Project Documentation

Heart Of Demaxius

Samuel Crispin

Contents

[Introduction 1](#_Toc519169989)

[Proposal 2](#_Toc519169990)

[Specifications 3](#_Toc519169991)

[Design Methodology 4](#_Toc519169992)

[Design 4](#_Toc519169993)

[Implementation 5](#_Toc519169994)

[Testing 7](#_Toc519169997)

[Refactoring 9](#_Toc519169998)

[Screen Shots 10](#_Toc519169999)

[Conclusion 12](#_Toc519170000)

# Introduction

My project is a game based off of another game called *FTL: Faster Than Light* developed by Subset Games. It is best described as a fantasy roguelike game with a rudimentary point and click combat system.

There are a few reasons that I came to this decision. I have always wanted to make a full blown functioning game, I have gotten far many times before in spare time but I have never followed through to the end and completed it, so that’s where the game aspect came from. As for the basis of the game, the roguelike genre has always been a personal favourite of mine, with *FTL* being one of my all-time favourites. The roguelike genre is best defined as a subgenre of the RPG (role-playing game) identified by dungeon crawling through procedurally generated levels, turn-based combat/gameplay and the most iconic characteristic, permanent death for the player. When a player dies in a roguelike game, they lose all progress and have to start again making the difficulty curve generally quite high and the replayability of a good roguelike incredible.

As for the fantasy aspect, I have always been a sucker for fantasy, whether it be in books, films, TV, games etc. SciFi was a close second as a genre but seeing as *FTL* is already based around space travel and combat, I thought it best not to plagiarise it too much. I would have liked to have made a more impressive combat engine but with time restrictions and my capabilities at the start of this project, a basic point and click was all I could come up with.

Overall, the project went fairly well. There was quite a bit of deviation from the original proposal for multiple reasons which will be talked about later.

# Proposal

Overview

The general concept of my project will be a text based (maybe slightly graphical) rpg. You play as a person fighting randomized enemies getting progressively harder until an eventual boss fight where the player will have the option to end the game after defeating the boss or carry on until they are eventually defeated.

Story

You play as the grandson of a disgraced warlord by the name of Evandar Hrothgar, Evandar’s grandfather fled the scene of a great battle between the Kingdom of Demaxius and the Dark Elves of the south beyond the Dead Plains (a large Death Valley like dessert). The war lasted years but the Elves were eventually repelled hence why the Demaxius still stands.

Your family has since had no place as descendants of a deserter within the new rebuilt Kingdom and you strive to find a way to recover honour and pride back to your family. You hear tales of an ancient, precious and powerful relic that was lost during the Great War, supposedly stolen by the Elves over its rumoured power.

You plan to retrieve the relic and restore some honour to your name. The story starts as you have just made the treacherous and unlikely journey across the Dead Plains and have arrived at the Elven town of Elaris. You have very little on your person, just a few gold coins left over from the time spent in Demaxius, now worth little here due to the abundance of gold mines south of the Plains, The Elves favour a bronze based currency due to it being more expensive. You also have your trusty oak staff and your mastercrafted broadsword passed down from your grandfather.

As much as there is a ‘peace’ between humans and Dark Elves, there is still tension between the two races so keeping you identity hidden could prove beneficial.

You control Evandar around the town and eventually into multiple caves/dungeons/ruins in search for the fabled relic. You eventually find under the guard of a great cave troll, a species long thought extinct, instead they live deep in caves feeding on travellers stupid or brave enough to go deep enough into their domain. Cave trolls have adapted to being able to live weeks and even months without a meal due to their very lazy life of lying in a cave guarding their treasure, hence being thought extinct as they are never seen. You fight the troll and, depending on the result of the fight, retrieve the relic and escape or die to the troll and restart the game.

Gameplay

The gameplay itself is inspired by a few different sources, the main one being a game by the name of *FTL: Faster Than Light.* A space travel game where interactions are completely randomized (dependant on your location within the game, previous interactions and equipment and crew etc.). Interactions between you and the inhabitants of the Land of the Elves will be randomized like interactions in *FTL* as well as having different options depending on your progress, characters stats, inventory etc. Outcomes of interactions will be mostly random with some having fixed outcomes. The game will also try to follow the difficulty of *FTL* in which death means restarting the game.

When defeating an enemy, it will drop loot. The looting system will be inspired by games like the *Diablo* or *Borderlands* series in which every enemy will drop some form of loot. This loot can be as little as some currency all the way up to legendary weapons or accessories. Like in the game series mentioned earlier, the weapons and the accessories will be graded in rarity by colour: white (common), green (uncommon), blue (rare), purple (epic) and orange (legendary). Tougher enemies are more likely to drop higher tier loot but any loot can come from any enemy.

The game map will be randomly generated, a 20x30 grid will be used to represent where points on the map can be with a random number of points (between 15 and 20) generated and placed on the map. When clicked on, you will be taken to a location with an encounter. The encounter will be chosen randomly from a pool of set interactions, this encounter will then be taken out the pool so it’s not repeated in another location. One location will always be a town which will include different shops, such as a merchant (low level gear as well as food and other supplies), smithy (higher level gear), an inn (stay the night to restore health) and an arcane merchant (potions and occasionally magical gear with special effects). The town will have 3 different layouts, one of which will be chosen randomly so it won’t be as repetitive (obviously it won’t change throughout a single game).

