Design Process

When starting out to design my game I had already used assets for my game proposal, so I decided to keep using them. They were cute 8x8 tiles from Itch.io, a website used by many indie game developers to show off & share their games and assets. These were free assets from Pico8, a free virtual machine & game engine I had played games on before.

I knew I wanted to use the provided tilesets, images made up of many small images known as “tiles” which are very often used to build entire games’ levels since they make it both more efficient when designing and easier to manage in the long run. I wanted to use Tiled, a free map editor designed to use tilesets in any way you could possibly want, since you can simply “draw” your map out as opposed to individually placing every single tile.

I started out designing a testing map since I would need to make sure I knew how it all worked before I went about building levels, or bits of levels, as I had originally planned. This was relatively simple, import tilesets, embed them to the map file, create the tile layers I’d be using, and get placing tiles. I then exported this as a .JSON file which is the format that Phaser supports tilemaps in and made sure every file I used was in the assets folder.

Then got to importing the tilemaps, the sprites, background images, tilesets, etc.

Once this was all done, I used our previous Software Project labs to figure out the syntax on how to get the basics made; the controllable player drawn on the canvas, the enemies, and collision detection. I then referenced the Phaser documentation to figure out how to use my tilemap and add collision to the solid platforms layer of my map. I had also placed items around the tilemap on a different layer, and a background. I assumed if I assigned a property to the individual tiles in the tilesets I was using in Tiled, I would be able to easily access them in Phaser and work them into the game that way. But after many an hour going through the documentation and solutions online, I gave up. I quickly enough figured out how to make an enemy, how to make it follow me once I got close, how to have it kill me when it’s bounding box came into contact with the character, etc. But using the properties I had set in Tiled proved to be a lot more difficult than I had originally expected. I decided I would add each item in later as individual sprites, but it was hard to bring myself to do it after spending so long trying to figure out how to use the tilemap items, so I instead focused on a more worrying problem..

After playing around in my testing map with the enemy for a while, I noticed something strange. The slime enemies I had added would throw incorrect collision detection events when trying to see if I had landed on a slime’s head or not as I wished to kill them when this happened. I intended for it work like Mario, you touch an enemy’s side, and you get hurt, land on them they die. The problem was not only would they die but I would also die every time I landed on them. After scouring the internet for issues with Phaser’s Arcade physics engine, I found some posts that claimed the issues I was having here and also a strange bug I’d noticed when jumping against tiles above me (which would teleport me to the top of the tiles) were due to small sprites. I thought this very strange since surely small assets are quite common, but since I could not figure out how else to fix it.

Originally when I mentioned this to our lecturer, he suggested I scale up everything within the program. This would have been a far quicker solution but scaling up the map proved to be a nightmare, so I gave up. I used Aseprite to scale up every asset in my game by a factor of 2. Problem was, this forced me to recreate my tilemap, and I was already behind schedule with the game. I spent half an hour designing a new testing map with all the things I wanted to test for, using my new 16x16 modified Pico8 tilesheets. During my time in Aseprite I also recoloured the tiles to make them more "cave-like" and duplicated the main tiles file and made very dark versions of them to use as my background.

In this testing map, the clipping glitch that caused me to phase through a block I was underneath / to the side of stopped. This was good. However, the slime glitch persisted. In the end I used multiple workarounds to make it work, specifying that the player must be touching downwards while the slime touches upwards while the player is also above the slime by less than a pixel. Mostly this was working but the odd time if I landed on the left or right edge of the bounding box, I would die once I got to the floor. Much later in the design process I realized it was due to me having an overlap collider *as well as* a normal collider.

I decided to leave it where it was, as I was wasting far too much time on it. I knew I wanted to implement procedural map generation from pre-made tilemaps in the form of "building blocks" and since I had done this in Unity before I thought it would not be too bad. Sadly, I was wrong. It was already proving to be very difficult to use Tiled tilemaps in the ways I thought would be easy, and after a little research it turned out it would be hellish to implement in the 2-ish weeks I now had left to work on my game. So, I scrapped the randomized map concept and got to building an actual level.

After building the level, I realized I'd now need to manually place the enemies, since there was no easy way I could find to access the tile properties set in Tiled. If there had been, I'd planned to make a function which randomly placed enemies in all the available spots, but without this ability I was forced to either have enemies spawn inside solid walls or place them manually. I got to making a function to allow me to show my X & Y coordinates on screen when I pressed a button. I set this to F1. The reason I didn't use console.log() readouts is because they endlessly proved to be **extremely** resource heavy. Often making Firefox grind to a halt and not obey refresh commands until it had got to the end of its long list of logging tasks.

