Names: Malhar Shah, Edwin Zhang

Lab: Thursday 9 - 12

Station: 14

Game Title: Infinite Swimmer

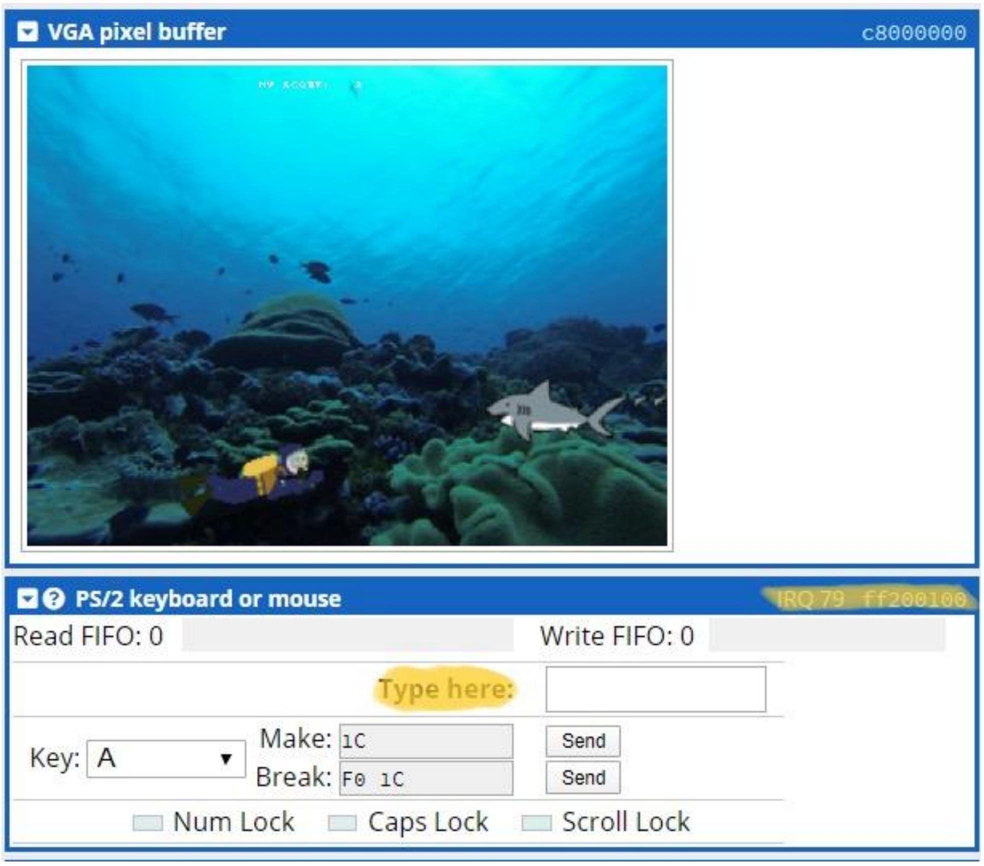
Project Description:

You are staying on your private island during the quarantine and you decided to go for a scuba dive. Suddenly you see and endless number of sharks! You need to swim your way through without touching or coming too close otherwise you can seriously get hurt! The waves are strong so you don't have great control over your body, but you must do the best you can.

This game is an infinite runner, or in this case, an infinite "swimmer." The sharks will randomly spawn for until the player dies by hitting the shark.

Controls:

Used W-A-S-D keys on the keyboard to move your character. Simply tapping is enough to make your character to move in a direction (Holding down overloads CPUlator PS/2 port). Make sure to type/enter W-A-S-D into the **PS/2 Port** with address **ff200100**. It is recommended that you drag the tab to be right under the VGA display as shown below. It is important to select the box **Type Here** before trying to move the player.



VGA pixel buffer c8000000

PS/2 keyboard or mouse IRQ 79 ff200100

Read FIFO: 0 Write FIFO: 0

Type here:

Key: A Make: 1C Send

Break: F0 1C Send

Num Lock Caps Lock Scroll Lock

How to load the game on CPUlator:

Our project is done on CPUlator. Since CPUlator does not support multiple files, we have included all our work combined in **main.c** file. Simply **copy the entire file and paste it in CPUlator in C mode.** Once copied, press the Compile and Load Button. Next, press continue twice to begin the game. Enjoy!

Our Code:

As our code is quite lengthy, this section summarizes how it works.

To draw our sprites, we preload images into a bitmap. This can be seen by the large arrays at the top of the file. Our draw functions only iterate the required elements of the bitmap array by using player/obstacle positions as boundaries to the iteration. To re-draw sprites after movement, our clear functions only draws the background over the previous sprite position. Doing the techniques mentioned enables for faster animation times.

Our obstacle spawning algorithm is based off a randomized spawning interval and position. Obstacles are stored in a linked list as it enables for easy deletion once the obstacle is offscreen.

The algorithm used for collision is by position comparison. The algorithm compares the player position with respect to each shark position to determine if they are collided. This algorithm is less accurate than our original pixel detection algorithm but faster in terms of runtime. This sacrifice was necessary as the pixel detection would have caused excessive lag on CPUlator and rendering the game unplayable.

Attribution table:

The work done in this project is 50-50.

Malhar	**Edwin**
• Player graphics • Background graphics • Obstacle graphics • Score Display On Screen • Integration with game functionality	• Obstacle spawning algorithm • Player & obstacle movement with keyboard integration • Player collision detection • Any data structure requirements ie. Player structs, linked list