Mobile Web Based Mini Games for Locatify

Final Report

Autumn 2013

Eric Jon Nielsen
Jón Rúnar Helgason
Þórður Hermannsson

Teacher: Hallgrímur Arnalds
Instructor: Árni Hermann Reynisson
Examiner: Stefán Freyr Stefánsson

T-404-LOKA
Reykjavík University

This document is a final report for the final project Mobile Web Based Mini Games for Locatify. The project was done in the course T-404-LOKA from Reykjavík University in order to qualify for a BSc degree in Computer Science. In this course companies submitted project proposals that students then chose to work on for one semester or 15 weeks.

Table of contents

Introduction

The project that we chose was from the company Locatify ltd[1]. Locatify is a privately held company founded in November 2009 in Iceland by Leifur Björn Björnsson and Steinunn Anna Gunnlaugsdóttir. Locatify has developed an online system, Creator CMS[2], where users can create and publish their own treasure hunt and smartguide applications for smartphones. Locatify has also made TurfHunt[3], a location-based game for smartphones. These two products are well established and have been on the market for some time.

In TurfHunt players compete in teams to reach a final destination in the shortest amount of time. Each treasure hunt has a set of locations defined by a GPS coordinate that teams must visit. Once on each location they are presented with a challenge. Upon completing the challenge the players receive the next location to visit.

Currently TurfHunt has only 2 types of challenges, a memory game and question and answers quiz. The memory game is a 3 by 3 grid of square pictures that the player taps on and tries to find successive pairs. The question and answers quiz is a simple multiple choice game, a question is posed and the player selects one of 3 answers given.

Locatify wanted to offer more types of challenges and so they proposed this project on which we embarked.

[1] `http://locatify.com/`
[2] `http://locatify.com/creator-cms/`
[3] `http://locatify.com/turfhunt/`

Project Description

Our final project was to design standalone mini games for Locatify that could be integrated into their app TurfHunt to be played at locations during a treasure hunt game. Our games were not supposed to be solely designed for TurfHunt, instead they were supposed to be general stand-alone games and have the ability to be put in any other apps Locatify may make in the future as well. Another goal was to be able to deliver the games in such a way that they could easily be operated on and modified by the Locatify team. A visual representation of where our mini games fit into the project can be seen in Figure 1. These mini games had to meet a couple of requirements so that they could be used and extended by the staff of Locatify after our work was finished.

Figure 1: A visual representation of how our mini games fit into the Locatify architecture.

Requirements

There were several requirements our project and mini games had to fulfil which we list below.

Technical Requirements

Each game had to be written in HTML5[4] with additional logic and user interface code written in JavaScript and CSS. Additionally if our games needed to utilize any components of a mobile phone such as GPS or accelerometer we had to use Phonegap[5], a framework for wrapping HTML5, JavaScript and CSS to native apps on multiple kinds of mobile devices. This focus on mobile platforms meant that we had to focus on making our mini games work on multiple mobile platforms. We chose to focus on Android 2+ on both phones and tablets, iPhone 4+ and iPad 3 because these were the devices we had access to for testing.

The team also made sure that, when possible, the staff at Locatify had access to a configuration file within each game to change certain style features so that they could easily change a background of a scene or a color of a component within the game simply by changing a variable.

Documentation Requirements

Each game had to be documented carefully and each method or functionality of the code explained thoroughly. The team made sure that documentation was kept up to date as implementation of each game went on.

Additional Build Requirements

One additional requirement came from Locatify where they wanted an additional native build for both iPhone and Android phones. This meant implementing a native Xcode and Android project with native bindings where each platform would be running native code to communicate with our games in a web view.

[4] `http://goo.gl/HX8ufK`
[5] `http://phonegap.com/`

Game Ideas

After an initial brainstorming session the team and Locatify came up with 14 ideas for mini games which we whittled down to four important games. The requirement, however, was only to make three games at the very least. These four games were:

- **Tic-Tac-Toe**: A simple 3x3 Tic-Tac-Toe game where the player plays against an AI opponent.
- **The Water Bucket Balance**: A PhoneGap game where the user walks a certain amount of distance while trying not to spill too much water from their water bucket.
- **Crosswords**: A crossword puzzle game that could be given a list of words and would randomly generate a crossword puzzle that the user would try to finish in a certain amount of time.
- **Jigsaw Puzzle**: A jigsaw puzzle game that would allow the user to upload their own image which would then be broken up into puzzle pieces that the user would be tasked with putting back together.

We came up with ideas for other games which we decided not to implement. The names of these games are slide puzzle, marble maze, point-to-point race, multiplayer Tic-Tac-Toe, jumping contest, free flow clone, pictionary, screaming contest, picture matching and googly eyes.

