# **CS162 Final Project**

1. What does your program do?

Our program is a fishing game using PyGame. The user controls a cat fisherman on a boat with a catch goal and time limit. Fish cross the screen from left to right. The user can change the location and the strength of the cast. A fish is hooked when the head of the fish touches the fishing hook. Once hooked, the score increases based on the type of fish caught. There are four types of fish, c: common, uncommon, rare, and the ultimate catch. The user loses the game by not reaching the catch goal within the time limit. The user wins the game by completing the catch goal within the time limit.

2. How is it executed? What libraries are needed to run the program?  
 (include a few screenshots if necessary)

This program, called "Fish Game, " is implemented using the Pygame library. Pygame is a popular library for creating games and multimedia applications in Python. The program uses pygame, os, random, and pathlib libraries. You can run the program by calling the main\_menu() function, which starts the game and opens the main menu screen.

3. What features does the program have?

Main Menu:

* Display the main menu screen.
* Create buttons for different options (Start Game, Background Story, Fish Encyclopedia, Controls, Quit).
* Handle button clicks and perform corresponding actions (e.g., start the game, show the background story, display fish encyclopedia, show controls, quit the program).

Start Game:

* Initialize the game environment.
* Set up the game window.
* Load background image.
* Create fish objects of different rarities (Common, Uncommon, Rare, Ultimate).
* Create the FisherCat object.
* Create the scoreboard object.
* Handle user input for controlling the cat-fisherman and catching fish.
* Update fish positions and check for collisions with the fishing bob.
* Draw the game elements on the screen (background, fish, cat, scoreboard).
* Control the frame rate.

Background Story:

* Display the background story screen.
* Create a button to go back to the main menu.
* Handle button click to return to the main menu.

Fish Encyclopedia:

* Display the fish encyclopedia screen.
* Create buttons for different fish categories (Common Fish, Uncommon Fish, Rare Fish, Ultimate Fish, and Back).
* Handle button clicks and perform corresponding actions (e.g., show specific fish category, return to the main menu).

Controls:

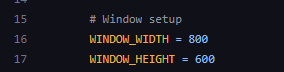
* Display the controls screen.
* Create a button to go back to the main menu.
* Handle button click to return to the main menu.

4. Which part of your program satisfies what requirements? Include screenshots if necessary.

**Variables, conditionals, loops, and collections**

Our program uses variables, conditionals, loops, and collections throughout the code.

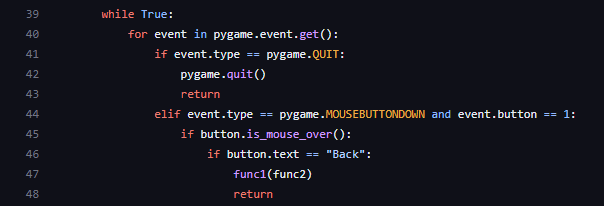
**Variables(encyclopedia\_funcs.py, line 16)**

****

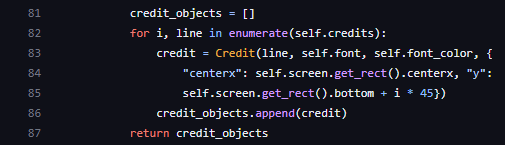
**Conditionals - if statement(encyclopedia\_funcs.py, line 41)**

