

# Program 1: Markov Chains

Eric Wolfe

Dr. Mayer

CS 490/CS 525

## Source Code

### Main.java

```
1 public class Main {
2
3     public static void main(String[] args) {
4         System.out.println("Starting tests: ");
5         // Experiment 1
6         new MarkovChain(0, 30, "test1.csv");
7         System.out.print("-");
8         new MarkovChain(1, 30, "test2.csv");
9         System.out.print("-");
10        new MarkovChain(2, 30, "test3.csv");
11        System.out.print("-");
12        new MarkovChain(3, 30, "test4.csv");
13        System.out.print("-");
14        // Experiment 2
15        new MarkovChain(100, "test5.csv");
16        System.out.print("-");
17        new MarkovChain(1000, "test6.csv");
18        // Experiment 3
19        System.out.print("-");
20        new MarkovChain(1000, "test7.csv");
21        System.out.print("-");
22        new MarkovChain(1000, "test8.csv");
23        System.out.print("-");
24        new MarkovChain(1000, "test9.csv");
25        System.out.print("-");
26        new MarkovChain(1000, "test10.csv");
27        System.out.print("-");
28        new MarkovChain(1000, "test11.csv");
29        System.out.print("-");
30        new MarkovChain(1000, "test12.csv");
31        System.out.print("-");
32        new MarkovChain(1000, "test13.csv");
33        System.out.print("-");
```

```

34         new MarkovChain(1000, "test14.csv");
35         System.out.print("-");
36         new MarkovChain(1000, "test15.csv");
37         System.out.print("-");
38         new MarkovChain(1000, "test16.csv");
39         System.out.print("-:\n Finished tests");
40     }
41
42 }

```

## MarkovChain.java

```

1  import java.io.IOException;
2  import java.util.Random;
3  import java.io.PrintWriter;
4  import java.io.FileWriter;
5
6  public class MarkovChain {
7
8      private double[][] transitionMatrix = new double[][]{
9          {0.8, 0.2, 0.0, 0.0},
10         {0.1, 0.8, 0.1, 0.0},
11         {0.1, 0.0, 0.7, 0.2},
12         {0.1, 0.0, 0.0, 0.9}
13     };
14     private double totalWalks = 1.0;
15     private double gCount = 0.0;
16     private double yCount = 0.0;
17     private double oCount = 0.0;
18     private double rCount = 0.0;
19     private String fileName;
20
21     /*
22     * Constructor that will run a walk with a random initial state
23     */
24     public MarkovChain(int numberOfSteps, String fileName){
25         this.fileName = fileName;
26         Random random = new Random();
27         int initial = random.nextInt(3);
28         run(initial, numberOfSteps);
29     }
30
31     /*
32     * Constructor that will run a walk with a known initial state
33     */
34     public MarkovChain(int startPosition, int numberOfSteps, String fileName){
35         this.fileName = fileName;
36         run(startPosition, numberOfSteps);
37     }
38
39     /*

```

```

40 * Function for going through the steps and recording the ratio of times reached a state
41 */
42 private void run(int startPosition, int numberOfSteps){
43     double[] gArray = new double[numberOfSteps];
44     double[] yArray = new double[numberOfSteps];
45     double[] oArray = new double[numberOfSteps];
46     double[] rArray = new double[numberOfSteps];
47     String[] stepArray = new String[numberOfSteps];
48     for(int i = 0; i < numberOfSteps; i++, totalWalks++){
49         startPosition = walk(startPosition);
50         switch (startPosition){
51             case 0:
52                 gCount++;
53                 stepArray[i] = "G";
54                 break;
55             case 1:
56                 yCount++;
57                 stepArray[i] = "Y";
58                 break;
59             case 2:
60                 oCount++;
61                 stepArray[i] = "O";
62                 break;
63             case 3:
64                 rCount++;
65                 stepArray[i] = "R";
66                 break;
67         }
68         gArray[i] = gCount / totalWalks;
69         yArray[i] = yCount / totalWalks;
70         oArray[i] = oCount / totalWalks;
71         rArray[i] = rCount / totalWalks;
72     }
73     printToCSV(gArray, yArray, oArray, rArray, stepArray);
74 }
75
76 /*
77 * Function for deciding where the next walk will be
78 */
79 private int walk(int currentState){
80     // Variable for return
81     int nextState = 0;
82     // Random number from figuring out where to walk to next
83     Random random = new Random();
84     double nextProbability = random.nextDouble();
85     // Switch case values
86     double case0 = transitionMatrix[currentState][0];
87     double case1 = transitionMatrix[currentState][1] + transitionMatrix[currentState]
88 [0];
89     double case2 = transitionMatrix[currentState][2] + transitionMatrix[currentState][0]
90 + transitionMatrix[currentState][1];

