OULUN SEUDUN
AMMATTIKORKEAKOULU

Mika Pesonen

MOBILE GAME FOR IPHONE: UNDERWATER

MOBILE GAME FOR IPHONE: UNDERWATER

Mika Pesonen
Bachelor’s Thesis
Spring 2013
Degree Programme in Software Engineering
Oulu University of Applied Sciences

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

Tekijä(t): Mika Pesonen
Opinnäytetyön nimi: Mobiilipeli iPhonelle: Underwater
Työn ohjaaja(t): Markku Rahikainen
Työn valmistumislukukausi ja -vuosi: Kevät 2013 Sivumäärä: 35

Tämä opinnäytetyö toteutettiin koska opiskelija halusi oppia kuinka suunnitella ja toteuttaa mobiilipeli Apple iPhone alustalle. Aihe opinnäytetyölle valittiin myös työnantajien näkökulmaa silmälläpitäen. Peliala tarvitsee nykyään enemmän osaavia ohjelmistokehittäjiä.

Tämä opinnäytetyö dokumentoi käytetyt työkalut, pelin suunnitelman ja suunnitelman toteutuksen ohjelmakoodissa. Työ käy myös hieman lävitse Cocos2D ohjelmointia.

Työstä käy ilmi että on yllättävän helppoa tehdä pelistä pelattava. Kuitenkin pelin viimeistely ja hiominen ottaa enemmän aikaa kuin aloitteleva pelikehittäjä luulee. On tärkeää että peli suunnitellaan mahdollisimman yksinkertaiseksi ja käytetään mahdollisimman paljon valmiita työkaluja. Tämä säästää kehitysaikaa ja kustannuksia.

Alkuperäinen tavoite tälle työlle oli saada peli julkaistua Apple App Storeen ennen opinnäytetyön julkaisua. Tämä ei kuitenkaan onnistunut aikamäärittelyssä tehtyjen virheiden takia. Peli on edelleen kehityksen alla ja arvioitu julkaisuaika on ennen 2013 elokuuta.

Lopputulos työstä tätä kirjoittaessa on se, että n. 70% pelistä on valmis. Pelimoottori on n. 99% valmis, mutta pelin sisältöä puuttuu yhä. Aiomme jatkaa pelin kehitystä ja korjata tämän ongelman.

Asiasanat: iphone, peli, apple, objective-c, ios, appstore

ABSTRACT

Oulu University of Applied Sciences
Degree programme in Software Engineering

Author(s): Mika Pesonen
Title of thesis: Mobile Game for iPhone: Underwater
Supervisor(s): Markku Rahikainen
Term and year when the thesis was submitted: Spring 2013 Pages: 35

The objective of this final year project was to learn how to design and implement mobile game for Apple iPhone platform. The topic for this thesis was also chosen from the basis that mobile gaming industry currently needs more developers.

This thesis documents the tools used, the design of the game and the final implementations of the design in code.

It turns out that it was easy to actually make a game playable, but it takes more time to hone it to release state. It is also quite important for the design to be as simple as possible and the use of the ready-made tools is almost mandatory. This saves development time and costs.

The original intent for this thesis was to release the game on Apple App Store before publishing this thesis. This goal was not completed. Errors in time estimations were made. Currently the game is still under development and the estimated time of release is before August 2013.

The final results at the time of writing this thesis is that the game is 70% completed. Game Engine is estimated to be 99% complete, but content is still lacking. In the future all of this will be fixed.

Keywords: iphone, game, apple, objective-c, ios, appstore

TABLE OF CONTENTS

GLOSSARY

App	Application software for mobile devices.
Apple	Apple Inc. Multinational corporation that designs, develops and sells consumer electronics.
iOS	First known as iPhone OS. Operating system used by Apple iPhone ja iPad products.
Objective-C	Programming language used in making Apps for iOS devices.
IDE	Sofware Application that provides all essential tools for programmers to do software development.
Virtual Machine	Simulation of a machine (usually another computer type)
Mac	Macintosh Computer
SDK	Software Development Kit

1 INTRODUCTION

In 2007 Apple Inc unveiled a new Apple iPhone. Phone featured a touchscreen input and had only one physical button. Steve Jobs called this phone “revolutionary”. In a sense it was. Other manufacturers had revealed touchscreen phones, but could not amass such hype and interest for the device that Apple did. iPhone went to become an overnight success when it was released for public to buy in November of 2007.

iPhone’s market share has climbed up to 25.1% worldwide in 2012. Many people are using the phone. Research also suggests that iPhone users are more willing to spend money on apps than any other group of smartphone users (Elmer-Dewitt, Philip. 2011. Date of access 22.3.2013).

With this information it makes sense for most single or small developers to at least consider developing for iOS (first known as iPhone OS). From this consideration the game idea for Underwater was born.

I and my friend (who studies at Oulu University of Applied Sciences / Business Information Technology) had long discussed about making and releasing a mobile game. Moreover, both of us needed a topic for our bachelor’s thesis.

Making a game seemed like a good idea for several reasons. First of all I had strong background in Objective-C. Secondly my friend had some background in developing web graphics and layouts. Thirdly it might improve your job opportunities. Many Finnish mobile companies are currently looking for knowledgeable game developers and graphics artists.

In this thesis I will design and implement a game for iPhone, which on a later date will be released to the public in Apple App store. Underwater will be a simple arcade type game where player controls a squid to complete goals as fast as possible.

2 TOOLS AND SDKS

Some tools for achieving the goals of the thesis were chosen by necessity and others by choice. There is not much room for maneuvering in coding tools if you plan to actually get your software approved for Apple App Store. On the other hand the choice in software development kits, graphics editing programs and digital audio workstations are all for the teams to make.

