To begin working on this project, I first broke down what I'd need, and created a short workflow list of the different problems to break down, and an order to break them down.

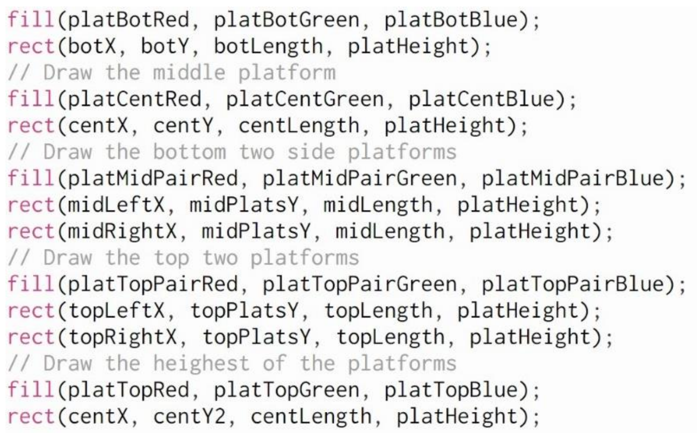
1. Create a series of Platforms over a static background
2. Create an avatar with the following parameters:
    - -Moves left and right with Player input
    - -Single Jump
    - -Land effectively on platforms
    - -Loops from the Left to the Right and vica versa
    - -Loops from bottom to top of screen
3. Create a number of collectables that, when the player crosses them, add to the score
    - -Eventually, the collectables will deal damage to enemies – code later
4. Create an enemy class that generates on a timer and follows a path
    - -When they encounter the player, the player dies
    - -When a collectable is grabbed, the oldest enemy dies
5. Create a secondary enemy class that generates randomly off the same timer
    - -This enemy chases the player, noting what platform they touched last
    - -I threw this out after I found the game's difficulty was already enough
6. Create a boss that begins spawning after a certain number of points (10)
    - -Collectables damage the boss instead of killing mobs
    - -The boss occasionally attacks a platform, making it dangerous
    - -After dying, the boss reappears after a set period of time
7. An opening screen with simple instructions, and a visually pleasing 'Begin'
    - -A game-over state that shows the score and allows for the player to restart
8. Adding in music that fits the feel of the game (The Easiest Section)
9. Replacing the filler images with actual sprites, designed by my little brother.
    - -Main Character Standing Still (Left and Right)
    - -Main Character Walking (Left and Right)
    - -Main Character Jumping (Left and Right)
    - -Collectable
    - -Pathed Enemies (Left and Right)
    - -Roaming Enemies (Left and Right)
    - -Background
    - -Background for Boss
    - -Boss
    - -Boss Attacking
    - -Boss Dying

In order to best explain how I went about this entire project, I'll go through each of these sections in order, explaining the logic and process and challenges that each brought.

<u>Goal 1: Platforms</u>

Firstly, Goal 1 was fairly simple – create a series of platforms over a static background. I created a filler background, and added the platform.js file to the currently small list of files. I drew out on paper where I wanted all the platforms to land, than created variables for each platforms X-value, Y-value, and length. All the platforms shared height, so that simplified things quite nicely.

After this, I realized that, far down the road, I'd need the platforms to be able to change color separately to properly communicate the boss's attack to the player. Because of this, I separated the three color variables into a massive list of 15 variables – which made the platform.js file significantly more complicated. However, I kept the three old variables, and when I first declared all of these new variables, I set them equal to the original color variables; that way, I could change all the platforms default colors at once.



```
fill(platBotRed, platBotGreen, platBotBlue);
rect(botX, botY, botLength, platHeight);
// Draw the middle platform
fill(platCentRed, platCentGreen, platCentBlue);
rect(centX, centY, centLength, platHeight);
// Draw the bottom two side platforms
fill(platMidPairRed, platMidPairGreen, platMidPairBlue);
rect(midLeftX, midPlatsY, midLength, platHeight);
rect(midRightX, midPlatsY, midLength, platHeight);
// Draw the top two platforms
fill(platTopPairRed, platTopPairGreen, platTopPairBlue);
rect(topLeftX, topPlatsY, topLength, platHeight);
rect(topRightX, topPlatsY, topLength, platHeight);
// Draw the heighest of the platforms
fill(platTopRed, platTopGreen, platTopBlue);
rect(centX, centY2, centLength, platHeight);
```

