CHALMERS

Running with Sound

Android Application Simulating Sound Sources at GPS Coordinates Using Smartphone Sensors

Bachelor of Science Thesis in Computer and Information Technology

Marcus Bernhard
Daniel Johansson
Joakim Johansson
Linus Karlsson
Anton Palmquist

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2014

Running with Sound
Android Application Simulating Sound Sources at GPS Coordinates Using Smartphone Sensors

Marcus Bernhard
Daniel Johansson
Joakim Johansson
Linus Karlsson
Anton Palmquist

Examiner: Arne Linde

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000
Department of Computer Science and Engineering
Göteborg, Sweden June 2014

Abstract

Abstract/Summary text

Contents

1 Introduction

There are various reasons to why people run; some people do it to stay healthy, others do it for the competition. However, a large part of the population seems to struggle finding motivation to run or to exercise at all for that matter. A way to motivate those people could be to make running more entertaining by adding game-like elements, hence abstracting away from the aspects of exercising that might be considered as unattractive.

1.1 Background

In recent years smartphones containing GPS (Global Positioning System) sensors have become increasingly common {citation needed}. As a result of this, multiple smartphone applications making use of the GPS for tracking the user position and movement have emerged. One such example, *Endomondo Sports Tracker* ("Endomondo" 2014),2014-03-28} launched in 2008 {Endomondo, 2014-04-04}}. The Endomondo application tracked the position and time of the smartphone while the user was running and utilized those values to calculate statistics such as speed, distance and multiple averages.{Endomondo, 2014-04-04}. Another application, called *Zombies, Run!* ("Six to Start" 2014), launched in 2012 and had a lot of similarities with the application described in this report: Zombies, Run! focused more on a fun game experience than on running statistics and therefore provided an illusion of zombies chasing the user. This was done by generating the sound of zombies (a type of monsters) approaching the user. The sound would only disappear if the user ran fast enough over a period of time, thereby evading the zombies. However, the zombie sound effects are not directional. Thus, the user did not have to run in any specific direction to evade these monsters.

Meanwhile, binaural sound had been around in computer games for several years {citation needed}. [—-| Include some example here |—-] In these games, however, the sound generation was based on the movement of the virtual character, not the movement of the actual user. The possibility of tracking the head movements of the actual user using a smartphone was however studied as a master thesis by Paul Lawitzki in 2012 called Application of Dynamic Binaural Signals in Acoustic Games (Lawitzki 2012).

Combining both the orientation sensors of the modern smartphone with GPS and the concept of the gamified running application, this thesis investigates how binaural

sound can be used to motivate exercise.

1.2 Purpose

The purpose of the bachelor project is to design and implement an android application, which will be easy and fun, for android smartphones. It will be easy and fun to use so that it will encourage running amongst the sedentary society of the present time. The goal of the application is that it will contain functionality that depend on the hardware sensors that we can find in the modern smartphones as well as the ability to track ones performance while using the application. The application will be a fully functional running game that will be possible to take out and use in real life.

1.3 Report Outline

The following section describes the overall layout of the report.

Chapter two takes up the theoretical framework behind the most important parts of the application. The subsections of chapter two are Sound, Sensors, Database and the Design patterns of the application. Furthermore, it describes how a random location is generated by using predefined algorithms.

The third chapter describes how the different parts in chapter two and the features of the application was implemented into the application.

The last part of the report consists of the result, discussion and the conclusion. The Result part describes the end product that is the application as well as the result of the user evaluation that was completed to evaluate the performance of the application. The whole design and implementation process is discussed as well as possible future work.

2 Theoretical Framework: Sensors, sound 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 the some of the sensors in a smart phone works. It also describes different the various design patterns concerning the interface and the structure of the database.

2.1 Sound

The functionality and usability of the application are vastly depending 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's 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 the sound. ITD, on the other hand, describes the time difference 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, the incoming sound is being filtered even further (Algazi and Duda 2011). Along with the directional characteristics of the torso and head, it helps us determine where a specific sound source is located. The filtering properties of the pinna even makes 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 filtering 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) - as with our application (Kleiner 2011).

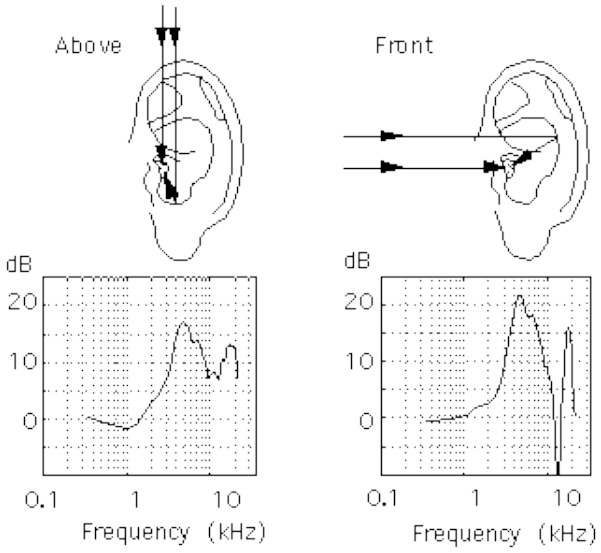

Figure 2.1: Figure showing the difference in measured frequency response depending on the location of the sound source.
(*Elevation Cues.* 2011)

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 input 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. Using only stereo headphones it's therefore possible, with the help of HRTF, to create the illusion of a sound signal being heard in an anechoic chamber (Algazi and Duda 2011). That, however, assumes individualized HRTF measurements. Since the pinna is shaped differently on every 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) (*General information on 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 seems to have been done on sound localization through spatial audio. A study made on four sighted inviduals showed that the average error in localizing sound was 8° (Barreto 2009). Under the same conditions, four blind inviduals 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 invidualized and generic HRTFs, with the virtual sound's position only moving in the vertical plane. The study suggest not only that the accuracy of the elevation

perception decreases as the virtual sound is moved further up and down. But also that the HRTFs being used has a great impact on the elevational accuracy (the invidualized HRTFs provided greater performance).

A general problem with non-invidualized 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 audio

In Auditory Display: Sonification, Audification, And Auditory Interfaces, Kramer (1994) mentions several reasons to why using audio feedback can be advantageous over visual feedback. First of all, 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 - hence letting the user 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 to not see something, but it's more 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 can not easily be achieved using it, and hence adding to the annoyance of audio feedback previously mentioned. Overall, nonspeech sounds been been proven to be a better choice when providing the user with continuous information (Sears and Jacko 2007).

