# Contents

1	Basic Test Results	2
2	README	3
3	AUTHORS	4
4	asteroid.py	5
5	asteroids main.py	7
6	screen.py	12
7	ship.py	20
8	torpedo.py	22

#### 1 Basic Test Results

```
Starting tests...
1
    Thu Dec 24 00:07:40 IST 2015
    e87f4ea2769cbf2cfa4d7ac0d88262d827245449 -
4
    Archive: /tmp/bodek.m4tG4o/intro2cs/ex9/elinorperl/presubmission/submission
     inflating: src/asteroid.py
8
      inflating: src/asteroids_main.py
    extracting: src/AUTHORS
9
10
     inflating: src/screen.py
     inflating: src/ship.py
inflating: src/torpedo.py
11
12
     inflating: src/README
13
14
    Testing README...
15
16
   Done testing README...
17
    Listing AUTHORS...
18
   elinorperl
19
    ynatovich
20
21
22
23
   Tests completed
25
   Additional notes:
26
    No presubmission tests of the code this week.
```

#### 2 README

```
elinorperl
1
2
    329577464
    Elinor Perl
4
    vnatovich
    204858914
6
    Yifat Natovitch
   I discussed the exercise with Talya Adams, Yonaton Iluz
11
12
    = README for ex9: Asteroids 00P =
14
15
16
17
    = Description: =
18
19
20
21
    In this exercise, using OOP, we defined three different classes "Ship",
    "Asteroid, and "Torpedo", each playing a part in the game we created
22
23
    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
25
26
    into smaller asteroids until they are small enought to dissappear.
27
28
29
    = Special Comments =
    _____
30
31
32
    We used stackoverflow.com
33
34
    Throughout programming this game, we encountered obstacles about how to
    proceed with our game, amongst them:
35
36
37
    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
38
39
    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
41
42
    upon placing the ship's movement in its own class and calling it
43
    to our intermmediary when necessary.
44
    2. In the class "Torpedo", we debated on whether to define a constant in the
45
    initial creation of itself, or to give the ability to an outside user to change
46
47
    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.
49
50
    3. When ending the game, we didn't know whether to make a one function
51
    to end the game or to split it into two, to give it a "cleaner" look,
52
    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

1 elinorperl, ynatovich

## 4 asteroid.py

```
from random import randrange, uniform
1
    from screen import Screen as Scr
    import copy
4
    import math
    from ship import Ship
    class Asteroid:
8
        DELTA_X = Scr.SCREEN_MAX_X - Scr.SCREEN_MIN_X
9
10
        DELTA_Y = Scr.SCREEN_MAX_Y - Scr.SCREEN_MIN_Y
        RADIUS_COEFFIC = 10
11
        NORM_FACTOR = -5
12
13
        BIG = 3
        MIDDLE = 2
14
        SMALL = 1
15
16
        def __init__(self):
17
18
            self.__x_speed = uniform(-5, 5)
            self.__y_speed = uniform(-5, 5)
19
            self.__x_position = randrange(Scr.SCREEN_MIN_X, Scr.SCREEN_MAX_X)
20
21
            self.__y_position = randrange(Scr.SCREEN_MIN_Y, Scr.SCREEN_MAX_Y)
            self.\_size = 3
22
            self.__radius = self.__size * self.RADIUS_COEFFIC + self.NORM_FACTOR
23
24
        def get_x_position(self):
25
26
            return copy.copy(self.__x_position)
27
        def get_y_position(self):
28
29
            return copy.copy(self.__y_position)
30
31
        def get_x_speed(self):
            return copy.copy(self.__x_speed)
33
34
        def get_y_speed(self):
            return copy.copy(self.__y_speed)
35
36
37
        def get_size(self):
            return copy.copy(self.__size)
38
39
40
        def change_position(self, new_position):
41
42
            Changes the position of the asteroid according to the new position
43
            input.
44
45
            self.__x_position, self.__y_position = new_position[0], new_position[1]
46
        def change_size(self, size):
47
48
            Changes the size of the asteroid according to the size input.
49
50
            self.__size = size
51
52
53
        def move(self):
54
55
             This function calculates our new coordinate of the position for our
57
             asteroid.
58
            self.__x_position = ((self.get_x_speed() + self.get_x_position() -
59
```