Enemies stats will be randomly generated within certain bounds (depending on how far in the player is through the story and potentially the difficulty chosen, I am yet to decide whether I modify player stats and loot or enemies stats and loot depending on difficulty), these stats will be used to generate a score for the player defeating the enemy which will be totalled into a leaderboard.

The boss fight location will be chosen randomly but the player won’t be able to access the fight until the player has cleared every other location (I’m not sure exactly how I’m going to implement this believably just yet, most likely along the lines of the character doesn’t feel ready for the fight yet or something). After the player has defeated the boss, they will get the message ending the story and then asked the question if they would like to continue to try and beat a high score. The map will be regenerated and the player carries on with their current inventory.

# Specifications

As mentioned previously *FTL* was my main inspiration for this game and its mechanics. Because of this, all the navigation and all the combat is done by point and click (*FTL* also uses keyboard short cuts for navigation like being able the press the 1 key as well as clicking the first option which I would have implemented had I know of event handlers earlier) and the game is very menu based, as shown in the screenshots just before the conclusion.

Basic Feature Overview

* Menu based navigation via point and click
* Map populated with clickable points that include interactions
* One of the points being a town which includes 4 visitable shops
* Shops randomly generate items which the player can buy and equip
* Shops regenerate items after a point has been cleared
* One of the shops is an inn where you can spend the night and restore health
* Have an inventory menu where you can access different item types and equip them
* Separate screens for each item type
* Make tooltips appear with the items stats when you hover over them
* Combat system
  + Turn based, player will have a way of knowing it’s their turn
  + Will have 2 options: attack and eat
  + Attacking does damage to enemy
  + Eating something from your inventory heals you by a set amount
  + Damage will be randomly generated between 2 amounts depending on players attack and enemy’s defence
  + If players health is reduced to 0, player loses the game and has to restart
  + If enemy’s health is reduced to 0, the player wins the battle and then they have the option to collect loot
* Loot system
  + A random number of items (between 1 and 3, with 1 being more common than 2 and 2 more common than 3) is generated and the player can take as many of them as they want
  + The items stats will depend on how far through the game the player is, again randomly generated between 2 bounds
  + 9 types of item (listed in my implementation section under pseudocode so I won’t repeat it here)
  + Will generally be called after a victorious battle but can be called from an interaction function as well
  + Consumables won’t be generated as loot, just equipment

# Design Methodology

As for design methodologies, my project is a mix and match of a few different ones. At a basic level, it is bottom up, I created separate parts of the game and then later stuck them together to form a whole. Drilled down, it is a mixed bag of object oriented, functional and procedural programming. The object oriented side of things came into the project in the much later stages due to the fact that I didn’t really have a grasp of objects and how to use them at the start of this project, also aided by the fact that JavaScript isn’t really an objectified language like Java of C++. Although it came in later into the project, I did go back and refactor some old code to accommodate my new knowledge of objects and their interactions. Things like turning the players separate statistics into one player object along with methods related to the player, along with creating other functions to help reduce repeating code like creating a changeHealth function instead of repeatedly writing code to do so.

# Design

* The front end was all done in HTML, CSS and JavaScript
* Game logic was all written in JavaScript
* The few graphical assets I had were drawn myself
* Code design itself was done as pseudo code in my notebook, examples will be shown later

As well as code design, a lot of other graphical design work was done in my notebook, things like designing how menus, GUI’s and interactions/battles look for the user. Even though there was a decent amount of rewriting early code, a lot of it has stood up and is still being used in my project now.

I use pseudocode as my first and only step of design and planning, this is due to the fact that my mind is already thinking about the next step of my project whilst finishing the implementation of the previous step or during lunch etc. and buy the time I get round to it, I already have a very good idea of what and how I’m going to implement it. I use my pseudocode to run through my process a few times and try and iron out any kinks or sport any mistakes in my thought process and also as a reference for if I go to work on something else or forget something. My pseudocode also tends to be closer to code than pseudocode normally is.

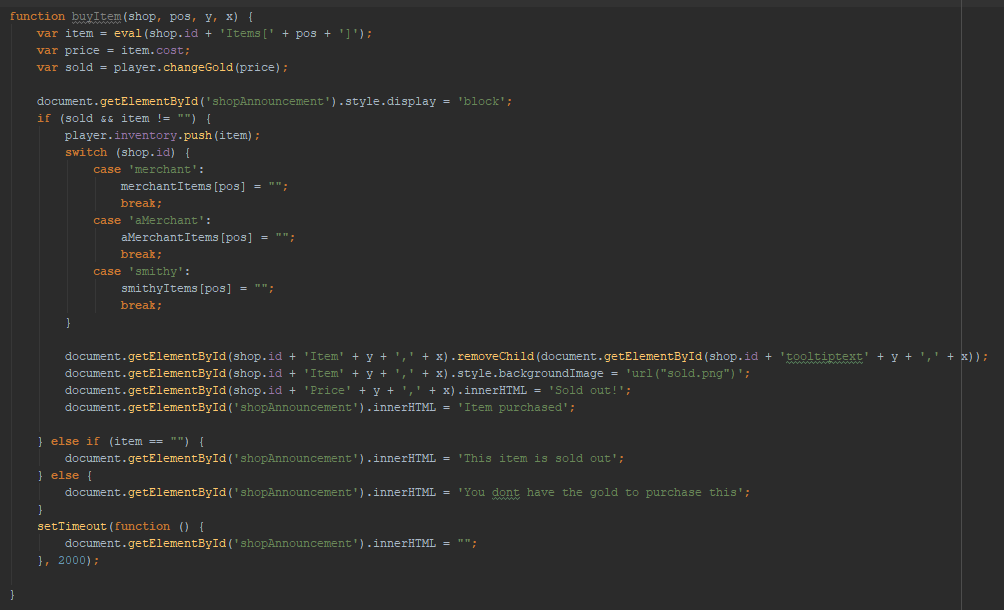
# Implementation

Here I will show a copy of both pseudocode from my notebook and the actual implementation of the same function:

