Dakar Falls

Major Project – Final Report

Interactive Multimedia Design

2014

Dean Coulter – B00560473

Group A - Jonathan Wallace

Acknowledgements

I would like to give my acknowledge and extend my thanks to those who have helped this project to it's successful completion.

George Moore (Lecturer and Module Co-ordinator) for managing the module and guidance over the past year.

Jonathan Wallace (Module Mentor), for his constant personal guidance and advice over the course of the module.

Oliver Biot, Jason Oster, Aaron McLeod (MelonJS Development Team) for providing a great open-source framework and for their constant support in helping to understand, fix and utilise it.

Paul McCormack, Tim Potter, Gabriel Muldoon, Chris Murphy (Lecturers) for their insightful design workshops throughout the year.

Contents

1. Introduction

For the final year of IMD we have been tasked with coming up with a concept and taking it forward to create a final product. The aim of this project was to create a game. It would be developed on html5 canvas and ported to a desktop app through node-webkit in an attempt to distribute it as broadly as possible on multiple platforms.

The game itself is a 2D platformer, inspired classic 90's games such as Super Mario or Megaman. To actually create the game we used javascript, along with a framework and game engine called MelonJS, and rendered it out onto a canvas element. Originally the plan was to also release the game as a native app for android and IOS, and target mobile gamers. However, later during the project performance issues and a flaw with melonJS meant the game couldn't be ported to mobile platforms. At this point the focus of the project switched to desktop users, and providing a desktop app through node-webkit, an app run time for turning web apps into windows and osx apps.

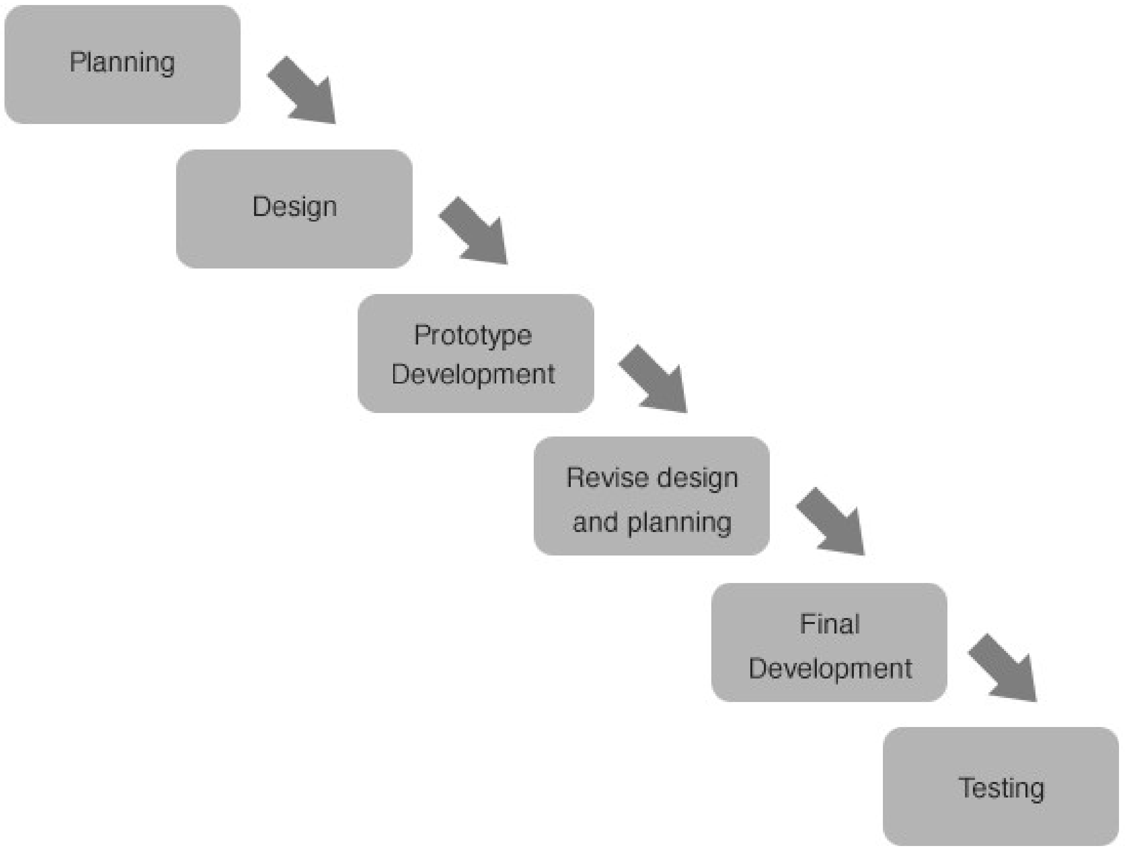
The bulk of the work that went into the project was for creating original artwork and developing code. Other aspects to game such as the music and sound effects were taken from free to use sites, or created using an app called bfxr. The methodology used for the project started out as a waterfall one, but as the project progressed and requirements changed, it switched to an agile methodology to accommodate the vast of amount changes being made to the project mid-way through.

2. Concept Definition and Testing

2.1 Methodology Selection

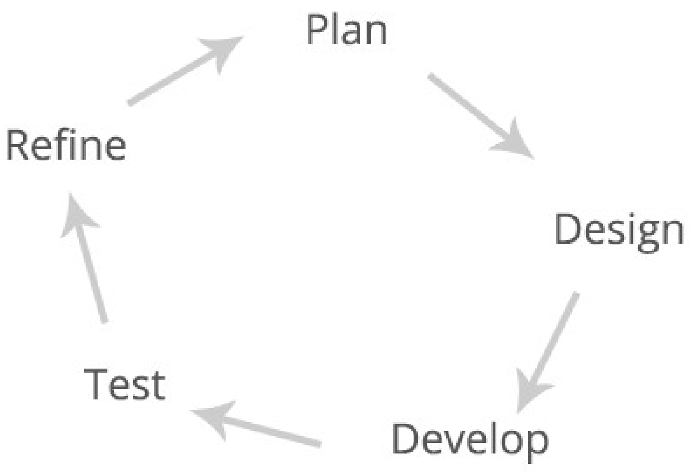
For this project it was decided that the most efficient way of completing the work would be to break everything up into clearly defined stages. So for instance the artwork and development stages would be separate and wouldn't overlap, and work on development could only start once all the assets had been finalised. This way there would be less risk of major changes being made during the development stage, which would result in chunks of code potentially being scraped because a feature has been removed. Overall this means less time wasted on creating features that aren't needed in the end, which should help move the project along more efficiently.

Therefore waterfall was chosen as the methodology for this project, because it most closely met the needs that were just specified. However, usually waterfall means that once a stage is completed it shouldn't be revisited, but for this project, features and artwork might need to be revisited after receiving early testing feedback. So for this reason the approach has been modified slightly and an extra stage was been added for going back and potentially revising parts of the design and planning.



After the christmas period and after releasing an alpha prototype to the public for feedback, work began on developing the finished product. A few weeks into that process major changes were being made to the game, both technical and design, based on feedback and circumstances, such as problems deploying the game to mobile platforms. So for this reason a different methodology was needed to cope with constant influx of changes.

The decision was then taken to switch to an agile methodology. Now for each feature that was to be included into the game, it would planned out, designed, developed in code, tested internally and then any changes that needed to be made based on testing and feedback would be done. Overall this was positive change because at this stage the game was being modified a lot, with mentor sessions and the alpha test feedback. Agile's iterative approach really helped to accommodate these changes a lot more than waterfall had.



2.2 Idea Generation

To begin with, 3 ideas for the major project were selected. For each of these ideas a concept definition statement was created that gave a summary of the idea, what the final product would be, and the resources that would be needed. This helped when analysing each idea to compare their pros and cons.

The first of the three ideas was to create an online multiplayer game using both canvas and html5 websockets. The second was to create a lightweight cms using node.js, that would also have some unique features, such as being being able to push live updates.

Then thirdly was an idea of having a service that housemates could sign up to, and have their own feed to chat, keep track of bills and events.

Eventually it was decided to go with the first idea based on feedback from the mentoring sessions saying it was the most interesting of the three, and with the right approach, it would be something unique. However, after some further research it was decided that the online aspect should be dropped, due to how complicated and time consuming it would be to complete. Also, accurately and throughly testing an online game would require many participants to ensure the stress testing was sufficient. Gathering enough participants for this would be both time-consuming and difficult. However the option to add some smaller online aspects, such as leaderboards or social interactions was left open.

Later, the idea of porting the game to android and iOS mobile devices was explored after discovering cocoonJS, a mobile deployment platform that allows developers to take web apps and deploy them as native mobile apps. Normally playing a canvas based game on a mobile device results in very poor performance because canvas isn't hardware accelerated like it is on desktop browsers, and thus it can't take advantage of the GPU to render graphics at a faster rate. However, using cocoonJS these games can become hardware accelerated by packaging them natively. The developers of this deployment platform Ludei claim that their apps can reach near native performance, however whilst testing some of their demo's on an android phone, the frame rate sat at about 45fps, which is about 15fps lower than a standard game. Although slightly noticeable, it's still an acceptable level of performance.

As a contingency plan, in the event of something going wrong with the mobile releases, such as the performance of cocoonJS not being up to scratch, or being rejected by either the itunes app store or google play store, the game can still be released as a web only experience. On top of that, the option to explore exporting the game as a desktop app was considered, which would involve less risk in regards to performance, since desktops and laptops are generally much more powerful than phones and tablets.

The actual genre of the game was to be a 2D platformer inspired by classic games for consoles like the snes or sega gensis, as well other some modern games that also take their inspiration from late 90's games. During a lectures it was advised that the project should be something that would interesting to work on, seeing as after several months, it can become disinteresting looking at the same project for so long. Which is why it was decided to create a game that would be personally interesting, rather than trying to judge what is popular now, and what would other people want to play.

However, this creates a risk where users might not be interested at all in the game being developed, which will have a negative impact on the success of the project. To counter this, to decision was taken to create an alpha version and release it to the public to play, that would feature the core concepts of the game, as well as the first few levels. During this test feedback will be collected to gauge user interest in the game, and from there any necessary changes to make the game more appealing the general mass can be made.

2.3 Requirements Specification

A requirement specification is a description of what an application will do and how a user should expect it to behave. It should explicitly list all the functionality and capabilities an application should have, and it essentially a written assurance that we understand the users needs.

The Volere Template was used in order to help create a requirements specification. The Volere template is tried and trusted template, and should therefore undoubtedly be suited for this project. Using a template is much preferred to simply making one up, because it has already proven to work with multiple companies in the real world.

Requirement: Auto Saving	**Requirement Type**: Functional
Priority High	
Description: After each level is completed, the game will have to save progress the user has made.	
Rationale: If the game didn't save the progress each user makes throughout the game, they would have to restart the game each time they loaded it up, and have to replay levels they've already completed. This repetitiveness would mean they would quickly lose interest.	
Fit Criteria: If the game successfully's saves the progress of every user during testing on multiple systems, this will be considered successful.	

Requirement: Menu Screen	**Requirement Type**: Functional
Priority High	
Description: After each level is completed, the game will have to save progress the user has made.	
Rationale: This will provide users with option to choose different game modes, or access options for the game.	
Fit Criteria: User feedback during testing should bring to light any issues with this screen.	

Requirement: Level Select Screen	**Requirement Type**: Functional
Priority High	
Description: So that users can easily replay each level, there will be a level select screen available.	
Rationale: Users will need to replay levels that they've already completed to find things they may have missed, so for that reason a screen that allows them to replay any levels they're already completed will be provided.	
Fit Criteria: User feedback during testing should bring to light any issues with this screen.	

Requirement: Hosting & URL	**Requirement Type**: Functional
Priority High	
Description: The game and website and will need to be hosted online using a reliable hosting provider and domain name.	
Rationale: For users to gain access to the site and game, I will need to purchase hosting, and a relevant URL.	
Fit Criteria: If the server can withstand the load of however many users are using it, then it will be a success. Otherwise I will have to look at getting a server with more bandwidth.	

Requirement: Tracking and Analytics	**Requirement Type**: Functional
Priority Medium	
Description: A system that tracks how many users visit the site, what browsers and software they are using is needed. Mostly likely Google Analytics.	
Rationale: It's important to check what software people are using to view a site because of how differently sites might appear and perform on different browsers and operating systems. Some browsers like IE8 can be difficult to develop for, but if a significant number of user aren't using to view your site, then there's no need to waste time on it.	
Fit Criteria: If the analytics software is able to provide accurate, quick and detailed data, it will be perfectly suitable for the project.	

Requirement: Gamepad Support	**Requirement Type**: Functional
Priority Low	
Description: Ability for users to plug a usb gamepad into their machine and control the game through	
Rationale: Gamepads are the traditional method in which games are controlled through, so adding support for them will provide a more familiar and comfortable way of playing the game for a lot of users.	
Fit Criteria: Testing multiple gamepads on multiple systems and browsers will be needed to ensure this functions correctly.	

Requirement: Fullscreen Support	**Requirement Type**: Functional
Priority Low	
Description: Option for users to expand the game to take up the fullscreen	
Rationale: Many users enjoy the option of being able to make the media they are viewing take up all the possible screen space. Firstly to remove any unnecessary distractions and secondly because subjectively they enjoy viewing things on larger scale.	
Fit Criteria: This will need to be tested across multiple browsers and devices.	

2.4 Paper Prototyping

During one of our design workshops we were introduced to a prototyping method know as 6up's. This is where you would take a screen from your application and sketch out 6 different possible layouts for it. For this project the only real screen users will see will be the menu screen, so thats what was focused on. Things were kept pretty simple so that users could quickly choose one of a few options and get straight into the actual game as soon as possible. Therefore the only buttons that were included here were for play, about and settings.

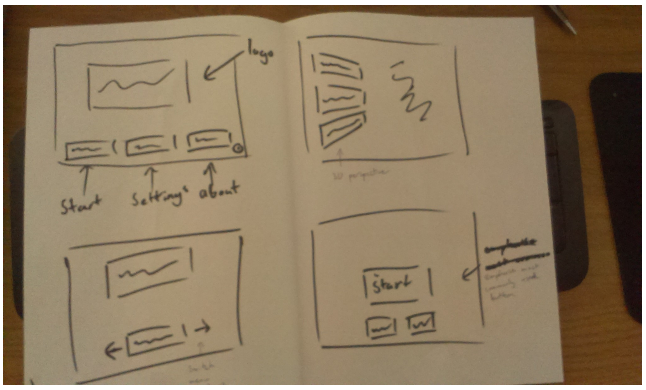


Figure 2.1 - 6up sketch 1

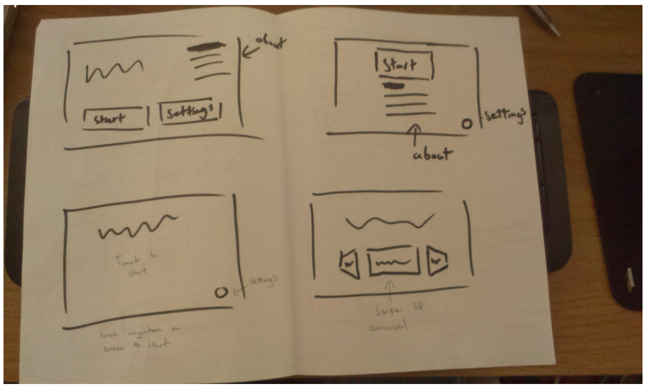


Figure 2.1 - 6up sketch 2

Realistically this approach suited applications with much more information and options that our app, that all needed to be organised and squeezed onto a screen. Also the method only encouraged people to sketch out ideas multiple times. There was no incentive to look at the design from different angles, such as how an advanced or novice user would differently approach an application, and didn't really help in providing new insight to prototyping.

However it encourage experimentation which wasn't a bad thing. Sometimes when mocking up ideas, people can come up with one or two very good initial ideas and stick to those. This approach though encourages people to look beyond those good ideas and even if a better idea doesn't come up, it might next time which is positive thing.

Some sketches for the main site that will hold the game were also done up. The best one of those (figure 2.3) simply includes a logo and play box at the top, the play box either being the game or a link to it. It's important that the game itself is the first thing people see, as it's the thing they will coming looking for first. Then hopefully after they'll read a bit more about if they're interested enough.