The fill and draw for the Platforms

As I write this now, I wished I had been taking screenshots throughout the entire process to help visualize all of these stages. I only have the final product, so I apologize if anything is difficult to understand.

Overall, this first goal was very basic – as a simple test of variables and creating a canvas with items where I wanted them, things went rather quickly.

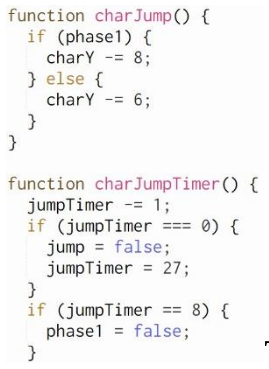
<u>Goal 2: The Avatar</u>

Creating the Avatar is where things got complicated. I began by creating the avatar.js file, and drew the character. I decided where I wanted the player to generate, and built the y location of the sprite off of the y-location of the middle platform. I worked out key presses, and had a character that could move left and right in no time (even if it was just a rectangle).

Afterwards, I created the jump.js file, and started with a simple static increase – when the up key was pressed, the character now flew upwards for a limited amount of time. All according to plan.

Then, platform checking and gravity came into play, and this is where I hit my first major wall. I created the gravity() function, that checked if the player was either interacting with a platform – and if they weren't they started falling. I started the project, and my player began to infinitely fall directly through everything.

I ended up finding a rudimentary way to get the jump to work. I played with the jump to even it, and made the jump always land the character on a number divisible by '5' on the y-axis. I changed the platforms y-locations to all land on the same kind of numbers, and set the effect of gravity to make "charY += 5". That way, the character would always perfectly land on a platform – this solved the problem, even if it is a messy way to create a jump.



```
function charJump() {
  if (phase1) {
    charY -= 8;
  } else {
    charY -= 6;
  }
}

function charJumpTimer() {
  jumpTimer -= 1;
  if (jumpTimer === 0) {
    jump = false;
    jumpTimer = 27;
  }
  if (jumpTimer == 8) {
    phase1 = false;
  }
```

The finished code to the jump:

14 frames x 8 + 8 frames x 6 == 160 total yShift; a number divisible by 5, and a large enough number to make all necessary jumps.

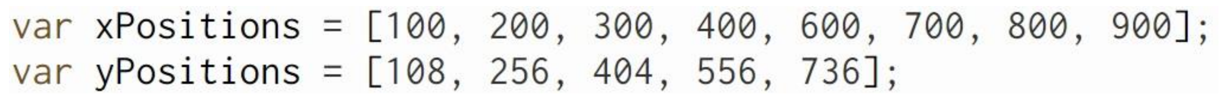
Next, I created locationCheck() to have our avatar loop around the screen. This went easily, with a small amount of bug-testing, as we've been using the basic statement progression I used throughout the entire semester. The character was now fully mobile and moved about as smoothly as I thought I could get them to move.

Goal 3: Collectables

When I first thought of the collectables, I thought I'd need to be creating a class that would generate and collect based off of location – however, things turned out simpler than that.

I quickly realized that, since only one collectable was onscreen at a time, I only need one collectable that APPEARED to be disappearing – but, in actuality, collecting the collectable just forces the shape's 'xPlace' and 'yPlace' to randomize, making it seem as if a new collectable had appeared. I added a score meter on the top of the screen that began to count, and slowly but surely worked out some awareness functions to check the player's location to allow for collection, and quickly realized two problems.

