Contents

1 Basic Test Results

```
Starting tests...
Thu Dec 24 00:07:40 IST 2015
e87f4ea2769cbf2cfa4d7ac0d88262d827245449  -


Archive:  /tmp/bodek.m4tG4o/intro2cs/ex9/elinorperl/presubmission/submission
  inflating: src/asteroid.py
  inflating: src/asteroids_main.py
 extracting: src/AUTHORS
  inflating: src/screen.py
  inflating: src/ship.py
  inflating: src/torpedo.py
  inflating: src/README

Testing README...
Done testing README...

Listing AUTHORS...
elinorperl
ynatovich


Tests completed

Additional notes:

No presubmission tests of the code this week.
```

2 README

```
elinorperl
329577464
Elinor Perl

ynatovich
204858914
Yifat Natovitch


I discussed the exercise with Talya Adams, Yonaton Iluz

=========================================
=  README for ex9: Asteroids OOP  =
=========================================


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

In this exercise, using OOP, we defined three different classes "Ship",
"Asteroid, and "Torpedo", each playing a part in the game we created -
Asteroids. Each class was defined according to the attributes we expected
from them in the game in which the object is for the ship to avoid and destory
the asteroids, defeating them by using the torpedos which split the asteroids
into smaller asteroids until they are small enought to dissappear.

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

We used stackoverflow.com

Throughout programming this game, we encountered obstacles about how to
proceed with our game, amongst them:

1. When creating a function to move our ship, we were unsure in which
class it belonged. On the one hand, we’re moving the ship and movement
is an attribute belonging to itself. On the other, in order to display
our movement on the screen we needed the "intermediary" game which
acts as the bridge between the two classes. In the end, we decided
upon placing the ship’s movement in its own class and calling it
to our intermmediary when necessary.

2. In the class "Torpedo", we debated on whether to define a constant in the
initial creation of itself, or to give the ability to an outside user to change
it throughout the game. The advantages of the changability throughout give the
user the flexibility to use the constant.We decided against this method, because
we wanted to save the initial attributes according the game rules.

3. When ending the game, we didn’t know whether to make a one function
to end the game or to split it into two, to give it a "cleaner" look,
rather than repeating the same actions three times. The layout we ended
with is longer and maybe not necessary but we felt it was more sleek.
```

3 AUTHORS

```
elinorperl, ynatovich
```

4 asteroid.py

```python
from random import randrange, uniform
from screen import Screen as Scr
import copy
import math
from ship import Ship

class Asteroid:

    DELTA_X = Scr.SCREEN_MAX_X - Scr.SCREEN_MIN_X
    DELTA_Y = Scr.SCREEN_MAX_Y - Scr.SCREEN_MIN_Y
    RADIUS_COEFFIC = 10
    NORM_FACTOR = -5
    BIG = 3
    MIDDLE = 2
    SMALL = 1

    def __init__(self):
        self.__x_speed = uniform(-5, 5)
        self.__y_speed = uniform(-5, 5)
        self.__x_position = randrange(Scr.SCREEN_MIN_X, Scr.SCREEN_MAX_X)
        self.__y_position = randrange(Scr.SCREEN_MIN_Y, Scr.SCREEN_MAX_Y)
        self.__size = 3
        self.__radius = self.__size * self.RADIUS_COEFFIC + self.NORM_FACTOR

    def get_x_position(self):
        return copy.copy(self.__x_position)

    def get_y_position(self):
        return copy.copy(self.__y_position)

    def get_x_speed(self):
        return copy.copy(self.__x_speed)

    def get_y_speed(self):
        return copy.copy(self.__y_speed)

    def get_size(self):
        return copy.copy(self.__size)

    def change_position(self, new_position):
        """
        Changes the position of the asteroid according to the new position
        input.
        """
        self.__x_position, self.__y_position = new_position[0], new_position[1]

    def change_size(self, size):
        """
        Changes the size of the asteroid according to the size input.
        """
        self.__size = size


    def move(self):
        """
         This function calculates our new coordinate of the position for our
         asteroid.
