Program 1: Markov Chains

Eric Wolfe
Dr. Mayer
CS 490/CS 525

Source Code

Main.java

```java
public class Main {

    public static void main(String[] args) {
        System.out.println("Starting tests: ");
        // Experiment 1
        new MarkovChain(0, 30, "test1.csv");
        System.out.print(":-");
        new MarkovChain(1, 30, "test2.csv");
        System.out.print("-");
        new MarkovChain(2, 30, "test3.csv");
        System.out.print("-");
        new MarkovChain(3, 30, "test4.csv");
        System.out.print("-");
        // Experiment 2
        new MarkovChain(100, "test5.csv");
        System.out.print("-");
        new MarkovChain(1000, "test6.csv");
        // Experiment 3
        System.out.print("-");
        new MarkovChain(1000, "test7.csv");
        System.out.print("-");
        new MarkovChain(1000, "test8.csv");
        System.out.print("-");
        new MarkovChain(1000, "test9.csv");
        System.out.print("-");
        new MarkovChain(1000, "test10.csv");
        System.out.print("-");
        new MarkovChain(1000, "test11.csv");
        System.out.print("-");
        new MarkovChain(1000, "test12.csv");
        System.out.print("-");
        new MarkovChain(1000, "test13.csv");
        System.out.print("-");
        new MarkovChain(1000, "test14.csv");
        System.out.print("-");
        new MarkovChain(1000, "test15.csv");
        System.out.print("-");
        new MarkovChain(1000, "test16.csv");
        System.out.print("-:\n Finished tests");
    }

}
```

