**Project Two Game**

**Ben Weller and Aaron Beyer**

**CSCI 313**

**5/1/2017**

**Final Game Specification**

**Story:**

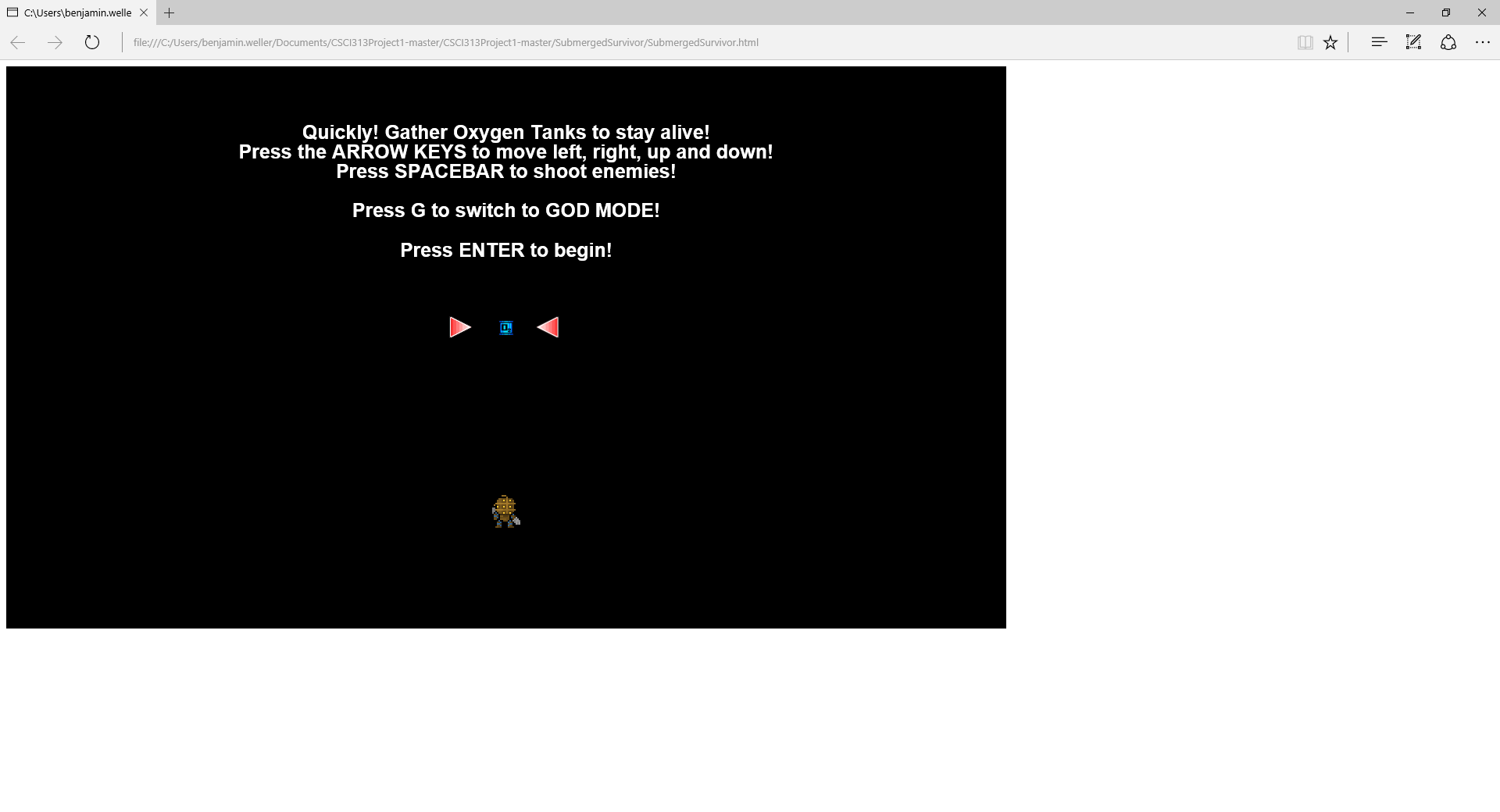
You are a scuba diver in peril, as you drain the last bit of oxygen from your tank you realize that you have enemies surrounding you, and that you need to find more oxygen and fast! With limited charges in your blaster you have to navigate your way through the series of platforms trying to find your next oxygen fix, all while avoiding enemies.

