Thesis to get the degree Bachelor of Science

Running with Sound

Android Application Simulating Sound Sources at GPS Coordinates Using Smartphone Sensors

Marcus Bernhard
Linus Karlsson
Anton Palmqvist
Daniel Johansson
Joakim Johansson

Date (optional)

CHALMERS
UNIVERSITY OF TECHNOLOGY

Chalmers University of Technology
Department of Computer Science and Engineering

Dean
Prof. Dr. xxx yyy

Referees
Prof. Dr. aaa bbb
Prof. Dr. xxx yyy

Date of the graduation (optional)
xx.yy.zzzz

Dedication, cite from a famous person, or whatever you like
(optional)

Contents

Bibliography **37**

Abstract

Abstract/Summary text

1. Introduction

Long before humans could drive to a grocery store or call a pizza delivery service to get food, we had to actually hunt for it. But humans were not the fastest nor the strongest animal. Instead, our feet, perspiratory system and ability to hold our head steady while moving suggest that persistence hunting might have been our way of acquiring food{citation needed, ?}. Persistence hunting works by running and tracking an animal, chasing it into exhaustion.

Nowadays however, persistence hunting barely practiced anymore. We do not run as much as we did. [Include example of why that is bad]. Exercise as a mean of survival has been replaced by exercise motivated by other reasons. So how to motivate running in this day and age? Some run to stay healthy or to win races, but that does not work for everyone. One solution is to make a game out of hunting. Instead of just running for a faster time, run toward the sound of an imaginary prey.By using modern technology: smart phones, GPS and headphones, it is possible to exercise in the most ancient of ways.

1.1. Background

In recent years smartphones containing GPS (Global Positioning System) sensors have become more and more common {citation needed}. As a result of this, multiple smartphone applications making use of the GPS for tracking the user position and movement have originated. One such example, Endomondo Sports Tracker {Endomondo, 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! {Zombies, 2014-03-28}, launched in 2012. 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 didn't have to run in any specific direction to evade these monsters.

Meanwhile, directional sound or rather 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.

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

Investigate how sound, sensors and design patterns can be implemented to encourage running.

1.3. Goals

The goal of the assignment is to create a fully functional running game that is fun to use. To do this, the following milestones have been created.

1.4. Method

Ha med i method: Arbetssätt? Scrum User evaluation

(genomförande) Behöver inte vara kronologiskt Implementerat, designat BETA, testat, omdesignat

2. Theoretical Framework: Sensors, sound and design

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 the user were to get confused by the audio, they would probably not be able to follow its instructions - and hence the whole purpose of the application would be ruined. It's therefore vital to have an audio core that is easy to interpret and follow, and that 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). ILD basically stands for the difference in volume between the closest ear and the ear being 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 {KTH-SAKEN som källa}.

When the sound reaches the pinna, which is essential to many of the directional properties of our hearing, the incoming sound is being filtered even further. The pinna is located in the outer ear, together with the eardrum and ear canal, and is the visible part of the ear. As a result of the shadowing actions caused by the shape of the pinna - along with the directional characteristics of the torso and head - humans are able determine where a specific sound source is located. It even makes it possible for us to differentiate between sounds appearing from the front and back respectively (Kleiner, 2012). Its properties are of great value when synthesizing binaural sounds seeming to appear from a certain point in space, creating a surround environment using only two speakers. As Mendel Kleiner mentions in Acoustics and Audio Technology (2012), such binaural localization is very useful when simulating virtual realities (like in 3D games and architecture) - as with our application.

To generate virtual audio environments, one can use so called head-related transfer functions (HRTFs) (Simulation of virtual auditory space with HRTF, 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. That, however, assumes individualized HRTF measurements {HRTF-BOKEN}. 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. Ways to capture data in the best way possible are currently being researched, as General information on SOFA (2014) suggests. SOFA is a common format to store spatial data measurements in.

NÅGON BILD HÄR

2.1.2. Human-Computer Interaction using sound

VÄLDIGT LÅNGT IFRÅN KLAR MED DETTA KAPITEL!

