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	main.py:	The "main file" run debug on this for the game, or just run it.
function	none	Code runs from file, no functions to call. Artifact from previous team

name	tile.py	File to hold the Tile class, used to store locations and symbols on a board

class	`class Tile:`
	`  def __init__(self, row, col):`
	`  self.row = row`
	`  self.col = col`
	`  self.symbol = " "`
		Class that holds row, column and symbol for each entry in the board, used mainly for display with the symbol detail.

func	`def is_sunk(self):` `  return False`	Superclass function for checking if tile is sunk, overloaded in ship.py
func	`def str(self):` `  return self.symbol`	Return tiles symbol for print statements
func	`def __repr__(self):` `  return self.symbol`	Returns the tiles symbol for debugging and console display

name	ship.py	Class file for Ship class, holds data about a ship. Mostly the attributes are called directly, other than sink detection
class	`class Ship:` `  def __init__(self, name, size, symbol):`	Class constructor for ship class. Takes in three arguments, string name, string size (ex "1x3"), and string symbol.

attributes	`self.name = name`
	`self.size = size`
	`self.symbol = symbol`
		Name is the name of the ship
		Size is a string representation of the size (ex 1x3)
		Symbol represents how a ship is displayed
		Coords is the list of coords the ship is on,

	`self.coords = []`
	`self.tiles = []`
	`self.hidden = False`
	`self.hp = int(self.size[-1])`
		defined in a different function
		Tiles is a list of Tiles that the ship occupies
		Hidden keeps track of if the ship is hidden, for display logic
		Hp keeps track of num of unhit ship parts

func	`def is_sunk(self):` `  return self.hp == 0`	Determines if ship is sunk based on HP
func	`def sink(self):`	Convert all tiles to S to indicate sunk ship
func	`def __str__(self):` `def __repr__(self):`	Used for print and display functions

name	game_object.py	File to hold GameObject, a superclass off game and player. Holds useful functions for both classes.
class	`class GameObject:` `  def __init__(self):`	Constructor of GameObject, needs no input.

attributes	`self.ship_size_to_name`
	`self.ship_size_to_symbol`
	`self.letter_to_col_index`
	`self.col_index_to_letter`
	`self.hit_syns`
	`self.miss_syns`
	`self.sink_syns`
	`self.win_syns`
	`self.syns`
		Attributes are dictionaries for conversions
		Names of attributes are self explanatory
		*_syns are lists of some strings indicating * where * is hit, miss, sink, or win. “Hit!” and “Miss!” for example.
		syns is a dictionary to map strings to *_syns. ex M representing miss_syns

func	`def br(self, char="=", gap=0):`	Print break line of + by default. If char is H M S or W then print a banner of strings in a mapped synonym
func	`def synonymizer_inator(self, char):`	Returns a random synonym for the given char from the syns lists.
func	`def valid_coord(self, coord):`	Function to determine whether coords are valid. Tests for proper formatting, then tests if coords are in map. Returns a bool, true if valid, false if not

func	`def coords_are_inbounds(self, coords):`	Check if a list of coords are all in the bounds of the board (For ship placement) Returns a bool
func	`def coord_translator(self, coord):`	Convert coord of form char int to int int. ex(b1 to (2,1))
func	`def print_board(self, board):`	Print a board, which a list of list of tiles. Checks if tile is a ship then displays ship symbol. Prints rows of chars with some spacing logic
func	`def quit(self, input):`	Check if input mean to quit, namely q,exit,quit or their capitalized versions. Exit program if it is.
func	`def get_input(self, message):`	Get input from user then check for exit then return the input.

name	player.py	Represents a player. Inherits from GameObject
Class	`class Player(GameObject):` `  def __init__(self, id, active):`	Pass in id to identify player, and active to determine which player is active

attributes	`super().__init__()`
	`self.id = id`
	`self.active = active`
	`self.board = []`
	`self.__build_board_of_tiles(self.board)`
	`self.attacked_coords = []`
	`self.ship_list = []`
	`self.super_shot = True`
	`self.opp = None`
	`self.opps_board = []`
	`self.__build_board_of_tiles(self.opps_board)`
		Super runs GameObject constructor, giving those dicts from above
		Id identifies player in some game logic
		Active determines which player is running
		Board is going to be the list of lists of tiles
		Build the board
		Attacked_coords is a list of all coords player has attacked
		Shiplist is a list of ships
		Supershot is a identifier determining if player can use supershot
		Opp is a player **or AI** that self is playing against
		Opps_board is a representation of the opponents board
		Building opponents board

func	`def set_opponent(self, opponent):`	Set opp to opponent, assumes good input Like most of this code.

