Hunter Leise

CSCI 3202

Problem Set 4

**Problem 4.1**

**Problem 4.2**

Code:

import random  
  
  
class City:  
 def \_\_init\_\_(self):  
 self.neighbors = [0] \* 6 + [1] \* 27 + [2] \* 27  
 random.shuffle(self.neighbors)  
 self.vacant = [i for i, x in enumerate(self.neighbors) if x == 0]  
  
 def iterate(self, iterations):  
 *“””  
 Iterate until all neighbors are satisfied or until the maximum  
 number of iterations is reached.  
  
 :param iterations: Maximum number of times the cycle will be run  
 :return: None  
 “””* for iteration in range(iterations):  
 if iteration % 20 == 0: *# print every 20 iterations* print(self.neighbors)  
  
 dissatisfied = []  
 for i in range(len(self.neighbors)):  
 if self.neighbors[i] != 0 and not self.is\_satisfied(i):  
 dissatisfied.append(i)  
 if len(dissatisfied) == 0: *# Yay! Everyone is satisfied!* print(**"all satisfied"**)  
 break  
 else: *# Move unsatisfied people to vacant house* rand\_dis = random.choice(dissatisfied)  
 self.neighbors[self.vacant[0]] = self.neighbors[rand\_dis]  
 self.neighbors[rand\_dis] = 0  
 self.vacant.pop(0)  
 self.vacant.append(rand\_dis)  
  
 def is\_satisfied(self, house):  
 *“””  
 Returns whether a given house is satisfied. In other words,  
 whether it has at least two neighbors of its own type within  
 two houses on either side.  
  
 :param house: index for the queried house  
 :return: True if satisfied, False if not satisfied  
 “””* same\_neighbors = 0  
 for i in range(1, 3):  
 if self.neighbors[(house + i) % len(self.neighbors)] \  
 == self.neighbors[house]:  
 same\_neighbors += 1  
 if self.neighbors[(house - i) % len(self.neighbors)] \  
 == self.neighbors[house]:  
 same\_neighbors += 1  
 return same\_neighbors >= 2  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 city = City()  
 city.iterate(400)  
 print(city.neighbors) *# Print final neighborhood*

Example Output:

[1, 1, 1, 2, 0, 0, 1, 2, 2, 2, 1, 1, 1, 2, 1, 2, 0, 1, 1, 0, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 0, 1, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 2, 0, 2, 1, 2, 1, 1, 2, 2, 1, 2]  
  
[1, 1, 1, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 0, 1, 1, 1, 0, 1, 2, 1, 1, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 1, 0, 1, 0, 0, 2, 1, 2]  
  
[1, 1, 1, 0, 1, 0, 0, 2, 2, 2, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 0, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 1, 1]  
  
all satisfied   
  
[1, 1, 1, 0, 0, 0, 0, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1]

The ring city does reach a totally satisfied state, and in surprisingly few iterations! This generally looks like a few very large groupings, such as the large grouping of two near the end of the example above, combined with a few smaller groupings.