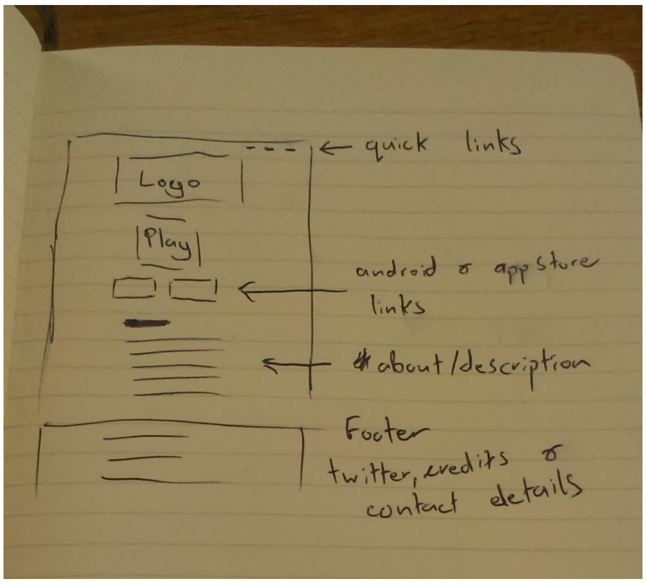


Illustration 1: Figure 2.3 - Website prototype sketch

For the main game artwork I aiming to create a really old-style pixel look. For this reason I decided that paper prototyping wasn't all that suitable for creating pixel artwork, so instead I focused on creating the game artwork straight on the screen.

3. Design

3.1 Visual Inspiration

Very early on, it was decided the art style for the game would take on a retro 90's pixel-art look, inspired by classic SNES and Sega Megadrive games such as Super Mario, Megaman and Sonic The Hedgehog. This is an art style that still proves popular today, especially amongst hit indie titles like Cave Story(figure 3.1), Fez or Superbrothers. Part of this is likely down to nostalgia, and the warm and positive memories a classic art style can invoke.

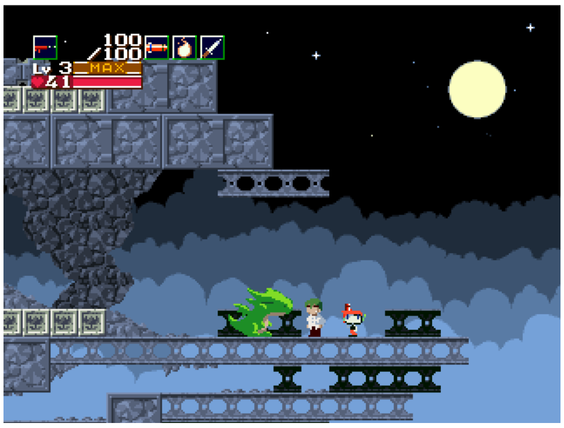


Illustration 2: Figure 3.1 - Cave Story Screenshot

Pixel art has proven very popular amongst independent developers, probably also because it's an easier style to work with. Originally games were limited in terms of art-style by the limitations of the hardware. Low memory space meant that small colour palettes and images had to be used. But as technology has developed, games have largely moved towards 3D graphics. For independent developers, 3D graphics provide a challenge because they are complex, and require multiple artists with the knowledge to create 3D imagery. Independent games are usually developed by between 1 to 3 people, meaning it's very unlikely they would have the human resources for 3D modelling and animation. For these reasons, this game has also gone down the path of 2D pixel art.

One of the challenges of pixel art however, was discovered over the course of the project. Whilst designing tiles or characters, everything has to be done in a 16x16 pixel space. This lack of space makes it very difficult to add detail to what your working on, meaning it's very easy to create flat artwork that looks unrealistic or out of place.

Designer and Developer Shaun Inman also provided a good source of inspiration for this project. Inman originally started off as a web-developer, and later moved into game design and development, creating a popular iPhone game called 'The Last Rocket'. One of reasons he was such an inspiration is because he show's that it is possible for a web-developer to make the move into game development, and make it successful.

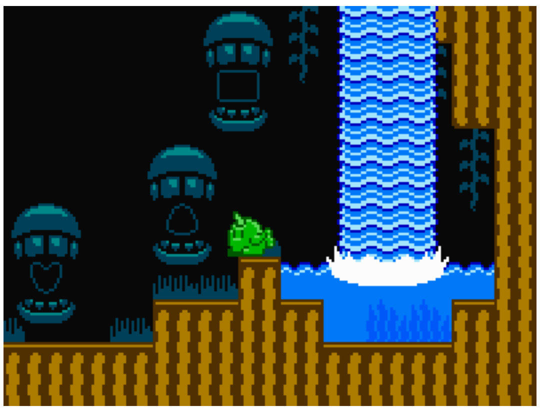


Figure 3.2 - Super Clew Land

The visual stylings of his own games were also a huge inspiration to project, because they also follow a retro pixel art look. Super Clew Land is one Inman's which was created in just 72 hours as part of a development challenge. One of the main things that stands out about the design is main character. Despite having very little space to work with on a 16x16 character, Inman has managed to create something that looks unique and memorable. Something which is important when trying to catch the eye of a user amongst many of other rival titles.

One negative aspect to the game though, was how when a player dies, the point at which they re-spawned was often quite far away from where they died. Which means if a player was dying often, they would find themselves replaying the same area over and over again for long periods of time, which can become repetitive and tedious, and put players off playing the game in any further. To avoid this issue on this project, the plan is to have many short levels, and some longer levels that will have regular checkpoints.

3.2 Character Design

For the main character in this game, the aim was to keep things simple, but to also create a recognisable character. Animating characters can be a very difficult and time-consuming task, which is why we planned to keep it simple and not waste too much time editing multiple frames of animation. It's also important to have a character who is memorable and would work well as mascot for the game.

One character who provided much initial inspiration was Gomez (figure 3.3) from the game Fez. During Fez, it's never explained what Gomez is, e.g. Human, Alien, animal, or where he came from. Instead he's just this abstract character that you control throughout the game. This is interesting idea, because games tend to exist in very bizarre and unrealistic worlds, so trying to add rational to them can become very complicated, because if you rationalise one thing, you have rationalise everything, and you can up create a very deep and detailed universe. With this project, the aim is to keep things simple, because we don't have the time and people to develop an in-depth story, so this approach would suit this project. The design of character is also simple, but without looking rushed. Which is exactly what we're looking for.

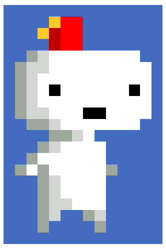


Figure 3.3 - Gomez

For the main character we ended up going an anthropomorphic crocodile, based on other games that also use humanistic animals for their mascots, such as Donkey Kong, Crash Bandicoot or Sonic The Hedgehog.

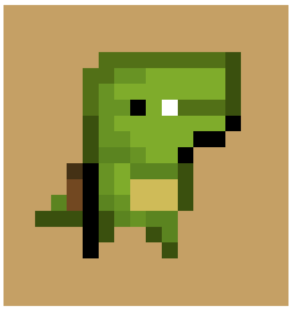


Figure 3.4 - Initial Character Design

Using this initial design (figure 3.4), work started on animating the character. Several different animation cycles were played around with (figure 3.5), with variations on walk cycles and head movement. In the end the last cycle was chosen, but further work still needed to be done adding additional animation states for jumping and being hurt.

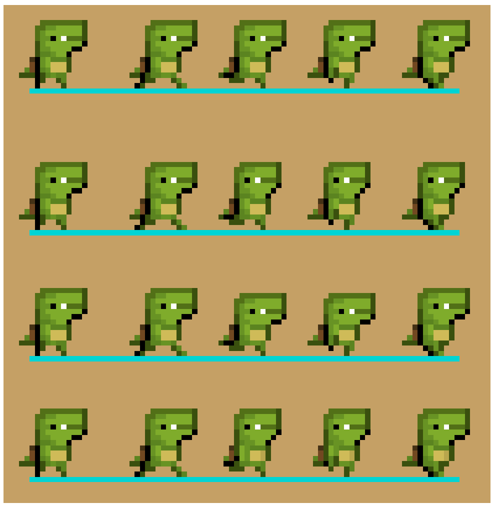


Figure 3.5 - Animation Tests

Overall there was a few issues with initial character design. Some comments from mentoring sessions revealed that people weren't sure exactly what the character was meant to be, with some thinking it was a dinosaur or a lizard. The anatomy also wasn't well enough defined and was difficult to work with, meaning work on further animations was halted.

In the end, the character was changed to a human protagonist (figure 3.6). Overall this made it much easier to complete the animation sequences because human anatomy is much more familiar and easy to work with seeing as there are more real life examples to work of. The character was also now pretty easy to distinguish for users.

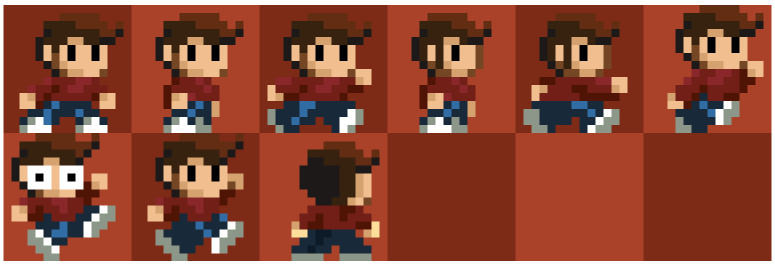


Figure 3.6 - Final Animation Sheet

For the persona of the character, the proposition of having a silent protagonist was considered. Originally silent protagonists were very popular appearing in big name games like Sonic The Hedgehog, Mario, Zelda etc. However this was more due to limitations of the hardware during the time, and need to use a little memory as possible. However, as gaming technology has advanced, silent protagonists have still remained relevant in some of the most popular modern games like Half Life, Bioshock and Portal. PC Gamer provides an interesting debate on whether these mute characters are superior or not[3]. The main argument in favour is that silent protagonists allow players to put themselves into the game and become more involved by expressing their own feelings, rather than listening to the thoughts and feelings of someone else, thus creating a more immersive experience. On the other hand, dialogue allows writers to create an interesting and memorable character that players can enjoy and appreciate.

In the end the decision was to go with a silent protagonist. Mainly to keep with creating a sense of nostalgia and creating a game that's reminiscent of classic console games. It also occurred that it would be difficult with the nature of the game to keep consistent and engaging dialogue going throughout a game that's focused on gameplay rather than story telling.

3.3 Name and Story evolution

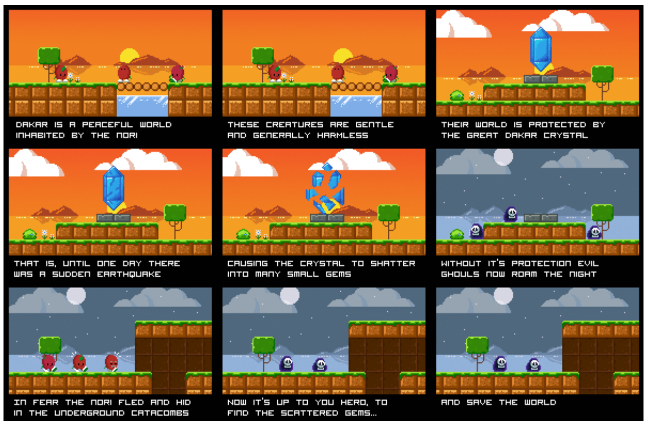
At the start of the project there was no definite decision on how the story line would be approached. However it was generally agreed that the story would not take center stage, but that it would only provide flavour for the game, and that the gameplay would be the focal point of the project.

Eventually ideas were thrown around and it was decided that the game would be based in a distant world named Dakar. This also where name of the game came from, Dakar Falls. Usually this world would be protected by a gem, but after this shatters the world is thrown into disarray and evil spirits begin taking over. Therefore our antagonist must collect the broken pieces of the gem to restore the world to order. The idea of collecting these pieces to progress through the game came first, so really the story here adds purpose and reason to what the player is doing.

To help actually defining the exact details of the plot some research into story telling was done and an article that listed Pixar's 22 rules of story telling was found [5]. Although this plot won't be nearly as long as something you'd find in a feature film, some of the rules still provided a good direction in which to take the story.

The 4th rule from the article provided a good way to structure a story and get down the main points of the story. So for my own game it would go like: Once upon a time there was a world known as Dakar. Everyday, the Nori lived there peacefully, protected by a powerful gem. One day, an earthquake caused the gem to shatter. Because of that, a great evil was unleashed onto the world and the inhabitants of the world were chased into the catacombs. Until Finally, our great hero found all of the shattered pieces and restored peace to the world.

To actually add the story to game it was decided to show a series of storyboards (figure 3.26) at the start of the game, accompanied by text to tell the story. The other option was to create an animated intro, but this would have proved incredibly time-consuming during time were the project was already over-run with tasks.


DAKAR IS A PEACEFUL WORLD INHABITED BY THE NORI
THESE CREATURES ARE GENTLE AND GENERALLY HARMLESS
THEIR WORLD IS PROTECTED BY THE GREAT DAKAR CRYSTAL
THAT IS, UNTIL ONE DAY THERE WAS A SUDDEN EARTHQUAKE
CAUSING THE CRYSTAL TO SHATTER INTO MANY SMALL GEMS
WITHOUT IT'S PROTECTION EVIL GHOULS NOW ROAM THE NIGHT
IN FEAR THE NORI FLED AND HID IN THE UNDERGROUND CATACOMBS
NOW IT'S UP TO YOU HERO, TO FIND THE SCATTERED GEMS...
AND SAVE THE WORLD
Figure 3.7 - Ingame Storyboard sprite sheet

Overall it was good to get the story element finished, but it was evident that it wasn't give enough thought early on in the project, which created a situation where the story was restricted by the work that had already been in the game. In the future how the story and gameplay fit together should be given much more thought at the start of the project.

3.4 Level Design

During the research process for this game, a trend was noticed amongst a number of the games that were being explored and played. A lot of them like Sonic The Hedgehog (figure 3.7), Super Mario World and Fez all started off in very vibrant and grassy areas. It's understood that the reason for this was because designers wanted to ease players into the game at the start.

Generally in all games the first couple of levels are the easiest. This is in order to help the user get used the mechanics behind the game, and then as they progress and get better at the game, the levels should also ramp up in difficulty. If the initial levels of this project were very difficult, then it's likely players would get frustrated at not being able to progress through the game, and stop playing. It can then be assumed that the reason designers give the first level a vibrant and grassy look is to create a relaxing atmosphere, which would help avoid creating a feeling of frustration.

For the reasons mentioned above, we also started of by trying to create a calming atmosphere for our first level. To do so, brown earthy tones were used throughout, which have been proven to provide a sense of warmth and wholesomeness [1].

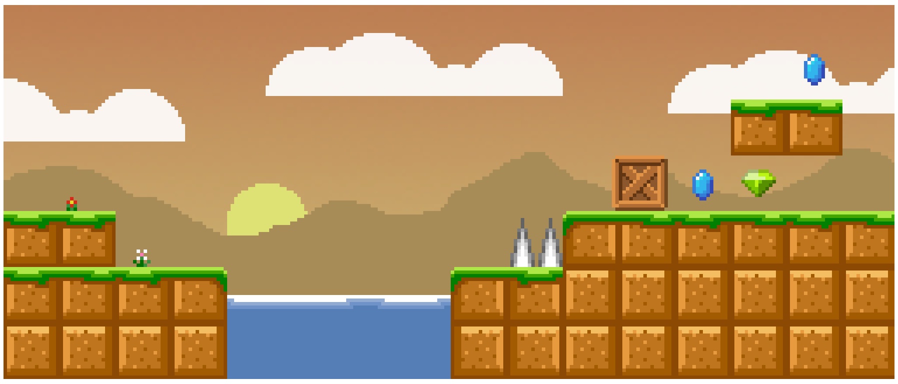


Figure 3.8 - Initial Level Design

For the level design, it was also important to keep everything to a 16x16 tile design, in order to make things easier during the development stage. To create levels, we are using an editor called tiled, which requires the use of sprite sheets that must be tiled. Therefore everything needs to fit into a 16x16 space. It could also be a 32x32 or 64x64, but 64x64 wasn't popular with older games, and wouldn't really create that sense of nostalgia we want. 16X16 and 32x32 were popular though, but 16x16 was chosen because it was in use by more of the games that inspired this project.

During a design workshop, it was mentioned that often when designers are stuck in a rut, it helps to leave and work something else for a few days, and then comeback with a fresh view on the artwork. After doing this, it was decided that the first design looked a little flat and dull. To rectify this, more depth was added to background and foreground, and a more vibrant colour palette was used to for the background (figure 3.9). Overall this was big improvement, and the technique of coming back to design really helped give a fresh view on the overall look.

