each of its axes (Android Developers 2014i). While it has a lot less disturbances than an accelerometer and magnetometer compass, it only determines the change of direction and not the direction itself. Hence, a gyroscope can report that the phone has rotated 90 degrees around the x-axis; but without knowing at what angle it started, one cannot know at which angle it has ended up (only that it is 90° larger).

As seen in Fig. 2.6 the gyroscope has the same coordinate system as the accelerometer, the axes here labeled x, y and z. The values are measured by the gyroscope as a 3-dimensional vector $g = \begin{Bmatrix} g_1 \\ g_2 \\ g_3 \end{Bmatrix}$, where g_1 is the rotation around the x-axis of the device, g_2 is the rotation around the y-axis and g_3 is the rotation around the z-axis.

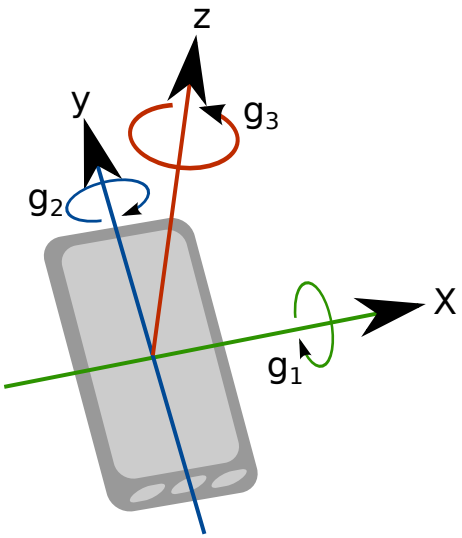


Figure 2.6.: Coordinate system and measurement vector of the gyroscope. The gyroscope measures the rotational speed around the x-, y- and z-axis in radians per second.

2.3. Generation of random locations: Different approaches

To enable the user to instantly start running using the application, the sound sources (checkpoints) are generated using a predefined algorithm. The checkpoints must be generated at reachable locations and can for example not be placed in a lake or inside a building, but instead on a road or trail. During the project two methods to generate a location that is reachable was invented and is described in this section.

2.3.1. Colour classification method

The first method generates a random location nearby the location of the user. This location is evaluated if it is located in an appropriate environment by extracting the colour that the location has on a map. By classifying all the colours on a map by its colour, for example blue is equal to water and light green is equal to grass, one can compare the colour of a location and classify it thereafter. If the colour is blue, the location is most likely located in water of some kind. In this scenario a new random location has to be generated and evaluated in the same way. Only if the location has a colour that is accepted, for example light green, it is saved and can be used in the route. For this to be implemented, the map has to have accurate colour regions and be linked to some kind of coordinate system. It must also be possible to extract the colour of the random location.

2.3.2. Google geocoding API request method

Another approach is to use Google maps and its geocoding API (Google Developers 2014). Initially, a random location is generated as in the first method. Then a geocoding API request is sent to Google asking for the directions from the users location to the random location. The response from Google contains the directions to the closest reachable point to the requested location. This closest point is then extracted and used as the random location.

In this project the second approach is implemented and this is described in sec. 3.3. The different approaches is also discussed in sec. 5.3.

2.4. Database: Relational model and visual representation

The information retrieved from the GPS when the user is running (along with other information, such as time and distance) has to be stored in a way that makes it accessible at any time. This is done by using a database.

A relational database collects data in tables. The columns of a table represent the different kinds of data stored and each row is a specific object containing these data. It is important to have a column that stores an unique identifier (ID) for each row. This makes it possible to obtain a specific row from the database and to specify relationships between the different tables.

An example of a simple database is that of a hotel. Room type, price per night and an ID would be the columns of the table. In this case the ID could be the room number.

If a user would like information about room number 123 then a request would be made to the database with the ID "123" and the answer would consist of the row highlighted in Fig. 2.7. That would be that the room is a double room and the price of the room is 100 per night.

The first step when designing a database is to make a visual representation of it. This can be done with a so called ER-Diagram (Entity/Relationship-Diagram). An ER-Diagram consists of three main parts: entities, relationships and attributes (Garcia-Molina and Ullman 2009, ch. 4.1). The entities are the different objects that will be stored in the database and the attributes are the different data of the object. In the previous example, the Room could be the entity and room number, room type and price be the attributes of the room. The entities are what will become the tables of the database and the attributes are the columns of the tables. The relationship part of the ER-Diagram holds relationships between different entities of the database, for example it could hold the relationship between hotels and their rooms. In the ER-Diagram entities are represented by rectangles, attributes by

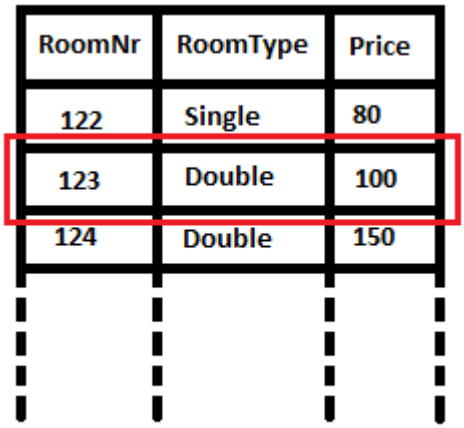


Figure 2.7.: A simple table describing how a hotel room could be represented in a database.

ellipses and the relationships by diamonds. In a relational database a relationship is often a separate table that stores the relationship between different entities.

To create and manage the database a DBMS (Database Management Systems) needs to be used. The DBMS is a system that allows the application to communicate with the database, it makes the necessary requests to the database using SQL (Structured Query Language). The DBMS handles the requests such as adding and retrieving data from the database.

2.5. Structure of Android applications

To understand the implementation in chapter 3, the different concepts of Android applications will here be explained. The core of Android applications is the concept *Activity* (Lehtimäki 2013). An Activity is a controller handling the components visible on the screen. These components are called *User Interface Widgets* and consist of everything from buttons to more complex components like gallery widgets. User Interface Widgets are collected in containers called *Layouts*. Each Activity has a Layout and communicates with it during its controller process.

Layouts may also belong to a so called *Fragment* which basically is a Child-Activity in which parts of the content of the owning Activity can be put. Fragments are modular sections of an Activity meaning that they are flexible in a way that lets them adapt to the current screen size (Android Developers 2014e). If an Activity contains one Fragment displaying a list and one Fragment displaying a detailed view of a list element, the small screen on a mobile device might only show one of these Fragments, while a tablet with a larger screen might show them both at the same

time. This way the same code can be used no matter what device is intended to run the application.

The communication between Activities is done with messaging objects called *Intents* (Android Developers 2014g). These Intents can be explicit, meaning that they are used to start a certain Activity within the application itself, but may also be implicit letting the operating system choose another application to handle the action that the intent wants to perform. Intents can include extra data if it is needed to perform the desired action.

2.6. Graphical design

A graphical user interface (GUI) is a way to interact with an application. There are different patterns that can be applied when designing GUI components.

2.6.1. Standard design patterns

Tidwell (2011) states that easily used applications are designed to be familiar. It is further discussed that when parts of an application are recognizable and have clear relationships between each other the users will be able to apply previous knowledge and understand what to do.

The design pattern concept is a way to capture and group common structures together, thus easing the usability when applied. Below follows a description of the patterns that are used.

Prominent Done button One common design pattern is the *Prominent Done Button (Tidwell 2011),* which is used in the final step of a transaction. The pattern is applies by creating a button that actually looks like a clear button, and not a link or another navigation element. It is often good practice to also make a Prominent Done Button large and with bold colours and well defined borders, thus making it stand out. The placement of the button should be after the elements related to the work flow of a task, which usually is on the bottom right of a page. In the example in Fig. 2.8 the sign up-button is prominent and thereby clearly telling the user that it should be pressed after the form has been filled out.

Carousel and Filmstrip pattern Another design pattern is the *Carousel*, which is a horizontal swipeable or scrollable list of items that are visually interesting (Tidwell 2011). The pattern is useful when the user does not know exactly what item they wants and thereby wants to browse between the options. A picture representing an item and optionally some meta data such as name, is usually enough representation in a Carousel. In Fig. 2.9a it is clear that the item in the middle is in focus since it

Figure 2.8.: Example of a *Prominent Done Button.*

is enlarged. The elements immediately around the focused item will give the user a clue about the possibility to browse.

The *Filmstrip* pattern basically is a Carousel for mobile devices. A difference between Filmstrip and Carousel is that Filmstrip usually fills up the whole screen horizontally (Tidwell 2011). This is generally good practice when the screen is small, which usually is the case for mobile devices. There is no space for clues of the adjacent elements on the sides, which usually is solved by small dots in the bottom of the screen as shown in Fig. 2.9b.

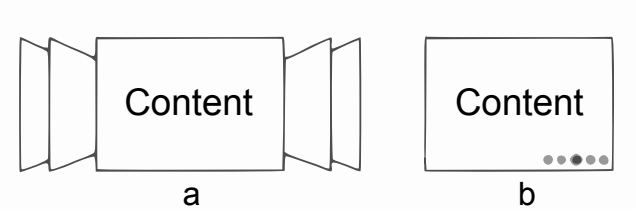


Figure 2.9.: Concept of the Carousel (a) and Filmstrip pattern (b).

2.6.2. Android design patterns

Android standard design patterns are used where applicable in the application. A great advantage of using Android design patterns is that the application becomes consistent to other applications on the Android platform (Lehtimäki 2013). Below follows a description of the patterns that are used.

Action Bar pattern One of the most defining design components in Android is the *Action Bar* pattern. As the name suggests it is a bar containing components, as the ones numbered in Fig. 2.10, which can perform actions to the application. The Action Bar has been around for a long time and is one of the most recognizable patterns (Lehtimäki 2013).