```
60
                        {\tt Scr.SCREEN\_MIN\_X)} \ \% \ {\tt self.DELTA\_X} \ + \ {\tt Scr.SCREEN\_MIN\_X)}
            self.__y_position = ((self.get_y_speed() + self.get_y_position() -
61
                        Scr.SCREEN_MIN_Y) % self.DELTA_Y + Scr.SCREEN_MIN_Y)
62
63
64
        def get_interaction(self, obj):
            distance_x = math.pow((obj.get_x_position() - self.get_x_position()),2)
65
            distance_y = math.pow((obj.get_y_position() - self.get_y_position()),2)
66
            distance = math.sqrt(distance_x + distance_y)
67
68
            if distance <= self.__radius + obj.RADIUS:</pre>
               return True
69
            return False
70
71
        def speed_after_interaction(self, torpedo, direction):
72
73
74
            A function that calculates the speed of the new asteroids that were
            created after a collision with a torpedo, where one of the parameters
75
76
            the two directions of the new asteroids. The asteroid splits into two,
77
            allowing us to use the indexes for 0 and 1 as our direction guide. \theta
78
            for the positive direction and 1 for the negative.
79
80
            speed_denominator = math.sqrt(math.pow(self.__x_speed,2) +\
81
                                         math.pow(self.__y_speed,2))
82
            self.__x_speed = ((torpedo.get_x_speed() + self.get_x_speed()) /
83
84
                              speed_denominator)
            self.__y_speed = ((torpedo.get_y_speed() + self.get_y_speed()) /
85
                              speed_denominator)
86
            if direction == 1:
87
                self.__x_speed *= -1
88
89
                self.\__y\_speed *= -1
```

## 5 asteroids main.py

```
1
    from screen import Screen
    import sys
    from ship import Ship
    from asteroid import Asteroid
4
    from torpedo import Torpedo
    import copy
    from random import randrange
8
    DEFAULT_ASTEROIDS_NUM = 5
9
10
11
    class GameRunner:
12
13
        DELTA_X = Screen.SCREEN_MAX_X - Screen.SCREEN_MIN_X
14
        DELTA_Y = Screen.SCREEN_MAX_Y - Screen.SCREEN_MIN_Y
15
        MAX_TORPEDOS = 15
16
17
18
         def __init__(self, asteroids_amnt=DEFAULT_ASTEROIDS_NUM):
             self._screen = Screen()
19
             self.screen_max_x = Screen.SCREEN_MAX_X
20
21
             self.screen_max_y = Screen.SCREEN_MAX_Y
            self.screen_min_x = Screen.SCREEN_MIN_X
22
23
             self.screen_min_y = Screen.SCREEN_MIN_Y
             self._ship = Ship()
24
            self._asteroids_amnt = asteroids_amnt
25
26
             self._asteroids = []
27
             self._torpedos = []
             self._score = 0
28
29
             self._screen.set_score(self._score)
            self.create_asteroids()
30
31
         def get_score(self):
            return copy.copy(self._score)
33
34
        def create_asteroids(self):
35
36
37
             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
38
39
             registered each one.
40
            for i in range(self._asteroids_amnt):
41
42
                 asteroid = Asteroid()
43
                 self.get_random_position_asteroid(asteroid)
                 self. asteroids.append(asteroid)
44
45
                 self._screen.register_asteroid(asteroid, asteroid.get_size())
46
47
         def get_random_position_asteroid(self, asteroid):
48
             This function places the asteroid in a random location, and if the
49
50
             asteroids location lands on the spaceship's initial location, the
             function recursively relocates the asteroid.
51
52
53
             {\tt asteroid.change\_position((randrange(Screen.SCREEN\_MIN\_X,
                                                 Screen.SCREEN_MAX_X),\
54
                                       {\tt randrange} \, ({\tt Screen.SCREEN\_MIN\_X} \, ,
55
                                                 Screen.SCREEN_MAX_X)))
56
             while ((asteroid.get_x_position(), asteroid.get_y_position()) == \
57
58
                     (self._ship.get_x_position(), self._ship.get_y_position())):
                 self.get_random_position_asteroid(asteroid)
```

```
60
         def run(self):
61
62
              self._do_loop()
              self._screen.start_screen()
 63
64
65
          def _do_loop(self):
              # You don't need to change this method!
66
              self._game_loop()
67
68
              # Set the timer to go off again
69
              self._screen.update()
70
71
              self._screen.ontimer(self._do_loop, 5)
72
73
          def operate_torpedo(self):
74
              This function operates the torpedo's functionality. Calling the class
75
76
              {\it Torpedo\ and\ input ting\ ship's\ attributes\ (serving\ as\ the\ ship\ and}
              torpedo's bridge of connection), adding each torpedo to the list
77
              created and drawing it on the screen.
78
79
              torpedo = Torpedo(self._ship.get_x_position(),
80
81
                                             self._ship.get_y_position(),
82
                                             self._ship.get_x_speed(),
83
                                             self._ship.get_y_speed(),
84
                                             self._ship.get_heading())
85
              self._torpedos.append(torpedo)
              {\tt self.\_screen.register\_torpedo(torpedo)}
86
87
              self._screen.draw_torpedo(torpedo, torpedo.get_x_position(),
                                         torpedo.get_y_position(),
88
89
                                         torpedo.get_direction())
90
          def game_movements(self):
91
92
93
              This function checks if the user pressed left, right or up and
              depending on the input, it returns the function accordingly.
94
95
96
              if Screen.is_left_pressed(self._screen):
97
                  self._ship.change_heading_left()
              elif Screen.is_right_pressed(self._screen):
                  self._ship.change_heading_right()
99
100
              elif Screen.is_up_pressed(self._screen):
                  self._ship.accelerated_ship()
101
102
              elif Screen.is_space_pressed(self._screen):
103
                  if len(self._torpedos) < self.MAX_TORPEDOS:</pre>
                      self.operate_torpedo()
104
105
106
          def draw_objects(self):
107
108
              This function calls the functions draw ship and draw (each) asteroid
109
              on the screen.
110
              self._screen.draw_ship(self._ship.get_x_position(),
111
112
                                self._ship.get_y_position(),
113
                                self._ship.get_heading())
              for asteroid in self._asteroids:
114
                  self._screen.draw_asteroid(asteroid, asteroid.get_x_position(),\
115
116
                                        asteroid.get_y_position())
117
              for torpedo in self._torpedos:
                  \verb|self._screen.draw_torpedo(torpedo, torpedo.get_x_position(),\\
118
119
                                          torpedo.get_y_position(),
120
                                          torpedo.get_direction())
121
          def move_asteroid(self):
122
123
              This function moves each asteroid from the list of asteroids that we
124
125
              created, moving each asteroid in turn and having them appear on the
              screen.
126
127
```