Methodology

The team tried out a couple of different variations of Scrum[6] methods but we soon realised that since we were working with multiple smaller projects rather than one big project we would need something that would include smaller iterations yet be easily extendable. Throughout the project we followed a few scrum basics quite consistently. We setup iterations in a Scrum style, applying story points with Scrum tools like planning poker. But in the end we cherry picked ideas from Scrum but did not follow it to the extreme.

We used sprints throughout the project and each sprint was initially set to be 2 weeks. We ended up extending some sprints to 3 weeks where we were nearing completion of a game. We did that so we would not have sprints where we were working on 2 games at the same time so that we could stay focused on one game at a time.

Our Scrum master initially was Leifur, the owner of Locatify. He and Steinunn, the co-owner of Locatify, were the product owners throughout the whole project. Half way through our project Leifur stepped down as Scrum master because of misunderstandings with the structure of the final project. The team was quite autonomous and didn't feel the need for the role of a Scrum master but during critical times and retrospectives Eric was Scrum master after Leifur stepped down.

To manage our Scrum progress we used the online tool ScrumDo[7]. Initially we set out to use a simple spreadsheet but the simplicity of ScrumDo really attracted us and we are generally happy we made that decision. We used a Google spreadsheet to keep track of hours that we spent on the project.

Our work schedule throughout the semester had us meeting in person on Tuesdays and Wednesdays and working remotely from home on Thursdays. At the beginning of the project we followed this schedule rather strictly. As the project went on and we hit rougher patches in our schooling and life the team showed excellent resilience and flexibility in being able to work when was needed and creating a synergistic remote atmosphere.

[6] `http://en.wikipedia.org/wiki/Scrum_(software_development)`
[7] `http://www.scrum.do`

Code Style, Tools and Code Architecture

At the beginning of the project we agreed upon a small set of code style rules. Our code style was as follows:

- **Modules**: GetDude(), pascal case
- **Functions**: getDude(), camel case
- **Variables**: getDude, camel case
- **Indentation**: one indentation is 4 spaces

The software tools we used to help us in our project included:

- **ScrumDo**: An online Scrum managing system
- **Dropbox**: Used for hosting our games online as well as sharing game assets and reports
- **GitHub**: Our version control system
- **WebStorm**: A web development IDE by Jetbrains
- **Sublime Text 2**: A text editor
- **TeamCity**: Our continuous integration system
- **PageKite**: A simple tool for exposing locally hosted files to the internet which we used to access TeamCity
- **Skype**: Voice and instant messaging while working remotely
- **Google Drive**: For keeping track of hours worked, writing reports and preparing presentations
- **Ripple Phone Emulator** : An online PhoneGap emulator
- **Adobe PhoneGap Build**[8]: An online resource for building PhoneGap apps remotely

We used the following external frameworks and libraries in our projects:

- **PhoneGap**: Framework to wrap web pages into native mobile apps
- **Kinetic JS**[9]: HTML5 Canvas JavaScript library
- **jQuery**[10]: JavaScript library
- **jQuery.mb.audio**[11]: JavaScript library for making sound work on Android and iPhone
- **Zepto.js**[12]: JavaScript library for adding touch events

[8] `https://build.phonegap.com/`
[9] `http://kineticjs.com/`
[10] `http://jquery.com/`
[11] `https://github.com/pupunzi/jquery.mb.audio`
[12] `http://zeptojs.com/`

- **Move.js**[13]: JavaScript library for easy animated CSS movements

Because PhoneGap plays such a big role in all of our apps it's important to explain its functionality more thoroughly. PhoneGap is a free open source framework that allows you to create mobile apps using HTML5, JavaScript and CSS instead of device-specific languages such as Objective-C(iPhone) or Java(Android). The resulting applications are hybrid, meaning that they are neither truly native (because all layout rendering is done via web views instead of the platform's native UI framework) nor purely web-based (because they are not just web apps, but are packaged as apps for distribution and have access to native device APIs). It also allows you to deploy to 6 different mobile platforms. This flexibility does come with a few drawbacks however. PhoneGap apps do not perform as well as a native app would for a particular platform, they have long build times to see the app on your phone (depending on the platform) and PhoneGap does not have explicit control over a lot of small details that you would normally have control over when building a native app.