```

```

89         double case3 = transitionMatrix[currentState][3] + transitionMatrix[currentState][0]
+ transitionMatrix[currentState][3] + transitionMatrix[currentState][2];
90         // Check where to walk next
91         if (nextProbability > 0 && nextProbability <= case0){
92             nextState = 0;
93         } else if (nextProbability > case0 && nextProbability <= case1) {
94             nextState = 1;
95         } else if (nextProbability > case1 && nextProbability <= case2) {
96             nextState = 2;
97         } else if (nextProbability > case2 && nextProbability <= case3) {
98             nextState = 3;
99         } else {
100             System.out.println("There was an error");
101             System.exit(1);
102         }
103         // Return the next walk location
104         return nextState;
105     }
106
107     /*
108     * Function for printing the results of the walk to a csv file
109     */
110     private void printToCSV(double[] array0, double[] array1, double[] array2, double[]
array3, String[] array4){
111         try {
112             FileWriter fileWriter = new FileWriter(fileName);
113             PrintWriter printWriter = new PrintWriter(fileWriter);
114             for (int i = 0; i < 5; i++) {
115                 for (int j = 0; j < array0.length; j++) {
116                     switch (i){
117                         case 0:
118                             printWriter.printf("%.2f,", array0[j]);
119                             break;
120                         case 1:
121                             printWriter.printf("%.2f,", array1[j]);
122                             break;
123                         case 2:
124                             printWriter.printf("%.2f,", array2[j]);
125                             break;
126                         case 3:
127                             printWriter.printf("%.2f,", array3[j]);
128                             break;
129                         case 4:
130                             printWriter.printf("%s,", array4[j]);
131                     }
132                 }
133                 printWriter.println();
134             }
135             printWriter.close();
136             fileWriter.close();
137         } catch (IOException e){