```
128
             for asteroid in self._asteroids:
                  asteroid.move()
129
130
         def move_torpedos(self):
131
132
              This function moves each asteroid from the list of asteroids that we
133
              created, moving each asteroid in turn and having them appear on the
134
              screen.
135
136
             for torpedo in self._torpedos:
137
                 torpedo.move()
138
139
         def interaction_ship(self, asteroid):
140
141
142
              This function deals with the interaction between the ship and
              asteroids by removing one of the users lives, and displaying a message
143
144
              to the user to notify them that they hit an asteroid and thereafter
             removing them.
145
146
             Screen.remove_life(self._screen)
147
             self._ship.update_life()
148
149
              \# in the event that there are two collision in one turn:
              if self._ship.get_life() == 0:
150
                 self.ending("Out of lives", "Maybe next time...")
151
              self._screen.show_message("BOOM!","You hit"
152
153
                                           " an asteroid. BE CAREFUL!")
              self._asteroids.remove(asteroid)
154
155
              self._screen.unregister_asteroid(asteroid)
156
157
         def interaction_torpedo(self, asteroid, torpedo):
158
              This function deals with the occurence of interaction with torpedos
159
160
              and asteroids, removing the torpedo from the screen, and updating
161
              the user's score.
162
              self._torpedos.remove(torpedo)
163
164
              self._screen.unregister_torpedo(torpedo)
165
              self.update_score(asteroid)
166
              self.split_asteroid(asteroid, torpedo)
167
         def create_new_asteroids(self, asteroid, torpedo, size):
168
169
              This function splits the current asteroid to 2 new smaller ones and
170
171
              defines its attributes and position on the screen.
172
173
              get_x = asteroid.get_x_position()
174
              get_y = asteroid.get_y_position()
             for i in range(2):
175
176
                  asteroid_i = Asteroid()
177
                  asteroid_i.change_position((get_x, get_y))
                  asteroid_i.change_size(size)
178
                  asteroid_i.speed_after_interaction(torpedo, i)
179
180
                  self._asteroids.append(asteroid_i)
181
                  self._screen.register_asteroid(asteroid_i, asteroid_i.get_size())
182
         def split_asteroid(self, asteroid, torpedo):
183
184
185
              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
186
187
              well.
188
189
              self._asteroids.remove(asteroid)
              self._screen.unregister_asteroid(asteroid)
190
191
             if asteroid.get_size() == asteroid.BIG:
192
193
                  self.create_new_asteroids(asteroid, torpedo, asteroid.MIDDLE)
              if asteroid.get_size() == asteroid.MIDDLE:
194
195
                  {\tt self.create\_new\_asteroids(asteroid,\ torpedo,\ asteroid.SMALL)}
```

