Description:

I want to create an graph of a map that the user is allowed to openly explore, kind of like a MUD game. I will challenge myself to build a traveral algorithm, that functions like a bot exploring all the rooms on the map in the minimal number of steps. The map will be an adjacency list containing room data.

Traveral Algorithm Goals

- **1**: Traverse the entire world without missing any rooms.
- **2**: Traverse the world in < 2000
- **3**: Traverse the world in < 1000

File Break Down:

adv.py: This is the application entry point and will basically function as the runtime. This is where we will instantiate a world, define out traversal algorithms, test these traversals visit every room atleast once, count the steps the algorithm took, and run a REPL to take user commands to traverse the world room by room.

world.py: This is the world object that contains the graph data by loading it in on instantiation

room.py: This will be the object that contains all of the room data including coordinates and an adjacency list of references to surounding rooms.

util.py: This will just store handy utilities I made like stacks and queues

player.py: This will be the player object that stores player data including their location on the map so current room data since I intend on allowing the players to independently enter commands via repl to traverse the map manually

Graph traversal structure as its being built:

Room 1 -> n

```
#{
#  <room>: {<Direction>: <Room>}
#}

{
  1: {'n': '?', 's': '?', 'w': '?', 'e': '?'}
}
```

Room 5 ->

```
{
  0: {'n': '?', 's': 5, 'w': '?', 'e': '?'},
```

```
  5: {'n': 0, 's': '?', 'e': '?'}
}
```

Stretch Problems

It is very difficult to calculate the shortest possible path that traverses the entire graph. Why?

This is the traveling salesman problem, it is np no compute considering their may not even possibly be one best solution. This is because we can not simply loop all of the rooms in a singular path where every room is only visited once. This means no matter what we do we have to go over rooms over and over again while traversing, i.e. back tracking before we can continue to look for new rooms. So even with my algorithm that is consistently < 950 moves to traverse 500 rooms, it will vary every time. Can you build a more effecient algorithm?