People associate different sounds to different things. Hence, a sound can be positive to one person but negative to another. It's therefore vital for the application's sounds to not confuse the listeners by sending off a wrong signal. When the user is in moving towards the goal, for example, it makes sense for the sound to give off a positive vibe. If the user is moving in the wrong direction it should be clear that something is wrong.

The are several reasons to why using sounds to interact with the user is advantageous. First of all, it reduces the demands on visual attention (http://books.google.se/books?id=npl
An issue with today's smartphones is that it's hard to pay attention to a screen when being on the move. By giving information to the user through sound, the need to look at a screen would be eliminated and hence lets the user focus on their surroundings - a vital part when running. XXX also mentions that sound is attention grabbing. They explain this by saying that one can choose not to watch something, but it's more difficult to choose not to hear something.

While using sounds enables the user to focus on its surroundings, it has its limitations. In XXX (1994), Kramer presented a few difficulties with using sound to interact with the user. According to him, one of the biggest problems is the low resolution - meaning that it's hard for a user to perceive small differences in sounds, such as changes in sound volume and panoration. Another problem mentioned in the book is the annoyance that can arise because of the attention grabbing properties previously mentioned. If the user dislikes the sound being used it's hard to ignore and will therefore bring annoyance.

2.2. Sensors

In this project it was important to determine the *user direction*, the direction in which user is facing. 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. Using a combination of multiple sensors however, a sufficiently accurate user direction can be calculated.

Strengths and weaknesses with different sensors.

2.2.1. GPS

The Global Positioning System - GPS is used to determine the global coordinates of the device in latitudes and longitudes through satellites. This is utilized for determining the user position relative to the virtual objects in the game, such as coins and monsters. Furthermore, by keeping track of previous GPS locations, it is getting user coordinates as well as the user bearing

2.2.2. Accelerometer

The accelerometer gives the linear acceleration of device ["An14].

These values can be used together with magnetic field sensor to obtain device orientation, or together with the gyroscope to obtain rotation in a global coordinate system. 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.

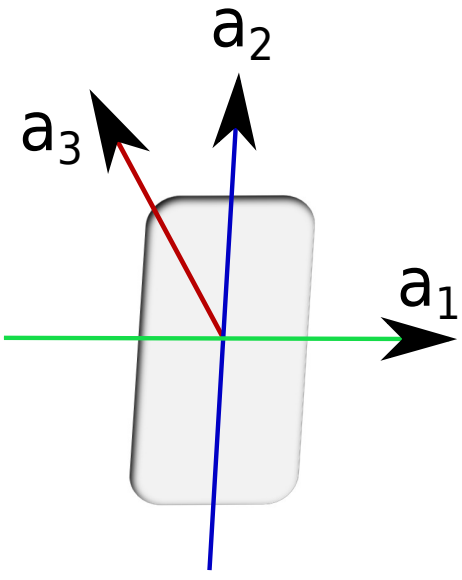

Figure 2.1.: Coordinate system of the accelerometer

2.2.3. Magnetic field

Determines the magnetic field strength. Can be used together with accelerometer obtain device orientation.

2.2.4. Gyroscope

Gives the rotational speed of the device around each of its axises ["An14]. 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 accelerometer 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. 2.2), g_2 is the rotation around the y-axis (blue in Fig. 2.2) and g_3 is the rotation around the z-axis (red in Fig. 2.2).

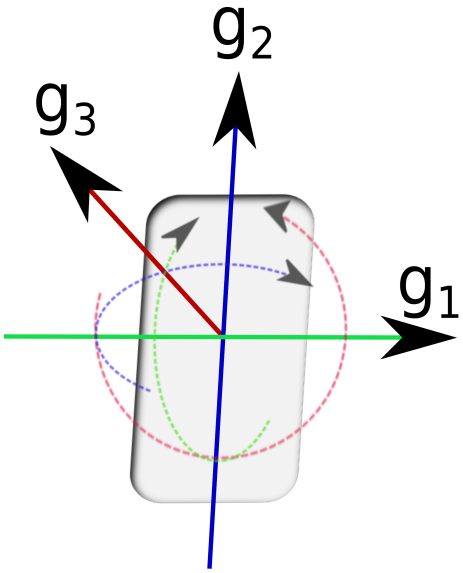

Figure 2.2.: Coordinate system of the accelerometer

2.2.5. Sensor fusion

Just general info about what sensor fusion is. Combining multiple sensors to get a better accuracy then one would with only one.