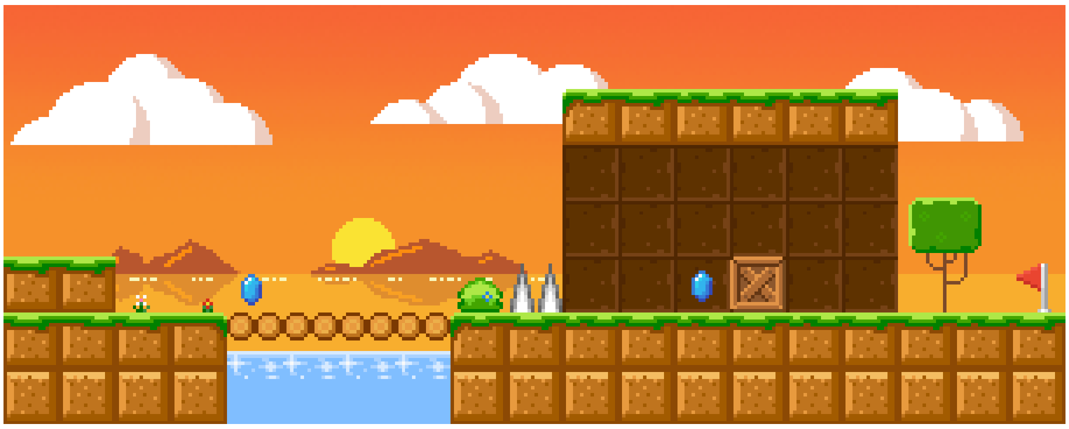


Figure 3.9 - Improved Level Design

However, with design it's important to not just rest on your laurels, so further experimentation was done. One of the results of which, was trying out different times of day in the background (figure 3.10 & 3.11). The outcome of which was really good, and whilst trying to decide which background would be best to use, a new idea hatched. The idea was to use all three backgrounds and have different levels take place in during time periods, each with their own unique style of gameplay.

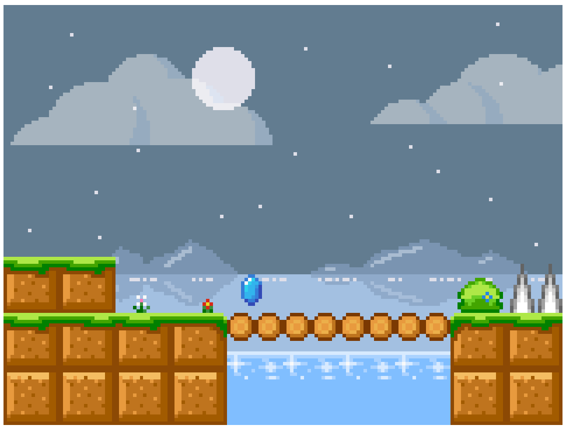


Figure 3.10 - Night Background

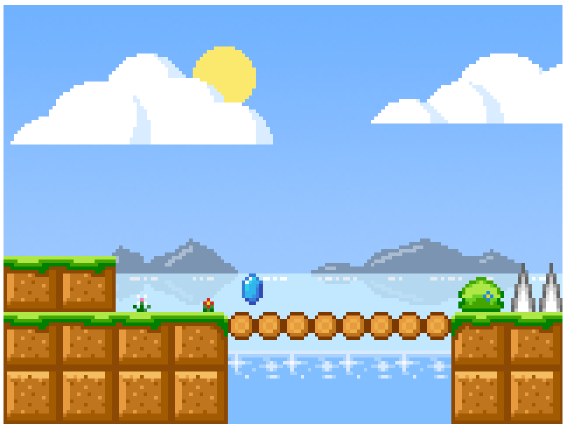


Figure 3.11 - Sun Background

Within games variety is a really important aspect that so far we had overlooked. If a player finds him/herself playing a game for a few hours that lacks variety in it's gameplay, then they're likely to become disinterested due the games repetitive nature. To ensure this didn't happen, each time zone was designed to have it's own challenges. The day stages would concentrate on platforming and puzzles, sunset stages would be a race against time, and night stages would concentrate more combating enemies. Overall this seemed to have a positive effect on the gameplay, based on the positive feedback from the alpha test.

Continuing on with the idea of variation, we started looking at how variety in the environment is also important for the player. If they are stuck looking at the same imagery for a long time, it could begin to feel monotonous. Therefore having different area's and environments allows the game to keep itself fresh and interesting. Different environments also work as milestones within the game. By reaching a new area the player can feel that they've made progress and accomplished something. All of these things are positives in keeping the player interested for the duration of a game.

Each new area also needed to feel different and unique, to keep with the idea of creating variety. Which is why each new zone has it's own unique colour palette. The first zone sticks to warm earthy colours with browns, oranges and greens, whilst the newly designed ice zone (figure 3.12) goes with a colder palette, using greyish-blues, and whites.

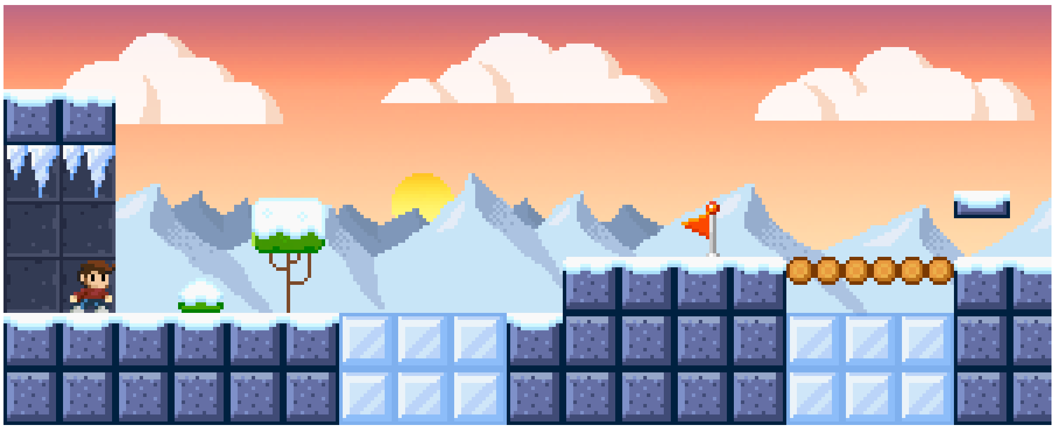


Figure 3.12 - Ice Level Design

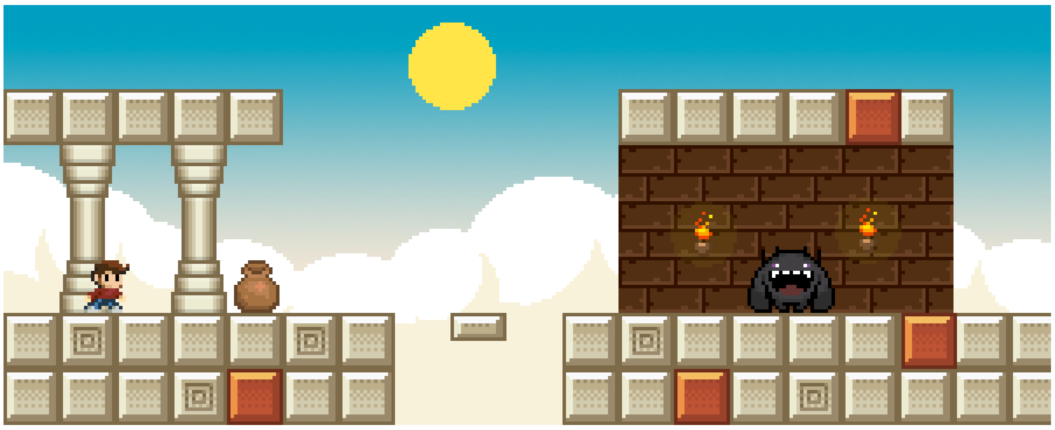
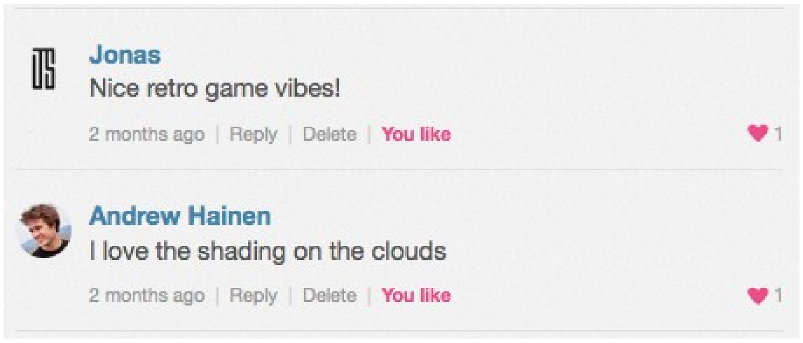


Figure 3.13 - Temple Level Design

To get some early feedback, I posted some of the artwork up on dribbble, to get some feedback from other artists. Overall the feedback was pretty positive (figure 3.14), and the post ended up being one of my most favourited shots. The artwork also received some positive mentions during the alpha test, so overall the level artwork was in a very good place.


Jonas
Nice retro game vibes!
2 months ago | Reply | Delete | You like 1

Andrew Hainen
I love the shading on the clouds
2 months ago | Reply | Delete | You like 1

Figure 3.14 - Dribbble feedback

3.5 Mob and NPC Design

For every mob/enemy in the game, the aim was that they would each be unique, present their own challenges to the player and sort of have their own personality. This idea was inspired by Markus Perrson or Notch, the creator of the hugely popular Minecraft. Ages ago he talked in a blog about how he wanted every mob in the game to be unique, stand out and not just be a re-coloured or re-skinned version of another enemy, something which a number of games today are guilty off. This is an approach that clearly works, seeing as the mobs in Minecraft such as the Creeper have been cult hits across the internet.

In the end four different enemies were created (figure 3.14). Since these enemies are planned to feature primarily in the night time stages, it made sense for them to dark and creepy creatures. Therefore a lot of inspiration came from the sort of things you see around halloween, like ghosts, spectars, bats etc.

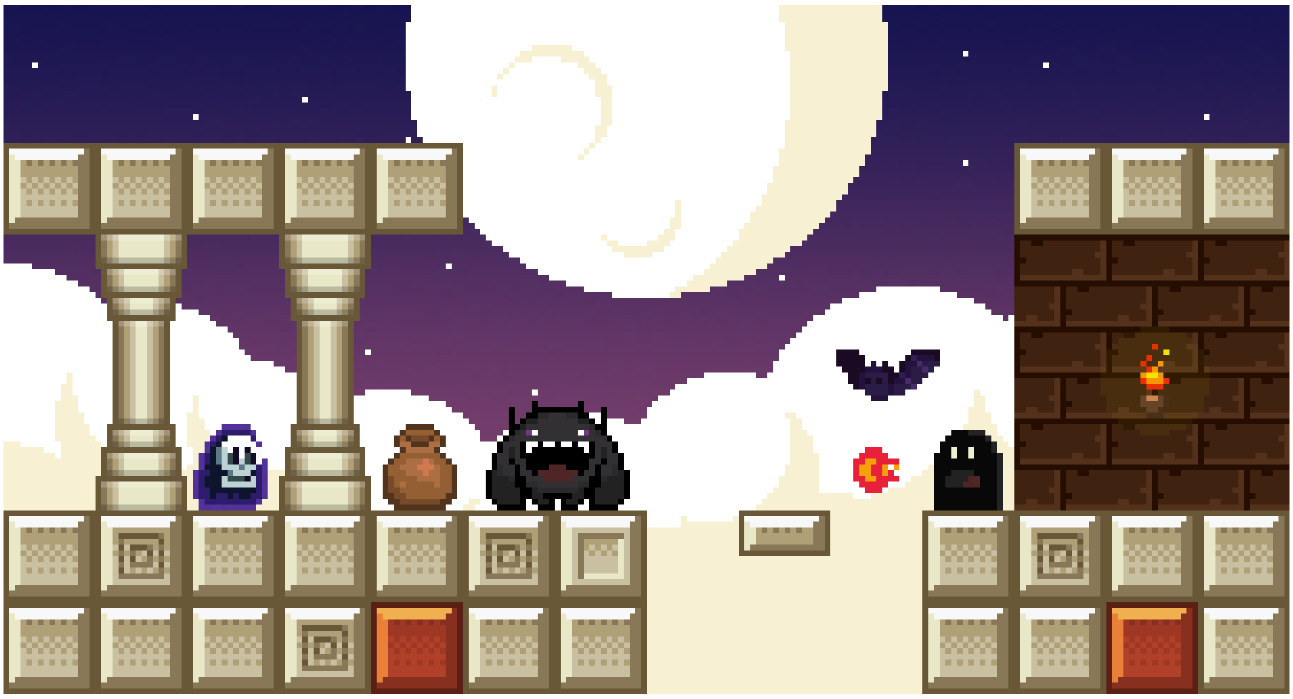


Figure 3.14 - Mob Design

In the end only two of these enemies were featured in the game, which was pretty disappointing seeing as it doesn't provide a good variety of enemies to challenge during the game. The main reason for this was because of a lack of ideas regarding how to implement the larger enemy and the bat. Work on those mobs got put back and eventually the project reached a stage were most of the levels were built and it was to late to go back re-design them to implement new mobs. In hindsight this was poor management, and work on the mobs should've have been finished before the project reached that stage. In the future better planning and clearer objectives could prevent these issues.

Later in the project, when the story for the game started coming together, it was decided the it also needed some NPC's to add life to the world, and to aid with telling the story. It was also important that the world felt populated and not just a baron landscape. For this reason some sketches were done for characters that would be known as the 'Nori'.

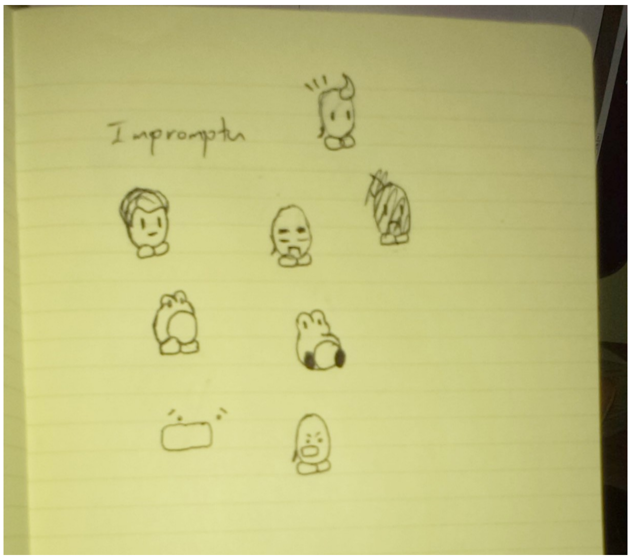


Figure 3.15 - NPC sketches

These were then finished up in photoshop, and given animation frames(figure 3.16). Four different varieties of Nori were created in the end, each with distinctive traits that helped with story telling. For instance, it would make sense that older Nori could provide knowledge to player. Overall the process went very smoothly and the NPC's were a really positive addition to the game.

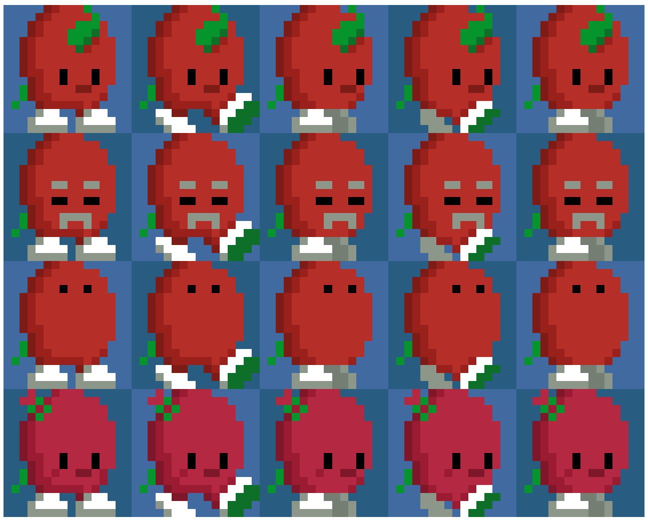


Figure 3.16 - Finished NPC animation sheet

3.6 UI/UX Design

The process of creating a user interface for the project started off with some research into how other games handles their menu screens, in order to get the best direction to take our UI design. It started by looking at how the classic games that inspire this project handled their interfaces. Generally these interfaces were extremely minimalistic, and usually the only option the player had was regarding how many people would be playing. Nowadays though menu screens are bit more crowded, with extra options for the user to explore and toggle.