## buyItem()

Pseudocode

buyItem  
if (enough gold)(shopItems[pos] != “empty”)  
 shopItems[pos].inShop = false  
 player.inv.push(shopItems[pos])  
 pl.gold -= shopItems[pos].cost  
 doc.getId(shopItem…).bgimage = none  
 .innerHTML = “sold”  
 (shopPrice).innerHTML = “Sold out”  
 announcement.inner = “Purchased”  
  
elif (shopItem[pos] == “empty”)  
 announcement.inner = “This item is sold out”  
else   
 announcement.inner = “You don’t have the gold to purchase this”

Actual Implementation

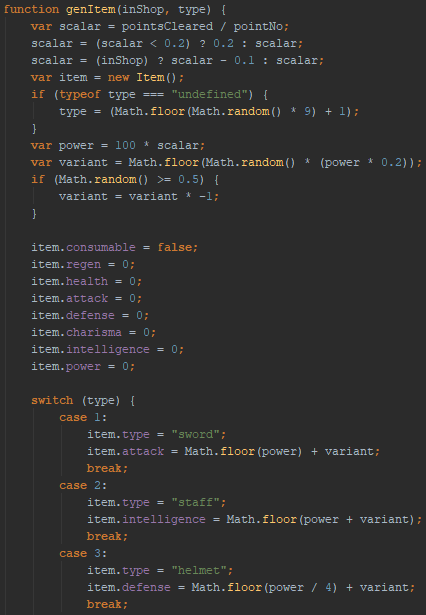
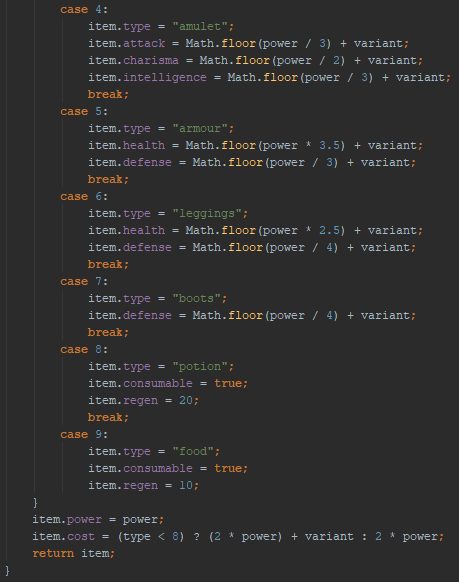
While there are some clear deviations from my pseudocode, these were mainly to do with the later implementation of objects into my code, the other one being using a case statement to wipe the item from the shop. This was because I wasn’t planning to have the items and shop arrays set out like they are now originally.

## genItem()

Pseudocode

genItem()  
pointNo – global  
pointsCleared – global  
var percentCleared = (cleaed/pointNo)  
var item = new Item()  
var type = Math.floor(Math.random()\*8)+1  
switch (type)  
1  
-  
9  
  
1 type – sword  
 consumable – false  
 instantHealth – 0  
 health – 0  
 attack – rand 3 – 5  
 defence – 0  
 charisma – 0  
 intelligence – 0  
 abilities – “”

Item types  
sword – 1  
staff – 2  
helmet – 3   
amulet – 4  
armour – 5  
leggings – 6  
boots – 7  
potion – 8  
food – 9

