Kari Sainio

Virtual Orienteering Game For Smartphones

Developing Proof of Concept

Helsinki Metropolia University of Applied Sciences

Master’s Degree

Information Technology

Master’s Thesis

17 September 2015

Author(s) Title Number of Pages Date	Kari Sainio Virtual Orienteering Game for Smartphones 68 pages 17 September 2015
Degree	Master’s Degree
Degree Programme	Information Technology
Instructor(s)	Ville Jääskeläinen, Principle Lecturer

Orienteering is a relatively famous sports activity in Nordic countries. Today technology provides attractive means for transition from traditional orienteering to virtual orienteering. Virtual orienteering can be thought as orienteering without using a traditional printed map and compass. On the other hand smartphones contain technology for implementing virtual orienteering.

There are different smartphone platforms and types, making application development challenging because several code bases and variations are needed. One way to avoid developing an application to several platforms is to use platform agnostic application frameworks. Game engine is one example of such application framework, which supports the deployment of the application to different platforms. A game engine can be used for implementing a virtual orienteering game application. Game engine, such as Cocos2d-x is one famous open source based engine that supports application development to several smartphone platforms.

The purpose of this study was to study available technology for virtual orienteering and create an implementation of proof of concept. The aim for the proof of concept was to demonstrate that Cocos2d-x game engine could be used to implement a virtual orienteering game deployed to several different smartphone platforms.

The study goes through the theory of orienteering and virtual orienteering, including description of restrictions and usage of traditional orienteering maps and how publicly available online maps could be utilized. Satellite navigation system technology and its usage are also described. Device sensor technology in the form of magnetometer is also covered. General game development principles and requirements for the proof of concept game are discussed, an example and simplified source code snippets are provided. Information on how necessary adaptations could be made to Cocos2d-x to support iOS and Android systems is also provided.

A proof of concept game of virtual orienteering was implemented successfully. The game demonstrated that Cocos2d-x is a valid choice for developing virtual orienteering or other location based applications. It was also shown that Cocos2d-x adequately supports different screen sizes and different platforms.

Keywords	Orienteering, Cocos2d-x, iOS, Android, GNSS, Magnetometer, Compass, Map

Contents

Abstract

Table of Contents

List of Figures/Tables

Abbreviations

1 Introduction 1

1.1 Background 1

1.2 Technology Challenge 2

1.3 Objective 3

1.4 Research Method 3

2 Definition of Orienteering 5

2.1 Classic Orienteering 5

2.2 Virtual Orienteering 7

2.3 Summary 9

3 Virtual Orienteering Technology 10

3.1 Map Technology 10

3.1.1 Online Web Maps 12

3.1.1 Orienteering Maps 16

3.1.2 Artificial Maps 17

3.2 Electronic Compass 18

3.3 Location Technology 20

3.4 Summary 22

4 Orienteering Game Development Principles 24

4.1 Instructional Design 24

4.2 Gameplay Design 25

4.3 Summary 26

5 Cocos2d-x Game Engine 27

5.1 Cocos2d-x 27

5.2 C++ with Cocos2d-x 28

5.3 Structure of Cocos2d-x Game 29

5.4 Limitations of Cocos2d-x as to Virtual Orienteering 31

5.5 Apple iOS Platform 33
5.6 Android Platform 35
5.7 Design Principle for Missing APIs and Features 37
5.8 Summary 37

6 Implementation of Virtual Orienteering Game 38

6.1 Requirements for Proof of Concept 38
6.2 Development Environment 39
6.3 Device Variability 39
6.4 Support for Different Screen Resolutions 40
6.5 Game Menu System 43
6.6 Map Implementation 44
6.7 Orienteering Track Implementation 49
6.8 Using Game Engine Physics 50
6.9 Location Implementation 52
6.9.1 iOS Location API 52
6.9.2 Android Location API 53
6.10 Compass Implementation 56
6.10.1 iOS 57
6.10.2 Android 57
6.11 Summary 58

7 Results and Analysis 61

8 Conclusions 64

References 66

List of Figures/Tables/Listing examples

Table of figures

Figure 1. Sprint orienteering map (orienteering.org) 6
Figure 2. Mercator projection (Microsoft, 2015:Bing Map) 12
Figure 3. Tile Matrix Set (OSC, 2010:10) 14
Figure 4. Artificial game map created with Tiled (Tiled: 2015) 18
Figure 5. Android device orientation in natural portrait orientation 19
Figure 6. The principle of satellite navigation (Mistra, Enge, 2009:23) 21
Figure 7. iOS architecture (Apple Developer, 2015) 34
Figure 8. Android System Architecture (Android, 2015) 36
Figure 9. Different scenes of the game 43
Figure 10. The placement of map tiles on screen 48
Figure 11. A fan orienteering track 49
Figure 12. Screenhot of the map scene 62

Table of tables

Table 1. Virtual orienteering games (IOF, 2015: Software for Orienteering) 7
Table 2. Mobile Virtual Orienteering Games (IOF, 2015) 8
Table 3. Supported device resolutions. 40

Table of listings

Listing 1. Selecting resolution (Hussain et al, 2014) 42
Listing 2. Creating game menu (Hussain et al, 2014) 44
Listing 3. Fetching tile from tile server 46
Listing 4. An example of HTTP request 47
Listing 5. Calculating x and y coordinates of map tile 48
Listing 6. Calculating x and y coordinates from latitude and longitude 50
Listing 7. Drawing a circle 51
Listing 8. Applying physics engine 51
Listing 9. Reading location in iOS system 53
Listing 10. Android location manager 54
Listing 11. An example of JNI interface 55
Listing 12. Reading location in Android 56
Listing 13. An example of reading Android sensor 58

Abbreviations

ADK	Android Development Kit
A-GPS	Assisted GPS
ART	Android Run-time
DPI	Dots Per Inch
GLONASS	Globalnaya Navigazionnaya Sputnikovaya Sistema
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GIS	Geographical Information
HAL	Hardware Abstraction Layer
HTTP	Hyper Text Transfer Protocol
HUD	Heads Up Display
IDE	Integrated Development Environment
JNI	Java Native Interface
KML	Keyhole Mark-up Language
LCD	Liquid Chrystal Display
MEO	Medium Earth Orbit
NDK	Native Development Kit
NMEA	National Marine Electronics Association
WLAN	Wireless Local Area Network
WMTS	Web Map Tile System
XML	Extensible Mark-up Language

1 Introduction

Game application category is one of the biggest application categories for mobile devices and the number of sports applications in smartphones is growing. Both application categories utilize different environmental censor technology such as device orientation and magnetometer. A suitable technology can be found as built-in in today’s smartphones. Combining sports and games can create an interesting area for application development where virtual orienteering games can be seen as one example. GNSS (Global Navigation Satellite System) based location system has become one of the most important features in mobile devices by enabling car navigation. Also a magnetic compass censor starts to be common in even mid-range smartphones. For virtual orienteering game development a combination of these two technologies provides a good playground.

A similar user experience can be provided to different smartphone users with different models by doing a multiplatform application. Typically developing an application for a smartphone is done using its own programming language (e.g. Apple having Objective-C/Swing, Android having Java and Windows Phone having C#). Application developer needs to study platform dependent frameworks and user interface design. Each system requires its own code base, which makes the maintenance of the software more difficult. Application developers need to focus considerably on a certain platform and this may lead to a big group of dedicated application developers because each platform may need its own developers. This might not be a competitive solution after some time and might delay time-to-market for the application release because the same application cannot be distributed at the same time to different platforms.

1.1 Background

Nowadays Smartphones and tablets play an important role by providing platform for various applications. Sports applications have evolved a lot and they typically utilize different kind of censor functions that are used to provide feedback to the user about his performance. Censors such as 3D acceleration and location awareness are used in the applications in this category. Among the sports applications orienteering can be seen as one example. Basic elements of classic orienteering consist of the combined usage of a compass and a map. Checkpoints placed and found in nature and their markings on the map form an orienteering track. Orienteering can also be worked out as virtual where checkpoints do not physically exist in nature but on the screen of a device. For this reason a smartphone provides a good platform for developing virtual orienteering games with the flavour of sport and fun.

OrientGame (http://www.orientgames.com) is a mobile orienteering application that has been developed in the past for virtual orienteering. It has been developed for each platform (iPhone, Android and Windows Phone) separately using their own native frameworks and programming languages. The first versions of these applications have been developed as outcomes of Metropolia mobile programming courses. One of the problems in the past development is the different user experience due to different kind of user interfaces provided. Also several code bases are required to maintain the application.

Several cross-platform development tools can be used for smartphone application development. One genre of such tools is mobile game engines that provide a way for creating games in a fast pace that can be deployed simultaneously to several major platforms with a relatively low adaptation work. Adapting a game engine to suit a virtual orienteering game creates an interesting challenge to study.

1.2 Technology Challenge

The technical challenge was to find the way how a famous cross platform mobile game engine Cocos2d-x could be used for developing a virtual orienteering game.

Cocos2d-x itself does not provide necessary support for virtual orienteering such as acquiring user location, reading compass information and providing a suitable map. These features can be seen the basic functions in all orienteering games. In addition, portability of the same code base and user interface for several smartphone platforms is beneficial.

Cocos2d-x game engine provides good capabilities for developing graphics intensive and user interactive applications. When comparing to the major existing mobile software platforms such as Android they already provide good built-in and non-agnostic frameworks for developing location aware and map utilizing applications per se. However, these built-in functions and frameworks do not work directly with other platforms such as iOS and vice versa.

1.3 Objective

This study evaluated how virtual orienteering game could be developed using a general game engine. It will show an answer the following question:

How Cocos2d-x game engine can be used for developing a multiplatform virtual orienteering game application?

The outcome of this study contains theory and technology of virtual orienteering and an implementation of a proof of concept game. The actual proof of concept game was developed with Cocos2d-x game engine that can be used with two different platforms: Apple iOS and Google Android. The game application itself is a small mini game that proofs the functionality and portability of Cocos2d-x game and makes it possible to further develop the game or its variations.

1.4 Research Method

The project progressed following a normal software development process and it consists of three main parts. First, introduction to virtual orienteering is given. Secondly, the applicability of Cocos2d-x game engine for virtual orienteering had been explored and suggestions for necessary add-ons presented. Thirdly, an application has been created that is suitable for virtual orienteering and demonstrates the suitability of Coco2d-x usage. The step-by-step project path included the following stages:

1. Virtual Orienteering Technology evaluation and scoping
2. Cocos2d-x game engine feasibility
3. Implementation of virtual orienteering

This study contains the following chapters. Chapter 2 introduces orienteering in general. Chapter 3 describes technologies that can be used for virtual orienteering. Chapter 4 specifies the requirements and design principles for the virtual orienteering game. Chapter 5 analyses the suitability of Cocos2d-x and modifications needed. Chapter 6 describes the implementation and Chapter 7 evaluates the success of the product/software. Chapter 8 analyses the next steps and future development ideas.

2 Definition of Orienteering

Orienteering as sports has been first practiced in military (IOF, About Orienteering) and in Finland it has been a hobby and sport activity for the last 60 years. From the beginning its objective has been to complete an orienteering track as fast as possible using a map and a compass. The orienteering track is drawn on the map and specific flags indicating checkpoints on the map are placed on terrain. (Nikulainen et al, 1995: 2-1, translated from Finnish). International Orienteering Federations states:

> Orienteering is a sport that combines both a physical and a mental element. The basic idea in orienteering is to proceed from course start to finish by visiting a number of control points in a predetermined order with the help of map and compass. In order to choose the best possible route, orienteers look at the characteristics of the terrain, and the winner is determined by the fastest time to complete the course. What is unique to orienteering is that an orienteer must navigate and make quick decisions while running at high speed. (IOF, 2015: About Orienteering)

Orienteering is nowadays a wide spreading sport having International Orienteering Federation consisting six regions containing 79 member federations (IOF, 2015: About IOF). World championship competitions are held every year. In addition to the competition, orienteering is also performed as hobby without competition and several different forms exist.

2.1 Classic Orienteering

Classic orienteering can be thought as progressing in terrain by using an analogic compass with a printed map. In this sense foot-orienteering term is used. An oorienteering map contains two-dimensional projection of the terrain where the objects on the map have been drawn using specific graphical symbols. International Orienteering Foundation (IOF) has approved 1977 international instructions on how maps are formed and created (Häggman, Mäkinen, Oikarinen, 1980: 58, translated from Finnish). IOF maintains map information and the current specification is available at their site (IOF).

Orienteering can be divided into several types or forms. One of the types is classic orienteering and is performed in non-urban terrain such as that found in the woods of Nordic countries. The terrain for classic orienteering is typically challenging and maps used contain accurate and small details. The scale of the map is typically 1:10 000 i.e. one centimeter corresponds 100 m in the real terrain. Mastering classic orienteering requires a good map and corresponding terrain reading skills. In addition personal good shape helps moving on terrain fast and efficiently. Orienteering track lengths used in race orienteering vary but are typically somewhere around 4-9 km depending on the class. Different classed are used which are identified based on age, sex and in some cases ranking is used for elite athletics. Figure 1 illustrates a sprint orienteering map of Kristiansten festning.

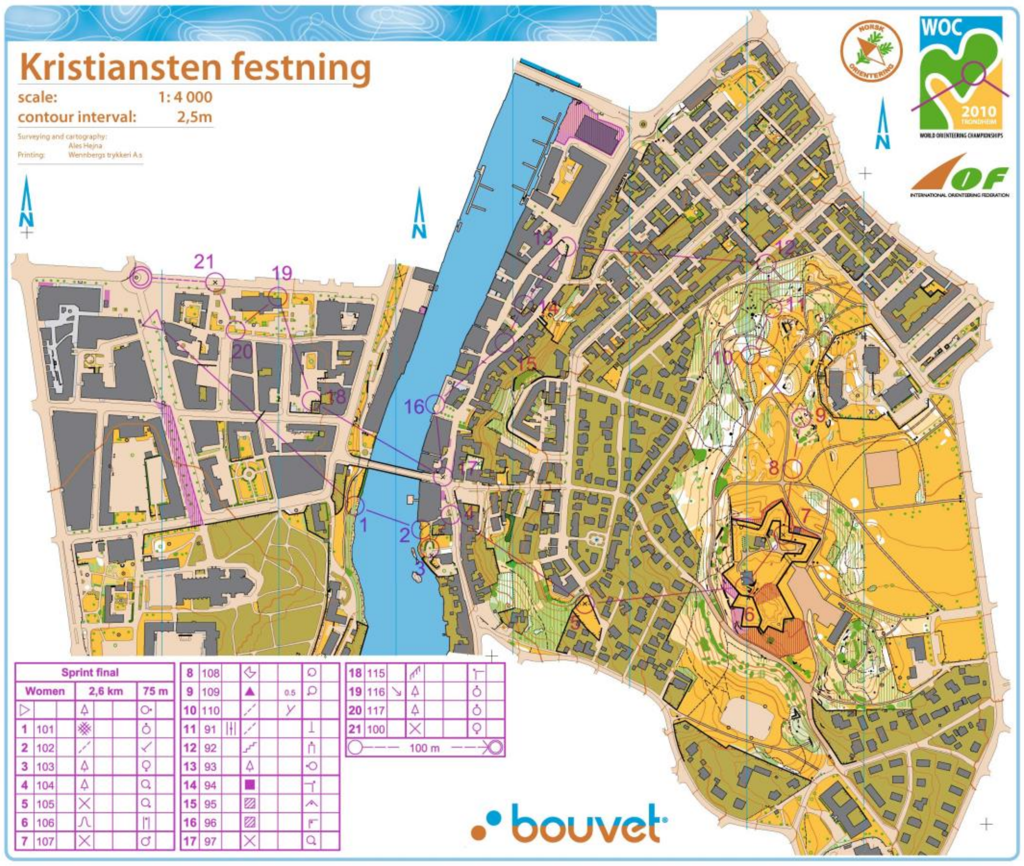

Kristiansten festning
scale: 1: 4 000
contour interval: 2,5m
Surveying and cartography: Ales Hejna
Printing: Wennbergs trykkeri A.s
WOC 2010 TRONDHEIM
WORLD ORIENTEERING CHAMPIONSHIPS
IOF
INTERNATIONAL ORIENTEERING FEDERATION
Sprint final
Women 2,6 km 75 m
100 m
bouvet

Figure 1. Sprint orienteering map (orienteering.org)

Sprint orienteering has become popular in the recent years. In the sprint orienteering maps are considered more urban and somewhat simple due to the urban terrain. Specially created maps are used in sprint orienteering that suite better for park kind of terrain found typically in central Europe. An example of a sprint type of orienteering map is given in Figure 1. Track lengths in sprint orienteering are typically a few kilometers. The idea behind the sprint orienteering is to bring orienteering visible to spectators and easier for cameras to follow. Running speed in the sprint orienteering is biased and is approaching a normal track and field running speed.