Most of the software used were free or received through student license programs. This kept the cost of investment down and meant that team is actually bound to make money if anybody actually uses our app.

Hardware can be a lot more expensive than software. Coding for iOS means that you have to have at least 1 Macintosh computer that runs the newest / relatively new version of the OS X Operating system. Only IDE that is officially supported by Apple is Xcode and it only runs on OS X operating system (iOS Dev Center, 2013. Date of access 15.2.2013.). Fortunately for this project we already had a first generation Mac Mini available for our coding efforts.

2.1 Hardware and Operating System

Apple goes to great lengths to ensure that OS X is able to be installed only on their hardware. Basically you need a Macintosh computer to be able to install the OS and tools that Apple provides for software development (OS X Mountain Lion Technical Specification, 2013. Date of access 17.2.2013).

These rules can be broken by installing OS X on a virtual machine. This procedure is however not officially supported by Apple. It is still the safest bet to just open up your wallet, and buy either a second hand or a new Macintosh computer.

2.1.1 Mac Mini

Mac Mini is a small form factor desktop computer. It is the cheapest of all Macs and offers you just the basic system unit of a computer in a small package. Only the unit and electronic cords are included in the package. Peripherals need to be bought separately or acquired some other way (Mac Mini Technical Specifications, 2013. Date of access 17.2.2013).

Mini is in no way the ideal development computer. It is relatively slow and does not support multiple monitor arrangements. Also, it can run into problems when using original PC hardware as peripherals. Mainly it is just something that gets the job done cheaply.

For this thesis we used 2011 generation Mac Mini with a PC keyboard and a mouse. These peripherals were Logitech UltraX Keyboard and Microsoft Intelimouse 3.0. We compensated for the lack of support for multiple monitor arrangements by using 37 inch LCD-TV as a monitor. This gives somewhat more desktop space than normal 22-24 inch monitors would.

2.1.2 OS X

OS X is Apple's own Unix-based operating system. It was first released on March 24, 2001. To this date there have been 9 different versions of it. Cheetah (v10.0) being the oldest and Mountain Lion (v10.8) being the newest. All of the releases are named after big cats.

The operating system is designed to exclusively run on Macs. It can also be run in virtual machines that simulate Macintosh computer or by hacks that enable it to be run on PC hardware. Both of these ways are not officially supported and it is best to avoid them.

The operating system is mainly GUI-driven although Unix-style console is still offered. This can be useful for the users who are familiar with Unix-systems. Normal user will get by just using the graphical tools that Apple has included with the operating system.

For software development it is generally adviced to get the newest or at least a relatively new version of the OS X. This stems from the fact that the newest updates of the developer tools are only offered for newer OS X releases. Developer can get by with older versions of the tools, but if planning to support newer Apple hardware it is best to upgrade to the newest operating system version

(Xcode, 2013. Date of access 11.3.2013). These upgrades usually cost between 20 to 30 €.

For this thesis we used the newest OS X release, Mountain Lion (v10.8). The idea was to support iPhones 4 and newer ones. This meant that developer tools needed to be able to simulate the newest iOS enviroments. Upgrading the operating system was relatively simple. It can be done from the App Store application included in all OS X releases. After the upgrade, developer tools can be installed the same way. We encountered no errors in this process.

2.2 iOS (previously iPhone OS)

iOS is a mobile operating system for Apple’s handheld devices. iOS was originally derived from OS X and shares the same foundation. iOS is a fairly closed system and will only run on Apple’s own devices. It is not licensed to other manufacturers. These restrictions have not prevented it from succeeding, far from it. It currently holds 14.9% share of just the smartphone mobile operating systems. (Graziano, Dan. 2012. Date of access 18.2.2013).

Most of the devices that run iOS are touch devices, but the OS has been in some occasions modified to accept different kind of input. For instance iPod was known for its scroll wheel that was used to control the devices in sequence with some buttons (iPod nano 5th generation, 2013. Date of access 18.2.2013). In the current generations these devices use a touchscreen too. This is a good thing for developers, as they can now support only one type of input.

Interaction with the touch devices is made possible by touch gestures and interface control elements. Touch gestures are used mainly for rotating, zooming, scrolling, etc. The interface control elements offer activable elements that an user can touch to trigger different kinds of actions.

Underwater does not use most of these features. Instead of using original UI-elements from Apple, we use customized “virtual joystick” type of configuration for command and control. We do however, use the phones touchscreen sensor to figure out where the user is touching.

2.3 Xcode

Xcode is an IDE for developing software for Macintosh Computers running OS X and iOS mobile devices. It also integrates many software development tools, and is therefore essential for almost any native software development done on a Mac. Xcode was first released on 2003 and has progressed to the version 4.6 (released on January of 2013). Xcode is offered free of charge for two newest versions of OS X, Lion(v10.0.7) and Mountain Lion (v10.0.8).

Although the software is free, developing for iOS devices with it is not. Running a non-released software on a mobile device requires provisioning profiles and keys that developers are only able to get after subscribing to a $99 per year Apple Developer Program. You can run any non-released software on a iOS Simulator, which can be used to simulate iPhone and iPad environments without paying for the subscription. (iOS Developer Program, 2013. Date of access 18.2.2013)

2.4 Cocos2D SDK for iPhone

Cocos2D for iPhone is a Software Development Kit for building 2D games for iPhone and iPad. SDK is originally based on a python framework, but is instead written in Objective-C language. For distinction the name now includes “for iPhone” at the end. The SDK project is open source driven, but it is not released under any big open source license. It uses its own “free do to anything” license (Cocos2D for iPhone – Licence, 2013. Date of access 18.2.2013).

