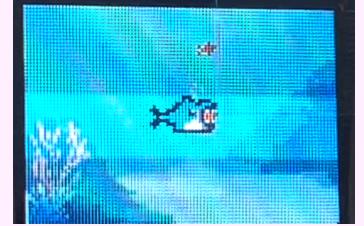
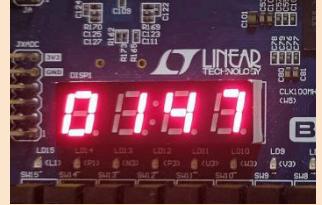


PERSONAL AND TEAM IMPROVEMENTS		
Student and Improvement Name	Improvement Description	Images/Photos
Team S1_12 "Famished Fish"	<p><u>Controls</u></p> <p>btnU / btnD -> Scroll Menu Options btnC -> Choose Selected Menu Option btnL -> Previous Menu btnR -> Pause Game Mouse L/R-> Set Target Location For Shark to Swim To Middle Mouse Button -> Activate Shark Nitro</p> <p>SW[3:0] Must be on for the game to start</p> <p><u>Intro</u></p> <p>Our project is largely inspired by the Hungry Shark mobile game series. In Famished Fish, the player controls a shark displayed on the OLED to consume as much fish as possible before its satiety bar becomes empty.</p> <p>The player must continuously eat fish to replenish the bar. If satiety(all LED turns off) hits zero, the shark dies instantly, and the game ends. If the shark collides with seaweed at the bottom of the screen, the player is taken to a separate screen to "battle" the seaweed. Failing this challenge will end the game.</p>	  
Student A: Zhang Yijian Cursor/Shark Control, Fish Spawn System, Shark Collision Mechanics, and Score Display	<p>The target location is set using the left/right mouse button. Once a target location is chosen, the shark will navigate towards the target location automatically at a speed that is based on its current position and speed. The target can be set dynamically even before the shark reaches the target location, and the shark's movement will be updated accordingly to match the new target location. The path the shark takes is indicated by the white line drawn on the screen.</p> <p>The 'nitro' feature allows the shark to move at 150% speed by adjusting various game timers. However, using 'nitro' will also decrease the satiety bar 150% faster.</p> <p>There are three scenes in the game which the shark can traverse into. The shark starts in the middle scene. If the shark reaches the border, it will be sent to the adjacent scene if it exists. If there is no adjacent scene (e.g. no scene to the left of the leftmost scene), the shark will remain at the border of the scene.</p> <p>The fish spawn system uses a LFSR to randomise fish spawn locations at a set frequency. Fish movements are handled at every game tick, where a linear search is used to track all existing fish and move them down by one pixel. A collision area (hit-box) was made for both the shark and the fish spawned to detect collision.</p> <p>The seven-segment display was also programmed to show the score, which increments whenever the shark successfully consumes the fish.</p>	 <p>Shark Pathing / Fish Spawn (Raw)</p>  <p>Shark Pathing / Fish Spawn (With Graphics)</p>  <p>Score Display</p>