Studies of enviromental 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's 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 amongts some

users (Kramer 1994). It's suggested that there are two type 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 was important to determine the direction in which user is facing; the *user direction*. Since the orientation of the phone relative to the user is unknown (a user may carry a phone in many ways), the problem of acquiring a user direction is not elementary. However, a sufficiently accurate user direction can be calculated using a combination of multiple sensors.

2.2.1 GPS

The Global Positioning System - GPS 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 put out in six different orbits circulating 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 pseudorandom number codes (PNC). Each PNC consists of information of the satellite's position 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) (Bajaj and Lalinda Ranaweera 2002).

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

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

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 and the center being the satellite. To acquire the GPS receiver's location on the sphere,

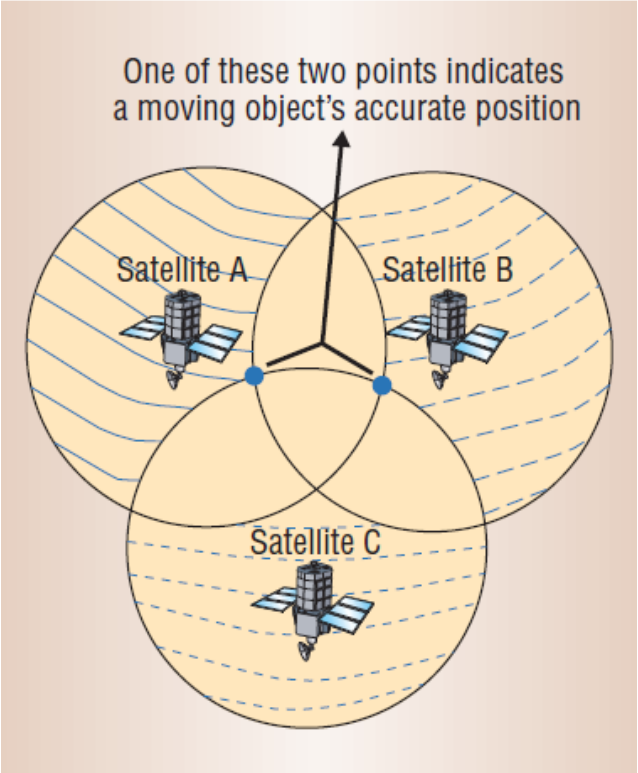

Figure 2.2: Location finding using trilateration
(Bajaj and Lalinda Ranaweera 2002)

the method trilateration as shown in Fig. 2.2 (Bajaj and Lalinda Ranaweera 2002) is used.

In the context of GPS technology, trilateration is a way to narrow down the possible locations by determining where spheres intersect. 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. 2.2. 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 Earths surface. 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 is having 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 clock in the satellite that is being synchronized to is atomic and thereby very accurate (Bajaj and Lalinda Ranaweera 2002).

The positioning accuracy is however affected by some factors of which one has to do with the speed of light only being constant in vacuum. This might cause a

slight delay to a PNC as it gets slowed down by electrically charged particles in the ionosphere and troposphere (Drawil and Amar 2012).

Another factor is the so called satellite ephemeris error, which is caused by differences in the anticipated and the actual position of a satellite. Both this error and the error caused by the electrically charged particles can be adjusted for giving a final accuracy of a few meters (Drawil and Amar 2012). The adjustment is done with a technique called differential GPS, which in short consists of comparing the data acquired with the data in a nearby stationary GPS receiver with a known location. It is likely that the location offset in the stationary receiver is the same in mobile GPS receivers nearby, which thereby will receive correction signals (Bajaj and Lalinda Ranaweera 2002).

Differential GPS can however not adjust to the so called multipath error which occurs when PNC's bounces on the surrounding topography, as buildings or hilly terrain. To avoid this error the receiver needs to have a clear view to the satellites. Otherwise the multipath error might, according to Drawil, Amar and Basir (Drawil and Amar 2012), cause an error up to 80 meters.

In this project, the GPS technology is mainly utilized to calculate the user position relative to the virtual sound sources, such as coins and monsters. However, by keeping track of previous GPS locations, it is possible to acquire a user bearing. This bearing is the direction that 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" 2014b). 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.

2.2.2 Accelerometer

The accelerometer gives the linear acceleration of device in the device's coordinate system("Android developers" 2014c). As illustrated in Figure Fig. 2.3 on page 9, the accelerometer gives values as a 3-dimentional 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 rotation in a global coordinate system.

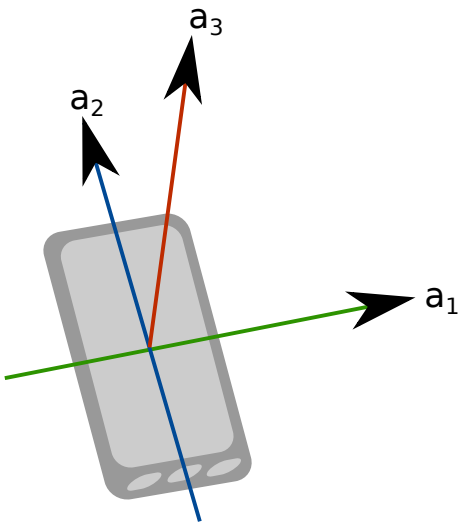

Figure 2.3: Coordinate system and measured vector of the accelerometer.

2.2.3 Magnetic field

The magnetic field sensor determines current magnetic field strength ("Android developers" 2014c). As mentioned above, it can be used together with the accelerometer obtain device orientation.

2.2.4 Gyroscope

Gives the rotational speed of the device around each of its axises ("Android developers" 2014c). While it has a lot less disturbances than a accelerometer and magnetometer compass, it only determines change of direction and not the direction itself. The gyroscope gives values as a 3-dimentional 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 (green in Fig. 2.4), g_2 is the rotation around the y-axis (blue in Fig. 2.4) and g_3 is the rotation around the z-axis (red in Fig. 2.4).

2.3 Generation of random locations: Different approaches

To be able to quickly use the application when the user want to get out and run the sound sources (checkpoints) are generated out of a predefined algorithm. The checkpoints must be generated at reachable locations and can for example not be placed in a lake or a building but instead on a road or trail. There are several

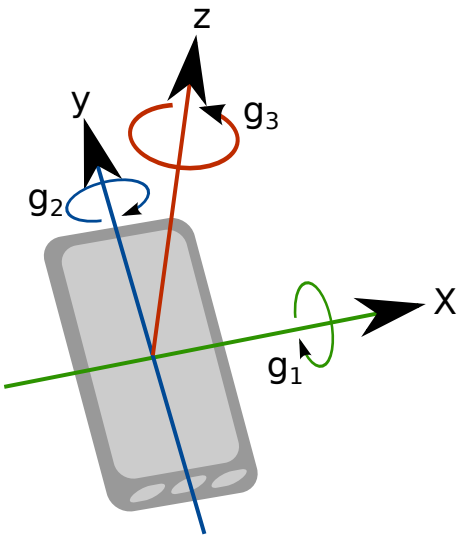

Figure 2.4: Coordinate system and measurement vector of the gyroscope.

methods to generate a location that is reachable and two methods is described in this chapter.