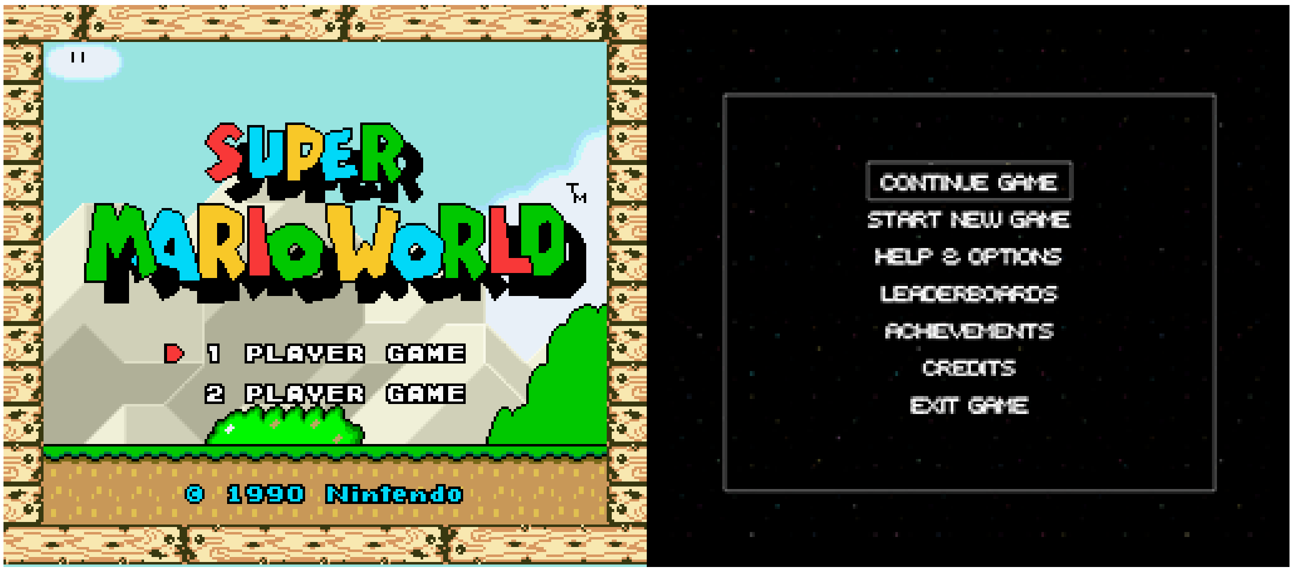


Figure 3.17 - Super Mario World Menu and Fez Menu

However, because this project was originally aimed at mobile users rather the those with game controllers, we started looking elsewhere for ideas, and more towards touch interfaces.

Recently Microsoft released their latest OS, Windows 8, which has taken a drastic departure from the UI used in the previous versions of windows. Typically Windows machines would have a taskbar at the bottom, and application shortcuts covering the screen. Instead they're now presented with whats know as their metro ui, which is aimed at accommodating touch screens with larger application icons dominating the screen. The idea is that Microsoft want to create a UI that bridges the gap between desktops and touch devices, but so far the reception has mixed. The interface has received a positive response from tablet users, but desktop users appear to prefer the previous versions of windows and have been reluctant to update.

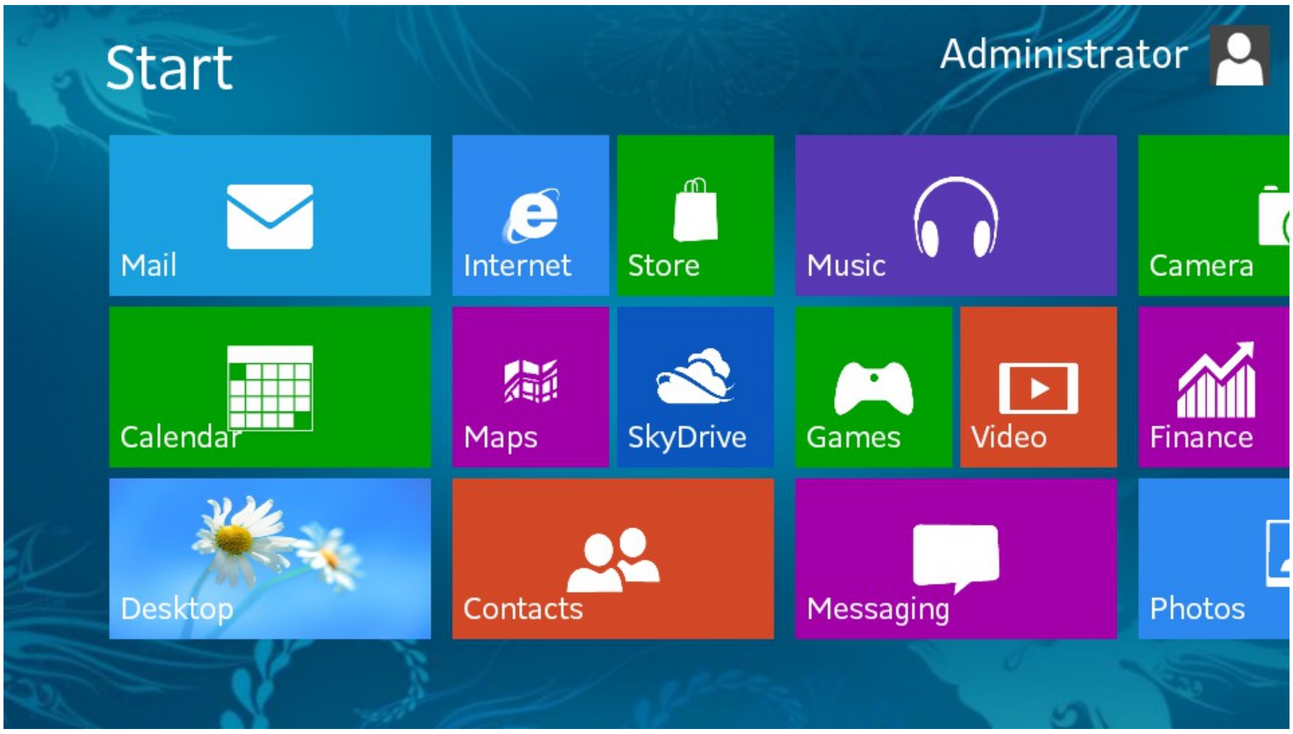


Figure 3.18 - Windows 8 Interface

Generally touch users require much larger buttons to aim at because using their fingers on a smaller screen is nowhere as accurate as a mouse/trackpad on a big screen. Overall we felt the Metro UI provided a stylish solution to the issue and that similar approach would work well on this project. As for desktop users, so little time should be spent looking at the menu screen that it shouldn't bother them, and it's unlikely they'll have an issue trying to use such a simple menu screen.

The initial menu screen (figure 3.19) included 3 buttons that made use of the entire screen. The idea being that we would use as much space as possible to make the screen as easy to interact with on a touch screen as possible. The play button was given extra emphasis here because it's the button that users will be interacting with the most, therefore it is of more importance.

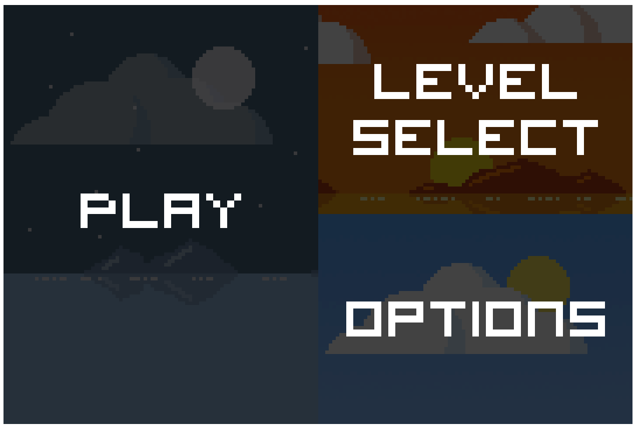


Figure 3.19 - Initial UI Design

Initial thoughts on the first screen were that it was too dark and dull, and didn't fit with the bright nature of the in-game artwork. Therefore it was reworked (figure 3.20) to make more of the background visible, and the most vibrant backdrop from the level artwork was used to fill the background.

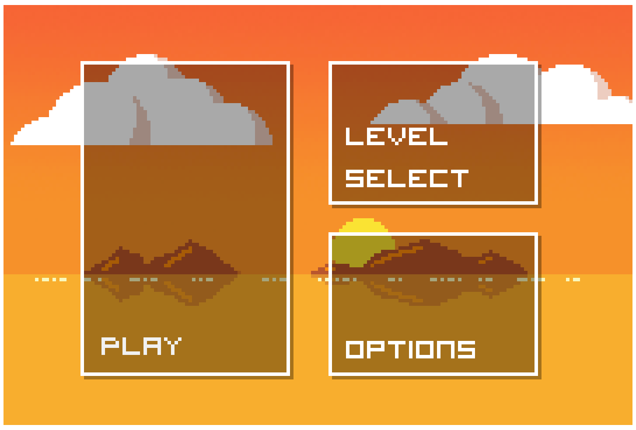


Figure 3.20 - Final UI Design

Later in the project, the mobile version of the game was dropped but the menu was kept the same. During the alpha testing phase, no one complained about the menu so we felt there was no need to waste time re-designing what already worked well.

For the level select screen, something similar to the menu screen was attempted. The background would represent what zone the present levels were from, and each square represent a level. This was a good attempt, but one major flaw was that there wasn't enough room to represent all the information needed. The screen needed to display how many gems had been found for each level, and how many had been found overall.

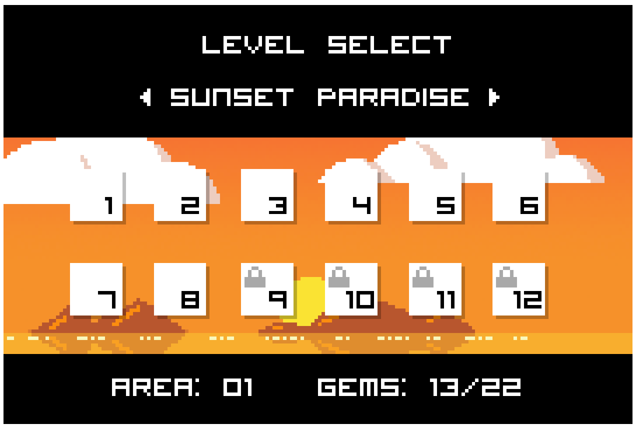


Figure 3.21 - Initial Level Select Screen Design

Instead the menu moved to a scrolling list (figure 3.22), which gave more room to the display the information needed, and any additional information that might be added later.

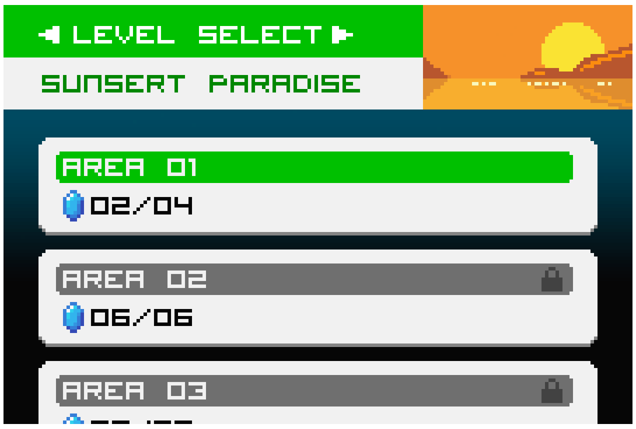


Figure 3.22 - Level Select Screen Design

Later in the project, the select screen was again revisited and given a major overhaul. After considering the users journey, and how after each level was completed they would have to come back to the level select screen, it was decided that this approach would really break up the players immersion in the game by constantly taking them out of the game world. The other option was to skip the level select screen, and simply move the player straight into the next level, but because replaying levels to collect every gem is important, this wasn't really an option. Instead it was decided that the best option would be to have an in-game hub that the player could explore and use access each level.

This meant a design was now needed for an in-game hub. The idea also had to fit into the story, to explain why the player was able to move about the world so quickly. Therefore we came up with idea of having catacombs, which are man-made subterranean passageways, that used to be used by religious practices to move about. It was also important to differentiate which zone each hub belonged to, as not to confuse players. To do this some of tiles from the levels are used their respective hubs (figure 3.23).

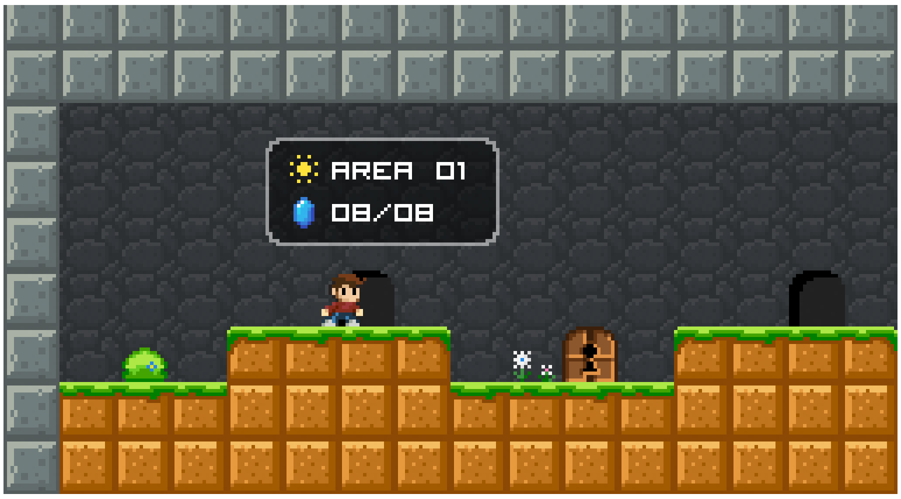


Figure 3.23 - Game World Level Hub Design

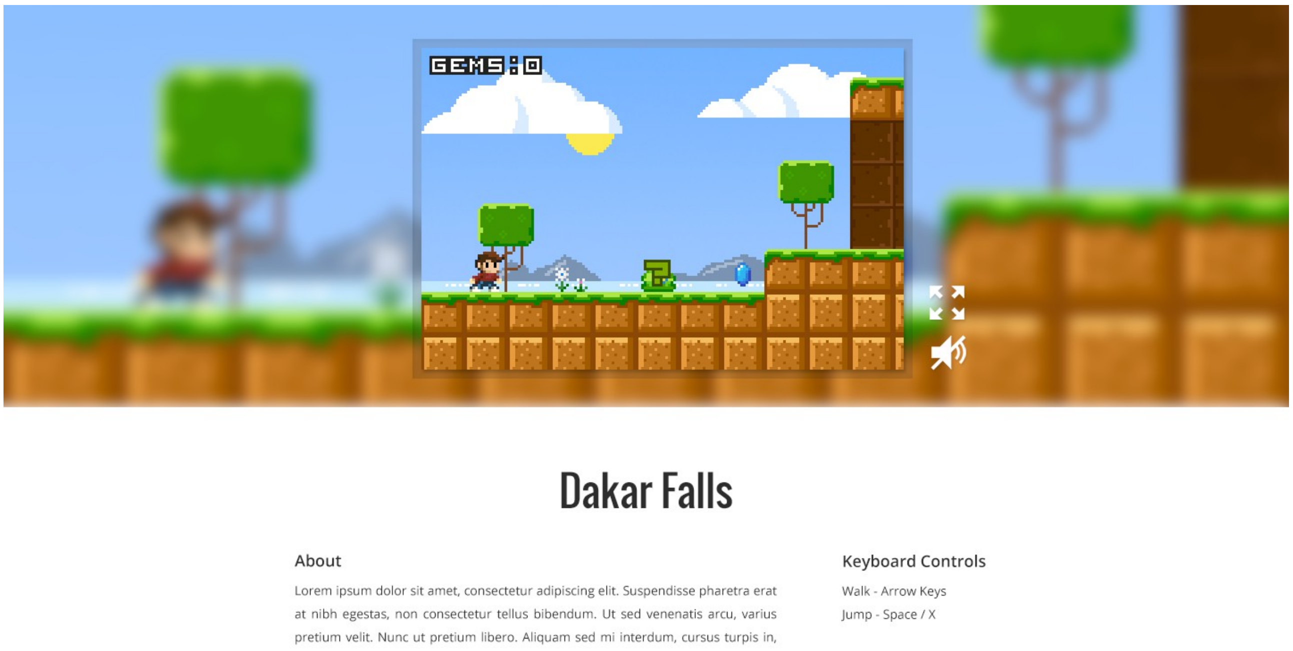
Since replaying levels is important to over-arching gameplay, it's important to display information regarding each level at their entrance. So when approaching a level entrance a window will appear letting the player know how many gems they've collected, the level number, and an icon will represent what time of day the level takes part at. All of these things help the player remember the level, and whether or not they need/what to replay it.