Framework offers a huge array of features specifically meant to make coding of 2D games as easy as possible. This lessens the development time and the framework code is usually better tested than in-house code (Cocos2D for iPhone – About, 2013. Date of access 18.2.2013).

2.5 Objective-C

Objective-C is the primary programming language when developing software for OS X or iOS. It is based on a C-language, but also adds some of the features from Smalltalk programming language. It is mainly used when developing software for Apple’s products. This means that there generally are fewer libraries and other supporting tools made specifically for it. Some of these problems are remedied by the high quality of the tools that Apple itself offers.

Language is object-oriented and relies on messaging the objects rather than calling them. It is generally a high-level language, although it is entirely possible to write low-level code in it if needed (About Objective-C, Date of access 20.2.2013). Syntax is derived entirely from the Smalltalk language.

2.6 Mercurial

Mercurial is a distributed source control management (SCM) tool. It is used to add, track and remove versions of files in software repository (About Mercurial. Date of access 20.2.2013).

Using SCM makes it possible for several developers to work on the same code-base. SCM can track all changes to the files and resolve conflicts between files. (Why Version Control, 2011. Date of access 20.2.2013). It is also good for safety reasons. Most of the SCM servers will also backup data daily, so the work is not lost in the event of hardware or software failures

When developing Underwater, we used source control to allow both of the team members to work on different times of day. Other person can track what changes there has been while he was gone, and instantly know if some file has been modified.

2.7 Tiled Map Editor

Tiled is a graphical tile map editor that uses XML-format called TMX to represent the maps. It currently supports orthogonal and isometric maps and uses tileset (a collection of tile images) to paint over the tiles with chosen images (Tiled Map Editor. Date of access 2.3.2013).

Cocos2D SDK for iPhone has a native support for TMX-maps, and is convenient if the developer wants to avoid making their own map format.

3 DESIGN

Underwater (Picture 2) was designed to be a easy to play, easy to pause, easy to quit game to kill time on a mobile phone. Game was designed freely, and we mainly used face-to-face meetings to achieve the final design. We did not follow any set design guidelines.

First we had to choose a platform to make our game for. This was surprisingly hard because there are several good reasons backing up iPhone, Android and PC development. PC development was dropped early on. We felt that making even an indie game for this environment was going to take too long, and there are no guarantees that your game will succeed getting into Steam or some alternative digital distribution platform. Digital distribution is almost mandatory for any small developer. It makes buying the game easy and in the best case scenario will also advertise for you by featuring your game.

On the case of iPhone vs. Android it was decided in favor of iPhone mainly because Mika Pesonen had years of previous experience in Objective-C, and felt that the experience would help to get the job done as soon as possible. There were other factors too. For instance, research indicates that iPhone users generally are more likely to buy software. Also, it is generally easier to make sure that your software works on every available device when developing for Apple. Android will always have issues because there are such huge amounts of different hardware running Android.

Next issue was finding an idea for the game that was easy to play for the end-user and not too time consuming to make. We had several meetings on the subject and in the end had a list of 16 old games and couple of original game ideas. We did not consider the original platform for a game, and the list contained anything from old Atari games to newer flash games.

The most interesting one was a flash game playable in any internet browser. It is called Owl Spin. In Owl Spin you control an owl, try to avoid walls while owl's wings are constantly spinning and try to complete a stage in the shortest time possible. We felt that a game like this would be a great idea for a mobile phone, and as far as we could find there was no such game for the iOS platform. The final design was soon ready.

3.1 Setting

We planned several different settings for the game. Everything from snowy worlds to beaches and palm trees were considered. We did not want to rip off Owl Spin completely so we did not want our protagonist to be a bird. Still, the game mechanics needed a character whose limbs can rotate while moving.

After a long meeting we decided that our main character would be a squid (Picture 1). Squids have tentacles that can in our 2D world spin like the wings of a bird. Also it made sense to set the whole game world to underwater environments, as in nature squids are found there.

3.1.1 Story

We opted to not write any heavy backstory for the game. Mainly this was done to save development time. More resources were thus geared towards making the game actually look good graphically.

Game's protagonist and only playable character is a nameless squid. Player will control this squid through various challenges to receive the ultimate reward: getting home alive.

Strange things have been happening in the squid's watery home called Underwater. Once so peaceful place has been invaded by sharks. Sharks tend to eat squids and make moving in the world of Underwater harder that it should be. Sharks are not smart by any measure, so our squid still has a change to complete his task.

3.1.2 Graphics & Animations

All the graphics would be made in PNG format for two reasons. First of all Apple encourages to use this lossless compression format. Secondly every major imaging tool can open and modify PNG.

For resolutions we opted to support two kinds of resolutions. 320x480 which is the screen size on older iPhone models (iPhone 3GS and older) and 640x960 which is the newer retina resolution (iPhone 4 and newer). With iPhone 5 you get 640x1136 resolution, but this does not really make a difference. iPhone 5 can use the same size elements as the normal retina resolution.

Animations would be made from several PNG images, which are changed in a quick looping sequence. For instance, the movement of the squid would be animated in this manner. Animations will not always be totally fluid to save development time. We could have simply used animated GIF format to achieve this, but we wanted to have possibility to change animation images on the fly. We decided to use animation layers offered by the Cocos2D SDK.