In our JavaScript code we used the module pattern so that we could imitate the functionality of classes. An example of a module pattern is as follows:

```
1  var WeirdCalculationModule = function() {
2       var a = 2; // a private variable
3
4       // Do some unsafe calculations
5       function doCalculations(x) {
6          return x * a;
7       }
8
9      // list functions/variables here to make them public
10     return {
11         doCalculations : doCalculations
12     };
13 }();
```

To use this module one could simply write:

```
1  var result = WeirdCalculationModule.doCalculations(1337);
```

After executing this the variable result would then be 2674. Note that although JavaScript is technically object oriented, through prototyping, we opted for the module pattern instead for the sake of simplicity.

[13] http://visionmedia.github.io/move.js/

Testing and Build Environments

In our project we relied mostly on manual tests as well as a testing application we built ourselves. At the beginning of the project we set up Mocha[14], a functional testing suite for JavaScript. As the project continued we found that we had to prioritize making the game over writing functional tests. Because the code for each game was much smaller than working on something like a large website, we decided this was not a terrible decision. Most of our testing also was based around limitations of mobile devices and had to be done simply by looking at the game on those particular devices. We were able to find an online PhoneGap emulator called Ripple[15] which sped up our testing time considerably as we did not have to always build to phones to test things like how the UI looked and simple accelerometer functionality and GPS functionality. We also ended up building our own testing application for The Water Bucket Balance that let us record movements that we made with our phone so that we could replay these movements in our development environments and see how the water in the bucket reacted to real phone accelerometer data.

For continuous integration we used TeamCity[16], a build management and continuous integration server by JetBrains[17]. TeamCity mainly handled automatically compiling all code we pushed to GitHub and automatically created runnable applications for Android and iPhone. Our setup was in essence continuous delivery. We weren't able to continuously deploy new versions to our mobile devices but they were just an install away since the build automation had already taken care of creating the artifacts for deployment. Although the project did not call for it we learned the value of such a setup. Other automatic tasks we set up included general clean up, running any functional tests we had made, and sending email to team members if any tests failed or the code could not be built every time new code was submitted.

We hosted the TeamCity server on one of the team member's private computer, a Mac Mini, and made it accessible via PageKite[18]. By utilizing PageKite, a product by the Icelandic startup company The Beanstalks Project, all team members could easily access the build server and make changes if necessary. We would like to note our liking to PageKite, it was a very handy and

[14] `http://visionmedia.github.io/mocha/`
[15] `http://ripple.incubator.apache.org/`
[16] `http://www.jetbrains.com/teamcity/`
[17] `http://www.jetbrains.com/`
[18] `http://pagekite.net/`

easy to use tool for exposing any locally hosted web page on our computers to the internet. It helped us greatly in being able to work remotely.

Final Product

Our final product ended up being three mini games: Tic-Tac-Toe, The Water Bucket Balance and Crosswords.

Tic-Tac-Toe

Our final version of Tic Tac Toe is a simple Tic Tac Toe game against an AI bot that can be set to easy, medium or hard through global parameters that would be changed in Locatify's CMS system. It includes sound effects and has been tested and works on iPhone 4+, phones and tablets with Android 2+, iPad 3+ and the web browsers Safari, Chrome and Firefox. It is a standalone web game that can easily be added inside any app by Locatify simply by utilizing a web view. When the user wins, loses or gets a draw it sends parameters through AJAX so that the app housing its web view can determine how the player did. Global parameters have been set so that the color of the grid, icons and the background can be set easily by Locatify. Any size of grid (3x3, 4x4, etc.) can also be fed through a global parameter and the graphics will draw a grid that size. The AI however only works on a 3x3 grid.

The Water Bucket Balance

Our final version of The Water Bucket Balance is a game where the user has to walk a certain distance without spilling too much water from a graphical water bucket. As the user loses water the background fills up with water. When the screen is full of water then the user loses. It includes sound effects and utilizes the accelerometer and GPS of mobile devices and has been tested and works on iPhone 4+. It partly works on phones with Android 4 but the team was not able to get it fully working on Android. It is unknown whether it works on phones and tablets with any operating system below Android 4 or iPads. There are global parameters available to Locatify to change the difficulty of the game to be either easy, medium or hard. The harder the difficulty is the less water the user can spill before losing the game. The graphics are not as easily customizable as Tic Tac Toe but all the graphics are done with CSS which can easily be changed by someone with a bit of CSS knowledge. The Water Bucket Balance can not be played as a stand-alone

web game and has to be either a PhoneGap app or wrapped inside native apps using a PhoneGap web view. The Water Bucket Balance is a playable game and is close to looking professional but still needs a bit of polish by Locatify before being able to release it. We were also not able to implement a timer into the game, which was a requirement that was given to us later in the development of The Water Bucket game.