Colour classification method One way is to first randomly generate a location. This location is then evaluated if it lies 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, light green is equal to grass, and so on one can compare the colour on a location and then 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 an colour that is accepted, for example light green it is saved and can be used in the route. In order to implement this an accurate map is required that is linked to some kind of coordinate system. It must also be possible to extract the colour of the random location.

Geocoding API request method Another approach is to use Google maps and the function directions. As before a random location is generated. Then a geocoding API request is sent to Google asking for the walking direction from the users location to the random location. The answer from Google gives the direction to the closest reachable point nearby the location. This closest point is then extracted and is used as the random location ("Google Developers" 2014).

In this project the second approach is implemented and it is described how in chapter 3, **??**. The different approaches is also discussed in chapter 5...

2.4 Database: Relational model and visual representation

A relational database collects data in tables. The columns of the table represents the different kinds data that is stored and each row is a specific object. It is also important to have a column that stores a unique identifier(ID) for each row, this makes it possible to obtain a specific row from the database and to specify relations between different tables. An example of a simple database is that of a hotel, the columns will store the room type, the price per night as well as an ID, in this case the ID could be the room number. Then a specific row could look like this: "Double, 100, 123", by selecting the the row with the room number 123 you will get the information about the room, that it is a double room and costs 100 per night.

The first step when designing the database is to design a is to make a visual representation of the database, this is done with a so called Entity/Relation-Diagram(ER-Diagram). An ER-Diagram consists of three main parts: entities, relations and attributes (Widom 2009, ch. 4.1). The entities are the different object that we want to store 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 the what will become the tables of the database and the attributes are the columns of the tables.. The relation part of the ER-Diagram holds relation between different entities of the database, for example it could hold the relation between hotels and their rooms. In the ER-Diagram entities are represented by rectangles, attributes by ellipses and the relations by diamond. In the database a relation is often a separate table that stores the relation between different entities.

2.5 Software models

activities, fragments, etc

2.6 Standard Design patterns

In (Tidwell 2011) it is stated 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. Design patterns is a way to capture and group common structures together, thus easing the usability when applied.

2.6.1 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. 2.5 the sign up-button is prominent and thereby clearly telling the user that is should be pressed after the form has been filled in.

Figure 2.5: Example of a *Prominent "Done" Button.*

2.6.2 Carousel- and filmstrip pattern

Another design pattern is the *Carousel*, which is a horizontal swipe- or scrollable list of items that are visually interesting(Tidwell 2011). The pattern is useful when the user does not know exactly what item he/she 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. 2.6 it is clear that the item in the middle is in focus since it is enlarged. By seeing the elements immediately around the focused item the user will get a clue about the possibility to browse.

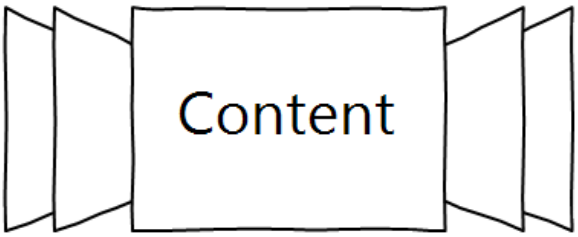

Figure 2.6: Concept of the carousel pattern.

2.7 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 application on the Android platform(LehtimÃ€ki 2013). Below follows a description of the patterns that are used.

2.7.1 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. 2.7, which can perform actions to the app. It has been around for a long time and is one of the most recognizable patterns(LehtimÃ€ki 2013).

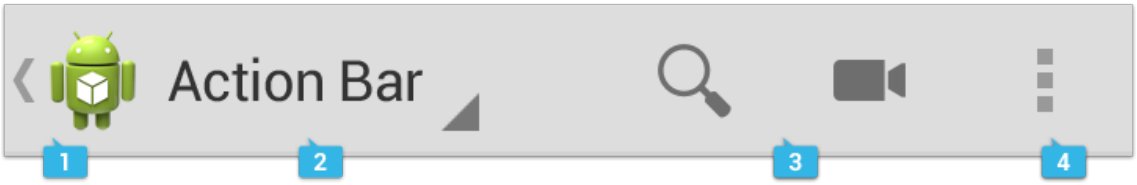

Figure 2.7: An Action Bar and its components App Icon (1), View control (2), Action Buttons (3) and Action overflow menu (4).
("Android Developers" 2014a)

The most important feature of the action bar is perhaps, as the name suggests, its actions. The actions that are most important in the app should be put among the Action Buttons as in (3) in Fig. 2.7. 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(LehtimÃ€ki 2013). A final criteria might be whether the action

Question: Are we allowed to point to numbers like this?

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 put in the overflow menu instead. The overflow menu is opened by a click on the menu button (4) in Fig. 2.7. 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 app logo (1) in Fig. 2.7 which establishes the identity of the app. When an app has different views the app logo becomes a consistent identifier letting the user know which app is currently running. The app logo 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. 2.7, or a home button taking the user to the home view("Android Developers" 2014a). The button of the app logo may also be used to open a side menu called navigation drawer, which will be covered in sec. 2.7.2) with more navigation options(LehtimÃ€ki 2013).

Another component giving a sense of location is the View control as (2) in Fig. 2.7, which apart from displaying the name of the app 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.

2.7.2 Navigation Drawer

Android Developers (("Android Developers" 2014d)) suggests that actions should be put to the right and navigation to the left. However there is a risk that putting navigational options into the action bar, as in (4) in Fig. 2.7, 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 to partly cover the current view. It is accessed by swiping inwards from the left outside part of the screen or by pressing the app logo as shown in the leftmost figure in **??**. The menu is then used to open a desired screen as shown in the middle and rightmost figures in **??**("Android Developers" 2014d).

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 app icon as inFig. 2.8.