MarkovChain.java

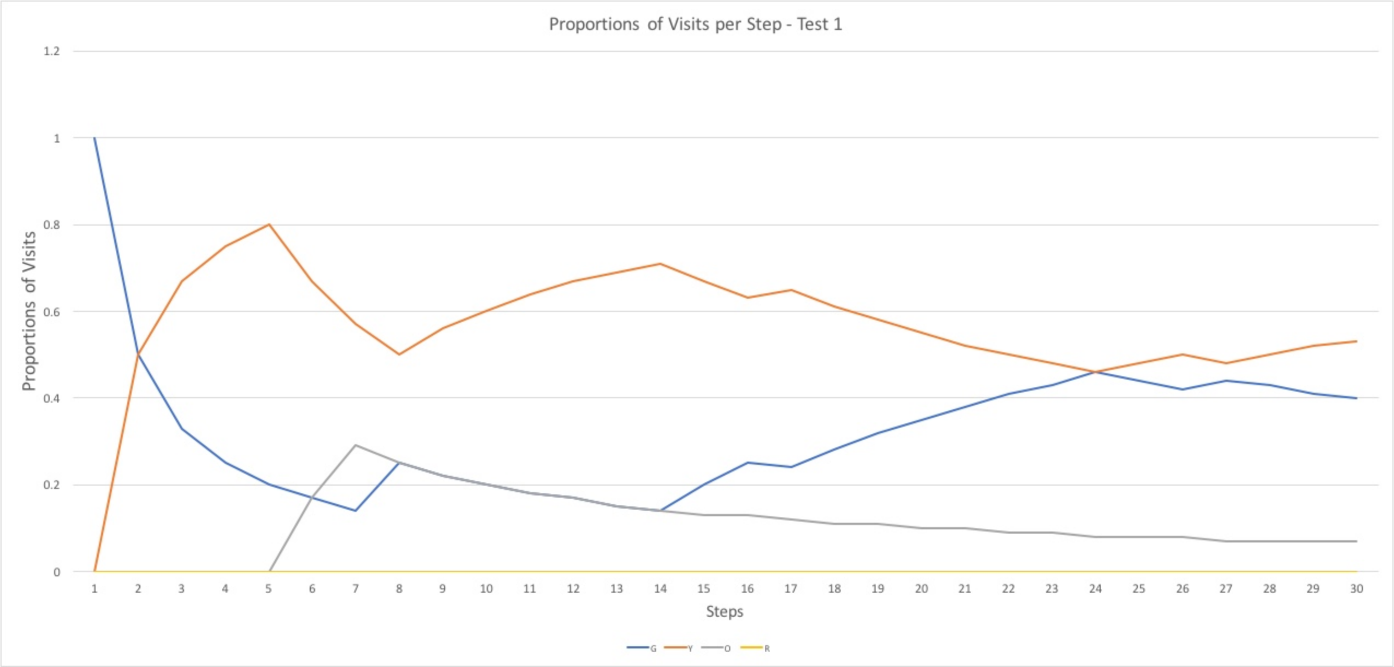
```
import java.io.IOException;
import java.util.Random;
import java.io.PrintWriter;
import java.io.FileWriter;

public class MarkovChain {

    private double[][] transitionMatrix = new double[][]{
            {0.8, 0.2, 0.0, 0.0},
            {0.1, 0.8, 0.1, 0.0},
            {0.1, 0.0, 0.7, 0.2},
            {0.1, 0.0, 0.0, 0.9}
    };
    private double totalWalks = 1.0;
    private double gCount = 0.0;
    private double yCount = 0.0;
    private double oCount = 0.0;
    private double rCount = 0.0;
    private String fileName;

    /*
    * Constructor that will run a walk with a random initial state
    */
    public MarkovChain(int numberOfSteps, String fileName){
        this.fileName = fileName;
        Random random = new Random();
        int initial = random.nextInt(3);
        run(initial, numberOfSteps);
    }

    /*
    * Constructor that will run a walk with a known initial state
    */
    public MarkovChain(int startPosition, int numberOfSteps, String fileName){
        this.fileName = fileName;
        run(startPosition, numberOfSteps);
    }

    /*
```

```java
    * Function for going through the steps and recording the ratio of times reached a state
     */
    private void run(int startPosition, int numberOfSteps){
        double[] gArray = new double[numberOfSteps];
        double[] yArray = new double[numberOfSteps];
        double[] oArray = new double[numberOfSteps];
        double[] rArray = new double[numberOfSteps];
        Random random = new Random();
        String[] stepArray = new String[numberOfSteps];
        for(int i = 0; i < numberOfSteps; i++, totalWalks++){
            startPosition = walk(startPosition, random);
            switch (startPosition){
                case 0:
                    gCount++;
                    stepArray[i] = "G";
                    break;
                case 1:
                    yCount++;
                    stepArray[i] = "Y";
                    break;
                case 2:
                    oCount++;
                    stepArray[i] = "O";
                    break;
                case 3:
                    rCount++;
                    stepArray[i] = "R";
                    break;
            }
            gArray[i] = gCount / totalWalks;
            yArray[i] = yCount / totalWalks;
            oArray[i] = oCount / totalWalks;
            rArray[i] = rCount / totalWalks;
        }
        printToCSV(gArray, yArray, oArray, rArray, stepArray);
    }

    /*
    * Function for deciding where the next walk will be
     */
    private int walk(int currentState, Random random){
