APS106 – Lab #9

Preamble

This week you will practice defining and using custom classes by writing a program to analyze the placement of wind turbines in a wind farm.

Deliverables

1. A Python file named lab9.py with your functions.

For this lab, you will implement two classes: `Circle` and `WindTurbine` as well as a function that uses `WindTurbine` objects.

For the `Circle` class, you will implement the following methods:

- `move`

For the `WindTurbine` class, you will implement the following methods

- `move`
- `overlap`
- `validate_placement`

You will write the following functions that analyze `WindTurbine` objects:

- `load_turbine_placements`
- `check_turbine_placements`

Five test cases are provided on MarkUs to help you prepare your solution. **Passing all these test cases does not guarantee your code is correct.** You will need to develop your own test cases to verify your solution works correctly. Your programs will be graded using ten secret test cases. These test cases will be released after the assignment deadline.

IMPORTANT:

- Do not change the file name, function names, class names, or method names
- Do not use input() inside your program

Problem

This week we will returning to the problem from lab #4 where we interested in checking whether the proposed locations of wind turbines for a wind farm conflicted with the placements of other turbines. We modelled these turbines as a circle and wrote a function to detect when two circles were overlapping. This week we will be extending this exercise by writing a program that will represent multiple wind turbines and their proposed placement. The program will be able to check whether any of the wind turbine placements are invalid due to turbines having overlapping placement areas.

You will complete this lab in four parts. In the first part, you will complete a `Circle` class that will be used to represent the placement of our wind turbines. In the second part, you will complete the `WindTurbine` class. In the third part, you will read a csv file that contains the attributes of different turbines and creates a list of turbine objects. Finally, in the third part, you will write a function, `check_turbine_placements`, that will analyze a list of `WindTurbine` objects and identify any turbines that have overlapping placements.

Part 0 – Point Class

For this lab, we will utilize a simple `Point` class to represent points in two-dimensional space. `Point` objects have two integer attributes x and y which represent the x- and y-coordinates of a point on a two-dimensional plane. This class has no methods other than the constructor (`__init__`) and `__str__` method. You do **not** need to make any modifications to this class.

Part 1 – Circle Class

In this part, you will complete the `Circle` class that will be used to represent the size and placement of circular areas on a two-dimensional coordinate plane. Our `Circle` objects will have two attributes named `centre` and `radius` which are a `Point` and an int, respectively. These attributes represent the centre and radius of a circle. As a refresher from lab #4, a circle can be completely defined by its centre coordinate and a radius (figure 1).

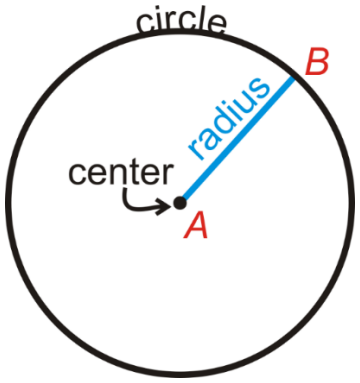

Figure 1. A circle can be defined with its centre coordinate and its radius.

The `Circle` class has four methods:

- `__init__` (constructor)
- `__str__`
- `overlap`
- `move`

We have written the `__init__`, `__str__`, and `overlap` methods for you. You should review these methods and understand how they work.

For this part of the lab, you will need to complete the `move` method. This method changes the placement of a Circle object by moving the centre coordinate by specified distances along the x and y axes. The inputs to this method are as follows:

- `self` – The Circle object to move
- `horizontal_translation` – an integer specifying how many units to move the Circle along the x-axis. The value can be positive (move right) or negative (move left).
- `vertical_translation` - an integer specifying how many units to move the Circle along the y-axis. The value can be positive (move up) or negative (move down).

The method should return `None`. The method should **not** create a new Point object.

Example usage

```
>>> c1 = Circle(0,1,4)
>>> print(c1)
Circle with centre coordinate (0,1) and radius 4
>>> c1.move(3, -9) # move the circle 3 to the right and 9 units down
>>> print(c1)
Circle with centre coordinate (3,-8) and radius 4
```

Part 2 – WindTurbine Class

In this part, you will complete the `WindTurbine` class. `WindTurbine` objects have three attributes:

- `id_number` – an integer identifying the wind turbine
- `placement` – a Circle representing the proposed placement of the turbine
- `overlapping_turbines` – a **list** of other `WindTurbine` objects whose placements overlap with the `WindTurbine` object

This class has five methods:

- `__init__`
- `__str__`
- `move`
- `overlap`
- `validate_placement`

The `__init__` and `__str__` methods are provided for you and do not require modification for this lab. Note that when a `WindTurbine` object is created, the `overlapping_turbines` list attribute is initialized to an empty list. We will add turbines to this list when executing the `validate_placement` method.