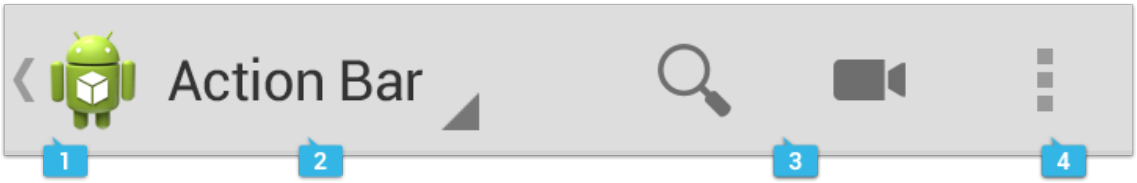


Figure 2.10.: An Action Bar and its components: Application Icon (1), View control (2), Action Buttons (3) and Action overflow menu (4).
(Android Developers 2014a)[1]

The most important feature of the Action Bar is perhaps, as the name suggests, its actions. The actions that are most important in the application should be located among the Action Buttons as in (3) in Fig. 2.10. To decide if an action is important one might consider how frequent it will be used and whether it is typical and contextual for the view in question (Lehtimäki 2013). A final criteria might be whether the action may work as a good selling point of the application, as for example if it represents a neat feature (Android Developers 2014a).

The actions that, according to the above criteria of importance, does not belong among the Action Buttons should be located in the overflow menu instead. The overflow menu is opened by a click on the menu button (4) in Fig. 2.10. If the phone however does have a hardware menu button, it will instead be used to open the overflow menu and the three dotted button will not be visible (Android Developers 2014a).

An advantage with the Action Bar pattern is that it displays the application logotype, (1) in Fig. 2.10, which establishes the identity of the application. When an application has different views its logotype becomes a consistent identifier of the application. This lets the user know which application is currently running. The application logotype can also have the function of an up button taking the user one step up in the hierarchy, as is the case in (1) in Fig. 2.10, or a home button taking the user to the home view (Android Developers 2014a). The button of the application logotype may also be used to open a side menu called *Navigation Drawer* (which will be covered in sec. 2.6.2) with more navigation options (Lehtimäki 2013).

Another component giving a sense of location is the View control (2) as in Fig. 2.10, which apart from displaying the name of the application or the current view may provide navigation options to different views. In this case navigation may be done by choosing views from a drop down menu.

Navigation Drawer Android Developers (2014j) suggests that actions should be put to the right and navigation to the left. However, there is a risk that putting

[1] Fig. 2.10 and Fig. 2.11 are reproduced from work created and shared by the Android Open Source Project (https://code.google.com/policies.html) and used according to terms described in the Creative Commons 2.5 Attribution License (http://creativecommons.org/licenses/by/2.5/).

navigational options into the Action Bar, as in (2) in Fig. 2.10, may make the view cluttered due to limited space. To avoid this the design pattern *Navigation Drawer* can be used. The Navigation Drawer is a menu that may be dragged from the left part of the screen or accessed by pressing the application logotype as shown in the left figure in Fig. 2.11. The menu partly covers the current screen and is used to open a desired screen as shown in the middle and rightmost figures in Fig. 2.11 (Android Developers 2014j).

When a view is provided with a navigation drawer it is important to let the user know about its existence by putting the navigation drawer indicator of three stripes to the left of the application icon as shown in Fig. 2.11.

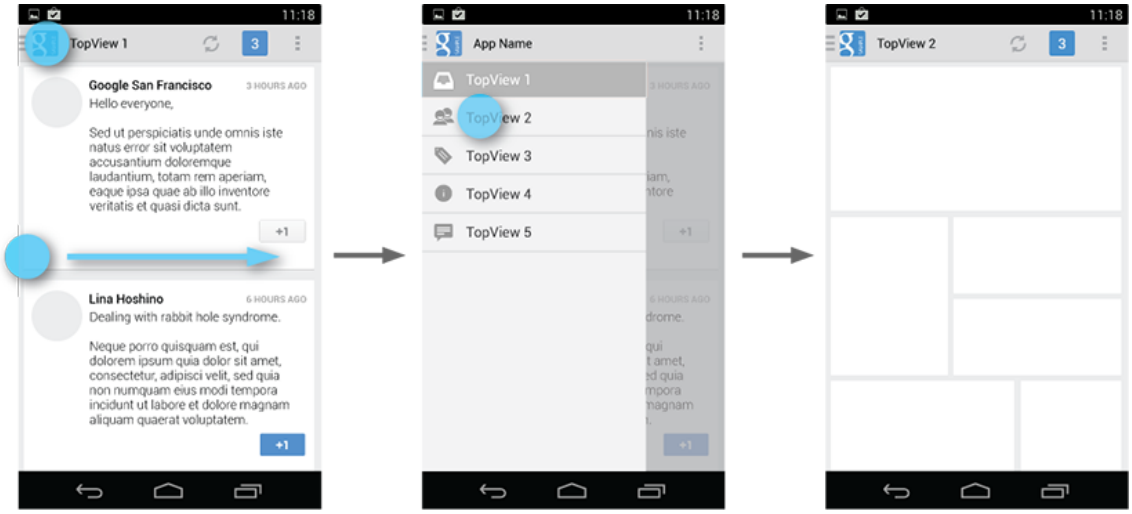


Figure 2.11.: The left figure shows how the Navigation Drawer is opened. The middle figure shows an opened Navigation Drawer. The right figure shows when a screen is opened through the Navigation Drawer.
(Android Developers 2014j)[1]

Navigation Drawer provides quick access to different views of the application without forcing the user to go back via some main menu like a *Dashboard*. The Dashboard pattern is basically a landing screen consisting of big icons which take the user to different parts of an application (Lehtimäki 2013). The Dashboard pattern used to be one of Google's official recommendations but has, more or less, been replaced by the Navigation Drawer since it provides more direct access throughout an application. The avoidance of the Dashboard pattern is especially effective at an application's start-up since the user then can be taken directly into the view that is mostly used instead of selecting it. From there the user may quickly switch to other views with the Navigation Drawer, thus eliminating redundant clicks back and forth from a Dashboard.

Swipe Views As mentioned above, the Navigation Drawer provides quick navigation between different views. However, there are times when functionality that logically belongs to the same view does not fit into a single screen. This can be solved by the pattern *Swipe Views* which puts different parts on different tabs that the user can swipe horizontally between (Lehtimäki 2013).

This is considered as a good solution as long as the number of tabs are limited to three or less (Lehtimäki 2013). If the number exceeds three however, it may be better to use a pattern called *Title Strip* where the user may swipe between an unlimited number of tabs.

3. Implementation

The following chapter describes how the the different systems were implemented into the application. The implementation part is based on the theoretical framework from chapter two and the different system elements are presented in the same order.

3.1. Sound in an Android application

The API*s (*Application Programming Interfaces*)* provided by Android as a part of the *SDK (*Software Development Kit*)* provides several classes for handling audio, like AudioTrack, MediaPlayer and SoundPool. However, none of these currently provide any support for 3D audio operations (Ratabouil 2012). To use binaural audio effects in an Android application, one needs to access operations that are not available in the SDK. Fortunately, Android provides an API for accessing low level functions called NDK *(*Native Development Kit*)* (Android Developers 2014c).

Android's NDK allows developers to implement part of an application using other languages than Java, such as C. The use of low-level languages does not generally result in better performance and on top of it, it also increases the application's complexity (Android Developers 2014c). However, it enables developers to use operations that are not accessible using only Java - such as handling 3D audio. Currently, there seem to be two APIs available providing this functionality: OpenSL ES and OpenAL Soft — both using general HRTFs (Khronos 2014)(*OpenAL Soft.* 2014).

3.1.1. Binaural sound using frameworks

OpenSL ES is a cross-platform API that enables audio functionality in native applications on multimedia devices (Ratabouil 2012). It has been part of Android's NDK since version 2.3 and is divided into three different profiles: game, music and phone. For a phone to take advantage of the functions provided by such a profile, it needs to have an appropriate compilation. The only profile enabling functions to spatially position audio is the game mode (Khronos 2014). There is very little information available on what phones supports what profile. In Android Beginner's Guide (2012) the author mentions that no devices support the game profile at the time the book was written.

OpenAL Soft is another API that enables the creation of spatial sound environments independent on the soundcard setup (*OpenAL Soft.* 2014). As a result, it eliminates

the problem with programmers having to write different code to each device because of each soundcard supporting different instructions. It is licensed under GNU Lesser General Public License, meaning that one can freely include it in software — even for proprietary (closed source) and commercial use (GNU 2014). Unlike OpenSL ES, OpenAL Soft is not part of the NDK provided by Android. To access its functions it is necessary to interact with code written in another language than Java.

Because of OpenSL being mostly unsupported, Run For Life was developed using OpenAL and OpenAl4Android as audio core. OpenAL4Android is a small library provided by Martien Pielot that serves as an intermediary between the native OpelAL Soft code and the Java application (Pielot 2010).

3.1.2. Sound modulation

Before going into detail, it might be worth quickly describing how the spatial positioning of the audio is determined in our application. In sec. 2.3, it is described how points can be generated on a map using GPS technology. As the user is moving, their angular offset in terms of such a generated point can be calculated using Sensor fusion (see sec. 3.2), varying from -180° to 180°. An angular offset of 0° would hence mean that the user is moving perfectly in the correct direction. With regards to the study in sec. 2.1.2, showing that virtual sounds positioned above or below the listener are difficult to localize, vertical sound movements has not been taken into account when developing the audio core of the application.

The sound instructing the user in what direction to move is a pure sine wave synthesized in *Propellerhead Reason 7*, an application for making music (Propellerhead 2014). Sine waves are generally seen as neutral and gentle sounds (Schnupp and King 2010), making them a good fit for applications like Run For Life. Apart from being completely free of modulations on frequency and amplitude, it does not contain overtones. It is in a sense "pure". Since the user should be focusing on the direction of which the sound is appearing from (panoration), it would make sense for the sound itself to be as neutral as possible. Taking advantage of the 3D audio functionality provided by the OpenAL Soft API, the sine wave is positioned around the user's head using the angular offset previously mentioned. For example, a sound with an angular offset of -90° would appear to come from the left, while 130° would appear to come from four o'clock.

As mentioned in the theory chapter, non-individualized HRTFs suffer from back-front confusion. It would be risky to rely single-handedly on the 3D properties of HRTFs in an application depending entirely on the directional properties of its audio. For that reason, some other factors were introduced apart from the spatial panoration to make it easier to navigate correctly.

Pitch When the user moves, the pitch of the sound changes depending on the angular offset; the bigger the offset, the lower the pitch. A lower pitch essentially

means a lower frequency, assuming that the sound is distinguishable from noise (NE 2014). It makes the sound appear to be darker. On the contrary, the sound will become brighter as the user is facing the generated point.