        """
        self.__x_position = ((self.get_x_speed() + self.get_x_position() -
                    Scr.SCREEN_MIN_X) % self.DELTA_X + Scr.SCREEN_MIN_X)
        self.__y_position = ((self.get_y_speed() + self.get_y_position() -
                    Scr.SCREEN_MIN_Y) % self.DELTA_Y + Scr.SCREEN_MIN_Y)

    def get_interaction(self, obj):
        distance_x = math.pow((obj.get_x_position() - self.get_x_position()),2)
        distance_y = math.pow((obj.get_y_position() - self.get_y_position()),2)
        distance = math.sqrt(distance_x + distance_y)
        if distance <= self.__radius + obj.RADIUS:
            return True
        return False

    def speed_after_interaction(self, torpedo, direction):
        """
        A function that calculates the speed of the new asteroids that were
        created after a collision with a torpedo, where one of the parameters
        is direction, therefore allowing us to later differentiate between
        the two directions of the new asteroids. The asteroid splits into two,
        allowing us to use the indexes for 0 and 1 as our direction guide. 0
        for the positive direction and 1 for the negative.
        """
        speed_denominator = math.sqrt(math.pow(self.__x_speed,2) +\
                                      math.pow(self.__y_speed,2))
        self.__x_speed = ((torpedo.get_x_speed() + self.get_x_speed()) /
                          speed_denominator)
        self.__y_speed = ((torpedo.get_y_speed() + self.get_y_speed()) /
                          speed_denominator)
        if direction == 1:
            self.__x_speed *= -1
            self.__y_speed *= -1
```

5 asteroids main.py

```python
from screen import Screen
import sys
from ship import Ship
from asteroid import Asteroid
from torpedo import Torpedo
import copy
from random import randrange

DEFAULT_ASTEROIDS_NUM = 5


class GameRunner:

    DELTA_X = Screen.SCREEN_MAX_X - Screen.SCREEN_MIN_X
    DELTA_Y = Screen.SCREEN_MAX_Y - Screen.SCREEN_MIN_Y
    MAX_TORPEDOS = 15

    def __init__(self, asteroids_amnt=DEFAULT_ASTEROIDS_NUM):
        self._screen = Screen()
        self.screen_max_x = Screen.SCREEN_MAX_X
        self.screen_max_y = Screen.SCREEN_MAX_Y
        self.screen_min_x = Screen.SCREEN_MIN_X
        self.screen_min_y = Screen.SCREEN_MIN_Y
        self._ship = Ship()
        self._asteroids_amnt = asteroids_amnt
        self._asteroids = []
        self._torpedos = []
        self._score = 0
        self._screen.set_score(self._score)
        self.create_asteroids()

    def get_score(self):
        return copy.copy(self._score)

    def create_asteroids(self):
        """
        In this function, we created a list of the asteroids according to the
         amount give (if none was given, according to the default amount) and
         registered each one.
        """
        for i in range(self._asteroids_amnt):
            asteroid = Asteroid()
            self.get_random_position_asteroid(asteroid)
            self._asteroids.append(asteroid)
            self._screen.register_asteroid(asteroid, asteroid.get_size())

    def get_random_position_asteroid(self, asteroid):
        """
        This function places the asteroid in a random location, and if the
        asteroids location lands on the spaceship's initial location, the
        function recursively relocates the asteroid.
        """
        asteroid.change_position((randrange(Screen.SCREEN_MIN_X,
                                           Screen.SCREEN_MAX_X),\
                                 randrange(Screen.SCREEN_MIN_X,
                                           Screen.SCREEN_MAX_X)))
        while ((asteroid.get_x_position(), asteroid.get_y_position()) == \
                (self._ship.get_x_position(), self._ship.get_y_position())):
            self.get_random_position_asteroid(asteroid)

    def run(self):
        self._do_loop()
        self._screen.start_screen()

    def _do_loop(self):
        # You don't need to change this method!
        self._game_loop()

        # Set the timer to go off again
        self._screen.update()
        self._screen.ontimer(self._do_loop, 5)

    def operate_torpedo(self):
        """
        This function operates the torpedo's functionality. Calling the class
        Torpedo and inputting ship's attributes (serving as the ship and
        torpedo's bridge of connection), adding each torpedo to the list
        created and drawing it on the screen.
        """
        torpedo = Torpedo(self._ship.get_x_position(),
                                      self._ship.get_y_position(),
                                      self._ship.get_x_speed(),
                                      self._ship.get_y_speed(),
                                      self._ship.get_heading())
        self._torpedos.append(torpedo)
        self._screen.register_torpedo(torpedo)
        self._screen.draw_torpedo(torpedo, torpedo.get_x_position(),
                                  torpedo.get_y_position(),
                                  torpedo.get_direction())

    def game_movements(self):
        """
        This function checks if the user pressed left, right or up and
         depending on the input, it returns the function accordingly.