Another bit of research that was done, was regarding how long mobile users would spend playing a game each day. This was in order to get an idea of how long each level should be. Too long and we would run the risk of creating something people don't feel they have enough time to invest in. Generally iPhone users spend an average of 14.7 hours a month playing games, whilst Android users only spend 9.3 hours[4]. This averages out at about 24 minutes a day. On top of that, that time could also be divided up throughout the day, so a player might only spend 12 minutes at a time playing the game. Therefore our aim was to ensure a player could complete 2 or 3 levels within 12 minutes.

However, since the focus of the project switched to a desktop experience, this greatly increases the time we can expect players to spend on the game. To accommodate this many of the levels have been lengthened and now include checkpoints. This ensures the player gets an engaging experience, but at the same time the checkpoints mean s/he doesn't become frustrated by having to repeat large sections of the game when dying, thus avoiding a repetitive experience too.

As had been planned out at the start of the project, there was to be an alpha site to start gathering early information on user interest. The design itself roughly followed the layout of the final site from the paper prototyping stage. The game was placed at the very top of the screen to it would be easy as possible to access and the center off attention. What we wanted to avoid was hiding the game behind a wall of text and potentially putting users of playing it all.

For the actual design of the site we went with a very clean and modern look. A retro sort of pixelated look for the site was considered but seemed to predicable. Instead we wanted to do something little be different and unique to help stand out (figure 3.24).



Dakar Falls

About

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse pharetra erat at nibh egestas, non consectetur tellus bibendum. Ut sed venenatis arcu, varius pretium velit. Nunc ut pretium libero. Aliquam sed mi interdum, cursus turpis in,

Keyboard Controls

Walk - Arrow Keys
Jump - Space / X

Figure 3.24 - Alpha Site Design

Info regarding the game and it's controls were placed just below the game. It was important to make sure the controls were in a visible location so that player didn't start the game simply guessing how everything worked.

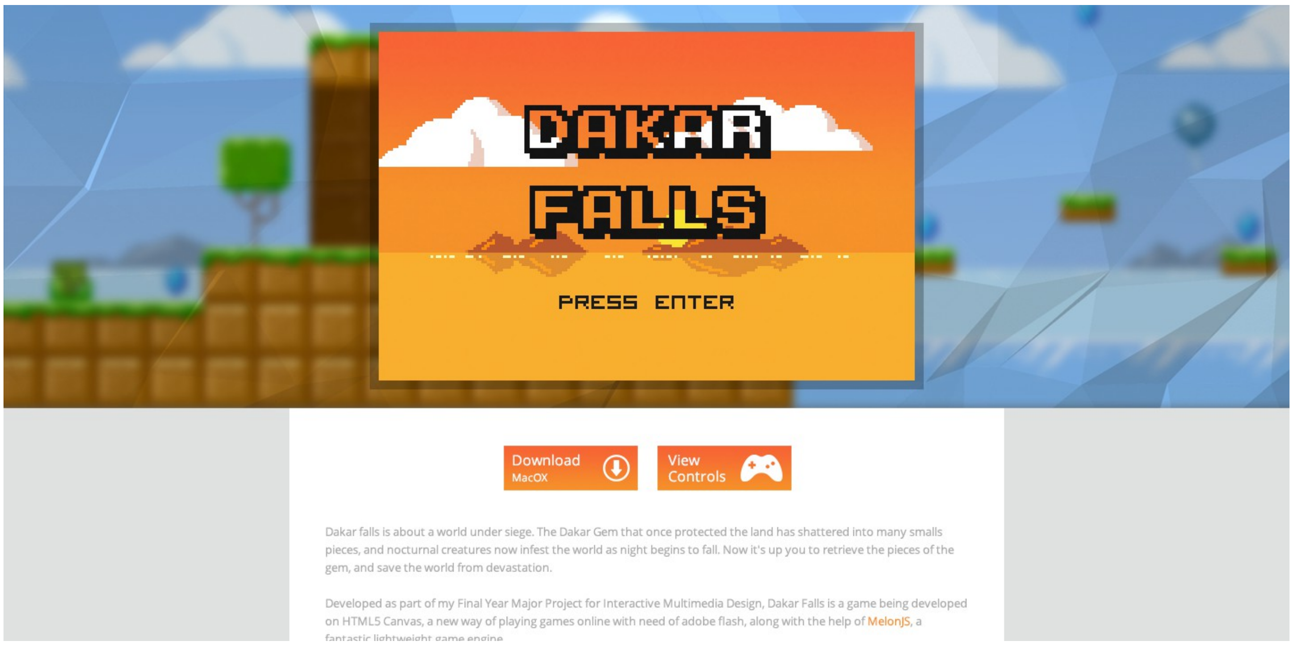
The feedback form then got placed at the bottom of the site. Looking back this was probably a poor decision seeing as no one made use of the form and instead all the feedback was provided through social media and forums. Ideally the feedback form or a link to it should have been visible without needing to scroll away from the game, which would have made it easier to find.

For the design of the final site, much of the layout was kept the same as the alpha site. With there being no feedback form anymore, nothing from the alpha site layout/design stood out as being in need of major change.

One change that was made was to remove the title from below game, and simply display the games title screen above it, thus avoiding unnecessarily repeating the title. The title screen would also be a flat image now, and not generated on canvas. The reason being that for browsers that don't support canvas, we would still need to display something there. Once users click on the title screen or press enter, it would then disappear or alert you that your browser doesn't support canvas.

Using a flat image also enables us to hide all of loading that goes in the canvas before it would be able to display anything, and generally makes everything a bit tidier.

For the background banner we decided to do something a little bit different from the alpha site. Firstly so that people knew this was a different site, and secondly so that didn't think the first banner was simply kept out of laziness, which would create a poor image for the project.



Dakar falls is about a world under siege. The Dakar Gem that once protected the land has shattered into many smalls pieces, and nocturnal creatures now infest the world as night begins to fall. Now it's up you to retrieve the pieces of the gem, and save the world from devastation.

Developed as part of my Final Year Major Project for Interactive Multimedia Design, Dakar Falls is a game being developed on HTML5 Canvas, a new way of playing games online with need of adobe flash, along with the help of MelonJS, a fantastic lightweight game engine

Figure 3.25 - Final Site Design

The banner was created by editing some of the code in the game to make the dimensions of the game screen meet the dimensions of the computer screen, giving us the ability to take large panoramic screen shots of different levels. These screenshots were then overlayed on top of some images generated using an online flat surface shader (http://matthew.wagerfield.com/flat-surface-shader/).

4. Implementation

4.1 Technology/tool selection

In order to develop games on the web, there's a few different choices of technologies to use. The most popular method of doing this has been for many years, adobe flash. Flash for many years has been what users use to view multimedia such games, animations and videos on the web. However more recently developers are being pushed away from the use of flash due to it's lack of support on most mobile devices. HTML5 has brought in a series of new technologies aimed at supporting multimedia through the browser and not through external plugins like flash. One of those new technologies is Canvas which provides developers with the ability to script and render 2D shapes and bitmaps to the web via javascript.

For this game it was decided to use canvas. The advantage canvas has at the moment is that unlike flash it is supported by most mobile browsers. However, the performance of canvas apps on mobile is still a developing area, and most canvas apps actually perform very poorly. But it's an area that should improve and allow canvas to be a much more versatile platform. Another reason for choosing canvas was already having much more experience in developing canvas games, compared to having none in flash which was a major factor in deciding what technology to use.

The game being created is a 2D platformer. This meant the game would need physics to control variables such as gravity and velocity. Features such as these are incredibly time consuming to implement, and a physics engine would be another project in itself, which is why the decision to use a game engine was taken.

The first engine that was considered for the task was impactJS. Out of all game engines that were reviewed, impactJS had the most impressive range of features, with built in support for mobile devices, a level editor and even books to help learn about the engine. But unfortunately it also counts $99 to purchase, which is very expensive, especially when most other engines are open source and free to use. Which is why the decision to not use impactJS was taken.

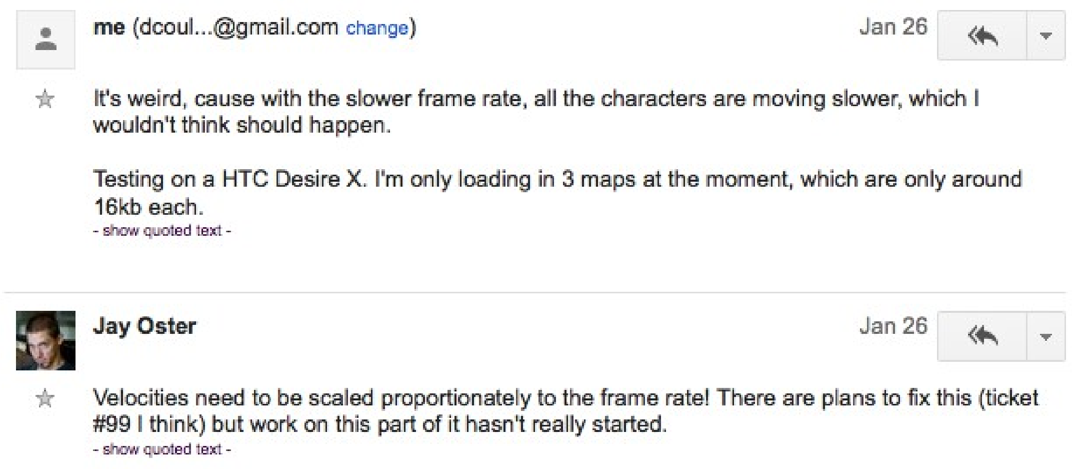
Box2D.js was another engine that was looked and is a port of a physics engine of the same name, written in C++. The physics themselves looked very solid and realistic, and there was a couple of tutorials on how to get started using box2d to develop games, but overall the engine looked fairly simple as the range of features was limited simply to physics and nothing else.

The engine that was chosen in the end was MelonJS. Although there's very few tutorials and documentation online for it, the tutorial that MelonJS provide themselves was really detailed and provided a great step by step guide to creating a 2D platformer. In fact, the engine was mainly tailored towards creating 2D platformers, making it perfect for this project. On top of that, the developers of the MelonJS themselves were really involved with the community, constantly answering questions on theirs forums and helping developers to get started. On top of being a game engine, MelonJS is also a framework. Considering the vast amounts of code that go into a game, it's important to keep it all organised and understandable, and melonJS provides various javascript classes that help organise code.

Overall it would have been hard to have found a better engine to use for this project. MelonJS provided a framework, physics and integration with a level editor called tiled, which was everything that was required for this project. The lack of tutorials and information online was an issue though, since whenever problems with MelonJS arose, it was very rare to find a solution online. However, they developers of the engine do a great job of providing support on their forums and more often than not provided solutions to problems.

In order to port the game to mobile devices CocoonJS was used, which is a deployment app that allows developers to export web apps as native android and iOS apps. After testing out some of their demo's on a phone, it was decided the performance was suitable enough to use for this project.

However, it wasn't until the first working prototype of the game was completed that a serious performance issue was discovered, that meant creating a mobile app wasn't a realistic possibility. What was found was that the performance of the game dropped from 60fps on desktop, to 30fps on mobile, which was a lot more than what was expected based on the fps from CocoonJS's demo's. Although 30fps should still be playable, the actual game speed was being halfed as well. So the player and all other objects were now moving half as fast as they should be. After messaging the developers of MelonJS regarding the issue it discovered that this was an actual issue with MelonJS. Essentially the movement speed of all the game objects should scale to the performance of the game, so at 30fps everything should be moving twice as fast to make up for the reduced number of frames, however in MelonJS the speeds don't scale.


me (dcoul...@gmail.com change) Jan 26

It's weird, cause with the slower frame rate, all the characters are moving slower, which I wouldn't think should happen.

Testing on a HTC Desire X. I'm only loading in 3 maps at the moment, which are only around 16kb each.
- show quoted text -

Jay Oster Jan 26

Velocities need to be scaled proportionately to the frame rate! There are plans to fix this (ticket #99 I think) but work on this part of it hasn't really started.
- show quoted text -

Ideally however, with canvas the game logic and rendering logic should be in separate loops, which they're aren't in melonJS. So any code regarding movement and data should be in a game loop that runs at a set 60fps. Then any code that draws images onto the canvas should be in a requestAnimationFrame loop. RequestAnimationFrame is a new html5 api that was added for canvas, to give developers a loop that adjusts to a computers performance. So whereas requestAnimationFrame would run at 60fps on a desktop computer, it might only run at 30fps on a tablet.

The reason the two loops should be kept separate is to avoid float values. If a developer was to use requestAnimationFrame to run all their game code, they would need to use delta timing to calculate movement speeds. This is where you would calculate the time elapsed between each frame, and multiply that delta by a set value to get an objects movement speed. Therefore the objects speed would be the same at 30fps as it is at 60fps. But this creates a problem where the objects speed would be a float value like 1.345465, and when you move an image on canvas by 1.345465 pixel per second, you find canvas can't render half pixels, so you end up with a very jerky looking game.

Unfortunately there were no plans within MelonJS to fix this issue soon, and there was no reliable fix/patch to the issue, meaning that work on a native mobile app had to be abandoned. Alternatives like using another engine could had been explored, but were not possible with the amount of time that was left at this stage.

Overall this was a fairly big setback. However, in hindsight a smaller and much more basic prototype of the game should have been developed earlier on in a short space of time in order to spot this issue earlier.

After cancelling plans to release a mobile app, research started on the possibility of creating a desktop app. Node-webkit was very quickly chosen as the perfect solution for this. Node-webkit is an app runtime that combines chromium and node.js to run web pages as native desktop apps on windows, mac and linux, and is incredibly simple to use. All the work that was required was copying all the files from the game into the package contents of node-webkit, and then creating a json file that contains default settings for the app. After testing the app the performance was perfect, probably even smoother than the web version, and no bugs were found. Overall node-webkit was perfect for this project and it's incredible ease of use meant it didn't affect the rest of the project in terms of time management.

For the sounds effects and music used in the game, sites like freemusicarchive.org and www.flashkit.com were used to find sounds and music suitable for the game. However, a web app called bfxr was also used to create custom sounds effects for the game. This app was perfectly suited to the project because it was designed to create sound effects similar to the classic 90's game's that inspired this one. On top of that, there was a consistent quality to the sound effects. Usually when downloading sound effects from different sites and creators, you get a disparity in the quality and style of the sound effects. This gives the user a sense that the game was hastily put together and of lesser quality because you have some sound effects are much crisper than others, but by having consistent sound effects provided by bfxr, the project was given a much more professional sound.

When considering how to approach saving players data, databases were looked at as a potential solution, but that would've required users having to register to the site first so we could associate a save file with their account, which was a negative because of many potential users would have turned away at the thought of registering to a site just to play a game. It also would have made development of the desktop app more difficult, because suddenly what was an offline app would need an internet connection. On top of that, node-webkit becomes a much more complicated framework when dealing with databases.

Local databases like indexed db were also looked at, but local storage was chosen because of it's ease of use and better compatibility. However, during development one shortcoming was quickly found and that was the inability to store javascript objects within localstorage. Instead localstorage was only able to store text strings, but a work around was quickly found. Before saving the data, the JSON object would converted to text, using JSON.stringify(), and then whenever it was being received, it would be converted back into a JSON object using JSON.parse(). Memory also wasn't a problem, because across the browsers, a minimum of 5mb is provided for localstorage, a size which despite storing some very long strings, we didn't get near. So overall, localstorage was the perfect solution to our issue.