**Scene:**

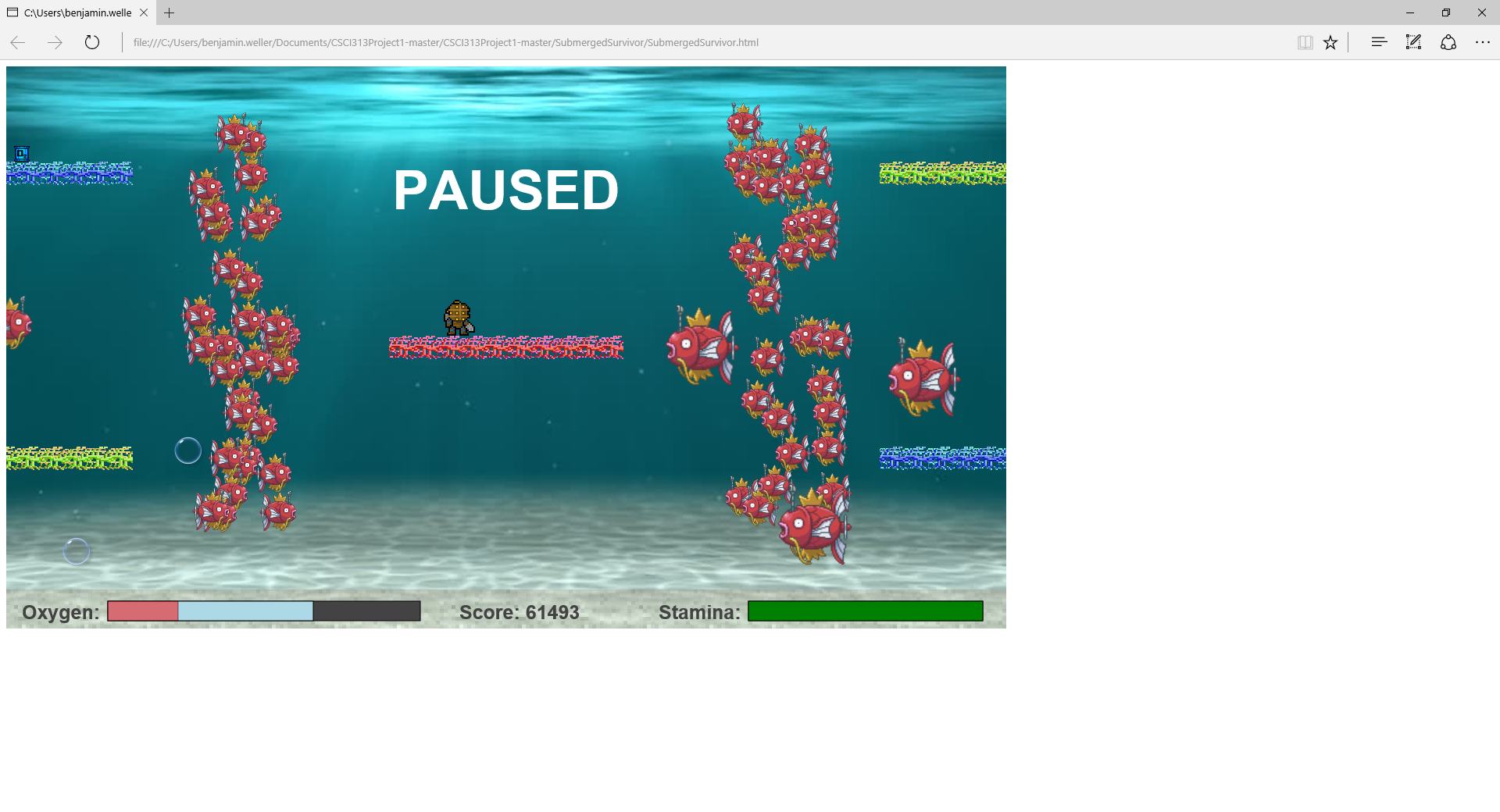
The scene is placed underwater, there is a “sandy” bottom and five platforms located in the arena. There is one center platform and two platforms that wrap around each side of the screen.