Part 2.1 – Move method

The fist method you will implement is the `move` method. Calling this this method will change the placement of the wind turbine. The inputs to the method are:

- `self` – The WindTurbine object to move
- `horizontal_translation` – an integer specifying how many units to move the wind turbine along the x-axis. The value can be positive (move right) or negative (move left).
- `vertical_translation` - an integer specifying how many units to move the wind turbine along the y-axis. The value can be positive (move up) or negative (move down).

This method should call the `move` method from the `Circle` class. The method should also reset the `overlapping_turbines` attribute to be an empty list (since we are moving the turbine, we will need to run the `validate_placement` method again to find any overlapping turbines).

Hint #1: this method should call the move method from the Circle class.

Hint #2: this method can be written in two lines of code.

Hint #3: this method should not create a new Circle object.

Example usage:

```
>>> t1 = WindTurbine(1, 4, 5, 10)
>>> print(t1)
Wind Turbine ID: 1, Placement: Circle with centre coordinate (4,5) and radius
10
>>> t1.move(-1,4) # move one unit to the left and 4 units up
>>> print(t1)
Wind Turbine ID: 1, Placement: Circle with centre coordinate (3,9) and radius
10
```

Part 2.2 – Overlap method

The next method you will implement is the `overlap` method. Calling this this method should detect whether the placements of two `WindTurbine` objects overlap. The method returns a boolean indicating whether the placements of the two input turbine objects have overlapping placements. The inputs to this method are:

- `self` – The first `WindTurbine` object
- `turbineB` – The second `WindTurbine` object

Hint: The method should call the overlap method from the `Circle` class. The method can be written in one line of code.

Part 2.3 – validate_placement method

The `validate_placement` method checks if a `WindTurbine` object's proposed placement overlaps with any other `WindTurbine` object's placement. The inputs to this function are:

- `self` - The `WindTurbine` object whose placement is being validated
- `turbines` – a list of `WindTurbine` objects

The method should check for overlap between the "`self`" wind turbine object and each of the turbine objects within the `turbines` input parameter list. All `WindTurbine` objects from the `turbines` list that overlap should be appended to the "`self`" wind turbine's `overlapping_turbines` attribute list.

Note if the "`self`" turbine object is included in the `turbines` input list, it should **not** be added to the `overlapping_turbines` list. **Hint** you can use the `id_number` attribute to check if two objects refer to the same turbine.

Example Usage

```
>>> t1 = WindTurbine(1, 4, 5, 10)
>>> t2 = WindTurbine(2, 4, 6, 5)
>>> t3 = WindTurbine(3, -5, -9, 3)
>>> t4 = WindTurbine(4, 100, 2000, 44)
>>> turbine_list = [t1, t2, t3, t4]
>>> t1.validate_placement(turbine_list)
>>> print(len(t1.overlapping_turbines)) # print the number of turbines found
to overlap with t1
1
>>> print(t1.overlapping_turbines[0]) # print the turbine that overlaps
Wind Turbine ID: 2, Placement: Circle with centre coordinate (4,6) and radius
5
```

Part 3 – Create Turbines using csv file

In this part of the lab, you will write the `load_turbine_placements` function. This function takes in the name of csv file as a string as its only parameter and returns a list of `WindTurbine` objects. The csv file passed to the function will contain four columns containing an ID number, x-axis centre coordinate, y-axis centre-coordinate, and radius. You can assume that the values in each column will be integers. An example file is shown below.

ID Number	Centre Coordinate X	Centre Coordinate Y	Radius
1	8	44	12
52	-9	71	5

Your function will need to extract the values from each line in the file, create a `WindTurbine` object with the values, and then append that object to a list. Your function will then return that list of `WindTurbine` objects.

Note, you will need to ignore the first line of the csv file which contains the column headers. Two files tubines1.csv and turbines2.csv are provided to help you test and debug your code.

Part 4 – Check Turbine Placements Function

In this final part of the lab, you will write the `check_turbine_placements` function. This function takes a list of `WindTurbine` objects as an input and validates each turbine's proposed placement to check for overlaps with any other turbines in the input list. The function should return the number of turbines whose placement overlaps with at least one other turbine's placement. In other words, this function should count the number of turbines that have an invalid placement because it overlaps with one or more other turbines in the input list.

Example Usage

```
>>> t1 = WindTurbine(1, 4, 5, 10)
>>> t2 = WindTurbine(2, 4, 6, 5)
>>> t3 = WindTurbine(3, -5, -9, 3)
>>> t4 = WindTurbine(4, 100, 2000, 44)
>>> turbine_list = [t1, t2, t3, t4]
>>> num_invalid = check_turbine_placements(turbine_list)
>>> print(num_invalid)
2
```

The two invalid turbines are turbine 1 and turbine 2.
Hint: use a for loop and call the `validate_placement` method, then check the length of the `overlapping_turbines` attribute for each object.