Rate In sec. 2.1.2 it was mentioned that using sound level as a cue is not recommended. To inform the user about the current distance to the generated point, a short sine wave is repeated throughout the run. The length of the interval between each repetition depends on the distance to the point; the sound will repeat faster the closer to the point the user is. Both the change in pitch and change in rate can be related to recommendations seen in sec. 2.1.2.

Apart from the sound being modulated in previously mentioned ways, the sine wave is sometimes completely exchanged to another sound. If the user is moving in the correct direction, with an offset of maximum 20 degrees, an electronic beat is played to indicate they are heading straight towards the generated point. If the user is running in the wrong direction, however, the sine wave is exchanged to a sound informing the user they are doing so. Fig. 3.1 contains a simple visual representation of how the sound is modulated in 360°, with the top being right ahead.

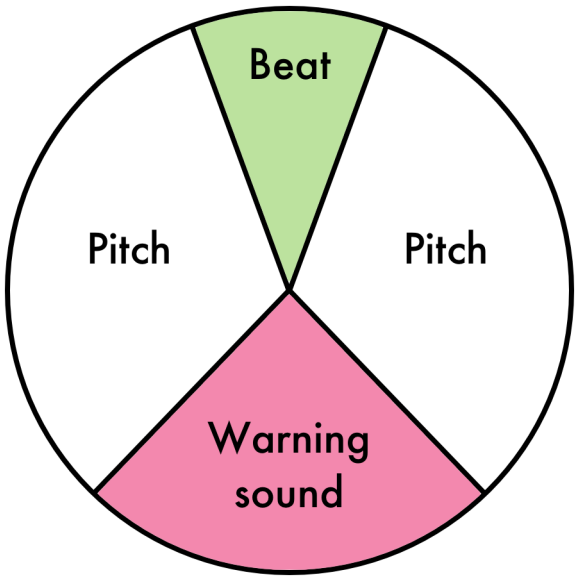


Figure 3.1.: Picture describing how the sound is modulated based on the angular offset.

3.2. Sensor fusion: Combining sensor values to calculate user direction

Sensor fusion is the concept of combining multiple sensors in order to increase the accuracy of a measurement. The method proposed below will utilize the accelerometer, gyroscope and GPS bearing in an attempt to obtain faster and more accurate orientation values than what would be possible by the GPS bearing values alone.

As covered in sec. 2.2.4, the gyroscope values measures rotation in the local coordinate system of the device, the gyroscope readings has to be converted into the global coordinate system. Because the accelerometer readings contain the acceleration toward the ground, these can be used to determine the orientation of the axis orthogonal to the ground in the device coordinate system.

Given that the vector $\mathbf{g} = \begin{Bmatrix} g_x \\ g_y \\ g_z \end{Bmatrix}$ is the raw gyroscope readings from the device (as depicted in Fig. 2.6) and $\mathbf{a} = \begin{Bmatrix} a_x \\ a_y \\ a_z \end{Bmatrix}$ is the raw accelerometer readings (as depicted in Fig. 2.5), the scalar product of the two vectors, $\mathbf{g} \times \mathbf{a}$, gives the rotational speed, here referred to as ω, around the axis orthogonal to the ground (Rocketmagnet - Electrical Engineering Stack Exchange 2012). Rotations around the other axes are not required, since the altitude of sound sources will not be considered.

$$\mathbf{g} \times \mathbf{a} = (g_x * a_x) + (g_y * a_y) + (g_z * a_z)$$

However, in order to make the rotation correct around all axes, trials show that two of the addends requires negation.

$$\omega = -(g_x * a_x) + (g_y * a_y) - (g_z * a_z)$$

Each sensor value update event also has a time stamp, t, associated with it. If the n:th event has the time stamp t_n the difference in time, $\triangle t_n$, is equal to $t_n - t_{n-1}$. Subsequently multiplying ω [radians/time] with $\triangle t$ [time] will give the radians rotated since the last event, $\triangle v$.

$$\triangle v_n = \omega_n * \triangle t_n$$

In order to calculate the current direction (at the time of the n:th element) one simply sum up all previous rotations:

$$v_n = \sum_{i=0}^{n} \omega_i (t_i - t_{i-1})$$

However, this method only calculates a current angle relative to a start angle. In other words, the user would clearly perceive a direction that the sound is coming from, the sound levels updating correctly when the user is turning around, but the perceived direction is not the same as the one the sound source is actually located in.

This can however be solved by involving the GPS bearing. As previously stated, the GPS bearing updates only when the user is in motion. When a new GPS bearing β_u is acquired at time u, the user orientation o_u is set to be equal to β_u. Until the next GPS bearing update however, the user orientation is affected by ω_n:

$$o_n = \begin{cases} \beta_n & if\ new\ GPS\ bearing \\ o_{n-1} + \omega_i (t_i - t_{i-1}) & else \end{cases}$$

This will cause the user direction to automatically adjust itself. Trials show that it works quite well while moving slowly, but since higher speeds creates more frequent adjustments and more confusion for the user the gyro-GPS fusion described above is only implemented as optional in the final application.

3.3. Generation of random locations

As presented in sec. 2.3, the sound sources (checkpoints) are generated out of a predefined algorithm. To accomplish this, the application uses the Google Maps geocoding API and its functions to find directions, sec. 2.3.2. In this section it is explained how the algorithm has been implemented in this project. All the calculations are made using the geographic coordinate system using latitude and longitude, Fig. 2.4, and has been inspired by atok (2013).

Every time the algorithm is executed, a theoretical route is generated based on a predefined number of checkpoints and a distance, d. As seen in Fig. 3.2, a circular dotted route has been generated with four checkpoints and a distance, d. This is done by setting the starting point of the circle at the position of the user and moving the origin to the middle of the theoretical dotted circle. By dividing a full revolution (2π) by the number of checkpoints, x, the angle, α, between them is calculated as in 3.1.

$$\alpha = \frac{2\pi}{x} \tag{3.1}$$

To determine the next checkpoint, the calculated angle, α, is added to the starting position and create checkpoint 2. The same calculations are made for every checkpoint until all have been created.

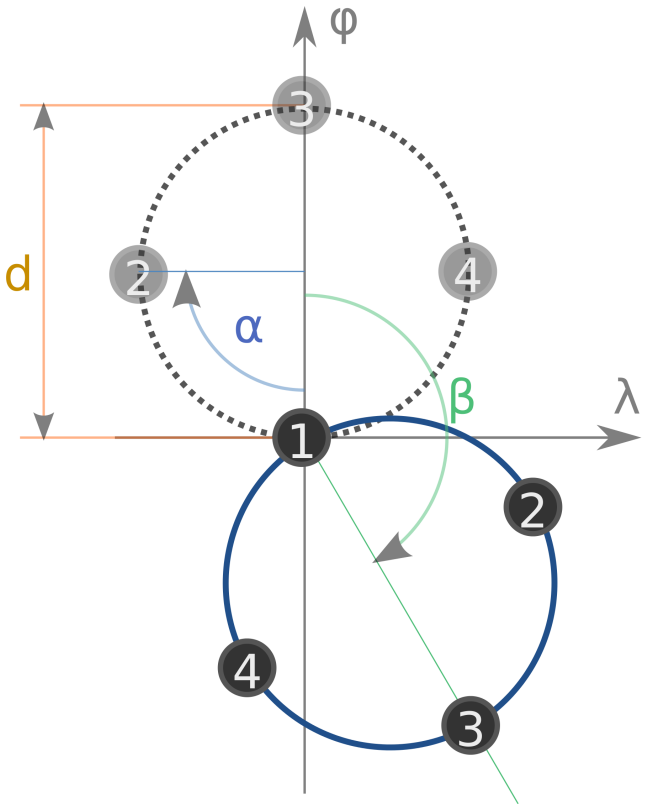


Figure 3.2.: A theoretical route represented by a dotted circle is rotated around the location of the user by an angle β, forming a random route.

To make the routes different from each other every time the algorithm runs, the theoretical route is rotated around the user's location by a random angle, β. The random angle is calculated by multiplying a full revolution (2π) by a random number between 0 and 1. The rotation is performed by multiplying the difference in longitude, λ, and latitude, φ, of every checkpoint compared to the location of the user, with a two-dimensional rotation matrix with the angle, β, as in 3.2 and 3.3.

$$\lambda_{new} = \cos(\beta) * \lambda + sin(\beta) * \varphi \tag{3.2}$$

$$\varphi_{new} = -sin(\beta) * \lambda + cos(\beta) * \varphi \tag{3.3}$$

The new longitude, λ_{new}, and latitude, φ_{new}, is then added to the longitude and latitude of the user's location. This is done for every checkpoint until a random route is generated, illustrated as a continuous lined circle in Fig. 3.2.

Since the locations from the previous algorithm is completely random, it is a big chance that they are not on a road but instead in a forest or a lake. To make a randomly generated location reachable, a geocoding API request of directions from the user's location to the new location is sent to Google (Google Developers 2014). The response from Google contains a list of locations that forms the shortest route to the closest reachable location relative to the requested location. This location is set as a checkpoint and the same procedure is preformed for every location of the generated route. Fig. 3.3 contains an example of a generated random route with the checkpoints located along the blue route.

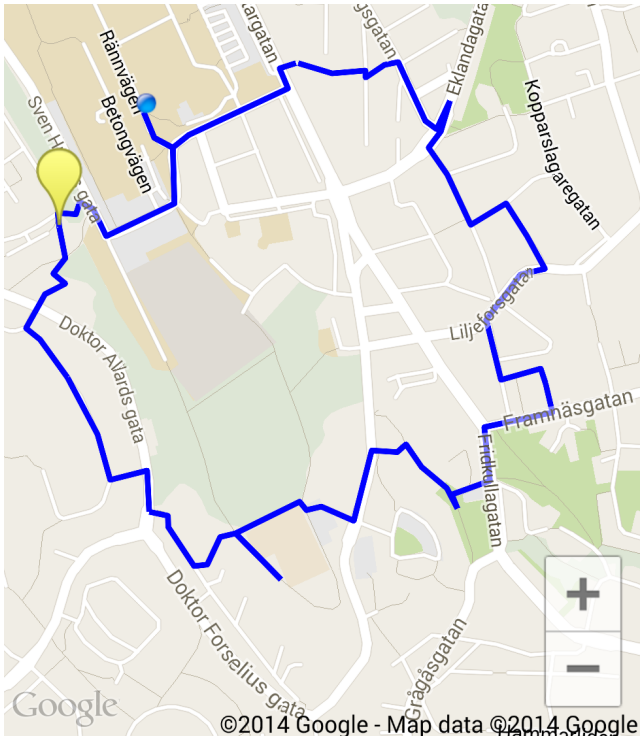


Figure 3.3.: An example of an random route, generated from the random route algorithm. The blue dot is the location of the user and the yellow marker is the location of the first checkpoint.