One, I had been randomizing locations by having the 'xPlace' and 'yPlace' variables each pull a random number from the following arrays.



```
var xPositions = [100, 200, 300, 400, 600, 700, 800, 900];
var yPositions = [108, 256, 404, 556, 736];
```

There were two problems with this. Firstly, the combination of:

xPlace = 100 && yPlace = 736

as well as

xPlace = 900 && yPlace = 736

created a spawn point in the lower left and right corner, that, without a perfectly placed jump, would be near impossible to grab – but all the other points worked so well, that I didn't want to give them up. So I created a do-while loop that forced the code to continuously pull from the arrays until they didn't get either of those combinations.

The second issue was double scoring – on rare occasions, the collectable would generate the same numbers it previously had, and the player would collect the second in the next frame; so I had to create an 'oldXPlace' and an 'oldYPlace' that would set, and add in another check for the do-while loop. Since the framework was already set from solving the first problem, this also went quite well, and I was able to move on quite well.

<u>Goal 4: Pathed Enemies</u>

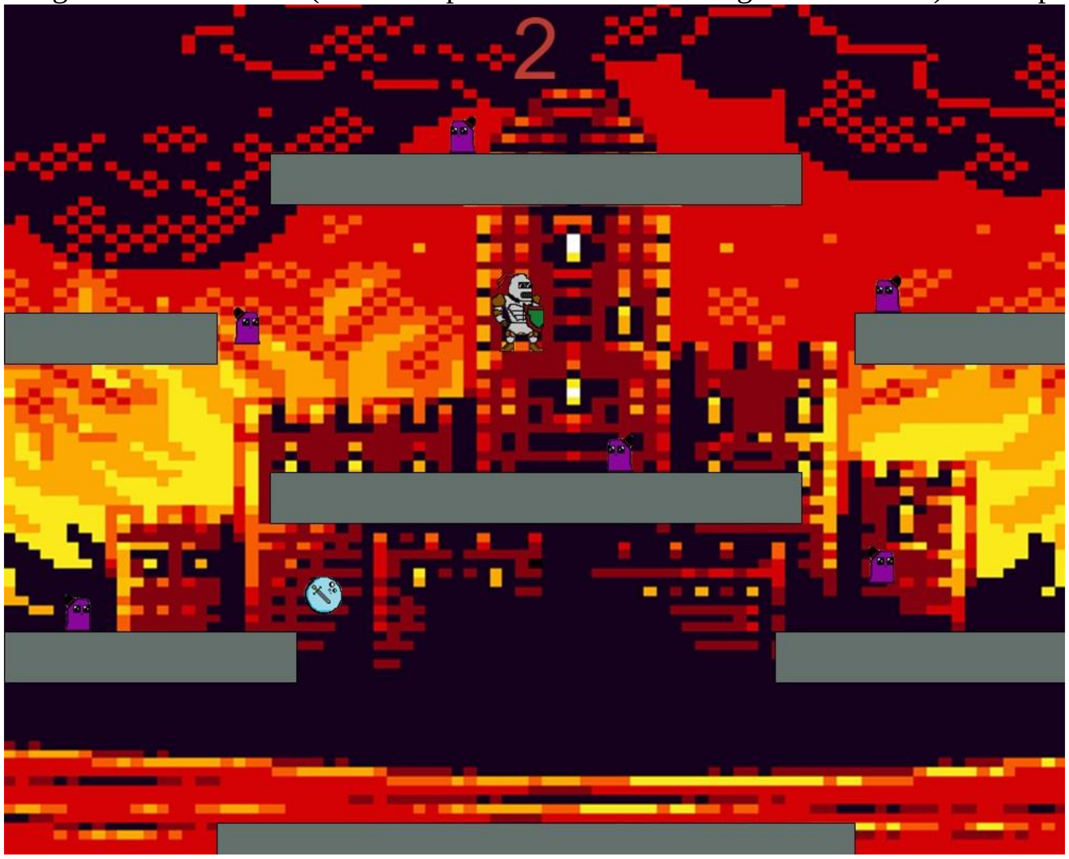
This is where I hit a roadblock, and eventually needed your help to continue. I created the PathedEnemy() class, and everything was going swimmingly. I generated a single enemy at the start, and set up all of the functions fairly quickly. The enemy used the same gravity() function as the avatar, and they were quickly moving exactly how I had hoped they would – and so I went to create the timer that would spawn another in the new spawnTimers.js file.

