**Report of TD 1 Markov Model**

Positioning Systems techniques and Applications Master 1,

Internet of Things

The University of Franche-Comté

### By: Uendi MUCA and Omar KHALIL

### Submit To: Dr. Phillipe CANALDA

### Date:25.03.2025

Table of Contents

1. Introduction to Markov Model
2. Problem Introduction
3. Conceptualization of the Problem
4. Proposed Architecture
5. Class Diagram
6. Data Structure
7. Pseudo-Code
8. Algorithm
9. Results from code execution
10. Results from manually solving
11. Comparison between code execution and mathematically solving
12. Showing a more complex scenario
13. Conclusions

## Introduction to Markov Model

Markov models are widely used probabilistic models applied in multiple fields such as physics,engineering,economics and biology to understand and analyze complex systems that undergo random changes over time.The main idea behind a Markov model is to model the probabilities of different states that a system can be in and the rates of transitions among them. In other words, a Markov model captures the dynamics of a system by describing how it moves from one state to another over time.What makes a Markov model different from other models is that it assumes that the future states of the system depend only on its current state, not on the events that occurred before it. This property of Markov model simplifies the modeling process by reducing the number of variables to describe system behavior.Rather than tracking all potential system paths, a Markov model only requires transition probabilities between states, which can be estimated from observed data or theoretical considerations.

Markov models can be represented graphically as a set of nodes connected by edges with associated probabilities.

* **Nodes (states)** representing different conditions of the system.
* **Edges (transitions)** with associated probabilities indicating the likelihood of moving from one state to another.

The probability of moving from one state to another depends on the transition rate, which represents the speed at which the system moves between states.

### Hidden Markov Model (HMM)

Hidden Markov Model(HMM) extends the original Markov model by adding hidden states idea.Hidden Markov Models are probabilistic models that can be used to predict the likelihood of a sequence of hidden states given a sequence of observable outputs.HMMs are used to model the probability of system transiting from one hidden state to another but only is is being able to observe an output that is generated by current state.

Hidden Markov model has two main components:the state transition model and the observation model.The state transition model describes the probability of transitioning from one hidden state to another, while the observation model describes the probability of generating an observable output given the current hidden state

Mathematically HMM is defined as a **5-tuple** (S,V,A,B,π)(S, V, A, B, \pi)(S,V,A,B,π), where:

1. S is the set of hidden states
2. V is the set of observable outputs
3. A is the state transition probability matrix ,where A[i,j] represents the probability of transitioning from hidden state i to hidden state j.
4. B is the observation probability matrix , where B[i,j] represents the probability of observing output j given hidden state i.
5. π is the initial state distribution where π[i] represents the probability of starting in hidden state i.

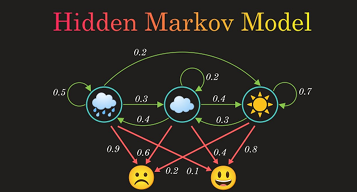
HMMs use two key probability models:

1. **The state transition model**, which determines the likelihood of moving from one hidden state to another.
2. **The observation model**, which defines the probability of producing an observable output given the system’s current hidden state.

HMMs key algorithms are:

1. **Viterbi Algorithm**: Finds the most likely sequence of hidden states that produced the observed outputs.
2. **Forward-Backward Algorithm**: Computes the probability of a given sequence of observations based on the model parameters.

Example of Hidden Markov Model:



In the image we can see there are 3 hidden states corresponding to weather condition :rainy ,cloudy and sunny.Moreover there are also two observable state :happy and sad.Each state transition to another with a certain probability .States has transition probabilities between them shown as green arrows with numerical values .Red arrows represent emission probability linked to observed emotions.Here HMM may model how weather influence the mood.

## 

## 

## Introduction of Problem

Mobility prediction is very important and necessary for different applications like traffic management,location-based services and mobile advertising.It is important to know how to predict future locations or movements of mobile objects in order to help to improve efficiency and decision-making in these areas.