2.3. Random route generation algorithm

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. It can for example not be placed in a lake or a building but instead on a road or trail.

There are several methods to generate a location that is reachable. One way is to after a randomly generated location is acquired this location is evaluated if it lies in an appropriate environment by extracting the color this location has on a map. By classifying all the colors on a map by for example blue is equal to water, light green is equal to grass, and so on one can compare the color on a location and then classify it thereafter. If the color 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. In order to implement this an accurate map is required that i linked to some kind of coordinate system. It must also be possible to extract the color of the random location.

Another approach is to use Google maps and the function directions. As before a random location is generated. The a request to Google direction is sent with the walking direction from the users location to the random location. The answer from Google gives the direction to the closest reachable point to the location. This closest point is then extracted and is used as the random location. {https://developers.google.com/maps/documentation/android/}

2.4. Database

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 (databaseboken , kap 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 model

activities, fragments, etc

2.6. Design patterns

2.6.1. Android design patterns

Fråga: Sätta referensen här eller i slutet av stycket?

Android standard design patterns are used where applicable in the app. A great advantage of using design patterns is that the app becomes consistent to other apps on the Android platform.[Leh13]{Smashing s 288} This consistency creates a more comfortable user experience since the user knows what can be done with the present design components.

2.6.1.1. Action Bar pattern

One of the most defining design components in Android is the Action Bar pattern. {Smashing 292} As the name suggests it is a bar containing components which can perform actions to the app. (see Fig X.X). Since it has been around for quite long it is one of the most recognizable patterns.{Smashing 292}

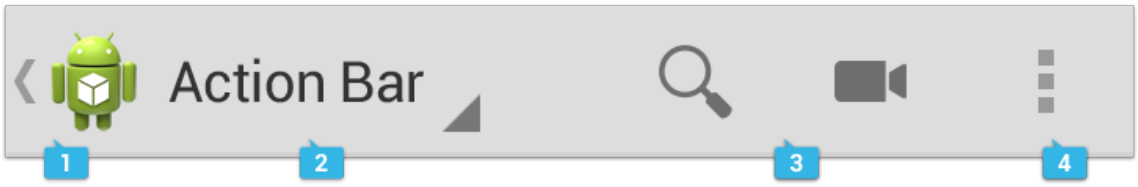

Figure 2.3.: Figure of an Action Bar and its components App Icon (1), View control (2), Action Buttons (3) and Action overflow menu (4)

{http://developer.android.com/design/patterns/actionbar.html}

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

råga: Får
nan göra
ähär med
änvisning
ed siffror?

Action Buttons as in (3) in Fig X.X. 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. {Smashing 299} 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 cool feature. {http://developer.android.com/design/patterns/actionbar.html}

The actions that, according to the above criterias 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 in the picture marked with three dots Fig x.x (4). 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. {http://developer.android.com/design/patterns/actionbar.html}

An advantage with the Action Bar pattern is that it displays the app logo(1) which establishes the identity of the app.{http://developer.android.com/design/patterns/actionbar.html} 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 Fig X.X) or a home button taking the user to the home view.{http://developer.android.com/design/pattern The button of the app logo may also be used to open a side menu called navigation drawer, which will be covered in section 2.6.2) with more navigation options. {Smashing 293}

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

2.6.1.2. Navigation Drawer

Google suggests that actions should be put to the right and navigations to the left. However there is a risk that putting navigational options into the action bar, as mentioned above, 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 Fig X.X.. {https://developer.android.com/design/patterns/navigation-drawer.html}

When a view is provided with a navigation drawer it is important to let the user know that by putting the navigation drawer indicator of three stripes to the left of the app icon as in Fig X.X. {https://developer.android.com/design/patterns/navigation-drawer.html}

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. The Dashboard pattern used to be one of Google's official

