Intro: Hi it's me. I made this because I like having a searchable group of functions and their usage for my coding endeavors. Here is that.

name	toplevel.py:	The "main file" run debug on this for the game, or just run it.
function	main()	No inputs, used to run the game.

name	test.py	Used for testing ship placement logic. Feel free to look at it but its just for testing. Most if not all functions in here are also in pygameloop.py but better in there.
class	`class Image:` `def __init__(self, picture,x,y):`	Class that holds an image and the x,y coordinates to display it. Used to help with display functions, also defined in pygameloop.py
func	`check_ship_v_map(direction, ship_length, x, y):`	Original version of the ship check, not used anymore as doesn't account for the game loop or something, just use the new one.
func	`check_ship_v_ship(direction, ship_length, coords, other_ships: list[Ship]):`	Original version of ship v ship check, "" same as above, just use the new one.
misc	`while running:`	The whole loop runs immediately upon running the file, so do with that what you will. If you want some reference on how the code should run, check this with debug, otherwise, forget about it.

name	ship.py	Class file for Ship class, has some useful functions for hit and sunk detection, bottom level of the project, if you change something here make sure it doesn't affect the other classes.
class	`class Ship:` `def __init__(self, coords: list[tuple[int,int]], direction="none"):`	Class constructor for ship class. Takes in two arguments, a list of int tuples, defined clearly so you can't mess that up. And a direction value with a default string "none" for dummy implementation. Most functions

		wont work with dummy instances.

attributes	`self.coords = coords`
	`self._direction = direction`
	`self._size = len(self.coords)`
	`self._hitFlag = []`
	`for _ in`
	`range(len(self.coords)):`
	`    self._hitFlag.append(False)`
		Coords is a list of coords from the constructor, used to define where the ship is. Public currently. Author didn't consider privacy.
		Direction is the direction string, used in display logic
		Size is the number of coord entries. Used for display logic
		Hitflag is a list of bools for where the ship has been hit. This is unused, as board took over that job, but shot logic still uses it so it remains

func	`isHit(self, coords: tuple):`	Function determines if a certain coords hits the ship. If hit, return true, if not, return false. Notably returns false if coords has been hit before, this prevents double shots in board class logic.
func	`in_ship(self, coords: tuple):`	There was an error using isHit for ship placement, so this is a simpler version of ishit that returns true if coords are in ship coords, else false.
func	`numHits(self):`	Unused function that was asked for by GTA to determine number of times a ship has been hit. Iterates through hitlist and counts number of hit flags
func	`def get_length(self):`	Returns private var length
func	`def get_direction(self):`	Returns private var direction

name	board.py	Effectively the "player". Has a board and a ship list, and handles shooting. IMPORTANT NOTE: board contains player information about player board and player ships, battleship class takes care of shooting at opponents

class	`class Board:`
	`    def __init__(self, shipList:`
	`list[Ship]):`
		Constructor for Board class, takes in a list of ships and makes a Board

attribute	`self.shipList = shipList`	shipList is a list of the ships a Board has.

s	`for ship in self.shipList:`
	` for coord in ship.coords:`
	`self.coordsMatrix[coord[0]][coord[1]] = 2`
		coordsMatrix is a 10x10 matrix representing player board logically.
		`Coord integer meanings:`
		`    - 0 = empty, unshot`
		`    - 1 = empty, shot`
		`    - 2 = occupied, unhit`
		`    - 3 = occupied, hit`
		`    - 4 = occupied, sunk`

func	`def allSunk(self):`	Calls isSunk on all ships in shipList, if all are sunk, then allSunk returns True, else false
func	`def shoot(self, coordsTuple):`	Be shot at these coords. If ship in shipList isHit by coords, then change coordsmatrix entry to 3 or if the ship was sunk with that hit, change all values for that ship to be sunk. Return true if it was a valid shot, false otherwise (if the place has been shot before.)

name	battleship.py	Effectively the “game”. Takes two Boards and pits them against each other. Controls shooting and win logic.