        // Variable for return
        int nextState = 0;
        // Random number from figuring out where to walk to next
        double nextProbability = random.nextDouble();
        // Switch case values
        double case0 = transitionMatrix[currentState][0];
        double case1 = transitionMatrix[currentState][1] + transitionMatrix[currentState][0];
        double case2 = transitionMatrix[currentState][2] + transitionMatrix[currentState][0] + transitionMatrix[currentState][1];
        double case3 = transitionMatrix[currentState][3] + transitionMatrix[currentState][0] + transitionMatrix[currentState][3] + transitionMatrix[currentState][2];
        // Check where to walk next
        if (nextProbability > 0 && nextProbability <= case0){
            nextState = 0;
        } else if (nextProbability > case0 && nextProbability <= case1) {
            nextState = 1;
        } else if (nextProbability > case1 && nextProbability <= case2) {
            nextState = 2;
        } else if (nextProbability > case2 && nextProbability <= case3) {
            nextState = 3;
        } else {
            System.out.println("There was an error");
            System.exit(1);
        }
        // Return the next walk location
        return nextState;
    }

    /*
    * Function for printing the results of the walk to a csv file
     */
    private void printToCSV(double[] array0, double[] array1, double[] array2, double[] array3, String[] array4){
        try {
            FileWriter fileWriter = new FileWriter(fileName);
            PrintWriter printWriter = new PrintWriter(fileWriter);
            for (int i = 0; i < 5; i++) {
                for (int j = 0; j < array0.length; j++) {
                    switch (i){
                        case 0:
                            printWriter.printf("%.2f,", array0[j]);
                            break;
                        case 1:
                            printWriter.printf("%.2f,", array1[j]);
                            break;
                        case 2:
                            printWriter.printf("%.2f,", array2[j]);
                            break;
                        case 3:
                            printWriter.printf("%.2f,", array3[j]);
                            break;
                        case 4:
                            printWriter.printf("%s,", array4[j]);
                    }
                }
                printWriter.println();
            }
            printWriter.close();
            fileWriter.close();
        } catch (IOException e){
            System.out.println(e);
        }
    }
}
```