Other kinds of orienteering types or forms exist but common for all types is the need and capability to read terrain, map and direction and all sort of combination of these. In all cases, the idea is how to get from point A to point B in the most efficient and fastest way. In that sense an analogy with car or even aviation navigation can be considered. All orienteering is not just foot orienteering but also bicycle, skis and even cars can be used. Among these different variations exist and navigation on the sea or even aviation can also be thought of as a sort of orienteering.

2.2 Virtual Orienteering

Virtual orienteering needs a strict scope because it can be defined in several ways. For some people it can be orienteering without actually moving in real terrain and it includes just sitting on the sofa playing something. For others it can be augmented reality where an additional device can provide information about the terrain and environment (reference). In this context virtual orienteering is defined to be orienteering using a mobile device and its capabilities. Neither normal paper map nor additional compass is used.

Different implementations of the virtual orienteering exist and orienteering software available for personal computers and smart phones can be found, as listed in Table 1.

Name	Description
CatchingFeatures	Catching Features is an orienteering game you can play at home. Use it for rainy-day training or rest-day enjoyment. Several different modes of play are available.
GPSSeuranta	Complete solution for GPS tracking.
Orienteering Way	A real time orienteering game. Navigate the main hero through control points in 2D region using map and compass. You have to be smart and crafty to finish competitions fast enough to unlock next races.
pcM Orienteering Game	pcM Orienteering Game is a fun game that lets you navigate an orienteering course laid out on a map. It will test your map reading and route making skills, measuring your time based on your route choices.
Rasor lite	Online radio orienteering simulator
Virtual Orienteering	Virtual Orienteering is a game played outdoor inspired from Orienteering sport. The game concept is to use a GPS capable mobile device to replace the printed map and real control check-posts.

Table 1. Virtual orienteering games (IOF, 2015: Software for Orienteering)

Several applications can be found in the major application stores using the key word 'orienteering' (April 2015). In addition, using different key words one can find a countless number of navigation related applications. Table 2 lists a few virtual orienteering applications (Google Play, 2015):

Name	Description	Platform
Map-n-Compass	Virtual orienteering game with KML based tracks and maps possible.	iOS
OrientGame	Virtual Orienteering Game using build-in maps and Web based tools for creating tracks.	iOS, Android
Orienteering Companion	Virtual Orienteering with own imported maps.	iOS
SpotHunter	Finding spots on build in maps. XML based filed uploaded for the check points.	iOS
Orienteering for Beginners	Virtual 3D environment to learn orienteering.	Android
MOBO	Mobile Orienteering using maps and NFC for punching the checkpoints.	Android, iOS, Windows Phone
Virtual Orienteering	Mobile orienteering using map and GPS to locate checkpoints.	Android

Table 2. Mobile Virtual Orienteering Games (IOF, 2015)

This thesis will focus on the virtual orienteering using the public map of the smartphone and virtual checkpoints (i.e. non-existing checkpoint flags) placed on the device map. The map can be also virtual and not corresponding to any real world terrain. Suitable map technology used for virtual orienteering is covered later in the study. Furthermore other technologies found important in virtual orienteering are discussed.

2.3 Summary

Traditional orienteering uses a printed map and compass where on the contrast virtual orienteering does not. Virtual orienteering is not a new thing and there are existing applications available through application stores. Smart devices or smartphones have evolved onto the technical level that they can be used for virtual orienteering and the traditional orienteering map can be replaced by an electronic map.

3 Virtual Orienteering Technology

Before jumping to the implementation virtual orienteering requires several technological solutions and challenges to be solved. First, the most of the orienteering performed with the co-operation of map although exceptions exist. The quality of the map and used production technology has an important role. A good map can tell to the user accurate information about the terrain and help to navigate through terrain in the optimal way. Second, as compass indicates the right heading it can be used to place the map in the right direction. When the map quality is not good or accurate maintaining the right direction in terrain is vital. Without the map compass role becomes dominant as it can be used to maintain the right direction and different means to measure distance can be applied. Third, with the use of the current location technology can user location estimated accurately. This is important because it can be used to indicate where the user is on the map. These three technologies mentioned form the base for implementing a virtual orienteering game.

3.1 Map Technology

Today's orienteering relies heavily on good quality maps. The maps are designed in a way that the reader can interpreted the location and objects in terrain and can find corresponding match on the map. In general level map types can be divided into three most common types: Thematic, Topographic, and Cadastral. Thematic maps show specific topics and their geographic relationships and distributions. Weather forecast is one example of the thematic map type. Cadastral maps show how land is divided into real property. Topographic maps show physical characteristics of land in area (Harvey, 2008:13). A map suitable for orienteering is typically based on a topographic map. A topographic map describes land or terrain in a detailed manner. Physical characteristics of a topographic map contain curves describing the altitude of terrain, notable objects such as rocks, hills and buildings. Additionally other characteristics of land such as rivers, lakes and swamps are drawn.

International Orienteering Foundation has defined standard´s of how an orienteering map should be created and what graphical objects should be used to describe the terrain. This also helps map users to interpret the map in a correct way.

A map is 2-dimensional representation of 3-dimensional terrain where projections are used to make this conversion. They are used for transforming a 3-dimensional world globe to a 2-dimensional map and are abstractions of the earth's surface (Harvey, 2008:55). As several projection types exist such as Mercator, Sinusoidal, Equidistant Cylindrical and Azimuthal is the Mercator projection is one of the most used. It is suitable for small and large areas but only in navigation (Harvey, 2008:55). Some web map examples of Mercator projection usage are Google Maps and Microsoft Bing Map. Mercator preserves the shape and distance relationship of small areas (Harvey, 2008:55). Also a straight line in Mercator projection can be seen as constant compass bearing (Harvey, 2008:77). Mercator projection makes it relatively good for orienteering as it includes the good attributes. Microsoft also states that the shape of buildings in aerial imagery does not get distorted because Mercator projection preserves the shape of relatively small objects (Microsoft, 2015:Bing).

Mercator projector is a cylindrical projection, which means that north and south are always straight up and down, and west and east are always straight left and right.
In that sense this feature makes it fit well for orienteering because maps are the forms of quadruples. The Mercator projection goes to infinity at the poles and does not actually show the entire world (Microsoft, 2015:Bing). The maximum latitude shown is approximately 85.05 degrees (Microsoft, 2015:Bing). To visually understand how Mercator projection works it is easy to note that e.g. Finland looks as quite a big country in relation to others (Figure 2) because on map the area is shown wider when compared to e.g. the U.K.

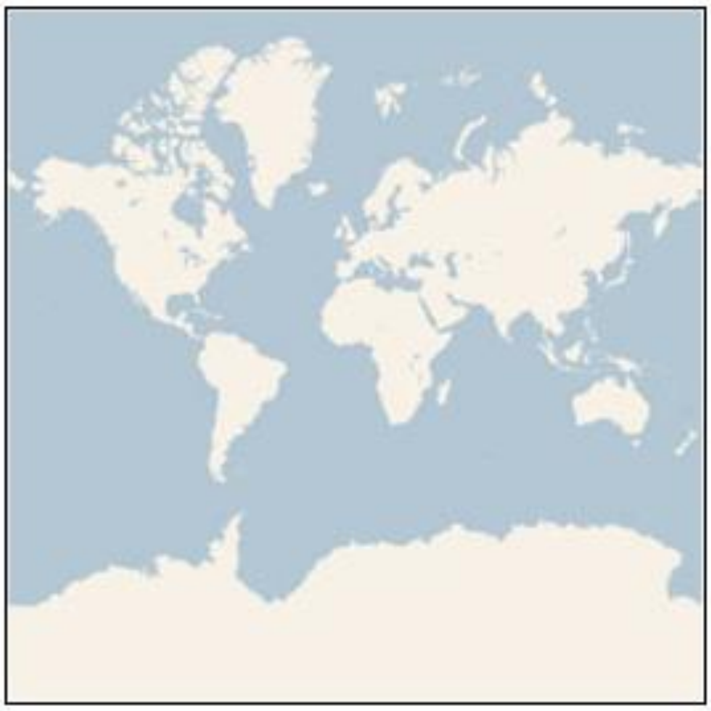


Figure 2. Mercator projection (Microsoft, 2015:Bing Map)

Map technology as such is a large topic and deeper analysis of map technology is left out of the scope of the present study. For smartphones and other connected devices there are several different map technologies that can be used. There are also online and offline maps that both have their pros and cons. The following chapters describe basic map technologies that can be considered for virtual orienteering.

3.1.1 Online Web Maps

Nowadays the Web is full of different kind of online maps. Their quality is typically not suitable for accurate orienteering. However, they form a good basis for a map covering large areas of the globe. The map information can be downloaded on demand and for the area of interest. This makes it attractive to be used in virtual and mobile orienteering because from the application perspective that can be used anywhere on the globe. Google Maps, Microsoft Bing Maps and Nokia HERE maps are examples of online Web maps suitable for navigation by car and tourism. Map information is publicly downloadable and can be used by agreeing with the license, which fee is typically free when the amount of usage of map information is kept low. Over the last years these online maps have been augmented with walking and bicycling route properties. Furthermore different Geographical Information (GIS) has been introduced e.g. rush hour indicators for certain roads and weather forecasts. More accurate online map information is available by online map provider OpenStreetMap. OpenStreetMap base its information for community activity where basically anyone can update information (OpenStreetMap: 2015). It is more detailed than e.g. HERE maps by having map details also for non-car users.

One common technology used to deliver online map information is to split a map to several tiles i.e. a map is constructed from one or combination of several tiles. Open Geospatial Consortium Inc. (2010) has standardized a way (Web Map Tile Service, WMTS) how a map tile service can be implemented. The idea behind the map tiles is that the graphics in a map have been constructed already in the server and the tiles are delivered as picture images. Images are either taken from a satellite picture or generated cartographic images from the terrain with different scales or zoom levels. Different zoom levels are provided so in case a user wants to dig into deeper details another image with higher accuracy is available. Because online web maps are typically taken from wide areas it makes sense that the map information is available online by demand and does not require a lot of storage space in the device where the map information is used. This makes the service usable through the whole world because only the area in interest can be downloaded to the device. Figure 3 shows how map tiles are constructed and how zoom level affects to the map scale level.

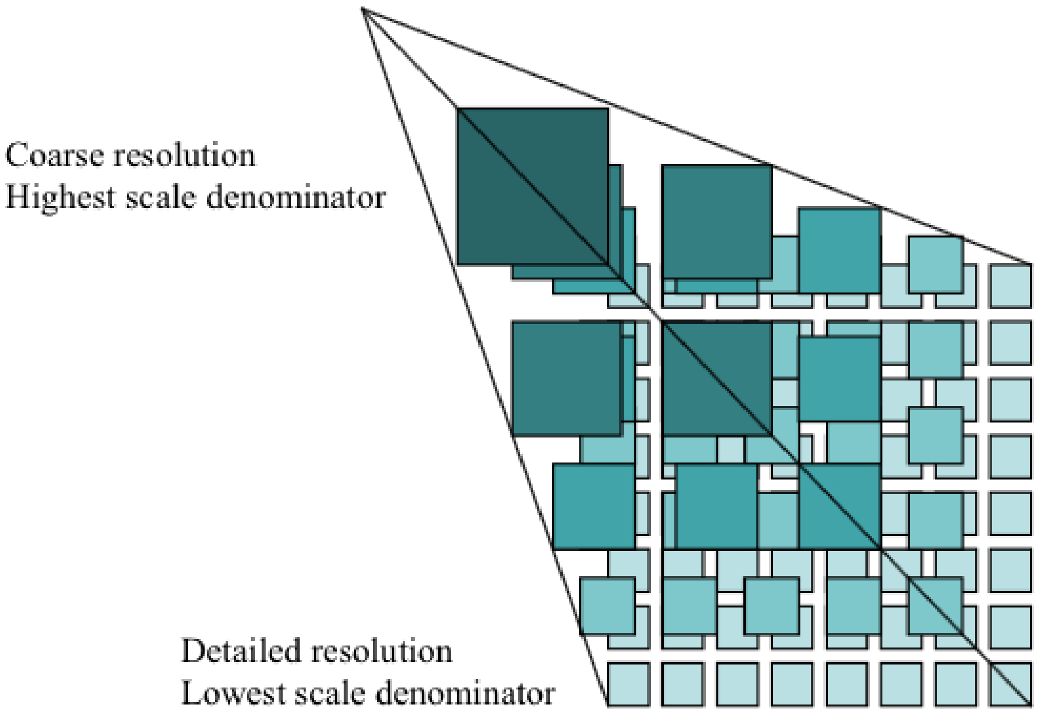


Figure 3. Tile Matrix Set (OSC, 2010:10)

Microsoft Live Map projections and zoom levels define a well-known scale. Level 0 allows representing the whole world in single 256 x 256 pixels. The next level represents the whole world in 2 x 2 tiles of 256 x 256 pixels. Furthermore, this continues to deeper level in powers of 2. (OSC, 2010:105). Online map providers such as Bing maps (based on Nokia HERE maps) and OpenStreeMap use similar scaling and zoom levels.

An example of an online tile map implementation can be found in Microsoft documentation. Microsoft Bing map uses pre-rendered 256 x 256 pixel map image tiles at many scales that can be retrieved and displayed fast (Microsoft, 2015:Bing). At the base level of 1 Microsoft Bing map can be displayed as 512 x 512 of total pixels tiled image (i.e. combination of 4 tiles) and going to level of 2 is the map having 1024 x 1024 of total pixels image (i.e. combination of 16 tiles) and so on and so forth. Geographic coordinates (latitude, longitude according to WGS 84 datum) can be converted to world pixel coordinates used by tile maps in the following formula (Microsoft, 2015:Bing):

```
sinLatitude = sin(latitude * pi/180)
pixelX = ((longitude + 180) / 360) * 256 * 2 level
```

```
pixelY = (0.5 – log((1 + sinLatitude) / (1 – sinLatitude))
/ (4 * pi)) * 256 * 2 level
map width = map height = 2 level tiles
```

As the Earth is not a perfect sphere but a bit oval the result of this formula is non-accurate but sufficient for the virtual orienteering application's purpose. Pixel x and y coordinates resolve to the nearest tile in the system. The given latitude or longitude does not necessary correspond the tile's top left most corner but some point of the individual tile. To find the actual world coordinate to be placed on the tiled map requires additional computation to find which tile pixel corresponds to the world coordinate.