Class	`class Battleship:`
	`    def __init__(self, shipList0: list[Ship], shipList1: list[Ship]):`
		Pass in two shipLists to create a battleship instance

attributes	`self.boardZero =Board(shipList0)`
	`self.boardOne = Board(shipList1)`
	`self.turn = 0`
	`self._curBoard = self.boardOne`
		boardZero is the board for player0, red
		boardOne is the board for player1, green
		Turn is player xs turn, initialized to 0, red
		curBoard is boardOne as that is the board player0, red, shoots at

func	`def takeTurn(self, coords):`	Call shoot on opponent's board at coords Check for win, if win then return current player, otherwise switch turn and return 2 to indicate no winner, return 3 if shot was invalid, allowing another shot
func	`def _switchTurn(self):`	Private method to switch current player. Called in takeTurn

name	pygameloop.py	Controls game logic, display logic, and setup.

Class	`class Image:`
	`    def __init__(self, picture,x,y):`
		HelperClass that holds an image and the x,y coordinates to display it. Used to help with display functions, also defined in test.py

Class	`class PyGameLoop:`
	`    def __init__(self):`
		Constructor for loop, doesn't take in anything

attributes	`self._scale = 2`
	`self._width= 2*singleBoardWidth+50+100`
	`self._height=1004 + 100`
	`self._offset = 4 / self._scale`
	`self._battleship =`
	`self._display_info`
	`self._margin = 200 / self._scale`
	`self._screen`
		• scale is a scaling factor for the window as the board is too big for most screens
		• width is the imagine width of the screen, should be 2158 unless board texture changes
		• height is height of board given board size, should be 1104 unless texture is changed
		• offset is 4, which is how many pixel off the corner of the board the texture should start
		• battleship is the battleship instance, dummy at the start, is updated after shiplists are made
		• display_info is debug stuff, wanted to make better window scaling but oh well
		• Margin is a margin at the bottom of the board for text notifications
		• Screen is the window size,which is width plus margin divided by scale by height plus margin divided by scale.

func	`def check_ship_v_map(self, direction, ship_length, coords):`	Checks if ship goes off edge of map given a ship length, direction and coords. Returns true if ship DOES NOT go off the edge
func	`def check_ship_v_ship(self, direction, ship_length, coords, other_ships: list[Ship]):`	Checks if ship intersects other ships given direction, ship length, coords, and list of ships to check against Returns true if ship DOES NOT intersect another ship
func	`def _coordsToPix(self, boardNum, coords):`	Convert a coords value to a pix tuple, for usage in display logic. Takes in a board num to determine left (0) or right(0) board
func	`def _pixToCoords(self, pix`	Inverse of coords to pix. Takes pix and converts

	`:tuple, board):`	the coords on the given board num.
func	`def _drawShips(self):`	Draws all ships on the board. Internally determines which ships to draw given the turn of the battleship attribute.
func	`def _drawShots(self):`	Draws all the shots from both boards when called. (checks turn but doesn't need to. Fix that if you want)
func	`def _placeShips(self, turn, num_ships):`	Called to place ships for player(turn). Internal logic does board display. Returns a shipList for use in construction
func	`def run(self):`	The holy grail, the game logic.

	EXTRA DETAILS ON RUN	
section	Local Variables	This sets the values of needed local variables. Mainly, the images for the game, the gamephase, the passinscreen bool, the winner val, the number of ships, the max_y for display
section	Game loop	While running 1. Update screen to be white 2. Check if in passing screen currently a. If so, display the proper passing screen b. If a button is pressed or the mouse is pressed end the passing screen 3. Else, determine game phase a. Gamephase 0 i. Display welcome screen, get input for number of ships to play, once gained, switch to gamephase 1 b. Gamephase 1 i. Run placeships for player 0 (red) ii. Once placed, update game phase and start passing screen c. Gamephase 2 i. Run placeships for player 1 (green) ii. Once placed, update

		battleship with new ship lists, update gamephase to 3, start passing screen d. Gamephase 3 (Shooting) i. Draw ships and shots based on turn ii. Get events from the window iii. If the mouse has been clicked somewhere, determine if its a valid shot 1. If yes, do shot and switch turn 2. If no, let them shoot again iv. Upon shooting, if shot wins game, update to gamephase 4 e. Gamephase 4 (Win screen) i. Display the winner, whoever that is, wait for button press ii. On button press, go back to beginning (gamephase to 0, winner to 2)