Battleship project plan – Short 10

First of all, well need a board class, obviously

I plan on using a private two dimensional array using [x][y] coordinates (as opposed to [y][x]) to represent the shape of it. This can be created using list comprehension based on the input variable:

```
board_array = [['.' for i in range(size)] for j in range(size)]
```

This will allow for a lot less confusion when trying to give inputs and outputs. This will all be done in the __init__ function as well as initializing an empty private array of ships. As stated in the spec, an assertion to test positivity of the size variable will be included.

As many of the board functions require a ship, I will describe the ship class first then come back to the board.

Ships will have a private string "name", a private int tuple array "shape" as well as a private int array "hit" of equal length to that of "shape" to keep track of hits. A "position" variable will also be added as None then changed when the board's "add_ship" method is called

For the print function, the program will loop through the "hit" array, if there is a True it will print an "*" otherwise it will print the first character in "name." There will have to be an assertion or some other check to make sure that "name" is at least 1 character long. It will then print

```
' ' * (10 - len(name)) + name
```

to finish the line.

For is_sunk I will simply return `all(hit)` as if all the spaces in the ship have been hit, the ship is sunk

With rotate I get to break out some knowledge from my high school linear algebra. I learned that dotting any 2d vector by the matrix [[0, 1], [-1, 0]] you get the vectors position after a 90 degree rotation. I will need to write a separate function to handle dot products without any imports. This function can be called on each position in the array for "amount" number of times then update the variable to the new array.

Now that the ship class is defined, I'll return to the board class.

The add_ship method will set the given ships position to the one given and add the ship to the private array of ships. Also, in this function will be an assertion to make sure the new ship does not overlap with the bounds or any other ship already in the array.

The board.print method will make be done in multiple different parts:

1. The top bound of the board must be printed. This can be done with the following equation:
    a. ' ' * (int(math.log10(size))+2) + '+' + '-' * (size*2 + 1) + '+'
    b. The funky stuff at the start is to account for the varying lengths of sizes greater than 9

2. For each row of the grid the number can be shown by formatting the int to a string with padding equal to the log base 10 of the size + 1. The values for each of the tiles will be determined by the private 2d array. Prior to printing I will loop through each of the ships on the board and append add the appropriate characters to the 2d array. This will be done with some “get_positions” method in the ships
3. Another horizontal boarder will be added at the bottom with the same equation
4. The next lines are determined by the length of the size (int(math.log10(size))+1) and then the column number // the row printed will determine the shown value at each column

has_been_used will assert 0 <= position[0] < size and 0 <= position[1] < size as well as check the board array at the given position and return True if it is a ‘o’ ‘X’ or ‘*’ and return False otherwise

attempt_move will assert 0 <= position[0] < size and 0 <= position[1] < size as well as the spot returning False from has_been_used