Once I had this built, I walked around the map, jotting down the X/Y coordinates onto a piece of paper at every position I wanted a slime to be added. This took about an hour or two just because of how manual it was, especially considering I had to go and type out each of the 58 slimes I had written down into the program too. At this point I realized I may be better off making a class to build me the slimes, since I would be handling a lot of things at once and did not want large chunks of duplicated code. I had tried making groups of slimes but manually positioning them turned out to be a pain using that method, so I went with building a class.

After I had all my slimes placed and the class somewhat working, I realized all I had was a difficult platformer with a boring cave, 58 slimes, and no end. I had wasted so many hours

doing things that led to nothing, so I had to figure out an alternative to my original game concept. I had kept trying to come up with ways of sticking to my original idea but so much of it would now have to be scrapped, what with the lack of randomized levels, and my difficulty with the (seemingly) most trivial of tasks.

I moved on to adding treasure around the map, since this would at least give the player a goal. This was relatively easy because I decided not to manually place them. I used a random value to pick the x & y coordinates relative to the level size, and then made a for loop to generate 100 gems to the map. Luckily for me the gems that spawned in walls would drop to the floor due to the collision properties, so other than one or two getting stuck in walls it was perfect, and added a little bit of variation between runs of the game.

I implemented a scoring system based on how many gems had been collected and got to figuring out how to end the game. There was about a week left to finish making the game by this stage, so I decided to just focus on the bare minimum for now. I made a fully black sprite in Aseprite, the width of two tiles, and placed it at the bottom of the map where I'd left a gap for such an exit. I then added collision properties to it and told it to run a new function similar to the gameOver() function I called when I died, except it was gameWin().

So now I had a bad Mario-like game. Collect gems, squash slimes, get to the end.

I now needed extra scenes, to fill the brief requirements. A settings screen, a help screen, and a high score scene. I also needed sound effects and music, and more background images to use in some of the scenes.

I got to designing the background for my game over scene and game win scene, and acquired some free SFX packs from *opengameart.org*, a website used for sharing open-source game assets, much like Itch.io but exclusively open-source. I also used *freesound.*org, a site for sharing samples. These were just simple 8-bit sound effects as used in thousands of games over the years. Simple square oscillator tones and blips. I opened Audacity and trimmed them so they would all start tight to the sound wave. I also amplified one sample that was far too quiet. I had to convert them all from .WAV to .OGG (Ogg Vorbis) & .MP3 respectively to reduce the file size. I used level 8 .OGG encoding and VBR V0 encoding on the MP3s since they would turn out tiny either way and this provided better quality.

Adding these to the game was relatively simple, and making them trigger when doing something was mostly fine too, at least after I figured out I needed to not only load them but add them to the scene too... Luckily for any sounds that were too loud, I was able to use the volume property to quieten them, and so I normalized the levels within the game.

Making the new scenes was not too bad, by this stage I was getting a bit more comfortable with Phaser, and it did not take me too long to get each scene built and connected to the others. I altered the button class given to us at the start to take more button types. This was because the class did not change the position of the text it used in its constructor. This made no sense to me because the button image underneath it would drop as if pressed, so the text would just float above it is clipping the borders. I bypassed this by modifying some

button assets I acquired from Itch.io in Aseprite, making 2-4 frames per button, and manually adding the text within the image so it followed the button as I pressed it.

Sadly, the button class was not intended to be used this way and so it was a little broken. I managed to get all the buttons working but when you hovered over them, it would push down the button but the “unpushed” button image was left behind the pushed image, so I had to go back and modify every second frame I’d made. Instead of the button pushing down I just added an outline.

Now that I had the scenes working and the SFX added, all I needed was to figure out how to pass variables around between scenes. I needed this to pass the controls scheme selected in the settings menu to the game scene, and to pass scores from the game scene to the death or win screens. This did not take too long to figure out. I just needed to pass the key “data” into the init() function of any scene I wanted to take data, and when switching scenes I just had to add the data after the scene I wished to start.

Finally came high scores and music. I had scores passing from scene to scene but to keep the highest score as a localStorage object as required by the brief, I had to do some research. It luckily turned out to be super easy. All I needed to do was use simple if statements to check each score against the last greatest score, and if it was greater, pass that score to the localStorage variable of whatever name I wanted. Later in the high score scene I just needed to retrieve that score.