3.4. Database: Design and Implementation

This subsection describes how the database was designed and thereby implemented in the application to allow the information about the runs to be stored in a way so that it can be retrieved and used in the future. It mentions both the system that manages the functionality of the database as well as a visual representation of the design and the tables that the database consists of.

3.4.1. Database management system

The database that was implemented uses the DBMS SQLite, which is a public domain system that is supported for use in Android development. Since SQLite requires no human support it is well suitable when implementing an embedded database for portable devices (SQLite 2014). An embedded database means that the database is stored locally on the device that it is running on, this implies that the user will only be able to review their running history from their own device and application.

The database is implemented by making a Java class that inherits the SQLite library, then methods have been created to do operations to the database. These methods contain the appropriate SQL-queries that are executed on the database. For example if the application wants to add an object to the database it calls a method from our DBMS that takes the object as an argument, this method executes the appropriate SQL-query to the database to add a row to the appropriate table.

3.4.2. The design of the database

An ER-Diagram (See sec. 2.4) was used to visualize how the database should be implemented into the application. The ER-Diagram helped to keep track on what the different elements of the database consists of as well as how they relate to each other.

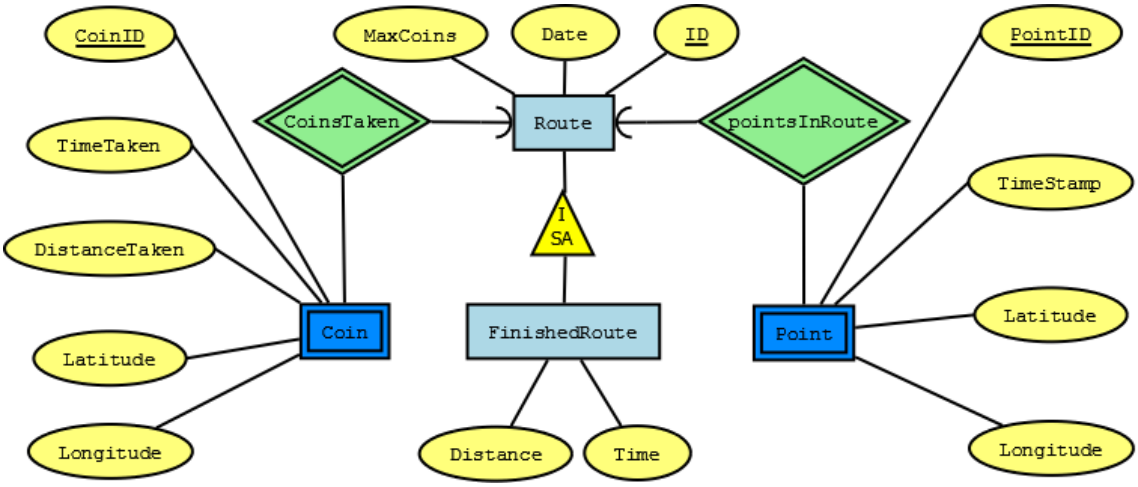


Figure 3.4.: The implemented database represented by an ER-diagram.

The implemented database is centered around the entity “Route” (see Fig. 3.4). A route is basically the user’s running session, containing a unique identifier and the date that the route started. Also the route saves data on the number of coins (being the checkpoints in the game mode Coin Collector which is explained in sec. 3.6.1)

that were generated for the route. The number of coins is set by the user in the settings menu before the running session. The "Point" entity represents a specific location of the users route and is noticeably different by the double border, this means that the entity is a weak entity. A weak entity is dependent on another entity to exist (Garcia-Molina and Ullman 2009, ch. 4.4) in our case a point can not exist without being connected to a route. In practice the weak entity in our database is that the point table has an additional column with the route that is connected to the point. This allows retrieval of all the points that is connected to a specific route. Each point stores data about the location of the user at the time that the point was added, that is latitude and longitude, as well as the timestamp, and a unique ID. The "Coin" entity works in a similar fashion as the point but it stores data of each of the collected coins. When a coin is collected it is stored in the database as a unique id with the distance and time that the user had run when it was collected.

The "FinishedRoute" entity also differs from the other entities because of the "ISA" relationship to the route. This relation can be read as "FinishedRoute is a Route", this means that a finished route inherits each of the attributes that the route has. When a route is finished it is stored in the FinishedRoute table along with the total time and distance of the route.

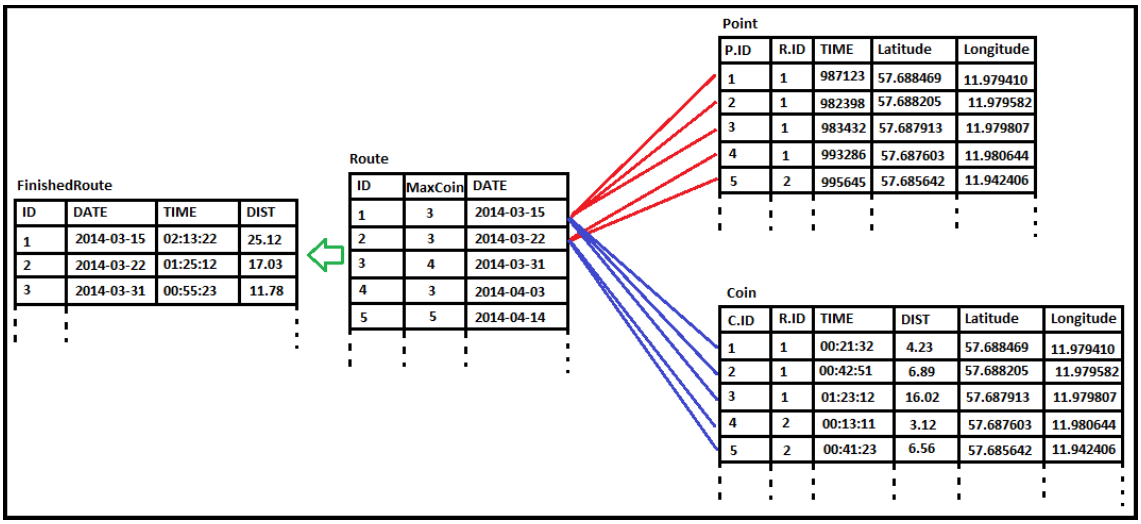


Figure 3.5.: Visualization of the tables in the database. Note that the values in the cells are random values, the purpose of this is to describe the tables of the database as well as the relationships between them.

Fig. 3.5 is a visualization of how the tables in the database could look with some data in them. The column in the points and coins table named "R.ID" (red/blue lines) represents which route that the row is associated with. This is how the supporting relationships that are represented with double borders in Fig. 3.4 was implemented. While all routes, both uncompleted and completed, are stored in the "Route" table, the finished routes are stored in a separate table with additional columns.

3.4.3. Using the data

The purpose of the database is that it should be possible to recreate a previously saved route in the future. From the point table the latitudes and longitudes associated with a route can be extracted and used to draw the user's running path on a map. Distance and time is retrieved from the finished route table and with that information the average speed and pace of the run can be calculated. With the locations from the coin table the coins are drawn on top of the path on the map to visualize the positions where every coin was collected. The information in the coin table is also used to display a visible table of the exact time and distance passed when each coin was retrieved. Since all the the routes are stored with the date that they were created on, a neat list of all finished runs is also easily displayed for the user to browse through in the running history.

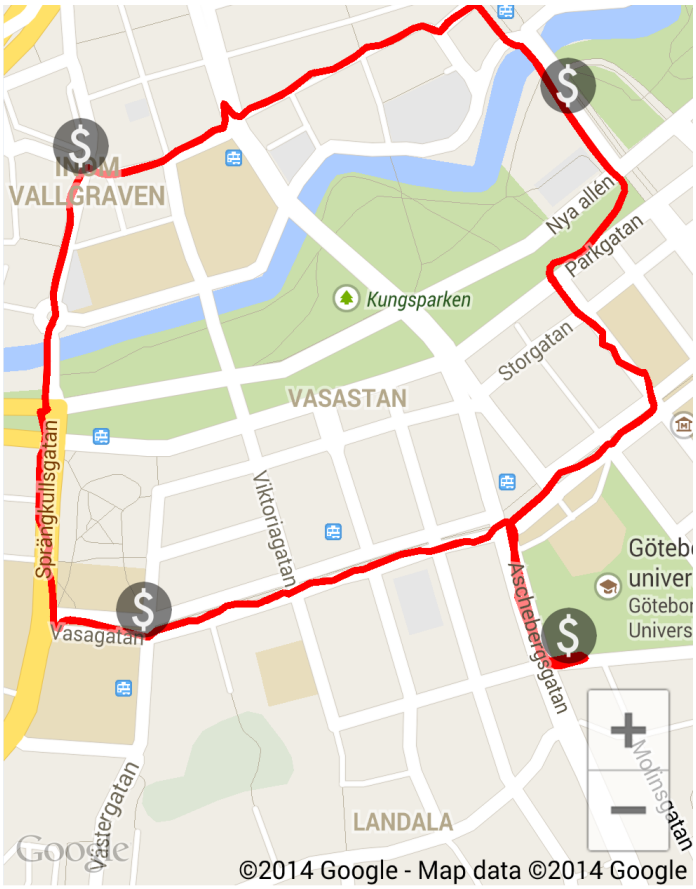


Figure 3.6.: A display of a finished run, where the $ symbols shows the location of every collected checkpoint and the red line is all the GPS data from the users running session.

With this information the user can for example view their average speed from a run finished some weeks back. Another useful feature is if the user discovered a new

great running route, as the one in Fig. 3.6, and wants to run it again. In that case the user can go back to that run through the history and view which path was taken to be able to run it again.

3.5. Graphical design choices and usability

The application is designed with the usability aspect in mind, meaning it is intended to be easy to learn, effective to use and enjoyable from a user perspective (Rogers 2013). To assure this the process of interaction design was applied. This process consists of establishing requirements, designing alternatives and creating prototypes and also conducting user evaluations (Rogers 2013). The main component of this process was to come up with the graphical design. It was however also important to evaluate other parts as the sound perception and if the game was experienced as fun or not.