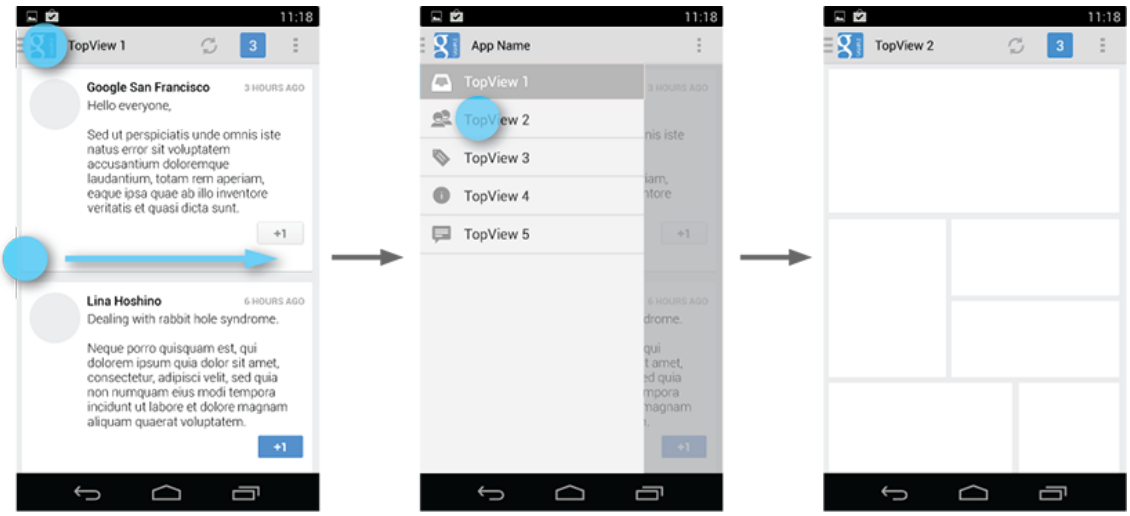

Figure 2.4.: Figure indicating how the navigation drawer may be accesed either by swiping from the left or by pressing the app icon.

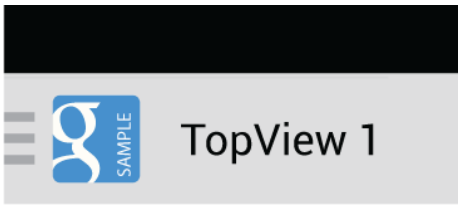

Figure 2.5.: Figure showing an app icon provided with the navigation drawer indicator

recommendations but has been more or less replaced by Navigation Drawer since it provides more direct access throughout an application. The avoidance of the dashboard pattern is especially effective at an application's startup since the user 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 eliminating redundant clicks back and forth from a dashboard. {Smashing, s332}

2.6.1.3. 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. {Smashing 322}

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 unlimited number of tabs. {Smashing 324}

2.6.2. Standard Design patterns

2.6.2.1. Prominent Done button

2.6.2.2. Carousel pattern

3. Implementation

3.1. Sound

The 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 supports 3D audio operations at the time this is written (NDK beginner's guide reference). To implement binaural audio, one needs to access operations that aren't covered in the SDK.

3.1.1. Native Development Kit (NDK)

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). However, it enables developers to use operations that aren't accessible using only Java - such as handling 3D audio.

3.1.1.1. OpenSL ES

OpenSL ES is a cross-platform API that enables audio functionality in native applications on multimedia devices. It's been part of Android's native development kit (NDK) since 2.3 and is divided into three different profiles - game, music and phone {NDK Beginner guide}. For a phone to take use of the functions provided by such a profile, it needs to have an appropriate compilation. The only profile enabling functions that lets the developer control audio in the spatial space is the game mode. Sadly, there is not a lot of 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 3D test application using OpenSL ES and its features, testing it on our own phones* did indeed result in a SL_RESULT_FEATURE_UNSUPPORTED 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

3.1.1.2. OpenAL

VÄLDIGT OKLAR TEXT

OpenAL is an application programming interface (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.

3.2. Sensors

Our implementation of sensor fusion.

Converting gyroscope coordinates into global coordinate system using accelerometer:

Given that $g = \begin{Bmatrix} g_x \\ g_y \\ g_z \end{Bmatrix}$ is the raw gyroscope readings from the device.

$g' = -(g_x * a_x) + (g_y * a_y) - (g_z * a_z)$

Gyro → Global coordinates → delta angle → delta angle * time constant→ differ from gpsBearing with delta angle * constant (longer time = smaller angle)

3.3. Random route generation algorithm

Har inte refererat bilderna till texten rätt än!