Map tiles can be retrieved using HTTP Requests. Nokia HERE map tiles can be retrieved using the following HTTP Request:

http://l.aerial.maps.cit.api.here.com/maptile/2.1/maptile/newest/terrain.day/z/x/y/size/png8?app_id=XXXXX&app_code=YYYY";

Where parameters are:
l = load balancing; number of request 1…4
z = zoom level
x = requested tile x coordinate
y = requested tile y coordinate
size = tile size in pixel e.g. 256
app_id = individual registration code from Nokia HERE
app_code = individual registration code from Nokia HERE

Online maps can be adapted to virtual orienteering. Integration to a game can be done relatively easily because one can use necessary tiles based on latitude and longitude information. Online maps however do not specify a high detailed presentation of terrain but could be used for beginner level orienteering or in terrain, which is relatively simple like urban area. Calculating ground resolution of the tile map can be performed using the following method:

> The ground resolution indicates the distance on the ground that's represented by a single pixel in the map. For example, at a ground resolution of 10 meters/pixel, each pixel represents a ground distance of 10 meters. The ground resolution varies depending on the level of detail and the latitude at which it's measured. Using

an earth radius of 6378137 meters, the ground resolution (in meters per pixel) can be calculated as (Bing:2015):

ground resolution = cos(latitude * pi/180) * earth circumference / map width
= (cos(latitude * pi/180) * 2 * pi * 6378137 meters) / (256 * 2 level pixels)

In addition, Google maps provide an API where centre of the map tile can be explicitly defined using latitude and longitude information (Google Maps: 2015). Furthermore, tile size can be specified. In Google case a map can be constructed from one whole image and not as combination of fixed size tiles. Usage of Google maps API is easier because there is no need construct map from several tiles as whole map area can be fetch using one request. However there is limitation of image size so large areas cannot be fetched with one request but similar tiling mechanism is required.

3.1.1 Orienteering Maps

As orienteering is a sport where the intention is to complete a course of control points in the shortest possible time (IOF, 2009:1) a map plays important role. That is why an accurate and legible map is a reliable guide for the choice of the route (IOF, 2009:1). Because of the competition or tactical sport it is vital that the map shows the terrain in the most reliable way. All competitors need to interpret the map in a similar manner. International Orienteering Foundation (IOF) has standardized the method how orienteering maps can be produced. Whereas typical online maps show only major roads and buildings, an orienteering map provides a detailed topographic map. An orienteering map must also contain magnetic north lines and peripheral text is placed in a manner that helps orientate the map to north (IOF, 2009:2). Accuracy is important if a positioning system is used together with orienteering map (IOF, 2009:2). For this reason it is self-evident that an orienteering map would be the best solution for a virtual orienteering game, also. Orienteering maps are created for a special purpose such as competition. The coverage of orienteering maps is not 100% of the globe terrain and therefore these maps can be used in certain areas only. It should be noted that typical orienteering maps are formed on a very small surface of the Earth. This means that the difference between different projection types has less influence because Earth surface can be estimated as plane. Local maps can be formed by taking pictures of the area from above or by using precise laser scanning. This means that the projection comes quite automatically as the picture is taken or scanned from the visible terrain.

In Finland orienteering maps are copyright and basically each orienteering team or city manages their own terrain and produces maps of them. It is possible to create a map from one's own playground or neighbourhood. Guide provided by Lonka (Lonka: 2012) gives information about how this kind of a map can be created for easy orienteering. Creating a professional map is costly due the amount of work and tools to be used. In Finland individual maps are typically sold separately in different kind of occasions to cover the expenses and gain money for the orienteering team. A typical tool used for creating an orienteering map is software called OCAD. The OCAD is a commercial tool that can use basic low detailed map as the base and special orienteering symbols can be drawn on top of the map. To use this kind of orienteering map in the virtual orienteering game would require the capability to distribute and license map information to the virtual orienteering game user.

3.1.2 Artificial Maps

Various games based on construction of areas as such typically use artificial maps that are based on an imaginational environment that some game artist has worked on. This kind of map would provide an interesting way of creating a map for the virtual orienteering game. One example for game map creation is the Tiled application. Maps created with Tiled can be used with Cocos2d-x game projects. One example of this can be seen in Figure 4. The idea behind artificial game maps is typically to use ready bitmap symbols and replace these symbols on the map several times. The map area contains a sheet that describes object placements from which the game engine can then form a map on the fly during the progress of the game. One of the reasons for this is to reduce the needed memory on the device. Basically a presentation of real life map could be created in a similar manner. In that case the terrain would be drawn with standardized symbols.

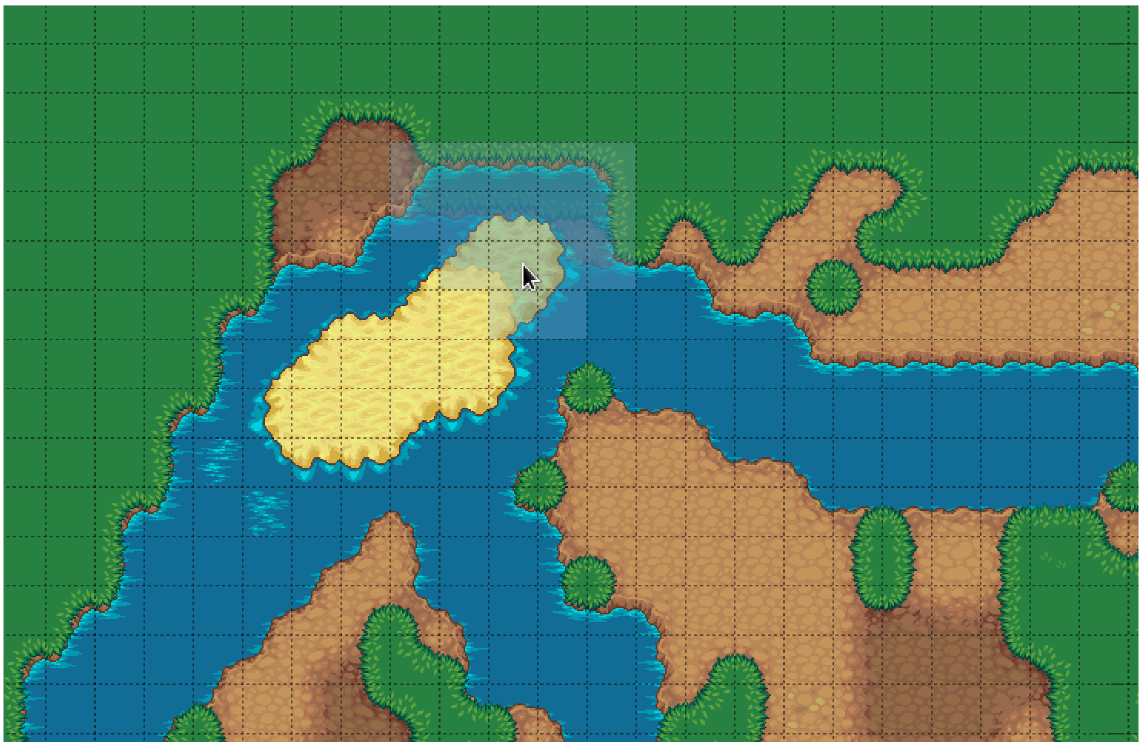


Figure 4. Artificial game map created with Tiled (Tiled: 2015)

Producing an artificial map is left out of the scope of this study but that would make an interesting and new way how a virtual orienteering game could be done.

3.2 Electronic Compass

Today's mid-range and high-range smartphones contain magnetic field censors. Electronic compass or magneto meter measures the strength of Earth's magnetic field and in smartphones they provide raw data and computed bearing (Schirme, Höpfner, 2015:12). Thus, it can be used for implementing an electric compass which could be furthermore utilized in virtual orienteering game. Different mobile platforms have different ways of providing magneto sensor readings to the applications.

Android is based on Linux technology and provides low-level API for finding out device direction in relation to Earth. This information can be used for implementing an electronic compass. Android sensors are virtual devices that provide data coming from a set of physical sensors. (Android Source:2015). Android platform provides Geomagnetic Field sensor and Un-calibrated Magnetometer (Android, 2015:Position Sensors). Android uses the combined information of geomagnetic field and orientation sensor to provide device orientation and bearing. Orientation sensor gives information about device orientation in three-dimensional coordinates. In addition geomagnetic field can be read per each coordinate axes separately. For actual magnetic north reading this information requires calculation that device heading information can be read correctly. Additional calculation is required of the device orientation and correcting values per each coordinate with the magnetic field sensor. Android reports the device orientation based on reference frame i.e. natural portrait orientation (Figure 5). However, it is possible to change this reference e.g. in cases that device is used in landscape mode and magnetic north is wanted to be pointed differently.

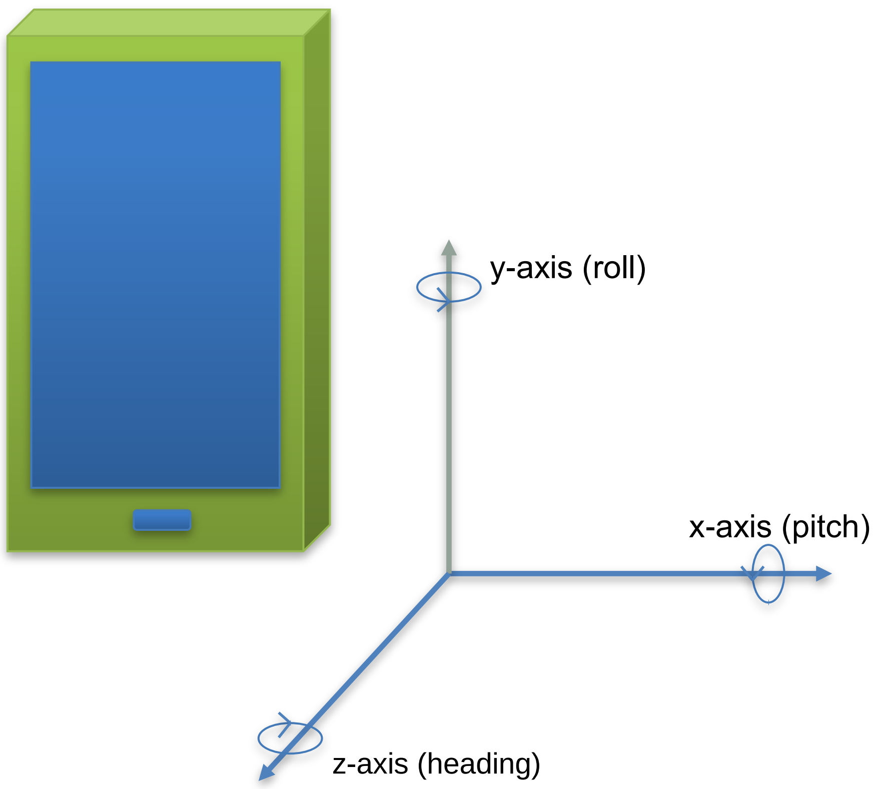


Figure 5. Android device orientation in natural portrait orientation

Apple iOS system supports two different ways for reading device direction. Magnetometer can tell in which direction device is pointing what is called as heading information. Location API can tell direction where device is moving and this is called course (Apple 2015:Location and Maps Programming Guide). In an orienteering game application the heading of the devices has a more important role, as it corresponds compass directly. Course information can be used for a tracking device for later purposes. Apple iOS makes reading of heading information much easier than with Android by providing a high abstraction level without direct API access to the magnetometer.

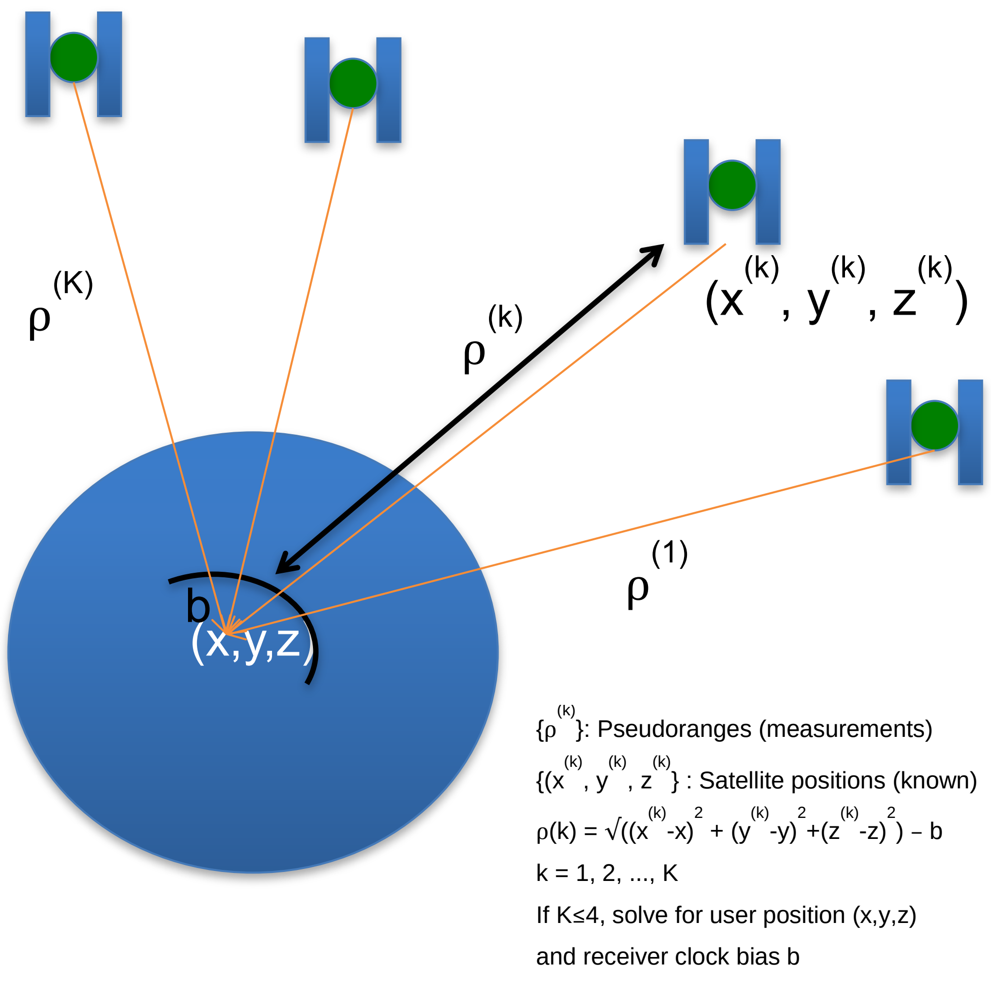
3.3 Location Technology

Accurate location technology is based on GNSS (Global Navigation Satellite System) although different other low accurate systems are used. These low accurate systems are typically based on the location of a cell tower or WLAN station and dependant on the area. Accurate location determination is needed for virtual punching when a checkpoint is found in the virtual orienteering as it forms the base for the orienteering. In virtual orienteering, however, the intention is not to use location technology to guide the player but to keep track of the smartphone and the user. This is needed for marking the visiting of the virtual checkpoint. Strictly speaking smartphone location technology is categorized as sensor technology and is closer to digital radio technology.

Today navigation and location determination is based on satellite navigation a.k.a. GNSS that covers the whole globe. The root of satellite navigation can be thought to have begun together with the space age which can be thought to have begun in 1957 when Sputnik I was launched by the Soviet Union (Mistra, Enge: 2009:19). This caused both United States and Soviet Union to allocate lots of resources to space race (Mistra, Enge:2009:19). The initial idea for the satellite navigation came when Sputnink I launched:

> The pattern of Doppler shifts in the signals transmitted by Sputnik I measured from a single ground station at a known position was enough to determine the satellite's orbit. Now the deduction: If the satellite orbit were known, a radio receiver measuring Doppler shifts could determine its position on the earth. (Mistra, Enge, 2009: 19)

GNSS and GPS satellite positioning system is based on a similar idea. GPS contains several satellites that have orbit with altitudes 5000 - 20.000 km and do 2 – 4 orbits per day. The orbit of its satellites is called Medium Earth Orbit (MEO). MEO constellation is 24 satellites in GPS and a set of same satellites are visible several hours per day. Location determination in GPS is based on three-dimensional coordinates for defining spatial position and accurate time. Synchronization of time between the satellite and receiver is important for measuring true transit time of the signals transmitted by the satellites. GPS satellites contain very accurate clock and receivers typically use inexpensive quartz oscillators. The bias in the receiver clock affects all observed satellite transit times equally. Pseudo ranges are called the measurements of the transmit times related to received time (receiver clock with bias). Because receiver bias is not known the time is the fourth unknown among the spatial coordinates. (Mistra, Enge, 2009:19-23) Introduce Figure 6 here.