        """
        if Screen.is_left_pressed(self._screen):
            self._ship.change_heading_left()
        elif Screen.is_right_pressed(self._screen):
            self._ship.change_heading_right()
        elif Screen.is_up_pressed(self._screen):
            self._ship.accelerated_ship()
        elif Screen.is_space_pressed(self._screen):
            if len(self._torpedos) < self.MAX_TORPEDOS:
                self.operate_torpedo()

    def draw_objects(self):
        """"
        This function calls the functions draw ship and draw (each) asteroid
        on the screen.
        """
        self._screen.draw_ship(self._ship.get_x_position(),
                         self._ship.get_y_position(),
                         self._ship.get_heading())
        for asteroid in self._asteroids:
            self._screen.draw_asteroid(asteroid, asteroid.get_x_position(),\
                                 asteroid.get_y_position())
        for torpedo in self._torpedos:
            self._screen.draw_torpedo(torpedo, torpedo.get_x_position(),
                                  torpedo.get_y_position(),
                                  torpedo.get_direction())

    def move_asteroid(self):
        """
        This function moves each asteroid from the list of asteroids that we
        created, moving each asteroid in turn and having them appear on the
        screen.
        """
        for asteroid in self._asteroids:
            asteroid.move()

    def move_torpedos(self):
        """
        This function moves each asteroid from the list of asteroids that we
        created, moving each asteroid in turn and having them appear on the
        screen.
        """
        for torpedo in self._torpedos:
            torpedo.move()

    def interaction_ship(self, asteroid):
        """
        This function deals with the interaction between the ship and
        asteroids by removing one of the users lives, and displaying a message
        to the user to notify them that they hit an asteroid and thereafter
        removing them.
        """
        Screen.remove_life(self._screen)
        self._ship.update_life()
        # in the event that there are two collision in one turn:
        if self._ship.get_life() == 0:
            self.ending("Out of lives", "Maybe next time...")
        self._screen.show_message("BOOM!","You hit"
                                    " an asteroid. BE CAREFUL!")
        self._asteroids.remove(asteroid)
        self._screen.unregister_asteroid(asteroid)

    def interaction_torpedo(self, asteroid, torpedo):
        """
        This function deals with the occurence of interaction with torpedos
        and asteroids, removing the torpedo from the screen, and updating
        the user's score.
        """
        self._torpedos.remove(torpedo)
        self._screen.unregister_torpedo(torpedo)
        self.update_score(asteroid)
        self.split_asteroid(asteroid, torpedo)

    def create_new_asteroids(self, asteroid, torpedo, size):
        """
        This function splits the current asteroid to 2 new smaller ones and
        defines its attributes and position on the screen.
        """
        get_x = asteroid.get_x_position()
        get_y = asteroid.get_y_position()
        for i in range(2):
            asteroid_i = Asteroid()
            asteroid_i.change_position((get_x, get_y))
            asteroid_i.change_size(size)
            asteroid_i.speed_after_interaction(torpedo, i)
            self._asteroids.append(asteroid_i)
            self._screen.register_asteroid(asteroid_i, asteroid_i.get_size())

    def split_asteroid(self, asteroid, torpedo):
        """
        This function splits the asteroid when it gets hit. and deletes the
        asteroid when it has reached its smallest size and has been hit as
        well.
        """
        self._asteroids.remove(asteroid)
        self._screen.unregister_asteroid(asteroid)

        if asteroid.get_size() == asteroid.BIG:
            self.create_new_asteroids(asteroid, torpedo, asteroid.MIDDLE)
        if asteroid.get_size() == asteroid.MIDDLE:
            self.create_new_asteroids(asteroid, torpedo, asteroid.SMALL)

    def update_score(self, asteroid):
        """
        A function that updates the score according to the asteroid’s size.
        """
        if asteroid.get_size() == asteroid.BIG:
            self._score += 20
        if asteroid.get_size() == asteroid.MIDDLE:
            self._score += 50
        if asteroid.get_size() == asteroid.SMALL:
            self._score += 100
        self._screen.set_score(self._score)

    def interaction(self):
        """
        Using a copied version of our asteroid_list, we checked if there was a
        collision with each asteroid and the spaceship. If there was a inter-
        action, the spaceship removes a life, and displays a message that
        there has been a collision and removes an asteroid from our asteroid
        list.