****

**Loops- While loop(encyclopedia\_funcs.py, line 39)**

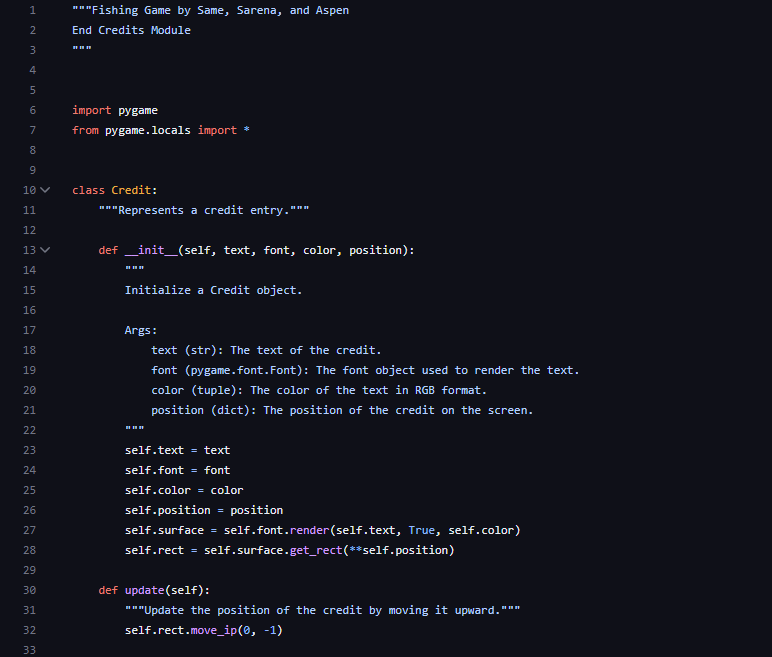
****

**Collections(endcredits.py, line 81)**

****

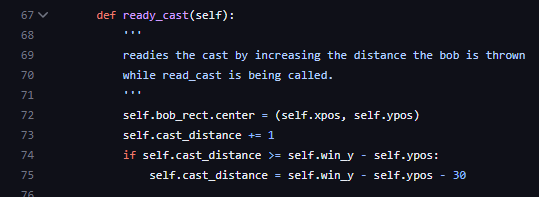
**Code organization**

Our program follows a well-structured organization with folders and files. The code demonstrates effective code organization practices, including appropriate formatting, meaningful variable and function names, and concise function definitions. We ensure proper indentation and line spacing throughout the code, enhancing readability. The function definitions are articulate and precise. The file endcredits.py shows examples of formatting, meaningful variable and function names, and line spacing.



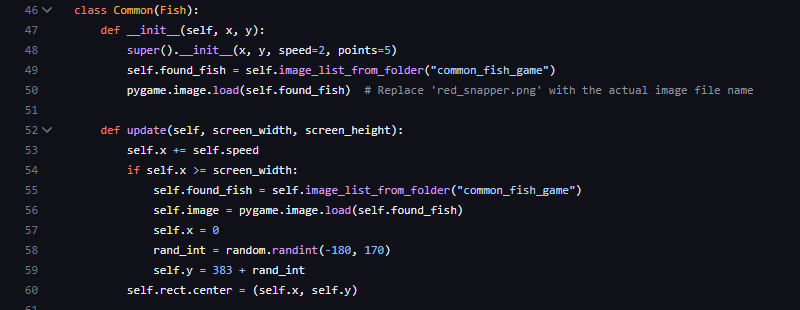
**Code decomposition (functions, classes, methods, and modules)**

Our program is organized into many different classes and functions. The file fisher\_cat\_class.py shows an example of a class with multiple functions, including draw(), move\_left(), and ready\_cast() on line 67.



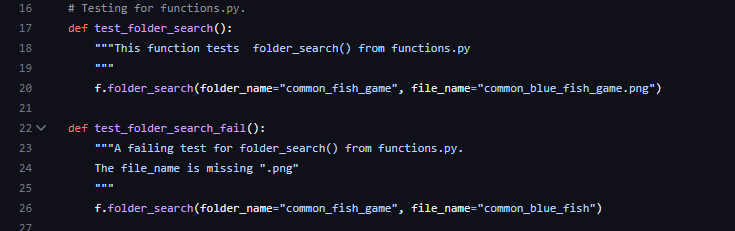
**An understanding of design (including the hierarchy of an inheritance tree)**

Our program shows an example of inheritance in the file fish\_classes.py. The class Common inherits from the Fish class on line 46.



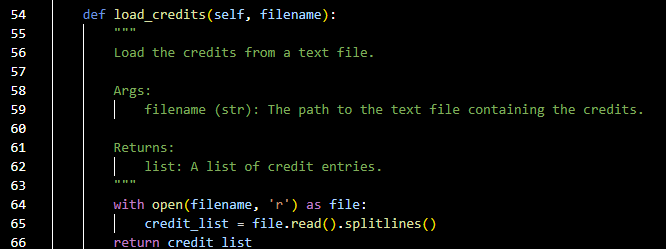
**An understanding of testing (test your methods and attributes, maybe have a whole automated example!)**

We used multiple test functions to test our code and the code as we worked on it. Test functions can be found in the file test\_functions.py starting on line 17.

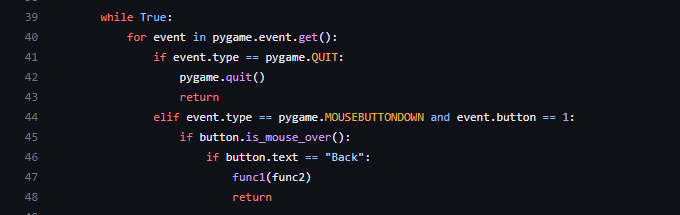


**User IO, file IO, and input validation.**

Our program uses user IO to add game controls and file IO to load credits. You can find file IO being used in endcredits.py on line 64.