Crosswords

Our final version of Crosswords is a game where the user can complete a simple timed, dynamically generated crossword puzzle. The user loses if the timer runs out before they are able to successfully finish the crossword puzzle. It has been tested and works on phones and tablets with Android 2.3.6. It is unknown whether it works on iPhones, iPads or other versions of Android.

Included in the Crosswords code is a method to dynamically generate a crossword from a given list of words. If a crossword cannot be created from the given set of words an error will be thrown. However this is only a consideration when the code will be implemented in Locatify's CMS. The Crossword game on an actual device would be passed an approved crossword by the CMS creator. The graphics are all done in CSS so it can be customized with someone knowledgeable in CSS. Although we were able to deliver a playable game, it lacks polish and would require some graphical work from Locatify before they would be able to totally release it.

Cancelled Plans

Although we had listed the Jigsaw Puzzle as another important game to finish we were unfortunately not able to get to it. Time constraints and technical setbacks made it so that we were not able to even begin the game. However, we did make user stories for it and were prepared to work on it if we had the time.

We were also not able to reach our additional build requirements mentioned earlier of wrapping all of our games inside a native Xcode and Android project as web views. We were only able to deliver a native build for Tic-Tac-Toe on iPhone. With our game that depended on PhoneGap accessing phone components (The Water Bucket Balance) we were not able to get it working as a web view inside an existing app. We also did not wrap Crosswords inside any native projects as a web view.

Code and Documentation Location

The structure of the code handed in with this report is as such. Our games are located at `/root/games` and then split up into `/waterbucket`, `/tic_tac_toe` and `/crossword`. After going inside one of our game folders the main code is in the folder `/www` except for in `/tic_tac_toe` where the code is in it's main folder. `/www` and `/tic_tac_toe` contains our JavaScript, CSS, images and everything relating to the web part of the game. As an example, if you wanted to see the JavaScript files for The Water Bucket Game the full path you would follow would be `/root/games/waterbucket/www/js`.

The documentation for all games is in is in `/docs` under the respective games folder. The documentation of our code is in the form of a website for each game. For instance, if you wanted to look at The Water Bucket Balance documentation you would launch it from `/root/games/waterbucket/docs/index.html`. From this website you will find links to both Tic-Tac-Toe and Crosswords documentation. The documentation for those two can also be accessed by launching the index file inside their `/docsfolder`.

Challenges

In this chapter we will list some of the most notable challenges we faced while working on this project. Additionally we tell of our workarounds where we found some. As mentioned earlier, one of the technical requirements for this project was to use the framework PhoneGap. As things progressed we ran into more and more issues and the project slowly turned into a research project to document pitfalls to avoid.

Phonegap

Our getting started experience with PhoneGap was a bit mixed. Although PhoneGap's JavaScript API documentation was generally very good, the documentation to get started was conflicting and sometimes outdated. This was evident when preparing to build code locally on our machines. We spent a lot of time on trial and error to get up and running, time we would much rather have spent on actually creating the games.

Another problem we had with PhoneGap was that during programming of the Crosswords game there was a need to bring up the keyboard to be able to input letters. The problem was that virtual keyboards on mobile devices vary widely and can be quite unpredictable in size. Granted, the PhoneGap documentation did have some configurations to control the virtual keyboards, but the methods were different for each platform. We tested the Crosswords game primarily for Android 2 and we were unable to get any of the configuration options to work.

We also ran into a problem when wrapping one of our games inside Xcode, the native source editor for iPhone apps, as a web view. With our first game, Tic-Tac-Toe, we got the project up and running without any platform specific problems and user interaction worked as expected. With The Water Bucket Balance we were using more hardware specific utilities such as the phone accelerometer. We gave up running it inside a web view since we could not get a response from these hardware utilities and decided to focus more on making the game perform well as a an app within the device. We were not able to find a solution for this problem.

CSS

With our initial game, Tic-Tac-Toe, performance was not really an issue since the game is comprised of very little animations and mostly static images. The

result was that the game was made entirely with the HTML5 drawing canvas. This changed dramatically when moving to our next game, The Water Bucket Balance, since it required faster animations. We had to throw the canvas out of the window and build the game using CSS only for all animations. But this brought up another problem. Moving an element by accessing its position can be heavy for the CPU, this is because a single element moved from point a to point b needs to be redrawn at every single position any time that it moves. This is on top of all other rendering happening with other elements at that time.

We were able to fix this when we discovered transform, a CSS property that will elevate the element to it's own layer on the GPU called Render Layer. Being on its own layer, calculations concerning translation, rotation and scaling become much faster.