Hidden Markov Models (HMMs) are widely used for mobility prediction because they can effectively analyze sequential data and manage noisy sensor readings. In this context, the problem of mobility prediction with HMMs can be defined as the task of predicting the future location or movement of a mobile object based on its past movements and environmental context. However, predicting mobility patterns in real-world scenarios is challenging due to uncertainties, complex movement behaviors, and the need for accurate, real-time predictions.

In this project, we design an HMM-based architecture for mobility prediction and test its performance using real-world mobility datasets. Our goal is to improve the accuracy and reliability of predicting future locations based on past movements and environmental factors.

## Conceptualization of the Problem

As we explained above Hidden Markov Model and Markov Model are mathematical models or frameworks that are used to analyze and predict sequences of events in a system.This is also the model we will apply in our project to study the statistic and predict future behavior of user on a website with five pages.The goal to track the user navigating through the site ,analyzing it’s movement patterns and use this information to enhance visitors experience.By predicting future actions based on probability of past interactions so we can understand how visitors navigate through the website and use that information to we can optimize and improve user experience.

## Proposed Architecture

Hidden Markov Models are probabilistic models that can be used to predict the likelihood of a sequence of hidden states given a sequence of observable outputs.We will use HMM to estimate how system transition is between states when only partial information is available.

HMMs consist of two main components: the **state transition model** and **the observation model.** The state transition model describes the probability of transitioning from one hidden state to another, while the observation model describes the probability of generating an observable output given the current hidden state.So the nodes here will correspond to 5 pages in website while the

Mathematically, an HMM is represented as **(S, V, A, B, π)**:

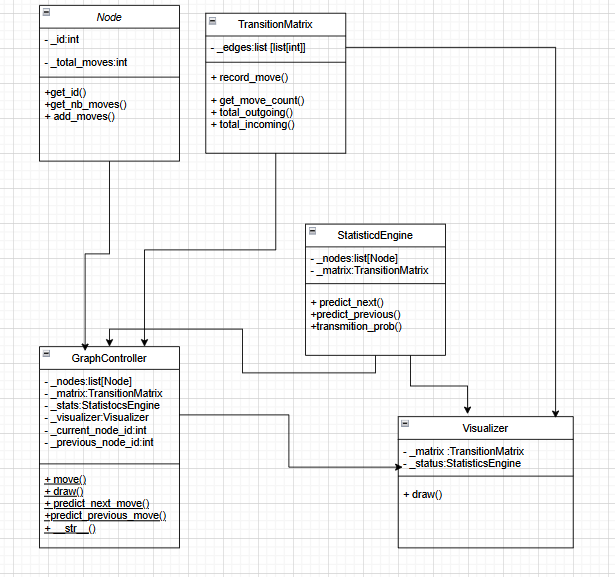
* **S (Hidden States):** The possible internal states of the system.
* **V (Observable Outputs):** The visible events or observations.
* **A (State Transition Matrix):** Probabilities of moving between hidden states.
* **B (Observation Probability Matrix):** Probabilities of different observations occurring in each hidden state.
* **π (Initial State Distribution):** Probability of the system starting in each hidden state.

### **System Requirements**

To implement this model, we use:

1. **Python 3.8**
2. **VS Code or any preferred IDE**
3. **Numpy** (for numerical computations)
4. **Scipy** (for mathematical functions)
5. **Seaborn** (for data visualization)
6. **Tabulate** (format list into table format)

## Class Diagram



The **node class** represent a single page (node) as mentioned above one node is one page ,in total there will be 5 pages so 5 nodes.This class has 2 private attributes:

* **\_id** : Represents the unique identifier of the node (page number).
* **\_total\_nb\_moves** :Tracks how many times a move was made from this node.