$\{\rho^{(k)}\}$: Pseudoranges (measurements)

$\{(x^{(k)}, y^{(k)}, z^{(k)}\}$: Satellite positions (known)

$\rho(k) = \sqrt{((x^{(k)}-x)^2 + (y^{(k)}-y)^2 + (z^{(k)}-z)^2)} - b$

k = 1, 2, ..., K

If K≤4, solve for user position (x,y,z)

and receiver clock bias b

Figure 6. The principle of satellite navigation (Mistra, Enge, 2009:23)

In GPS and GNSS generally by knowing where the satellites are located up in the sky at a certain time one can measure accurate receiver’s location based on reversed information. Figure 6 shows how Pseudoranges are utilized when calculating the coordinates of the GNSS receiver. Among the measurements GPS requires specific radio and receiver technology that is left out of this study. GPS can provide position accuracy 5 m horizontal and 7.5 m vertical (Mistra, Eng, 2009:24). By using Differential GPS accuracy can be achieved even better. In Differential GPS receiver can obtain second (or multiple) signal from other GPS source placed elsewhere and by comparing two measurements can subtract errors caused by climatic and other errors occurring when signals are transmitted from a satellite to GPS receiver. Differential GPS is very accurate and can be used in overlay applications used in TV sports to highlight where different objects or persons locate.

In the worst case when a GPS receiver is started without knowledge of the previous location it will take at least 20 seconds to tune itself to the frequency of transmitting satellites. After this it will take 30 seconds to receive transmitted data from the satellite. The total time for the first satellite fix will take approximately 1 minute at minimum. If different kind of disturbances occur i.e. receiver going under tree or building then receiver needs to wait another 30 seconds for data retransmit. If the receiver has prior knowledge of its location is can utilize this knowledge and make assumption of the satellite locations and their initial transmitted data. (Van Diggelen, 2009:33)

Modern smartphones uses A-GPS (Assisted GPS) to find its location. A typical and non-assisted GPS receiver basically needs first to find the visible satellites by scanning the frequencies of all satellites. After finding the right satellite transmitting frequency the data is transferred slowly using low bit rate of 50 bps (bits per second). A-GPS technology improves this by speeding up by offering immediately the available satellite frequencies and transmitting data with higher speed than 50 bps. In this technology similar data than transmitted by satellite can be sent much faster to smartphone and its A-GPS receiver. This makes an A-GPS receiver to fast acquire necessary satellites because in the transmitted data is also send information about the visible satellites. Now an A-GPS receiver does not need to spend time for first finding the right satellites and then decode the information transmitted in slow bitrate. It can basically jump directly to listening to visible satellites and tune up to their frequency to listen to information to find a more accurate position. This makes location acquiring much faster and an A-GPS receiver (smartphone) can find its location in a few seconds. When thinking ofa virtual orienteering game and starting the game the player does not need to wait a long time for the game to locate itself.

3.4 Summary

For virtual orienteering different maps can be used and there are different technologies available. Topographic maps form the basis for the normal orienteering maps. However, their availability is limited and typically copyright exists. Furthermore, online maps available through the Internet provide basically free of charge usage and full globe coverage making them ideal to be used in virtual orienteering and for demonstrating the concept. On the other hand, to have more gaming experience artificial maps would provide a more interesting way to perform virtual orienteering.

As magnetometers start to be common in almost all range smart phones it is quite easy to use them in virtual orienteering and have them play the role of the compass. A magnetometer may not be as reliable as an analogical compass but gives an interesting demonstration to be used in a virtual orienteering game.

In practise all modern smartphones have some sort of capability to sense location. GNSS provides good accuracy to locate the device and to make possible to use virtual checkpoints that are triggered when the device is in a specified area anywhere on the globe. GNSS is also very sensitive technology and it should be highlighted that its accuracy depends on several factors. These factors are e.g. being covered in the woods or locating near a high building in city area, not forgetting the quality of the receiver.

4 Orienteering Game Development Principles

For a proof of concept the first target was to create a virtual orienteering game that concentrates utilizing the capabilities of Cocos2d-x game engine and secondly to find out how platform dependent censors can be integrated. A commercially playable finished and polished game is left out of the scope of this study. This sets certain conditions that may be different when comparing to typical game development i.e. creating a story, script and other design and so forth and so on. Typical game development and design is not always straightforward. Game design can be though as building a recipe and selecting ingredients. Scott refers to this as making chili (Scott, 2014:18): to make a game interesting there is not always a need to make good graphics but playability is important. A good quality of ingredients helps but at the end of the day it is a matter of designing an interesting game.

Designing a virtual orienteering game for a mobile device has certain challenges. On the one hand it is a kind of serious navigation application. On the second hand it is a fun game that is meant for people who are not necessarily so interested in orienteering itself. A serious game development can be split to actual game design and instructional design (Iuppa, Borst, 2010:123). In addition as Rogers tells game will be created following own passion albeit different tips can be found around (Rogers, 2014: loc1164). This design principle is followed with the game design. The following chapters discuss game design principles that are applied for an orienteering game, as it can be implement as a serious game.

4.1 Instructional Design

The actual game design can begin when it is known how a game will be functioning and presented to the user. Iuppa and Burst (Iuppa, Borst, 2010:124) describe how instructional design specifies a document how a player can go through in playing the game, the order in which the steps have to be taken, the worst-case scenarios for failure, and as well the best case scenarios for success. By following this guidance, instructional design specifies how a game can be played. This is important because there is no actual customer who could specify how the game should work.

Classic orienteering is performed with a map and a compass. Player’s aim is to do orienteering on the map by going through a track containing one or more checkpoints. In an orienteering game application the intention is that these checkpoints are virtual ones (i.e. not visible for the player in terrain like in typical orienteering). When the last available checkpoint has been reached the player has finished the game. Orienteering can be thought as climbing on ladders where there is a need to take each step to reach up the roof and in the right order. The following steps can be included to describe the game steps:

1. The player starts the application and enters the game main menu.
2. The player can select a new game from the game menu.
3. A new game is started.
4. In a classic orienteering a track time is started when a so-called K point is reached (i.e. the first checkpoint). Player’s current location is selected as the starting point.
5. When a player reaches the first checkpoint time will start and the score will display the past checkpoints.
6. After this player will go to each checkpoint in a free order.
7. When the player has checked the last orienteering control point the game will stop and return to the main menu.

In the proof of concept phase the game was kept very simple and not all exceptions were handled i.e. the game can be paused but if it is restarted it is not necessarily keeping the last know state. The game target is to find all check points as fast as possible. There are lots of possibilities how the game can continue or reward the player. For the sake of simplicity, the game was left on a short mini game level.

4.2 Gameplay Design

Gameplay design specifies the details for the game that it can be implemented and boundaries can be set. Because the virtual orienteering game was kept simple not too much time was spent on high grain graphical design or the usability design. The target was to create an orienteering game skeleton that can be used later on for building a commercial game keeping actual game design minimal. Designing a gameplay can be though as an art and a science (Iuppa, Borst, 2010:187). To make a game more fun is science as the player is performing certain stories and environments and art because of a sort of choreography is needed (Iuppa, Borst, 2010:187).

For the sake of simplicity has the game menu system kept at minimum. Only mandatory selections are possible that the game can be started, paused and restarted. The actual orienteering part of the game utilizes a downloadable online map around the location of the player. As the used map is utilized directly from the provider as is no design is spent. Checkpoints used to construct the orienteering route are drawn on the screen as images having an orienteering flag symbol. A track can be drawn as a line connecting all checkpoints together but that can be left out to enable free orienteering i.e. going through the checkpoints in a random order. The game could have time indicator that shows how long the game has been on-going and a scorecard to show the progress. However it was decided not to use time indicator in the latter version. The player's location is indicated on the screen as a graphical symbol to make it easy for the player visualizes his own place on the map in relation to the checkpoints. This kind of feature also helps player's who do not have much experience in orienteering and to lower the barrier to use the game.

4.3 Summary

A virtual orienteering game can be thought of a serious game. When designing a serious game good procedure is to split design into instructional design and actual game design. These are separate designs where an instructional design creates a manuscript on how the game will be played before jumping to the actual game design. The game design includes details and describes more thoroughly the game mechanisms and technical considerations. Before starting to implement an orienteering game some time was spent with the principles of game design by going through processes that help the actual implementation to be successful. The principles included the steps how the game will operate.

5 Cocos2d-x Game Engine

The usage of smartphones for a gaming device has been growing rapidly. Independent game developers and start-up companies can create amazing mobile games and distribute them easily through different application stores. These mobile games are almost simultaneously released in different stores and different platforms to increase revenue. (Shekar, 2014: 1). Game developers benefit of using tools that support easy and fast application development. One of the tools is Cocos2d-x game engine that is used in several companies such as Zynga and Disney (Hussain, 2014: 246). Cocos2d-x is a cross-platform tool i.e. the same source code can be run on several devices having it attractive choice for developers. Furthermore this tool can be used to implement applications that are not directly amusement games but more of serious ones. Considering this makes it attractive also for developing a virtual orienteering game.

5.1 Cocos2d-x

Today there are several famous developer tools for creating mobile games called game engines suitable for mobile game development. A game engine is a software framework that provides common game-alike functions for building games (Cocos2d-x.org:Chapter 2). It is a piece of software that provides APIs for different purposes. Game engines such as Cocos2d-x, Unity and Moai to name a few, all support cross-platform development. Unity is probably the most known platform at the moment and games can be programmed with same Javascript and deployed to several different platforms. The case is similar with Moai, which has been implemented with C++ but also provides Lua scripting to program games (Tufró, 2013:loc 13).

Cocos2d-x is a cross-platform and an open source game development framework implemented in C++ code and can be deployed for several mobile devices such as Android, iOS and Windows Phone. A typical C++ program is constructed from several modules (Bronson, 2010: 44). This can be seen also in applications created with Cocos2d-x because necessary modules are compiled based on the desired platform. Co-

cos2d-x is specialized for designing 2D games although latter versions of 3.x include features to support 3D game development. Cocos2d-x has also other versions that are used for certain purposes such as Cocos2d-js, which is a game engine that can be programmed with JavaScript. Another variant is Cocos2D, which is targeted for Apple iOS devices only. The structure of all of these variant engines is quite similar because Cocos2d-x is based on a portable version of Cocos2D. The intention of Cocos2D is to support iOS game development as it has been implemented with Objective-C, which is the major programming language for iOS systems. Basically both share similar functions. As Cocos2d-x is developed as a continuous open source project (Cocos2d-x.org:Chapter 1) it is evolving all the time. It has been developed since 2010 and currently version 3.8 has been published (Cocos2d-x.org:Chapter 1). A dedicated open-source community maintains the code base and is adding new features to Cocos2d-x. In the latest versions support for 3D gaming has been added.

Cocos2d-x supports different screen sizes. This is one of the most important features when designing an application for several different devices. Typically the developer needs to program an application to support different screens sizes, however, it is easy to program Cocos2d-x to find out bounds of the screen and make necessary adjustments automatically. When screen objects are placed on the screen in a relative manner it does not matter what the screen size actually is. Among screen size dots per pixels (DPI) play on important role. An application designed for high DPI does not necessary look good if the screen size is big but DPI is low and vice versa. For this purpose it is possible to develop an application that figures out the screen size and its density and applies dedicated and scaled screen objects to a specific device. This requires that an application developer will create high and low density graphics for each necessary object on the screen that are selected on runtime to fit on the available screen. In practise testing for this should be done with all supported screen sizes to see the real world situation.

5.2 C++ with Cocos2d-x

Cocos2d-x games are programmed with C++ programming language, which can be difficult because the programmer needs to take care of the memory management. In C++ programming language memory management allocation and de-allocation of different objects is needed and specific functions are used for this purpose. This is one reason why programming with C++ is sometimes challenging and different instability problems may occur in the software caused by memory leakages. These memory leaks can happen when the objects being allocated are not de-allocated causing system memory to run out. Typically in C++ programming language new and delete operations are used. Cocos2d-x uses autorelease pools and retain counts which can be seen inherited from the iOS version of Cocos2D. Memory can be allocated either in a static or in normal C++ language dynamic way. In Cocos2d-x static way of memory allocation is recommended (Engelbert, 2013: loc 668-669). In the static way special Cocos2d-x factory methods provided by the base classes are used. In the normal dynamic way typical C++ language new and delete operations are used.

In general C++ programming language provides a relatively portable and common ground for all platforms. In the case of iOS and Android development it is a perfect choice as iOS Objective-C code can be mixed directly with Cocos2d-x C++ and with Android Native Development Kit (NDK) can be used.

5.3 Structure of Cocos2d-x Game

Cocos2d-x is a structured game engine i.e. it contains a collection of classes and methods optimized for 2D games. So called container objects are used to maintain individual game screens. They are important objects because they manage collection of sprites and other containers inside of them (Engelbert: 2013:loc 557). The main part of any video game is to render things on the screen (Muzykov, 2012: 44). For this reason a sprite object plays an important role in any video game and is the base element of any Cocos2d-x game. The sprite is an object of a rectangle image or a texture that can be placed anywhere on the screen with depth called Z order. This Z order will tell the daring order of the objects on the screen keeping the object with the highest Z order number as the top most. Cocos2d-x Sprite class has been derived from Node class (Jordán: 2015, 27). This makes it easy to combine them together as a typical game consists of several sprites that do overlap and collide. Sprites are graphical objects that Cocos2d-x engine will take care of after they have been programmed and placed on the screen. OpenGL is used for drawing and hiding low-level graphics functions in order to make programming games easier for the programmer. In general game engines specify game loops that are used to control graphical objects with some actions and respond to user actions. In that sense game engine can be though as programmable logic or automation system where different events causes engine to react and control all objects. Sprites objects live their life in the automated system based on a programmed behaviour or a user interaction. This is the case also with Cococs2d-x based games.

A typical Cocos2d-x based game is formed out of a collection of suitable sprites, which are placed on the screen. Intention is not to draw any vector graphics. Controlling of tens or hundreds of sprites individually is a complicated work and that is why Cocos2d-x implements layer and node (collection) classes to help bundle objects together. Sprite objects can be aggregated to layers and nodes that are easier to manage as entities. If a game character is formed from multiple sprites like an animated character, it is easy to control the whole aggregation of sprite objects (graphical object formed from several sprites) as a one instead of individual sprite objects. This characteristic makes Cocos2d-x well suitable for virtual orienteering game because when a game map is constructed from a number of map tiles and orienteering track is formed out of multiple checkpoint objects (sprites), which can easily manipulate the whole collection of sprite objects. This is important when the whole map including a track and user sprites is needed to scale or rotate. Scaling or rotating of sprite is an example of Cocos2d-x actions (Hussain et al, 2014:loc 835). An action can be attached to the node containing all layers of sprites (Hussain et al, 2014:loc 835). In this manner one action can be used by a combination of sprites.

Cocos2d-x uses scenes to build up game screens. A scene can be though as an individual playground for different drawable game objects (sprites) used in the game. When thinking of old handheld LCD (Liquid Christal Display) game terminals from the 80's they typically contained only one scene. Nowadays a typical game contains several scenes such as an opening, a main menu, the actual game and an end title. Each scene is typically constructed by connecting layers and nodes together. Each node then again contains objects that can be grouped together using other nodes. Director is a special class in Cocos2d-x. It is a singleton object that is used for managing scenes and controlling the whole application itself. Change from one scene to another is performed with the director. An example of this is when the game is ending and end screen is displayed. The director also provides caching of objects so that it is not necessary to load them from a device permanent storage every time they are needed.

In addition, Cocos2d-x provides means for drawing other type of graphic objects like text and menu items. Due to the nature of the game engine it has ways of how interaction of graphics objects can be detected and different effects created. Furthermore, Cocos2d-x contains a lot of other features that can help creating several different game styles. To name two of them are the support for audio playback and the different effects of the screen such as particle handling. A high number of particles can be used for creating effects like snowing, campfire or different water effects movement.