Another problem was sizing CSS UIs to all devices. It is usually only a minor inconvenience when developing a website, but we found that keeping this flexibility when making complicated graphics that often use hacks was much larger than a minor inconvenience. Although our UI was developed all with dynamic sizing and percentages instead of pixels, we found that our UI would still not size correctly, especially when displaying on tablets.

The solution to this was using aspect ratios. For instance, in our case the height of the phone was always throwing off the water bucket graphics. What we ended up doing was always making sure the area where the water bucket graphics were was a square. So no matter what, the height of the graphics container would always be the same as the width. This made it work well enough in all screen sizes. We also learned never to use negative percentages in our CSS because it can make unexpected things happen.

HTML5 / Javascript

HTML5 Audio Problems On Mobile Devices

Mobile platform support for the HTML5 audio element is all over the place. Figuring out how to get sound effects correctly working across all platforms took considerably more work than we expected. Below we will list some of the main problems that we encountered with HTML5 sound on mobile devices.

Using the HTML5 <audio> tag (or calling audio from JavaScript) significantly

affects performance when played in both iOS and Android. Audio is not played asynchronously by the browser on iOS or Android which greatly impacts performance when a sound is run.

Also, playing multiple sounds at the same time is almost impossible using the <audio> tag on Android. On Android, when a sound is played right after another, the first sound stops and can never be played again unless it is reloaded completely. Additionally, there is absolutely no way to preload a sound effect on page load. Neither Android nor iOS allow you to load a sound effect when the page loads.

We ended up finding a JavaScript library that takes care of all these problems on Android 3.9 and below and all versions of iOS called jQuery.mb.audio. This library uses sound sprites and queues. A sound sprite is a single audio file with multiple sound effects. When you want to play a sound you simply give it the start time of the sound effect and the end time that it should pause the sound file. jQuery.mb.audio makes this much easier. It also allows you to put sounds on a queue (for Android specifically) that make it so that if another sound is triggered before the end of an active sound, it will not stop the playback of the active sound but will instead play it after the current sound effect finishes.

However, in both Android and iOS there must be a touch event that triggers the loading of the sound sprite, as it cannot be loaded on page load. On iOS specifically you must do

```
1 player.play();
2 player.pause();
```

after you have loaded the sound sprite or else it will not be able to find the sound effects within the sprite. For Android, the sound sprite has to be a high bit rate or else it will play the wrong sections of the sound sprite when called. Loading two sound sprites does not affect performance but more than two should not be loaded. Also, iOS is the only platform that can play two sounds at once and the Android version has to have any background music disabled (or any other sounds that will be playing at the same time as the main sound effects).

Jquery.mb.audio does not work on Android 4. Android 4 can play <audio> tags fine and it must be done the traditional way by adding <audio> tags

and playing them through JavaScript. This method is still unreliable and sounds have a tendency to be inconsistent at times. A website called "*Making HTML5 Audio Actually Work on Mobile*"[19] proved to be an invaluable resource for fixing these issues.

JavaScript Hardware Acceleration

We learned that some phones do not have the ability to optimize JavaScript code for performance. This is called hardware acceleration. Hardware acceleration is when the hardware of a given device is used to run code faster than it could be implemented in the software code itself. A good example of this is the transform property within CSS (mentioned above) where the code is optimised to render matrix transformation on a separate GPU layer. For example, on Android phones the HTML5 canvas element is not hardware accelerated meaning that there will be no speedups whether you draw a single line or a thousand lines moving across the screen.

Our solution to this problem was to switch entirely to using elements that would for sure be hardware accelerated by all platforms by using CSS transform. This works for both Android and iPhone devices.

[19] `http://pupunzi.open-lab.com/2013/03/13/making-html5-audio-actually-work-on-mobile/`

Progress Reports

Here we list our Scrum sprints. We briefly describe what work they entailed, when they took place, and include our Scrum sprint retrospectives where applicable.

Sprint 0: Bringing Order To Chaos

In this "sprint" the team got together for the first time. Here we brainstormed on game ideas and revised the project proposal. We did not do a retrospective in this sprint as things were very chaotic and work procedures had not been established yet. Duration: 2013.09.02 - 2013.09.16.

Sprint 1: Tic-Tac-Toe

In this sprint we started our work on our first game, Tic-Tac-Toe. We had many games to pick from but we decided to start on a relatively trivial game to ease our way into programming with PhoneGap and generally program for mobile devices. Duration: 2013.09.17 - 2013.10.03.