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 is basically a landing screen consisting of big icons which take the user to different parts of the app(LehtimÃ€ki 2013). The Dashboard pattern used to be one of Google's official recommendations but has been more or less 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

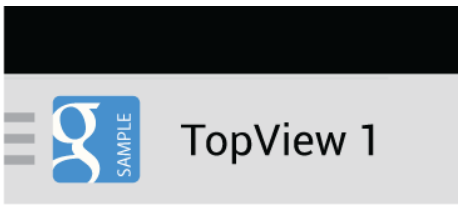

Figure 2.8: An app icon provided with the navigation drawer indicator. ("Android Developers" 2014d)

then can be taken directly into the view that is mostly used instead of picking 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.

2.7.3 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 putting the different parts on different tabs that the user can swipe horizontally between(LehtimÃ€ki 2013).

As long as the number of tabs are limited to three or less it is considered as a good solution. If the number exceeds three however it may be better to use the pattern called title strip where the user may swipe between an unlimited number of tabs(LehtimÃ€ki 2013).

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 Audio in an Android application

The application programming interfaces (APIs) provided by Android as a part of the Software Development Kit (SDK) provides several classes for handling audio, like AudioTrack, MediaPlayer and SoundPool. However, none of these provides any support for 3D audio operations as for now (Ratabouil 2012). To implement binaural audio, one needs to access operations that aren't covered in the SDK. Fortunately, Android provides an API for accessing low level functions called Native Development Kit (NDK) (*Android NDK.* 2014).

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 doesn't generally result in better performance and on top of it, it also increases the application's complexity (*Android NDK.* 2014). However, it enables developers to use operations that aren't accessible using only Java - such as handling 3D audio. Currently, there seems to be two APIs available providing this functionality: OpenSL ES and Open AL.

3.1.1 Spatial audio using external libraries

OpenSL ES is a cross-platform API that enables audio functionality in native applications on multimedia devices (Ratabouil 2012). It's 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 (*OpenSL ES.* 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 supports the game profile at the time the book was written.

After implementing a simple test application using OpenSL ES and its spatial audio features, testing it on our own phones* did indeed result in a SL_RESULT_FEATURE_UNSUPPORT

being thrown. That basically means the phone is not compiled with the correct profile supporting the feature implemented in the application {OpenSL EL manual}. That makes the API useless to our purpose - implementing 3D audio - and hence blabalbalbal

OpenAL is, just like OpenSL ES, an API that enables the creation of spatial sound environments independent on the soundcard setup. As a result it eliminates the problem with programmers having to write different code to each device because of each soundcard supporting different instructions.

The OpenAL version we used has been adapted to Android by Martins Mozeiko and Chris Robinson. They have patched the code to make it work on Android. (http://repo.or.cz/w/openal-soft/android.git) To make it easier to access native functions we are using a small library provided by Martien Pielot, called OpenAL4Android (http://pielot.org/2011/11/10/openal4android-2/). It serves as an intermediary between the native OpelAL code and our Java application and contains all methods necessary to create a binaural environment. Apart from moving the sound source in three dimensions, it's also possible to move the 'listener', meaning where in the 3D space the listener is supposed to interpret the audio signal from.

Our sound sources places in the 3D environment are static (since that's how our application works). The only thing getting updated as the user is moving is the angle of the 'listener' to sound source.

3.1.2 Feedback to the user

One of the biggest part of the application is the

Something about

As mentioned in the theory chapter, non-invidualised HRTFs suffer from back-front confusion. It would therefore be risky to rely single-handedly on the 3D properties of HRTFs in an application depending entirely on the directional properties of its audio. To avoid

BILD PÅ HUR LJUDET ÄR FÖRDELAT

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 order to get faster and more accurate orientation values than when using 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. Since 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 and $\mathbf{a} = \begin{Bmatrix} a_x \\ a_y \\ a_z \end{Bmatrix}$ is the raw accelerometer readings, 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 other axes are not needed, since altitude of sound sources will not be considered.

$$\mathbf{g} \times \mathbf{a} = (g_x * a_x) + (g_y * a_y) + (g_z * a_z) \tag{3.1}$$

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

$$\omega = -(g_x * a_x) + (g_y * a_y) - (g_z * a_z) \tag{3.2}$$

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, $\triangle v$.

$$\triangle v_n = \omega_n * \triangle t_n \tag{3.3}$$

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

$$v_n = \sum_{i=0}^{n} \omega_i (t_i - t_{i-1}) \tag{3.4}$$

However, this method only calculates a current angle relative to a start angle. In other words, the user would clearly perceive a direction of 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 coin 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:

$$o_n = \begin{cases} \beta_n & if\ new\ GPS\ bearing \\ o_{n-1} + \omega_i(t_i - t_{i-1}) & else \end{cases} \tag{3.5}$$

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 is only implemented as optional in the final application.

3.3 Generation of random locations

To be able to quickly use the application when the user want to get out and run the sound sources (checkpoints) are generated out of a predefined algorithm. The checkpoints must be generated at reachable locations and can for example not be placed in a lake or a building but instead on a road or trail. To accomplish this the application uses Google Maps geocoding API and its routing features sec. 2.3. In this sub chapter it is explained how the algorithm using Google Maps geocoding API has been implemented in this project. All the calculations has been done with the geographic coordinate system using latitude and longitude and has been inspired from (atok 2013).

Every time the algorithm runs is a theoretical route generated based on the number of checkpoints and the distance that has been set in the settings by the user. If the algorithm is based on the same settings this theoretical route will always look the same. As seen in Fig. 3.1 a circular dotted route has been generated with four checkpoints and a distance k. 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 $(2 * \pi)$ by the number of checkpoints, the angle x, between them is calculated as in 3.6.

$$x = \frac{2 * \pi}{numberOfCheckpoints} \tag{3.6}$$

To get the next checkpoint, the calculated angle is added to the starting position and create checkpoint 2. The same is done for every checkpoint until all have been created.

To make the routes different from each other every time the algorithm runs the theoretical route is rotated around the location of the user by an random angle y. The random angle is calculated by multiplying a full revolution $(2 * \pi)$ by a random number between 0 and 1. The rotation is done by multiplying the difference in latitude and longitude of every checkpoint compared to the location of the user, with a two-dimensional rotation matrix with the angle y as in 3.7 and 3.8.

$$newLongitude = \cos(y) * differanceInLongitude + sin(y) * differenceInLatitude \tag{3.7}$$

$$newLatitude = -sin(y) * differanceInLongitude + cos(y) * differenceInLatitude \tag{3.8}$$

The new longitude and latitude is then added to the location of the user for every checkpoint and a random route is now generated, continuous lined circle in Fig. 3.1.