3.2 Controls

Controls were planned to be as simple as possible and we decided to avoid inputs that are triggered by turning and shaking the phone. Instead, all of the inputs would be made available by on-screen touch controls.

3.2.1 Menus

Menus are made by menu controls offered by Cocos2D SDK. These are items that contain text, and can trigger actions when touched. For instance, these actions can be the opening of new menu screens, starting the game or exiting from the game. Fonts and colors for these items can be customized.

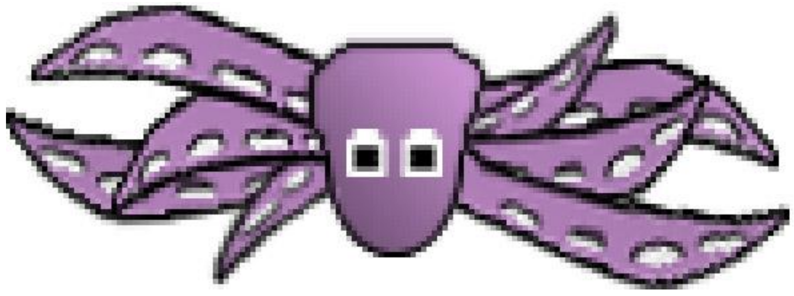
We decided to make menu text as big as it is possible. Too small touch controls make it harder for people to use them properly.

There will be 3 different menu items. Start, Settings and Quit. Settings will open another menu screen where user can choose to enable or disable different settings for the game.

3.2.2 Game

The in-game protagonist will be the only thing controlled and there will be only one simple control, a virtual joystick. Virtual joystick is a virtual remake of a normal joystick control. It will be activated when user touches the virtual joystick's area on the screen and it will read input until user raises his finger off the control. This allows fluid movement in 360 degrees.

There is a downside to this approach which cannot really be avoided. There is no real feel to the joystick. User cannot know without looking the game if the joystick is actually reading his inputs. We could have made phone vibrate whenever the finger enters or leaves the control area, but decided against it. Constant vibrating would most likely get annoying over time.



Picture 1. Early model of the squid character

3.3 Maps

Game will be using TMX-map format. This is because Tiled Map Editor is a well established tool, free and several SDKs have native support for it. We can thus skip creating our own map format and editor.

Map tiles will be the size of 512x512. The reasoning for this is that with limited memory capacity available for mobile phones it is impossible to load one huge map. Cocos2D SDK will automatically load the tiles for viewing, and unload them when they are not viewed on the screen.

Maps will also control the movement of the game’s antagonists, the sharks. There will be a waypoint system created into map’s tiles that tells the game where sharks will go. Thus, we can create static patrol routes for our antagonists.

Because tiles are basically just images, we need to do some collision detection for them. In this case it is not possible to use Tiled’s own collision detection system. For our purposes two tiles colliding is not nearly accurate enough. Instead we will be doing pixel perfect collision detection on the tile where our protagonist currently resides. This saves processing power because you only check for one area at a time.

Collision detection will be determined by colors. If the pixels are transparent, then we can pass through. If not, the squid will run into a wall or to a shark and die.

3.4 Campaign sets

We tie our maps to a campaign set. One campaign set can in theory have several hundreds of maps. To unlock the next map in order, the user needs to complete the newest unlocked map.

Campaign sets were designed because we needed an easy way to add more maps for the game at a later date. Originally there will only be 20 maps in one campaign, but we will make more available in the future.

3.5 Finances

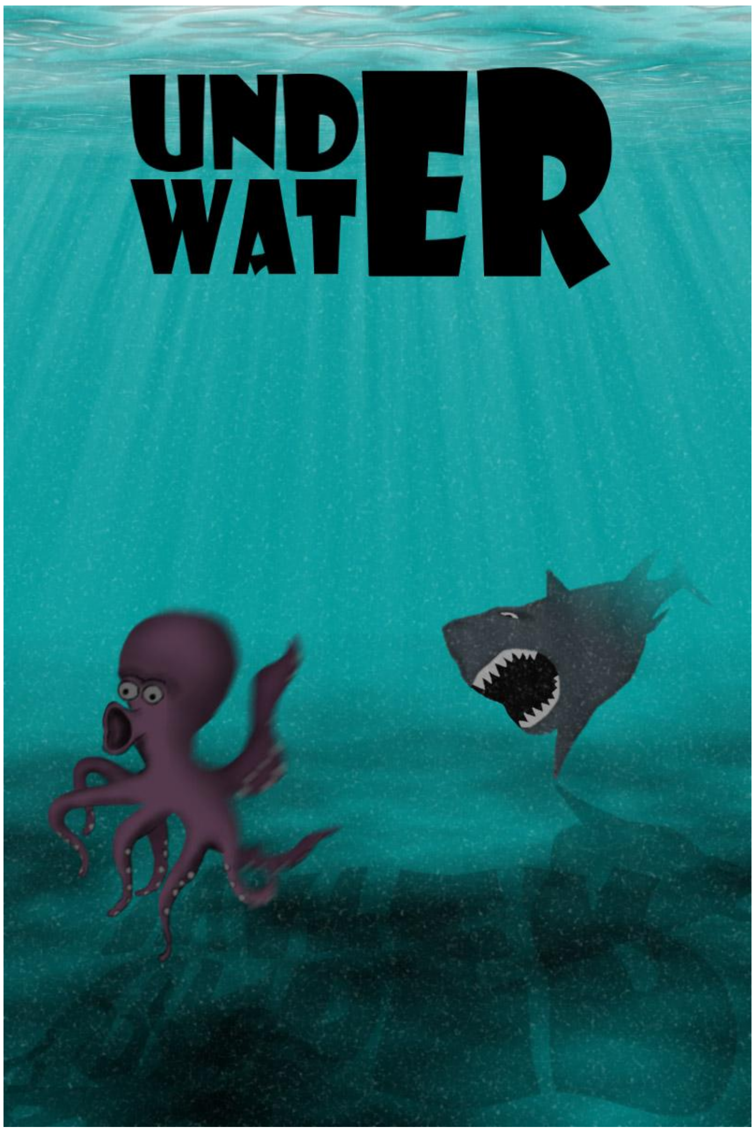
The cost of making the game will be paid from our own pockets. This amount will be relatively low. The money will mainly go to the Apple Developer Program and software student licences.