3.5.1. Establishment of requirements

As a starting point inspiration was taken from the applications *Endomondo* (Endomondo 2014a) and *Zombies, Run!* (Six to Start 2014). These applications are in some sense their opposites. *Endomondo* focuses on the running experience and lets the user quickly start a run. *Zombies, Run!* on the other hand focuses more on external game elements like outrunning zombies, picking up items and upgrading the user's base camp. *Endomondo* puts a lot of focus on detailed statistics which is not the case for *Zombies, Run!*. It would certainly be a challenge to combine the best of these two worlds into one single application.

The requirements that was established was to make the application detailed when viewing statistics of previous runs, fun by including game elements and straight forward when starting a new run.

3.5.2. Design of alternatives and creation of prototypes

To decide the final design solution an iterative process took place. Many alternatives were implemented and tried out before the final prototype was finished. Below the design decisions are motivated.

General The pattern Navigation Drawer was implemented in the Main Activity as shown in Fig. 3.7. The reason the pattern was chosen was because of its quick and easy access to the different Fragments and their screens. In the current version there only exist the three different Fragments: Main, History and Help. This means that tabs at the moment, as mentioned in sec. 2.6.2, would have been the preferable

choice. However, to enable the possibility of extension later on, and to save the space on the screen that would have been covered by tabs, Navigation Drawer was considered the best choice.

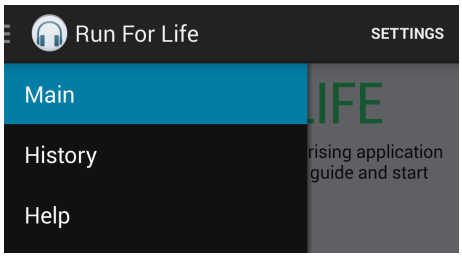


Figure 3.7.: Quick navigation through the Navigation Drawer.

The History and Help Fragment were both designed with the intention of being as simple and intuitive as possible; the History Fragment being a list of clickable previously completed runs and the Help Fragment being an instructional page with sample sounds to obtain a better understanding of the concept. The Main Fragment however has a lot of interesting design components which will be explained below.

Main screen Since the application's main focus is the running part it was decided to, on start up, take the user directly into the screen of the Activity that chooses a running session. This reduces the excise for the user by one click (compared to going via a Dashboard, which is lifted in sec. 5.4) when they wants to start a run, which is the use case believed to be used most frequently.

To easily browse between different game modes the swipeable pattern Filmstrip was implemented as in Fig. 3.8. Since Filmstrip is not a standard pattern in Android it was implemented with *ViewPager* (Android Developers 2014l), which is a layout supporting horizontal swiping. The ViewPager was set up with the adapter, FragmentStatePagerAdapter, handling each page on the Filmstrip (Android Developers 2014f). The pattern was considered appropriate to the application since it provides a default value of a chosen game mode and thereby supports a quick start. In the context of choosing between different game modes Filmstrip also serves a good cause since new users gain a clear understanding of a, to them unknown, game mode by viewing its picture and name supplied by its page on the Filmstrip. If the user needs more information to choose game mode an information icon in the upper right corner of the current filmstrip page indicates that more information will be shown if clicked. Further the affordance of swiping is assured with the dots in the bottom right corner.

Below the Filmstrip there is a bar containing icons and a next button as shown in Fig. 3.9. The tendency of users reading content from top to bottom and from left to right (Tidwell 2011) has been taken in account to make sure that the user gets to this bar after choosing game mode. The icons are symbols notifying the user

Figure 3.8.: Horizontal swiping in the Filmstrip.

whether the GPS and headphones are connected or not. To make sure that the icons are perceived as icons and not buttons they have been re-designed until they looked discrete enough. It was also important to notify the user if disconnection occurred. By putting a cross over the symbol in the icon in case of disconnection the user is meant to clearly notice that something is not right, getting the opportunity to fix it by enabling the GPS and/or inserting the headphones.

Figure 3.9.: The bar on the bottom of the screen showing: a connected GPS icon and a disconnected headphones icon to the left and and a Prominent Done Button to the right.

The tendency of reading from left to right, as mentioned before, is well taken in account in the bar to make sure that a possible disconnection is discovered before the next button is pressed. To let the user know that the next screen is reached by clicking the next button, it is implemented as a Prominent Done Button.

Run screen Since the Run Activity includes information about the run itself, the map and the statistics of each coin taken; the pattern Swipe Views is used. The reason for this is that all this information does not fit in one single screen. By putting it in three different Fragments (Run, Map and Stats) and by letting the Run Activity use the adapter FragmentPagerAdapter to handle the Fragments (Android Developers 2014d), they will become swipeable tabs, as shown in Fig. 3.10.

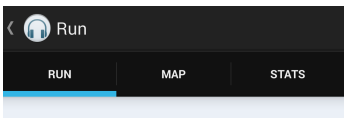


Figure 3.10.: Run, Map and Stats Fragment implemented as swipeable tabs.

The bar from the Main Activity also appears in the Run Activity to make the application consistent and let the user know directly if there is a disconnection of the GPS or headphones. To motivate the user to keep running, a progress bar which increases with every coin taken, is displayed at the bottom of the Run Fragment. The user may also view time, distance, speed and pace in the Run Fragment to get assured that the logging of the run is taking place.

The Prominent Done Button pattern is also applied in the Run Activity, but this time as a play/pause button. When the run is paused a stop button will also appear, to let the user end the run if desired. If the run is ended the Run Activity will turn into a finished state thus removing the progress bar and the information of current speed and pace. The finished state of a Run Activity is the format that each finished run is saved as, and is thereby reachable for each finished run from the History Fragment.

The Map and Stats Fragments are visible during a run but are most important when a run is finished. In the Map Fragment the user can see a map with the route on which coins will appear as well as the next coin to run to. This is preferably not viewed during a run, but was decided to be kept in case of the user getting completely lost and needing exact directions. When viewing a finished run however this Fragment will display the track that was taken as well as the places on which coins were taken. This is intended to give the user feedback and note if a better route could have been taken.

The requirement of detailed statistics was highly considered while designing the Stats Fragment. By having a table with rows for each coin the user will have a way to receive feedback as each coin is retrieved. The information displayed is the chronological number of the coin and the time, distance and pace achieved while running to it. While this is great feedback during the run it is perhaps most interesting when the run is finished and the data of each coin can be compared to coins taken on previously finished runs.

3.5.3. User evaluation

To make sure that the application was usable, understood by users and filled the requirement of being fun, a user evaluation was done to receive feedback on what to improve. The tasks and questionnaires are attached in Appendix A. Below follows a summary of the received feedback from the data of the interviews and observations that were done in connection with the user tests.

Concerning the graphical design, some users tried clicking on the coin collector picture to start a run. When nothing happened they were confused. This was fixed by making the next button more prominent and thereby assuring the user that it needed to be pressed in order to go further. Also it was decided to show an information text when the picture of a game mode was clicked on. This information explained the mode but also let the user know that something happened upon the click, thus eliminating the misconception that the application was frozen while loading.

It also turned out that some of the users tried to press the GPS and headphones icon, since they thought they were buttons. This led to the re-design of the symbols that was explained in sec. 3.5.2.

Another problem that became evident during the user testing was that there needed to be a more clear warning if the run was started without the GPS enabled. In one of the user testings the phone got its location via the mobile network, which is very imprecise compared to GPS. However the application still tried to make sense of the location and thus provided the user with, seemingly, random direction noises. This problem was fixed by not letting a user start a run if GPS connection was missing.

Regarding the instructional sounds of a run most of the users understood in what direction to run. Some found it hard in the beginning but understood it after getting it explained and tried out. This insight led us to implement the the Tutorial and a Help Fragment, which was hinted about at the very first start up of the application.

Most users found the beeping sound annoying after running for a while and would rather like to hear music. This feedback was used to replace the beeping sound when running in the correct direction into an affirming “techno music” sound.

In two of the tests a coin ended up in a private property and inside the Liseberg theme park. These kinds of errors are hard to adjust and will be disregarded.

The general overall opinion from the users was that they liked the concept and felt entertained and motivated by it.

3.6. Game modes

One of the purposes of the application is to prevent sedentary amongst people. It would therefore make sense to reach out to an as large audience as possible. To achieve this, Run For Life is designed in a modular way, making it easy to add new game modes. In the final application, however, there is only one game mode and one tutorial implemented. In sec. 5.5, some possible game mode concepts are discussed for future development.

3.6.1. Coin collector mode

Coin Collector lets the user locate coins (checkpoints) placed in a circular route, generated by the algorithm described in sec. 3.3, using spatial audio. During the whole run, a short navigation sound is repeated periodically. As the user moves, the navigation sound is being panorated and modulated (or completely exchanged) as a result of the angular offset mentioned in chapter 3. The interval between each repetition is gradually shortened as the user approaches hundred meters. In connection with the user passing it, the repetition interval is set back to default and audio feedback telling the user the distance left to the coin is played.

Whenever a user reaches a coin, a voice informs them that the coin has been reached. Unless it is the final coin, the same voice will let the user know that another one has been generated. Since the final coin always generates at the starting point (see sec. 3.3), a finished run should result in the user being back to where they started.

3.6.2. Tutorial mode

The tutorial mode was implemented to allow the user to try the application's functionality and become familiar with the spatial sound without having to run. It takes advantage of the gyroscope sensor, explained in sec. 2.2.4 and converted to a global reference frame as explained in sec. 3.2. By tapping on a button and rotating, following the navigation sound, the user's task is to find a hidden point on the screen by controlling an arrow moving freely on the screen. If the user hears the electronic beat while tapping the button, it means they are heading straight towards the goal.

4. Result: Application Simulating Sound Sources while running

This bachelor project has resulted in an Android application using spatial sound to guide the user. It is currently available on Google Play, named *Run For Life* (https://play.google.com/store/apps/details?id=se.chalmers.group42.runforlife). The following chapter presents its result and describes the final application and its screens.

