Battleship - Documentation

Ship Class (Ship.py):

First of all, need to create a public type Ship class which will be responsible for handling the data members i.e. Ship_name (Private String type), Ship_shape (Private Integer type list). This class holds the information of one ship. We will be creating the Getters/Setters to access the private data members of the class.

In the second step, we will be writing the methods of the class. The methods include Ship_Init (Self, Ship_name, shape) creating/naming a ship, Print_Ship (Self) for printing ship's information, Pos_Init (Self) responsible for allocating the dimension to the ship (creates a new Pos object), Ship_Sunk (Self) checks ifa ship has sunk on a hit or is still on the board. This method either returns true or false. Ship_Rotate (self, rotate) which will take a "rotate" as an argument & rotate the ship as per given argument i.e. Rotated by units of 90 degrees. If rot is 0, return the original object (itself, or a duplicate). If rot is 1, return the same coordinates, but rotated clockwise by 90 degrees; if rot is 2, rotate by 180 degrees; if rot is 3, rotate by 270 degrees. The Ship_Rotate (Self, rotate) method will return a new/updated Pos object.

There will some helping methods used which will be responsible to keep the track of which ship has been hit or miss, which part of ship has been hit (Front, end or middle), a method to return a Char "*" for hit & "name" for miss, method to keep the track of number of hits/miss & a method to keep the shape 9 spaces long which returns either true or false.

The technique going to be using for rotation 90 degree in clockwise direction is taking a matrix transpose (converting rows into columns & columns into rows) performing operation for 1 time in clockwise direction, for rotation of 180 degree we will repeat the above step one more time (2 times in total) & for rotation of 270 degree we will do take transpose of a matrix 3 times in a clockwise direction.

The ships will be represented in various shapes i.e. Horizontal, Vertical & "tee" type. The helping function Shape_Ship (Shape) will be responsible to return a shape of the ship. The shapes will be creating using the coordinates implemented in the Shape data member of Ship class.

Board Class (Board.py):

The board class has data member i.e. Board (Public 2-D arrays) & a Single Ship object. Since we are working on only one board so we don't need to create a list of board. In case, if we want to have multiple boards then we will be creating (List of 2-D arrays). We need to have multiple ships on a board so, we are going to call Ship_init (Self, Ship_name, Pos) method every time when we want to have a new ship on the board. Hence, Ship object has been composed in a board Class. We are using composition because If a ship destroys it won't be affecting (destroying) the whole board but if a board destroys, all ships would be destroying too. Since, Ship object has been composed into the Board Class. The constant board Size variable will be created to handle the size of a board. Since, the board is square so we need only one user input i.e. 9x9, 3x3 e.t.c

The Board_Print () will print the whole board on the screen. On updating the board, a screen will get clear & an updated board will be displayed. The clear () helping method will be responsible to clear the screen & display an updated board. The Board_Init (Self, Size) will be responsible for creating & initializing a board. This method takes a board object & size of a board as a parameter. The Board_Add_Ship (Self, Ship_name, position) method will be calling a method Ship_init (Self, Ship_name, Pos) from Ship Class using an object of Ship class in the board class & add the ship into the board. While adding the ship into the board a helping method Location_Reserved (Pos) to check if location is already being reserved by another ship or is empty. This method returns either true or false on taking position an argument. The helping methods i.e. dots () will return dot characters for empty spaces on the board, Hit_target () will return an asterisk (*) to record a place on a ship which has been Hit (but not where the ship has not been sunk), Sunk () method will return 'X' character on a location (Coordinates) where the ship has been sunk & miss_target () will return 'o' on the coordinates where the target was miss. The method Board_has_been_used (Self, Pos) will keep the records of targeted locations (hit/miss). The method Board_attempt_move (Self, Pos) will be used to hit the targets on the board. The method assert would be helping in knowing that if the coordinates are within the range or available on the board.

The method Board_attempt_move (Self, Pos) will be called in a helping function named Shots (Board). This method will take inputs (Positions to hit) from the user on command line. A loop will be used to take multiple shots per user.

Battleship Class (Battleship.py):

The Battleship.py will be responsible for declaring & initializing the Board class object & user input size of the board. After initializing, the default constructor will automatically be called. First of all, Board_Init (Self, Size) would be called then the helping method Shots (Board, Size) will be called through the reference of Board class object.

____