```

```

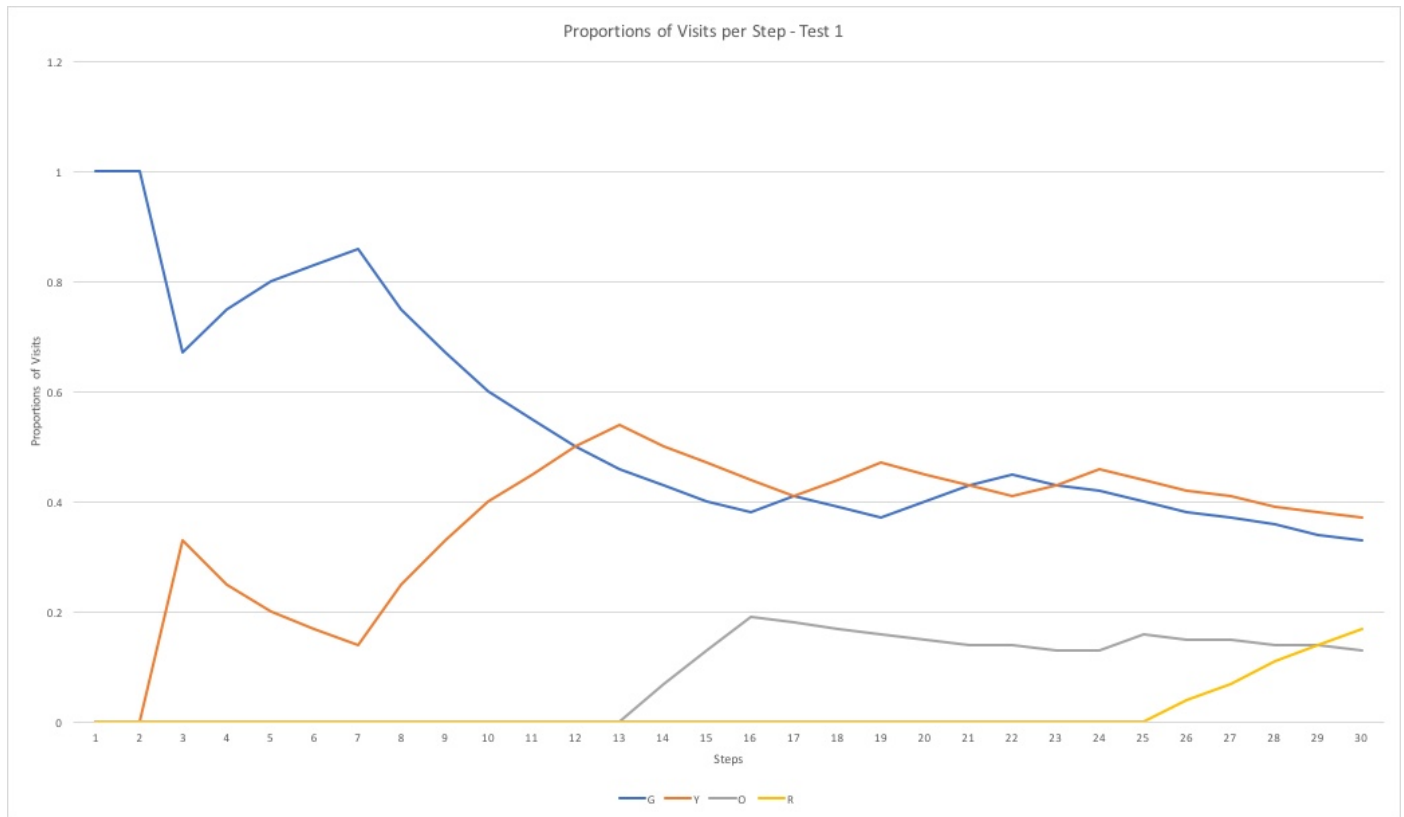
138         System.out.println(e);
139     }
140 }
141 }