For this project we also needed to build two small websites, one for the alpha build and one for the final release. For both of these little consideration into what front-end technologies was needed as pretty much every website runs on a front-end of html, css, javascript and jQuery. However, recently css compilers have become very popular amongst developers. What these allow people to do is write css using some nicer syntax, variables and some low level scripting.

Variables in css have become very useful because they allow developers to store properties that will be used over and over again such as hex-codes. The advantage of this is that hex-code's are typically hard to remember and reading them on screen doesn't tell you what colour they are. However by storing them in variables with names like blue, css becomes much easier to use, read and understand. Additionally if you ever need to change a colour throughout a site you can do it by changing one variable, which saves developers a tonne of time.

For this project Sass was chosen because compared to it's closest competitor Less, it has much bigger range of functions and is currently more popular among developers, meaning it's more likely to become a standard tool for developers such a jQuery at some point.

Another tool that utilised throughout the project was Github. Github provides users with a free online repository where they can store code and version control it. This was important for the project incase something went wrong and original code was lost, because it provided a backup that was easily re-downloaded. Had the code for this project been completely lost at any point it would have been catastrophic because of the amount time it would taken to completely re-write.

The version control that Github provides is also incredibly useful incase at any point a bug is proving to difficult to simply fix or locate. In this instance developers can roll back to a previous version where this bug doesn't exist. With each new commit to a repository a change log is recorded that contains each line of code that was changed between versions. These are very useful for cases where something is broken in the latest version of the code but not in the previous one. Developers can simply look through the change log and narrow down where the problem could be coming from.

Since these repositories are also public, they are very helpful when asking for assistance from other developers. Often enough simply trying to explain a problem to someone in words isn't the most efficient way for them to understand the problem. But providing them with a link to the repository allows them to look through all the code and properly understand where the problem could be coming from.

4.2 Technology/tool use

As a framework, melonJS provides a number of javascript classes that can be extended to create objects for the game world. These classes contain a number of default values and preset functions that can be used and modified. For instance, for the main player a class of objectEnitity would extended. This class already contains an init, collision, update and draw function that will be called by the engine if the object is present in melonJS's entity pool. This entity pool is simply a list of all the objects used in the game.

These classes and functions also help to structure the code used in the game. Every object in the game has three main functions that will be called, which are init, collision, and update. This makes it easier for developers to come back later and look at code they wrote weeks ago, and be able to find exactly what they're look for, because it's all organised into clearly defined sections.

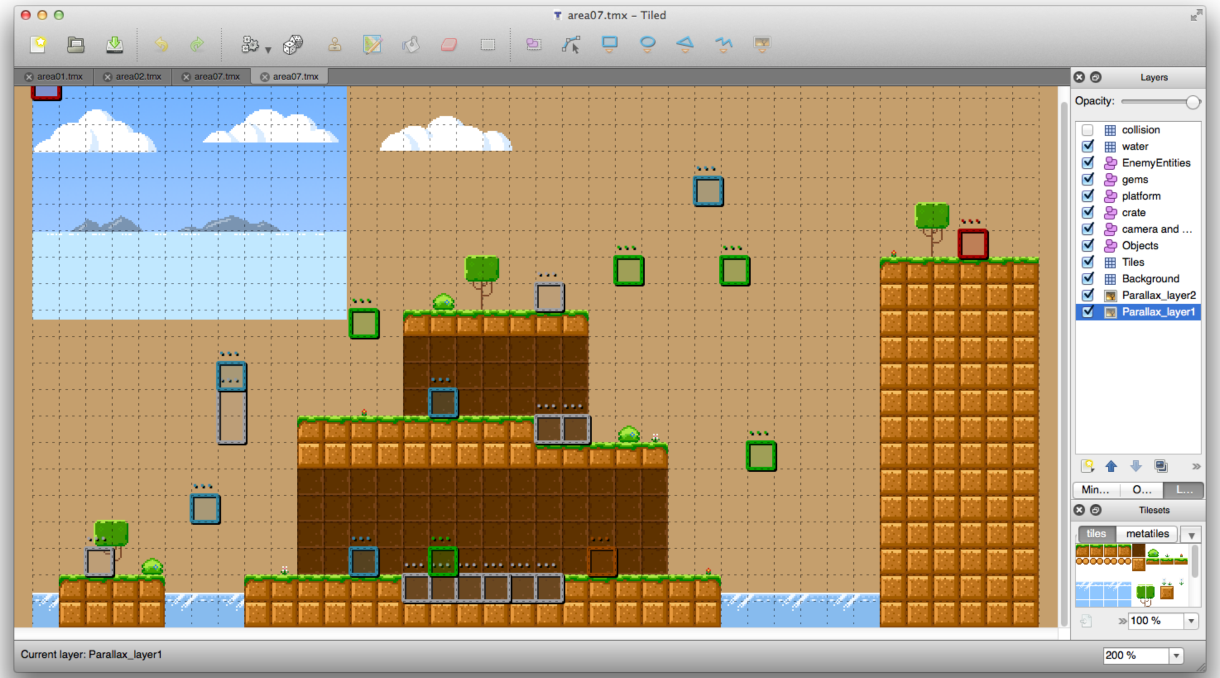
Init encompasses all the code that will run when an object is pulled into the game. Typically this would include it's positioning, size, image and animation frames. Collision is then the function that gets called when an object makes contact with another object. So if an object was to collide with a wall, this is were the code that stops the object moving any further would be placed. It's also important than an objects collision function only contains code that affects itself and not other objects. So for instance, the wall object would not contain code telling other objects to stop once they hit it. This is because if there's an error during a collision, such as an object flying off screen when it shouldn't, you know the error is code for that object and not elsewhere. Otherwise developers could spend forever searching through other objects looking for this error.

Finally, the update function contains code that runs on every game tick. Mainly this includes the logic that decides what velocity an object should be moving at, so if a player is pressing right, move the main player right at a velocity of 2.

MelonJS divides it's code up into 4 main sections. Game, Resources, Screens and Entities. The first of these is a file called game.js which acts as a hub for the game code to be pulled into. It also contains global variables, initialisation data such canvas dimensions, a list of all the objects in the entity pool, and key bindings.

Screens is then a folder containing files for each screen in the game. These different screens include the menu screen, intro screen that shows the opening story, and game screen that displays the actual gameplay. Screens are essentially different states for the game that define what code should currently be running. For instance, during the title screen, only objects used during the title screen should be pulled in and run, and we don't want the actual game running in the background. So within these screen files we can extend a class of ScreenObject and from there define what objects need to be pulled in by writing me.game.world.addChild(this.TITLE), where this.TITLE is the variable containing the object we're pulling in.

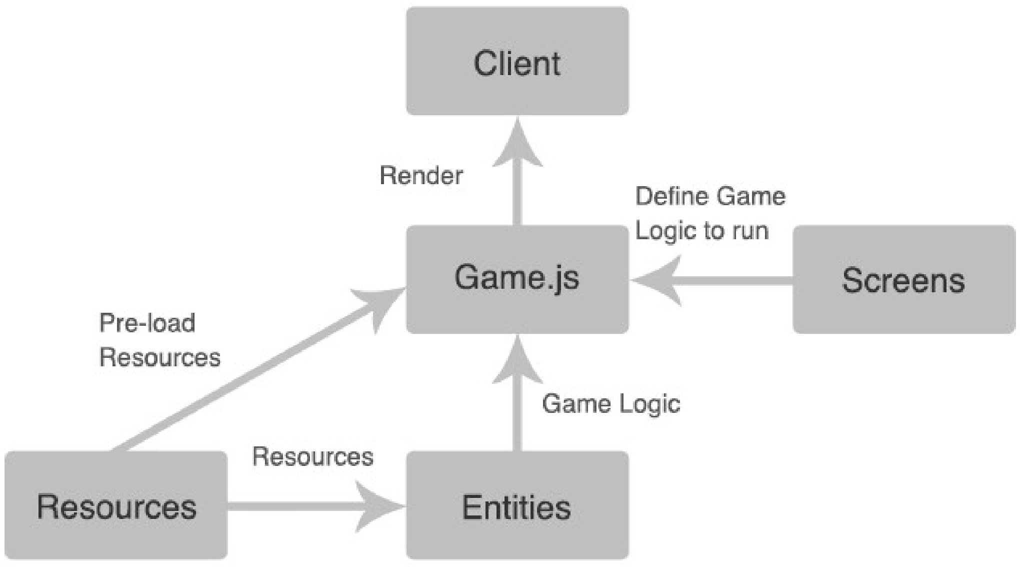
For the actual gameplay screen it's slightly different though. Rather than pull individual objects, melonJS allows developers to pull in a .tmx file. These are level files that can be created using a GUI editor called tiled. Within tiled a user can draw a background and a add in objects that have been defined in the code. MelonJS looks at these .tmx files, and pulls in all the objects that are present in the file.



Having a GUI level editor was really important for the time management of this project. Having to write out the code for a level from scratch would have taken considerably longer compared to using tiled. It would also have made making amendments to levels much more difficult, seeing as it's a lot easier to understand a GUI editor than trying to comprehend how code would render on screen.

Resources is then simply a file that lists out all of the images, sounds, and level files that are used in the game. On load melonJS pulls in this resource list and preloads all the data for best performance.

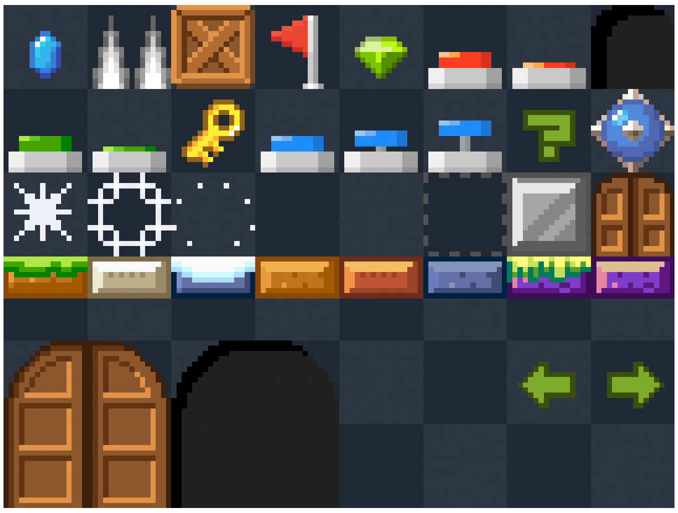
Finally Entities is a folder containing all the files for our game objects. Each game object uses a class from melonJS and is stored in a global variable. Since these variables are global, essentially they could be defined anywhere. But for the sake of structure and maintaining understandable code, they are kept in the entities folder.



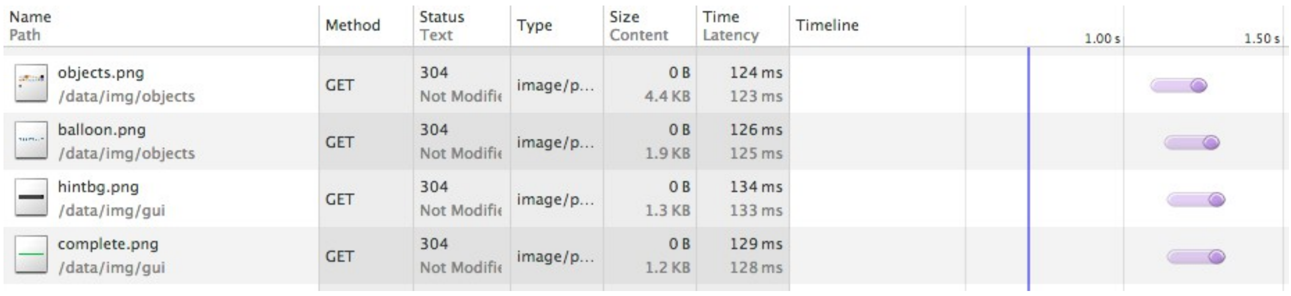
When loading images into a web page, the browser makes a DOM request. For each image the browser has to make a new DOM request, meaning if you have loads of images, multiple DOM requests are made and can take up a substantial amount of loading time. The size of the image doesn't affect the length of this request, so loading in many small images can really hurt the loading time of a web page. To reduce the number of requests developers can use sprite sheets.

Sprite sheets are images that contain multiple images. By using these instead of multiple images, we reduce those multiple requests down to a single request which greatly improves the loading time. Then within the code we have to dissect the image up to display only the image we want. In the case of css, you would set the sprite sheet as a background image for an element, and then use the background-size and background-position properties to define the area of the sprite sheet to display.

This technique was used during this project for most of the images in the game. The project contained many images and the number of individual image files for objects alone would have been around 32.

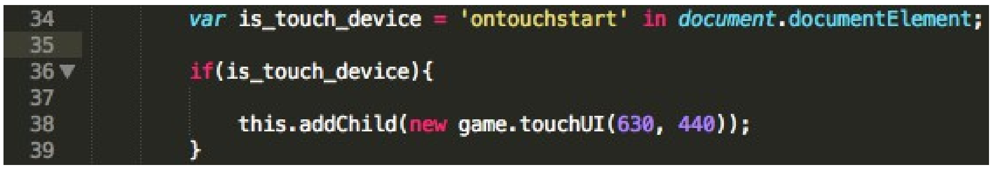


Within google chrome you can use the network tab on the developer tools to view how each resource is being loaded into a website. Below you can see that objects.png, which is the image above, is taking around the same time to load as hintbg.png and complete.png, two images which aren't on spritesheets. If objects.png was 32 different images though, it would take longer due to the high number of requests it makes and then fact that browsers will only run a certain number of requests at the same time.



Name Path	Method	Status Text	Type	Size Content	Time Latency	Timeline 1.00 s 1.50 s
objects.png /data/img/objects	GET	304 Not Modifie	image/p...	0 B 4.4 KB	124 ms 123 ms	
balloon.png /data/img/objects	GET	304 Not Modifie	image/p...	0 B 1.9 KB	126 ms 125 ms	
hintbg.png /data/img/gui	GET	304 Not Modifie	image/p...	0 B 1.3 KB	134 ms 133 ms	
complete.png /data/img/gui	GET	304 Not Modifie	image/p...	0 B 1.2 KB	129 ms 128 ms	

Whilst development was still going on for a mobile release, touch controls were developed for devices without physical buttons to control the game. A lot of this work was handled through melonJS's .registerPointerEvent() method, which allows developers to bind functions to designated pointer events, such as mousedown and mousemove. Whenever a user touches the canvas, we are able to get the x and y co-ordinates of where they touched through the event argument that .registerPointerEvent provides. If these co-ordinates match the co-ordinates of an onscreen button, then we trigger a key event using .triggerKeyEvent. This is another method provided by melonJS which allows us to trigger a key press, so we can essentially emulate a user pressing the x or space keys, allowing them to control the character in the game.



```
        var is_touch_device = 'ontouchstart' in document.documentElement;

        if(is_touch_device){

            this.addChild(new game.touchUI(630, 440));
        }
```

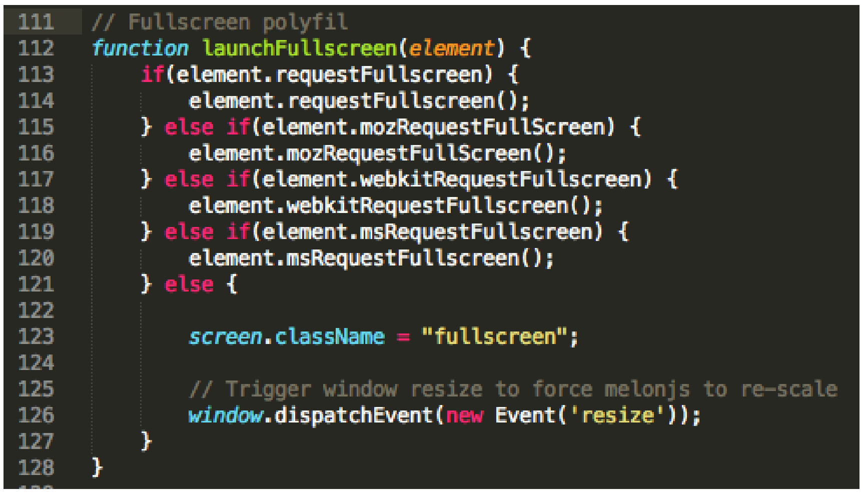
In order to hide the touch controls from non-touch devices, we detect whether a device is touch or not. It's important here not to just target mobile devices, largely because of how the touch device landscape is changing. With windows 8, Microsoft are now trying to provide a touch experience to desktop computers, meaning any device today or in the future could feature a touchscreen. Therefore to detect if a device is touch capable, we grab a reference to the root element of the document using document.documentElement and check if it contains the method ontouchstart. This returns a boolean, that we can then use to either start the touch ui or not.