func	`def __build_board_of_tiles(self, board):`	Fills out a given list with 10 nested lists of 10 Tile items
func	`def set_ship_list(self, num_ships):`	Add the proper number of properly sized ships to self.ship_list. Ship(name, size, symbol)
func	`def selected_ship(self):`	Return the ship that is currently selected, AKA at the front of self.ship_list
func	`def selected_ship_length(self):`	Return the length of the selected ship. (e.g. The length of a 1x3 ship is 3)
func	`def _clear_selected_ship_from_board(self):`	Restore the initial ship placement to be a Tile.
func	`def hide_ships(self):`	Loop until all ships are hidden. Gathers the root coord for the selected ship to be oriented from. Calls private function
func	`def __hide_ship(self, row, col):`	Hide the selected ship
func	`def __orient_ship(self, row, col):`	Generate coords for ship based on direction of ship and starting coords row,col
func	`def __select_direction(self):`	Asks player for direction u d l or r
func	`def hide_selected_ship_in_valid_location(self):`	Verify if the selected ship's list of coordinates are all on the board and vacant Tiles
func	`def direction_to_coord(self, direction, row, col, i):`	Translate up, down, left, or right in relation to a coordinate into the new coordinate corresponding with the direction.
func	`def attack_ship(self, coord, super_shot):`	**New function for supershot implementation converts coords to row,col, and either supershoots or does logic to hit or miss at row, col**
func	`def hit(self, row, col):`	Hit at row,col and update boards to represent that for you and your opponent
func	`def super_shoot(self, row, col):`	Perform shots in a 3x3 area centered on row,col. If shot is off board ignore it.

func	`def miss(self, row, col):`	Mark miss at row col on each board.
func	`def mark_shot(self, board, row, col, result):`	Mark board at row,col with symbol result.
func	`def print_shot_result(self, result):`	Print a banner of the result of a shot, hit, miss, sink.
func	`def update_board(self, sinking_player, other_player, row, col):`	Update boards when a ship is sunk, so display is S instead of H or ship marker.
func	`def update_sunk_ships(self, player, opponent, row, col):`	Update ship_list of players when ships are sunk
func	`def print_ship_list(self):`	Method to print the list of ships for the player
func	`def print_remaining_ships_to_hide(self):`	Method to print the remaining ships that need to be hidden during ship placement.

name	game.py	Controls game logic, display logic, and setup.
Class	`class Game(GameObject):` `def __init__(self, player_bank):`	Created from a list of players.

attributes	`super().__init__()`
	`self.player_bank = player_bank`
	`self.player1 = player_bank[0]`
	`self.player2 = player_bank[1]`
	`self.turn_count = 1`
	`self.active_player = self.get_active_player()`
		• Player_bank is a list of players in the game. Only 2 players
		• Player 1 is first player in bank
		• Player 2 is second player in bank
		• Turn_count is number of turns played so far, for display purposes
		• Active_player is the current player doing ship shooting.
		• Num_ships is the number of ships players get each

	`self.num_ships = 0`	
func	`def __switch_turns(self):`	Switches active player to other player
func	`def start(self):`	Initialize the game, get num ships, make ship lists for players, setup boards, regen ship lists to update based on boards, and start take_turn recursion.
func	`def valid_input(self, input):`	**Helper function to determine good input without ending turn as other logic doesn't catch all cases. New for project 2**
func	`def __take_turn(self, turn_count):`	Does most of the logic. If a player is playing, not ai, print boards, ask for shot, take shot if valid, do shot logic, print updated boards after shot, switch players, do next turn.
func	`def get_active_player(self):`	Return active player in player_bank
func	`def __setup_boards(self):`	Method to set up the boards for both players
func	`def __get_num_ships(self):`	Method to get the number of ships from the user
func	`def __set_ship_lists(self):`	Run set_ship_list for all players to create ships based on num_ships
func	`def end_game(self):`	Once a player wins, do some display logic, display winner, and exit.

name	ai.py	Ai class for upgrade Does the same stuff as a player, but automatically.
Class	`class AI(Player):` `  def __init__(self, difficulty, id, active):`	Created from a difficultly setting, an id, and an active bool.
attributes	`super().__init__(id, active)` `self.difficulty =`	Call super constructor of PLayer to do most player logic Set difficultly to given difficulty self._moveStack is a stack to hold moves for ai

	`difficulty`
	`self._moveStack = []`
	`self._shotCoords = []`
	`self._oppShipCoords = []`
		logic
		_shotCoords is a list of all shots taken by ai
		_oppShipCoords is a list of all the coords of opp's ships for hard difficult logic.

func	`def locate_opp_ships(self):`	Update _oppShipCoords with actual coords once place by player.
func	`def hide_ships(self):`	Overridden hide_ships from the player class to do the placement logic randomly.
func	`def __hide_ship(self, row, col):`	Overridden __hide_ship to suppress the printing of ai's board.
func	`def __orient_ship(self, row, col):`	Overridden to allow for random placement from the ai.
func	`def get_input(self, message):`	Overridden to use ai_turn instead of user input
func	`def update_sunk_ships(self, player, opponent, row, col):`	Overrides player version to clear moveStack as well as original logic.
func	`def aiTurn(self):`	The Ai turn logic
details		Extra details about aiturn logic ● Checks difficulty with if difficulty == "easy" medium hard, ● Easy logic ○ Take a random coord and shoot it ● Medium logic ○ Check if something was hit last turn ■ If something hit then shoot around that spot until ship is sunk ■ If not, shoot random coord, like easy. ● Hard logic ○ Take opponents ship coords and pop each turn (Effectively runs through hit logic without the extra

		func calls)
func	`def _aiHit(self, coord):`	Returns true if hit, false if miss