```

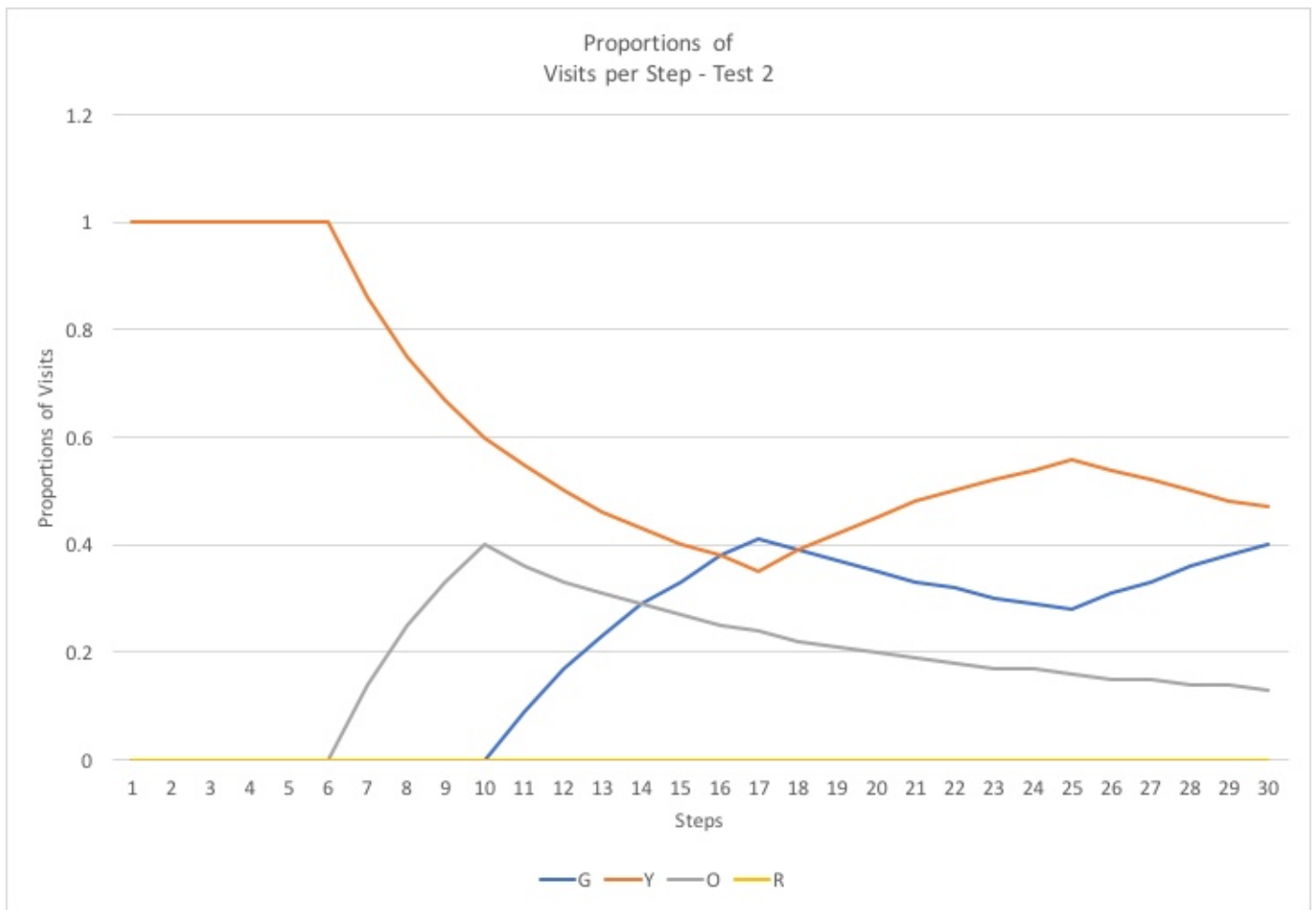
# Charts

## Experiment 1

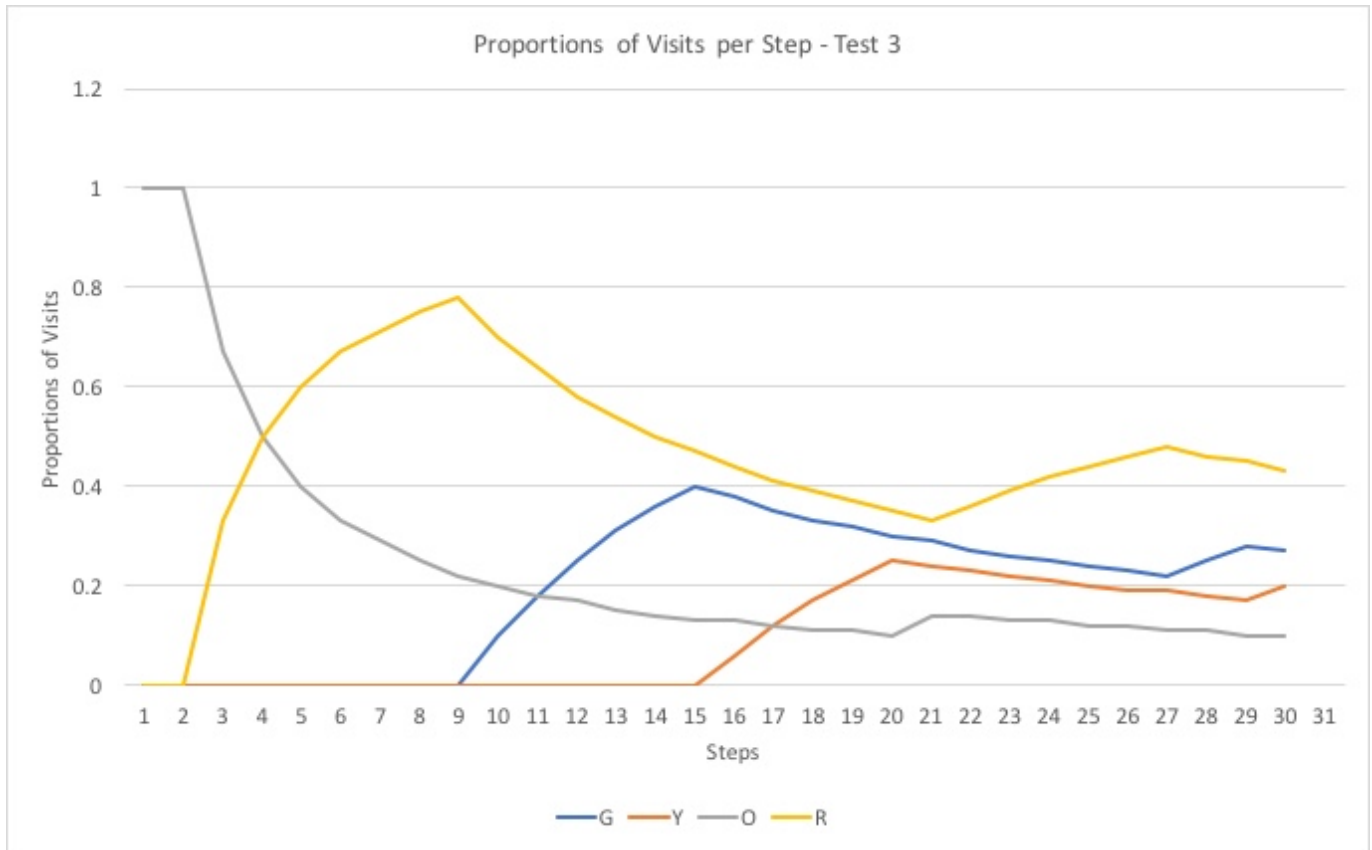
### Test 1 (Initial state G)