```
196
         def update_score(self, asteroid):
197
198
              A function that updates the score according to the asteroid's size.
199
200
201
              if asteroid.get_size() == asteroid.BIG:
202
                  self._score += 20
              if asteroid.get_size() == asteroid.MIDDLE:
203
204
                  self._score += 50
              if asteroid.get_size() == asteroid.SMALL:
205
                  self._score += 100
206
207
              self._screen.set_score(self._score)
208
209
          def interaction(self):
210
              Using a copied version of our asteroid_list, we checked if there was a
211
212
              collision with each asteroid and the spaceship. If there was a inter-
213
              action, the spaceship removes a life, and displays a message that
              there has been a collision and removes an asteroid from our asteroid
214
215
              list.
216
217
              copied_asteroid = copy.copy(self._asteroids)
              copied_torpedos = copy.copy(self._torpedos)
218
219
             for asteroid in copied_asteroid:
220
                  if asteroid.get_interaction(self._ship):
221
                      self.interaction_ship(asteroid)
             for asteroid in copied_asteroid:
222
223
                  for torpedo in copied_torpedos:
                      if asteroid.get_interaction(torpedo):
224
225
                          # if there is a collision, loop continues to other objects
226
                          if (asteroid in self._asteroids) and \
                                  (torpedo in self. torpedos):
227
228
                              self.interaction_torpedo(asteroid, torpedo)
229
          def update_lifespan_torpedos(self):
230
231
232
              Updates the lifespan of each torpedo on our list
233
              copied_torpedos = copy.copy(self._torpedos)
234
             for torpedo in copied_torpedos:
235
236
                  torpedo.update_lifespan()
                  if torpedo.get_lifespan() == torpedo.LIFESPAN_MAX:
237
                      self._torpedos.remove(torpedo)
238
239
                      self._screen.unregister_torpedo(torpedo)
240
241
          def end_game(self):
242
              Returns message according to the reason for finishing the game and
243
244
              calls to the function that displays the message and closes the tab.
245
              if len(self. asteroids) == 0:
246
                  self.ending("WINNER!!!", "YOU ARE THE CHAMPION MY FRIEND! YOU "
247
248
                                           "HAVE DESTRYOED ALL THE ASTEROIDS :)")
249
              if self._ship.get_life() == 0:
                  self.ending("Out of lives", "Maybe next time...")
250
              if self._screen.should_end():
251
                  self.ending("QUITTER", "Byebye!")
252
253
         def ending(self, title, msg):
254
255
256
              This function ends the game by showing the message with the reason
257
              for ending, then closes the screen.
258
              self._screen.show_message(title, msg)
259
260
              self._screen.end_game()
261
              sys.exit()
262
263
         def _game_loop(self):
```

```
^{264}
265
             This is the function that ties the whole game together, calling each
             function in turn, and creating each round of the game.
266
267
             self._ship.move()
268
             self.move_asteroid()
269
270
             self.move_torpedos()
             self.game_movements()
271
272
             self.draw_objects()
             self.interaction()
273
             self.update_lifespan_torpedos()
274
275
             self.end_game()
276
277
278
     def main(amnt):
         runner = GameRunner(amnt)
279
         runner.run()
^{280}
281
     if __name__ == "__main__":
282
         if len(sys.argv) > 1:
283
284
            main( int( sys.argv[1] ) )
         else:
285
286
             main(DEFAULT_ASTEROIDS_NUM)
```

#### 6 screen.py

```
1
    import sys
2
    import tkinter
    import tkinter.messagebox
4
    from turtle import *
    class ShapesMaster:
8
        ASTEROID_BASE_SHAPE = "asteroid%d"
        SHIP_SHAPE = "ship"
9
10
        TORPEDO_SHAPE = "torpedo"
11
        ASTEROIDS_TYPES = 3
12
13
         ASTEROID_3_LAYOUT = ((-20, -16), (-21, 0), (-20, 18), (0, 27), (17, 15),
14
15
                                  (25,0),(16,-15),(0,-21))
16
         ASTEROID_2_LAYOUT = ((-15, -10), (-16, 0), (-13, 12), (0, 19), (12, 10),
17
18
                                  (20,0),(12,-10),(0,-13))
19
         ASTEROID_1_LAYOUT = ((-10,-5),(-12,0),(-8,8),(0,13),(8,6),(14,0),(12,0),(8,-6),(0,-7))
20
21
         ASTEROIDS_LAYOUTS = [ASTEROID_1_LAYOUT, ASTEROID_2_LAYOUT, ASTEROID_3_LAYOUT]
22
23
         SHIP_LAYOUT = ((-10,-10),(0,-5),(10,-10),(0,10))
24
25
        TORPEDO_LAYOUT = ((-2,-4),(-2,4),(2,4),(2,-4))
26
27
         def __init__(self, screen):
28
29
            This initializes the shapes controller, the screen passed is the screen
30
31
            controling the game, you should not call this method anywhere in your
33
34
            self.screen = screen
            self._shapes = {}
35
             self._updated = False
36
37
             self._add_base_shapes()
38
39
        def add_shape(self,name,cords,override = False):
40
             if override or name not in self._shapes:
                 self._shapes[name] = cords
41
42
                 self.screen.register_shape(name,cords)
43
        def _add_base_shapes(self):
44
45
            for i in range(ShapesMaster.ASTEROIDS_TYPES):
                 {\tt self.add\_shape(ShapesMaster.ASTEROID\_BASE\_SHAPE\%(i+1), \ \ }
46
47
                             ShapesMaster.ASTEROIDS_LAYOUTS[i])
48
             self.add_shape(ShapesMaster.SHIP_SHAPE, ShapesMaster.SHIP_LAYOUT)
49
50
             self.add_shape(ShapesMaster.TORPEDO_SHAPE, ShapesMaster.TORPEDO_LAYOUT)
51
        def get_shapes_dict(self):
52
53
            Returns a dictionary of all the shapes in the game in the format of
54
55
             (name, coordinates).
             You have no reason of calling this method anywhere in your code...
57
58
            return self._shapes
59
```