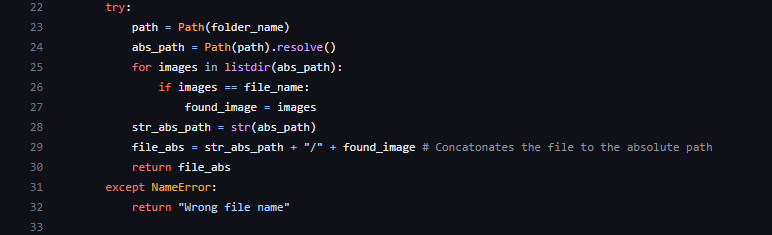
****

**Recursion**

We used recursion to help the user navigate the Main Menu; this example is in encyclopedia\_funcs.py on line 47.

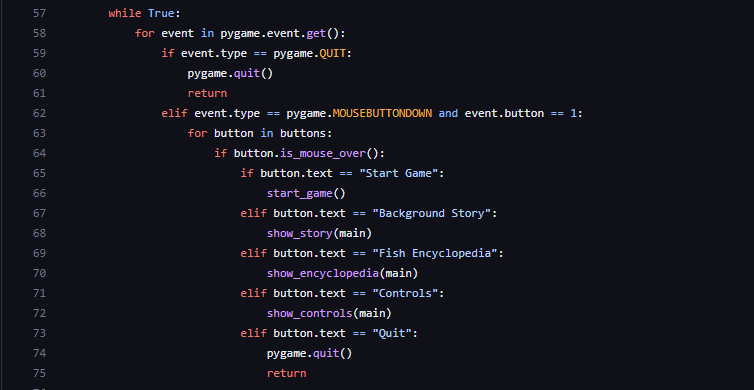
**Exceptions**

We used a try and except block to catch NameError in functions.py on line 22.



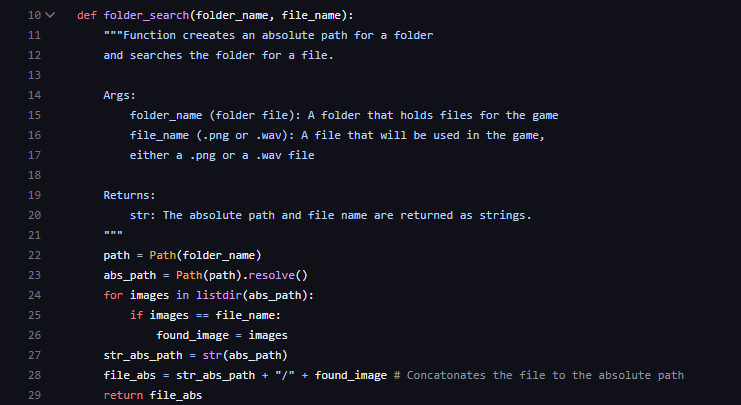
**GUI components and event-driven programming**

In our program, we utilized buttons on our main menu; when each button is clicked, it will take you to a new screen. You can see how the buttons are used in Final\_main.py on line 57.



**Documentation (docstrings, commenting, and maybe a README file)**

Our program includes docstrings, commenting, and a README file. You can find an example of docstrings and comments in functions.py starting on line 10.



5. What problems did you encounter, and how did you tackle the problems?

While working on this game, little issues constantly popped up as we added more and more features. One big project to tackle was to take all of the images we were using and make a function so that they were uploaded from a path rather than needing to be in the same folder and being called directly. This helped with the resilience of the code. After lots of researching, testing, and several hours we were able to write a code that is used for almost every .png and .wav file in the game. We had a similar time-sink when using .blit(). We were able to come together as a team to solve the issue. Another issue that we encountered was with creating the bar for casting. Because of the way that pygame rectangles work, it would move every time it got longer, so we needed to offset it by adding a way to move it up every time it grew to create the illusion that it was staying in place and growing.

6. References to resources you used for this project. For example, if you used the techniques you learned from a website or a video, include the title and the link to that resource.

[HOW TO MAKE A MENU SCREEN IN PYGAME! By BaralTech](https://www.youtube.com/watch?v=GMBqjxcKogA)

[Pygame Tutorial - 15 - Adding Text and Displaying Score By BuildWithPython](https://www.youtube.com/watch?v=Fp1dudhdX8k)

[Pygame in 90 Minutes - For Beginners By Tech With Tim](https://www.youtube.com/watch?v=jO6qQDNa2UY&t=1391s)

[Pygame Documentation](https://www.pygame.org/docs/ref/key.html)