Ryan Dwight:

My task for this part of the project was to add a timed user input system for our project. This is a key part of the game as there needs to be a time limit for the user to try and shoot the duck before it moves. As it currently stands, the code I added will repeatedly prompt the user for input and tell them they were too slow if they don't respond in time before prompting them for input again. The only way to end the cycle currently is to input the correct location which is required as a parameter when calling the play() method. In order to make these methods I had to learn how to use Event() and Thread() from threading. I learned that an Event object is simply a way of communicating between threads that something has happened which will prompt the thread to do something. As such, the methods have a cancel event which will kill the work in progress thread if the user either inputs the correct response or the timer runs out. The methods can be found in the RDwight_input.py file in our repository, which includes the import statements which download Event and Thread from the threading module and sleep from the time module. However, due to issues with github causing time constraints there currently is not a __main__ method to run the code from vscode. The best way to run the code is to copy all of it into a Juypter lab notebook and run it from there by calling play(). The user must have the location of the "duck" and time limit as arguments when calling play, for example play("A3", 4) would require the user to input A3 within 4 seconds. No specific account access or passwords are needed to run the code.

Farhan W. Quader:

I worked on developing the prototype grid for the project. In this grid, our program will have ducks set. The grid will change so that it will show hits and misses in the future. As of now, I have set up the grid as a list. It is a lot easier to set and find coordinates with a list. In the future, I hope to update the grid to be viewable on a GUI using such packages like *tkinter*. At the moment, I set up an empty board list. I used a for loop to append the list into a square-like grid using `**board.append("_|_|" * 10)**`. I ran the loop 10 times so it would create a 10x10 grid. I then made a `**print_board(board)**` function to print out the rows of the 10x10 grid so users can see how the grid appears at the current state.

Hunter Wright

The problem I worked on solving was making a "duck" appear randomly within the grid. The first thing I figured out was how to include a unicode character within my code. I found a potential unicode combination for a duck emoji, but it didn't seem to work within Python. However, I did manage to find a unicode combination that represents a soccer ball, so that will be the "duck" in the current state of our game. I also learned how to use the randrange() method from the random module. I used this to give the "duck" a random x and y coordinate throughout the grid. Randrange() requires a stop parameter, which is 10, given the size of the grid we are planning to use. I also learned how to use the pprint module. I learned that this module makes prettier prints of different Python features, including lists. I used this module to print the lists I created in the form of a grid. My

solution is contained within the “random_duck” file in the repository and the name of my function is rand_duck(). In order to run the code, you must import random and pprint first. Rand_duck() does not take any arguments at the moment. I included a main() function and “if __name__ == “__main__” statement to make running the code easier. To run it, just type “**python random_duck”** on the VS code command line.

Hunter Riportella

I worked on creating a scoreboard for our user. I used my knowledge of classes to create a scoreboard object that takes a players name and calculates their score based on the amount of times they hit the duck. To get this hit count, I made a skeleton class that simulates the action of shooting that would be in the user input area in Ryans code, at the time of writing the scoreboard class, I did not have a user input so I added the shooting class. This derives a hits count using a while loop and is based on the amount of times the user hits the duck and takes into account the amount of ammo the user has per round. This method returns hits which are then used to calculate score in the scoreboard class. The hits are then multiplied by the arbitrary aesthetically pleasing number that is 100, and this is the value for the user's score. The class then has a main function which prints a formatted string of the user's name and the score they have accumulated. This function is pretty basic and clear but it will incorporate a lot once it is fully integrated with the rest of the code. You can find this in the Scoring.py file on our team Github repository.