```
60
     class Screen:
 61
 62
          SCREEN_MIN_X = -500
 63
          SCREEN_MIN_Y = -500
 64
         SCREEN_MAX_X = 500
 65
         SCREEN_MAX_Y = 500
 66
 67
 68
          def __init__(self):
 69
              This inits our graphics class.
 70
 71
 72
 73
              self._boundKeys = []
 74
              self._init_keys_values()
              self._init_graphics()
 75
 76
              self._bind_keys()
 77
              self._screen.listen()
 78
 79
              self._ship = self._get_ship_obj(self._cv)
 80
         def _init_keys_values(self):
 81
              self._specialTorpedFired = 0
 82
              self._rightClicks = 0
 83
              self._leftClicks = 0
 84
 85
              self._upClicks = 0
              self._fireClicks = 0
 86
 87
              self._endGame = False
              self._lives = []
 88
 89
              self._asteroids = {}
 90
              self._torpedos = {}
 91
 92
         def _init_graphics(self):
 93
              self._root = tkinter.Tk()
              self._root.title("Asteroids!")
 94
 95
              self._cv = ScrolledCanvas(self._root,600,600,600,600)
              self._cv.pack(side = tkinter.LEFT)
 96
 97
              self._t = RawTurtle(self._cv)
 98
              self._screen = self._t.getscreen()
 99
100
              self._screen.setworldcoordinates(
                                               Screen.SCREEN_MIN_X,
101
                                               Screen.SCREEN_MIN_Y,
102
103
                                               Screen.SCREEN_MAX_X,
                                               Screen.SCREEN_MAX_X
104
105
106
              self._shapeMaster = ShapesMaster(self._screen)
              shapes = self._shapeMaster.get_shapes_dict()
107
108
109
              frame = tkinter.Frame(self._root)
              frame.pack(side = tkinter.RIGHT,fill=tkinter.BOTH)
110
111
112
              # add scores frame
113
              self._score_val = tkinter.StringVar()
              self._score_val.set("0")
114
              scoreTitle = tkinter.Label(frame,text="Score")
115
116
              scoreTitle.pack()
              scoreFrame = tkinter.Frame(frame,height=2, bd=1, \
117
                  relief=tkinter.SUNKEN)
118
119
              scoreFrame.pack()
              score = tkinter.Label(scoreFrame,height=2,width=20,\
120
121
                  textvariable=self._score_val,fg="Yellow",bg="black")
122
              ################
123
124
125
              score.pack()
126
127
              # Add Lives Frame
```

```
128
             # livesTitle = tkinter.Label(frame, \
                  text="Extra Lives Remaining")
129
             # livesTitle.pack()
130
131
             # livesFrame = tkinter.Frame(frame, \
132
                    height = 30, width = 60, relief = tkinter. \textit{SUNKEN})
133
              # livesFrame.pack()
134
              # self._lives_canvas = ScrolledCanvas(livesFrame, 150, 40, 150, 40)
135
136
              # self._lives_canvas.pack()
              # livesTurtle = RawTurtle(self._lives_canvas)
137
             # livesTurtle.ht()
138
              # livesScreen = livesTurtle.getscreen()
139
              # livesScreen.register_shape(ShapesMaster.SHIP_SHAPE, shapes[ShapesMaster.SHIP_SHAPE])
140
141
142
              # Add Lives Frame
             livesTitle = tkinter.Label(frame, \
143
144
                 text="Extra Lives Remaining")
             livesTitle.pack()
145
146
             livesFrame = tkinter.Frame(frame, \
147
                  height=30, width=60, relief=tkinter.SUNKEN)
148
149
             livesFrame.pack()
             livesCanvas = ScrolledCanvas(livesFrame, 150, 40, 150, 40)
150
151
              livesCanvas.pack()
152
              livesTurtle = RawTurtle(livesCanvas)
153
              livesTurtle.ht()
              livesScreen = livesTurtle.getscreen()
154
155
              livesScreen.register_shape(ShapesMaster.SHIP_SHAPE, shapes[ShapesMaster.SHIP_SHAPE])
156
157
             life1 = self._get_ship_obj(livesCanvas) # SpaceShip(livesCanvas, -35,0,0,0)
158
              life2 = self._get_ship_obj(livesCanvas) #SpaceShip(livesCanvas,0,0,0,0)
             life3 = self._get_ship_obj(livesCanvas) #SpaceShip(livesCanvas, 35, 0, 0, 0)
159
160
161
              self._draw_object(life1,-35,0)
              self._draw_object(life2,0,0)
162
              self._draw_object(life3,35,0)
163
164
             self._lives = [life1, life2, life3]
165
166
             self._t.ht()
167
168
             quitButton = tkinter.Button(frame, text = "Quit", command=self._handle_exit)
169
170
             quitButton.pack()
171
172
              self._screen.tracer(0)
173
174
          def ontimer(self, func, milli):
175
176
              This method is used to create a repeating action in your game.
177
              .. warning::
178
179
180
                  **You don't need to call this method, it was already called for you at the end of the main game loop.**
181
              :param func: The function to repeat after **milli** milliseconds have passed
182
              :type func: function
183
              :param milli: The amount of milliseconds to wait before starting the given
184
185
                 function
              :type milli: int
186
187
188
              self._screen.ontimer(func,milli)
189
          def _bind_key(self, key, func):
190
191
              This method is to allow you to add some functionality of your own,
192
              it allows you to bind the provided function to the desired input key.
193
194
195
             If there is already a function bound to this key it will do nothing.
```