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. It 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 and its routing features.

First of all a random location is generated in a predefined distance from your location that is set by the GPS. The location can be generated in a circle around you with the radius equals to the distance, see figure random 1. d{http://gis.stackexchange.com/questions/25877/ho to-generate-random-locations-nearby-my-location}. [ato13]First two random numbers, a and b, with a value between 0 and 1 is generated. Then the constant r calculated. r is the radius of which the location can be generated within converted from meters into the Geographic coordinate system. By multiplying the distance r with the random number a, a random distance w, is calculated. On line number 8 the random number b is multiplied by 2*pi. 2*pi is a complete revolution in the unit circle and by multiplying it by a random number a random angle, t, from the unit circle. To get the distance to be added to the location of the user in terms of longitude, x, cosine is taken on the variable t and then multiplied by the distance w. The same is done to get the latitude, y, but with sine on the variable t instead of cosine.

Since the longitude coordinates is shrinking the further away from the equator the user i located a method the counteract this is implemented. By taking cosine of the latitude of the user location a factor of how far from the equator the user is located. By dividing the calculated longitude to be added, x, with the factor a new variable, x_corrected, is acquired. Without this correction the possible location of the random location would not lie in a circle but instead inside an ellipse. As seen on line 13-14 the calculated latitude, y, and x_corrected, is added to the location of the user and a random location is acquired.

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	$b = random[0..1]$
6	$r = radiusInMeters/constantToConvertFromMetersToLatLong$
7	$w = r$
8	$t = 2 * \pi * b$
9	$x = w * cos(t)$
10	$y = w * sin(t)$
11	$xCorrected = x/cos(startLatitide)$
12	$locationOfRandomLatitude = startLatitude + y$
14	$locationOfRandomLongitude = startLongitude + xCorrected$

Table 3.1.: To fill...

To avoid that the location is generated to close to you it is instead generated within a range around the desired radius, see figure random 2. This is done by generate a location randomly around you within a circle of radius r. Then the bearing between you and the recently generated location is calculated. From the bearing the ratio between the catheti is calculated and multiplied with the desired distance d. From

this the distance of how much to add to the latitude and longitude is given and added to the current longitude and latitude. To make this randomly generated location reachable a request of directions from your current location to the new location is send to Google Maps. The answer from Google maps directions is a list of locations that you have to run by in order to reach the destination. It is a big chance that the generated location is not on the road but instead in the forest or in a lake. But by taking out the last location that is given by Google maps direction you find a location that is as close as possible to the generated location but it is reachable. This location is then set as a checkpoint.