<p>Student B: Brian Chan</p> <p>Main Menu, pause, end screen, guide, character design and control logic, LED Health Bar.</p>	<p>To access the main menu from the dynamic loading screen, you have to enter a password which is “SW0 to SW3”.</p> <p>For the main, pause and end menu, scroll up and down the options using btnU and btnD. Selection of the submenu is done by pressing btnC, and btnL is reserved for backtracking. Debouncing was implemented for the menus. Highlighting of the box was done using the if else statement, based on x and y coordinates of the boundary.</p> <p>For character selection, when on the character, the box will be red. When a character is selected (by pressing btnC), it will be green. Control logic for characters is done by outputting 1 into a register(select_shark/hammer) which is passed into the animation module, controlling the character selected.</p> <p>Created the initial logic for pause and retry page (btnC to select, btn L backtrack), which was refined and integrated by Zhan Hao.</p> <p>LED health bar takes in 4 MAIN inputs, game_start, game_active, shark_eat_trigger, as well as y_pixel. Health Bar starts at full and resets every game. Y_pixel determines the speed that LED turns off. Consuming a fish (shark_eat_trigger) will decrease the count (turn LED on). When below 50% health, the LED health bar will flicker. When LED health bar is off, game is over(enter end screen)</p>	 <p>Health Bar</p>  <p>End screen and Instruction</p>  <p>Loading screen and main menu</p>  <p>Character selection and pause menu</p>
<p>Student C: Cai Jiali</p> <p>Game Animation, Map design and LUT control</p>	<p>The visual elements in the main game were carefully designed to accommodate the OLED's low resolution, ensuring clear and smooth animations. Actions like the shark's eating were animated to enhance user experience, and both sharks have mirrored versions to provide a more natural feel when moving left or right.</p> <p>The three-scene underwater map was created, featuring details like bubbles rising and shrinking due to pressure and a brightening sky toward the top, enhancing realism. Seaweeds were designed to vary in size and shape to enrich gameplay, with each plant individually controlled to support gameplay mechanics.</p> <p>Memory limitations were carefully managed by separating the storage of dynamic objects like bubbles and static objects like the maps and storing complex graphics, such as the map, in block memory, reducing LUT usage, and optimising speed of bitstream generation.</p>	
<p>Student D: Hoe Zhan Hao</p> <p>Seaweed, Reset, and Pause Logic</p>	<p>Upon colliding with seaweed at the bottom of the screen, the player enters a minigame utilising mouse movement and left-click. Four seaweed strands appear on the OLED, highlighted in a randomised sequence for the players to follow. The sequence is determined by an LFSR generating random numbers from 0 to 23. A 0.5-second input delay is implemented to prevent accidental clicks. If the player cuts the seaweed in the correct order, they return to the game, with the collided seaweed disappearing to simulate the shark's escape. An incorrect sequence results in an instant loss.</p> <p>Clicking "exit" or "retry" on the pause or lose screen triggers a scene transition and game reset. "Exit" redirects to the main menu, while "retry" returns the player to the game. In both cases, the score, satiety bar, and seaweed are reset. The game remains paused while in the main menu or pause screen.</p>	 <p>Seaweed highlighted in sequence</p>  <p>Seaweed disappears once the minigame is won</p>

References

Map design:

<https://www.freepik.com/vector/sea-animal>

Seaweed design:

<https://opengameart.org/content/dancing-seaweed>

Main Menu Design:

Pixel Art Ocean Images - Free Download on Freepik

Shark/Orca Design:

[Orca Killer Whale Pixel Art Icon Stock Vector \(Royalty Free\) 1083427289 | Shutterstock](https://Orca Killer Whale Pixel Art Icon Stock Vector (Royalty Free) 1083427289 | Shutterstock)
Premium Vector | Cute shark in pixel art style

Pause, End, Main Menu Background:

Pixel art water background. Seamless sea texture backdrop. Stock Vector | Adobe Stock

Loading Screen Background:

Pixel Art Shark Photos, Images & Pictures | Shutterstock

BRAM:

https://circuitfever.com/store-an-image-on-fpga-verilog#google_vignette

<https://nandland.com/lesson-15-what-is-a-block-ram-bram/>

Usage of ChatGPT for miscellaneous tasks

pic2pixel.py: code for conversion of image to verilog code (framework generated by ChatGPT)

```
from PIL import Image
# Load the image
image = Image.open('image.png')
# Convert the image to RGB mode
image = image.convert('RGB')
# Get pixel values
pixels = image.load()
width, height = image.size
# Prompt the user for starting positions i and j
i = int(input("Enter the starting X position (i): "))
j = int(input("Enter the starting Y position (j): "))
# Prompt the user for ending positions x and y
x_end = None
y_end = None
prev_x = None
prev_y = None
# Print color ranges while traversing the image
def print_color_ranges(output_file):
    output_file.write("/* None */\n")
    x_start = None
    y_start = None
    prev_x = None
    prev_y = None
    for y in range(height):
        for x in range(width):
            r, g, b = pixels[x, y] # Get RGB values of the current pixel
            # Skip the pixel if it matches the excluded color
            if (r, g, b) == (0, 0, 0):
                continue
            shark_color = rgb_to_16bit(r, g, b) # Convert color to 16-bit binary
            # Check if this pixel continues the same range (same color and consecutive pixels)
            if current_color == shark_color and prev_x == x - 1 and prev_y == y:
                # Extend the current range
                x_end = x
                y_end = y
                prev_x = x
                prev_y = y
                continue
            else:
                # If the color changes or it's not consecutive, finalize the previous range
                if current_color != prev_color:
                    if y_start == prev_y:
                        # Single-row range
                        output_file.write(f"else if ((x >= {x_start} && x <= {x_end}) && (y == {y_start})) oled_color <= 16'b({current_color});\n")
                    else:
                        # Multi-row range
                        output_file.write(f"else if ((x >= {x_start} && x <= {x_end}) && (y >= {y_start} && y <= {y_end})) oled_color <= 16'b({current_color});\n")
                # Start a new range
                current_color = shark_color
                x_start = x
                y_start = y
                prev_x = x
                prev_y = y
    # At the end of each row, we still need to finalize the current range
    if current_color != prev_color:
        if y_start == prev_y:
            # Single-row range
            output_file.write(f"else if ((x >= {x_start} && x <= {x_end}) && (y == {y_start})) oled_color <= 16'b({current_color});\n")
        else:
            # Multi-row range
            output_file.write(f"else if ((x >= {x_start} && x <= {x_end}) && (y >= {y_start} && y <= {y_end})) oled_color <= 16'b({current_color});\n")
# Save the output to a text file
with open("color_ranges_output.txt", "w") as output_file:
    output_file.write("/* Generating Verilog code with starting X (i) = ({i}) and starting Y (j) = ({j})...\n\n")
    print_color_ranges(output_file)
    print("The output has been saved to \"color_ranges_output.txt\"")
```