```
196
197
              :param key: A key to bind.
198
              :type key: str
199
              :param func: The function to bind
              :type func: function
200
201
202
              if key not in self._boundKeys:
203
204
                  self._screen.onkeypress(func,key)
                  self._boundKeys.append(key)
205
206
207
          def _bind_keys(self):
              self._bind_key("Left", self._handle_left)
208
              self._bind_key("Right", self._handle_right)
209
210
              self._bind_key("Up", self._handle_up)
              self._bind_key("space", self._handle_space)
211
              {\tt self.\_bind\_key("q", self.\_handle\_exit)}
212
              self._bind_key("s", self._handle_special_torpedo)
213
214
215
          def _handle_special_torpedo(self):
              self._specialTorpedFired += 1
216
217
          def _handle_exit(self):
218
              self._endGame = True
219
220
          def _handle_left(self):
221
              self._leftClicks += 1
222
223
          def _handle_right(self):
224
225
              self._rightClicks += 1
226
          def _handle_up(self):
227
228
              self._upClicks += 1
229
          def _handle_space(self):
230
231
              self._fireClicks += 1
232
          def start_screen(self):
233
^{234}
              This is called to start our game (grphaics-wise).
235
236
237
              .. warning::
238
              **This method should not be called by you**
239
240
              tkinter.mainloop()
241
^{242}
          def update(self):
243
244
              This is called to update our game (grphaics-wise).
245
246
^{247}
              .. warning::
248
249
                  **This method should not be called by you**
250
              self._screen.update()
251
252
          def set_score(self, val):
253
254
255
              Sets the current game score
256
257
              :param val: The game score
258
              :type val: int
259
              self._score_val.set(str(val))
260
261
          def _get_ship_obj(self, canvas):
262
263
              ship = RawTurtle(canvas)
```

```
264
              ship.shape(ShapesMaster.SHIP_SHAPE)
              ship.color("purple")
265
266
              return ship
267
          def _get_asteroid_object(self, size):
268
              asteroid = RawTurtle(self._cv)
269
              {\tt asteroid.shape} ({\tt ShapesMaster.ASTEROID\_BASE\_SHAPE\%size})
270
              return asteroid
271
272
          def _get_torpedo_object(self):
273
274
              torpedo = RawTurtle(self._cv)
275
              torpedo.shape(ShapesMaster.TORPEDO_SHAPE)
276
              torpedo.color("blue")
277
              return torpedo
278
          def _draw_object(self,obj,x,y,heading=None):
279
280
              obj.penup()
              obj.goto(x,y)
281
              if heading:
282
283
                  obj.setheading(heading)
              obj.pendown()
284
285
          def remove_life(self):
286
287
288
              Remove one icon of life (starts with 3 lives)
289
              deadship = self._lives.pop()
290
291
              deadship.ht()
292
293
          def register_asteroid(self, asteroid, size):
294
              This is called to register a new asteroid in our system
295
296
297
              :param asteroid: This is your asteroid object
              :type asteroid: Asteroid
298
299
              :param size: The size of the asteroid (this should be in [1,2,3])
300
301
              :type size: int
302
              if size not in [1,2,3]:
303
                  print("Error: Wrong asteroid size: %d"%size)
304
                  sys.exit(0)
305
              elif id(asteroid) in self._asteroids:
306
307
                  print("Error: Asteroid id (%d) already exists"%asteroid_id)
                  sys.exit(0)
308
              asteroid_obj = self._get_asteroid_object(size)
309
310
              self._asteroids[ id(asteroid) ] = asteroid_obj
311
312
313
          def register_torpedo(self, torpedo):
314
315
              This is called to register a new torpedo in our system
316
317
              :param asteroid: This is your torpedo object
              :type asteroid: Torpedo
318
319
              if id(torpedo) in self._torpedos:
320
                  print("Error: Torpedo id (%d) already exists"%torpedo_id)
321
                  sys.exit(0)
322
323
              torpedo_obj = self._get_torpedo_object()
              self._torpedos[ id(torpedo) ] = torpedo_obj
324
325
326
          def draw_ship(self,x,y, heading):
327
              Draw the ship at the given coordinates with the given heading
328
329
              :param x: This is the X coordinate of the ship
330
331
              :type x: int
```