Actual Implementation

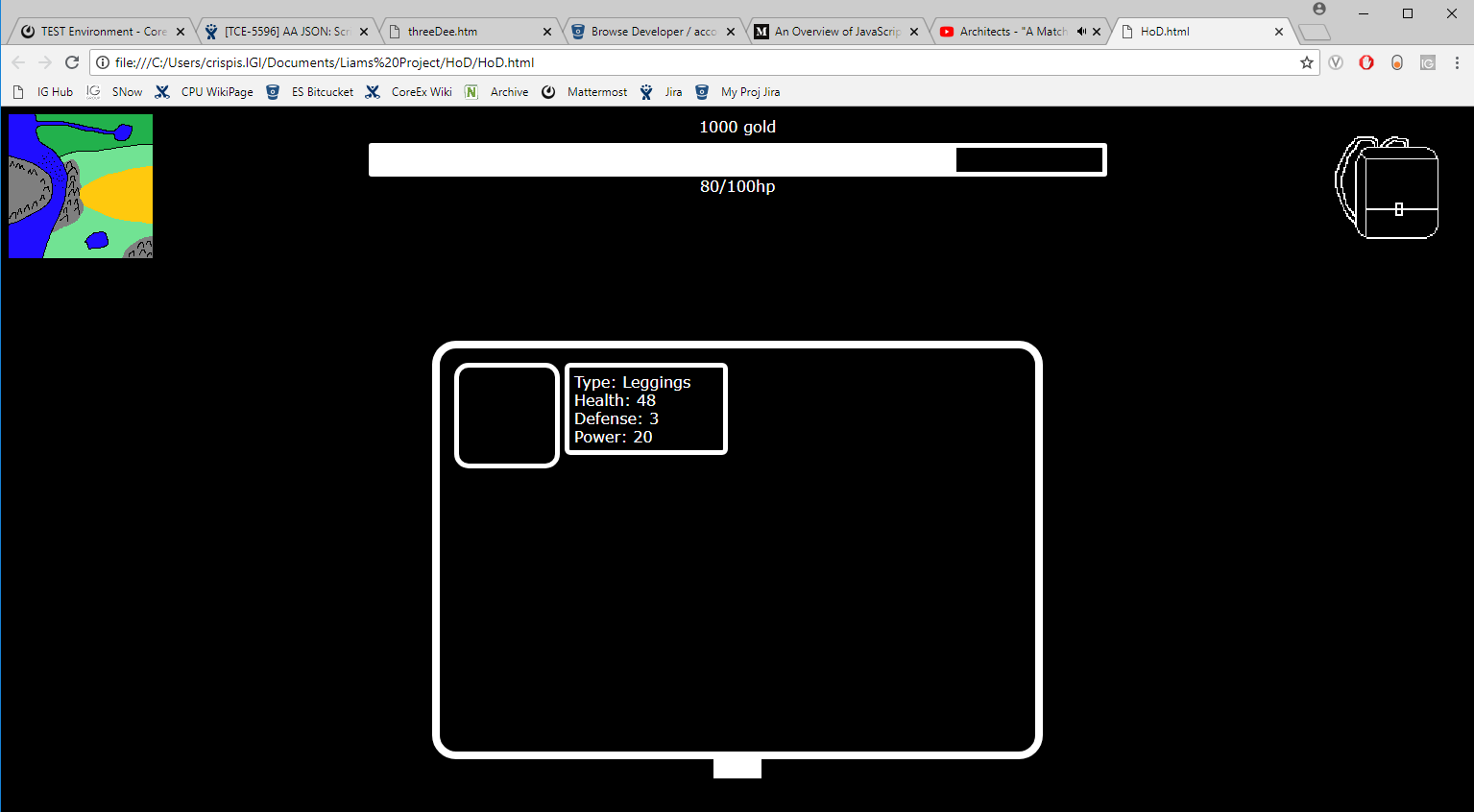
Like in the previous example, there is a good amount of variation from the original plan to the actual final implementation.

I soon realised that I could initialise the objects fields as 0 and save a lot of repetitive code regarding setting fields to 0. This reduced the size of my case statement dramatically and much more readable. Another change that I made was to be able to choose what item I would like generated if I have the need to, the type argument is read, if its undefined then I choose I random type, else I use the type defined in the function call. I was originally going to have a separate function for a hard coded type but with this argument I don’t have the need to anymore.

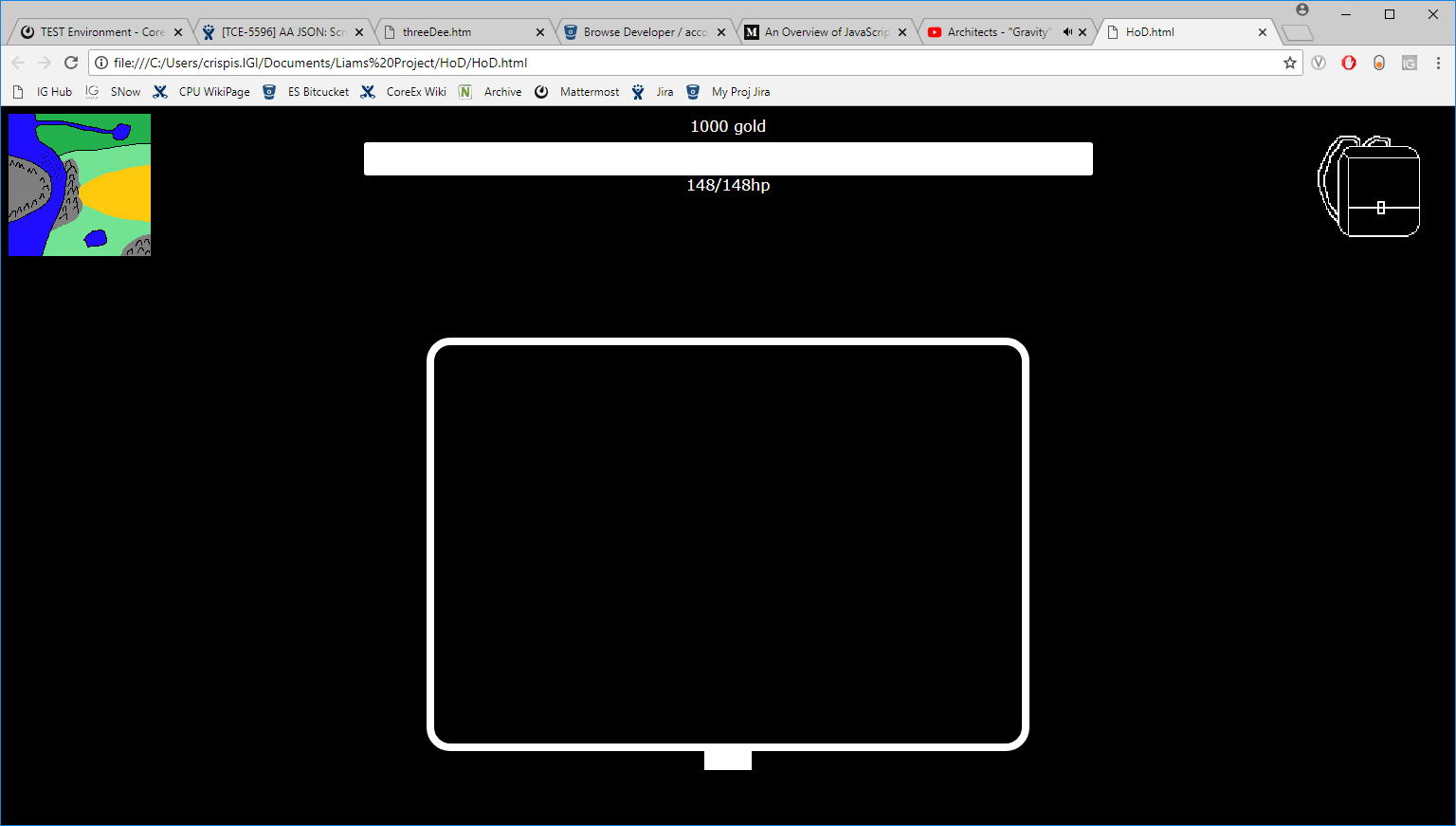
Likely the biggest change to this function was the implementation of a variant. This variable allows a variation in the cost and power of equipment generated, rather than all being the same like they previously were. This variant takes into account the amount of points cleared so far and weather the item is in the shop or not, I also gave it a minimum value of 0.2 as to not make values 0 or too close to it.