Retrospective

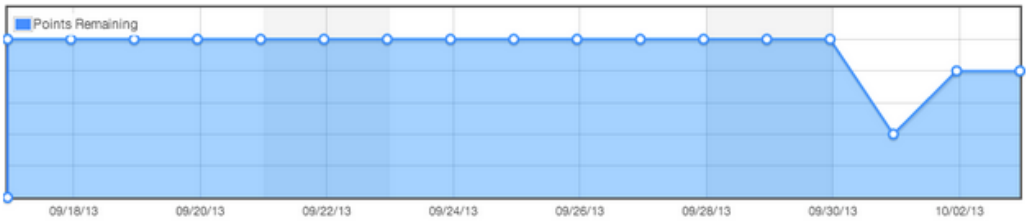
Things that went well:

- Made the game work on multiple platforms
- Managed to wrap the app inside PhoneGap to acquire builds for multiple devices
- Managed to wrap the app inside a native Xcode project for iPhone build

Things that could improve:

- Scrum planning, story points and scheduling were off

Burndown Chart



Sprint 2: Waterbucket 1st Iteration

In this sprint we started work on our second game, The Water Bucket Balance. Duration: 2013.10.08 - 2013.10.21.

Retrospective

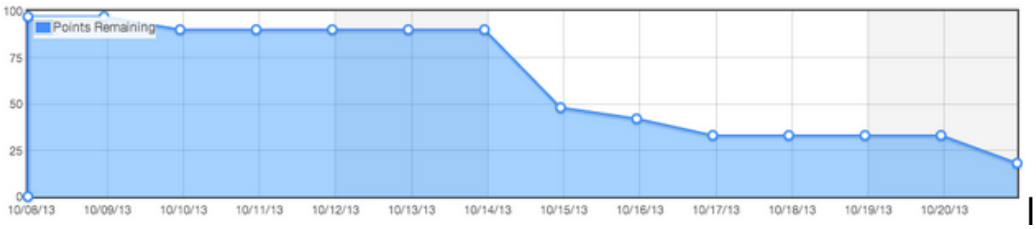
Things that went well:

- Worked better as a team, improved scrum planning
- Movement and animation of components were quickly set up

Things that could improve:

- Failed to anticipate that performance on all phones was not equal. The HTML5 canvas element is not compatible with all devices
- Lost some of our supervisors attention
- Some communication problems within the team

Burndown Chart



Sprint 3: Waterbucket 2nd Iteration

In this sprint we continued our work with The Water Bucket Balance. Duration: 2013.10.22 - 2013.11.11.

Retrospective

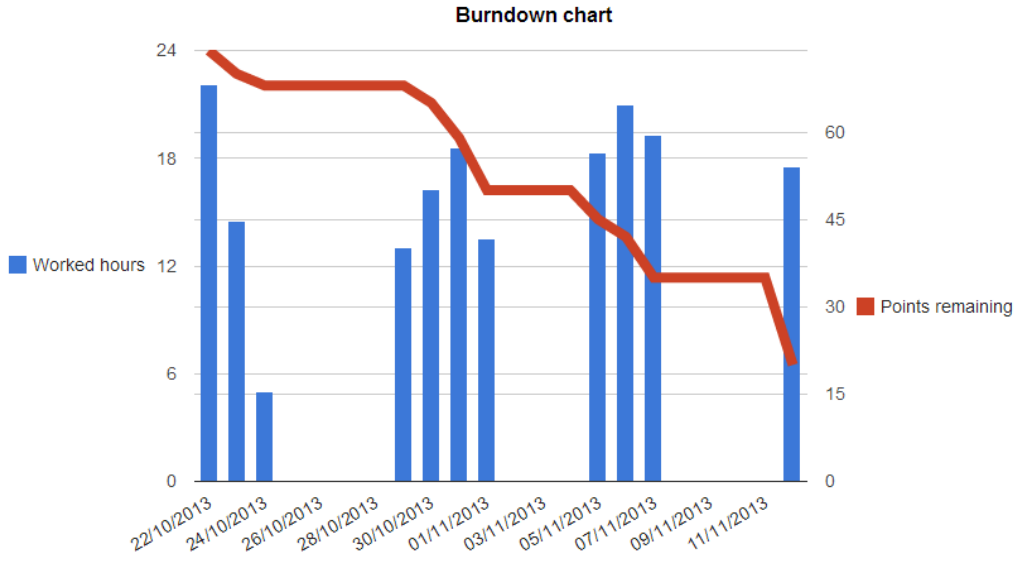
Things that went well:

- Got geolocation and distance tracking to work.
- Created a test simulator to run our game in for analysing purposes
- Waterbucket UI components mostly completed

Things that could improve:

- CSS optimisation
- Tasks took about 3 times longer than we initially expected
- Some communication problems within the team
- Supervisor communication continued to diminish

Burndown Chart



Sprint 4: Waterbucket 3rd, Crosswords, Puzzle Piece

In this sprint we planned to finish up the Waterbucket game, start and complete the Crosswords game and start development on the Puzzle Piece game. We managed to complete the Waterbucket game and nearly completed the Crossword game. In retrospect we were way too optimistic about starting work on the Puzzle Piece game. After consulting with our instructor/teacher it was decided we should rather focus on having 3 polished games instead of 4 unpolished. That's the reason for the high amount of remaining points in the burndown chart shown below. Duration: 2013.11.12 - 203.11.30.