pic2coe.py: code for conversion of image to .coe file for BRAM (framework from the open source stated in reference and ChatGPT)

```
from PIL import Image
def rgb_to_16bit(r, g, b):
    """Convert RGB values to a 16-bit integer in 5:6:5 format.
    Args:
        r (int): Red component (0-255).
        g (int): Green component (0-255).
        b (int): Blue component (0-255).
    Returns:
        str: A 16-bit binary string representation of the color.
    """
    r_5bit = (r >> 3) & 0x1F # Convert red to 5 bits
    g_6bit = (g >> 2) & 0x3F # Convert green to 6 bits
    b_5bit = (b >> 3) & 0x1F # Convert blue to 5 bits
    return f'{r_5bit:05b}{g_6bit:06b}{b_5bit:05b}'

def png_to_coe(png_file_path, coe_file_path, radix=16):
    """Convert a PNG image file to a .coe file with memory initialization for Vivado.
    Args:
        png_file_path (str): Path to the input PNG file.
        coe_file_path (str): Path to the output .coe file.
        radix (int): Radix for the memory values (16 for hexadecimal).
    """
    # Open the PNG image and convert it to RGB mode
    image = Image.open(png_file_path).convert('RGB')
    width, height = image.size

    # Prepare the header of the .coe file
    coe_content = f"memory_initialization_radix={radix};\n"
    coe_content += f"memory_initialization_vector={hex(0)}\n"

    # Collect the RGB values as 16-bit data in 5:6:5 format
    color_data = []
    for y in range(height):
        for x in range(width):
            r, g, b = image.getpixel((x, y))
            # Convert RGB to 16-bit binary format (in 5:6:5 format)
            if radix == 16:
                color_data.append(f"{int(rgb_to_16bit(r, g, b), 2):04X}") # Hexadecimal format
            else:
                color_data.append(rgb_to_16bit(r, g, b)) # Binary format

    # Join all pixel data with commas and add a semicolon at the end
    coe_content += "\n".join(color_data) + ";\n"

    # Write to the .coe file
    with open(coe_file_path, 'w') as coe_file:
        coe_file.write(coe_content)

    print("Conversion complete! The .coe file is saved as (coe_file_path)")

# Usage example:
png_file_path = "settings.png" # Replace with your PNG file path
coe_file_path = "settings.coe" # Desired .coe file path
radix = 16 # Set radix as 16 for hexadecimal (Vivado usually uses hexadecimal)
png_to_coe(png_file_path, coe_file_path, radix)
```