We will try to make money from the game by subscribing into Apple’s iAdd program. The game will be made available free of charge, but it will show advertisements on the menu and map selection screens. We might also release ad free version which will cost under a euro.

However, the main goal for this game is not to make money. Rather we would like to show that we can design and implement a successful game for the iPh-one.

4 IMPLEMENTATION

At the current date game is not yet 100% completed. There are some minor issues with collision detection, and we still lack some of the image and map resources we need for a complete game. However, this chapter describes as closely as possible how our design choices currently show up in the code.



Picture 2. Finished splash screen art

4.1 Menus

Menus are contained in menu classes that can when needed return an on-screen graphical presentation called a scene (Code example 2). Menus are all derived from Cocos2D CCLayer class (Code example 1).

Menu items and text labels are initialized and added to the graphical layer of the class. After initialization the menu scene can be made visible or unloaded if needed.

```
@interface MainMenuLayer : CCLayer
{

}
+(CCScene *) scene;
```

Code example 1. Menu interface

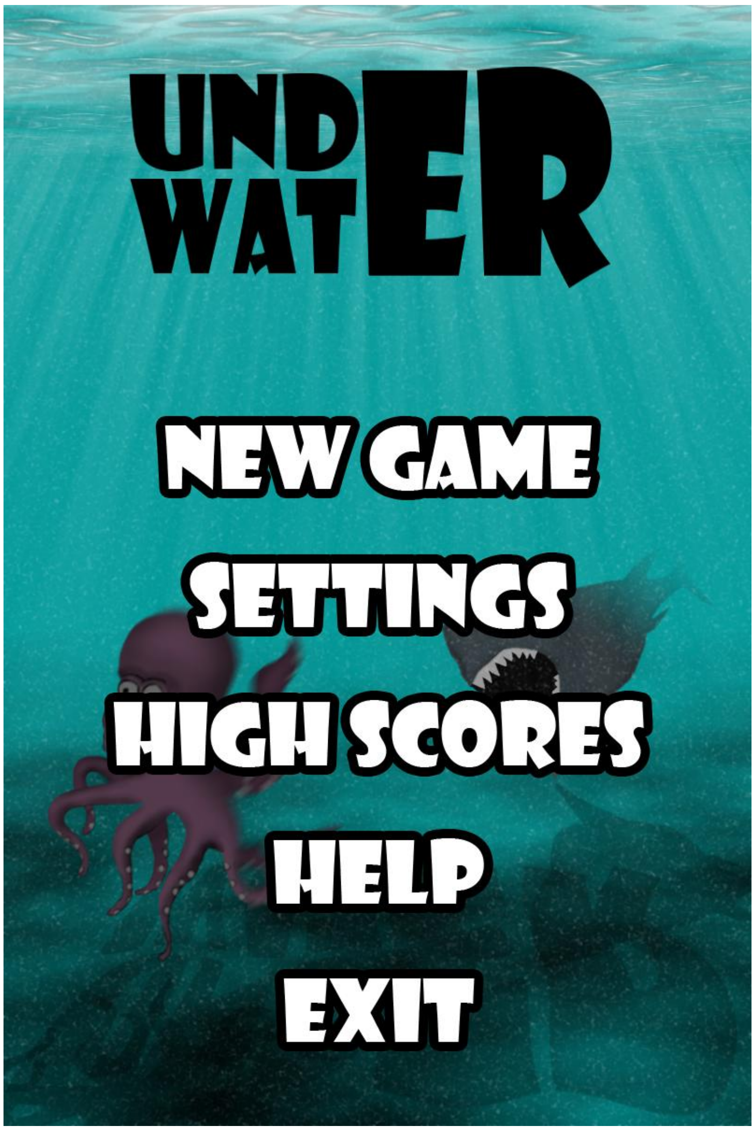
```
+(CCScene *) scene;
{
	// 'scene' is an autorelease object.
	CCScene *scene = [CCScene node];

	// 'layer' is an autorelease object.
	MainMenuLayer *layer = [MainMenuLayer node];

	// add layer as a child to scene
	[scene addChild: layer];

	// return the scene
	return scene;
}
```