Although plans to release a mobile version of the game were dropped, the landscape for touch devices as already mentioned, is changing. The game is still being released as a web experience, and some of the more powerful tablets are able to run canvas games fairly well, so this still provides a useful feature to users. In the future, it may even prove a useful feature for desktop computers with touch functionality. In that sense, it makes the game more future proof, which adds to it's longevity without having to constantly update it.

4.3 HTML API's

Throughout the project a couple of new HTML5 javascript api's have been put into use, one of which was the fullscreen api. This allows developers to trigger a fullscreen mode for a webpage that's then handled natively by the browser. Developers can now target certain elements and use .requestFullscreen() to present it in a fullscreen mode. This technology was perfect for this project because by targeting just the canvas element, the game itself became fullscreen and not the entire webpage, thus removing any distracting elements surrounding it.

Since the fullscreen api is fairly recent, most browsers require you to use a prefixed version of .requestFullscreen. So for firefox the function looks like .mozRequestFullscreen and on chrome and safari it would be .webkitRequestFullscreen. Unlike in css where you can just use both prefixes to solve the issue, using . mozRequestFullscreen will cause errors in chrome, and vice versa. To get around this issue you have to use a polyfill, which is a bit of code that detects which prefixed function works for the browser your using. It does not however, detect what browser your using which is important, since if it did, and firefox decide one day to switch from .mozRequestFullscreen to just .requestFullscreen, then your polyfill would break.



```
// Fullscreen polyfil
function launchFullscreen(element) {
    if(element.requestFullscreen) {
        element.requestFullscreen();
    } else if(element.mozRequestFullScreen) {
        element.mozRequestFullScreen();
    } else if(element.webkitRequestFullscreen) {
        element.webkitRequestFullscreen();
    } else if(element.msRequestFullscreen) {
        element.msRequestFullscreen();
    } else {

        screen.className = "fullscreen";

        // Trigger window resize to force melonjs to re-scale
        window.dispatchEvent(new Event('resize'));
    }
}
```

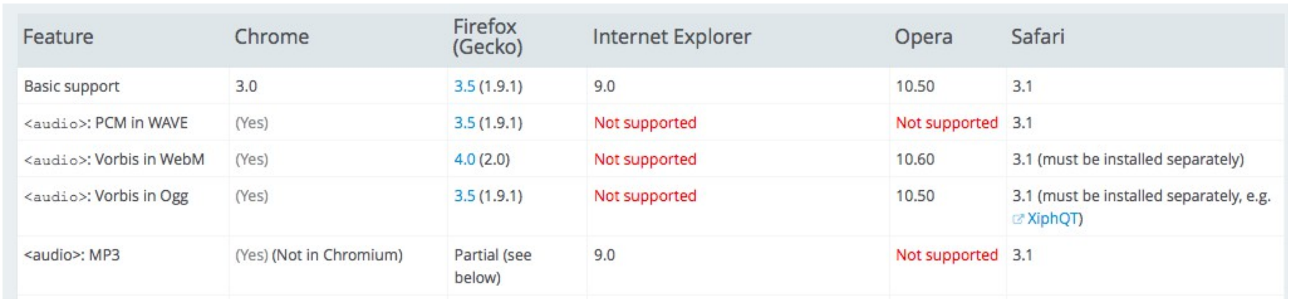
Above is the polyfill that was used in this project. When the launchFullscreen function is run, it simply checks for the existence of one of the requestFullscreen methods, and once it finds one, it runs the method on the element from the function argument. Currently the fullscreen api has really good support on desktop, and almost none on mobile devices. In the event that fullscreen isn't supported by the browser I added in a fallback that simply adds a class of fullscreen to the div that contains our canvas. This class then has a number of css rules that will make the canvas the size of the viewport. Although it's not a complete fullscreen solution, it still does a good job of imitating a fullscreen mode.

Very recently browsers have started to adopt the gamepad api, another feature of HTML5. This allows developers to access the input information from a range of usb game controllers through a javascript object that returns the current state of each button. This was of great use to this project as it allowed us to make use of a technology that gamers are very familiar with. In many cases, users who are interested in games will prefer to use a gamepad over a keyboard, because it's what they're used to, with console gaming being a bigger market than pc gaming. Also, because we're releasing the game as stand-alone desktop app, it will be competing with other pc gamers, the vast majority of which also support usb gamepads, so it's important that we don't provide less features than they do.

Currently only chrome and firefox are supporting the gamepad api, and both provide very different methods of accessing the controller information. With chrome, developers can use navigator.webkitGetGamepads to return a javascript object containing an array with every button and it's current state. By using a looped function, you can constantly check if a button has been pressed or not, and act on that information.

The gamepad api was a great addition to the project, and currently works perfectly on platforms were it's supported. However, it's unclear if firefox and chrome will continue their unique approaches or adopt another one. It's also unclear how other browsers will go about implementing the gamepad api. These kinds of changes could in the future break parts of the project, which is unfortunate because they are difficult to predicate and prepare for at the moment.

Towards the end of the project it was noticed that all of the project files together took up around 50mb of space. After some analysis it was discovered that most of the space was being taken by music files from the game. There was also two versions of every sound file, one mp3 and one ogg, in order to provide audio support over multiple browsers, which took up even more space.



Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	3.0	3.5 (1.9.1)	9.0	10.50	3.1
`<audio>`: PCM in WAVE	(Yes)	3.5 (1.9.1)	Not supported	Not supported	3.1
`<audio>`: Vorbis in WebM	(Yes)	4.0 (2.0)	Not supported	10.60	3.1 (must be installed separately)
`<audio>`: Vorbis in Ogg	(Yes)	3.5 (1.9.1)	Not supported	10.50	3.1 (must be installed separately, e.g. XiphQT)
<audio>: MP3	(Yes) (Not in Chromium)	Partial (see below)	9.0	Not supported	3.1

The idea of reducing the project to using just one type of audio file was brought up, but after checking a compatibility table for audio files in browsers it decided this wasn't possible because Safari and IE would not support ogg, and Firefox only had partial support for mp3. In the end though we did lower the bitrate on the music files to reduce the amount of space they took up. This also wasn't the worst issue seeing as MelonJS only loaded the audio files supported by each browser.

However it was decided that packaging both mp3 and ogg on the desktop app wasn't necessary because node-webkit utilises chromium, which only supports ogg. Removing mp3 files from the desktop really reduced it's size which was important because it also reduced the amount of time the app with take to download for every user.

4.4 Multiplayer Challenges

Later on in the project, a multiplayer aspect was added to the game, which provided a number of small technical challenges. The first of which was how would the camera follow both players. This could be solved in a number of possibly ways, all with their own advantages and disadvantages. One solution was to have the camera zoom out the further apart the two players are, but this wasn't possibly largely because melonJS doesn't provide support for zooming the camera in and out, and writing that logic into the library would be very difficult and time consuming. Another solution is center the camera on just one player. However the problem is that you have to ask the second player to follow close by to the first, limiting the possibility to have puzzles where two players split up and perform different tasks. In this case the gameplay becomes too similar to the single player experience, and doesn't provide a new variety of gameplay.

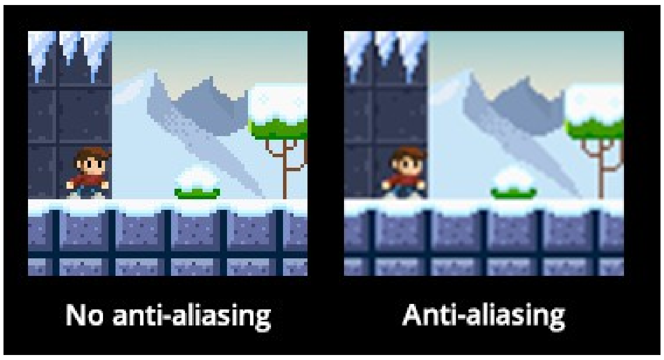
In the end a system was devised where the camera follows an invisible object that is positioned between the location of both players. Typically with melonJS, you can use their viewport object to get the camera to follow a specified object (me.game.viewport.follow(object)). This means you can't follow two objects. So instead there is now an invisible camera object that the game follows, who's positioned is calculated as the central point between both players. The calculation for which looks like camera.x = player1.x - (player1.x – player2.x)/2. The advantage now is that the camera adjusts itself, so both players can be close together or far apart and still both be on screen. Overall this opens new possibilities for gameplay, where both players have to help each other complete puzzles.

With multiplayer now implemented, it was also expected that players would want to use two gamepad controllers to play the game. Implementing this was fairly easy though as chrome provides two gamepad objects in an array if two are connected, so adding support for two controllers was simple as detecting if there was more than one object present in this array.

4.5 UI/UX Developments

Whilst creating the menu screen for the game, there was two options for developing it. The first was to create it with html and css, and the second option was to draw it onto the canvas. Using html and css was the much easier and quicker option, and seeing as both would produce the same result, it was also the chosen option.

Unfortunately whilst working on the menu screen, a bug in webkit browsers was discovered. Developers should be able to turn off anti-aliasing on image-scaling using -webkit-optimize-contrast, but this has no affect. It has been reported fixed in webkits bug tracker, but as the comments show, that is not the case [2]. The reason such a feature is needed, is because the menu needs to be scaleable to it can be displayed at different resolutions. For instance, the desktop app comes in a window that can be scaled to any size. With pixel art, anti-aliasing harms the quality of the artwork greatly on scaling, as you can see below.

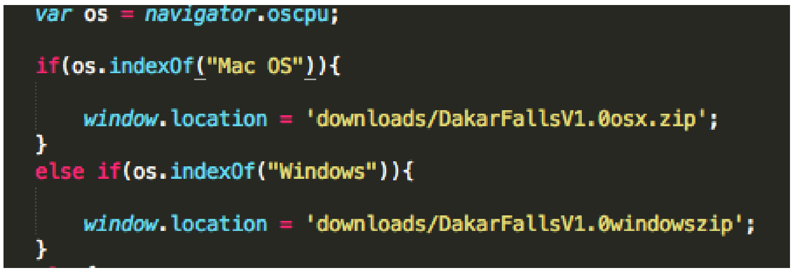


This was a big issue, because it would affect any users running chrome, safari, or the desktop app (which runs chromium). It was therefore decided to render the menu screen on canvas, where you could properly disable anti-aliasing on webkit browsers. This lengthened the time spent on developing the menu screen greatly, but unfortunately there was no other option in this situation.

For the actual site that will hold and promote the game, there is no backend required for any of it's features. This meant only the front-end of the site need developed, making the process very quick and easy. However, there were a couple of notable css and javascript challenges getting it to run.

The first challenge was provided the download button that would provide users with either the windows or mac version of the game. This was preferred over having two buttons, one for each version, because it makes it easier for user when they don't have to figure out what version they have to download. Within javascript calling the variable navigator returns a preset variable containing lots of information regarding the browser, os, and functions that are enabled (I.e cookies). One property of which is .oscpu, which returns a string containing the operating, system and it's version number.

This was perfect for what we needed, but the string that was being returned didn't simply say windows or mac. Instead it would say something like “Intel Mac OS X 10.9” or “Windows 7”. Since it wasn't reasonable to compare every possible string that could be returned, the code written simply checks the string for the keys words “Windows” or “Mac OS”, and provides the appropriate file to the user. This was good addition to the site in terms of making it more user friendly and was very easy implement.


```
var os = navigator.oscpu;

if(os.indexOf("Mac OS")){

    window.location = 'downloads/DakarFallsV1.0osx.zip';
}
else if(os.indexOf("Windows")){

    window.location = 'downloads/DakarFallsV1.0windowszip';
}
```

Unfortunately due to a lack of access to a Linux machine, we weren't able to compile and test a Linux version of the desktop app. To cover this there is an additional clause in the if statement to let users know their browser is not supported.

Another challenge was getting the css for the game's fullscreen mode setup properly. Whenever fullscreen mode is enabled using the new html5 api, it targets one specified element to display on the screen and hides the rest of the site. Applying fullscreen mode without any additional css would mean that the specified element (in our case the container holding the canvas element) would remain the same size and sit in the top corner of a black or white screen, which is far from the desired effect.

Fortunately developers can use :-webkit-full-screen or :-moz-full-screen to define fullscreen specific css rules, so for the game which changed it's width and height to 100% to stretch the size of the screen. One surprising difficulty here was that none of the rules that were written were being applied. After a while it was discovered that :-webkit-full-screen was not a prefix like many other css rules such as :hover or :active, but was instead a reference to the element targeted by the fullscreen mode. After understanding this though, everything was easy enough implement.

5. Testing

5.1 Testing Approach Selection

During this project, the game has undergone two major stages of testing. The first was an open-alpha test, used to perform some initial bug testing and gauge user interest that would used to direct the project. The second was a closed-beta used to test the overall integrity of the game before it's final release.

Both of these tests were black-box tests, where the users required no knowledge of the existing system structure that they were testing. This was the preferred approach to the first alpha test because it was the best testing approaching for getting as a wide a set of results as possible. Alternatively white box testing would have limited to test subjects to only those who could understand the system architecture. Since the aim of the alpha test was to gauge user interest, it was important to retrieve a large variety of results to insure they were as accurate as possible.

For the beta test, there was the option to use a white-box approach and get developers to create test cases to test the actual code for the game. However, due to the complexity and large quantity of code in this project, it was decided that it was not a realistic option because of the length of time it would have taken to do throughly and the limited time remaining on the project. Instead a black-box method was used again, which was fine because it should show up most of the issues that the majority of users would come across.

During testing, the alpha test was an open test, meaning any member of the public could opt to the test game. This was again because we wanted to get the largest return of information in order to enhance the accuracy of the results. Otherwise the project would run risk of relying on inaccurate data, and working on changes that might be received negatively by users.

The beta test on the other was more about checking the integrity of the game. Again, more results still means more accuracy, but leaving this test open to the public diminishes the impact of a release day because so many people have already played the almost finished game, and wouldn't have much interest in playing the same game a few days later. So instead the plan was to ask a number individuals to throughly test the game, report any issues and to have them fixed before the main release.

5.2 Testing Process

For the alpha test, a public website with the game and a feedback form was released. The feedback form was an open ended one where users could leave whatever feedback they wanted. This was preferred over having a rating system or a questionnaire. A rating system wouldn't have provided enough detail about exactly what users liked and didn't like, and a questionnaire might have put some users off leaving any feedback because they can sometimes be time-consuming.

To make people aware of the open-alpha, messages regarding the site were posted through various social media sites and forums. In the end, all of the feedback ended up coming through social media and forums rather than the feedback form on the site. Although this could be viewed negatively, overall it was pretty positive because we still received written feedback like we wanted one way or another, and there was no harm having the feedback form there incase anyone wanted to use it.

For the beta test messages were again put up on social media and forums asking if anyone would be interested in testing the game. After that we ended up with 8 participants who were all provided with a beta copy of the game. Some were testing the desktop app on windows or mac, whilst others tested on browsers, either Firefox, Safari or Chrome. This was to ensure the project working consistently across all platforms. They then simply had to attempt to complete the game and report any issues with the game, such as bugs or levels being too difficult to complete. Once these issues were fixed, the testers were provided with an amended version of the game to insure and all issues had been resolved.

The beta test worked out alright, although there could have been a more structured way for reporting bugs, rather than people just sending emails with issues. Instead we could have used an issue tracker like donedone so people could report issues, and I could notify them which issues I'm currently working on, and when they're ready to be retested. Overall this would make it easier to keep track of many issues need looked or retested, and help with time management.

5.3 Test Results

After the alpha test we had 10 replies, which was actually quite disappointing in terms of quantity. During the test I was running google analytics on the page to check how many hits the site got each day. Over a 3 day period the site received about 40 hits, meaning ¼ of people responded with feedback, which wasn't a bad return. Instead the issue was clearly more to do with how well the test was marketed to people. In future it would beneficial to broaden the number of sites messages were posted on to increase the number of hits the site got. It would also have helped to be more aggressive in posting multiple messages on social media such as twitter, in order to ensure as many people see the message as possible, but also not too many as to become overbearing and annoying to people.

Below is a table with all of feedback that was received and where it sourced from.