newconverter.py: code for conversion of image to OLED data (Refined by ChatGPT)

```

1  from PIL import Image
2
3  # Convert a 16-bit RGB value to the Verilog format (5-bit Red, 6-bit Green, 5-bit Blue)
4  def rgb_to_verilog(r, g, b):
5      # Scale RGB values to their respective bit-widths
6      red = (r >> 3) & 0b11111    # 5-bit Red
7      green = (g >> 2) & 0b111111    # 6-bit Green
8      blue = (b >> 5) & 0b11111    # 5-bit Blue
9
10     # Combine into a 16-bit number
11     verilog_color = f"16'b{red:05b}_{green:06b}_{blue:05b}"
12
13     return verilog_color
14
15  # Convert image to 16-bit color parameters
16  def image_to_color_parameters(image_path):
17      img = Image.open(image_path)
18      img = img.convert("RGB")    # Ensure RGB mode
19      img = img.resize((96, 64))    # Resize to 96x64 pixels
20      width, height = img.size
21
22      color_data = []
23
24      # Iterate over each pixel and convert to Verilog 16-bit format
25      for y in range(height):
26          for x in range(width):
27              r, g, b = img.getpixel((x, y))
28              color_data.append(rgb_to_verilog(r, g, b))
29
30      return color_data, width, height
31
32  # Generate Verilog assignments for the OLED data array
33  def generate_verilog_assignments(color_array):
34      verilog_assignments = []
35      for i, color in enumerate(color_array):
36          verilog_assignments.append(f'image_data[{i}] = {color};')
37
38      return '\n'.join(verilog_assignments)
39
40  # Example usage
41  image_path = 'C:\\EE2026\\famishedfish2.png'    # Replace with your image path
42  color_array, img_width, img_height = image_to_color_parameters(image_path)
43
44  # Generate the Verilog assignments
45  verilog_output = generate_verilog_assignments(color_array)
46
47  # Write the Verilog output to a file
48  with open("verilog_output.txt", "w") as f:
49      f.write("module image_data;\n")
50      f.write(" reg [15:0] oled_data[0:6143];\n")    # Adjust for the new size
51      f.write(" initial begin\n")
52      f.write(verilog_output + "\n")
53      f.write(" end\n")
54      f.write("endmodule\n")
55
56  print("Verilog output written to verilog_output.txt")
57
58

```

```

        color_data.append(pixel_to_color(r, g, b))

    return color_data, width, height

# Generate Verilog assignments
def generate_verilog_assignments(color_array):
    verilog_assignments = []
    for i in range(len(color_array)):
        verilog_assignments.append(f'oled_data[{i}] = {color_array[i]};')

# Example usage
image_path = 'C:\\EE2026\\famishedfish2.png'    # Replace with your image path
color_array, img_width, img_height = image_to_color_parameters(image_path)

# Generate the Verilog assignments
verilog_output = generate_verilog_assignments(color_array)

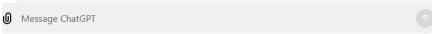
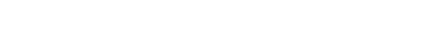
# Write the Verilog output to a file
with open("verilog_output.txt", "w") as f:
    f.write("module image_data;\n")
    f.write(" reg [15:0] oled_data[0:6143];\n")    # Adjust for the new size
    f.write(" initial begin\n")
    f.write(verilog_output + "\n")
    f.write(" end\n")
    f.write("endmodule\n")

print("Verilog output written to verilog_output.txt")

```

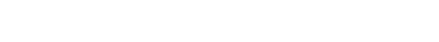
can you change this code to include ALL SPECTRUMS OF COLOUR THAT CAN BE FORMED WITH 16 BITS thank you

```

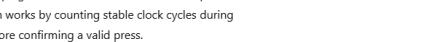
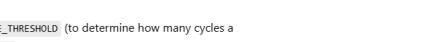
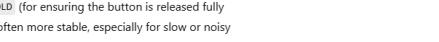
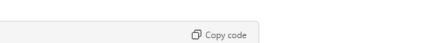
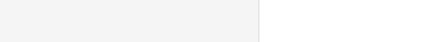
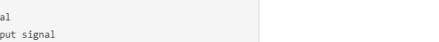




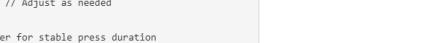
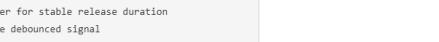
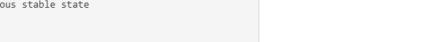
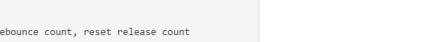
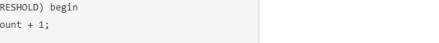
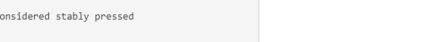
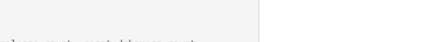









































































































































































































































































<img alt="Screenshot of
```