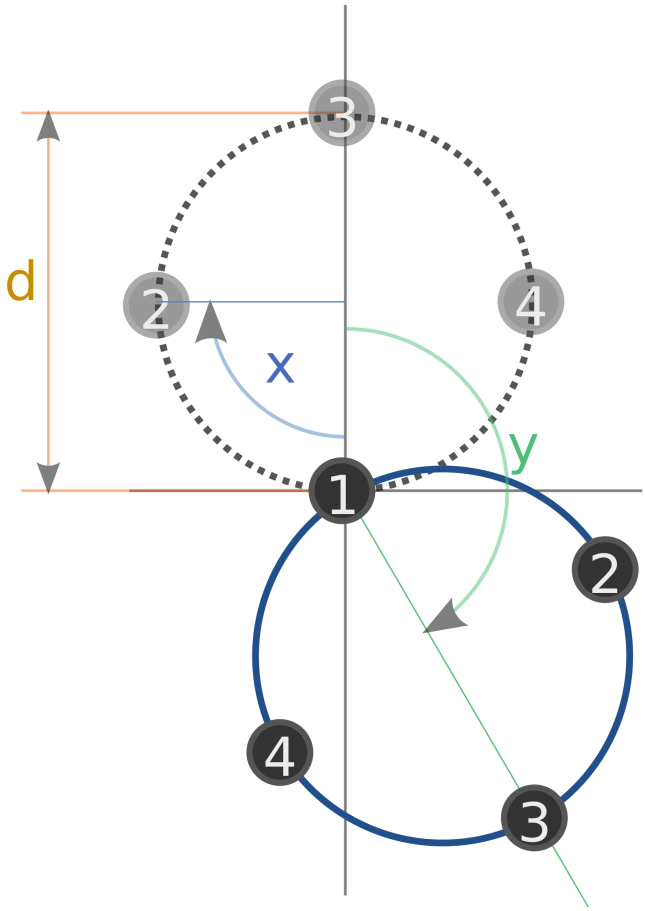

Figure 3.1: Random...

To make this randomly generated route reachable a geocoding API request of directions from your current location to the new locations is send to Google Maps ("Google Developers" 2014). The answer from Google is a list of locations of the shortest route to the destination. It is a big chance that the generated locations is not on a road but instead in the forest or in a lake. But by taking out the last location that is given by Google for every checkpoint you find the location that is as close as possible to the generated locations but it is reachable. This locations is then set as a checkpoints and the random route algorithm is finished.

Table 3.1: To fill... denna är nu fel...

1	$startLatitude$; *Latitude from the location of the user*
2	$startLongitude$; *Longitude from the location of the user*
3	
4	$a = random[0..1]$
5	$t = 2 * \pi * a$
6	$distance = distanceInMeters/constantToConvertFromMetersToLatLong$
7	$x = distance * cos(t)$
8	$y = distance * sin(t)$
9	$xCorrected = x/cos(startLatitide)$
10	$locationOfRandomLatitude = startLatitude + y$
11	$locationOfRandomLongitude = startLongitude + xCorrected$

3.4 Database: Design and Implementation

This subsection describes the implemented database that was used to save data of the users running history. It describes the system that manages the database as well as a description and some visual representation of the database.

3.4.1 Database management system

The Database Management System (DBMS) is the system that is used for the application to communicate with the database. It makes the necessary request to the database using Structured Query Language (SQL). The DBMS handles request such as adding or retrieving data from the database.

The database that was implemented uses the DBMS SQLite, it 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 taht 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 contains the appropriate SQL-queries that are executed on the database. For example if the application wants to add a 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 the relate to each other.

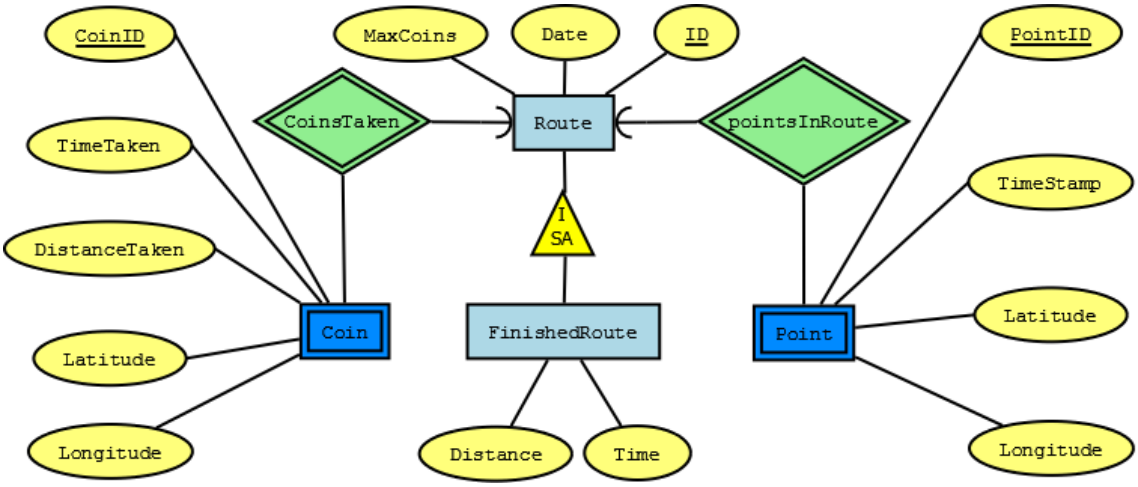

Figure 3.2: The implemented database represented by an ER-diagram.

The implemented database is centered around the entity “Route” (see Fig. 3.2), the route is basically the users running session, it contains a unique identifier and the date that the route started, also the Route saves data of the number of coins that was generated for the route, this number 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 (Widom 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 latitude, 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 inherit each of the attributes that the route has. When a route is finished we store it in the FinishedRoute table along with the total time and distance of the route.