Source	Name	Feedback
Dribbble	David Grimes	Reminds me of Commander Keen.
Dribbble	Emanuel Serbanoiu	Really liked the game! :D
Dribbble	Jonas	Nice retro game vibes!
Dribbble	Andrew Hainen	I love the shading on the clouds
Twitter	Stuart Kennedy	this is class, you should a a wee timer for speed runs! :P
Twitter	Stephen McCann	really cool so far lad!!
Twitter	Barry Hassan	Nice one! I was trying space bar to jump but that defaults as page down on Macs, so had to use the X key. Looking good!
MelonJS Forum	Jay Oster	Excellent graphics! And I was very pleased with the time of day progression throughout the stages. I found later stages extremely difficult, especially where you must jump over two sets of spikes that are up one level, and the last stage with the gauntlet of fireballs. I'm not great with timing based maneuvers, so these were the most difficult parts for me. I also had trouble finding ALL the gems on some stages. They just aren't in the general path to the end.
MelonJS Forum	Ellison Leao	Very nice game! Congratulations dude! Great graphics and gameplay.
MelonJS Forum	Olivier Biot	i personally found it quite advanced already for a quick demo :P and it's using new features from melonJS 1.0.0

		kudos for that, not to mention the gamepad support (not from melonjs though) very very good job, i love it and I can't wait how it will be once you call it a finished product

A lot of this feedback was taken on board throughout the project, but some of it was also neglectable. For instance the comment saying the game was reminiscent of another game called 'Commander Keen' was strange seeing as the games look very visually different.

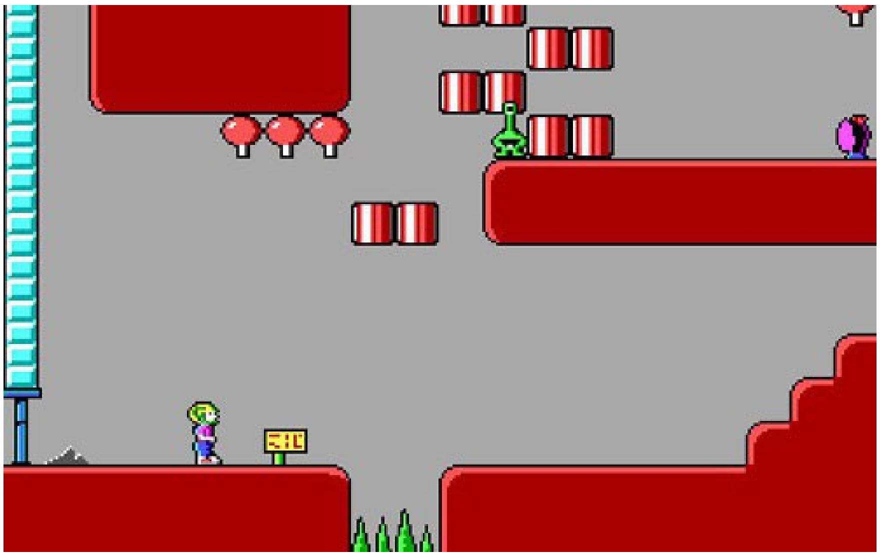


Figure 5.1 - Commander Keen Screenshot

Some of the feedback such the suggestion to implement a timer for speed runs was good, but unfortunately not feasible. Since the in-game character's speed does not vary, there would always be a set fastest time in which the player could complete a level, and once everyone starts achieving the same times, it makes competing in this area pointless.

Other feedback was taken into consideration though, such as comments on the difficulty of the latter stages, and adjustments were made to the levels to make them easier to complete. The comments regarding the gems being to hard to find was also taken onboard and now in the hub world for the game, NPC's will give hints as how to find the gems aren't easily visible.

Before the beta test started, a quick testing spec was done up listing all the browsers and operating systems we will aim to support. As has been stated before, both Windows and Mac will be supported, but due to a lack of a Linux machine test on, Linux won't be supported. The latest versions of all the major browsers, Firefox, Chrome and Safari will be supported too. Internet Explorer will be supported from version 9 onwards seeing as 8 and below don't support html5 canvas.

During the beta test any bugs found were logged in a table to keep track of everything that needed fixed.

Bug Number	**Browser & OS**	**Priority**	**Description**
#1	Firefox, Mac	High	Co-op button on main menu starts single player game.
#2	IE9, Windows	High	Game does not display. Console Errors: 'Howler is undefined' 'Object doesn't support property or method 'setV'
#3	Desktop App, Windows	High	After entering the first level the player instantly and repeatedly dies
#4	Desktop App, Windows	Medium	Fullscreen mode button does nothing when clicked
#5	Chrome, Windows	High	Two area's in zone 4 that display the wrong number of gems available in the level.
#6	Chrome, Windows	High	The big door in zone 4 brings to the start of zone 4 rather than the next zone

The testing phase here proved to be very beneficial seeing as it brought up a number of game breaking issues. The biggest issue of them all was #2, the IE9 one which appears to be an issue with the latest version of MelonJS. Rolling back a version was a not option here because the differences between 1.0 (version being used presently) and 0.9 effect how the code for the game is actually written. Thus reverting to 0.9 would cause a huge rewrite. Instead we can only make users aware of the issue, offer for them to download the desktop version and wait till the issue is fixed and update the game.

Fortunately the rest of the bugs were very easy to fix and didn't take more than a few minutes to find and retest. But had they been much more time-consuming, we were confident there was enough time left to handle anything. Instead the remaining time was spent reviewing code, making sure it was well commented and efficient.

6. Evaluation

Overall the outcome of the project was really successful, and it managed to meet most of the aims set out at the start of the project.

Probably the most successful element was the artwork with plenty of positive comments coming from the testing phase. Looking back more paper prototyping could have been done in order quickly churn out and explore different ideas, but in the end it didn't affect the overall results too badly. Probably the strongest area of the artwork were colour palettes used, with each one providing a unique and vibrant backdrop for the game. The character design was relatively successful as well, but during the redesign the uniqueness of the main character was lost a bit. Unfortunately this was an area which just lacked original ideas and is one that further help and advice could be looked for in the future.

In retrospective the methodology chosen at the start (waterfall) wasn't right for the project, but making the switch to agile definitely was a step in the right direction. This mainly due to constantly changing ideas and feedback that went into the project, which is something that wasn't predicated at the start. Developing a game was a pretty new experience and that probably somewhat accounted for not seeing this coming. It could also be argued that new ideas shouldn't have been coming in and it all should have been finalised during the planning stage. All of this will go down as learning experience and will be taken on board when next choosing a methodology to work with.

Time management was a big issue during the project, and the reason many features ended up being excluded from the project was down to a lack of time. This can probably be put down to inexperience in developing games. It was unforeseen just how much time would go into writing the actual game logic for the game, with all of the entities files alone consisting of around 2500 lines of javascript. This an element of game design that you can't really get around, but in the future the time expected to finish a project should be more accurate and number of features can be more realistically planned out.

The research phase itself went okay, but one consideration that was left out was how certain genres' of games work better on certain platforms. At no point did we consider that the most popular games on touch devices have mechanics that are based around the touch screen. Popular games such as Angry Birds, Doodle Jump or Fruit Ninja all take advantage of the accelerometer or touchscreen for swipe and touch controls. In the case of this project, traditional gamepad controls don't always work well on a touchscreen simply because there's no physical feedback and they're not as responsive. Fortunately in the end, this didn't matter so much because the project moved to a web and desktop experience where users could make use of a keyboard or a gamepad, but in the future it is a consideration that should be taken more seriously.

During development work, MelonJS proved to be an incredibly useful asset to the project. The structure it provided really helped to keep all of the code that was being written organised and easy to look through. With 13 separate javascript files and thousands of lines of code to manage, keeping all of this structured was a vital in making the code easy to write, as well fix.

MelonJS also provided exactly the features we needed for developing this game. As stated before, programming physics is an incredibly complex task and would've been extremely time-consuming to complete. The same can be said level editors to, which are essential in game development for providing a GUI that allows developers to quickly and easily create and edit level files. Without MelonJS's integration with Tiled, a custom level editor would have had to have been built, which again would be incredibly time-consuming. All in all, this MelonJS removed some of the painful elements to creating a game and allowed us to concentrate on programming the actual game logic.

One difficulty the framework did provide though, was the lack of online resources. With only one tutorial available, it meant a lot of learning had to be done through experimentation with the code, which is much more time-consuming and frustrating than simply being taught how to do something. Also, whenever there was a problem related to the framework, it was very rare you would be able to find someone online who had already had the same problem and solved it. Luckily though, the members of MelonJS are very active on their own forums and were great at providing support for those who needed it.

At one point during the development process, it was found that when MelonJS bound a key to a function, that key's default actions were disabled. So by bounding the letter x to jumping, users couldn't type x into the feedback form in the alpha test. After querying the developers about this, they provided us with an early release of the next version of MelonJS were this issue had been solved. Things like this would be very rare when working with larger javascript frameworks, where the developers don't have time to look through the hundreds of queries they would get everyday, and it was something that made MelonJS great to work with.

The switch from a mobile app to a desktop app was definitely the biggest setback of the whole project, and certainly caused a number of unexpected changes to be made. The problem arose when it was discovered that MelonJS would not scale the velocities for it's physics properly when the game dropped below 60fps. The result of which was a mobile app where the main character would move twice as slow and jump half as high, making the gameplay unplayable.

The only options at this point were to either change to a different framework, or abandon the mobile app. Since so much development work had already been done, switching away from MelonJS wasn't a realistic possibility. At this point, the contingency plan was to fall back on the web release, and explore the option of a desktop app. The contingency plan put in place at the start of the project was important here, seeing as otherwise, we could have focused solely on building a mobile app and had nothing to fall back on.

Looking back, the issues with releasing a mobile app could've been spotted earlier by doing some testing on a much smaller scale. The reason the issue took so long to discover was because development only started after the designs were in good place. This was an issue with the waterfall methodology in place at the time, which should have included an earlier feasibility testing stage. During that time, a very small and quickly put together demo might have brought this issue to light earlier on.

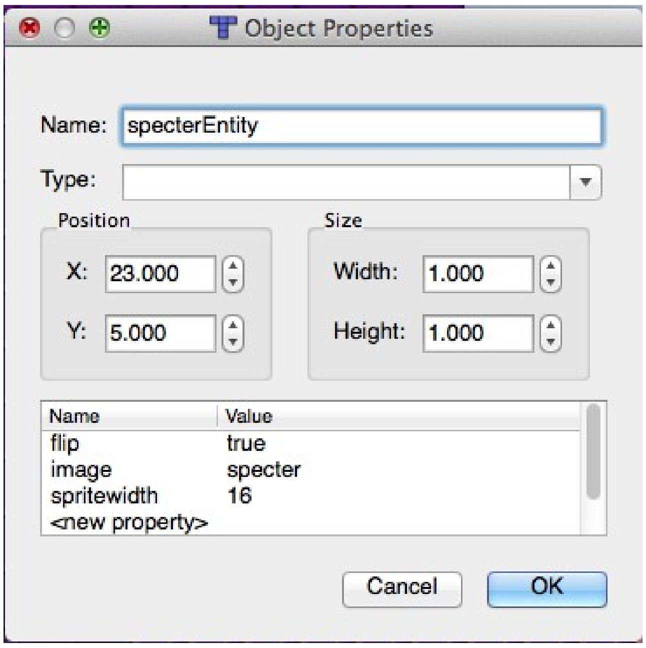
However, the switch to creating a desktop app was fortunately very painless and straight forward. Only minor adjustments were made to the actual game, such as using a larger resolution, and adding longer levels with checkpoints. Node-webkit, the app runtime used to deliver the game as a desktop app was incredibly easy to use, and only took a day or two to setup.

Over the course of project Github proved to be a very important asset. In the last few months of the project a hard drive failure meant that original code had been lost. But after a new hard drive was installed it was very easy to simply pull the last commit and pickup work from almost exactly the same place. If the project hadn't been backed up at all, it's questionable whether or not enough time would have been left to re-build everything from scratch.

One aim of the project that was never reached, was to add some sort of online aspect in, such as multiplayer or social interactions. Looking back, this idea was driven by the desire to make use of an exciting new technology, which was web-sockets. Web-sockets can be used to send data back forth between client and server very quickly in real time, and are perfect for creating online multiplayer experiences. During the final design workshop, it was mentioned that users generally don't care about the technology behind what your building, but instead are more interested in what the app can do for them. In the end an online aspect was never added to the game, because there was no ideas put forward that required an online interaction. Considering the advice from the design workshop, it was probably for the best that we didn't try to shoehorn an idea in, just for the sake of using web-sockets.

A suggestion that was put forward during a mentoring session was to include a level creator, where users could come in and create their own levels that they could then share with others. This would have greatly increased the longevity of the game. Currently when a player finishes the game it simply ends and unless they want to replay it, which is unlikely, they stop playing the game and move on. A level creator though would allow players to access a potentially huge database of user generated content. The idea was feasible to, because Tiled, the level editor used for this project, is free to download for anyone use, and is very easy to understand.

Unfortunately this idea came at later stage in the project, and because it would have been a sizeable task, not nearly enough time was left to implement it. First off all, a lot of documentation on how to create a level would have had to be written and made available. Every object in game has multiple parameters with multiple settings, all which would have needed to be documented.



Name: specterEntity
Type:
Position
X: 23.000
Y: 5.000
Size
Width: 1.000
Height: 1.000

Name	Value
flip	true
image	specter
spritewidth	16
<new property>	

A resource pack would also need to be put together containing the different assets from the game, such as background images and sprite sheets for each level. Then for the game itself, players would need a way to upload new levels, save them, organise and play them. All of which was entirely possible, but again would have taken a substantial amount of the time to test and develop.

Then finally, players would need a way to share the levels they created, which would involve setting up a backend and database to store entries. Currently the main site for the game is a static site with no back-end, so starting from scratch on one would again be very time-consuming. Then we also have to consider how to moderate each entry. If a user uploads a blank or broken level, there's no way to known until someone tries to play the level. The quality of each level created could also vary greatly. This would mean users would need a way to report broken levels, as well rate them, to let other users know which levels are worth playing. Again, the work involved in coding and testing these features could take up to two months, without other work going on in the background.

Once the game is launched however, the plan is not to stop development. Afterwards many bugs that weren't picked up in testing might be reported, and these will need to be worked on. Features that were not possible for launch will also be considered and possibly worked on.

Conclusion

The aim at the start of this project was to create an enjoyable game for a broad audience. From a few people's initial reaction this appears to have been achieved with both an appealing art style and enjoyable game mechanics, but reactions will be truly tested once the final release reaches the general public. With testing done across multiple browsers and operating systems we believe the game can work well on the majority of systems, and will allow it to reach a potentially large audience..

Although some features ended up being left out of the final product we believe what is currently available will be more than enough to satisfy users and in the future the game has potential to still see these features brought to fruition which will greatly increase the longevity of the project.

Overall there was a lot of challenging aspects to developing a game that were not obvious before the project started, but they have been great learning experiences and all the lessons learnt will definitely be put to use in the future.

References

1. Color Theory for Designers, Part 1: The Meaning of Color | Smashing Magazine. 2014. *Color Theory for Designers, Part 1: The Meaning of Color | Smashing Magazine*. [ONLINE] Available at: http://www.smashingmagazine.com/2010/01/28/color-theory-for-designers-part-1-the-meaning-of-color/. [Accessed 13 April 2014].
2. Bug 40881 – Authors require control over image interpolation methods. 2014. *Bug 40881 – Authors require control over image interpolation methods*. [ONLINE] Available at: https://bugs.webkit.org/show_bug.cgi?id=40881. [Accessed 14 April 2014].
3. Face Off: Are silent protagonists superior? | PC Gamer . 2014. *Face Off: Are silent protagonists superior? | PC Gamer* . [ONLINE] Available at: http://www.pcgamer.com/uk/2013/04/23/face-off-silent-protagonists/. [Accessed 21 February 2014]
4. iPhone users spend 14.7 hours a month playing games | Ars Technica. 2014. *iPhone users spend 14.7 hours a month playing games | Ars Technica*. [ONLINE] Available at: http://arstechnica.com/apple/2011/07/iphone-users-spend-147-hours-a-month-playing-games/. [Accessed 14 April 2014].
5. The 22 rules of storytelling, according to Pixar. 2014. *The 22 rules of storytelling, according to Pixar*. [ONLINE] Available at: http://io9.com/5916970/the-22-rules-of-storytelling-according-to-pixar. [Accessed 17 April 2014].