Retrospective

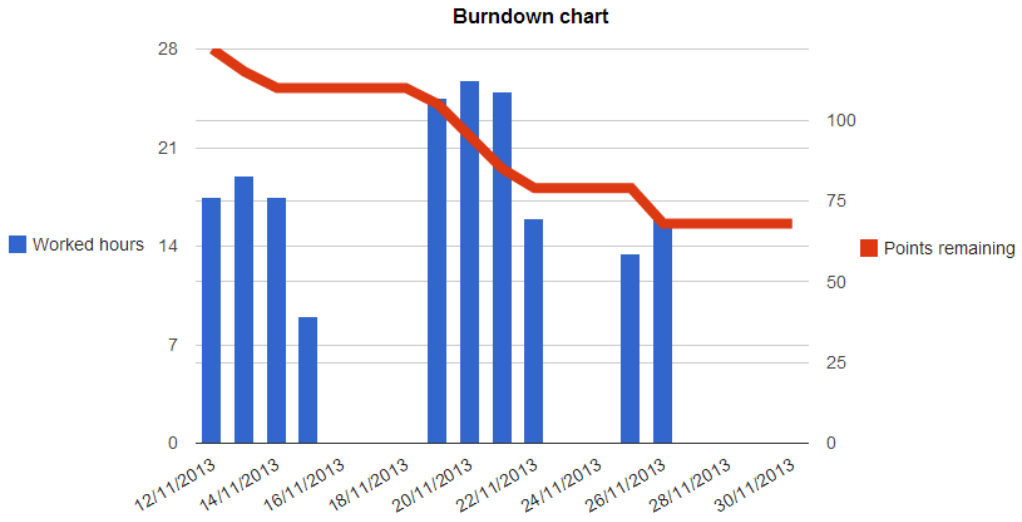
Things that went well:

- Crossword nearly completed
- Water bucket game playable
- Collision detection for water bucket game improved

Things that could improve:

- Not be too optimistic about our performance

Burndown Chart



Final sprint

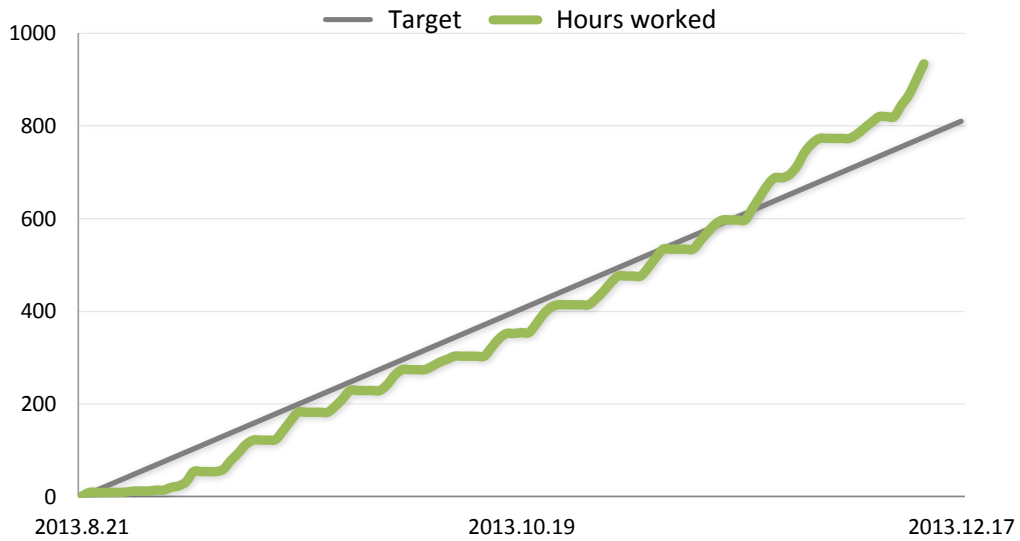
Technically not a sprint but more of a time we spent tying up loose ends. For example here we created this final report, put final polish on the Waterbucket and Crossword games and created and prepared for the final presentation. Duration: 2013.12.01 - 2013.12.17.