Code example 2. Menu’s scene implementation

4.1.1 Main Menu

Main menu has 2 different menu options. ”Play” and ”Settings” (Picture 3). Touches for these options on the menu have been assigned 2 different actions. Clicking ”Play” will trigger a method that will unload the main menu and proceed to the map selection scene. Clicking ”Settings” will unload the main menu and proceeds to settings menu scene (Code example 3). The main menu scene also contains a label for the game titled ”Underwater”.

```
-(id) init
{
        if( (self=[super init]) ) {

                // create and initialize a Label
CCLabelTTF *label = [CCLabelTTF labelWithString:@”Underwater"
fontName:@"Marker Felt" fontSize:54];

                // ask director for the window size
                CGSize size = [[CCDirector sharedDirector] winSize];

                // position the label on the center of the screen
                label.position =  ccp( size.width /2 , size.height/2);

                // add the label as a child to this Layer
                [self addChild: label];


                //
                // Game Menu Items
                //

                // Default font size will be 28 points.
                [CCMenuItemFont setFontSize:28];

CCMenuItem *itemPlay = [CCMenuItemFont itemWithString:@"Play"
block:^(id sender)
                {
        [self scheduleOnce:@selector(makeTransitionToPlay:) delay:0];
                }];

CCMenuItem *itemSettings = [CCMenuItemFont itemWithString:@"Settings"
block:^(id sender)
                {

        [self scheuleOnce:@selector(makeTransitionToSettings:) de-
lay:0];

                }];

CCMenu *menu = [CCMenu menuWithItems:itemPlay, itemSettings, nil];

                [menu alignItemsHorizontallyWithPadding:20];
                [menu setPosition:ccp( size.width/2, size.height/2 -
50)];

                // Add the menu to the layer
                [self addChild:menu];

        }
        return self;
}
```
Code example 3. Main menu’s init implementation



Picture 3. Finished main menu

4.1.2 Settings Menu

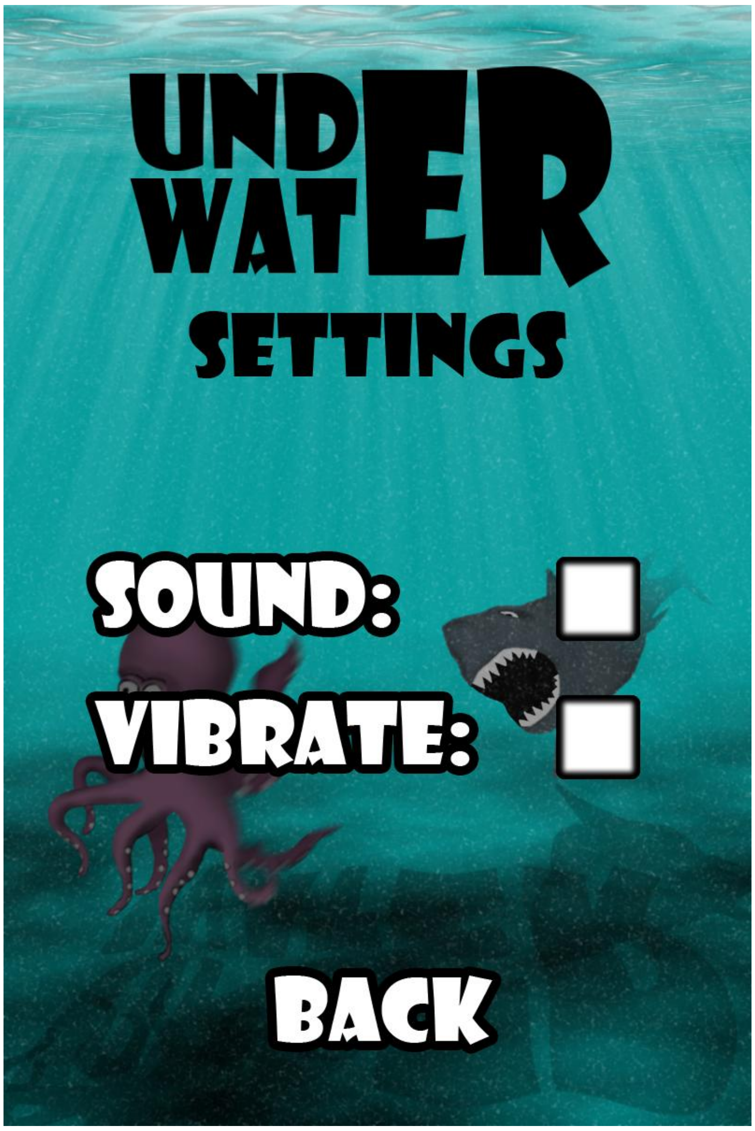
The settings menu (Picture 4) code is basically the same as main menus. Also, there are very little differences in any other menus that might be added in the future. The only major difference in this case is that for sound and vibration options we use custom-made class called CCMenuItemCheckBox.

Checkbox is derived from CCMenuItem class and as its name suggests it will work as a regular checkbox. It takes same arguments as CCMenuItem and uses static checkbox images to show checked CCSprite image on the screen when user touches the menu item and unchecked when user touches the menu item again.

We will also implement category for handling our new CheckBox to CCMenuItemFont class. This category will implement the method called itemWithStringAndChecked (Code example 4).

```
CCMenuItemCheckBox *willVibrate = [CCMenuItemFont
itemWithStringAndChecked:@"Settings" : NO : block:^(id sender)
{

}
```

Code example 4. Custom CheckBox initiation



Picture 4. Finished settings menu

4.2 Map system

Our map system has two parts. There is a MapSet class for our campaigns. This is basically a collection of maps that in-game are shown as campaigns. We did not want to make this unnecessarily complicated, and currently MapSets are loaded from a simple text file (Code example 5).

```
MapSet *campaingSetA = [[Mapset alloc] initFromFile:@”SetA.txt”];
```
Code example 5. CampaingSet initiation

The MapSet text file is composed of 2 parts. First row of the file will be the campaign title. Our first campaigns name is currently “Getting the spin on”. After that, all the rows are just plain filenames for the maps. TMX files already hold title and description of the map that user can view before playing the map. Thus, there is little need to complicate MapSet text files (Code example 6).