I made the timer with little issue – it's a simple countdown on a variable, where every call of the timer subtracts one from a variable until the variable hits zero, at which point the true function is called; this kind of timer is the timer I used throughout the entirety of the rest of the project. However, I couldn't get the new objects I was trying to generate to appear for the life of me.

Eventually, after a week of trying to figure out what to do with google, trial and error, anything I could think of, I came in and learned that, instead of a for loop to create enemies, I just needed the line;

```
pathed.push(new PathedEnemy());
```
.

So that started working – the game had now finally reached a playable state; here's an image of how it looked (but with sprites instead of rectangles and circles) at this point.



<u>Goal 5: Intelligent Enemies</u>

And then I took a bloody nap, because the code was starting to get complicated. It was already around 500 lines of code, and I'd poured around 16-20 hours into the project already. I looked at the project and realized that, even without the boss, these pathed enemies could get REALLY hard on their own – so I decided to cut the following enemies; not only to save time and actually get some sleep, but also to cut some slack on my brother (who was taking time out of his own finals) as well as to keep the game from getting too maddening for players.

At this point, I also started game testing. I did a few small tweaks to movement, and eventually nailed down a few spawn rates that kept the game interesting, but still simple understandable. Then, I moved on to the next phase.

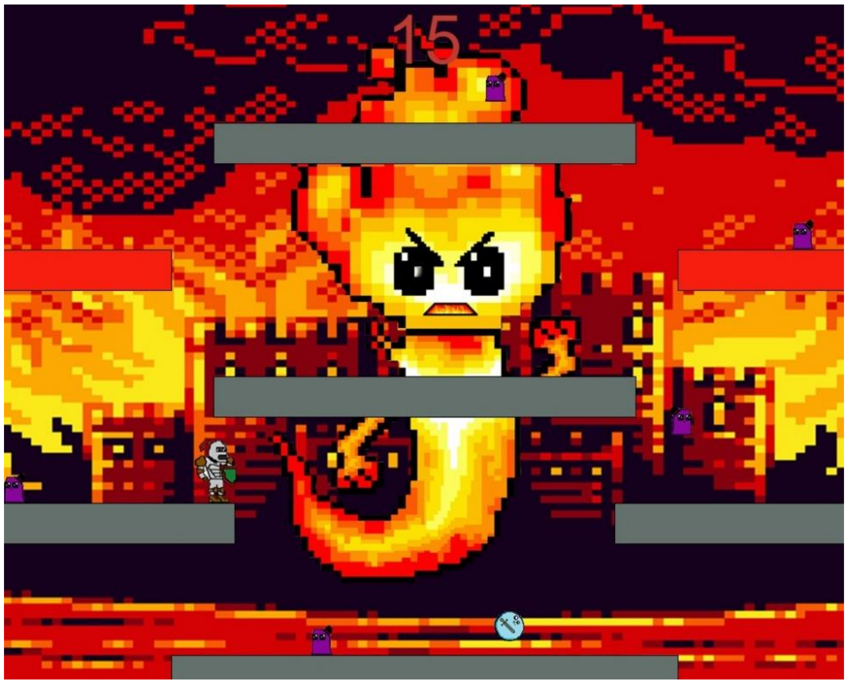
Goal 6: The Boss

This was, without a doubt, my favorite part of the project. Not only to code, but to logically think of, and to eventually play. Firstly, I needed a Boolean for if the boss was there or not, that I set into another timer that triggered after 10 tokens were grabbed. All my work setting up the platform variables came into play, and the bosses attacked slipped directly into what I had hope they'd be.