The class provides public methods:

* get\_id() : Returns the ID of the node.
* get\_nb\_moves() : Returns the total number of moves made from this node.
* add\_move() : Increments the total number of moves made from this node.

This class is used by the Graph class to store and manage individual nodes (pages) in the system.

### **TransitionMatrix class**

### Encapsulates the transition graph logic via an adjacency matrix.It has one attribute:

* **\_edges:** list[list[int]] :A 2D matrix storing counts of transitions from each node to others.

Methods:

* record\_move(i, j): Increments the transition count from node i to node j.
* get\_move\_count(i, j) :Returns the transition count from node i to j.
* total\_outgoing(i) :Returns the total number of moves originating from node i.
* total\_incoming(j) :Returns the total number of moves directed to node j.

**Used by:**GraphController ,StatisticEngine, Visualizer.

### **StatisticEngine Class**

This class is responsible for calculating transition probabilities and predictions based on the transition matrix.

Attributes:

* **\_nodes**: list[Node] : Reference to the list of nodes.
* **\_matrix**: TransitionMatrix :Reference to the shared transition matrix.

Methods:

* predict\_next(current\_node) : Returns the next most probable node based on outgoing probabilities.
* predict\_previous(current\_node) :Returns the most likely previous node based on incoming probability.
* transition\_prob(i, j) : Computes probability of transitioning from node i to node j.

**Used by:**GraphController,Visualizer

### **Visualizer Class**

Handles rendering the graph with color-coded edges and nodes.

**Attributes:**

* **\_matrix**: TransitionMatrix :To access transition counts.
* **\_stats:** StatisticEngine :To annotate edges with probabilities or percentages.

**Methods:**

* draw() : Visualizes the current state of the transition graph using NetworkX.

**Used by:**GraphController

### **GraphController**

Central controller that brings everything together — the nodes, transition matrix, statistics engine, and visualizer. Acts as the main entry point for user interactions. It processes movements, updates the transition matrix, queries predictions, and triggers visualizations.

**Attributes:**

* **\_nodes: list[Node]** : All pages in the graph.
* **\_matrix: TransitionMatrix** :Matrix managing transitions.
* **\_stats: StatisticEngine** :for predictions.
* **\_visualizer**: Visualizer :Used for graph visualization.
* **\_current\_node\_id**: int :Currently active node.
* **\_previous\_node\_id**: int :Last visited node.

**Methods:**

* move(next\_node\_id: int) : Updates move counts and changes the current node.
* draw() : Calls the visualizer to render the graph.
* predict\_next\_move() :Uses StatisticEngine to guess the next move.
* predict\_previous\_move() :Uses StatisticEngine to guess where the user came from.
* \_\_str\_\_() :Returns a tabulated overview of transition counts and percentages.

The GraphController manages navigation by integrating four components: Node, TransitionMatrix, StatisticEngine, and Visualizer. It uses Node to track page visits and TransitionMatrix to record transitions between nodes. The StatisticEngine analyzes transitions to predict likely next or previous moves. The Visualizer draws the graph using data from the TransitionMatrix and insights from the StatisticEngine. Both StatisticEngine and Visualizer depend on the TransitionMatrix, which acts as the central data source.

## Data Structure

**1.self.\_node:List[node]**

* Defined in the Graph class constructor
* Stores all Node objects representing different pages
* Each element is an instance of the Node class
* Used in **move()** and **node\_stats()**

**2.self.\_edges->List[List[int]]**

* Defines in the Graph class constructor
* Represent the adjacency matrix storing the transition counts
* Each row is current page and each column destination page
* Used in **move()** and **node\_stats()**

**3.stats:List[float]**

* Created inside predict\_next\_move()
* Store transition probabilities for predictions

## 

## Pseudo-Code

We will separate the pseudo-code one for each class to be more readable format.