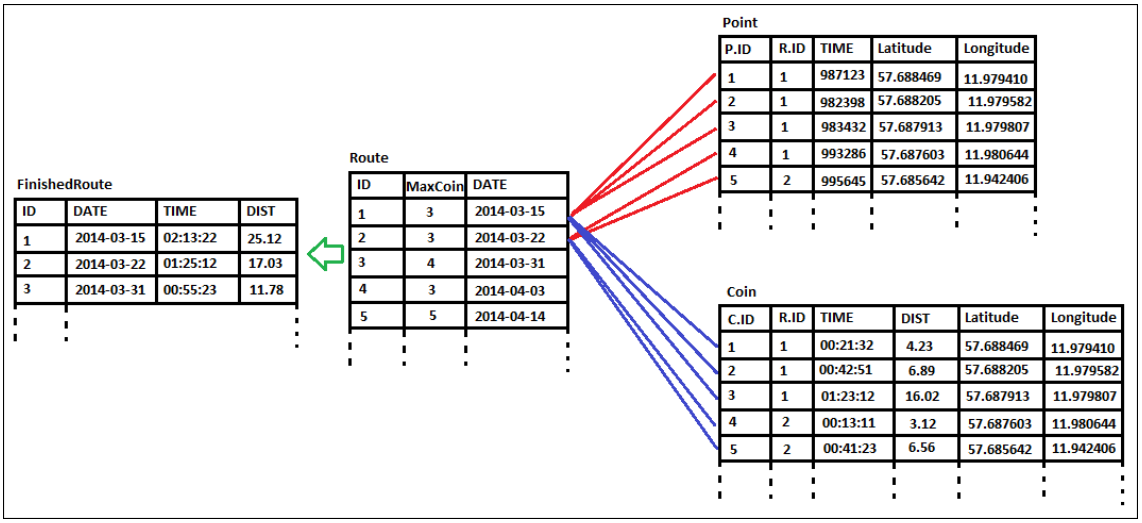

Figure 3.3: Visualisation of the tables in the database.

Fig. 3.3 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 weak relations that was represented with double borders in Fig. 3.2 was implemented. We can also see that the finished routes are stored in a table with the additional data about time and distance.

3.4.3 Using the database - Statistics

The purpose of the database is that it should be possible to recreate a stored route in the future. By extracting the points that is associated to the route we can draw the path that the user ran on a map and with the distance and time that we stored when we finished the route we can calculate the average speed and pace that the user ran in.

An special activity was implemented, called finished view, this view is where the data about a route is shown to the user. The user reaches this view first when it has finished a running session, and it displays a summary of the run. This finished

view is very similar to what the user sees on the screen during the running session but now the values are frozen and the buttons have changed appearance.

The finished view consists of three different parts, the run-, map- and stats screen. The run screen displays a variety of information of the run, such as time, distance and average speed. These data are retrieved from the FinishedRoute table in the database. Basic formulas are used to calculate the average, that is distance divided by time. The map screen displays a map that shows the path that the user ran as well as the location of the coin that was collected. The path is drawn by retrieving all the points associated to the specific route from the "Points" table of the database, then a line is drawn between all the points and thus forming the trail that the user ran. The coins are placed on the map in the similar way, the locations of the coins are retrieved from the database and displayed on the map with a small coin icon. The stats screen displays the time and distance of each collected coin in the form of a table.

A history list was implemented that contain all the routes in FinishedRoute, this is where the user can access all its previous running sessions. The elements in the list displays the date of the run as well as the time and distance of the run. These list elements are selectable and when a element is selected it will open a new window which is the finished view mentioned in the previous piece of this report. The route that was selected will be displayed in the finished view, this allows the user to examine and compare all its previous runs.

3.5 Design choices and usability

To make the game pleasant to use it has been designed taking usability factors in account. The process has been to establish desired requirements, creating design alternatives out of it and finally putting this in the final prototype.

3.5.1 Establishment of requirements

As a starting point inspiration was taken from the apps Endomondo ("Endomondo" 2014) and "Zombies, Run!" ("Six to Start" 2014).

These apps are in some sense their opposites. Endomondo focuses on the running experience and lets the user quickly get started with a run. "Zombies, Run!" on the other hand focuses more on external game elements like outrunning zombies, picking up items and the upgrading the user's base camp. Also Endomondo puts a lot of focus on detailed statistics which is not the case for "Zombies, Run!".

The desired requirement is to combine the best of the two worlds represented by the two apps. It is wanted to get the app to be as straight forward as Endomondo when starting a new run while still letting the app be fun and having game elements like in "Zombies, Run!".

3.5.2 Design of alternatives

Beskriv designvalen och förklara varför de är bra

3.5.2.1 General

When starting an app a splash screen may sometimes be motivated since it gives the user feedback that the app is loading. When the loading takes a long time, in games for example, it is a good idea to use a splash screen. However, the fact that the loading of the splash screen itself requires resources suggests that it is better avoided. In Run for Life a splash screen would probably take almost as long time to load as the app's start screen (also known as landing screen), which would make it inefficient. A better solution was making the landing screen light weight and low in its demanding of resources(LehtimÃ€ki 2013). Thereby the Run For Life-start screen was chosen not to include loading of all the heavy algorithms that are needed to calculate a route. Instead the screen provides a clear overview of what the app does and lets the user start a run from there.

3.5.2.2 Main screen

At first the dashboard pattern was considered for the landing screen. The reason for this is that it clearly presents all different parts of the app 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 he/she wants to start the run-mode, view the history or something else. {Smashing, 318-319} However, the fact that the app's main focus is the running part it seemed like a better idea to on startup take the user directly into the view that starts a running session. This reduces the excise for the user by one click when he/she wants to start a run.

Undvikande av dashboard

Having said that, the user will at startup arrive into a landing screen providing information about whether the GPS and headphones are connected. The user may then be able to fix these errors by enabling the GPS and inserting the headphones.

To easily browse between different game modes the swipeable pattern carousel was chosen{Källa från Designing interfaces boken}. Since carousel is not a standard pattern in Android the external library Fancy Cover Flow was implemented{https://github.com/davids

Coverflow for game selection, well known pattern

As mentioned earlier the running session is not started at the landing screen to minimize processor usage. However it was decided that it from there should be quick and easy to start the run. The running session is thereby easily started with

a prominent done button in the bottom right of the screen. {Källa från Designin Interfaces}

Filmstrip

Prominent done button. Likhet mellan grönt-grönt, rött-rött. <hitta källa i designing interface-boken>

Action bar

Action bar overflow menu

3.5.2.3 Running screen

In the running screen the pattern workspace is used. The reason for this is that there Too much information to fit one screen

Navigation Drawer -> History Motiverat eftersom det kommer vara många olika sidor

3.5.3 Implementation of prototype (Beta)

3.6 Game modes?

3.6.1 Coin collector

4 Results - Application Simulating Sound Sources when running

This bachelor thesis has result in an android application that with sound can be used to enhance the motivation for running exercise. The following chapter will present this result and describe what is included in the application.