```
332
              :param y: This is the Y coordinate of the ship
333
              :type y: int
              :param heading: This is the heading of the ship (in degrees)
334
              :type heading: float
335
336
337
338
              self._draw_object(self._ship, x, y, heading)
339
340
          def draw_asteroid(self, asteroid, x, y):
341
             Draw the given asteroid on the specified (x,y) coordinates
342
343
344
             :param asteroid: This is your asteroid object (remember to register it before)
345
              :type asteroid: Asteroid
346
              :param x: This is the X coordinate of the asteroid
              :type x: int
347
348
              :param y: This is the Y coordinate of the asteroid
              :type y: int
349
350
351
              asteroid_id = id(asteroid)
352
              if asteroid_id not in self._asteroids:
353
                  print("Error: Asteroid id (%d) not found. "%asteroid_id +
354
                        "Are you sure there is such an asteroid?")
355
356
                  sys.exit(0)
357
              self._draw_object(self._asteroids[asteroid_id], x, y)
358
359
         def draw_torpedo(self, torpedo, x, y, heading):
360
361
362
              Draw the given torpedo on the specified (x,y) coordinates with the given heading
363
              :param asteroid: This is your torpedo object (remember to register it before)
364
365
              :type asteroid: Torpedo
              :param x: This is the X coordinate of the torpedo
366
367
              :type x: int
              :param y: This is the Y coordinate of the torpedo
368
369
              :type y: int
              :param heading: This is the heading of the torpedo
370
              :type heading: float
371
372
373
              torpedo_id = id(torpedo)
374
             if torpedo_id not in self._torpedos:
                  print("Torpedo id (%d) not found. "%torpedo_id +
375
                        "Are you sure there is such a torpedo?")
376
377
                  sys.exit(0)
378
              self._draw_object(self._torpedos[torpedo_id], x, y, heading)
379
380
          def _remove_object(self, obj):
381
              obj.penup()
382
383
              obj.ht()
384
              obj.goto(Screen.SCREEN_MAX_X, Screen.SCREEN_MAX_Y*2)
385
386
         def unregister_torpedo(self, torpedo):
387
388
389
              This is called to un-register an existing torpedo in our system
390
391
              :param asteroid: This is your torpedo object
              :type asteroid: Torpedo
392
393
              torpedo_id = id(torpedo)
394
              if torpedo_id not in self._torpedos:
395
                  print("Torpedo id (%d) not found. "%torpedo_id +
396
                        "Are you sure there is such a torpedo?")
397
                  sys.exit(0)
398
399
             torpedo_obj = self._torpedos[ torpedo_id ]
```

```
400
             self._remove_object( torpedo_obj )
401
             self._torpedos.pop( torpedo_id )
402
403
404
         def unregister_asteroid(self, asteroid):
405
             This is called to un-register an existing asteroid in our system
406
407
408
              :param asteroid: This is your asteroid object
              :type asteroid: Asteroid
409
410
411
             asteroid_id = id(asteroid)
412
             if asteroid_id not in self._asteroids:
                 print("Asteroid id (%d) not found. "%asteroid_id +
413
414
                        "Are you sure there is such an asteroid?")
                 svs.exit(0)
415
416
             asteroid_obj = self._asteroids[ asteroid_id ]
417
              self._remove_object( asteroid_obj )
             self._asteroids.pop( asteroid_id )
418
419
420
         def _clear_screen(self):
              self._cv.delete('all')
421
422
423
         def should_end(self):
424
425
             :returns: True if the game should end or not (if "q" was pressed or not)
426
427
428
             return self._endGame
429
430
         def is_left_pressed(self):
431
432
433
              :returns: True if the left key was pressed, else False
434
435
             res = self._leftClicks > 0
             self._leftClicks -= 1 if res else 0
436
             return res
437
438
         def is_up_pressed(self):
439
440
441
              :returns: True if the up key was pressed, else False
442
443
             res = self._upClicks > 0
             self._upClicks -= 1 if res else 0
444
445
             return res
446
         def is_right_pressed(self):
447
448
              :returns: True if the right key was pressed, else False
449
450
451
             res = self._rightClicks > 0
452
             self._rightClicks -= 1 if res else 0
453
             return res
454
         def is_space_pressed(self):
455
456
              :returns: True if the fire key was pressed, else False
457
458
459
             res = self._fireClicks > 0
             self._fireClicks -= 1 if res else 0
460
461
             return res
462
         def is_special_pressed(self):
463
464
             :returns: True if the fire key was pressed, else False
465
466
467
             res = self._specialTorpedFired > 0
```