**1.Node Class Pseudocode**

CLASS Node:

INIT(id): set \_id = id, \_total\_nb\_moves = 0

METHOD get\_id(): return \_id

METHOD get\_nb\_moves(): return \_total\_nb\_moves

METHOD add\_move(): \_total\_nb\_moves += 1

**2.Graph Class PseudoCode**

CLASS Graph:

INIT(nb\_nodes=5): create \_nodes[0..nb\_nodes-1], \_edges[nb\_nodes][nb\_nodes] = 0

set \_current\_node\_id = 0, \_previous\_node\_id = NULL

METHOD move(next\_id):

IF invalid id: return False

\_nodes[current].add\_move()

\_edges[current][next\_id] += 1

update current and previous

return True

METHOD node\_stats(origin, dest):

total = \_nodes[origin].get\_nb\_moves()

IF total == 0: return (0, 0.0)

moves = \_edges[origin][dest]

return (moves, moves \* 100 / total)

METHOD predict\_next\_move():

get % of moves from current node

return index of max % or NULL

METHOD predict\_previous\_move():

get % of moves into current node

return index of max % or NULL

METHOD draw\_graph(): render graph with colors, labels

METHOD \_\_str\_\_(): return table of moves and % transitions

**3.Main (Testing)**

MAIN:

Prompt for mode

Create Graph(5)

IF mode == 1: interactive\_mode(graph)

ELSE: random\_mode(graph, 10)

**4.Interactive Mode(Manual Navigation)**

FUNCTION interactive\_mode(graph):

LOOP:

INPUT move

IF move == -1: BREAK

IF move == 5: graph.draw\_graph()

IF 0 ≤ move ≤ 4:

graph.move(move)

PRINT graph, predict\_next, predict\_prev

ELSE: print "Invalid input"

**5.Random Mode(Automated Navigation)**

FUNCTION random\_mode(graph, moves):

FOR i in 1..moves:

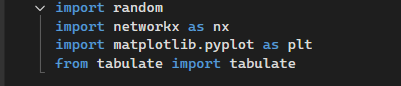
next\_move = RANDOM(0..4)

graph.move(next\_move)

PRINT graph, predict\_next, predict\_prev

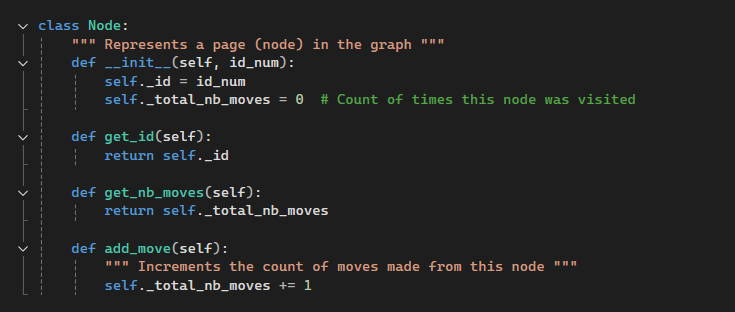
## Algorithm

1.Firstly we import libraries needed:



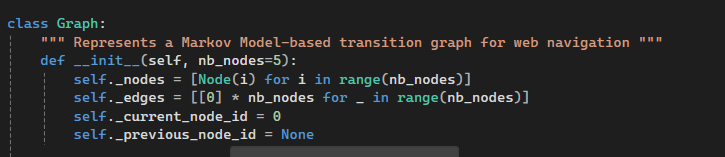
* **random** : Used for random page navigation.
* **tabulate** :Formats the transition matrix into a readable table.
* **Matplotlib**:Plotting heatmaps,charts for visualization.
* **networkx**:Graph visualization library that allows us to draw the Markov Chain as a directed graph.