Hours Worked

The final project course accounts for 12 ECTS units. According to the European Commission[20] each ECTS unit should account for 25-30 hours of work. For each member of the team (students) that works out to be 300 - 360 hours.

We estimated and targeted to spend 810 - 1080 man-hours for this project to comply with standard guidelines set forth by the European Commission. When this document was written we had worked for a total of 934 hours. We therefore achieved our goal on hours worked on the project.

The image below shows the progress of hours worked along with our target.



[20] http://ec.europa.eu/education/lifelong-learning-policy/ects_en.htm

Conclusion

In conclusion, here are the things that we as a group have decided that we learned most from in this project.

As a team we all feel much more confident in JavaScript and CSS than we ever did before. Navigating these web languages feels much more natural and we were able to make readable, well structured and optimized code. We also learned much about documenting our code. Our documentation skills have improved considerably and we now know how to make much more professional, readable documentation for real world JavaScript programming using jsDocs. Additionally, we were able to successfully setup and consistently use a continuous integration environment thanks to TeamCity. This helped prepare us for how things are done in real world software development and also saved a lot of time overall.

We did not use functional testing as much as we would have liked, although we had a major success in finding a way to test our PhoneGap apps before actually deploying them to a phone using the Ripple Emulator. This saved hours of our time. It was also a huge success that we were able to build our own simulator for simulating real accelerometer movements recorded from actual mobile devices.

We learned many things about web development and were able to learn many useful tricks in the web development world for optimizing CSS and JavaScript for Android and iOS with all their quirks included. We were able to successfully get one game (Tic-Tac-Toe) working in all versions of Android and iOS 5+. Even though The Water Bucket Balance only works completely on iOS and Crosswords only works completely on Android they are still both very close to working flawlessly on both platforms. The skills that we learned for mobile web development are invaluable to us in our future.

Working as a team throughout this project went very well although we hit a few speed bumps along the way. It wasn't a totally smooth experience but when we pulled together we communicated very well. Even though there were rough patches throughout the project it was always clear what everyone should be doing. We learned that one of the most important concepts for keeping the group on the same page was reporting, through e-mail or in person, what each individual had finished that day.

Being more strict with Scrum is something we feel we could have done better,

and we feel that it might have helped us in the long run. In the end the team felt so overwhelmed with the amount of user stories we had that we neglected Scrum so that we could get more work done. In hindsight we probably would have gotten more work done if we had given just a little more time to organizing that work.

The most negative area in the project was having to satisfy two supervisors, being the teachers and Locatify's owner, with differing ideas. This made tension at times between the team and Leifur, Locatify's owner. Leifur became apathetic towards the project halfway through as he felt a loss of control over the project when we were told by the teachers to change the order of the mini games to work on, resulting in the loss of a game he particularly wanted to be finished. It was after this point that the team almost became a separate unit from Locatify with occasional suggestions from Leifur.

PhoneGap Recommendations

It took us some time to configure and work with PhoneGap and to know how it worked, such as its shortcomings as well as mobile platform differences for HTML5 and JavaScript. We feel that unless you are making static UI's or very simple games with little moving animations then time would be better spent creating your program in the native environments for the different phones. If your program has complicated animations and lots of movement it depends heavily on your CSS skills how long it will take you to construct the graphics since drawing animation with CSS is a bit more advanced than using the HTML5 canvas element. This makes it much harder to quickly make graphics that sizes to all devices (for a game at least). Moreover, building a PhoneGap project to your phone for testing eats away hours of time. If we had not found the Ripple Emulator we would have not been able to accomplish half of the things we accomplished.

Final Thoughts

The goal of this project was to create at least 3 stand alone mini games that could be put inside Locatify's TurfHunt app (as well as others). As a team we succeeded in accomplishing this goal. We delivered three mini games, Tic-Tac-Toe, The Water Bucket Game, and Crosswords, that can be integrated into Locatify's products. We achieved our goal to work at least 810 man hours. The final number turned out to be 934 hours worked for all three students. Along the way we learned what it meant to work in a real work environment. We were pushed to our limits and we grew as programmers. We were taught the reality of planning too optimistically as we realized how few of the mini games on our list we would actually be able to finish. We also experienced how human relations can affect and possibly interfere with the outcome of a project, from disconnecting a bit with our supervisor to running into communication problems within the team. In the end we learned the bittersweet dichotomy between the joy of bringing software to fruition mixed with our own dissatisfaction that our product could be better when the deadline hits. A feeling we are sure will be repeated over and over again in our careers.