4.1. Home screen

The Home screen is the first thing the user sees when launching the application. If it is the first time using the application, however, a dialog box will show up on top. The dialog box provides a link leading to the Help screen (also accessible from the Navigation Drawer). Fig. 4.1 displays the Main screen and its different parts:

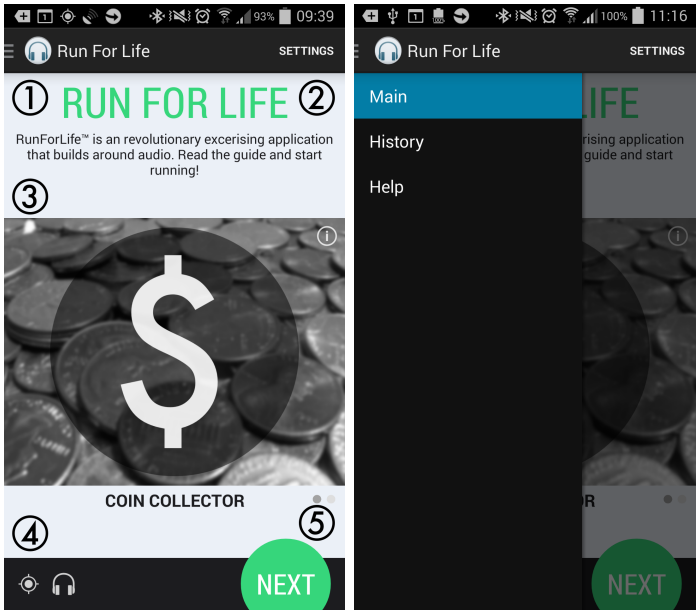


Figure 4.1.: The Main screen and its Navigation Drawer.

1 The three horizontal lines indicate that a hidden menu can be accessed by swiping from the left or by pressing the icon, as mentioned in sec. 2.6.2. From here, the user can choose to go back to the Main screen, view statistics on past runs in History or learn about how the application works in Help.

2 On the Settings screen, the user can customize their experience.

3 Using the selector in the middle, the user can swipe between the two available game modes: Coin Collector and Tutorial mode. If the user touches anywhere on the image, information regarding the selected mode will replace it.

4 The first icon tells the user whether the device has a valid GPS connection or not, and the second whether headphones are plugged in to the device or not. In the picture, none of the icons are crossed out, meaning that the GPS connection is valid and the headphones are plugged in.

5 Pressing the next button launches the mode currently selected in the middle of the screen.

4.2. Coin Collector screen

In Coin Collector, the user's task is to navigate to certain checkpoints (coins) using the sound coming from the headphones. The game is finished once all coins are found (see sec. 3.6.1 for a detailed description). The screen is made up of three different tabs, each having its own purpose; Run, Map and Stats.

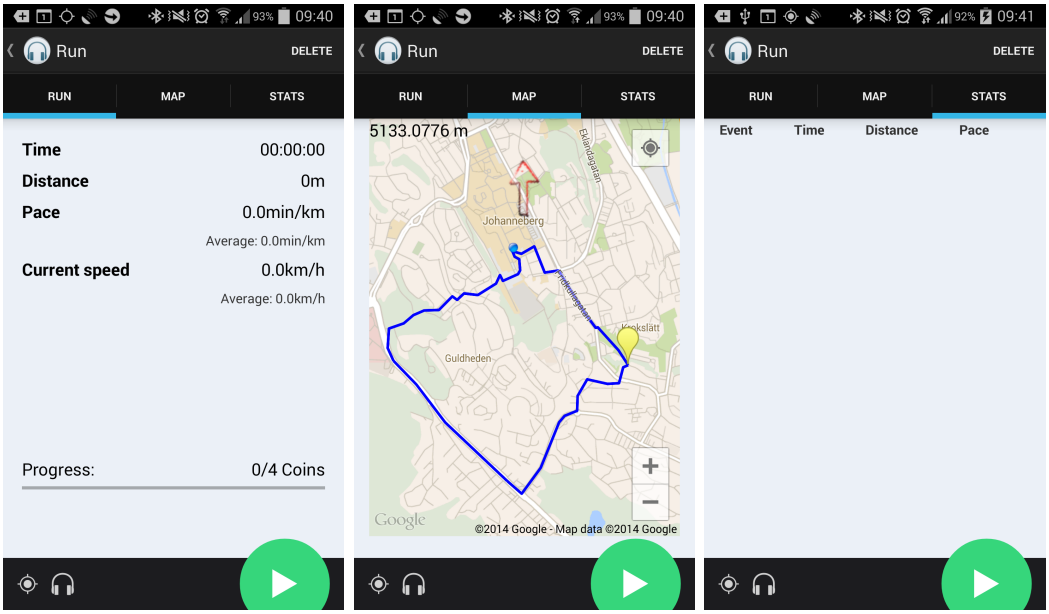


Figure 4.2.: An ongoing run in Coin Collector mode.

The first image in Fig. 4.2 displays the Run tab. It is the tab visible when launching the mode and it displays information about the distance, time, speed and pace. The second tab, Map, tells the user about their current position and route. In the Stats tab, the user can see how many coins they have collected and at what distance and time.

To start the run the user has to press the play button in the lower right corner. This will result in the navigation sound starting to play. However, in order for the navigation sound to be heard from the correct direction, the GPS has to calculate the direction of which the user is heading. This process usually requires that the user moves a few meters in order for the GPS to track the position of the user.

The run can be paused at any time by pressing the pause button located in the bottom right corner, replacing the play button while running. Pausing the run will stop the navigation sound from repeating and the GPS from tracking the user's location. To resume from a paused state, the user simply presses the same button one more time. If the user wishes to quit the run completely, they can do so by pressing the stop button located at the bottom.

When a run is finished, the screen looks similar to what it does while running; the Run tab displays basic run statistics and the Map and Stats tabs display information of where the coins have been found and at what time. By pressing the OK button, the user is taken back to the home screen described in the beginning of the chapter. For a visual representation, see Fig. 4.3.

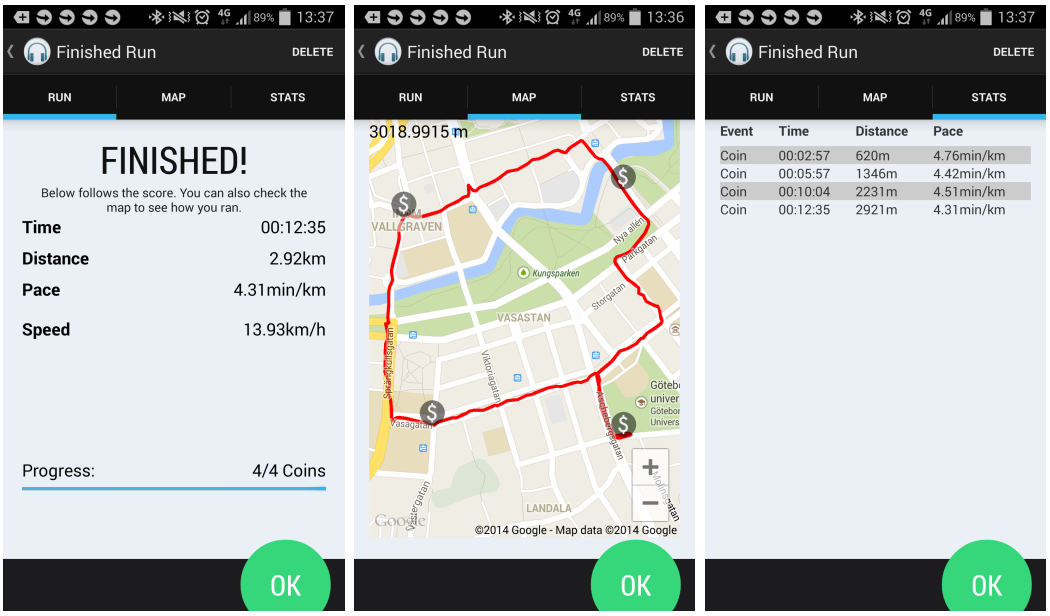


Figure 4.3.: A finished run in Coin Collector mode.

4.3. Tutorial screen

The Tutorial mode enables the user to try out the spatial sound of the application without running. When the mode is launched, a point is generated on the screen and the sound starts to play. A detailed description of how the mode works can be found in sec. 3.6.2. To move the arrow, the user simply taps the run button in the upper right corner of Fig. 4.4. "Cheat" is enabled by toggling the checkbox at the bottom of the same figure, making the generated point visible. Cheating is not enabled by default.

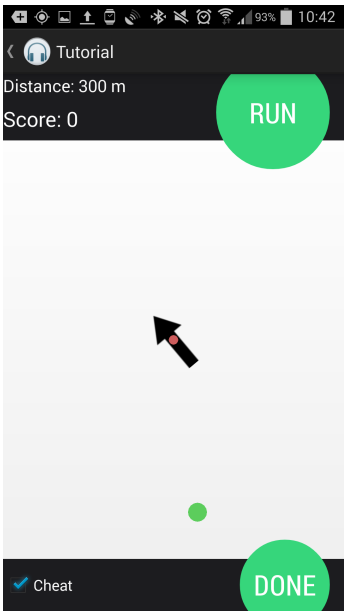


Figure 4.4.: Tutorial mode with cheat turned on.

4.4. Settings screen

On the Settings screen, shown in Fig. 4.5, the user can customize their experience with the application. Currently, the settings only apply to the Coin Collector mode, as the Tutorial mode is supposed to be straightforward and easy to play without having to worry about specific settings. To change the diameter of the circular route generated in Coin Collector or the number of points being generated, the user simply touches either of the sections. The user is then prompted to enter the desired value. The toggle button at the bottom enables the user to try out the sensor fusion adaption explained in sec. 3.2.

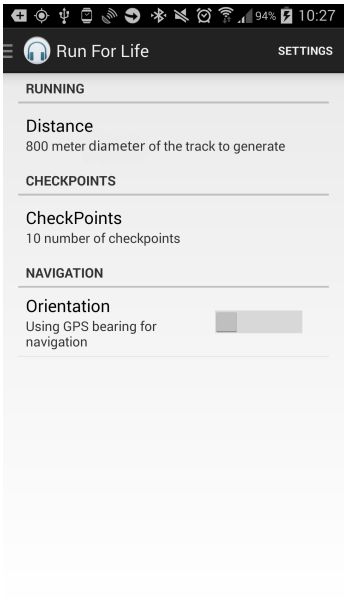


Figure 4.5.: Settings screen.