I went into the collectable.js and added a few more lines to influence the bossHealth variable whenever one was collected. I had to do a little testing of health and spawn rates for the mobs to balance the game, but once I eventually managed to get it working, the game had finally fallen into a playable state – I sent it out to my play testers once again.

Over the next few hours, they all helped me through the QA phase (however brief it was), giving me feedback on movement, things that confused them, and difficulty curves. Another shout-out to my brother for somehow managing to 46 points with 15 minutes of starting the game for the first time – in it's even more difficult state, no less.

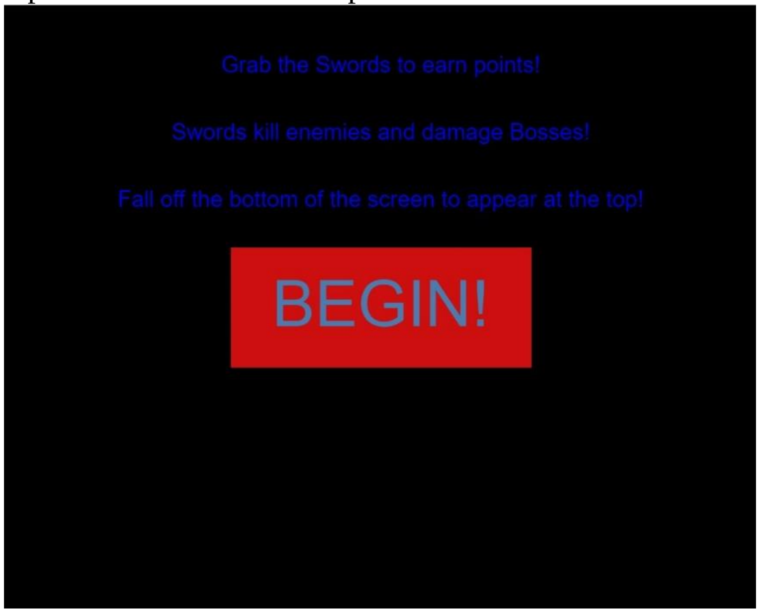
After spending a few hours working with the four of them continuing to fix some spawn rates and working the smallest bugs out, the boss was working within an hour of coding – by far the fastest section, and the most entertaining from a coding perspective. In addition, I now I had my game, gameplay wise, exactly where I wanted it – and I was very satisfied with everything I'd done.



The finished game (but there weren't sprites)

Goal 7: Start and End Screens

Another simple goal, I just added in a few if statements to allow the program to choose the screen the game would display – start screen, main game, or game over screen. From there, it was simply creating text, text effects, and a simple mousePressed() command – it was done within the hour, and the game had finally taken a concrete form. All that was left after these simple goals were to put in all the music and sprites.


Grab the Swords to earn points!
Swords kill enemies and damage Bosses!
Fall off the bottom of the screen to appear at the top!
BEGIN!

Goal 8 and 9: Music and Sprites

Initially, the game was supposed to have two backgrounds and two pieces of music – a cold background for when the boss wasn't present, and one for when the boss was. However, despite trying for an hour, I could not get the songs to fade in a way that didn't feel shoehorned in – so instead, you get a red background and 'Revenge of the Cyclops' for the entire game!

