Contents

1 Basic Test Results

```
Starting tests...
Wed Dec 16 19:00:14 IST 2015
8ca2fb8cdbb5926ceda392f6619738bce84e44fa  -


Archive:  /tmp/bodek.m4tG4o/intro2cs/ex8/elinorperl/presubmission/submission
  inflating: src/game.py
  inflating: src/README
  inflating: src/ship.py

Testing README...
Done testing README...

Running presubmit tests...
3 passed tests out of 3
result_code    game    3    1
57 passed tests out of 57
result_code    ship    57    1
Done running presubmit tests

Tests completed

Additional notes:

There will be additional tests which will not be published in advance.

Game.py is not tested very much.
```

2 README

```
elinorperl
329577464
Elinor Perl

I discussed the exercise with Talya Adams, Bnaya Pelet, Nophar Sarel

=========================================
=  README for ex8: Battleship OOP  =
=========================================


==================
=  Description:  =
==================

In this exercise, using OOP, I defined a class "Ship" creating different critera
to how I want my ship to function. It defines its movements according to the game
rules, becomes stationary once one of its cells is hit by a bomb, terminating the ship
once all the cells are hit. The class includes other attributes in accordance to its need.
The other main class defined is "Game", in which I defined how the game would play out (by
round and then continuing until there were no ships left).

======================
=  Special Comments  =
======================

I used stackoverflow.com
```

3 game.py

```python
############################################################
# Imports
############################################################
import game_helper as gh
import ship
import copy
#####################################
# Class definition
############################################################

class Game:
    """
    A class representing a battleship game.
    A game is composed of ships that are moving on a square board and a user
    which tries to guess the locations of the ships by guessing their
    coordinates.
    """

    BOMB_TIME = 3
    GAME_STATUS_ONGOING = "GAME"
    GAME_STATUS_ENDED = "END"


    def __init__(self, board_size, ships):
        """
        Initialize a new Game object.
        :param board_size: Length of the side of the game-board
        :param ships: A list of ships that participate in the game.
        :return: A new Game object.
        """
        self.__board_size = board_size
        self.__ships = ships
        self.game_bombs = {}
        self.__game_status = self.GAME_STATUS_ONGOING
        self.undamaged_ships = []
        for ship in self.__ships:
            self.undamaged_ships += ship.coordinates()


    def __play_one_round(self):
        """
        Note - this function is here to guide you and it is *not mandatory*
        to implement it. The logic defined by this function must be implemented
        but if you wish to do so in another function (or some other functions)
        it is ok.

        Te function runs one round of the game :
            1. Get user coordinate choice for bombing.
            2. Move all game's ships.
            3. Update all ships and bombs.
            4. Report to the user the result of current round (number of hits and
             terminated ships)
        :return:
            (some constant you may want implement which represents) Game status :
            GAME_STATUS_ONGOING if there are still ships on the board or
            GAME_STATUS_ENDED otherwise.
        """
        current_bombs = {}
        ship_bombs = []
        ships_damaged_cells = []
        current_ships = []
        undamaged_cells = []
        terminated_ships = 0
        bomb_pos = gh.get_target(self.__board_size) # Gets the bomb position
        self.game_bombs[bomb_pos] = Game.BOMB_TIME

        for ship in self.__ships:
            ship.move()
            for bomb in self.game_bombs:
                if ship.hit(bomb): # If any of the bombs hit the ship
                    ship_bombs.append(bomb) # it updates a list of bombs
                ships_damaged_cells += ship.damaged_cells_list
                # creates a list of the undamaged cells
            undamaged_cells += [cell for cell in ship.coordinates() \
                                if cell not in ship.damaged_cells_list]


        for ship in self.__ships:
            if ship.terminated():
                terminated_ships += 1
            else:
                # Created a new list of ships that leave out the terminated
                current_ships.append(ship)
        self.__ships = current_ships


        for key in self.game_bombs:
            if key != bomb_pos:
                self.game_bombs[key] -= 1
            if self.game_bombs[key] != 0 and key not in ship_bombs:
                # Created a new dictionary with the bombs that are still in
                # motion
                current_bombs[key] = self.game_bombs[key]
        self.game_bombs = current_bombs
        print(gh.board_to_string(self.__board_size, ship_bombs, self.game_bombs,
                                 ships_damaged_cells, undamaged_cells))

        if len(self.__ships) == 0:
            self.__game_status = self.GAME_STATUS_ENDED
        gh.report_turn(len(ship_bombs), terminated_ships)
        return copy.copy(self.__game_status)

    def __repr__(self):
        """
        Return a string representation of the board's game
        :return: A tuple converted to string. The tuple should contain (maintain
        the following order):