**Gameplay:**

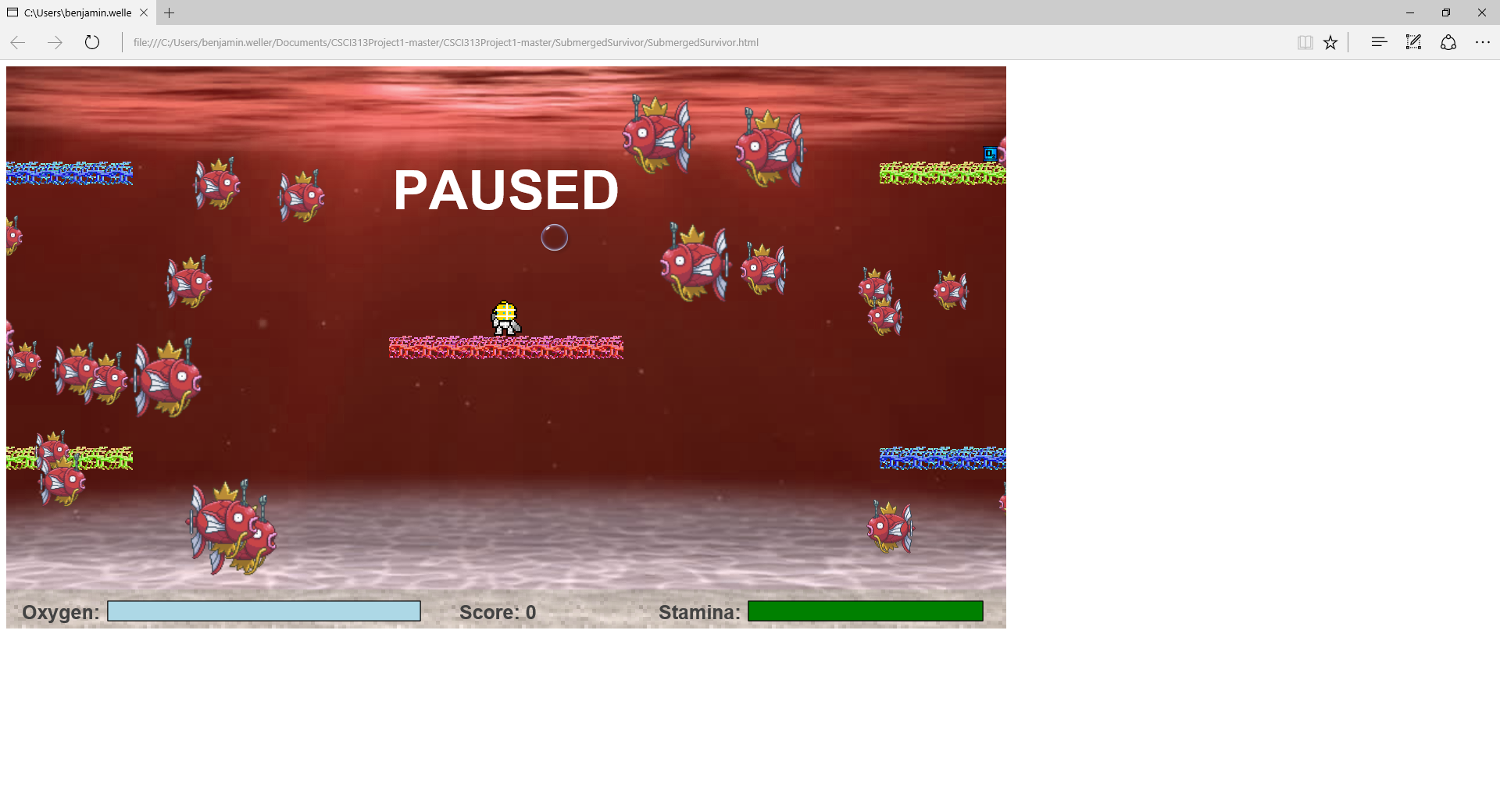
**Intro Screen:**



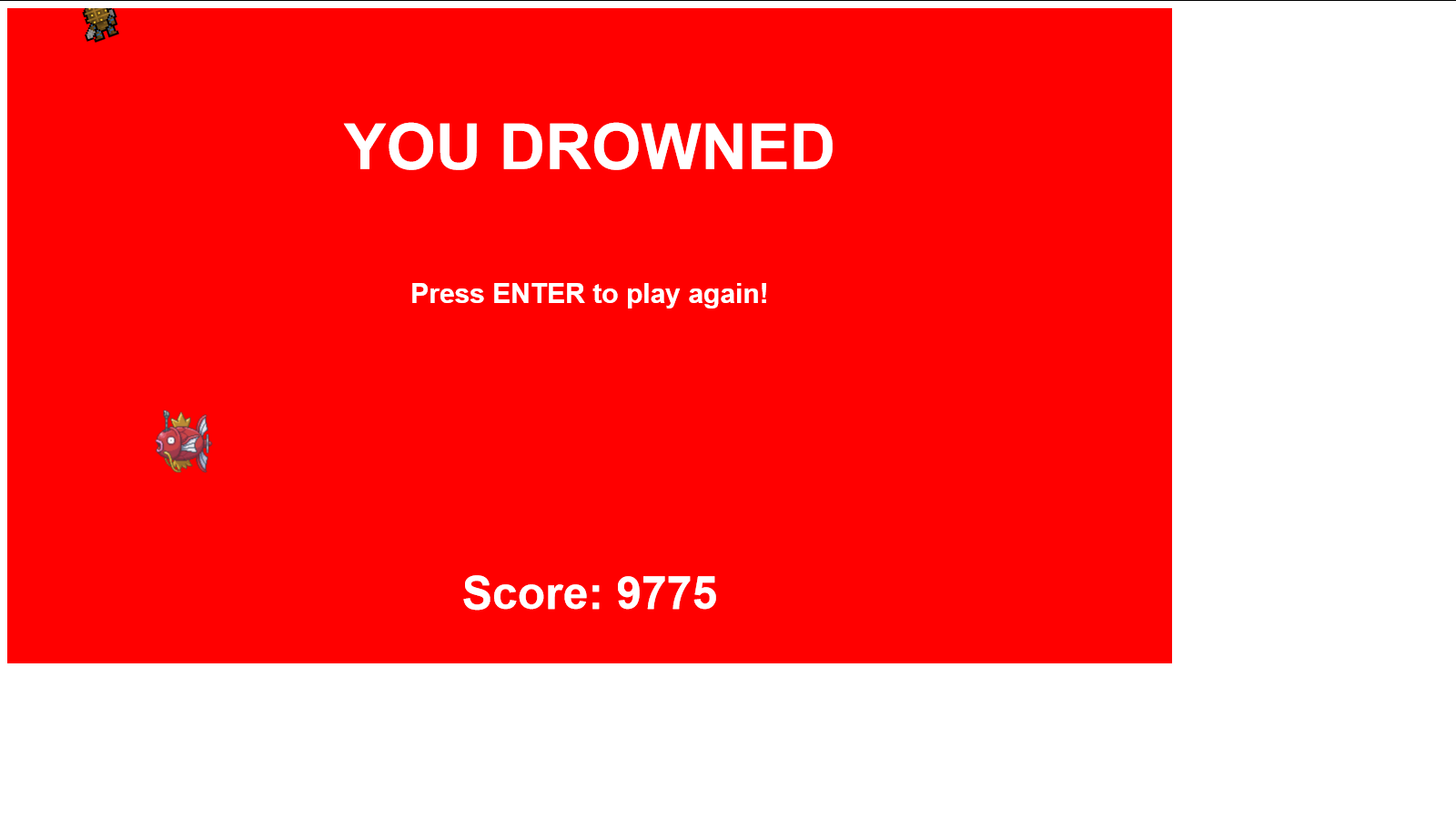
**General Game Example:**



**God Mode:**



**Death Screen:**



The game is played using the WASD/arrow keys for movement. To pause the game, use the enter key, and for firing bullets use the spacebar to fire a bullet in the direction the diver is facing.

The diver can wrap around the edges of the screen. He can go up through the bottom of a platform but will stand on a platform unless your direct him to go down through it. The diver can’t go through the “sandy” bottom and appear on the top of the screen, nor go through the top of the screen and appear on the bottom.

Enemies appear on both sides of the screen and swim away from where they came from. Once in a while there may be a mass production of enemies that form a wall blocking your progress.

You can shoot and remove enemies using the space bar. The diver will shoot whichever way he is facing. You can only shoot if you have an appropriate amount of stamina in the bar, if you deplete the bar, it will take longer to regenerate.

When you collect a tank your oxygen refills. As you collect more tanks your oxygen depletion rate increases, and your score that you get per tank also increases. When you collect a tank it is moved to a different random location among seven picked.

If you can’t collect an oxygen tank in time the drowning bar will increase, the game “slows down”. When you collect an oxygen tank after drowning it only refills half way. The drowning bar slowly drops.

As you increment your score, power-ups will fall down. These power-ups will either repair your tank (decrease the drowning bar), refill your tank (refill oxygen bar), or remove all enemies from the screen.

To enter “god mode” press “g”, in god mode your score will not be recorded and thus once your return to the “normal” game you won’t have your god mode score kept. In this mode you have unlimited stamina, and you will never drown.

The longer you stay alive the more of a score increase you get.

**Differences:**

Our game is different in that doesn't hide the 8-bit feeling. We make no effort to hide our game behind taxing graphics or overly complex gameplay. This is a simple what you see is what you get type game. There is no complex mouse following behavior and basically all instructions are given in one simple screen readout at the beginning. It is intended to become difficult very quickly and as such doesn’t lead to boredom during long drawn out “lives”.