```
self._specialTorpedFired -= 1 if res else 0
468
469
             return res
470
         def show_message(self,title, msg):
471
472
              This is a method used to show messages in the game.
473
474
              :param title: The title of the message box.
475
476
              : type\ title:\ str
477
              :param msg: The message to show in the message box.
              :type msg: str
478
479
              tkinter.messagebox.showinfo(str(title), str(msg) )
480
481
482
         def end_game(self):
483
              This ends the current game.
484
485
              11 11 11
              self._root.destroy()
486
487
              self._root.quit()
```

## 7 ship.py

```
from random import randrange
1
2
    from screen import Screen as Scr
    import copy
    import math
4
    class Ship:
8
        LEFT_MOVE = 7
        RIGHT\_MOVE = -7
9
10
        DELTA_X = Scr.SCREEN_MAX_X - Scr.SCREEN_MIN_X
        DELTA_Y = Scr.SCREEN_MAX_Y - Scr.SCREEN_MIN_Y
11
        RADIUS = 1
12
13
        def __init__(self):
14
15
            self.\_x\_speed = 0
            self.__y_speed = 0
16
            self.__x_position = randrange(Scr.SCREEN_MIN_X, Scr.SCREEN_MAX_X)
17
18
            self.__y_position = randrange(Scr.SCREEN_MIN_Y, Scr.SCREEN_MAX_Y)
            self.__heading = 0
19
            self.\_\_life = 3
20
21
        def get_x_position(self):
22
23
            return copy.copy(self.__x_position)
24
        def get_y_position(self):
25
26
            return copy.copy(self.__y_position)
27
        def get_x_speed(self):
28
29
            return copy.copy(self.__x_speed)
30
31
        def get_y_speed(self):
            return copy.copy(self.__y_speed)
33
34
        def get_heading(self):
            return copy.copy(self.__heading)
35
36
37
        def get_life(self):
            return copy.copy(self.__life)
38
39
40
        def change_heading_left(self):
41
42
             This function updates left movement for the spaceship.
43
            self.__heading += self.LEFT_MOVE
44
45
            return copy.copy(self.__heading)
46
47
        def change_heading_right(self):
48
             This function updates right movement for the spaceship.
49
50
             self.__heading += self.RIGHT_MOVE
51
            return copy.copy(self.__heading)
52
53
        def accelerated_ship(self):
54
55
            This function defines the x and y of the ship when we want our ship
            to accelerate.
57
58
            radian_heading = math.radians(self.__heading)
59
```

```
60
         self.__x_speed += math.cos(radian_heading)
         self.__y_speed += math.sin(radian_heading)
61
62
         \tt return \ self.\_x\_speed, \ self.\_y\_speed
      def move(self):
64
65
66
          This function calculates our new coordinate of the position for our
         ship.
67
68
         69
70
         71
72
73
74
      def update_life(self):
75
         This function updates the amount of lives that the ship has left by
76
         reducing one of the remaining lives.
77
78
         self.__life -= 1
```

#### 8 torpedo.py

```
import copy
1
    import math
    from screen import Screen as Scr
4
    class Torpedo:
8
         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
9
10
         from which are dependant on the ships position and speed.
11
12
13
        RADIUS = 4
         ACCELERATING_FACTOR = 2
14
        DELTA_X = Scr.SCREEN_MAX_X - Scr.SCREEN_MIN_X
15
        DELTA_Y = Scr.SCREEN_MAX_Y - Scr.SCREEN_MIN_Y
16
        LIFESPAN_MAX = 200
17
18
        def __init__(self, x_pos ,y_pos, x_speed, y_speed, direction):
19
            self.\_x\_position = x\_pos
20
21
             self.__y_position = y_pos
            self.\_x\_speed = x\_speed
22
23
            self.__y_speed = y_speed
            self.__direction = direction
24
            self.__lifespan = 1
25
26
            self.find_speed()
27
        def get_x_position(self):
28
29
            return copy.copy(self.__x_position)
30
31
        def get_y_position(self):
            return copy.copy(self.__y_position)
33
34
        def get_x_speed(self):
            return copy.copy(self.__x_speed)
35
36
37
         def get_y_speed(self):
38
            return copy.copy(self.__y_speed)
39
40
        def get_direction(self):
            return copy.copy(self.__direction)
41
42
43
        def get_lifespan(self):
            return copy.copy(self.__lifespan)
44
45
        def find_speed(self):
46
47
             This function calculates the speed of the torpedo.
49
50
            radian_direction = math.radians(self.__direction)
             self.__x_speed += self.ACCELERATING_FACTOR * math.cos(radian_direction)
51
            self.__y_speed += self.ACCELERATING_FACTOR * math.sin(radian_direction)
52
53
         def update_lifespan(self):
54
55
             This function "clocks" the lifespan by adding 1 each time the function
57
             is called.
            self.__lifespan += 1
```

```
60
      def move(self):
    """
61
62
          This function calculates our new coordinate of the position for our
63
          ship.
64
65
          self.__x_position = ((self.get_x_speed() + self.get_x_position() -
66
67
                   Scr.SCREEN_MIN_X) % self.DELTA_X + Scr.SCREEN_MIN_X)
          68
69
```