Since I only had a few days left now and was nearly done, it was time for the finishing touches. I opened up Ableton Live, a digital audio workstation (DAW) used to make music. I keyed in a little melody like that of Mario’s caves 01 and used that as my bassline. I then added a kick drum, a lo-fi snare, and an accompanying melody to prevent the tune from getting too boring. It amazingly only took about 10 minutes to get this track made. Though it was extremely simple I must admit. I normalized the volumes and passed the render to Audacity where I converted the file to .OGG and .MP3 so it was not overly large. Upon testing my tune in the game, it turned out that it looped incorrectly. This was annoying since I had tested how it looped in Ableton and it was perfect. So, I had to go back into Audacity and manually add a few milliseconds of silence so Phaser could handle the loop better. This mostly worked, so I left the music there.

Testing

As you can see, I tested my game throughout the entire design process. Whenever I had made a considerable chunk of progress, I would play through the game again to see how it was. I used console logging a lot, and also in-game text objects for things that would otherwise need to be in the update() function. I had so many bugs that testing was constant. From time to time my friends would open the link to the live site and play through it. This gave me more info as well, since they were able to playtest more than I ever did. I learnt about slimes killing the player when the player was a tile above them and just at the edge, as well as how difficult certain sections of the map were. I modified the map a bit to prevent

such errors and squished the slimes a bit on the y-axis to stop them from killing the player unintentionally. I also used WebStorm (the application I switched to about 2 weeks into the project due to its IDE functionality) to debug constantly, which helped an incredible amount. WebStorm was also able to read the definitions file far better than VSCodium which I was using before.

When testing the SFX I came across a brutal error with the slimes and how I handled their deaths and collisions. The squish sound effect that played when killing a slime would either play endlessly, or when I landed on gems from above too. This is because the first slime I killed would be removed from the scene and physics engine, but its collider was left over. To fix this I studied the documentation and found I could use callback functions and a process callback when initializing the collider, allowing me to remove an entire function I had in the slime class and instead handle it on a slime-by-slime basis. Once the collision happened, the collider was removed. This would stop it from firing constantly and breaking everything else.

One bug I was not able to fix is very strange and required a lot of playtesting to even find. One slime and one gem per level turned out to be broken. You cannot collide with it at all. I have a feeling it is something to do with the forEach loop I used to apply colliders to each enemy in my array, but I am still unsure.

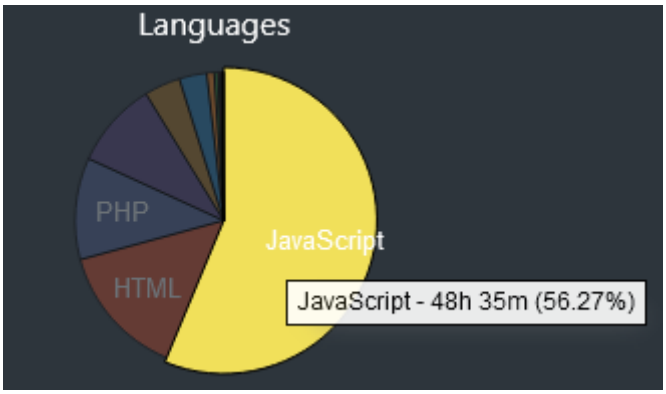
Time Management

During the development of Caves of Yurt, I stumbled into many a pothole, often losing motivation due to hours spent getting nowhere. At first, I was spending a few hours each day trying things out, but I ended up leaving it off and focusing on another CA with a closer deadline. It was only after I submitted this that I realized just how much work I had left to do on the game. I had installed some time tracking extensions such as CodeTime and WakaTime for VSCode but I forgot to get them for my JetBrains software until the final week. In the last two weeks of development I put in at least 50 hours of solid coding time into this project. I reckon it was closer to 60 or 70 though from the first two painful weeks.

As the second week started, I decided I needed to structure my weeks better. I was working into the night on the busiest college days and it was tiring me out a lot. I decided I must only work on my less busy days and for a set number of hours. This worked on the second week and stopped me from burning out, but during the second week I had to scrap this idea and put a solid 40 hours in, trying to bring my game up to a playable standard.

This project has taught me a lot about myself in terms of time management, mostly how bad I am at it. I am great at crunching down and writing a lot of code when I have to, but I wouldn’t have to so consistently stay up late coding if I just used my time more effectively from the beginning.

Here I have included an image of the pie chart from WakaTime showing my most used languages this month. As I stated I did not have it installed to WebStorm for the first two and a half weeks so it is a little off but as this is the module we've done the most JavaScript in, mostly all of these hours were spent on the game.



References

Button images:

https://latenighcoffe.itch.io/buttons-blocks-for-ui-gameplay

SFX:

https://opengameart.org/content/25-cc0-mud-sfx

https://freesound.org/people/joedeshon/packs/7491/

https://opengameart.org/content/512-sound-effects-8-bit-style

Tilesets:

https://kicked-in-teeth.itch.io/pico-8-tiles