In a virtual orienteering implementation the purpose is to use the most beneficial features and minimal subset of Cocos2d-x. These are high performing graphics that are needed to display a map, an orienteering track and HUD (Head Up Display). HUD can be used to display information such as a compass, spent time and information about the orienteering track (scores). To support multiple platforms and screen sizes Cocos2d-x supports implementation that this can be achieved. This can be seen a very valuable feature when application is developed for multiple platforms.

5.4 Limitations of Cocos2d-x as to Virtual Orienteering

Cocos2d-x is missing three important features that are needed to create a virtual orienteering game. These features need to be added before the actual game can be implemented.

The first and most important feature is to create support for a map suitable for virtual orienteering. Cocos2d-x supports sprite images that can be used as background images as a map could be constructed using background image. Because the map forms the base for orienteering it needs to be created first. On top of a map (background image) overlapping sprite objects can be used to augment the information of the map. Cocos2d-x supports a so called tiled map. Tiled map information is provided separately and it is created using an external tool. A tiled map file is provided as collections of small images that are scripted to be places on map to form a bigger map. This would reduce the memory needed to store large area maps. This approach could be used if an artificial tile map was used. However, to construct a real world orienteering map a tiled map makes no sense. Due to the complexity of real world orienteering maps and their limited and licenced availability existing Web online maps will used. Licensing and copyrights restrict also the usage of a scanned image of the real world map which usage is discarded.

Both iOS and Android provide their own frameworks to implement a map as they both support proprietary APIs for using their own build in maps. However the implementations of these are done at higher level than Cocos2d-x is meant for. Adapting Cocos2d-x to use platform dependent map would take too much effort and probably result would be as many implementations as platforms. Implementation of own map layer should be selected because then suitable map implementation can be designed to be platform agnostics as it is based on Cocos2d-x capabilities. Alternatively as a map layer is implemented as own layer that can be replaced later on.

Knowledge of the location of the player is important. There is no ready support at the moment in Cocos2d-x where a device location could be read. A virtual orienteering game needs to track the location of the user so that virtual punching of checkpoints can be made possible. For this reason additional location functions were implemented. Typically smart phones or devices have a GPS/GLONASS chip on board that makes the implementation of reading location information relatively easy. Cocos2d-x does not offer location API or framework like native iOS and Android devices. This means that some mechanism needs to be built for the virtual orienteering game so that this location information is available for the game designed for Cocos2d-x. Reading location information needs to be implemented agnostically independently on the used platform. The best way is that location information can be read as latitude and longitude coordinates that make locating the user on the map easy. Location information is also used to find right online map tiles. Online map tiles can be fetched from the service provider based on the information of the world coordinates.

The third missing feature important for orienteering is the lack of API for reading magnetometer of the device. Magnetometer is a specific feature for a given platform. Its API needs to be implemented separately per device platform. Android platform provides 3D magnetometer that needs additional handling to get reliable measures. In the iOS side so called heading information is given directly by its location API. Not all devices provide magnetometer information so it is important that the virtual orienteering game is not dependent on the magnetometer or compass.

5.5 Apple iOS Platform

Apple iOS platform is a closed development environment. That means that Apple does not share the source files of the system. In addition Apple is the only manufacturer of the iOS based devices. Good thing in this system is that the devices are always similar ones and basically all devices support the same operating system version. Apple iOS platform can be programmed with Objective-C programming language and nowadays also with Swing language. Swing is relatively new and is left out of the scope of this study. Objective-C is compliant with C and C++ languages. Actually, Objective-C is based on C with object-oriented extensions (Nahavadinpoor, 2013: loc 224). That means that although iOS is basically written using Objective-C the development is somewhat possible to do with C or C++ languages. This is a huge benefit when doing application with Cocos2d-x as it is fully C++ implementation. Apple development is done using XCode IDE tool that supports these languages directly. Because XCode IDE is needed in practise this means OSX based computer like Mac Book Pro. (Apple Developer:2015)

Debugging in iOS environment requires use of iOS product and a specific developer agreement that is renewed on yearly basis. Debugging is done with a real world iPhone or iPad. Apple IOS development program provides a license to debug software in a limited amount of specially licensed devices. Apple maintains the development environment and provides software distribution channel through its AppStore. (Apple Developer: 2015)

A typical iOS application is built on top of layers found in iOS architecture (Figure 6). The Cocoa Touch forms the basics of an iOS application. This layer provides functions for most of the typical applications seen by the user and provides basically all controls for the user. Apple also provides guidance on how iOS applications should behave. This means that certain frameworks have limited customization and are based on design patterns. Typically an application created using these layers has the same look and feel. Build-in applications found in the iOS are good examples of Cocoa Touch based applications. The Media Layer contains graphics and other technologies to implement multimedia services. Such applications like camera, video and music playback are examples of applications utilizing this layer. Cocos2d-x partly overrides this layer by providing own methods for handling graphics directly using OpenGL layer. OpenGL-ES is well supported by the iOS system and benefits the hardware level graphical acceleration. The Core Services layer provides e.g. location and networking functions that are also important for virtual orienteering game. The Core OS layer provides the lowest level functions to the application. This is typically the layer that is not used or available for normal iOS application developer. (Apple Developer: 2015). All these layers and their functions are provided for the Objective-C or Swing based applications.

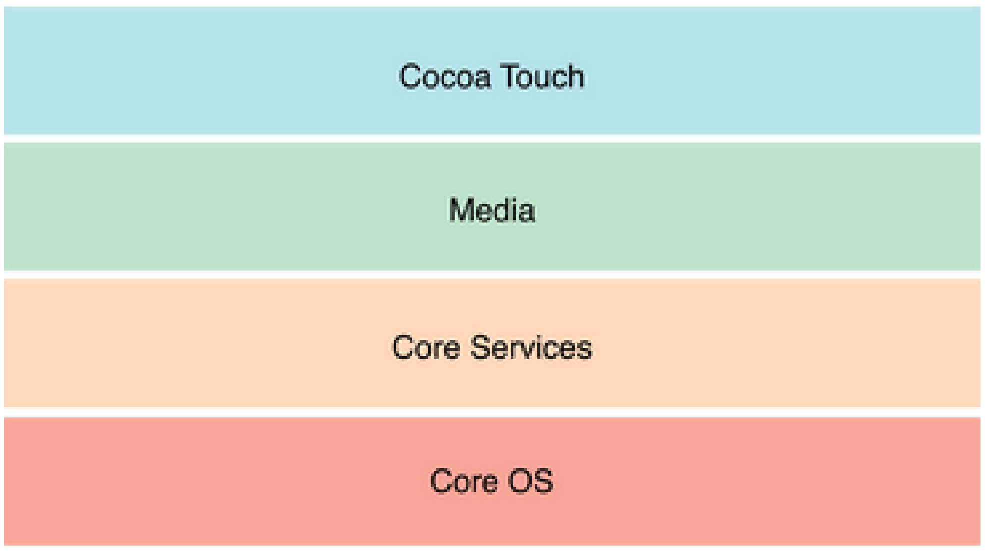


Figure 7. iOS architecture (Apple Developer, 2015)

Cocos2d-x application hides the iOS architecture, which is hardly visible and basically all functions are override by game engine specific functions. As Cocos2d-x has evolved from Cocos2d that was originally used to develop iOS games only it seems that starting of the version 3.x has Cocos2d-x development community hidden iOS related functions and creating more platform agnostic version of it. To develop Cocos2d-x application there is little that a developer would need to know about iOS internals to make a game. In practise only the main class of iOS application is used and that is used only to launch Cocos2d-x application. (Engelbert, 2013: loc 459)

5.6 Android Platform

Android platform is based on Linux and currently Linux kernel 3.x is used by the latest Android 4.x and 5.x platforms. Android is an open platform and its source files are available on Android developer site for device manufactures and developers. Most of the programs developed for Android are made with Java language and with Android Development Kit (ADK). Android Development Kit contains necessary source files and tools for compiling and debugging applications. Emulator environment is available for those not having a suitable device. Development of C++ based programs on Android is possible using separate Native Development Kit (NDK) (Sylvain, 2010: loc 262-263). As Cocos2d-x is done with C++ it covers graphics intensive libraries for Android created with Native Development Kit. On Android Cocos2d-x contains an abstraction layer on top of OpenGL-ES functions. This makes it possible to do e.g. game programming directly with C++. Android supports OpenGL-ES functions also in Java. So it is possible to do games using Java only. (Android, 2015: developer pages)

Low-level API access is restricted and controlled on Linux access level. These are e.g. Linux based device drivers. Android platform is based on Security Enhanced Linux (SE Linux) that restricts pure Linux use by providing additional access control for the applications. Most of the HAL (hardware abstraction layer) APIs can be accessed only through Java virtual machine called Dalvik (Android 4.x and earlier) or ART (Android Runtime) in the newer versions of Android 5.x (Lollipop). Figure 8 shows the relation of the HAL layer and other components of Android architecture. ART is run with a more privileged mode than normal application software and that is why it is mandatory to use that to gain access to certain APIs. To use e.g. different sensors from C++ language one needs to use Java Native Interface (JNI). That provides bridge between C++ and Java virtual machine. JNI methods are used to provide necessary APIs for virtual orienteering game that is built with C++ on top of Cocos2d-x game engine. (Android, 2015: developer pages)

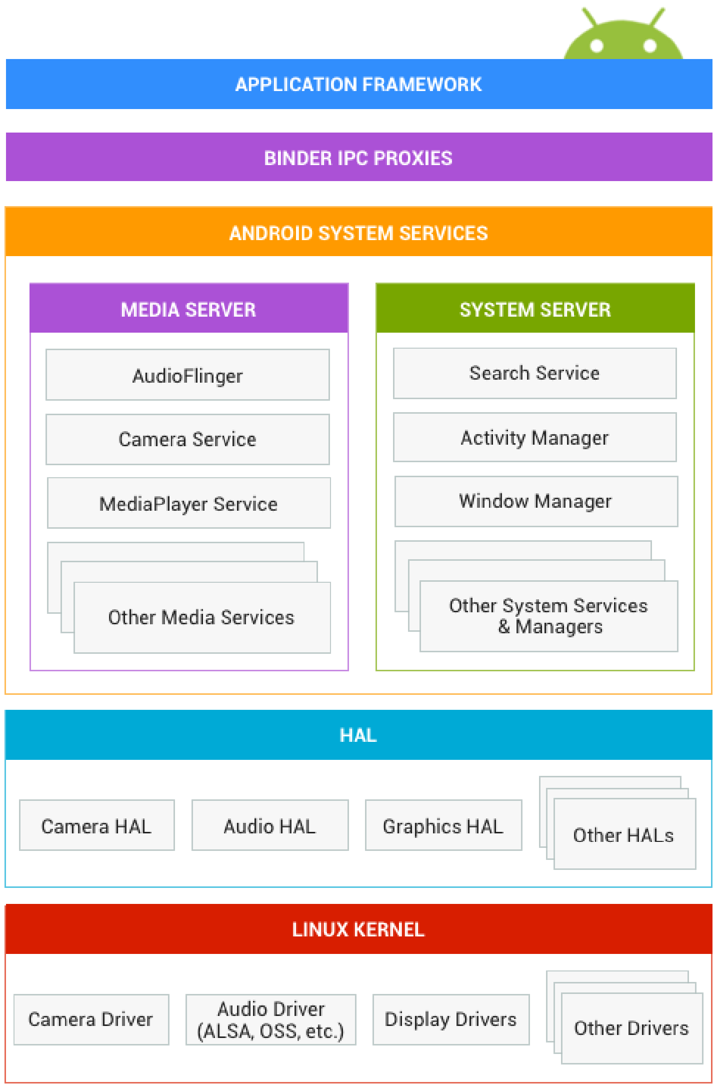


Figure 8. Android System Architecture (Android, 2015)

Android software can be built almost with any PC running Linux, Windows or Mac OS-X. ADK is provided for all of these platforms and it is freely available. On target debugging can be done basically with any Android device by enabling its developer features. A specific registration of a device is not needed. ADK provides also necessary emulators that can be used to test with different Android versions, screen sizes and skins. (Android, 2015: Developer Pages)

In Cocos2d-x application Android architecture is also hardly visible and basically all functions are override by game specific functions. As with iOS to develop Cocos2d-x application there is little that a developer would need to know about Android system to make a basic game with it. In that sense game development is similar as with iOS development.

5.7 Design Principle for Missing APIs and Features

Implementation of missing and required APIs needs to be done for each platform separately. Luckily Objective-C used by iOS can be compiled with the same C++ compiler used by Cocos2d-x, which comes along with XCode IDE (Engelbert, 2013: loc 436). In the Android environment the story is a bit different. As mentioned Android does not allow usage of C/C++ level functions but provides public Java based ones. This means that a specific C++ wrapper is required to be done for converting Java based APIs so that they can work in C++. Java programming language supports native interface (JNI) that helps of calling Java methods from C/C++ programming language. The intention is, however, that at the game level same C++ based classes and methods can be used in both systems and by using C++ define macros one can force compilation to select right source files or snippets to the used platform. Cocos2d-x supports built-in compiler directives that through the source code platform dependent source is selected when needed. (Cocos2d-x, 2015: Developer Pages)

5.8 Summary

Cocos2d-x is a general game engine designed for 2D games (SlackMoeri et al, 2015:2). In addition it provides many interesting features that makes it an ideal candidate to be used for developing a virtual orienteering game. The most beneficial features are effective graphics handling, support for different screen sizes and several platforms. To use Cocos2d-x in virtual orienteering it is necessary to create additional functionality that supports reading of a device location, an orientation to magnetic north and also a way to support for display scaling and large area moveable maps on the screen. Android and iOS have different architectures and they use different programming languages for the development. With the help of Cocos2d-x it should limit the platform maintenance work of the application because most of the application can be programmed or implemented with one design using C++ programming language only.

6 Implementation of Virtual Orienteering Game

The following chapters describe how the virtual orienteering game proof of concept was developed. The implementation demonstrates how to benefit of Cocos2d-x when implementing a virtual orienteering game. The intention was not to implement a final game to be published in any application store but to evaluate and prove the benefits of game engine usage. This can be beneficial when the game will be further developed for the commercial or other purposes. A virtual orienteering game can be though as a serious game application where all fun does not come from the user game experience but also from learning the principles of basic orienteering.

6.1 Requirements for Proof of Concept

For developing a proof of concept requirements for the virtual orienteering game were set. Requirements came from the actual game design. By fulfilling the requirements it was easy to evaluate the success. As the purpose was to evaluate the suitability of Cocos2d-x the intention was to keep the implementation at the bare minimum. As an outcome, additional requirements were studied and also ideas for further developing the game onto a level that it can be published in different application stores were identified. Furthermore, the intention was not to design all graphics but use available images for demonstration purposes.

To evaluate the success and to give guidance for the implementation the following mandatory requirements were set:

1. Basic navigation and menu structure will be in place in order that the user can start the game and can understand how to initiate the game.
2. Game should have start and stop functions with the possibility to pause and restart the game.
3. Freely available tile map service will be used. This is due to the fact that most of the orienteering maps are licensed. The quality of the maps for orienteering

does not matter because in practise any type of map from virtual to real orienteering map can be used later on.

4. Checkpoints are placed virtually in the terrain. The application will indicate when a checkpoint is found i.e. when the player has approached a checkpoint. GNSS based location is used to indicate when the user is in the right place.
5. Virtual checkpoints are placed on the map screen of the application.
6. Electrical compass indicator will be implemented and displayed.
7. Game can be run at least on iOS or Android based devices.
8. Different screen sizes and resolution will be supported.

Requirements in the list create the boundary for the game. They also present the idea of the game and what player’s target is.

6.2 Development Environment

To develop and debug applications for iOS devices an Apple OSX computer and iOS device are required. An Apple computer supports Android development because tools required for developing Android application support OSX. The development of the game was done using MacBook Pro having XCode IDE, Eclipse IDE and necessary platform dependent tools. Both XCode and Eclipse fulfils basic development. XCode provides comprehensive environment for all development needed for iOS systems. For Android systems additional tools such as Android Development Toolkit (ADT) contains tools for developing Java based applications and Native Development Kit (NDK) is needed for Android applications created with C++. In addition, Cocos2d-x sources are needed including necessary tools to make compiling possible.