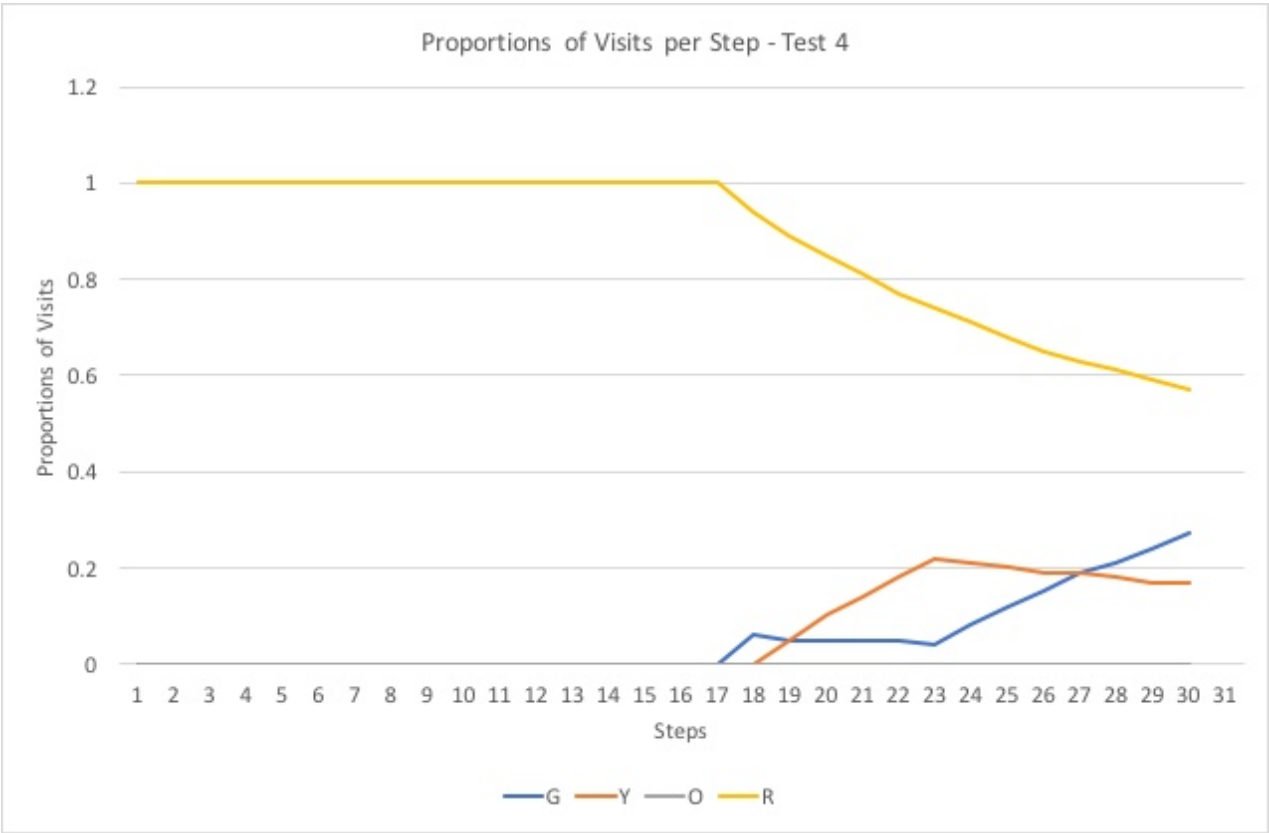
### Test 2 (Initial state Y)