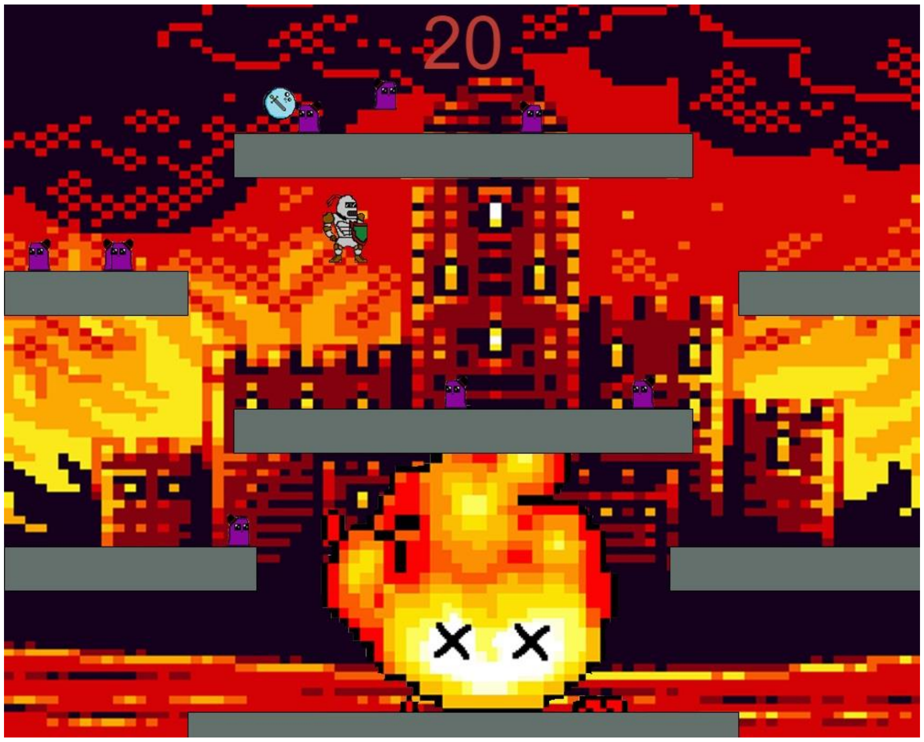
The sprites, made by my brother, were fairly simple to put in. I had to cut all of them out from white backgrounds, as my brother is in high school and the most advanced software he had access to was MS Paint. So, I took each of his exports into Photoshop, made sure they were sized perfectly for the dimensions, cut the sprites out and then place them over.

Two issues arose, and both were fairly easy to solve. Firstly, now that the avatar's feet were less wide than his shoulders, the platform check needed to be accounted for. I experimented, and eventually found a few variables that prevented 90% of cases where the knight would be floating on nothing. Now, the knight only really floats if the player goes out of their way to make the knight be JUST on the perfect edge.

Secondly, the collectable sprite was a square instead of an ellipse – so I had to add an 18 by 18 shift up and left so that the sprite appeared in the correctly location without having to change much of the code. Besides that, everything was simple if statements and image inserts.

At this point, I played with the platform colors to get them to a color that worked with the background, and made sure that everything could be easily seen and stood out from the background.

The project was now at a workable point, and I was satisfied. I did a few more bug fixes, changed a few more colors, and killed what is definitely the largest project I've completed this Semester.



The Boss in the middle of his dying animation

The Future

There are a few things I'd definitely consider doing if I want to keep pushing this, and I feel a list is the best way to convey these as well.

1. Add levels. Make each level have a different spawn, platform setup, and background – different enemies, different challenges, etc, and allow for scoring time on each of those levels. Giving each level a direct goal would be fun, making the game have a direct progression – this level (to the bosses' death) would probably be the first or second boss of the game.
2. Add more bosses. The fire elemental is a good start, but there are tons more bosses that could all exist as major challenges.
3. Add the boss' health as a display. I don't know where I'd place it, but that would tell new players how close they were to achieving their goal, hopefully creating more tension.
4. Create a running animation for the avatar, rather than a single still, and clean up the rest of the sprites.
5. Add dying animations to the fashionable slimes, as well as any other created enemies
6. Add textures to the platforms or create sprites with variable colors – they currently stand out from the other parts of the visual canvas as flat and bland.
7. Come up with a bloody Name. MART 340 Final Project is…meh.