CS 120: Introduction to Computer Programming II

Carcassonne Project, Short B

see the Short A spec for due dates

1 Overview

See the spec for Short A to see an overview for the entire multi-part project.

In this part, you will implement 4 new tile types (one is a little interesting - hopefully your Tile class can already handle it, without too much trouble). You will also implement a new function in the Map class, which traces a road to its end.

2 New Required Tiles (Short B)

In this part, all tiles from previous part(s) are still required, and we will add the tiles below.

Tile 13

Tile 14

This tile is different than Tile 09 in a critical way: the two portions of the city are not connected to each other. Make sure that `city_connects(N,E)` returns `False`.

Tile 15

Tile 16

(spec continues on next page)

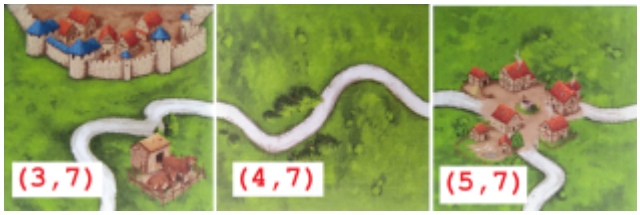
3 Tracing Roads

Other than adding 4 new tiles, this part of the Carcassonne project simply requires you to write one new method in the `CarcassonneMap`, named

```
trace_road_one_direction(self, x,y, side)
```

This method starts at a given point in the map, and follows a road until it ends (or loops). There are two ways that it can end: either by hitting a square where there is a missing tile, or because the road ends at a crossroads. A loop is simply a road which eventually returns to its starting location without hitting an end.

The parameters to this function are an (x, y) location and a side. The (x, y) location refers to a tile, and you can assume that the tile exists. You can also assume that the side in question has a road. For example, in the following picture, a valid starting point for this call might be `(3,7,E)`.



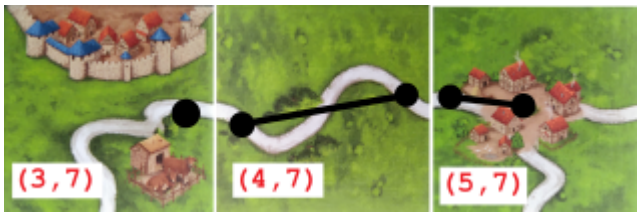
Your code will follow the road as if you are **leaving** the tile you started on. For each tile that you find (until you either find a hole, or the end of the road), you will report a 4-part tuple: the x,y location of the tile, the starting edge, and the ending edge. So the first tuple that you will discover is in tile $(4, 7)$, and it runs from the West to the East:

```
(4,7, W,E)
```

The next tile is $(5, 7)$; it runs from the West edge to the Center (which we represent with -1).

```
(5,7, W,CENTER)
```

This is the path that you found:



and you represent it by returning the following array of tuples:

```
[ (4,7,3,1), (5,7,3,-1) ]
```

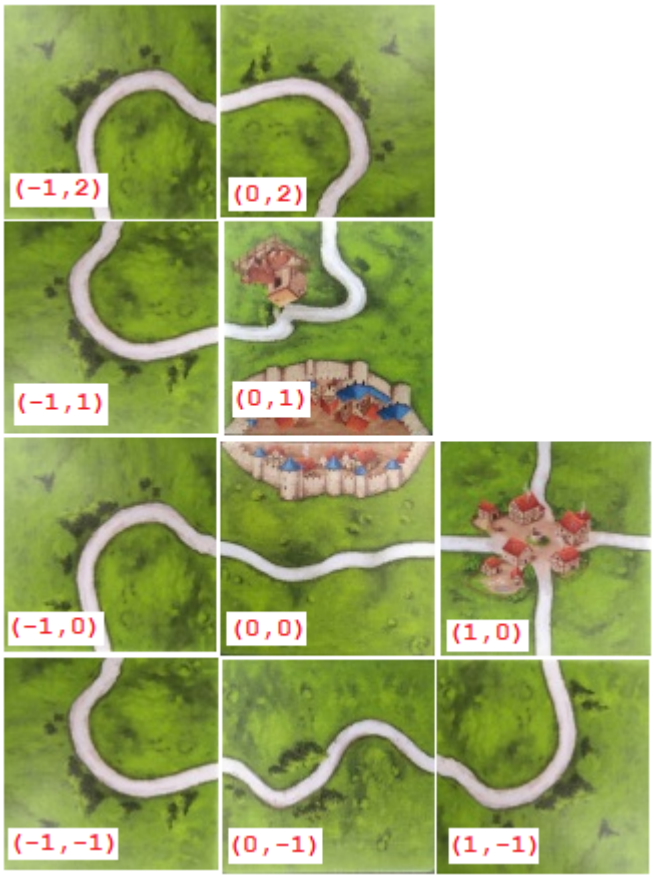
Notice, in the example I gave above, that the road also travels the other direction; it connects to $(3, 7)$ on the South side. However, you should **not** follow that for this function. (You'll write another function, in the next part of the project, which does this.)