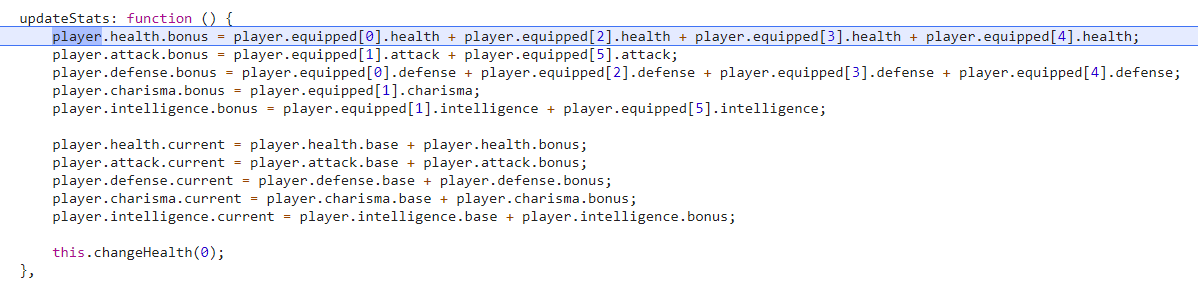
# Testing

The best way to test a game is play testing, I played through my game as much as is possible with this current version to try and find any bugs or features not working as intended.

The first bug I encountered was to do with equipping loot that I obtained after a battle:

Here you can see that I have lost health after being damaged in a battle and I have obtained some loot from the battle. There is one immediate problem, the div for the loot item doesn’t have a background picture like it is supposed to. The next problem arises when I try and equip the item:



The tool tip for the item says that it will give the player 48 health, meaning my maximum health should go up to 148 and my current health should increase b 48 as well. Clearly that hasn’t happened. My maximum health has increased by 48 correctly but my health also restored to full which wasn’t supposed to happen.

The problem for the health was in my update stats function. In the second section, I’m calculating the players current health by adding their base health (100) and their bonus health (in this case 48) meaning their health goes to full whenever this function is called.

To fix this bug, I removed the second half of the function and now when something is equipped and I call changeHealth and increase the player’s health by the equipped items health just before I call updateStats and now equipping items works as intended.

As for the other error to do with the background images, I was getting this error in the console for each icon that was broken, with a different name for different icons obviously. 

I only had to make a slight change to fix this: adding ‘img/’ to the start of the url of the image, I changed the folder and put images into a subfolder to clean up the project and clearly didn’t change all instances of calling an image. Proof is below:

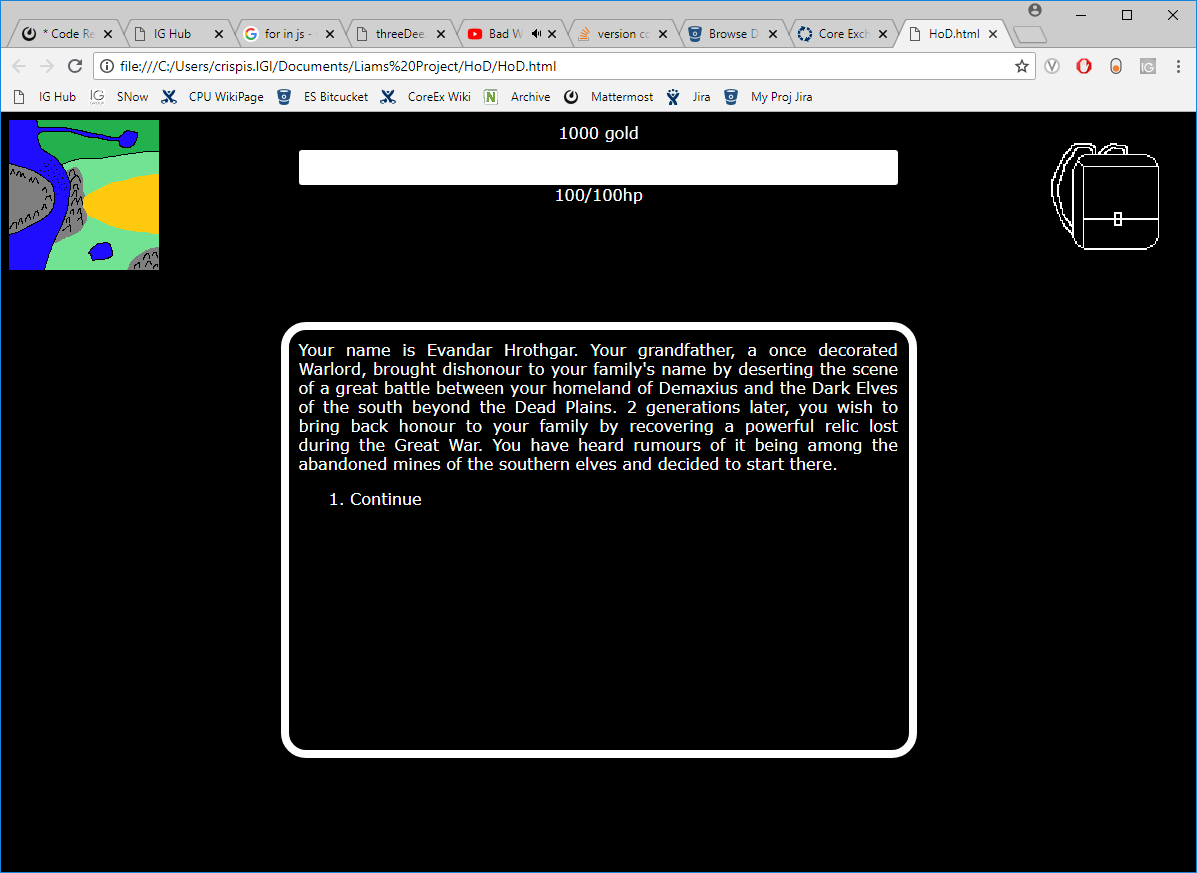
# Refactoring

As mentioned previously, there was a decent amount of refactoring during the lifetime of this project, the majority being to my interactions object. The original iteration of it was a massive switch statement, but this ended up being hard to work with and horribly ugly and hard to read. As I got further and further into my implementation, I came across more and more things that I had to find a workaround for and soon realised that this was poorly planned and needed to be reworked from the ground up.

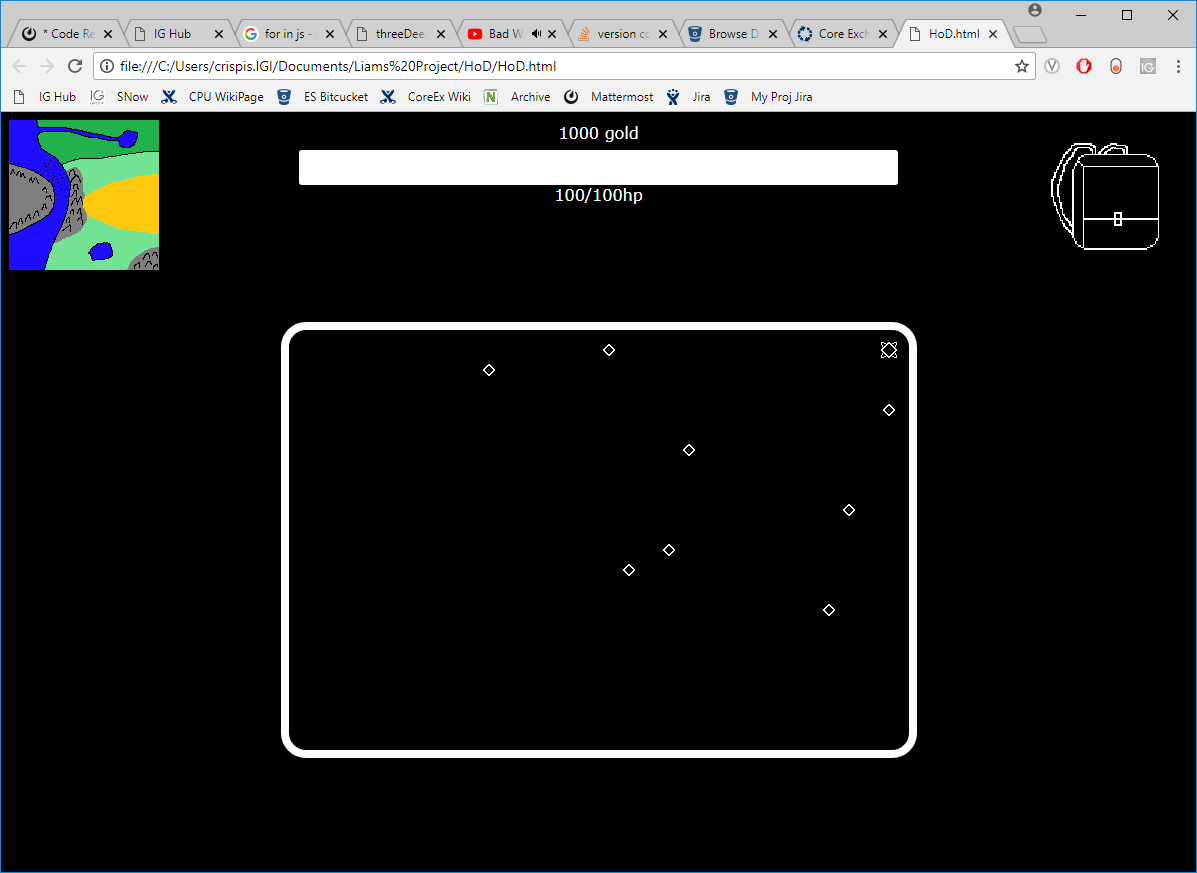
As per a suggestion, I looked into changing that big switch statement into an object and then use object lookup to decide what interaction to output next. There were many benefits of using object lookup rather than a big switch statement, normally the main one would be speed but seeing as this lookup only occurs when a certain thing is clicked (rather than something like a game loop happening 100 times a second), speed isn’t really effected. The main benefit of this was customisation and flexibility. Adding more options and adding new functionality to these options becomes so much easier now that I have handlers for function pointers and the ability to easily add new fields.