4.1 Presentation of the application

The main screen Fig. 4.1 A, is showed when the application is started. To help new users a dialog telling how the application is supposed to be used is showed the first time it is started as well as directions to the help section. In the lower left corner of the main screen two status icon is showed, one telling if you have a GPS-connection and one telling if your headphones is connected. If the GPS-connection or headphones are missed a cross is put above the icons. In the middle of the screen a mode selector is displayed. By swiping right or left it is possible to choose between two modes, Coin Collector and tutorial mode, which will be explained later. If the image for coin collector or tutorial mode is pressed a text is showed describing how this mode is played. In the upper right corner it is possible to change some settings. This settings is connected to the Coin Collector mode and is concerning how long the route is going to be as well how many coins (checkpoints) that should be collected. By pressing the hamburger icon in the upper left corner or by sliding from the left side on the screen the navigation drawer is displayed Fig. 4.1 B. From this drawer you can navigate to different pages, the main screen, the history view and the help section. In the history view a list of all the runs is saved containing all the data from the runs. More of this will described later. The help section contain information about how the application is suppose to be used. The help section also contains different sounds that is used in the application and describes when each sound is used.

The main objective of this thesis has been to implement a way to navigate with the help of sound and this has been implemented in the game mode, Coin Collector. By selecting Coin Collector in the main screen and then press the “Next” button the game mode is started as long a GPS-connection and headphones are plugged in. If GPS-connection or headphones are missing a pop up is showed telling what is missing. To be able to use the application in a proper way a GPS-connection and

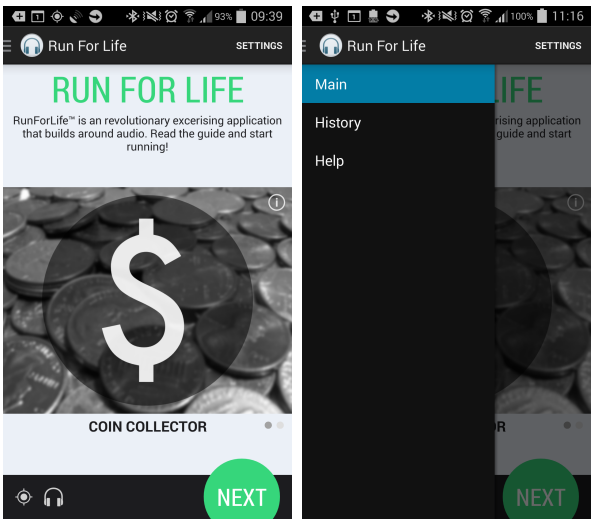

Figure 4.1: The main screen and navigation drawer

headphones has to be used. When the "Next-button" is pressed the game mode is started. The first thing that happens is that the route that the user is going to run is generated out from the settings that was specified from the main screen. For example if three coins (checkpoints) is selected, three coins is going to be generated in a specified distance from the user. The coins is generated in a circle from you and by integration with Google maps the coins is going to be located on reachable location such as roads and trails. To start the run the "play-button" in the lower right corner of Fig. 4.2 is pressed. Now starts a ticking sound simulating the source of the coin, but in order to hear the coin from the correct direction the user has to run for approximately 10 meters to acquire a bearing from the GPS. Out of this bearing the sound is balanced between the ears to imitate that the sound is coming from a specific direction. The user is now supposed to run in the direction of the sound in order to reach the coin. To easier understand the distance until the coin is reached a voice feedback is implemented, telling the number of hundreds meters to the coin. For example, if the user just passes 200 meters until he reach the coin a voice saying "200 meters" is played. The playback speed of the sound is also played faster the closer to every hundred meter the user gets to confirm that the distance to the coin is decreasing. When the user passes a hundred meter border the playback speed is restored to a slow speed again. To further enhance the navigation, a different, unpleasant sound is played when the user is heading in the wrong direction. When the user on the other hand is heading right against the coin a techno sound starts playing to confirm the user that he his heading in the exact right direction. In all other cases the ticking sound is played except that the pitch of the sound changes depending on how close to the right course the user are. If the user is almost on the right course the sound has an high pitch and if the user is out of course the sound has an low pitch. When the user reaches a coin a sound indicating that the coin has been reached is played as well as a voice telling the user that a new coin

is generated. The user now have to collect all the coins and when all the coins is collected a voice is going to tell the user that the run is finished. Since the last coin always is generated on the location that the user had when the route is generated the user is now back to where he started.

The run can at any time be paused by pressing the “Pause” button that under a run will take the place of the “Play” button. This will pause the sound and the collecting of the user locations. The run can be continued by again pressing the “Play” button or the user can finish the run by pressing the “Stop” button that will be showed when the run is paused. The user can at any time inspect the three different fragments that show information about the run so far. The first fragment “RUN” shows information about distance, time, speed and pace. The second fragment “MAP” shows information about the users location and where it has been running on the map. The last fragment “STATS” shows statistics about how many coins that has been collected and at what time and distance.

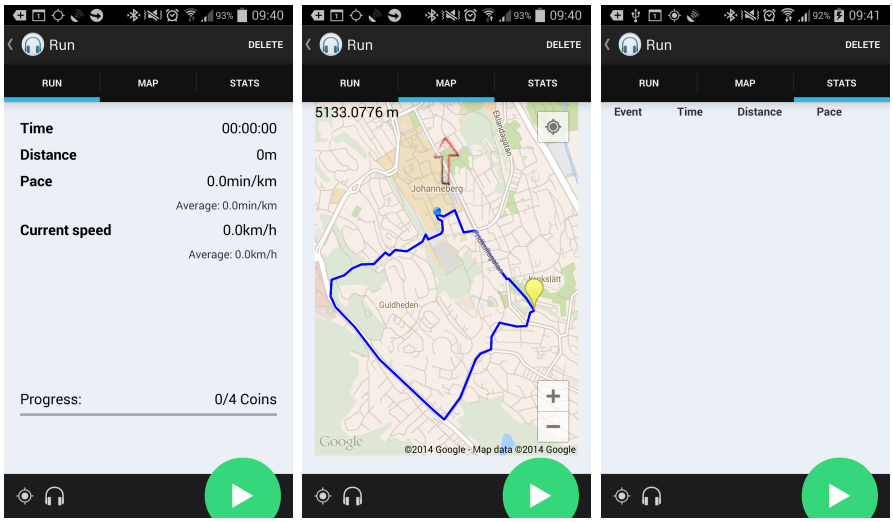

Figure 4.2: ...

When the run is finished the view from Fig. 4.2 is changed to a new view containing information about the run Fig. 4.3. On the first fragment named “RUN” can the user find information about the distance, time and speed of the run as well how many coins that was collected in relation to how many coins the route was containing. If the user swipes over to the “MAP” fragment, information about where the user has been running is displayed. The route that the user has taken is marked by red lines on the map and the dark coins is marking where the coins was collected. The last fragment “STATS” contains information about when the coin was collected, at what distance and what pace the user had. To exit this view the “OK” button is pressed sending the user back to the start screen of the application as well as saving the run to the database so that the user can access it from the history view.