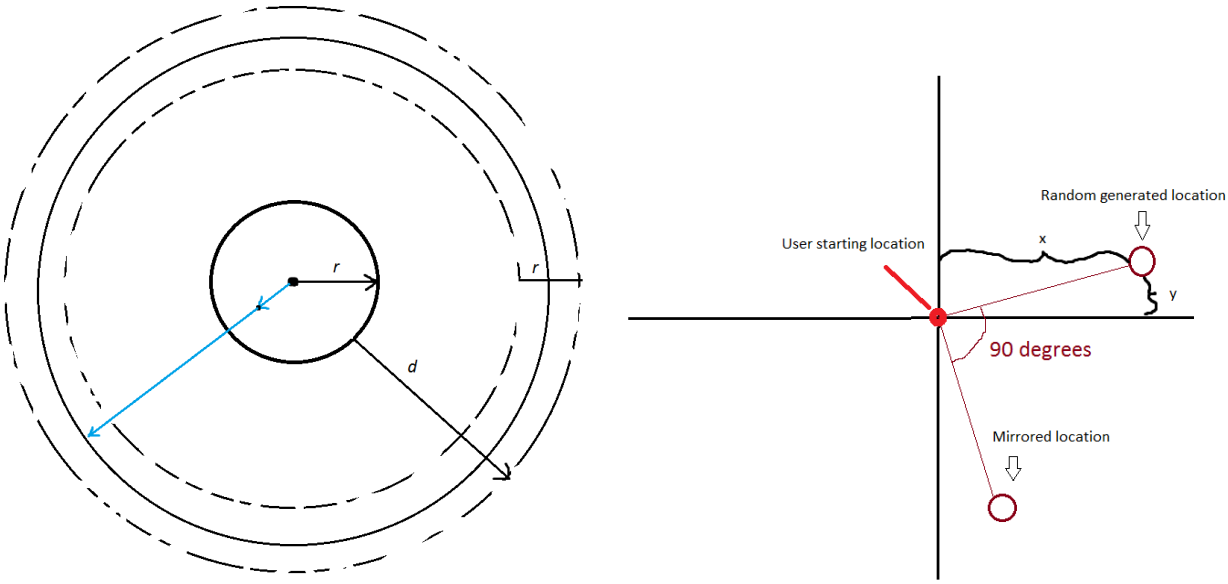

Figure 3.1.: Random...

To end up on the same location as you start when out are out running a routing algorithm was implemented. It creates a route based on the first generated random location. To create a route that not just go to one point and back the algorithm tries to make a route that is a circle of checkpoints and it end up on the same location as the user started on. This is done by first generating the a random location. This location is then mirrored over the geographic coordinate system but with the origo on the starting location of the user (Figure random 4). By taking the difference in the latitude, y, and longitude, x, between the users location and the newly generated location and adding/subtract them to the inverted axis (the difference in x direction is added to the y axis and the difference in y direction is added to the x axis) a mirrored location is achieved.

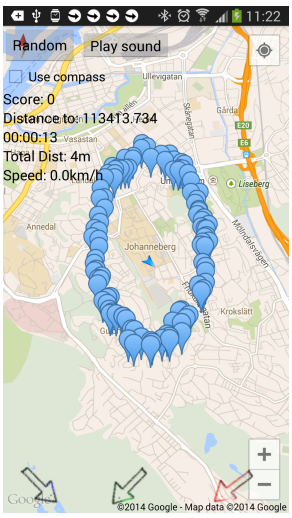

Figure 3.2.: Random... maps

3.4. Database

When exercising it is very important to keep track of your improvements over a larger time span. During the users running route we accumulate some information such as the time and distance of the route as well as information about the game objectives, these data needs to be stored somewhere so that the user can access it and gain knowledge about their improvement. To accomplish this a database is required. The database that was implemented is an embedded database, which means that the data is stored locally on the users device.

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 ["SQ14].

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 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.

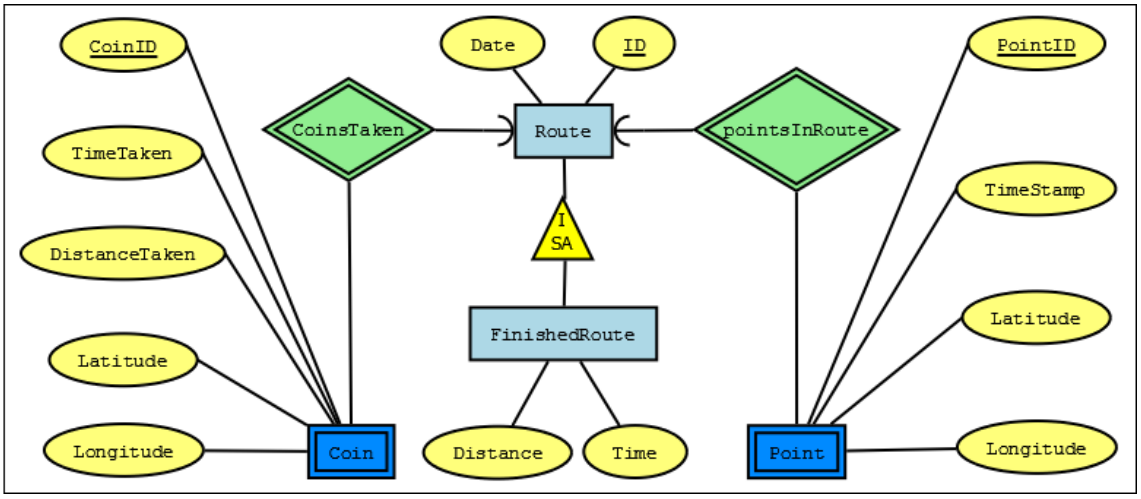

Figure 3.3.: The implemented database represented by an ER-diagram.