2.Node class which:



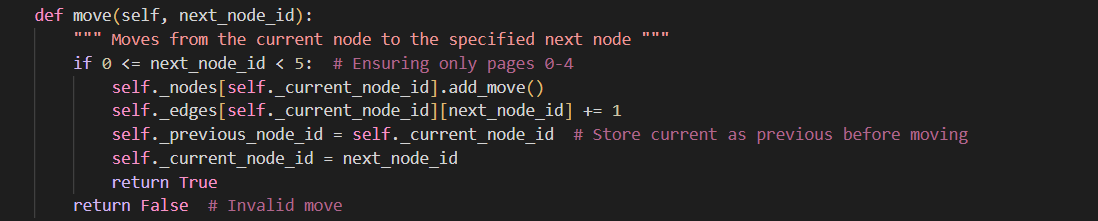
* Store information about a webpage where one webpage is one node
* Track how many times the page has been visited
* Provides methods to get page ID,get total visit count and increase visit count when visited.

3.1 init function



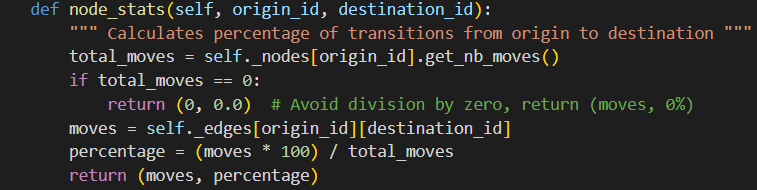
* Manage multiple node objects
* Tracks transition between pages in a 5x5 adjacent matrix
* Keep track of current and previous page

3.2 move



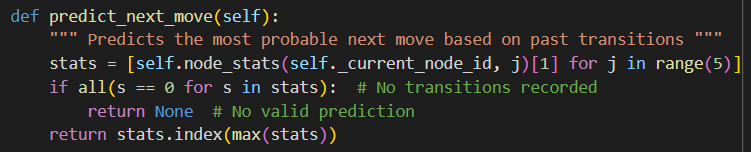
* Check if the move is valid (move is valid between 0-4 as we have only 5 pages/nodes)
* Increments the visit count for the current page
* Store current node as previous before moving
* Updates the transition matrix
* Update the current page

3.3 Node Stats



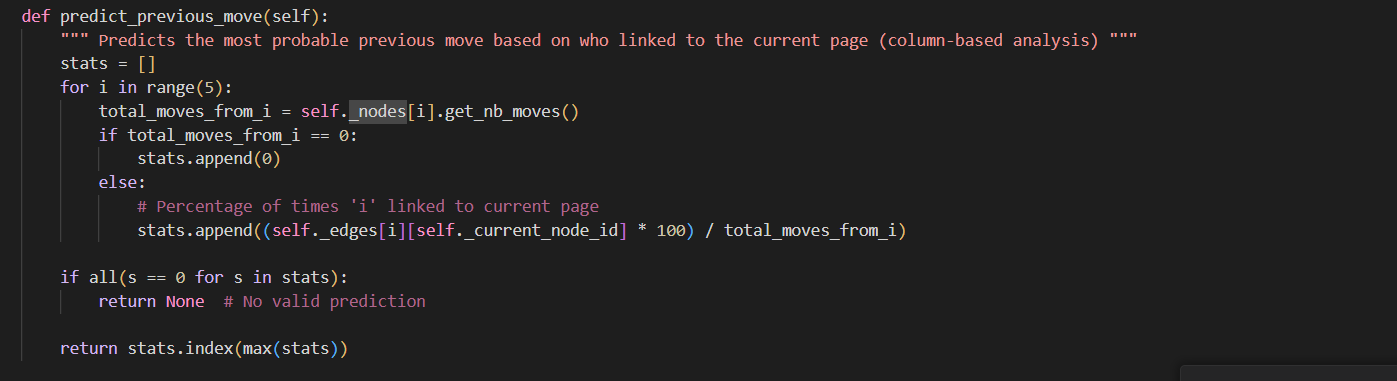
* Get total moves from the origin page
* Calculates the percentage of moves to the destination
* Return move count and percentage