            1. Board's size.
            2. A dictionary of the bombs found on the board
                 {(pos_x, pos_y) : remaining turns}
                For example :
                 {(0, 1) : 2, (3, 2) : 1}
            3. A list of the ships found on the board (each ship should be
                represented by its __repr__ string).
        """
        game_description = (self.__board_size, self.game_bombs, self.__ships)
        return str(game_description)

    def play(self):
        """
        The main driver of the Game. Manages the game until completion.
        completion.
        :return: None
        """
        gh.report_legend()
        print(gh.board_to_string(self.__board_size, [], self.game_bombs, [],
                                 self.undamaged_ships))
        while self.__game_status == self.GAME_STATUS_ONGOING:
            self.__play_one_round()
        gh.report_gameover()


############################################################
# An example usage of the game
############################################################
if __name__== "__main__":
    game = Game(5, gh.initialize_ship_list(4, 2))
    game.play()
```

4 ship.py

```python
############################################################
# Helper class
############################################################

import ship_helper
import copy

class Direction:
    """
    Class representing a direction in 2D world.
    You may not change the name of any of the constants (UP, DOWN, LEFT, RIGHT,
     NOT_MOVING, VERTICAL, HORIZONTAL, ALL_DIRECTIONS), but all other
     implementations are for you to carry out.
    """
    UP = "UP"
    DOWN = "DOWN"
    LEFT = "LEFT"
    RIGHT = "RIGHT"

    NOT_MOVING = "NOT MOVING"

    VERTICAL = (UP, DOWN)
    HORIZONTAL = (LEFT, RIGHT)

    ALL_DIRECTIONS = (UP, DOWN, LEFT, RIGHT)

############################################################
# Class definition
############################################################


class Ship:
    """
    A class representing a ship in Battleship game.
    A ship is 1-dimensional object that could be laid in either horizontal or
    vertical alignment. A ship sails on its vertical\horizontal axis back and
    forth until reaching the board's boarders and then changes its direction to
    the opposite (left <--> right, up <--> down).
    If a ship is hit in one of its coordinates, it ceases its movement in all
    future turns.
    A ship that had all her coordinates hit is considered terminated.
    """

    def __init__(self, pos, length, direction, board_size):
        """
        A constructor for a Ship object
        :param pos: A tuple representing The ship's head's (x, y) position
        :param length: Ship's length
        :param direction: Initial direction in which the ship is sailing
        :param board_size: Board size in which the ship is sailing
        """
        self.__pos = pos
        self.__length = length
        self.__direction = direction
        self.__board_size = board_size
        self.damaged_cells_list = []
        self.INITIAL_DIRECTION = direction


    def __repr__(self):
        """
        Return a string representation of the ship.
        :return: A tuple converted to string. The tuple's content should be (in
        the exact following order):
            1. A list of all the ship's coordinates.
            2. A list of all the ship's hit coordinates.
            3. Last sailing direction.
            4. The size of the board in which the ship is located.
        """
        ship_description = (self.coordinates(), self.damaged_cells_list,
                            ship_helper.direction_repr_str(Direction,
                                                           self.__direction),
                            self.__board_size)
        return str(ship_description)

    def move(self):
        """
        Make the ship move one board unit.