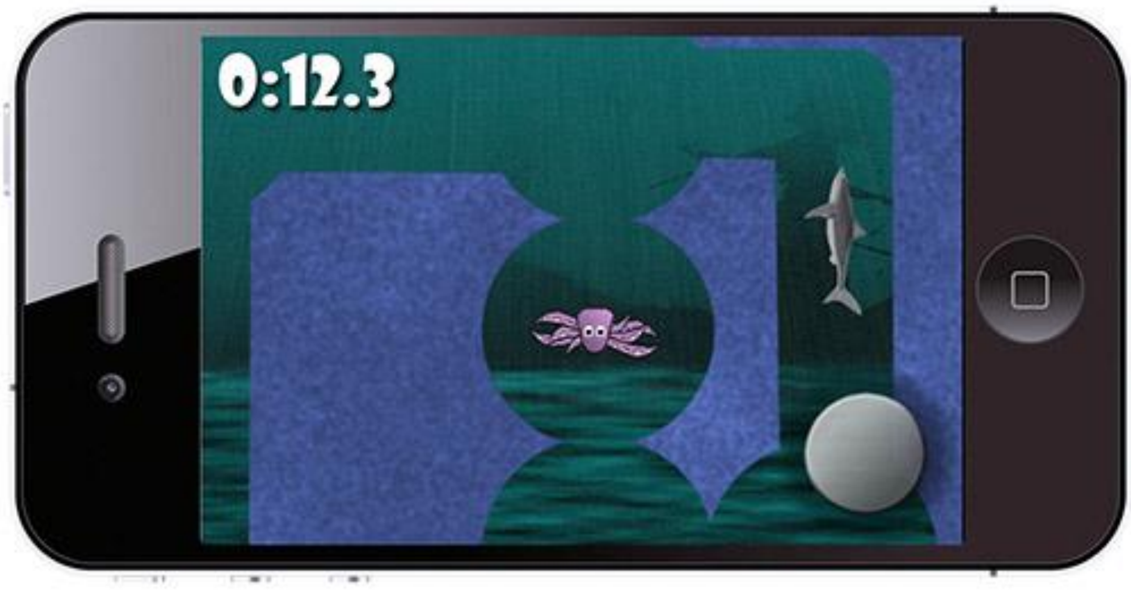
```
Getting the spin on
Spin1.tmx
Spin2.tmx
Spin3.tmx
```
Code example 6. MapSet text file

When user chooses a map (Picture 5) from the selection screen, the game will preload the map and show its title and description (Code example 7). If user chooses not to play (closes the description screen), the map is instantly unloaded.

```
self.tileMap = [CCTMXTiledMap tiledMapWithTMXFile:mapNameString];
```
Code example 7. Tilemap initiation

If user chooses to click the “Start” button on the description screen, the preloaded map file will be passed on to the main game scene (Code example 7). After this has been done, the game will change from map selection scene to the game scene.

```
-(void) startGame
{
        if(!self.tilemap)
        {
                return;
        }

GameScene *scene = [GameScene alloc] initWithMap:self.tilemap];

[[CCDirector sharedDirector] replaceScene:[CCTransitionFade
transitionWithDuration:1.0 scene:scene withColor:ccWHITE]];

}
```
Code example 7. startGame method implementation



Picture 5. Map geometry in test build

4.3 Collision detection

Collision detection works by firstly detecting on which tile or tiles our squid is currently occupying. After it has found this out, it will compare the pixels of the area which our squid occupies. This is done by hasCollidedWithMap method. hasCollidedWithMap will also map our squids global coordinates to the local tile coordinates. To save processing power we will only run collision detection once every 2 loops (Code example 8).

```
if(loopcounter % 2 == 0)
{
        if([gameMapObject hasCollidedWithMap : playerObject.position])
        {
                [playerObject setDead];
        }
}
```

Code example 8. Collision detection on game's updateloop

If pixels are all transparent on our tiles, we have not collided to anything and the method will return false. We do not need to do more complex detection because we do not care what we collided with. Every collision in the game will lead to squid dying and user having to restart the map.

Inside hasCollidedWithMap we also have a special check for colliding with enemies. Sharks are not really map objects in the sense that map coordinates are only used for pathing sharks routes. Still, we pass our list of the enemies to map object when map is initialized. This means we can have all of our collision detection in one place (Code example 9).

```
if(enemyListArray)
{
//Map enemy global coordinates to tilecoordinates and compare if col-
lided with squid.
        for(id enemy in enemyListArray)
        {
CGPoint tilePosition = [self mapToCurrentTileCoordinates :  ene-
my.position];
                if([self collidedWith : tilePosition])
                {
                        return true;
                }
        }
}
```

Code example 9. mapObject collision detection for enemies

4.4 Enemies

Enemies in the game are expressed by the Enemy class (Picture 6). Enemy class is quite simple. It only knows two things. What animation it should be playing when moving, and what is its global position. This is because our enemies follow a static path and cannot die during the game. Our squid will just have to dodge them.

Enemies are created from TMX map files when the map is originally initialized (Code example 10). The map has xml-information that tells freshly initialized enemy objects 2 things. Starting position on the map and the route it will statically follow for the duration of the map.

```
if([gameMapObject hasEnemies])
{
        for(id enemyRoute in [gameMapObject enemies])
        {
                Enemy *newEnemy = [Enemy alloc]
initWithRoute:enemyRoute];
```

```
                [EnemyListArray addObject : newEnemy];
        }
}
```
Code example 10. Creating enemy objects from map data

Currently the animations in Enemy objects are static (all of the enemies are the same kind of enemies). This could be easily changed by writing a new init method for our Enemy class.