**Test 3 (Initial state O)**

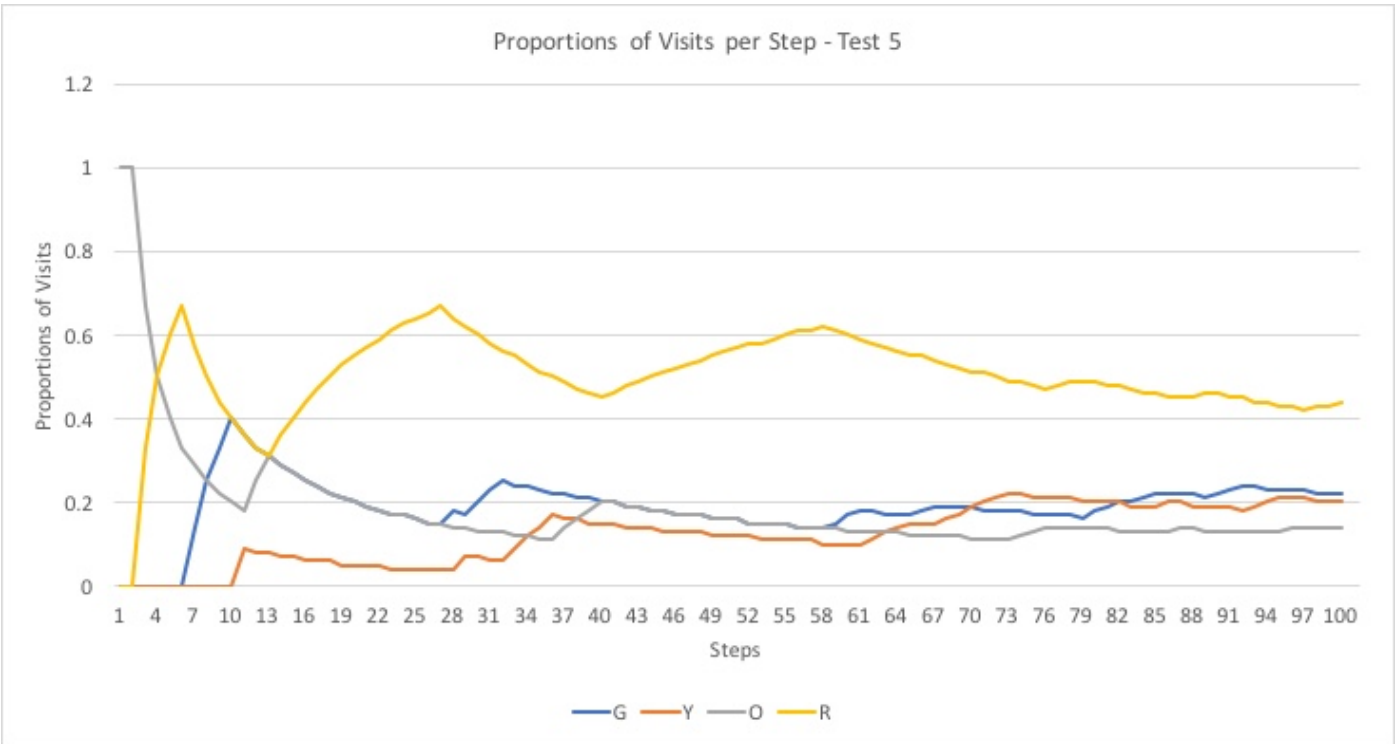


Test 4 (Initial state R)