        """
        copied_asteroid = copy.copy(self._asteroids)
        copied_torpedos = copy.copy(self._torpedos)
        for asteroid in copied_asteroid:
            if asteroid.get_interaction(self._ship):
                self.interaction_ship(asteroid)
        for asteroid in copied_asteroid:
            for torpedo in copied_torpedos:
                if asteroid.get_interaction(torpedo):
                    # if there is a collision, loop continues to other objects
                    if (asteroid in self._asteroids) and \
                            (torpedo in self._torpedos):
                        self.interaction_torpedo(asteroid, torpedo)

    def update_lifespan_torpedos(self):
        """
        Updates the lifespan of each torpedo on our list
        """
        copied_torpedos = copy.copy(self._torpedos)
        for torpedo in copied_torpedos:
            torpedo.update_lifespan()
            if torpedo.get_lifespan() == torpedo.LIFESPAN_MAX:
                self._torpedos.remove(torpedo)
                self._screen.unregister_torpedo(torpedo)

    def end_game(self):
        """
        Returns message according to the reason for finishing the game and
        calls to the function that displays the message and closes the tab.
        """
        if len(self._asteroids) == 0:
            self.ending("WINNER!!!", "YOU ARE THE CHAMPION MY FRIEND! YOU "
                                     "HAVE DESTRYOED ALL THE ASTEROIDS :)")
        if self._ship.get_life() == 0:
            self.ending("Out of lives", "Maybe next time...")
        if self._screen.should_end():
            self.ending("QUITTER", "Byebye!")

    def ending(self, title, msg):
        """
        This function ends the game by showing the message with the reason
        for ending, then closes the screen.
        """
        self._screen.show_message(title, msg)
        self._screen.end_game()
        sys.exit()

    def _game_loop(self):
        '''
        This is the function that ties the whole game together, calling each
        function in turn, and creating each round of the game.
        '''
        self._ship.move()
        self.move_asteroid()
        self.move_torpedos()
        self.game_movements()
        self.draw_objects()
        self.interaction()
        self.update_lifespan_torpedos()
        self.end_game()


def main(amnt):
    runner = GameRunner(amnt)
    runner.run()

if __name__ == "__main__":
    if len(sys.argv) > 1:
        main( int( sys.argv[1] ) )
    else:
        main(DEFAULT_ASTEROIDS_NUM)
```

6 screen.py

```
import sys
import tkinter
import tkinter.messagebox

from turtle import *

class ShapesMaster:
    ASTEROID_BASE_SHAPE = "asteroid%d"
    SHIP_SHAPE = "ship"
    TORPEDO_SHAPE = "torpedo"

    ASTEROIDS_TYPES = 3

    ASTEROID_3_LAYOUT = ((-20, -16),(-21, 0), (-20,18),(0,27),(17,15),
                            (25,0),(16,-15),(0,-21))

    ASTEROID_2_LAYOUT = ((-15, -10),(-16, 0), (-13,12),(0,19),(12,10),
                            (20,0),(12,-10),(0,-13))

    ASTEROID_1_LAYOUT = ((-10,-5),(-12,0),(-8,8),(0,13),(8,6),(14,0),(12,0),(8,-6),(0,-7))

    ASTEROIDS_LAYOUTS = [ASTEROID_1_LAYOUT, ASTEROID_2_LAYOUT, ASTEROID_3_LAYOUT]

    SHIP_LAYOUT = ((-10,-10),(0,-5),(10,-10),(0,10))

    TORPEDO_LAYOUT = ((-2,-4),(-2,4),(2,4),(2,-4))

    def __init__(self, screen):
        """
        This initializes the shapes controller, the screen passed is the screen
        controling the game, you should not call this method anywhere in your
        code.
        """
        self.screen = screen
        self._shapes = {}
        self._updated = False
        self._add_base_shapes()

    def add_shape(self,name,cords,override = False):
        if override or name not in self._shapes:
            self._shapes[name] = cords
            self.screen.register_shape(name,cords)

    def _add_base_shapes(self):
        for i in range(ShapesMaster.ASTEROIDS_TYPES):
            self.add_shape(ShapesMaster.ASTEROID_BASE_SHAPE%(i+1), \
                        ShapesMaster.ASTEROIDS_LAYOUTS[i])

        self.add_shape(ShapesMaster.SHIP_SHAPE, ShapesMaster.SHIP_LAYOUT)
        self.add_shape(ShapesMaster.TORPEDO_SHAPE, ShapesMaster.TORPEDO_LAYOUT)

    def get_shapes_dict(self):
        """
        Returns a dictionary of all the shapes in the game in the format of
        (name, coordinates).