5. Discussion

This section reflects upon the findings of the project: how to use sensor data to calculate a user's direction, how to generate a route of accessible and random locations, how to correctly design the application and how to use binaural audio in a way that creates the perception of a sound coming from a specific direction.

Initially, the completion of the problem assignment will be discussed. The following goals where set up for the project:

Translate location data obtained with sensors into navigation instructions with sound. This was accomplished by using the GPS sensor and feeding the bearing and location data into the HTRF-framework described in sec. 3.1.

Register running activity and present its statistics. As described in sec. 3.4, the user is able to view past runs as statistics are saved in an SQLite database.

Motivate users to exercise by implementing game elements. Game elements were implemented as the game mode *Coin Collector* described in sec. 3.6.1. Future possible game modes will also be covered in sec. 5.5. Coin Collector was received positively by testers as described in sec. 3.5.3. If this project can motivate more people to exercise it might contribute to a healthier world. We additionally hope that the project will inspire additional applications using the same techniques to further motivate a healthier lifestyle.

Create a graphical user interface that is straightforward and easy to understand. Effort has been put into creating a user interface with good usability, as is described in sec. 3.5 and which will be later discussed in sec. 5.4.

5.1. Finding the direction of the user

Run For Life uses GPS technology to determine the user's direction. A disadvantage of this is that the user needs to be in motion in order for the application to provide an accurate angle relative to the sound source. The reason why a user might stop while running is mostly because of uncertainty in the perceived direction of the

sound. However, when the user is stationary the GPS will not obtain any significant location updates and hence cannot calculate any direction of travel. If a user does not know this, it might prove very hard to use the application. Furthermore, the GPS cannot detect the user's head movements; something which lowers the sound perception. An example of this would be if the user runs in a certain direction and turns the head 90 degrees. Then this will not affect how the sound is played back.

The great advantage with using the GPS bearing was however that it allows users to wear their phones in any way they prefer. This was something that this project required, as it seemed more realistic that the users would run with their phones in a pocket or in a sports bracelet, than that they would attach the phone to their head or carry it in their hand like a compass. Hence, while a firmly attached phone would have made the user orientation a lot more accurate, it would be impractical for this project.

Comparing with the master thesis *Application of Dynamic Binaural Signals in Acoustic Games* (Lawitzki 2012), the approach of calculating the user orientation has as such been a lot more complicated in this project. Lawitzki (2012) had the smartphone firmly attached to the head of the user, giving him the possibility to measure orientation with great accuracy. Since the phone was then at a fixed position relative to the ears of the user, the perception of the sound was a lot better.

The alternative of combining the gyroscope and the accelerometer data described in sec. 3.2 was an attempt to a compromise between the GPS-bearing alternative and the orientation sensor fusion described by Lawitzki. It worked fairly good while moving slowly, but at higher speeds the sound starts to jump around too much, making the user confused. Probably some better algorithm for choosing when to use the gyroscope readings and when to adjust the direction with the GPS-bearing, perhaps taking speed into consideration, would improve the user experience.

Another benefit from using sensor fusion is that it captures the sometimes bad accuracy from the GPS. The GPS might temporary lose its connection and by using the values from other sensors, the loss can be compensated and the application can be used as normal anyway.

5.2. Using binaural sound to guide the user

In sec. 2.1.1 it was mentioned that the pinna is shaped differently on every person. Because of this, non-individualized HRTFs are far from suitable for everyone. In Run For Life, the sound has a great impact on the user experience, placing high demands on its ability to navigate the user. Unfortunately, the audio core currently suffers from being deceptive - it is difficult to hear from which angle a sound is appearing from. In order to make it easier for the user to perceive the offset angle, we added more factors to the navigation sound, such as pitch and rate. While it improved the user experience to some extent, the initial problem still remains.

Whether the uncertainty in angle accuracy is a result of the incorrectness of the location data or bad HRTFs provided by OpenAL Soft is difficult to tell, looking at the Coin Collector mode. It should be mentioned that the binaural audio seems to perform slightly better when relying on the smartphone's gyroscope in Tutorial mode. Judging by our own observations made during the development, the GPS seems to be the application's biggest fault, but its navigational properties of the audio are certainly not flawless. Our conclusion is that 3D audio through generic HRTFs can work pretty well in applications where the user's position and movements are not vital for the whole experience. At this point, however, binaural audio is not suitable for full-scale navigation applications like Run For Life.

5.3. Implementation choices of the random route generation algorithm

As stated in sec. 3.3, the random route algorithm in this project has used Google Maps and its geocoding API. The decision to choose this method instead of the colour classification method, was based on the available tools and knowledge at the beginning of the project. The method with the colour classification was the first method that was taken into consideration. Different approaches was evaluated in order to make this method viable. The basic and most important aspect to this method was to have an accurate map with a corresponding coordinate system attached to it. Different maps was considered, for example Google Maps[1] and OpenStreetMap[2]. The hard part was to extract the colour from the interesting location. This could be done by taking a print screen of the map and analyze the image with an algorithm that determines the colour of the location. There might also be functions available from the map-providers by sending a request to them and receiving a response containing the colour of the location. This was not investigated in detail since the idea of using Google Maps and its functions to find directions solved the problem for us. The method using Google Maps geocoding API was at that time more preferable since a lot of its functions was supported within Android and the APIs from Google.

The two methods both have their pros and cons. The bad thing with the method using the geocoding API is that it must be quite a lot of roads and trails nearby for the request to find any good directions. If there is only one straight road nearby, the sound sources are going to be mapped to the road and the distance between them will vary a lot. A conclusion of this method is that it is working good in a city or a village where there are a lot of roads and walkways. The colour classification model on the other hand does not depend on the location as much and might be better to consider while on the countryside.

[1] https://www.google.com/maps/
[2] http://www.openstreetmap.org/

5.4. Design choices

The responses from the user evaluation was generally positive concerning the usability. Most of the users had no problem in understanding what to press in order to achieve their tasks, which we interpret as a success in making the user interface straight forward and easy to understand. The motivations behind the design choices are explained fully in sec. 3.5.2. The motivations behind the patterns that were considered but rejected are however explained below.

During the design process the inclusion of a *Splash Screen* was considered. When starting an application a Splash Screen may sometimes be motivated since it gives the user feedback that the application is loading. This is often applied in games where the loading takes a long time. However, the fact that the loading of the splash screen itself requires resources suggests that it is better avoided (Lehtimäki 2013). In Run for Life a splash screen would probably take almost as long time to load as the application's start screen, which would make it inefficient. A solution was making the start screen light weight and low in its demanding of resources (Lehtimäki 2013). Thereby the Run For Life-start screen was chosen not to include the loading of the resource demanding algorithms needed to calculate a route.

At first the *Dashboard* pattern (Lehtimäki 2013) was considered for the start screen. The reason for this is that a Dashboard clearly presents an application's different parts with big icons leading to them. The pattern was earlier officially recommended by Google and have thereby become well known to users. The pattern could have been used to let the user choose if they wanted to start a run, view the history or something else. However, as mentioned in sec. 3.5.2 it was desired to keep the excise as low as possible when starting a run, which is why the Dashboard was omitted.

5.5. Future work: efficiency, modularity and prospects

As described in sec. 1.4, the application was not designed with any emphasis on efficiency and the code can probably be improved a lot in that manner. The rendering in the tutorial mode is sometimes slow; some saved running logs take a lot of time to open and the application drains quite a lot of power from the phone because of the sensor usage. The research on code optimization is extensive and the application could in all likelihood be improved a lot if given the time and effort.

Instead of efficiency however, the application has from the beginning been constructed in order to be as modular and easy to extend as possible. This will allow for implementation of future game modes that will further utilize the combination of binaural sound and smartphone sensors. One of the main issues with Coin Collector, currently the only game mode finished, is that the sometimes extensive distance to the sound source makes volume variation relative to distance impractical; if it is too far away it is impossible to hear. In Coin Collector this is solved by using the rate of

the sound to indicate distance (as explained in sec. 3.1.2), but that limits the types of sounds that can be used. With closer sound sources, variation of the sound volume might be more natural. Example of game modes enabling closer sound sources follows below.

Free running Instead of running along a pre-generated route of coins (sound sources) the user starts running anywhere they wants. The coins are then generated close to the user position at random intervals during the run, giving the impression of running in an area scattered with coins. The user will only hear a coin when it is close enough; but since there is no need to take a specific coin in any specific order this is no issue.

Escape This game mode would be based on escaping one or several "monsters" (sound sources). The monsters would hunt the user and the user needs to run away from them fast enough to not be caught. Apart from distance-relative volume, this will also require the development of an algorithm for moving sound sources appropriately. The easiest and most straight-forward way of doing that is to simply move the "monsters" forward the user at a certain velocity, disregarding buildings and roads. With time however, more advanced path-finding algorithms could be researched and implemented.

Endurance hunt Similar to the escape mode, the sound source is the prey instead of the user. When the user is close enough to the prey (sound source), it notices the user and starts to move in the opposite direction. When the prey has moved far enough from the user (almost out of audible distance, depending on an exhaustion factor) the prey will stop to take a rest. The run will continue until the user catches the prey (either by running fast enough or by exhausting it) or alternatively, until the user gives up.

One downside with the above game modes compared to Coin Collector is that the user will not necessarily end up at the start location when finished.

The embedded database that was implemented was sufficient for the purpose of the project. However, a better option would be to have a server-based database system with a web interface. It could even be merged with Facebook, or other social media, to allow the user to share their progress and read about their friend's achievements. This would make it easier for the users to access their data from different devices. Furthermore, the progress will not be lost if the user replaces their current smartphone. Finally a server-based system would allow users to share their sound sources with friends and collaborate on finding or avoiding them.

A way to improve Coin Collector would be to let the user select the beat illustrated in Fig. 3.1 and described in sec. 3.1.2 so that the beat can be any melody the user would like to listen to. This could be solved either by playing a chosen sound file within Run For Life itself or by sending an implicit intent using another media player application on the device. The possibility to choose music would probably prevent the monotony that might occur when navigating to the same sound.

While the previously discussed head-mounted smartphone would be impractical for the user, a device similar to Google Glass would be an improvement. Since Google Glass is worn like ordinary glasses and has built-in orientation sensors the measurements would equal those from the head-mounted smartphone investigated by Lawitzki (Lawitzki 2012). As the Google Glass will have a fixed position relative to the ears the user will always have turned their head whenever the sensors have provided a new value. Furthermore, the user could then also be provided with a visual representation of the sound source (for example a coin), hence augmenting the experience further.