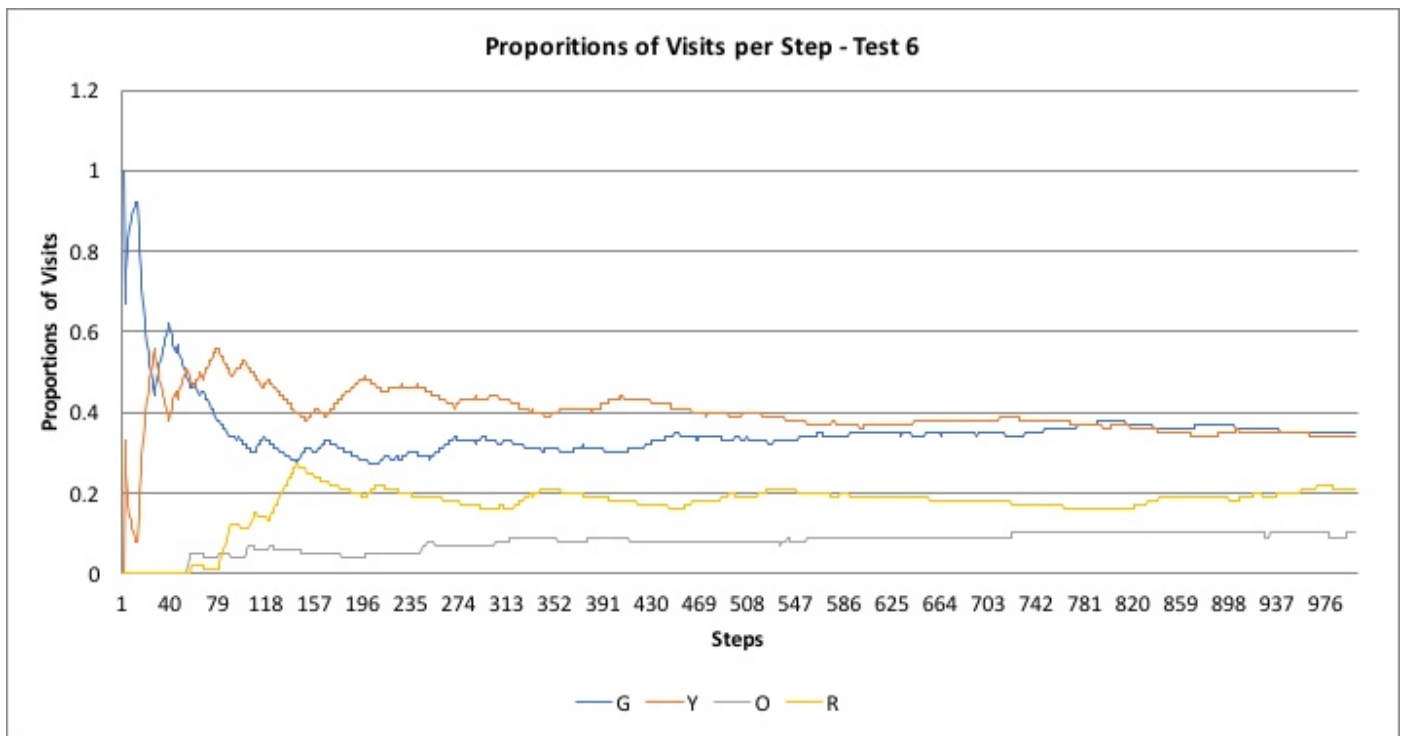


Experiment 2

Test 5 (Random initial state - 100 steps)