        You have no reason of calling this method anywhere in your code...
        """
        return self._shapes

```

```

class Screen:

    SCREEN_MIN_X = -500
    SCREEN_MIN_Y = -500
    SCREEN_MAX_X = 500
    SCREEN_MAX_Y = 500

    def __init__(self):
        """
        This inits our graphics class.
        """

        self._boundKeys = []
        self._init_keys_values()
        self._init_graphics()
        self._bind_keys()
        self._screen.listen()

        self._ship = self._get_ship_obj(self._cv)

    def _init_keys_values(self):
        self._specialTorpedFired = 0
        self._rightClicks = 0
        self._leftClicks = 0
        self._upClicks = 0
        self._fireClicks = 0
        self._endGame = False
        self._lives = []
        self._asteroids = {}
        self._torpedos = {}

    def _init_graphics(self):
        self._root = tkinter.Tk()
        self._root.title("Asteroids!")
        self._cv = ScrolledCanvas(self._root,600,600,600,600)
        self._cv.pack(side = tkinter.LEFT)
        self._t = RawTurtle(self._cv)

        self._screen = self._t.getscreen()
        self._screen.setworldcoordinates(
                                        Screen.SCREEN_MIN_X,
                                        Screen.SCREEN_MIN_Y,
                                        Screen.SCREEN_MAX_X,
                                        Screen.SCREEN_MAX_X
                                        )
        self._shapeMaster = ShapesMaster(self._screen)
        shapes = self._shapeMaster.get_shapes_dict()

        frame = tkinter.Frame(self._root)
        frame.pack(side = tkinter.RIGHT,fill=tkinter.BOTH)

        # add scores frame
        self._score_val = tkinter.StringVar()
        self._score_val.set("0")
        scoreTitle = tkinter.Label(frame,text="Score")
        scoreTitle.pack()
        scoreFrame = tkinter.Frame(frame,height=2, bd=1, \
            relief=tkinter.SUNKEN)
        scoreFrame.pack()
        score = tkinter.Label(scoreFrame,height=2,width=20,\
            textvariable=self._score_val,fg="Yellow",bg="black")

        ################

        score.pack()

        # Add Lives Frame
        # livesTitle = tkinter.Label(frame, \
        #    text="Extra Lives Remaining")
        # livesTitle.pack()

        # livesFrame = tkinter.Frame(frame, \
        #     height=30,width=60,relief=tkinter.SUNKEN)
        # livesFrame.pack()
        # self._lives_canvas = ScrolledCanvas(livesFrame,150,40,150,40)
        # self._lives_canvas.pack()
        # livesTurtle = RawTurtle(self._lives_canvas)
        # livesTurtle.ht()
        # livesScreen = livesTurtle.getscreen()
        # livesScreen.register_shape(ShapesMaster.SHIP_SHAPE, shapes[ShapesMaster.SHIP_SHAPE])

        # Add Lives Frame
        livesTitle = tkinter.Label(frame, \
           text="Extra Lives Remaining")
        livesTitle.pack()

        livesFrame = tkinter.Frame(frame, \
            height=30,width=60,relief=tkinter.SUNKEN)
        livesFrame.pack()
        livesCanvas = ScrolledCanvas(livesFrame,150,40,150,40)
        livesCanvas.pack()
        livesTurtle = RawTurtle(livesCanvas)
        livesTurtle.ht()
        livesScreen = livesTurtle.getscreen()
        livesScreen.register_shape(ShapesMaster.SHIP_SHAPE, shapes[ShapesMaster.SHIP_SHAPE])

        life1 = self._get_ship_obj(livesCanvas) #   SpaceShip(livesCanvas,-35,0,0,0)
        life2 = self._get_ship_obj(livesCanvas) #SpaceShip(livesCanvas,0,0,0,0)
        life3 = self._get_ship_obj(livesCanvas) #SpaceShip(livesCanvas,35,0,0,0)

        self._draw_object(life1,-35,0)
        self._draw_object(life2,0,0)
        self._draw_object(life3,35,0)

        self._lives = [life1, life2, life3]

        self._t.ht()

        quitButton = tkinter.Button(frame, text = "Quit", command=self._handle_exit)
        quitButton.pack()

        self._screen.tracer(0)

    def ontimer(self, func, milli):
        """
        This method is used to create a repeating action in your game.