**Major Topics:**

**Bitmaps:** The tank and diver are bitmaps displayed on the screen.

**Shapes:** The oxygen and stamina bar are both shapes that are modified using command objects.

**Animations:** The oxygen depletion and stamina regeneration are both examples of animation.

**User Input:** The user uses the keyboard to move the character around the screen to shoot and to pause the game (WASD/Arrow keys, Spacebar, and Enter respectively).

**Containers:** The bullets the diver shoots are containers. Additionally, the enemies are in containers and as such are easily removed when hit.

**Sprite Animations:** The magikar ps that swim across the screen are examples of sprite animation.

**Design:** Since we decided to build upon our project from the first half of the semester, it was pretty difficult for us to completely change the design of the code. We tried switching it over to a design including objects (using IIFEs) but had a lot of problems. In the end, we decided to focus on changing our code to be understandable at a high-level. In doing so, we split our init() and game\_step() functions into different functions to make the code easy to understand. Also, we focused on adding documentation for each function and other comments within the functions to help readers understand the code. Thus, in the end, we decided to add many more features to the game rather than completely redesigning the code. We still focused on making the design better, but we were not able to completely change the design to implement objects (using IIFEs).

In addition to IIFEs we had a few other design details that we dabbled in. Specifically, the use of generic functions, and the use of events in the case of game over and restarting the game.

With regards to the generic functions we created functions such as the genericCollisionMethod() and the removeFromPowerUpArray() methods to generically handle situations that would be a common occurrences during the game. These methods allowed us to not only shorten our code but also ensure that we had some functionality provided in the future lest we chose to use it.

With regards to the even driven portion we chose to “scrap” it seeing that it provided no further functionality and we felt it further convoluted the code. It was functional but it was more difficult to understand without providing any tangible benefits.

**Differences in Initial to Final Product:**

**Beginning:**

The game was initially thought up to be a swimming game where it simply got to be so difficult that there would be no room for error and as such you had to focus strictly on the game.

**Changes:**

The screen size was initially 800x600. During the second week of the project it was proposed to make the screen bigger and thus change the gameplay. This meant that platforms would have to be moved, tank spawns would have to be changed, the opening and closing screen would be modified and the oxygen and score indicators would have to be relocated. This change was implemented because the screen was just a bit too small and without the expansion it wouldn’t have had room for enemies.

The thought of enemies was also pitched in the second week and the game radically changed. Then we had to add to include collisions between the enemy and the player, and we would have to include some form of protection for the player (thus the advent of bullets). This change was implemented to further enhance the game (it felt a little bit plain without it).

Following the spring break the course shifted to a design focus. With 1,200 lines of code already written it wasn’t a reality for either of us to overhaul the design of the game. Needless to say, we did try through the use of IIFEs and events, but in the end we chose to scrap both of those simply because they fell short of their aim.

In an effort to still improve our game we implemented a plethora of additional features. These included “god mode”, power-ups, redesign of game textures, allowing enemies to come from both sides, and adding more sounds to immerse the player in the game. All of these were a significant undertaking, and involved a sizable commitment of time to successfully implement.

**Features:**

|  |  |
| --- | --- |
| Task: | Name: |
| Generic Methods | Ben |
| Updated game over logic | Ben |
| Added power-ups | Ben |
| Design document | Aaron/Ben |
| Added new sprites/background | Aaron |
| Initial “Java-doc comments” | Ben |
| Background music | Aaron |
| God Mode | Aaron |
| Enemies from both Sides | Aaron |
| Sound Effects for collisions with enemies | Aaron |
| IIFEs/Objects (Scrapped) | Aaron |
| Events for game over/restart (Scrapped) | Ben |
| Final “Java-doc comments” | Aaron |
| Simplifying of init()/game\_step() | Aaron |

**Code Table of Contents:**

**High Level Outline:**

Load()

Init()

game\_build()

set\_controls()

game\_start()

addEventListener(“tick”, game\_step)

game\_step()

check\_controls()

handleKeyDown(e) –Handles key events

handleKeyUp(e) –Handles key events

apply\_gravity()

onPlatform(p) –Gravity

add\_enemies()

createFish(Normal)/createFish(Wall) –Creates enemies

change\_oxygen\_and\_stamina()

oxygenBarLogic()

drowningLogic() –Does the downing logic

isGameOver() –Determines if the game is over

gameOver()

resetGame()

check\_collisions

checkBulletCollisions(index)

checkFishCollision(index) –Removes the fish if it’s been hit

createBullet() –Bullet Logic

getRandomColor()

checkTankCollision() –Checks collisions with the tank

genericCollisionMethod(arguments)

movesTank() –Moves the tank

powerUpLogic()

removeFromPowerupArray(array)

createPowerUp()

powerUpCollisions()

genericCollisionMethod(arguments)