6. Conclusion

This project concluded in a fully functional Android exercise application combining GPS technology and binaural audio, creating the perception of the sound arriving from a specific direction in order to guide the user. The final application suffers from the location data not being accurate enough, in the sense that it does not track the user's movements in a desirable way. That, in combination with the audio core not being precise enough, occasionally makes it difficult for the user to navigate. However, data gathered from the user evaluation indicates a generally positive attitude towards the application. Ultimately, the application, as well as the project, can be considered successful in that it meets the requirements established in the problem assignment.

Bibliography

Algazi, V. and Duda, R. (2011) Headphone-Based Spatial Sound. *Signal Processing Magazine, IEEE*.

Android Developers (2014a) Action Bar. *Website*. `http://developer.android.com/design/patterns/actionbar.html`. (Accessed: 2014-04-04).

Android Developers (2014b) ADT Plugin. `http://developer.android.com/tools/sdk/eclipse-adt.html`. (Accessed: 2014-05-17).

Android Developers (2014c) Android NDK. `https://developer.android.com/tools/sdk/ndk/index.html`. (Accessed: 2014-03-14).

Android Developers (2014d) FragmentPagerAdapter. *Website*. `http://developer.android.com/reference/android/support/v4/app/FragmentPagerAdapter.html`. (Accessed: 2014-05-16).

Android Developers (2014e) Fragments. *Website*. `http://developer.android.com/guide/components/fragments.html`. (Accessed: 2014-05-17).

Android Developers (2014f) FragmentStatePagerAdapter. *Website*. `http://developer.android.com/reference/android/support/v4/app/FragmentStatePagerAdapter.html`. (Accessed: 2014-05-16).

Android Developers (2014g) Intents and Intent Filters. *Website*. `http://developer.android.com/guide/components/intents-filters.html`. (Accessed: 2014-05-17).

Android Developers (2014h) Location - getBearing method. *Website*. `http://developer.android.com/reference/android/location/Location.html`. (Accessed: 2014-04-29).

Android Developers (2014i) Motion Sensors. *Website*. `http://developer.android.com/guide/topics/sensors/sensors_motion.htm`. (Accessed: 2014-04-08).

Android Developers (2014j) Navigation Drawer. *Website*. `https://developer.android.com/design/patterns/navigation-drawer.html`. (Accessed: 2014-04-04).

Android Developers (2014k) Platform Versions. `https://developer.android.com/about/dashboards/index.html?utm_source=ausdroid.net`. (Accessed: 2014-05-17).

Android Developers (2014l) ViewPager. *Website*. http://developer.android.com/reference/android/support/v4/view/ViewPager.html. (Accessed: 2014-05-16).

atok (2013) How to generate random locations nearby my location? http://gis.stackexchange.com/questions/25877/how-to-generate-random-locations-nearby-my-location. (Accessed: 2014-04-08).

Bajaj, R. et al. (2002) GPS: Location-Tracking Technology. *Computer*, vol. 35, no. 4, pp. 92–94. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=993780.

Barreto, Armando; Faller, K. A. M. (2009) 3D Sound for Human-Computer Interaction: Regions with Different Limitations in Elevation Localization. In *Proceedings of the 11th international ACM SIGACCESS conference on computers and accessibility*

Brieger, S. and Göthner, F. (2011) *Spelares inställning till HRTF-teknik i FPS-spel*. Kungliga Tekniska Högskolan.

OpenAL Soft. (2014) OpenAL Soft. http://kcat.strangesoft.net/openal.html. (Accessed: 2014-03-10).

Endomondo (2014a) About - Endomondo Blog. http://blog.endomondo.com/about/. (Accessed: 2014-04-08).

Endomondo (2014b) Facts - Endomondo Blog. http://blog.endomondo.com/facts/. (Accessed: 2014-04-04).

Garcia-Molina, H. et al. (2009) *Database Systems - The Complete Book (2nd Edition)*. Upper Saddle River, New Jersey: Prentice Hall.

GitHub (2014) GitHub - Build software better, together. https://github.com/. (Accessed: 2014-05-17).

GNU (2014) GNU Lesser General Public License. https://www.gnu.org/licenses/lgpl.html. (Accessed: 2014-03-10).

Google Developers (2014) The Google Geocoding API. https://developers.google.com/maps/documentation/geocoding/. (Accessed: 2014-05-13).

Grubesa, T. et al. (2004) Simulation of virtual 3D auditory space with HRTF. In *Electronics in Marine, 2004. Proceedings Elmar 2004. 46th International Symposium*; June 2004 , pp. 271–277.

Khronos (2014) OpenSL ES Specification. http://www.khronos.org/registry/sles/specs/OpenSL_ES_Specification_1.1.pdf. (Accessed: 2014-03-09).

Kleiner, M. (2011) *Acoustics and Audio Technology*. 3rd edition.

Kramer, G. (1994) *Auditory Display: Sonification, Audification, And Auditory Interfaces*.

Lawitzki, P. (2012) *Application of Dynamic Binaural Signals in Acoustic Games*. [Electronic] Nobelstraße 10 D-70173 Stuttgart: Media University Stuttgart (Hochschule der Medien). (Master's thesis). http://www.thousand-thoughts.com/wp-content/uploads/MasterThesis_Lawitzki_2012.pdf.

Lehtimäki, J. (2013) *Smashing Android UI: Responsive User Interfaces and Design for Android Phones and Tablets*.

M2 Communications (2012) Research and Markets: Global Positioning System Market 2011-2015 - Lack of Accuracy Affecting the Growth of GPS Market. *M2 Presswire*.

Mercator, P. (2010) Latitude and longitude graticule on a sphere. http://en.wikipedia.org/wiki/File:Latitude_and_longitude_graticule_on_a_sphere.svg. (Accessed: 2014-05-16).

National Coordination Office for Space-Based Positioning, Navigation, and Timing (2014) The Global Positioning System. *Website*. http://www.gps.gov/systems/gps/. (Accessed: 2014-04-25).

NE (2014) tonhöjd. http://www.ne.se/lang/tonhöjd. (Accessed: 2014-04-16).

Ordnance Survey - the national mapping agency of Great Britain (2002) *A guide to coordinate systems in Great Britain*. [Electronic] http://www.bnhs.co.uk/focuson/grabagridref/html/OSGB.pdf.

Palomäki, K. J. et al. (2011) Spatial processing in human auditory cortex: The effects of 3D, ITD, and ILD stimulation techniques. *Cognitive Brain Research*.

Pielot, M. (2010) OpenAL4Android. http://pielot.org/2011/11/10/openal4android-2/. (Accessed: 2014-03-13).

Propellerhead (2014) Reason. http://www.propellerheads.se/products/reason/. (Accessed: 2014-05-02).

Ratabouil, S. (2012) *Android NDK Beginner's Guide*.

Rocketmagnet - Electrical Engineering Stack Exchange (2012) Rotating a gyroscope's output values into an earth-relative reference frame. *Question at Electrical Engineering Stack Exchange*. http://electronics.stackexchange.com/a/29461. (Accessed: 2014-04-25).

Rogers, Y. (2013) *Interaction design: beyond human-computer interaction*. 3rd edition. Chichester, West Sussex, England: John Wiley & Sons, Inc.

Schnupp, J. et al. (2010) *Auditory Neuroscience: Making Sense of Sound*.

Sears, A. and Jacko, J. A. (2007) *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*. 2nd edition.

Six to Start (2014) Zombies, Run! https://www.zombiesrungame.com/press/. (Accessed: 2014-02-12).

So, R. et al. (2010) Toward orthogonal non-individualised head-related transfer functions for forward and backward directional sound: cluster analysis and an experimental study. *Ergonomics*, vol. 53, no. 6.

SOFA (2013) General information on SOFA. `http://www.sofaconventions.org/mediawiki/index.php/General_information_on_SOFA`. (Accessed: 2014-04-11).

SQLite (2014) Appropriate Uses For SQLite. `https://www.sqlite.org/whentouse.html`. (Accessed: 2014-02-20).

Tidwell, J. (2011) *Designing Interfaces*. 2nd edition. Sebastopol, California, USA: O'Reilly Media Inc.

A. Appendix: User evaluation

The test was carried out by letting the user do a number of tasks. The data was then gathered by asking questions. Before completing the tasks involved in the testing, however, the functionality was explained with the following text.

A.1. Explanation

The sound will appear to come from a certain direction while wearing the headphones. When you are running straight to the sound source the sound is playing with equal volume in both ears and with a high pitch. If the sound source is to your right it will be higher volume in the right headphone and lower in the left and with a medium pitch. If the sound source is behind you it is equal volume and a lower (dark) pitch.

To easier tell if you are heading in the right direction the playback rate of the sound increases the closer you get to the coin within each 100 meter interval from it. For every 100 meter interval you pass you are provided with feedback of how far it is to the sound source, for example “500 meters”.

A.2. Tasks

1. Start the application
2. Go to settings and choose:
    a) Distance: 700
    b) Checkpoints: 4
3. Start coin collector
    a) Choose Coin collector
    b) Press run
4. Collect the amount of coins
    a) Loop the number of times equivalent to the number of coins

        i. Follow the sound to the current coin
5. Finish the run
6. Check finished run status
    a) Check run stats
        i. Look under Run tab
    b) Check your tracks on the map
        i. Look under Map tab
        ii. Does the track seem legit? Yes or No:_____
    c) Check coin stats
        i. Look under Stats tab
7. Take printscreens of the three screens.
    a) Loop through Run-, Map- and Stats screen by swiping
        i. Take print screen (Default key combination is pressing home button and lock button simultaneously)
8. Exit the finished run
    a) Press Checkmark-button

A.3. Interview questions:

1. Gender? (f/m)
2. Age?
3. How often do you exercise?
    a) 3-7 d/week
    b) 1-2 d/week
    c) 1-4 d/month
    d) Fewer
4. What type of exercise do you perform?(Running, gym, other)
5. Have you used other applications while running? (If so, which ones?)
6. Do you like the concept of using an application to motivate running?
7. Did you understand in which direction you were supposed to run? Explain.
8. Was the design of the application easy to understand. Did you understand what you were supposed to press in order to achieve your goal?