Board class properties:

- -board size
- -ship, ship size
  - possible interaction point, type of ship and ship size combine to create pos of ship on board
  - add positions taken up by ship to a property in board class. since there will be only one unique point per ship, we can define this unique point and use it. store in seperate private attribute where hits happen, add a ship.hit value.
    - side note: while printing to map, ship collisions take priority over letters, and sunk ships take priority over letters and hits.
    - This means we store the position of the hits in the board class, and the number of hits per ship in the ship class.
  - 2D array containing all hit points in [x][y] format, like in sudoku_helper.py. We can recycle some of the iter code, etc.
  - Therefore, we can make a 2D list that is the same size and width as the size given and fill it with .s. Add *s where hits have happened. If a ship sinks, it calls for the position of the ship (which we can store in a dic. With either pos or name (probably name)) and Xs out the entire ship in the list
  - This means that at any given time, we have the entire board map laid out in memory. When we want to print it, we just go line by line, replacing any unwanted characters.
  - We don't actually need the ship to know exactly where it's been hit. if the amount of ship.hits is equal to the size of the ship, sink the ship by replacing all the bits in the array with Xs and updating ship.sunk to True.
- -attempt.move is made easier with this.
- Add_ship() will take the ship shape data, add the point you want to add it to to each point in the shape, make sure it is not overlapping or colliding with anything, then add a ship object to the board and add it to the board map.
- The ship is notified that it has been hit by adding to the ship.hit counter.

Ship class properties:

- -size, ship name, shape, etc.
- shape is handled by init. apparently, the center block is the origin and the rest of the ship is built of blocks surrounding it.
- -our 2d array containing hit points comes into use here. we should also keep track of the pos of each block of a ship in both the board and ship classes (or maybe only one), because that would make ship.print much easier.
- ship.rotate changes the game. we need to think about how we'll implement ship.shape that it makes this possible.
- Each ship is defined by a couple of blocks (points), and they need to be added to your origin point.
- There will be several methods- add_ship(), sink_ship() which Xs out all the values in the board for the ship, etc.
- The rotation is pretty basic- you take all your points, and for rot 1 flip the x and y coordinates and multiply all y coordinates with -1. For rot 2, flip x y coord and multiply x coordinates with -1. For rot 3, multiply all coordinates with -1.

- To place them, we take the point at which they have to be placed, add it to all the shape coordinates, and then put those values into the big ol array we have for the board map.