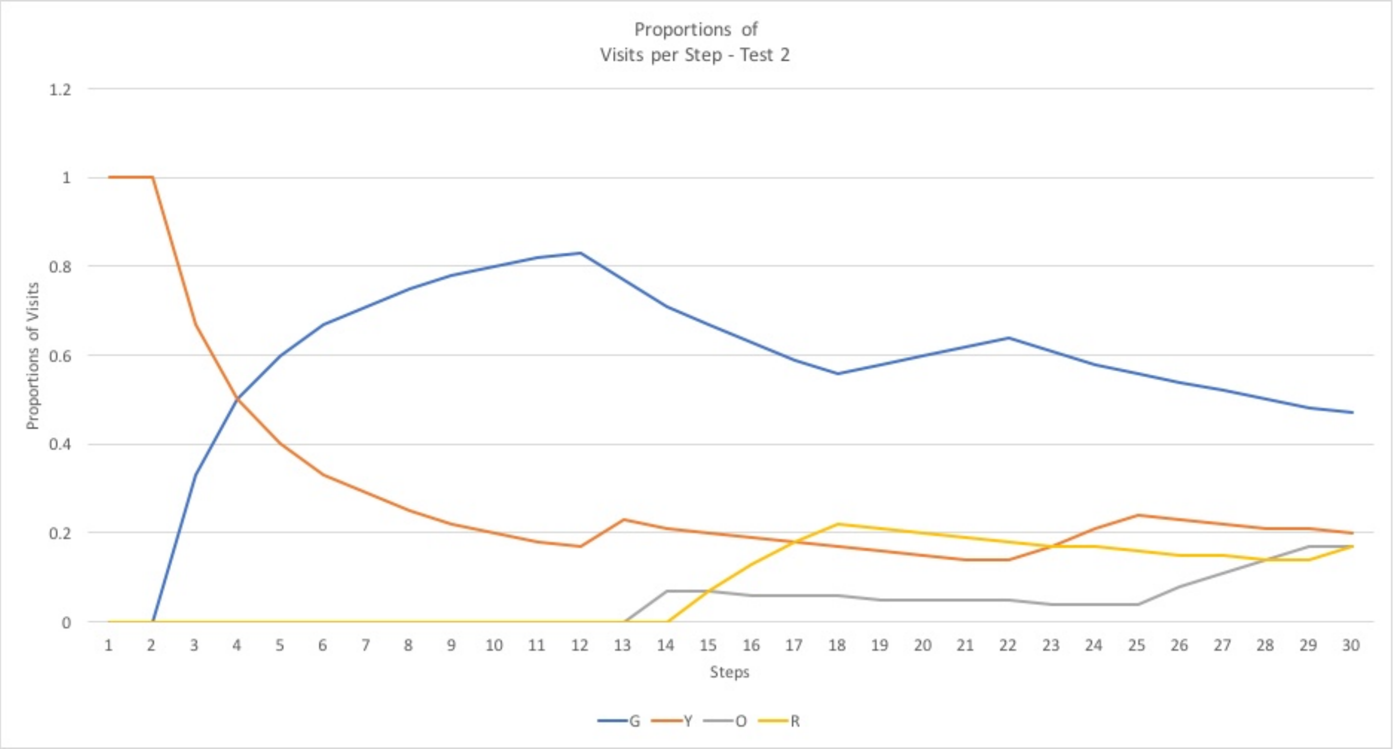
Charts

Experiment 1

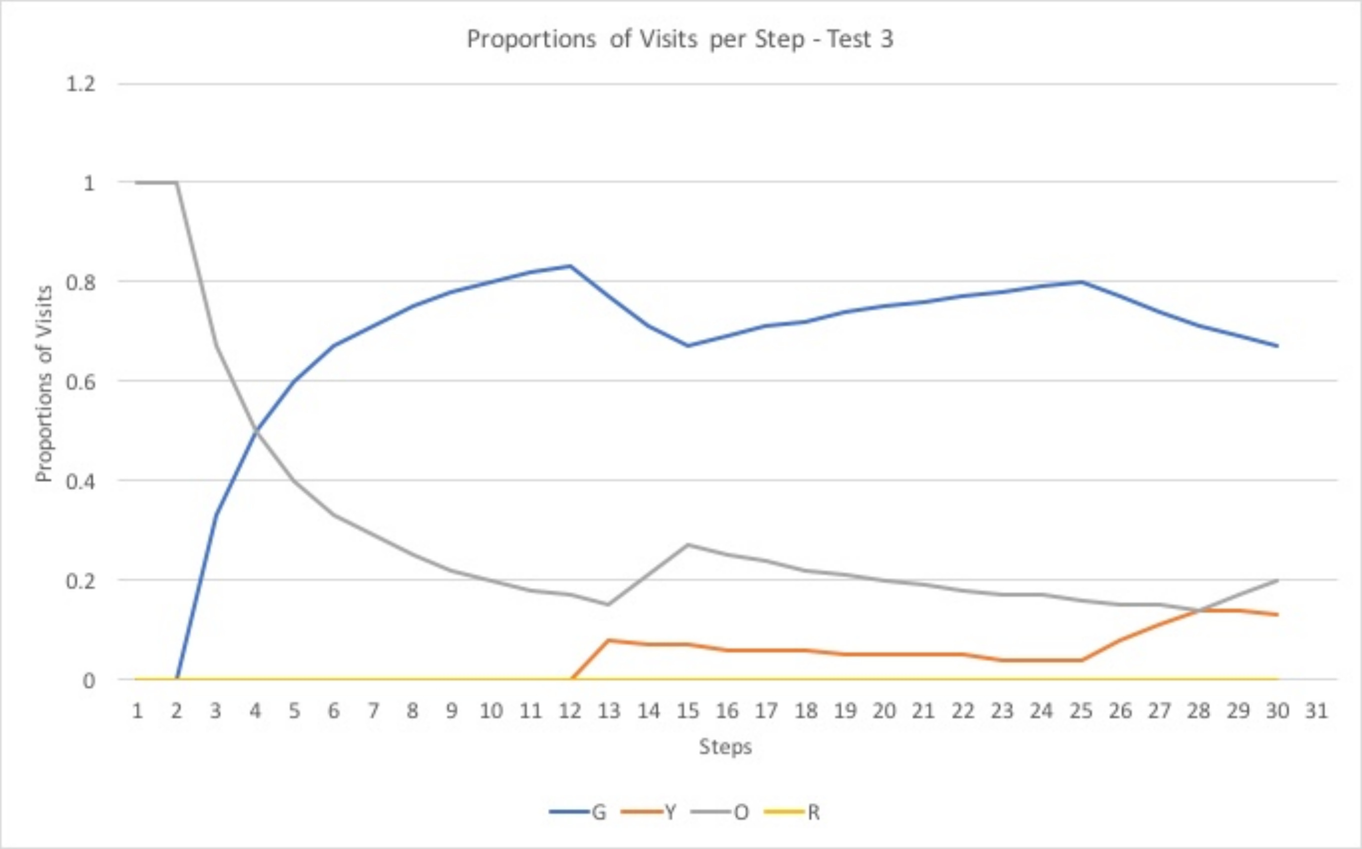
Test 1 (Initial state G)