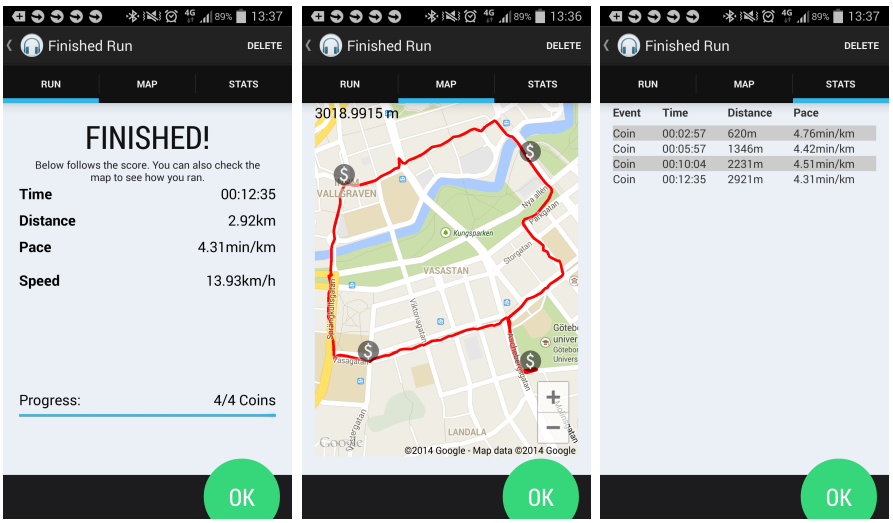

Figure 4.3: Bilder med finish läge

4.2 Usability evaluation

4.2.1 Observations

4.2.2 Interviews

5 Discussion

Förklarar vad resultatet innebär, bland annat. Diskuterar även metoden och relationen mellan resultatet och metoden. Skulle man kunnat göra på något annat vis, etc. Kan även koppla tillbaka till teorin.

Tänk dig trulig tonåring som svarar "och?". Provocera fram svar.

Lyfter fram resultatet och därmed ert bidrag till kunskap om ämnet.

Introduktion Påminner läsaren om det övergripande problemet i studien, sammanhanget, rapportens syfte, och vilken typ av undersökning som gjorts för att komma fram till resultatet

This project have developed an sound based running app ... by investigating android development smartphone sensors, etc ...

"Sammanfattning" En (kort) översikt över studiens genomförande, datainsamling, analysstrategi och resultat (i viss mån)

The project resulted in...

Diskussion - vad vi har kommit fram till - Belyser och lyfter fram det anmärkningsvärda, nya, överraskande, intressanta och sätter det i relation till det kända inom fältet, genom:

The fact that... However... Comparing with the master thesis *Application of Dynamic Binaural Signals in Acoustic Games*(Lawitzki 2012), the approach of calculating the user orientation has been a lot more complicated in this project. Lawitzky had the smartphone firmly attached to the user, giving him the possibility to measure orientation with great accuracy.

This project had the requirement of allowing users to wear their phones in any way they preferred. Hence, while a firmly attached phone would have made the user orientation a lot more accurate, it would be too impractical for this project. The workaround of using a GPS bearing and gyroscope combination is controversial...

Konsekvens: Detta innebär... (vad betyder det?) Påverkar resultatet och slutsatserna vi drar vår förståelse för problematiken i fråga? Kommer det vi har kommit fram till att påverka hur man genomför andra studier inom detta område? Kritisk reflektion: stämmer överens? Går emot? Hur ska detta tolkas ?

This means that...

5.1 Future work: efficiency, modularity and prospects

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 frequency to indicate distance (as explained in , but that limits the types of sounds that can be used. With closer sound sources, the sound might be more natural. An 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 he/she 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 get 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.

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), the prey 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 but a better option would be to have a server based database system that

would have a web interface or it could even be merged with Facebook or other social media to allow the user to share their progress as well as read about the achievements that their friends have made. This would make it easier for the users to access their data from different devices and the progress would not have been lost even if the user would replace their current smartphone.

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 a head-mounted smartphone (the user has actually turned their head when the sensors give new values). 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

Slutsatsen är ett oerhört kort kapitel där man skall svara på frågan: “Men vad kom ni fram till då?”

“Sammanfattningsvis så...”

Acknowledgments

Thank you, thank you, thank you

Bibliography

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

"Android Developers" (2014a) Action Bar. http://developer.android.com/design/patterns/actionbar.html. http://developer.android.com/design/patterns/actionbar.html.

"Android developers" (2014b) Location - getBearing method. *Website*. http://developer.android.com/reference/android/location/Location.html.

"Android developers" (2014c) Motion Sensors. *Website*. http://developer.android.com/guide/topics/sensors/sensors_motion.htm.

"Android Developers" (2014d) Navigation Drawer. https://developer.android.com/design/patterns/navigation-drawer.html. https://developer.android.com/design/patterns/navigation-drawer.html.

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

Bajaj, R. et al. (2002) GPS: Location-Tracking Technology. *Computer 35 (4)*.

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*

Drawil, N. M. et al. (2012) GPS Localization Accuracy Classification: A Context-Based Approach. *Intelligent Transportation Systems, IEEE Transactions (14) 1*.

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

Elevation Cues. (2011) Elevation Cues. http://interface.cipic.ucdavis.edu/sound/tutorial/psych.html. (Accessed: 2014-04-25).

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

OpenSL ES. (2014) OpenSL ES. http://www.khronos.org/opensles/. (Accessed: 2014-02-23).

"Endomondo" (2014) About - Endomondo Blog. http://blog.endomondo.com/about/.

"Google Developers" (2014) The Google Geocoding API. https://developers.google.com/maps/documentation/geocoding/. https://developers.google.com/maps/documentation/geocoding/.

Grubesa, T. et al. (2004) *Simulation of virtual 3D auditory space with HRTF*. University of Zagreb.

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.

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

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

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

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.

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/. https://www.zombiesrungame.com/press/.

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.

"SQLite" (2014) "SQLite". https://www.sqlite.org/whentouse.html. https://www.sqlite.org/whentouse.html.

Tidwell, J. (2011) *Design Interfaced*. 2ndth edition.

Widom, H. G.-M. J. D. U. J. (2009) *"Database Systems - The Complete Book (2nd Edition)"*. Upper Saddle River, New Jersey: Prentice Hall.