Also notice that, in some circumstances, you may return an empty array. Suppose, given the same map we had above, that you are asked to start at $(5, 7, S)$. This is a valid tile, and it's a road edge, just as required - but the tile to the South is missing. Therefore, you would return an empty array, since you cannot find any segments.

3.1 Loops and Near-Loops

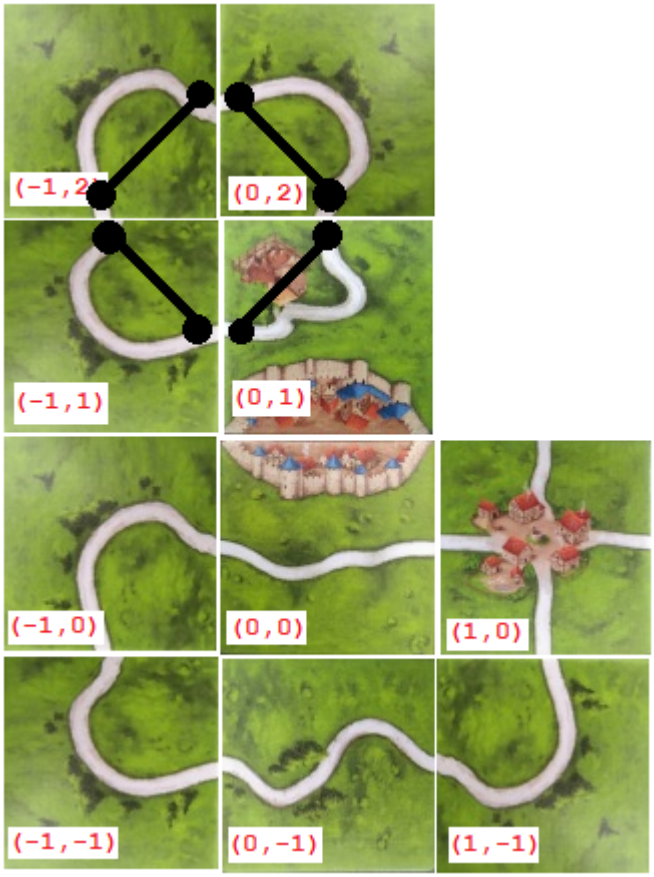
NOTE: Loops will be tested by the testcases, but will be a relatively small portion of your score. Focus on the simpler cases (the ones where the road terminates) first. Add loop detection only if you have time before the due date.

Consider the following map:



(spec continues on next page)

If you were to trace a road starting at `(0,1,N)`, then you would encounter a loop, as follows:

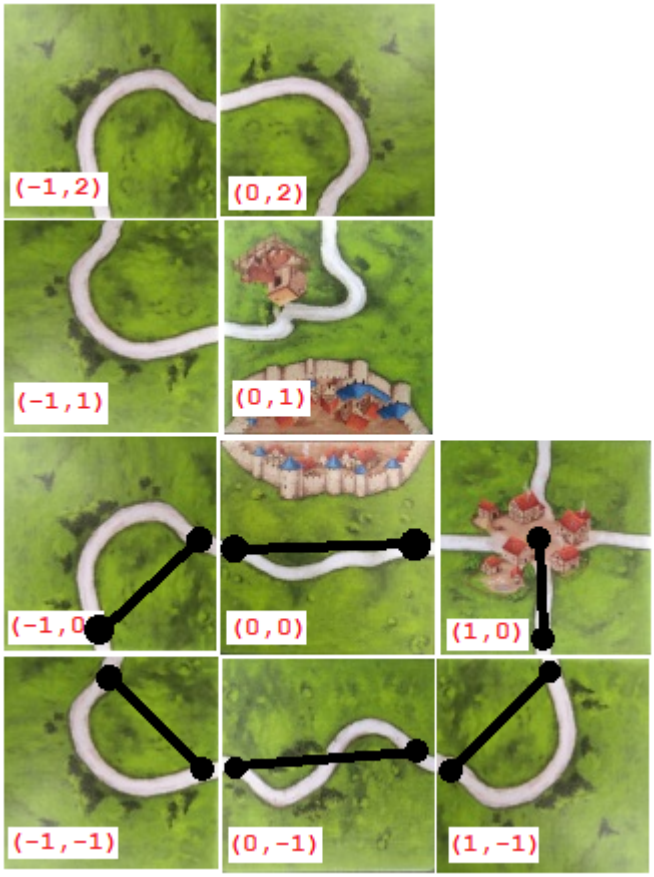


In this case, you would return information for all of the tiles that you traversed. You end when you return to your starting point:

```
[(0,2,S,W), (-1,2,E,S), (-1,1,N,E), (0,1,W,N)]
```

(spec continues on next page)

Of course, this is a loop because it doesn't hit any crossroads on the way. If we were to start at `(1,0,W)`, it would return to the original tile, but it would terminate there, not loop:




```
[(0,0,E,W), (-1,0,E,S), (-1,-1,N,E), (0,-1,W,E), (1,-1,W,N), (1,0,S,CENTER)]
```

4 Turning in Your Solution

You must turn in your code using GradeScope.