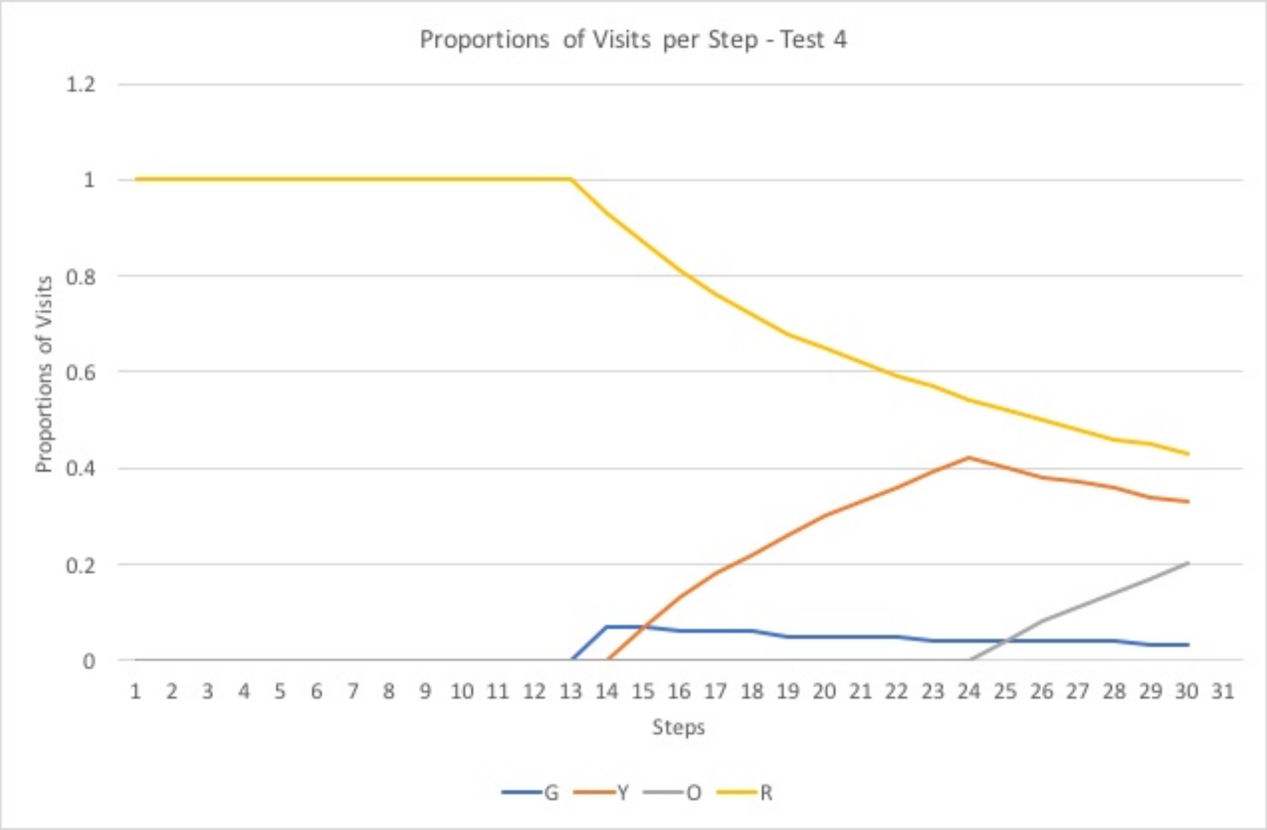
Test 2 (Initial state Y)



Test 3 (Initial state O)