6.3 Device Variability

Cocos2d-x supports implementation that is agnostic for used device i.e. the software can be compiled to several different operating systems and devices. Although Cocos2d-x supports many different devices the proof of concept implementation was restricted to support two different major device architectures. These are Apple iOS (iPad mini as reference device), and Android (Samsung Galaxy Tab as reference). Devices used for on-target debugging need not to be relatively new and efficient ones. Because of the usage of compiled C++ code Cocos2d-x should run on those without major performance issues. Different devices will be tried based on availability and with possible test users around. Table 3 lists supported devices and resolutions that were done the proof of concept.

Device	Resolution in pixels
Apple iPad Retina	2048x1536
Apple iPhone 6	1334x750
Apple iPhone 5	1136x640
Apple iPhone 4s	960x640
Android LG Nexus 5	1080x1920
Android Samsung 4 tab	800x1280
Android Samsung tab 2.0	1024x600

Table 3. Supported device resolutions.

Not all devices support magnetometer or electrical compass. Example of lacking magnetometer was seen on Samsung Galaxy Tab 2.0 device. In practise all newest iOS devices contain electrical compass but more variation can be found in the Android devices. GNSS support can be found basically in all smart devices on the market at the moment. Difference can be found of the supported GNSS systems on the device. All devices support GPS but GLONASS and other newer systems may not be supported. Support for GNSS or magnetometer should be checked from the technical specification of the device.

6.4 Support for Different Screen Resolutions

A virtual orienteering game as any game needs to support different screen size and pixel density (see Table 3). Reason for this is to maintain correct user experience. The screen size and pixels density together define the available screen resolution. Two devices having the same display size may have a different pixel density and thus a different resolution. To scale an image correctly the pixel density needs to be adjusted so that similar amount of graphical information can be displayed at low and high-density screens. Intention is that devices with different pixel density would show the graphics at the similar size although resolution could be coarse when compared to the higher density device.

Cocos2d-x provides in practise two approaches to support multiple screen sizes. The first one sets the drawing area to a specific screen resolution and lets the engine's OpenGL-ES based system to scale screen for a suitable size. This means that systems take care of zooming pixels automatically in a way that e.g. a target platform that has 800x600 pixel density and system virtual density can be e.g. 320x400. In a high-resolution device using a low-resolution mode this can be seen that individual pixels may become blurry due used antialiasing.

The other way to find out available drawing area (screen size) can also be used. Available screen size can be read with the following C++ statement that gives the available pixel resolution (also a variable will be initialized with the same statement):

```
auto screenSize = glview->getFrameSize();
```

Drawable area can be set with the following statements (768 x 1024 pixel area in this case):

```
glview->setDesignResolutionSize(768, 1024, ResolutionPoli-
cy::NO_BORDER);
```

As indicated the screen size (pixel resolution) is read from OpenGL-ES level which is common for all supported platforms. Based on a screen size and with the help of if-then-else clauses a correct image can be used by the application when available resolution has been interrogated beforehand. This is the case when different resolutions are supported. The following C++ code snippet (see Listing 1) shows how this can be done by selecting a right size of image for a given screen resolution:

```
    std::vector<std::string> resDirOrders;

    // check which assets the devices requires
    if ( 2048 == screenSize.width || 2048 == screenSize.height )
        // retina iPad
    {
        resDirOrders.push_back("ipadhd");
        resDirOrders.push_back("ipad");
        resDirOrders.push_back("iphonehd5");
        resDirOrders.push_back("iphonehd");
        resDirOrders.push_back("iphone");

        glview-
>setDesignResolutionSize(1536,2048,ResolutionPolicy::NO_BORDER);
    }
    else if ( 1024 == screenSize.width || 1024 == screenSize.height )
        // non retina iPad
    {
        resDirOrders.push_back("ipad");
        resDirOrders.push_back("iphonehd5");
        resDirOrders.push_back("iphonehd");
        resDirOrders.push_back("iphone");

        glview-
>setDesignResolutionSize(768,1024,ResolutionPolicy::NO_BORDER);
    }
```

Listing 1. Selecting resolution (Hussain et al, 2014)

In the code snippet example (Listing 1) Cocos2d-x game engine uses a specific directory structure to select used graphical objects based on this information. Each directory is named based on a device resolution or some other naming. This means that an application designer or a game artist can create two or more versions of the images to support low, medium or high-resolution displays on the devices (Engelbert, 2013: loc 795). Cocos2d-x sprite system supports image scaling. This also means that it is possible to design graphical sprites of one size only with a low resolution and scale size up the image on the fly. Scaling size up of the sprite is simple and can be adjusted based on the available screen size information, which can be programmed using if-then-else clauses. In many cases high-resolution images need to be created separately by a game artist. Scaling up a low-resolution image automatically may cause image to be shown too blurred. Furthermore, special design may be needed to make high-resolution graphics to look good enough. Cocos2d-x supports also a mode where a screen pixel resolution is set static. This is another way to support different screen sizes and the game engine will take care of adjusting graphics to a right screen resolution. This approach would make a generating of necessary game graphics (such as different menu items and sprites) less effort because then only one version would be required. Because the virtual orienteering game is a map-oriented application it is good to go with a support that utilizes the maximum available resolution to get as big a map area visible as possible.

6.5 Game Menu System

Each game requires some menus and a game screen so that a player knows when a game needs to be played. Figure 9 shows a game screen used in the application.

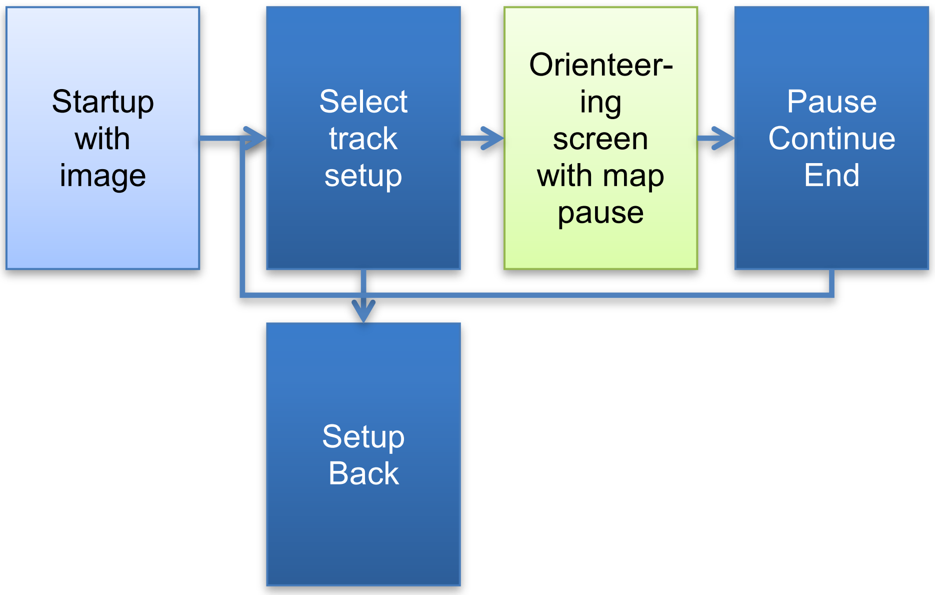


Figure 9. Different scenes of the game

Cocos2d-x defines game scenes that are used to describe each visual screen, which are controlled by a special director (SlackMoehrle et al, 2015:4). Each scene can contain its own functions and can act autonomously. These scenes can be tied together with logic that is called a director. Within the director different scenes can be swapped based on user selection or other game logic. As an example a virtual orienteering game utilizes scene swap snipped below:

```
auto scene = GameScreen::createScene();
Director::getInstance()->replaceScene(TransitionFade::create(1.0,  sce-
ne));
```

In this case a special transition fade effect is used which means that 1-second time is used to fade out the old scene before a new one is brought up instead. Cocos2d-x provides very easy way to navigate between different screens. When comparing to iOS and Android native application one needs a bit more code to give a similar effect. The game menu can be created easily as Listing 2 shows. Selected images corresponding menu items are placed to certain callback methods that are called when the menu item is touched on the screen.

```
auto resumeItem = MenuItemI-
mage::create("PauseScreen/Resume_Button.png",
"PauseScreen/Resume_Button(Click).png",
CC_CALLBACK_1(PauseScreen::Resume, this));
auto retryItem = MenuItemI-
mage::create("PauseScreen/Retry_Button.png",
"PauseScreen/Retry_Button(Click).png",
CC_CALLBACK_1(PauseScreen::Retry, this));
auto mainMenuItem = MenuItemI-
mage::create("PauseScreen/Menu_Button.png",
"PauseScreen/Menu_Button(Click).png",
CC_CALLBACK_1(PauseScreen::GoToMainMenuScene, this));
    auto menu = Menu::create(resumeItem, retryItem, mainMenuItem,
NULL);
    menu->alignItemsVerticallyWithPadding(visibleSize.height / 4);
    this->addChild(menu);
```

Listing 2. Creating game menu (Hussain et al, 2014)

To make the swapping of scenes natural a menu system was implemented. Cocos2d-x provides built-in functions to represent menus that can be placed on a desired scene. In practise menu items are graphical images.

6.6 Map Implementation

There are several options to implement a map used in virtual orienteering. For the proof of concept a tile based online map implementation was selected and the tile provider can be Nokia HERE or OpenStreetMap. By using an online map the information is available accurately on demand. An orienteering game can also be used anywhere independently on the location of the user. That is due the usage of the global map information. Typical online maps are not good enough for serious orienteering but they demonstrate well the idea behind. The implementation was done in such way that it is possible to change to any other implementation e.g. if offline orienteering specific maps will be used later on.

The online map used is constructed from a set of tiles of 256 x 256 pixels. The right set of tiles is downloaded on demand based on a desired location. Tiles are available with different zoom levels which corresponds the details of the map. For the proof of concept zoom level 17 was selected which shows roughly an area of 100 x 100 meters per one map tile (see table 2). With this accuracy creating a map of an area of 1 km x 1 km would require 10 x 10 = 100 tiles to be downloaded and in addition occupy 100 sprite objects.

Zoom level	**Scale (meters /pixels)**
15	4.78
16	2.39
17	1.19
18	0.60

Table 1. Zoom levels and scale (Nokia HERE Developer, 2015:63).

A map scale of zoom level 17 is used because that gives a relatively good view on the map. A zoomable map could be utilized but like in normal orienteering the map scale is always fixed and that approach has been selected.

The implementation was done utilizing Cocos2d-x layer object that is easy to be replaced when needed with another layer object. In the software this is called `TileMapLayer`. This layer object will take care of fetching necessary tiles online from the defined tile map service. The map layer is constructed from a set of sprite objects. Each sprite corresponds one map tile of 256 x 256 pixel. Sprite tiles need to be placed to the layer in the right position that map is constructed correctly i.e. map is a construction of several tiles. For network operations Cocos2d-x provides two classes called HTTPClient and HTTPRequest. Using these methods a correct map tile is fetched by HTTP request from the server. A map provider request is used together with possible licensing keys. A simplified method without any error handling is shown in the following code snippet (Listing 3):

```
void TileMapLayer::fetchOnelineTileXY(int x, int y)
{

    // request test image
    auto request = new HttpRequest();
    auto xtiili = std::to_string(x);
    auto ytiili = std::to_string(y);

    std::string url =
"http://2.aerial.maps.cit.api.here.com/maptile/2.1/maptile/newest/terrain.day/17/"+xtiili+"/"+ytiili+"/256/p
ng8?app_id=demo&app_code=demo";
    request->setUrl(url.c_str());
    request->setRequestType(HttpRequest::Type::GET);
    request->setResponseCallback(CC_CALLBACK_2(TileMapLayer::onHttpRequestCompleted, this));

    log("TileMapLayer->HTTP request");
    auto client = HttpClient::getInstance();
    client->enableCookies(NULL);
    client->send(request);
    client->setTimeoutForRead(1000);

    request->release();

}
```

Listing 3. Fetching tile from tile server

Map tiles are fetched asynchronously and a callback method is called after each download of a tile. This can be seen on Listing 4. An asynchronous request makes it possible to download tiles in background when part of the map is already shown to the user. The user will experience this by seeing map tiles appearing one by one on the screen. Callback method will place received map tiles to the matrix in the right order when request has been fulfilled. A code example is very simple and does not take care of the order of the received tiles. In case the tile server responses to the requests are not in the same order, the map may be constructed in a wrong order of the received tiles and user will see corrupted map.

```
void TileMapLayer::onHttpRequestCompleted(cocos2d::network::HttpClient *client,  co-
cos2d::network::HttpResponse *response)
{
   //log("HTTP Response : %ld %s", response->getResponseCode(), response->getHttpRequest()-
>getUrl());

   if (response->isSucceed()) {
      std::vector<char>* buffer = response->getResponseData();
      auto *image = new Image();
      image->initWithImageData(reinterpret_cast<unsigned char*>(&(buffer->front())), buffer->size());
      auto *texture = new Texture2D();
      texture->initWithImage(image);
      tileSprite[counter_fetched] = Sprite::createWithTexture(texture);
      tileSprite[counter_fetched]->setAnchorPoint(Vec2(0,1));
      tileSprite[counter_fetched]->setPosition(Point(queue[counter_fetched].x,
queue[counter_fetched].y));
      this->addChild(tileSprite[counter_fetched], 0);
      counter_fetched++;
   }
   else
   {
      log("HTTP Request failed : %s", response->getErrorBuffer());
   }
}
```

Listing 4. An example of HTTP request

Additional code is needed in order so that map tiles can be placed to the right position and a full map can be constructed. Figure 11 shows the principle how tiles are placed on a device screen. In the figure, a matrix of 16 x 16 map tiles is used. A map tile nearest to the user location is placed in the centre of the device screen. Other tiles are placed in the relation to the centre tile. Not all tiles can be fit to the visible screen and a method for scrolling the map is needed. Figure 10 shows that a map area covers 4096 x 4096 pixels, which in addition contains tile sprites. These are placed on the Coco2d-x layer object. By doing it this way the whole layer can be controlled as one entity e.g. it can be zoomed or scrolled in the screen easily as one object by changing its attributes. There is no need to control each individual tile sprites after they have been placed on the layer object. Origin point (0,0) specifies the left bottom corner of the device screen and Cocos2d-x `visibleSize` object holds information about the maximum coordinate of the top right corner of the screen.

