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

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

For the Rectangle 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:

- `check_turbine_placements`

Use appropriate variable names and place comments throughout your program.

The name of the source file must be "lab9.py".

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 rectangle and wrote a function to detect when two rectangles 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 three parts. In the first part, you will complete a `Rectangle` class that will be used to represent the placement of our wind turbines. In the second part, you will complete the `WindTurbine` class. 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 – Rectangle Class

In this part, you will complete the `Rectangle` class that will be used to represent the size and placement of rectangular areas on a two-dimensional coordinate plane. Our rectangle objects will have two attributes named `bottom_left` and `top_right` which are both `Point` objects that represent the bottom left corner and top right corner coordinates of the rectangle. As a refresher from lab #4, a rectangle can be completely defined by these two corner coordinates (figure 1).

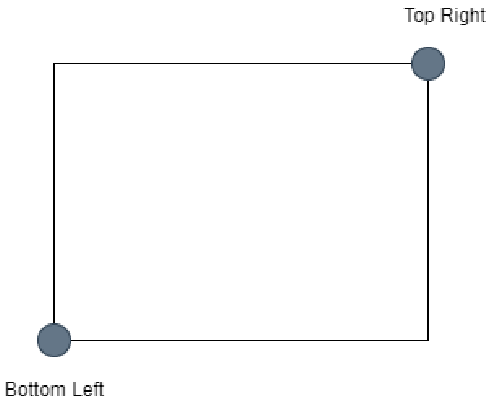

Figure 1. A rectangle can be defined with two non-adjacent corners. In this case, we are given the bottom left and top right corners. Because the angles at each corner are 90°, the other two points can be calculated using the given points.

The `Rectangle` 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 rectangle object by moving both corner coordinate by specified distances along the x and y axes. The inputs to this method are as follows:

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

The method should return `None`.

Example usage

```
>>> r1 = Rectangle(1,2,5,9)
>>> print(r1)
Rectangle with corner coordinates (1,2), (5,9)
>>> r1.move(-1,4)
>>> print(r1)
Rectangle with corner coordinates (0,6), (4,13)
>>> r1.bottom_left.x
0
>>> r1.bottom_left.y
6
>>> r1.top_right.x
4
>>> r1.top_right.y
13
```

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 rectangle 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 `Rectangle` 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).

Example usage:

```
>>> t1 = WindTurbine(1, 1, 2, 5, 9)
>>> print(t1)
Wind Turbine ID: 1, Placement: Rectangle with corner coordinates (1,2), (5,9)
>>> t1.move(-1,4)
>>> print(t1)
Wind Turbine ID: 1, Placement: Rectangle with corner coordinates (0,6),
(4,13)
>>> print(t1.placement)
Rectangle with corner coordinates (0,6), (4,13)
```

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

The method should call the overlap method from the `Rectangle` class.

Part 2.3 – validate_placement method

The `validate_positon` 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, 1, 2, 5, 9)
>>> t2 = WindTurbine(2, 0, 2, 4, 10)
>>> t3 = WindTurbine(3, 11, 12, 15, 29)
>>> t1.validate_placement([t1, t2, t3])
>>> len(t1.overlapping_turbines)
1
>>> t1.overlapping_turbines[0]
Wind Turbine ID: 2, Placement: Rectangle with corner coordinates: (0,2), 
(4,10)
```

Part 3 – 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.

Example Usage

```
>>> t1 = WindTurbine(1, 1, 2, 5, 9)
>>> t2 = WindTurbine(2, 0, 2, 4, 10)
>>> t3 = WindTurbine(3, 11, 12, 15, 29)
>>> t4 = WindTurbine(4, -1, 4, 1, 5)
>>> check_turbine_placements([t1, t2, t3, t4])
3
>>> len(t1.overlapping_turbines)
1
>>> t1.overlapping_turbines[0]
Wind Turbine ID: 2, Placement: Rectangle with corner coordinates: (0,2),
(4,10)
>>> len(t2.overlapping_turbines)
2
>>> t2.overlapping_turbines[0]
Wind Turbine ID: 1, Placement: Rectangle with corner coordinates: (1,2),
(5,9)
>>> t2.overlapping_turbines[1]
Wind Turbine ID: 4, Placement: Rectangle with corner coordinates: (-1,4),
(1,5)
>>> len(t3.overlapping_turbines)
0
>>> len(t4.overlapping_turbines)
1
>>> t4.overlapping_turbines[0]
Wind Turbine ID: 2, Placement: Rectangle with corner coordinates: (0,2),
(4,10)
```