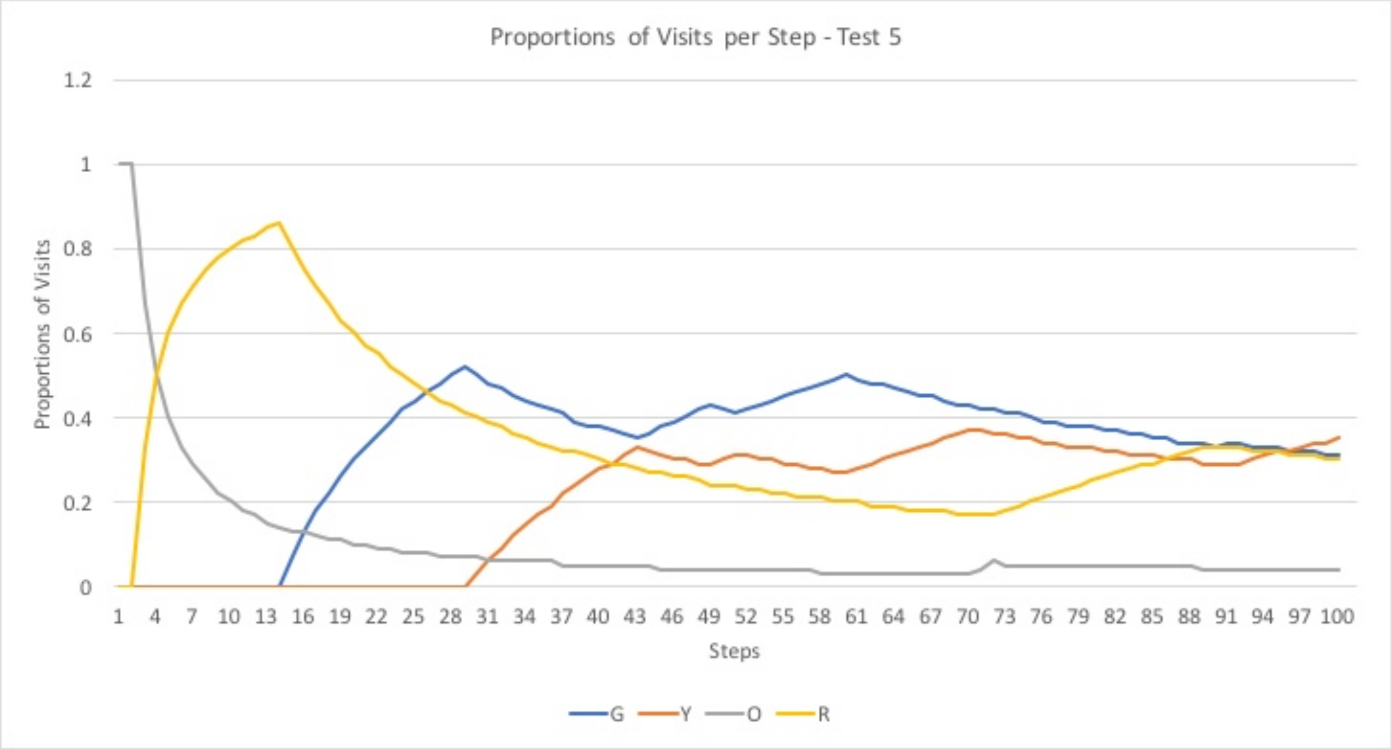


Test 4 (Initial state R)