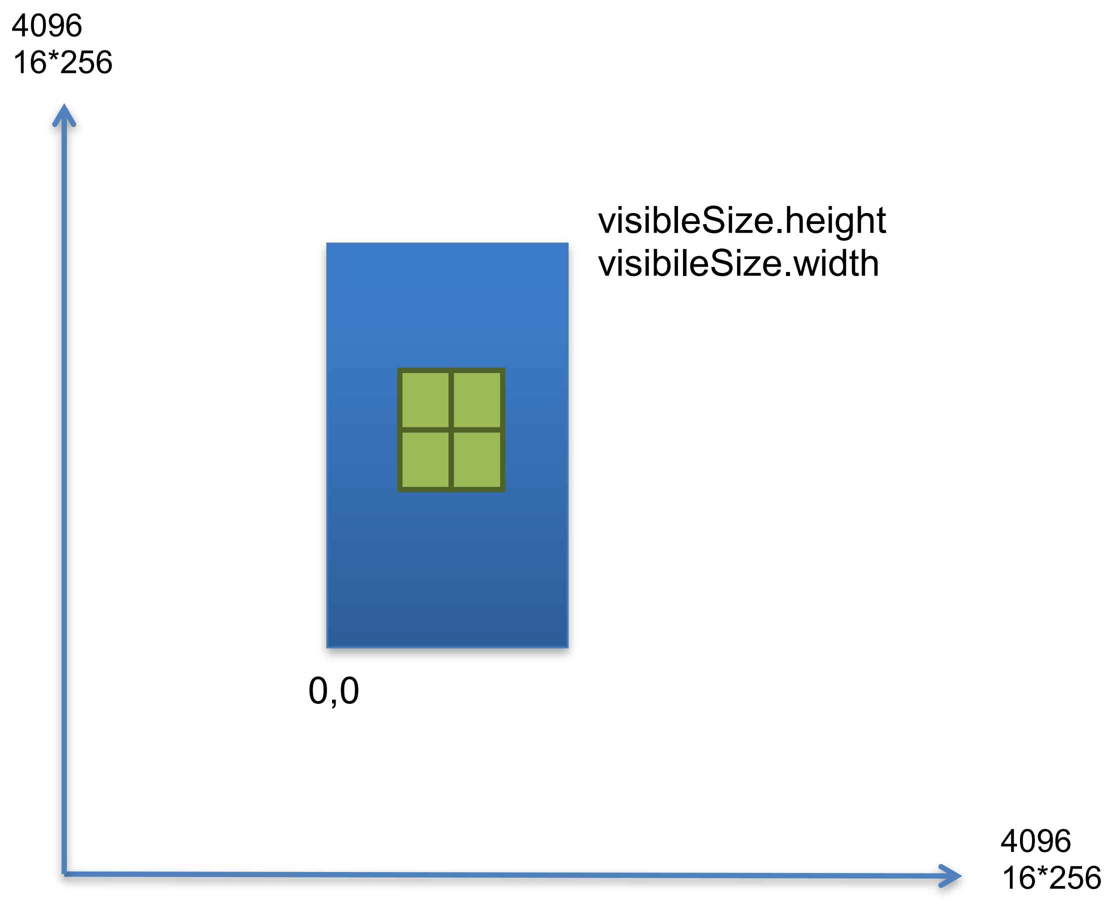


Figure 10. The placement of map tiles on screen

Calculation of how right tiles can be fetched from the server was described in the theory part. The implementation was done using a method that finds out the x and y coordinates of the specific map tile based on given longitude and latitude information (See Listing 5).

```
// Converts longitude to tile x coordinate
int TileMapLayer::long2tilex(double lon, int z)
{
    return (int)(floor((lon + 180.0) / 360.0 * pow(2.0, z)));
}


// Converts latitude to tile y coordinate
int TileMapLayer::lat2tiley(double lat, int z)
{
    return (int)(floor((1.0 - log( tan(lat * M_PI/180.0) + 1.0 / cos(lat * M_PI/180.0)) / M_PI) / 2.0 * pow(2.0,
z)));
}
```

Listing 5. Calculating x and y coordinates of map tile

6.7 Orienteering Track Implementation

Orienteering would not be a game if there were no tracks including checkpoints placed on the map and having motivation for the player. Orienteering is played using a track containing a starting point and several checkpoints. Different game modes exist that specifies how the track checkpoints are collected. For the proof of concept a very simple game mode was implemented. In this game all checkpoint coordinates are randomly picked near the user location when the game starts. In practise this does not make sense but enables the use of the game anywhere for demonstration purpose. Furthermore a simple “fan” orienteering can be implemented. In this mode (Figure 12) the intention is to find each checkpoint one at the time by returning to the starting place after each checkpoint. The track is indicated with red lines to be followed but in practise the player would find out the most suitable route on terrain. Typical starting point in orienteering is indicated by a triangle symbol to help see where track starts.

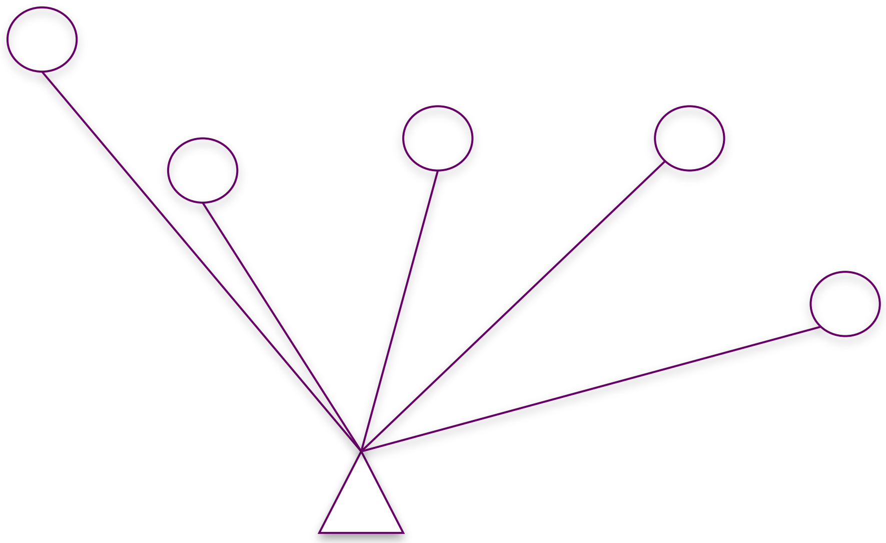


Figure 11. A fan orienteering track

The track was be implemented with own C++ class that uses random coordinates that will be generated when a game is setup. Cocos2d-x layer object was used including information about the checkpoints and the track. The reason was same as with the map layer case. Each checkpoint has been built as a sprite object and placed on the layer to the right place to correspond map coordinates. Checkpoint coordinates are based on latitude and longitude information. A right coordinate conversion is done so that checkpoint sprites can be placed on the device screen as in Listing 6.

```
/*
 calculates point in image when image center is lat&lng corresponds the center
 */
Point TrackLayer::fromLatLngToPoint(double lat, double lng)
{
    // Calculate pixel per degree ratio with Tile size of 256
    int pixelsPerLonDegree_ = 256 / 360;
    int pixelsPerLonRadian_ = 256 / (2 * M_PI);

    //auto pointx = origin.x + lng * pixelsPerLonDegree_;

    // Truncating to 0.9999 effectively limits latitude to 89.189. This is
    // about a third of a tile past the edge of the world tile.

   // auto siny = bound(sin(degreesToRadians(lat)), -0.9999, 0.9999);
    //auto siny = sin(degreesToRadians(lat));

    //auto pointy = origin.y + 0.5 * log((1 + siny) / (1 - siny)) * -pixelsPerLonRadian_;

    log("Point lat %f", lat);
    log("Point lng %f", lng);

    long pointx;
    long pointy;

    this->latLongToPixelXY(lat, lng, 17, &pointx, &pointy);
    log("Point x %ld", pointx);
    log("Point y %ld", pointy);

    return Point(pointx / pixelsPerLonRadian_, pointy / pixelsPerLonRadian_);
```

Listing 6. Calculating x and y coordinates from latitude and longitude

Normally checkpoints in orienteering are indicated as circle objects. In this game a graphical orienteering flag is used instead as a sprite object. Cocos2d-x sprites have a specific anchor point. By default an anchor point for Cocos2d-x sprite is in the middle of the image. This can be used to place orienteering flag in the centre of the coordinate location when placement is needed exactly in the centre of the object. Mandatory use of centre as an anchor point was noticed as it is required by the physics engine of Cocos2d-x, which is described in the next chapter.

6.8 Using Game Engine Physics

In a virtual orienteering game orienteering checkpoint flags are not visible in a terrain. A player location should be known and when the she approaches a virtual flag the system should notify this. This detection can be called as virtual punching or catching the flag. The game engines provide ways how this detection can be done. A typical way would be to compare the coordinates of the latitude and longitude of the checkpoint to the player coordinates. However, this would need an additional logic and would not be necessary the optimum way. To benefit game engine features Cocos2d-x implements a physics engine (SlackMoeri et al, 2015:95). This engine can be used to simulate physics to graphics objects (sprites) by e.g. implementing different objects that are falling or colliding each other. Collision detection of an object can be used in virtual punching of the checkpoints. In this way there is no need to compare location of a player and checkpoints but to let physics engine to manage checkpoints and user sprites and find out when there is a collision i.e. when user has found the virtual checkpoint. Physics is set with own methods and applied for all desired sprite objects separately. To enable physics for sprite object Listing 7 shows how this can be done.

```
auto body = PhysicsBody::createCircle(playerSprite-
>getContentSize().width/2);
    body->setContactTestBitmask(true);
    body->setDynamic(true);
    playerSprite->setPhysicsBody(body);
```

Listing 7. Drawing a circle

In this snippet a circular physics body is applied. As when an object is near another object physics engine will compare circular objects together when they collide. In addition other physical features can be applied for these circular objects if needed.

To specify what happens when collision is happening between sprites can be seen in the following code snippet (Listing 7):

```
    auto contactListener = EventListenerPhysicsContact::create();
    contactListener->onContactBegin =
CC_CALLBACK_1(GameScreen::onContactBegin, this);
    contactListener->onContactSeperate =
CC_CALLBACK_1(GameScreen::onContactSeparated, this);
    _eventDispatcher-
>addEventListenerWithSceneGraphPriority(contactListener, this);
```

Listing 8. Applying physics engine

Physics engine specifies callback functions for a collision beginning and ending. In a virtual orienteering game it is enough to check when a collision is occurring. When a collision has occurred it can be interpreted that the user has found the virtual checkpoint and necessary game related actions could be made. These are book keeping of found checkpoints and other needs for the game control.

6.9 Location Implementation

Location detection is needed to find out the location of the player and virtual checkpoints. After studying available APIs and based on the interpretation Cocos2d-x does not provide any location API directly and this needs to be implemented by the game designer. Location enabled smartphones can report a GNSS coordinates based on latitude and longitude values. In iOS this has been kept on a simple level but in Android it is possible to receive also NMEA messages that give more information about the satellites. Both platforms can report latitude and longitude values directly in a similar manner and a necessary C++ based wrapper class can be implemented that can be used in the general Cocos2d-x application. A low-level location API can be done by utilizing each platform's own location framework that is used by a wrapper class.

6.9.1 iOS Location API

Apple iOS developer guide provides Core Location framework (Apple, 2015:Location dand Maps programming Guide). A Device location can be acquired using the following simple code snipped (Listing 9):

```
- (void)startStandardUpdates
{
    // Create the location manager if this object does not
    // already have one.
    if (nil == locationManager)
        locationManager = [[CLLocationManager alloc] init];

    locationManager.delegate = self;
    locationManager.desiredAccuracy = kCLLocationAccuracyKilometer;


    // Set a movement threshold for new events.

    locationManager.distanceFilter = 500; // meters


    [locationManager startUpdatingLocation];

```

Listing 9. Reading location in iOS system

In addition relevant initializations should be made to get the code work correctly. To mention that Apple supports nowadays also Swing programming language but that is not considered because Swing is not directly compliant with C++ language used by Cocos2d-x. In iOS one typical the design pattern is delegate pattern. The code above states that a location manager itself is used as delegate and thus taking care of all location manager actions. This means that when location updates are started, a location manager is listening updates through `locationManager:didUpdateLocations:` method. IOS provides location information with latitude and longitude values that can be used directly without any conversion.

6.9.2 Android Location API

Android supports Location Service and uses Java class system called LocationManager (Android 2015). Location Service provides necessary means and methods for device to acquire its location. Listing 10 shows how that can be used.

```
// Acquire a reference to the system Location Manager
LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

// Define a listener that responds to location updates
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
      // Called when a new location is found by the network location
provider.
      makeUseOfNewLocation(location);
    }

    public void onStatusChanged(String provider, int status, Bundle
extras) {}

    public void onProviderEnabled(String provider) {}

    public void onProviderDisabled(String provider) {}
  };

// Register the listener with the Location Manager to receive location
updates
locationManag-
er.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, loca-
tionListener);
```

Listing 10. Android location manager

First, when a location service is taken into use `LocationManager` needs to be initiated and necessary Android Listerners defined. Second, through these listeners a location service will send a message about a location update and necessary actions can be taken. Also Android provides direct latitude and longitude information in similar manner as in iOS case.

To call Android Java methods from native application Java Native Interface call (JNI) invocation is required. In JNI C++ method can call a Java method running inside a Java virtual machine and respond data back. The following snippet shows how latitude and longitude information can be read from an Android Java virtual machine application to Cocos2d-x C++ based application (Listing 11).

```
double Location::getLatitude()
{
   cocos2d::JniMethodInfo methodInfo;
   if (!cocos2d::JniHelper::getStaticMethodInfo(methodInfo, "org/cocos2dx/cpp/AppActivity", "getLati-
tude", "()D")) {
      //return 0.0;
   }

   double jdouble = methodInfo.env->CallStaticDoubleMethod(methodInfo.classID, methodIn-
fo.methodID);
   methodInfo.env->DeleteLocalRef(methodInfo.classID);
   return jdouble;
}


double Location::getLongitude()
{
   cocos2d::JniMethodInfo methodInfo;
   if (!cocos2d::JniHelper::getStaticMethodInfo(methodInfo, "org/cocos2dx/cpp/AppActivity", "getLongi-
tude", "()D")) {
      //return 0.0;
   }

   double jdouble = methodInfo.env->CallStaticDoubleMethod(methodInfo.classID, methodIn-
fo.methodID);
   methodInfo.env->DeleteLocalRef(methodInfo.classID);
   return jdouble;
}
```

Listing 11. An example of JNI interface

The actual reading of the location information is performed with a Java based Android activity and example of this can be found in Android developer info (Android Developer, 2015:Location Manager). An example of this can be seen in the Listing 12.

```
// Acquire a reference to the system Location Manager
LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

// Define a listener that responds to location updates
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
      // Called when a new location is found by the network loca-
tion provider.
      makeUseOfNewLocation(location);
    }

    public void onStatusChanged(String provider, int status, Bundle
extras) {}

    public void onProviderEnabled(String provider) {}

    public void onProviderDisabled(String provider) {}
  };

// Register the listener with the Location Manager to receive loca-
tion updates
locationManag-
er.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0,
locationListener);
```

Listing 12. Reading location in Android

In this case Java methods are defined as static ones i.e. singleton methods in AppActivity activity. This activity is created automatically by Cocos2d-x initial project for Android. Java methods are called with a JNI helper method provided by Cocos2d-x and where it is also defined what kind of result is expected. Only a double type that corresponds a double precision floating-point number was used although other primitive types can be used. Necessary Java methods in Android utilize its Location Framework in normal Java enabled way and it can be coded as typical Android application.

6.10 Compass Implementation

Many midrange smart phones or devices contain a suitable magnetometer that can be used as compass. For a virtual orienteering game it is not mandatory to use compass but it gives nice additional user experience and in addition helps orienteering. A map in a device is always pointing to up of the device so use of compass helps to set the device position to face to the north.

The electrical compass implementation was kept simple and in a similar manner as API was shown previously in the theory part. A separate compass image (needle) was designed so that it can be used as a sprite object on the screen. The compass is basically implemented as just turning the sprite angle based on a device orientation compared to the magnetic north. The graphical compass is included in HUD layer (Heads Up Display) that gives other kind of information about the game. Both iOS and Android requires the implementation of APIs that gives heading information for HUD layer so that it can display graphical compass sprite correctly.

6.10.1 iOS

Implementing compass on an iOS device is easy. Heading information is supported directly with the location framework. Heading gives directly the necessary information that can be used to indicate a compass reading. The implementation of this part was very straightforward.