        Movement is in the current sailing direction, unless such movement would
        take it outside of the board in which case the shp switches direction
        and sails one board unit in the new direction.
        the ship
        :return: A direction object representing the current movement direction.
        """
        x, y = self.__pos
        if len(self.damaged_cells_list) != 0:
            self.__direction = Direction.NOT_MOVING
        elif self.__direction == Direction.RIGHT:
            if x + self.__length >= self.__board_size:
                x -= 1
                self.__direction = Direction.LEFT
            else:
                x += 1
        elif self.__direction == Direction.LEFT:
            if x == 0:
                x += 1
                self.__direction = Direction.RIGHT
            else:
                x -= 1
        elif self.__direction == Direction.UP:
            if y == 0:
                y += 1
                self.__direction = Direction.DOWN
            else:
                y -= 1
        elif self.__direction == Direction.DOWN:
            if y + self.__length >= self.__board_size:
                y -= 1
                self.__direction = Direction.UP
            else:
                y += 1
        self.__pos = (x, y)
        return copy.copy(self.__direction)


    def hit(self, pos):
        """
        Inform the ship that a bomb hit a specific coordinate. The ship updates
         its state accordingly.
        If one of the ship's body's coordinate is hit, the ship does not move
         in future turns. If all ship's body's coordinate are hit, the ship is
         terminated and removed from the board.
        :param pos: A tuple representing the (x, y) position of the hit.
        :return: True if the bomb generated a new hit in the ship, False
         otherwise.
        """
        for coord in self.coordinates():
            if coord not in self.damaged_cells_list:
                if coord == pos:
                    self.damaged_cells_list.append(pos)
                    self.__direction = Direction.NOT_MOVING
                    return True
        return False


    def terminated(self):
        """
        :return: True if all ship's coordinates were hit in previous turns, False
        otherwise.
        """
        if len(self.damaged_cells_list) == self.__length:
            return True
        else:
            return False

    def __contains__(self, pos):
        """
        Check whether the ship is found in a specific coordinate.
        :param pos: A tuple representing the coordinate for check.
        :return: True if one of the ship's coordinates is found in the given
        (x, y) coordinates, False otherwise.
        """
        if pos in self.coordinates():
            return True
        else:
            return False

    def coordinates(self):
        """
        Return ship's current positions on board.
        :return: A list of (x, y) tuples representing the ship's current
        position.
        """
        ship_coordinates = []
        (x, y) = self.__pos
        if self.INITIAL_DIRECTION in Direction.HORIZONTAL:
            for i in range(self.__length):
                ship_coordinates.append((x + i, y))
        elif self.INITIAL_DIRECTION in Direction.VERTICAL:
            for i in range(self.__length):
                ship_coordinates.append((x, y + i))
        return ship_coordinates


    def damaged_cells(self):
        """
        Return the ship's hit positions.
        :return: A list of tuples representing the (x, y) coordinates of the
         ship which were hit in past turns (If there are no hit coordinates,
         return an empty list). There is no importance to the order of the
         values in the returned list.
        """
        damaged_list = self.damaged_cells_list[:]
        return damaged_list


    def direction(self):
        """
        Return the ship's current sailing direction.
        :return: One of the constants of Direction class :
         [UP, DOWN, LEFT, RIGHT] according to current
         sailing direction or NOT_MOVING if the ship is hit and not moving.
        """
        ship_direction = copy.copy(self.__direction)
        return ship_direction

    def cell_status(self, pos):
        """
        Return the state of the given coordinate (hit\not hit)
        :param pos: A tuple representing the coordinate to query.
        :return:
            if the given coordinate is not hit : False
            if the given coordinate is hit : True
            if the coordinate is not part of the ship’s body : None
        """
        if pos not in self.coordinates():
            return None
        elif pos in self.damaged_cells_list:
            return True
        else:
            return False
```