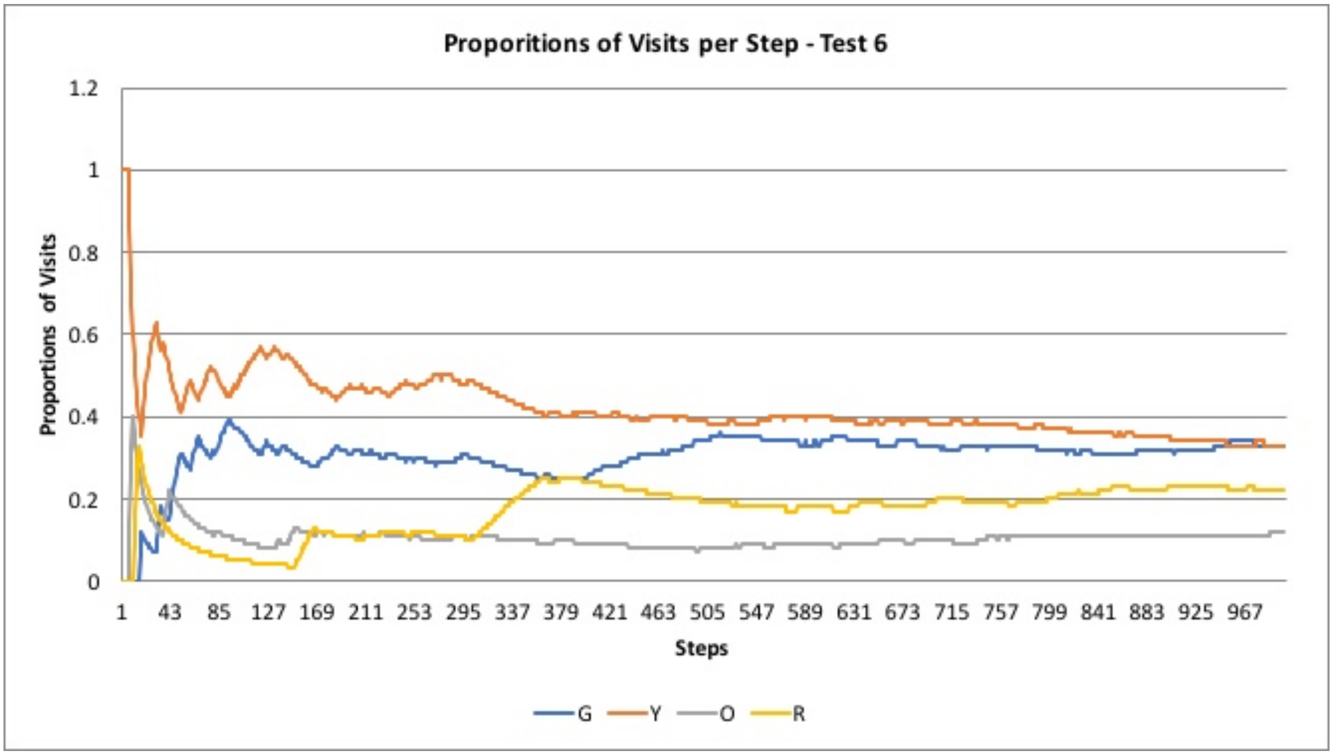


Experiment 2

Test 5 (Random initial state - 100 steps)

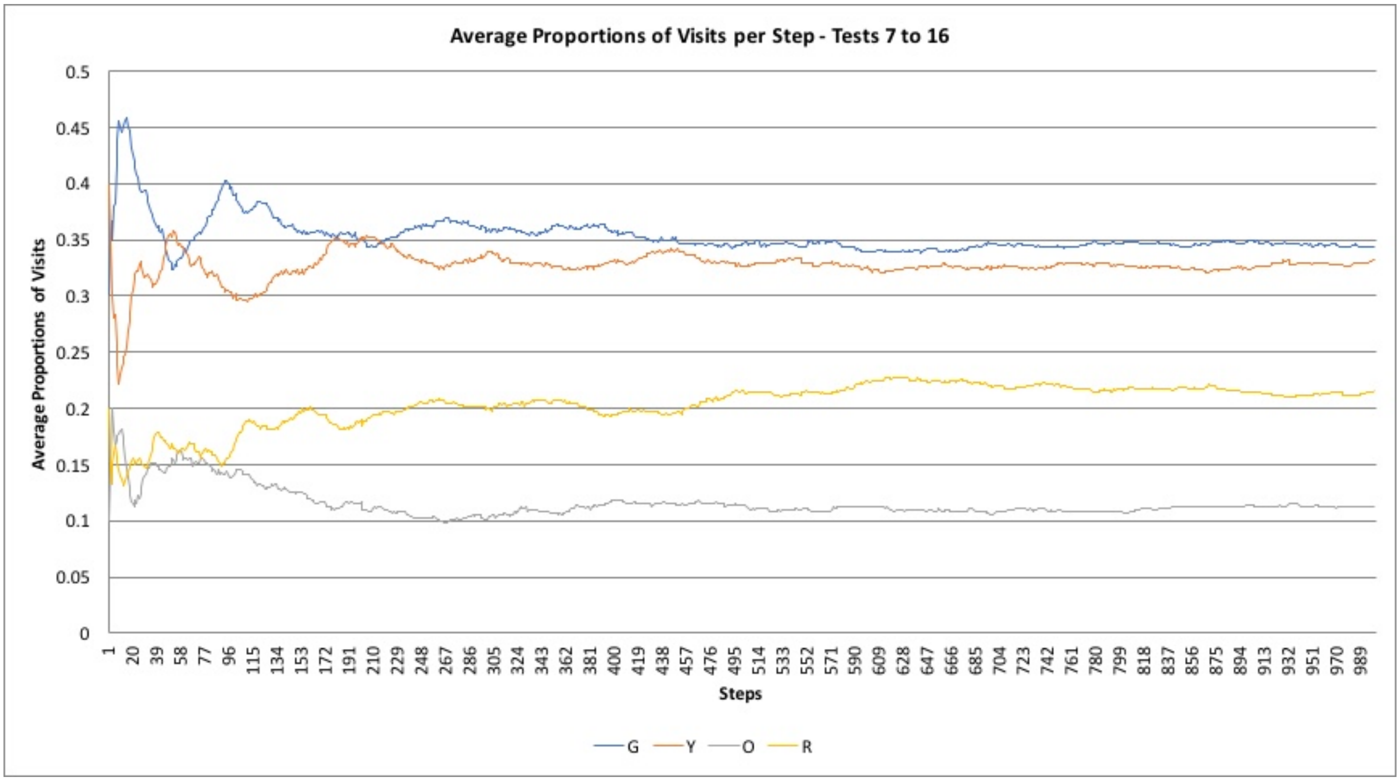


Test 6 (Random initial state - 1000 steps)



Experiment 3

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



Results

Results are avalible 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 proporiton 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 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 there 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.