        .. warning::

            **You don't need to call this method, it was already called for you at the end of the main game loop.**

        :param func: The function to repeat after **milli** milliseconds have passed
        :type func: function
        :param milli: The amount of milliseconds to wait before starting the given
            function
        :type milli: int
        """
        self._screen.ontimer(func,milli)

    def _bind_key(self, key, func):
        """
        This method is to allow you to add some functionality of your own,
        it allows you to bind the provided function to the desired input key.

        If there is already a function bound to this key it will do nothing.

        :param key: A key to bind.
        :type key: str
        :param func: The function to bind
        :type func: function
        """

        if key not in self._boundKeys:
            self._screen.onkeypress(func,key)
            self._boundKeys.append(key)

    def _bind_keys(self):
        self._bind_key("Left", self._handle_left)
        self._bind_key("Right", self._handle_right)
        self._bind_key("Up", self._handle_up)
        self._bind_key("space", self._handle_space)
        self._bind_key("q", self._handle_exit)
        self._bind_key("s", self._handle_special_torpedo)

    def _handle_special_torpedo(self):
        self._specialTorpedFired += 1

    def _handle_exit(self):
        self._endGame = True

    def _handle_left(self):
        self._leftClicks += 1

    def _handle_right(self):
        self._rightClicks += 1

    def _handle_up(self):
        self._upClicks += 1

    def _handle_space(self):
        self._fireClicks += 1

    def start_screen(self):
        """
        This is called to start our game (grphaics-wise).

        .. warning::

            **This method should not be called by you**
        """
        tkinter.mainloop()

    def update(self):
        """
        This is called to update our game (grphaics-wise).

        .. warning::

            **This method should not be called by you**
        """
        self._screen.update()

    def set_score(self, val):
        """
        Sets the current game score

        :param val: The game score
        :type val: int
        """
        self._score_val.set(str(val))

    def _get_ship_obj(self, canvas):
        ship = RawTurtle(canvas)
```

```python
        ship.shape(ShapesMaster.SHIP_SHAPE)
        ship.color("purple")
        return ship

    def _get_asteroid_object(self, size):
        asteroid = RawTurtle(self._cv)
        asteroid.shape(ShapesMaster.ASTEROID_BASE_SHAPE%size)
        return asteroid

    def _get_torpedo_object(self):
        torpedo = RawTurtle(self._cv)
        torpedo.shape(ShapesMaster.TORPEDO_SHAPE)
        torpedo.color("blue")
        return torpedo

    def _draw_object(self,obj,x,y,heading=None):
        obj.penup()
        obj.goto(x,y)
        if heading:
            obj.setheading(heading)
        obj.pendown()

    def remove_life(self):
        """
        Remove one icon of life (starts with 3 lives)
        """
        deadship = self._lives.pop()
        deadship.ht()

    def register_asteroid(self, asteroid, size):
        """
        This is called to register a new asteroid in our system

        :param asteroid: This is your asteroid object
        :type asteroid: Asteroid

        :param size: The size of the asteroid (this should be in [1,2,3])
        :type size: int
        """
        if size not in [1,2,3]:
            print("Error: Wrong asteroid size: %d"%size)
            sys.exit(0)
        elif id(asteroid) in self._asteroids:
            print("Error: Asteroid id (%d) already exists"%asteroid_id)
            sys.exit(0)
        asteroid_obj = self._get_asteroid_object(size)
        self._asteroids[ id(asteroid) ] = asteroid_obj


    def register_torpedo(self, torpedo):
        """
        This is called to register a new torpedo in our system

        :param asteroid: This is your torpedo object
        :type asteroid: Torpedo
        """
        if id(torpedo) in self._torpedos:
            print("Error: Torpedo id (%d) already exists"%torpedo_id)
            sys.exit(0)
        torpedo_obj = self._get_torpedo_object()
        self._torpedos[ id(torpedo) ] = torpedo_obj

    def draw_ship(self,x,y, heading):
        """
        Draw the ship at the given coordinates with the given heading

        :param x: This is the X coordinate of the ship
        :type x: int
        :param y: This is the Y coordinate of the ship
        :type y: int
        :param heading: This is the heading of the ship (in degrees)
        :type heading: float

        """
        self._draw_object(self._ship, x, y, heading)

    def draw_asteroid(self, asteroid, x, y):
        """
        Draw the given asteroid on the specified (x,y) coordinates

        :param asteroid: This is your asteroid object (remember to register it before)
        :type asteroid: Asteroid
        :param x: This is the X coordinate of the asteroid
        :type x: int
        :param y: This is the Y coordinate of the asteroid
        :type y: int

        """
        asteroid_id = id(asteroid)
        if asteroid_id not in self._asteroids:
            print("Error: Asteroid id (%d) not found. "%asteroid_id +
                  "Are you sure there is such an asteroid?")