6.10.2 Android

Android provides a much lower lever API than iOS for a compass reading and it is not granted that each Android device includes a magnetometer. To access an Android magnetometer API a special JNI (Java Native Interface) interface is required in a similar manner as in acquiring a device location. Only one additional method was required to be implemented so that a device reports heading in a similar manner as iOS. For Cocos2d-x application reading compass or heading interface is same. The following Java code snippet (Listing 13) shows that on an Android device additional calculation is required so that a device heading can be read (Meier, 2012))

```
final SensorEventListener compassListener = new SensorEventListener()
{

   float[] R = new float[9];

   /**
    * handler for notifying when heading has changed
    */
   public void onSensorChanged(SensorEvent sensorEvent) {
      if (sensorEvent.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD)
{
           //Log.d("Orienteering", "Compass sensed");
           orientationValues = sensorEvent.values;
           SensorManager.getRotationMatrix(R, null, accelerometerVal-
ues, orientationValues);
           SensorManager.getOrientation(R, orientationValues);
           orientationValues[0] = (float)
Math.toDegrees(orientationValues[0]);
           orientationValues[1] = (float)
Math.toDegrees(orientationValues[1]);
           orientationValues[2] = (float)
Math.toDegrees(orientationValues[2]);
           //Log.d("Orientgames", "orientation
"+orientationValues[0]+" "+orientationValues[1]+"
"+orientationValues[2]);
           heading = (int) orientationValues[0];
           if (heading < 0) heading = heading + 360;
       }
   }

   public void onAccuracyChanged(Sensor sensor, int accuracy) {
   }
};
```

Listing 13. An example of reading Android sensor

In Android reading of the heading information is based on the combination of a magnetometer and a device orientation. The implementation (Listing 13) seems to give quite much noise and heading information varies a lot. This makes the compass quite unreliable. The heading information could be improved by creating a low-pass filtering function in order to reduce the noise of the magnetometer reading.

6.11 Summary

Code examples show that a virtual orienteering game can be implemented using the Cocos2d-x game engine. A basic orienteering game was made where the user can find virtual checkpoints that do not exist on terrain but on a device map and screen. When the player moves near to the virtual checkpoint the device will indicate of the checkpoint found. For a proof of concept the game mode was kept very simple and more advanced orienteering games are left out of the scope for potential future implementations.

Cocos2d-x game engine supports necessary navigation between menu and game screens and the scaling of graphics to various device screen sizes. The game engine provides a sprite mechanism that can be used to display different graphical information. Additional functions were implemented on top of Cocos2d-x so that it can be used to implement a virtual orienteering game. The first one was a map implementation based on an online tile map. Methods were created to support fetching online map information from the Nokia HERE maps server. Secondly, track information was created as an overlay on top of map information. Both the map and the track were implemented as Cocos2d-x graphical layers. Cocos2d-x functionally needed augmentation so that user location and compass information can be read. Both iOS and Android require their own API implementation and a certain wrapper class is needed so that location and heading information can be used device agnostically.

The actual game consists of checkpoints and a user that needs to find and reach the checkpoints. The intention was that when a gamer reaches a certain checkpoint this is implemented as a collision between a gamer sprite and a checkpoint sprite. A collision detection of different game objects (sprites) can be done in several ways. The first way to come up was to match when pixel locations of the two or more sprites are at same coordinate. To code this kind of a feature would require an additional logic to select which conditions are valid for real collision. Nowadays many games use a physics engine. This can be seen in games where something is falling from the sky or colliding to something else. Cocos2d-x includes a physics engine called Chipmunk (or the other one) that is used to simulate different objects movements typically in the earth gravity system. For orienteering this feature was not used because all objects lie on the electronic map. To benefit of the physics engine in this game is that it can be used to find out “collision” of the user orienteering and the checkpoints of the map.

A virtual orienteering game was implemented by combining different pieces together where typical game features (such as menus and graphics) could be done directly by Cocos2d-x supported functions and with additional location and compass functions.
The actual game implemented is an orienteering game where intention is to find checkpoints on the map in a predefined or random order. To make this more interesting

and demonstrative, flags can be collected in free order. The purpose in this case is not to follow the track in an orienteering way i.e. finding checkpoints in the predetermined order. However, it is still important to find the checkpoints.

7 Results and Analysis

As a result the proof of concept game of virtual orienteering was created. The game can be run on both iOS and Android platforms. The implementation contained basic components of any game and was constructed as a typical game application created with Cocos2d-x. This included a menu system and simple graphics that can be scaled to several screen sizes. The game uses an online map provider, which in the game can be selected but is hardcoded in the source files. As an example Nokia HERE map was used as the primary online map provider. In addition a map proxy server providing OpenStreet map tile had been used (Figure 12).

The objective of the proof of concept game is to find all orienteering flags, which have been randomly placed on the map when the game starts. The player symbol is visible all the time on the map so it easy to follow where the smartphone is located at the moment. A simple scoring system was used to indicate how many flags are to be found and how many have been already found. The game could be paused and continued. The new game starts always by the loading of the map tiles from the online map server and randomly placing orienteering flags on the screen.

The implementation demonstrated that the Cocos2d-x game engine could be used for developing a virtual orienteering game. The game engine provided necessary graphical functions that software can be run on various screen sizes and devices. The application part of the software had been made using C++ programming language that can be used in both Android and iOS environments without any major modifications. The proof of concept application used a single code base with necessary platform dependent modifications. These modifications were done in order to read hardware related information such as the magnetometer and device location information and platform dependent code was used to cover this part. The map part of the application has been done using general methods (such as sprites) provided by Cocos2d-x directly. Any platform dependent changes were not needed when implementing the map part.

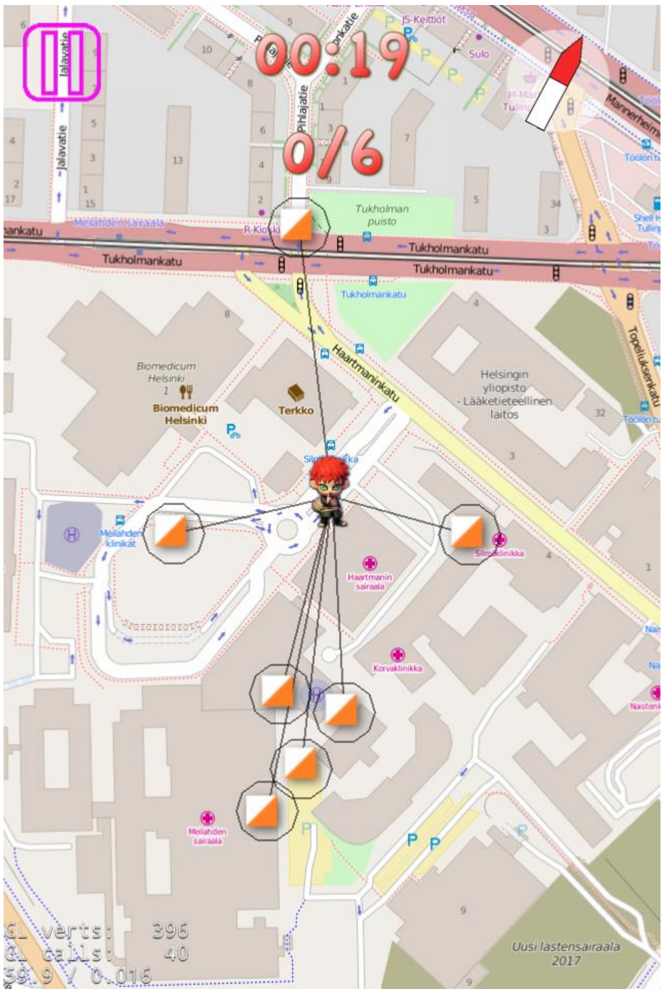


Figure 12. Screenhot of the map scene

All device manufactures provide their own tools and development environments to develop software to their devices. This would be a long road because several code based would be needed to develop. An application designer would need additional skills and effort so that a similar user experience could be achieved. Cocos2d-x suited well to program virtual orienteering game to several platforms and supports various other ways to implement portable applications.

During the implementation it was noticed how easily a virtual orienteering mini game can be done using Cocos2d-x game engine. Additional skills were needed to program with C++ programming language because that is not used by primary iOS and Android development environment. So if the programmer knows Android and iOS development there is a need to study Cocos2d-x in addition. Because Cocos2d-x uses a own pattern of developing the games there was no need to concentrate on developing C++ skills further and programming with C++ should not be a barrier.

Cocos2d-x is a commonly used game engine and lots of books and resources are available through the Internet. The learning curve is short and different kind of games can be programmed with low effort. During the study and the implementation the version of Cocos2d-x evolved from 3.2 to 3.6 and at the time being version 3.7 is out. This required some additional work with the implementation to upgrade the development environment now and then and some parts of the code. The most the adaptation work was needed in the Android system because C++ programming language cannot be directly used in the applications interfacing directly the location and magnetometer systems. The both hardware adaptations required in iOS and Android was kept as simple as possible and lots of error handling or performance optimization was dropped off.

The proof of game implemented the bare minimum for a virtual orienteering game and lots of additional features can be developed further. Perhaps the most important could be to use a real world orienteering map or a self-made artificial map of the terrain. There are also other applications where the results of this game could be used. Several geocaching or tourist applications could benefit of similar functions as implemented here and make the benefit of code once run everywhere.

8 Conclusions

At the beginning it was not sure how well a virtual orienteering game could be created using Cocos2d-x. The idea of using a freely available game engine that supports multiple device platforms was found to be an interesting subject to study. As virtual orienteering has not been widely implemented and it is not known if anyone has yet used a game engine for the implementation of this kind of an application this study was seen feasible. As a conclusion, a virtual orienteering game application can be implemented as a kind of serious game application to a smartphone.

Cocos2d-x is an open source game engine that can be expanded relatively easily to support functions required by virtual orienteering. Adaptations for iOS and Android environment to support functions needed by virtual orienteering were easy to implement. However that required studying a bit of the low level architecture of both systems and mechanisms how native applications are created with both systems. In the iOS system the benefit was to study or know some of the Objective-C environment so that the necessary adaptations could be made. The situation is the same with Android. In Android it is recommended to know how a typical Java based application is made prior to moving to program hardware related adaptations or native applications.

Creating the software-based bridge between different systems requires some knowledge of the C/C++ programming language because Cocos2d-x applications are created using that. However, comprehensive skills are not needed. During the implementation the version of Cocos2d-x evolved from version 3.2 to 3.6. That caused quite many problems with the implementation because some versions of the Cocos2d-x did not run well with both the iOS and Android systems. The most practical problems were faced with Android development and were caused by the bugs lying in the base Cocos2d-x. Problems such as using bitmap fonts caused the Android system to crash and the only solution was to wait for a newer version of Cocos2d-x. The Cocos2d-x 3.x versions are still sort of development versions and the programmer needs to study and follow an open source society quite tightly to overcome the problems caused by bugs in the game engine itself. In many cases bugs solutions were found in Web forums discussing Cocos2d-x.

At the end a proof of concept game could be made that could be run on multiple iOS and Android devices. The variation of Android devices is huge and finding support for all available and different screen sizes and hardware would still require additional programming and testing. Furthermore, the programming of the Android magnetometer was a challenge because very few devices used for testing contained a magnetometer. It was also found out that magnetometers are quite noisy causing the compass needle to be unstable and this could be overcome by adding low-pass filtering. In addition, acquiring the user location in an Android device had lots of variation. Even with the A-GPS enable devices it took sometimes a long time for an Android device to "lock" its position. To get the best user experience in the virtual orienteering it would require the use of high-end devices having good and supportive hardware. At the end of the day this is somewhat a drawback and sets the limitation of what kind of a device can be used for the purpose of virtual orienteering.

The Cocos2d-x game engine supports several other platforms e.g. Microsoft Windows Phone. The estimated workload to support Windows Phone or other platforms would be low. Most likely the approach would be similar with the Android environment. In that sense the use of Cocos2d-x looks even more attractive because the same application would be adapted relatively easily to support a Windows Phone.

The topic of the study was to find the answer to the research question how Cocos2d-x game engine could be used for implementing a virtual orienteering game. For the final conclusion the answer had been found. A virtual orienteering game using Cocos2d-x can be implemented easily but requires programming of additional hardware dependent adaptations. After necessary adaptations the actual Cocos2d-x application can be deployed and run on multiple different platforms using one single code base. Also Cocos2d-x can be seen a valid engine for other types of serious game development.

References

Android Developer pages. Available http://developer.android.com and http://source.android.com. [Accessed 22 Februaty 2015].

Apple (2015). *Apple Appstore*. Application Store. Available http://www.apple.com/fi/itunes/. [Accessed 26 Appril 2015].

Apple Developer pages. Available http://developer.apple.com. [Accessed 22 February 2015].

Bronson Gary J. (2010). *C++ For Engineers & Scientists*. Third Edition.

Cocos2d-x Developer Pages. Available http://www.cocos2d-x.org. [Accessed 22 February 2015].

Engelbert Roger (2013). *Cocos2D-x By Example Beginner's Guide*. Packt Publishing. Kindle edition.

Google (2015). *Google Play*. Application store. Available https://play.google.com. [Accessed 26 Appril 2015].

Google Maps for Ios, Available https://developers.google.com/maps/documentation/ios/ [Accessed 22 February 2015].

Häggman, Bjarne, Mäkinen, Matti, and Oikarinen, Erkki (1981). *Suunnista luontoon*. Suunnistajan käsikirja. WSOY.

Harvey, Francis (2008). *A primer of GIS, Fundamental Geographics and cartographic concepts*. The Guilford Press.

Hussain, Frahaan, Gurung, Arutosh, and Jones, Gareth (2014). *Cocos2d-x Game Development Essentials*. Packt Publishing. Kindle edition.

International Orienteering Federation, About Orienteering, IOF. Available http://orienteering.org. [Accessed 28 February 2015].

International Specification for Orienteering Maps, International Orienteering Foundation, IOF. Available http://orienteering.org. [Accessed 2 February 2015].

Iuppa, Nick, and Borst, Terry (2010). *End-to-End Game Development*.

Lonka, Eero-Antti (2012). *Pihakartan kuvausohjeet*. Suomen Suunnistusliitto.

Meier, Reto (2012). *Proferssional Android 4 Application Development.* Wronx.

Microsoft, *Bing Maps Tile System*, Harvey, Available http://msdn.microsoft.com. [Accessed 14 March 2015].

Mistra, Pratab, Enge, Per (2011). *Global Positioning System, Signals, Measurements, and Performance*, Ganga-Jamuna Press.

Muzykov, Kirill (2014). *Learning iPhone Game Development with Cocos2D 3.0*. Pact Publishing. Kindle edition.

Nahavadinpoor, Vandad (2013). *The iOS 7 Programming Cookbook*, O'Reilly

Nikulainen, Pekka, Vartiainen, Börje, Salmi, Janne, Minkkinen, Juha, Laaksonen, Petri, and Inkeri, Jukka (1995). *Suunnistustaito*. Suomen Suunnistusliitto.

Nokia, Map Tile Developer's Guide, Version 2.1.60.0, Available http://developer.here.com. [Accessed 6 April 2015].

Open Geospatial Consortium Inc. (2010). *OpenGIS® Web Map Tile Service* Implementation *Standard*, Version: 1.0.0. OpenStreetMap Foundation. Available http://wiki.osmfoundation.org/wiki/Main_Page. [Accessed 22 February 2015].

OpenStreetMap (2015). *OpenStreetMap*. Available https://www.openstreetmap.org. [Accessed 22 February 2015].

Rogers, Scott (2014). *Level UP! The Guide To Great Video Game Design*. Wiley. Kindle edition.

Schirme, Maximilian, Höpfner, Hagen, *Smartphone Hardware Sensors*, Mobile Media Group. Available https://www.uni-weimar.de/medien/wiki/images/Zeitmaschinen-smartphonesensors.pdf. [Accessed 1 March 2015].

Shekar, Siddharth (2014). *Learning Cocos2d-x Game Development*. Packt Publishing. Kindle edition.

SlackMoehrle, Ricardo, Justin, Nite, Kai, Minggo, Wenhai, Tony, Yingtao, Rao (2015). Cocos2d-*x Programmers Guide v3.3*. Available http://www.cocos2d-x.org/programmersguide/ProgrammersGuide.pdf. [Accessed 29 June 2015].

Sylvain, Ratabouild (2010). *Android NDK, Beginner's Guide*, Packt Publishing, Kindle edition.

Tufró Francisco (2013). *Developing Mobile Games with MOAI SDK*. Packt Publishing.

Van Diggelen, Frank (2009). *A-GPS Assisted GPS, GNSS, and SBAS*. GNSS Technology and Applications series.