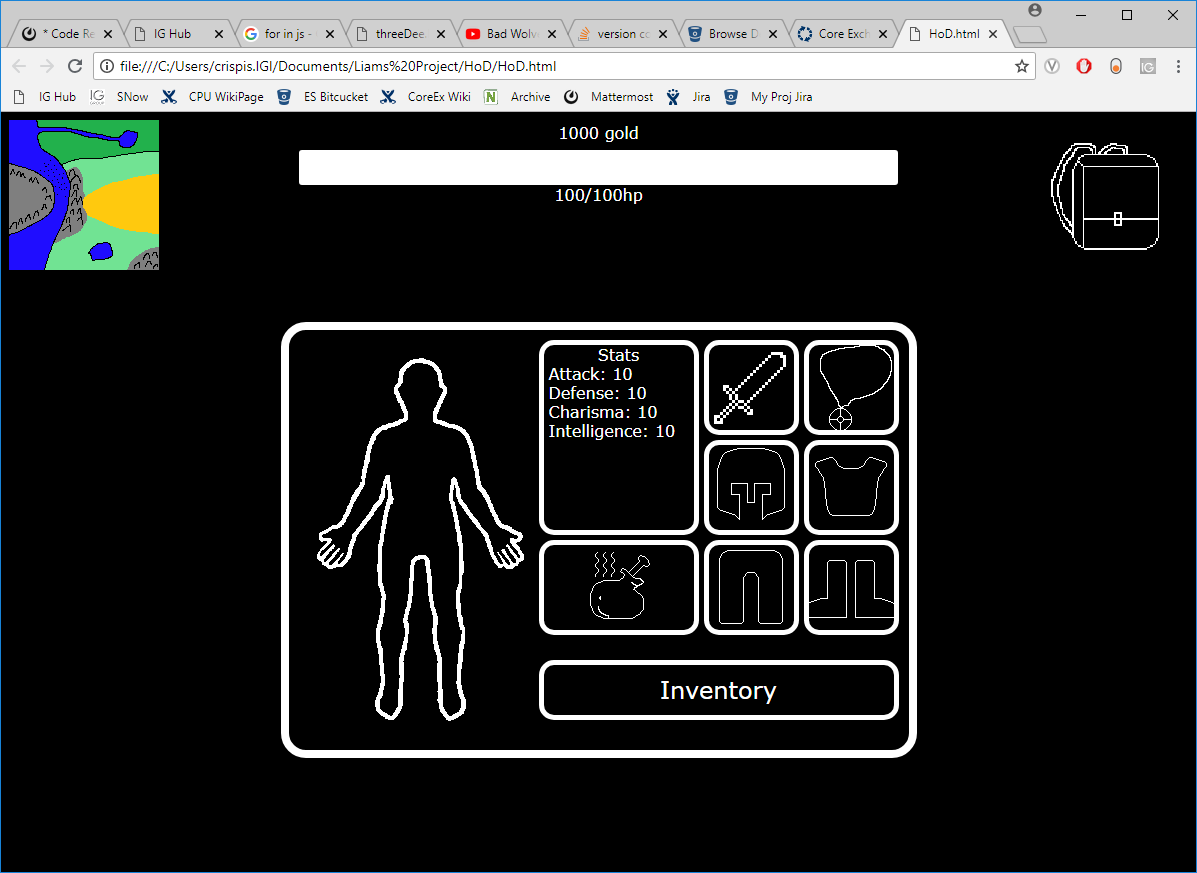
# Screen Shots

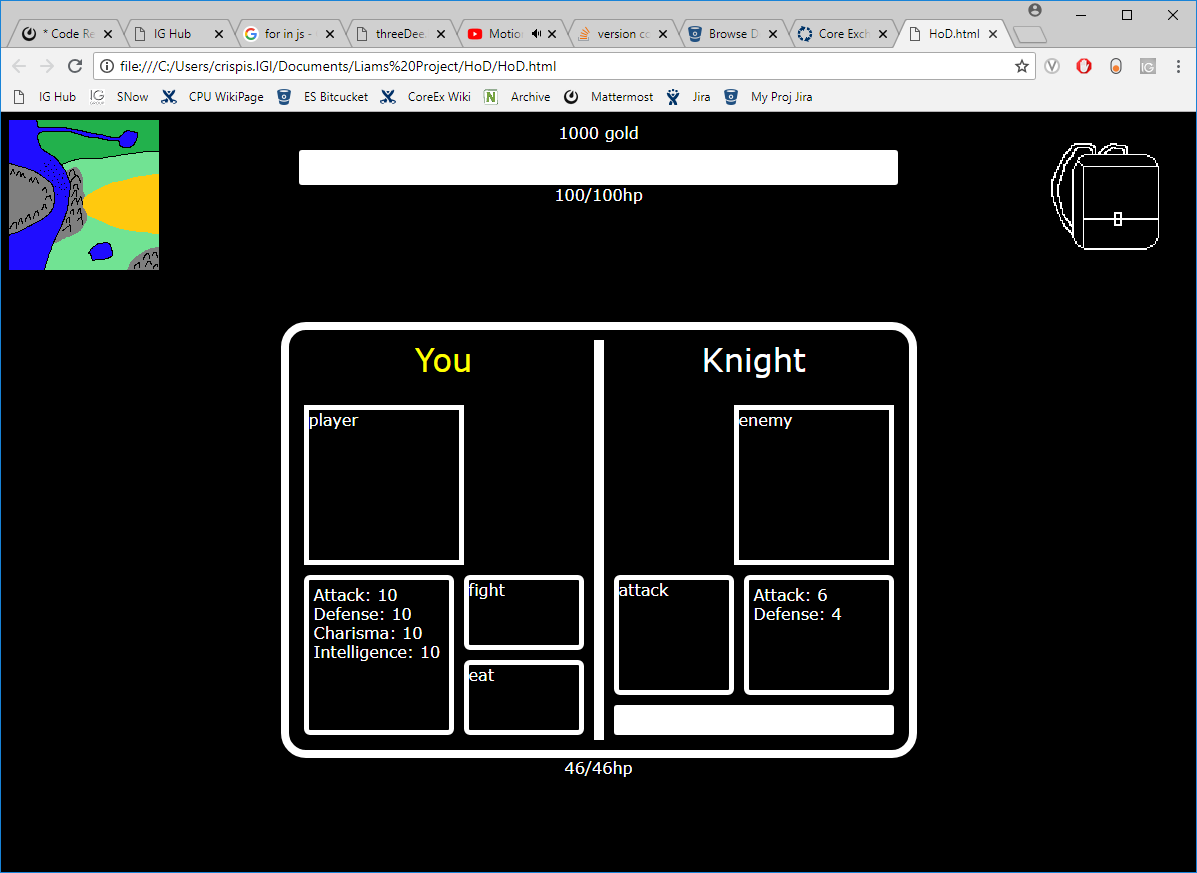
Opening Screen



Map Screen



Inventory Screen

Fight Screen

# Conclusion

In conclusion, I am relatively pleased with how my project went in total. Initially progress was pretty quick even if I did rewrite a lot of my early code pretty soon after. It helped also that I didn’t have any other work to do at the time so I could completely focus on this project. There was a few hiccups in the process, mainly the creation and then refactoring of the interactions object. There were about 3 or 4 different iterations of the object as per advice from my peers with the final version being something that I didn’t haven’t a complete understanding of which did present problems. I managed to get away with the understanding that I had and still progress with the project.

There were plenty of concepts and techniques that I learnt about during this project, the main one is probably the use of object lookup. This was used in tandem with the later versions of the previously mentioned interactions object. Again, with my interactions object, I used a few other things like event listeners and other bits related to it like using target and function pointers a lot of which I hadn’t touched before. There was a little bit of CSS that I hadn’t used before this project, the main bit being tooltips that appear when hovering over a certain div.

Although this project is closer to being finished than I expected at this point, it is ultimately unfinished. The two main bits that were unfinished were the infamous interactions object and the ending to the actual game itself.