Picture 6. Current shark model

5 CURRENT STATUS

Currently Underwater is approximately 70% complete. Minor issues in the game code are being eradicated by testing and we add new content weekly. Mainly this work consists of adding more maps, replacing placeholder graphics and honing the overall appearance of the game.

Outside of the game engine and the content, we still need to implement our money making tool, the iAdd system. This is fairly well documented on Apple's own site and should not pose a problem. iAdd will be probably the last feature we will add to the game.

We have not set a release date for the game yet. The main thinking behind this is that ”It is done when it is done”. We do not want to force ourselves to deliver a piece of gaming software that does not work properly.

After everything is done, we will be uploading the finished game binary to the App Store for Apple's review process and wait for a response. Estimated time to finish the game with current time we have allocated is between 2 to 3 months. This means that the real development time will be something between 5 to 6 months.

5.1 Game engine

All of the game engines features are implemented by now. There are still some bugs to hone out and it is entirely possible that we will encounter more while running tests.

Currently known issues:

- Collision detection will sometimes fail while colliding with a shark model
- Collision detection is not accurate enough for the most complicated patterns of walls
- Map selection will not always unlock the next map when the last one is completed
- Point calculation for objectives completed needs a better algorithm

If content takes longer than expected to deliver, and everything on the game engine’s part has been fixed, we might consider adding some new features. There have not been any discussion to decide this matter, but we consider it a real possibility.

5.2 Content & graphics

What we are mainly missing is content. This is because we designed the game mechanics to be as simple as possible. Thus making the game engine took less time than we calculated. With only one graphics artist doing this part-time we are running low on content and high on codebase.

After all of the known bugs have been squashed, we will both be doing mainly content. Graphics artist can then focus on the graphics and I will be making mainly maps.

Missing content:

- 10 maps
- Approximately 20% of the static images
- Most of the animations

6 CONCLUSION

Bachelor’s thesis process started in November 2012. We (me and my friend) had a meeting in which we decided that my thesis would be about designing and implementing the game as a whole. My friend would be doing his own thesis on graphical interface for this game. This split was convenient as I would be doing all of the programming and he would be doing all of the graphical elements and some of the map design.

For my part, the process in itself was quite smooth. I have years of experience of game development on a hobby basis. What felt most difficult was making enough time. I knew at the start that I would be working my 8 hours per weekday job, attending school and doing this thesis. From that start point I actually managed to pull through quite well (A fact which hopefully this document will show).

There obviously were some problems too. This is apparent from the fact that game is yet to be released. I estimate that without other interference it would still have taken us 3 months of full-time work to complete the game. It is quite time consuming to make games. Despite this I would still call the project a success.

I had not made a full game for mobile platform before, and this presented some minor difficulties. My main platform of game development before this project was PC, and the difference in available hardware processing power and memory capacity is huge. I had to deal with the problems with more efficiency than I would have in a PC enviroment.

Despite the differences in hardware, I still would see iOS as a good platform to start your game development. This is because the games do not need to be overly complicated. There is actually quite little room to work with small touchscreen and current hardware capacity. You can easily make the game unplayable on the device it is intended for.

My main points would be, keep it simple, keep it small. Also, from this perspective I would advice everyone to actually use Cocos2D for iPhone or some alternative framework to help with development. This saves a huge amount of development time.

I myself learned a lot from examples and guides posted on the different parts of the Internet. Getting information about the subject was not hard. I also gained valuable knowledge about tools like Xcode and Tiled Map Editor. My feelings are that this project has improved me not only as a game developer for iPhone, but as a game developer for every platform.

I would say that the project was a good learning experience and succeeded well. I learned a lot, my friend learned a lot and we are one thesis richer in information. With our skills we made this game and will soon be releasing it to the world. I am happy with the results.

BIBLIOGRAPHY

About Mercurial. http://mercurial.selenic.com/about/. Date of access 20.2.2013.

About Objective-C. http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html. Date of access 20.2.2013.

Cocos2D for iPhone – About. http://www.cocos2d-iphone.org/about. Date of access 18.2.2013.

Cocos2D for iPhone – License. http://www.cocos2d-iphone.org/wiki/doku.php/license. Date of access 18.2.2013.

Elmer-Dewitt, Philip. 2011. Apple users buying 61% more apps, paying 14% more per app. http://tech.fortune.cnn.com/2011/07/11/apple-users-buying-61-more-apps-paying-14-more-per-app/. Date of access 22.3.2013.

Graziano, Dan. 2012. Android Projected to Own the Smartphone Market for the Next Four Years, http://bgr.com/2012/12/04/mobile-market-share-2012-android/. Date of access 18.2.2013.

iOS Dev Center, 2013. https://developer.apple.com/support/ios/ios-dev-center.html. Date of access 15.2.2013.

iOS Developer Program, 2013. https://developer.apple.com/programs/ios/. Date of access 18.2.2013.

iPod nano 5th generation, 2013. http://support.apple.com/kb/ht1353#iPod_nano5G. Date of access 18.2.2013

Mac App Store – Xcode, 2013. https://itunes.apple.com/us/app/xcode/id497799835?mt=12. Date of access 11.3.2013

Mac Mini Technical Specifications, 2013. http://www.apple.com/mac-mini/specs.html. Date of access 17.2.2013.

OS X Mountain Lion Technical Specification, 2013. http://www.apple.com/osx/specs/. Date of access 17.2.2013.

Tiled Map Editor. http://www.mapeditor.org/. Date of access 2.3.2013.

Why Use Version Control?, 2011. http://soundsoftware.ac.uk/why-version-control. Date of access 20.2.2013.