            sys.exit(0)

        self._draw_object(self._asteroids[asteroid_id], x, y)

    def draw_torpedo(self, torpedo, x, y, heading):
        """
        Draw the given torpedo on the specified (x,y) coordinates with the given heading

        :param asteroid: This is your torpedo object (remember to register it before)
        :type asteroid: Torpedo
        :param x: This is the X coordinate of the torpedo
        :type x: int
        :param y: This is the Y coordinate of the torpedo
        :type y: int
        :param heading: This is the heading of the torpedo
        :type heading: float
        """
        torpedo_id = id(torpedo)
        if torpedo_id not in self._torpedos:
            print("Torpedo id (%d) not found. "%torpedo_id +
                  "Are you sure there is such a torpedo?")
            sys.exit(0)

        self._draw_object(self._torpedos[torpedo_id], x, y, heading)

    def _remove_object(self, obj):
        obj.penup()
        obj.ht()
        obj.goto(Screen.SCREEN_MAX_X, Screen.SCREEN_MAX_Y*2)


    def unregister_torpedo(self, torpedo):
        """
        This is called to un-register an existing torpedo in our system

        :param asteroid: This is your torpedo object
        :type asteroid: Torpedo
        """
        torpedo_id = id(torpedo)
        if torpedo_id not in self._torpedos:
            print("Torpedo id (%d) not found. "%torpedo_id +
                  "Are you sure there is such a torpedo?")
            sys.exit(0)
        torpedo_obj = self._torpedos[ torpedo_id ]
        self._remove_object( torpedo_obj )
        self._torpedos.pop( torpedo_id )


    def unregister_asteroid(self, asteroid):
        """
        This is called to un-register an existing asteroid in our system

        :param asteroid: This is your asteroid object
        :type asteroid: Asteroid
        """
        asteroid_id = id(asteroid)
        if asteroid_id not in self._asteroids:
            print("Asteroid id (%d) not found. "%asteroid_id +
                  "Are you sure there is such an asteroid?")
            sys.exit(0)
        asteroid_obj = self._asteroids[ asteroid_id ]
        self._remove_object( asteroid_obj )
        self._asteroids.pop( asteroid_id )

    def _clear_screen(self):
        self._cv.delete(’all’)


    def should_end(self):
        """
        :returns: True if the game should end or not (if "q" was pressed or not)
        """
        return self._endGame


    def is_left_pressed(self):
        """
        :returns: True if the left key was pressed, else False
        """
        res = self._leftClicks > 0
        self._leftClicks -= 1 if res else 0
        return res

    def is_up_pressed(self):
        """
        :returns: True if the up key was pressed, else False
        """
        res = self._upClicks > 0
        self._upClicks -= 1 if res else 0
        return res

    def is_right_pressed(self):
        """
        :returns: True if the right key was pressed, else False
        """
        res = self._rightClicks > 0
        self._rightClicks -= 1 if res else 0
        return res

    def is_space_pressed(self):
        """
        :returns: True if the fire key was pressed, else False
        """
        res = self._fireClicks > 0
        self._fireClicks -= 1 if res else 0
        return res

    def is_special_pressed(self):
        """
        :returns: True if the fire key was pressed, else False
        """
        res = self._specialTorpedFired > 0
        self._specialTorpedFired -= 1 if res else 0
        return res

    def show_message(self,title, msg):
        """
        This is a method used to show messages in the game.

        :param title: The title of the message box.
        :type title: str
        :param msg: The message to show in the message box.
        :type msg: str
        """
        tkinter.messagebox.showinfo(str(title), str(msg) )

    def end_game(self):
        """
        This ends the current game.