The interactions object was unfinished in more than one way, it still needed to be populated completely, it only had about ¼ of the amount of interactions that I wanted. I got caught up too much in refactoring it and other projects I had going at the time to really knuckle down and think of a bunch of scenarios to put the player in that aren’t too farfetched and aren’t too boring. It also needed a final touch of refactoring as well for it to have the complete functionality that I wanted.

The plan for the ending of the game was going to be a boss fight of which afterwards you would collect the relic that the protagonist was searching for and then you would have the option to continue the game with a map that would just regenerate every time you cleared it until you lost.

There was also smaller finishing touches that weren’t finished, bits like there were still a lot of place holder divs for assets that I hadn’t obtained/drawn yet.

As for things I would have done differently, the biggest part is the combat engine, or lack thereof more accurately. Given the chance to rewrite my game without time constraints, creating a proper combat engine would most likely be the biggest change I would have made. It would require a lot of thought that I haven’t yet put into it but it would probably be something along the lines of an ability based system where you have up to 4 abilities that you’ve learnt or gain from equipment which have different effects (some damaging, some defensive/healing and some utility like increasing loot drop rate etc.), very similar to the combat of the Pokémon series.

There were also bits that didn’t even come close to making it into the implementation. The main one being a save and load game function that would use local storage and JSON to save the games state so you can carry on with a play through.