The Fig. 3.3 is the ER-Diagram of the implemented database, it is centered around the entity "Route", the route is basically the users running session, it contains a unique identifier and the date that the route started. 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 ([Wid09], kap 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.

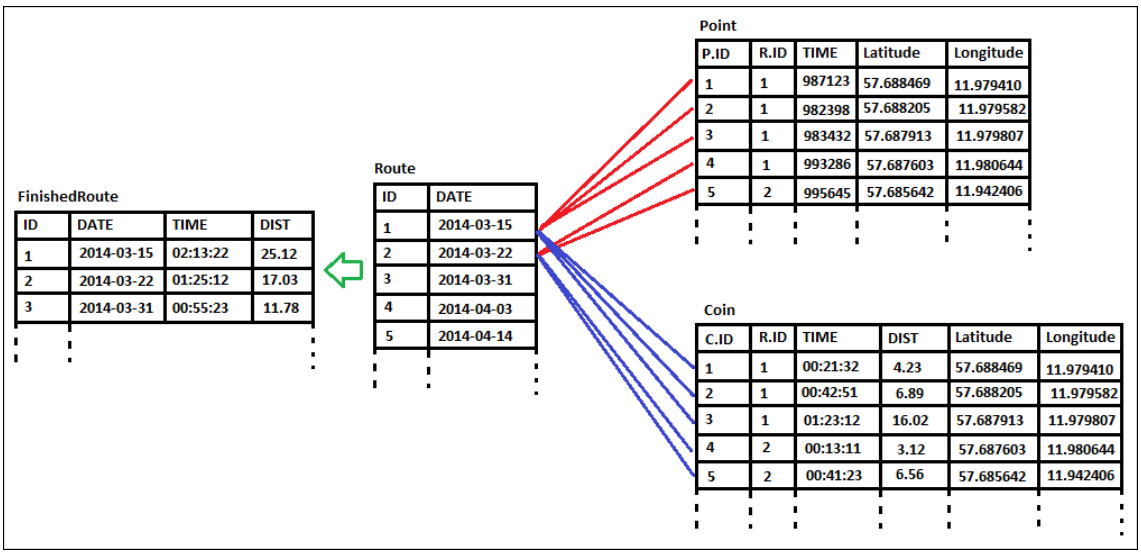

Figure 3.4.: Visualisation of the tables in the database.

Fig. 3.4 is a visualisation 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 accosiated with. This is how the weak relations that was represented with double borders in Fig. 3.3 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 to be designed in a an appealing way.

3.5.1. Establishment of requirements

As a starting point inspiration was taken from the apps Endomondo {https://play.google.com/store/
2014-03-28} and "Zombies, Run!" {https://play.google.com/store/apps/details?id=com.sixtostart.z
2014-03-28}.

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 what items were picked up during a run and the upgrading of the users basecamp. Also Endomondo is very detailed in its statistics while "Zombies, Run!" is not.

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

3.5.2.2. Main screen

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 long time like in games 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 makes it inefficient. A better solution was making the landing screen light weight and low in its demanding of resources. {Smashing s 350-351} Thereby the Run For Life-welcoming 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.

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/davidschre

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}

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. Usability evaluation

4.1. Observations

4.2. Interviews

5. Results

6. Discussion

7. Results

Acknowledgments

Thank you, thank you, thank you

A. Data from the usability evaluation

A.1. Overview

bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla roughness parameter R_a bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla, see [?].

A.2. The next section

Bibliography

["An14] "Android Open Source project". Android developers - motion sensors, April 2014.

[ato13] atok. How to generate random locations nearby my location?, 08 2013.

[Leh13] Juhani LehtimÃ€ki. *Smashing Android UI: Responsive User Interfaces and Design for Android Phones and Tablets.* John Wiley & Sons, Inc, 2013.

["SQ14] "SQLite". "SQLite", February 2014.

[Wid09] Hector Garcia-Molina Jeffrey D. Ullman Jennifer Widom. *"Database Systems - The Complete Book (2nd Edition)".* Prentice Hall, Upper Saddle River, New Jersey, February 2009.