        """
        self._root.destroy()
        self._root.quit()
```

7 ship.py

```
from random import randrange
from screen import Screen as Scr
import copy
import math


class Ship:
    LEFT_MOVE = 7
    RIGHT_MOVE = -7
    DELTA_X = Scr.SCREEN_MAX_X - Scr.SCREEN_MIN_X
    DELTA_Y = Scr.SCREEN_MAX_Y - Scr.SCREEN_MIN_Y
    RADIUS = 1

    def __init__(self):
        self.__x_speed = 0
        self.__y_speed = 0
        self.__x_position = randrange(Scr.SCREEN_MIN_X, Scr.SCREEN_MAX_X)
        self.__y_position = randrange(Scr.SCREEN_MIN_Y, Scr.SCREEN_MAX_Y)
        self.__heading = 0
        self.__life = 3

    def get_x_position(self):
        return copy.copy(self.__x_position)

    def get_y_position(self):
        return copy.copy(self.__y_position)

    def get_x_speed(self):
        return copy.copy(self.__x_speed)

    def get_y_speed(self):
        return copy.copy(self.__y_speed)

    def get_heading(self):
        return copy.copy(self.__heading)

    def get_life(self):
        return copy.copy(self.__life)

    def change_heading_left(self):
        """
        This function updates left movement for the spaceship.
        """
        self.__heading += self.LEFT_MOVE
        return copy.copy(self.__heading)

    def change_heading_right(self):
        """
        This function updates right movement for the spaceship.
        """
        self.__heading += self.RIGHT_MOVE
        return copy.copy(self.__heading)

    def accelerated_ship(self):
        """
        This function defines the x and y of the ship when we want our ship
        to accelerate.
        """
        radian_heading = math.radians(self.__heading)
```

```python
        self.__x_speed += math.cos(radian_heading)
        self.__y_speed += math.sin(radian_heading)
        return self.__x_speed, self.__y_speed

    def move(self):
        """
         This function calculates our new coordinate of the position for our
         ship.
        """
        self.__x_position = ((self.get_x_speed() + self.get_x_position() -
                    Scr.SCREEN_MIN_X) % self.DELTA_X + Scr.SCREEN_MIN_X)
        self.__y_position = ((self.get_y_speed() + self.get_y_position() -
                    Scr.SCREEN_MIN_Y) % self.DELTA_Y + Scr.SCREEN_MIN_Y)

    def update_life(self):
        """
        This function updates the amount of lives that the ship has left by
        reducing one of the remaining lives.
        """
        self.__life -= 1
```

8 torpedo.py

```python
import copy
import math
from screen import Screen as Scr

class Torpedo:

    """
    Torpedo is a class mainly based on the class Ship's data. Because the
    torpedo is derived from the Ship, it's position and speed are attributes
    from which are dependant on the ships position and speed.
    """

    RADIUS = 4
    ACCELERATING_FACTOR = 2
    DELTA_X = Scr.SCREEN_MAX_X - Scr.SCREEN_MIN_X
    DELTA_Y = Scr.SCREEN_MAX_Y - Scr.SCREEN_MIN_Y
    LIFESPAN_MAX = 200

    def __init__(self, x_pos ,y_pos, x_speed, y_speed, direction):
        self.__x_position = x_pos
        self.__y_position = y_pos
        self.__x_speed = x_speed
        self.__y_speed = y_speed
        self.__direction = direction
        self.__lifespan = 1
        self.find_speed()

    def get_x_position(self):
        return copy.copy(self.__x_position)

    def get_y_position(self):
        return copy.copy(self.__y_position)

    def get_x_speed(self):
        return copy.copy(self.__x_speed)

    def get_y_speed(self):
        return copy.copy(self.__y_speed)

    def get_direction(self):
        return copy.copy(self.__direction)

    def get_lifespan(self):
        return copy.copy(self.__lifespan)

    def find_speed(self):
        """
        This function calculates the speed of the torpedo.
        """
        radian_direction = math.radians(self.__direction)
        self.__x_speed += self.ACCELERATING_FACTOR * math.cos(radian_direction)
        self.__y_speed += self.ACCELERATING_FACTOR * math.sin(radian_direction)

    def update_lifespan(self):
        """
        This function "clocks" the lifespan by adding 1 each time the function
         is called.
        """
        self.__lifespan += 1

    def move(self):
        """
         This function calculates our new coordinate of the position for our
         ship.
        """
        self.__x_position = ((self.get_x_speed() + self.get_x_position() -
                    Scr.SCREEN_MIN_X) % self.DELTA_X + Scr.SCREEN_MIN_X)
        self.__y_position = ((self.get_y_speed() + self.get_y_position() -
                    Scr.SCREEN_MIN_Y) % self.DELTA_Y + Scr.SCREEN_MIN_Y)
```