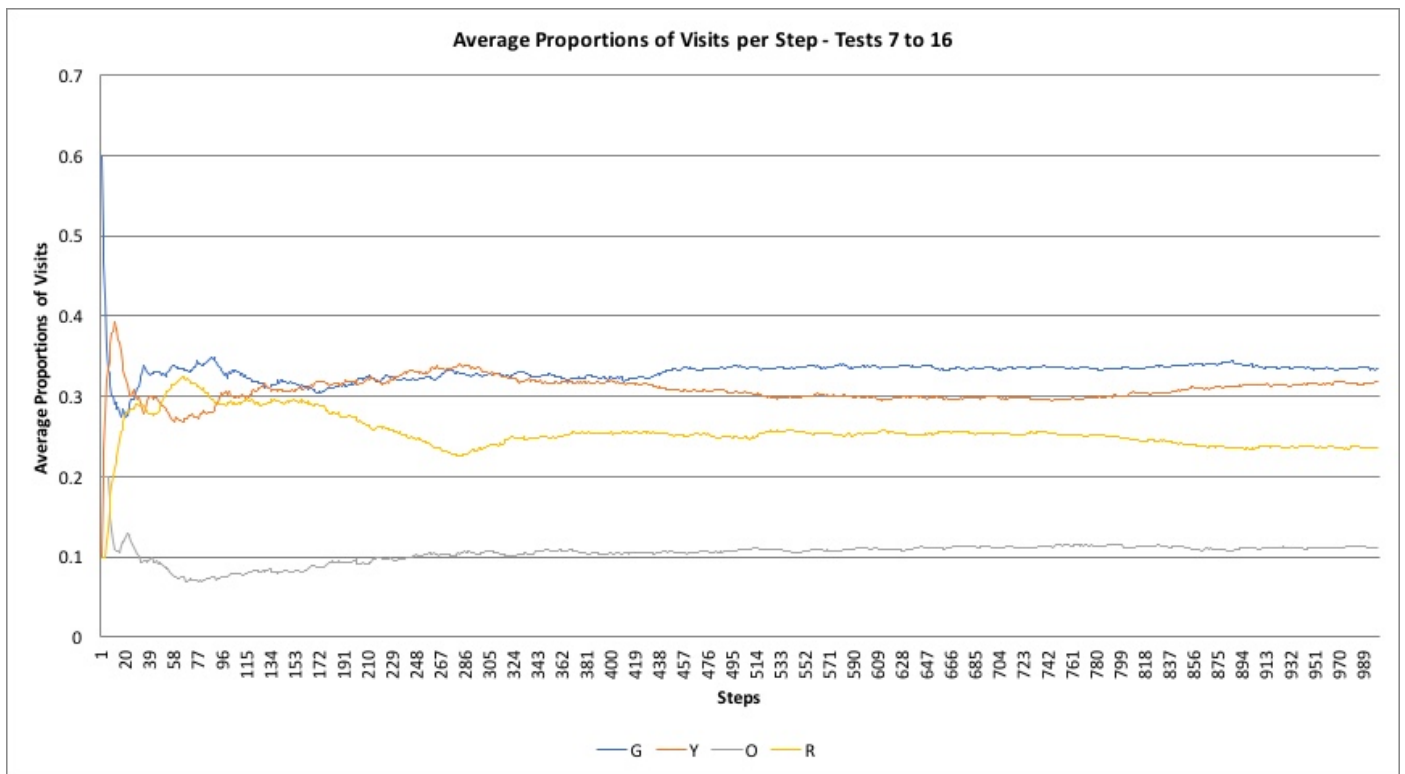


Test 6 (Random initial state - 1000 steps)



## Experiment 3

### Tests 7 - 16 (Average proportions of visits per steps)



## Results

Results are available in the Results.xlsx file and the csv files are in the submission folder after they are generated by the program.



For experiment 2, the results show that R has a higher proportion of visits in the 100 step run (test 5) compared to the other state even though it only has two states that it can go to (R and G). This is because it got stuck in the R state for quite a while. It isn't a clear representation of the actual steady state. However, in the 1000 step run (test 6) R has a lower proportion of visits than test 5 which is a better representation when compared to the actual steady state. It is pretty clear that running the test more times will get a better state and this experiment shows it.

Looking at the other states values for G, Y, and O. They are closer to their actual steady state values but were still off for test 5. Test 6 was a better representation of the actual values. I think a good test would be to run this experiment with 100 steps and 500 steps and see if it is still closer to the steady state values or not.