3.4 Predict next move



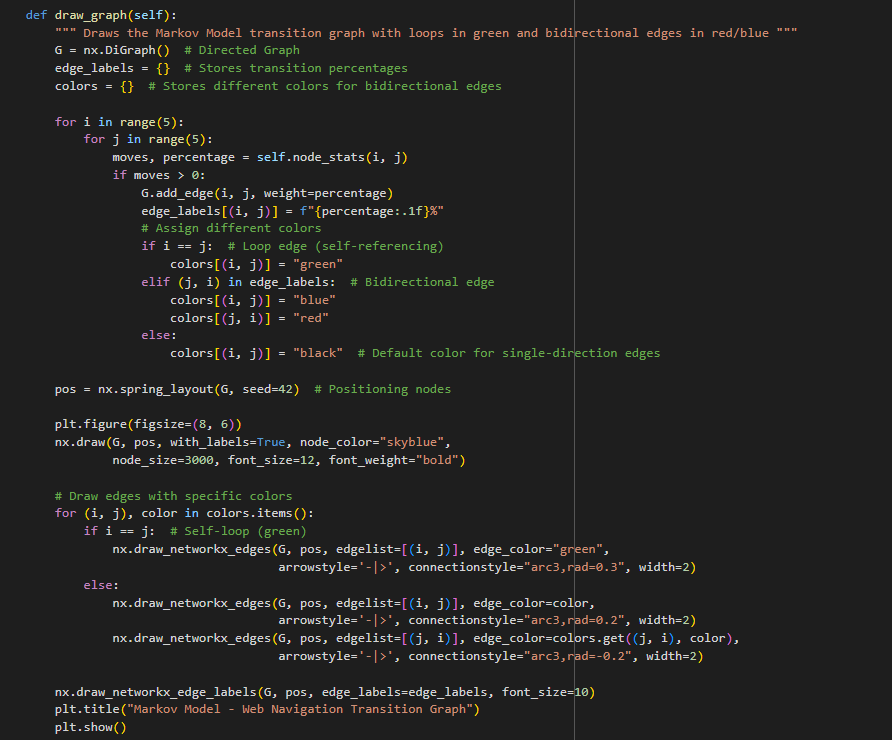
* Finds the page with the highest transition probability using data in rows(left to right)..
* Returns the predicted next page.

3.5 Predict previous move



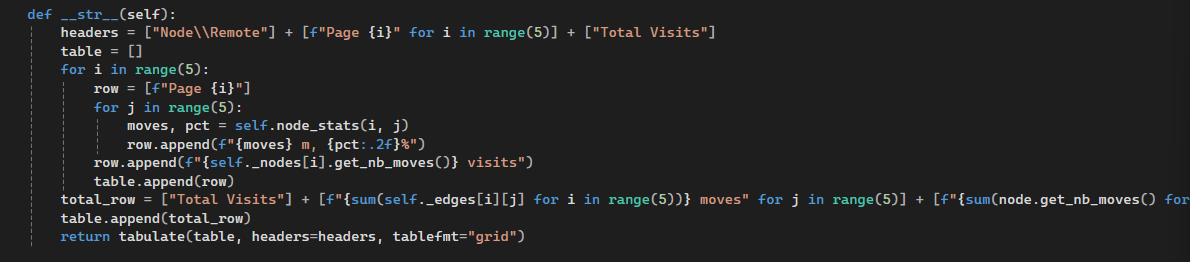
* Predicts which node (page) the user most likely came from.
* Uses column-based analysis of the transition matrix.
* Gets the total number of moves made from node i.
* Calculates what percentage of those moves ended up at the current node.
* Builds a list of percentages (stats) representing how likely each node is to be the previous one.
* If all values in stats are zero, returns None (no valid prediction).
* Otherwise, returns the node ID (i) with the highest percentage , this is the predicted previous node.

3.6 Draws the nodes diagram



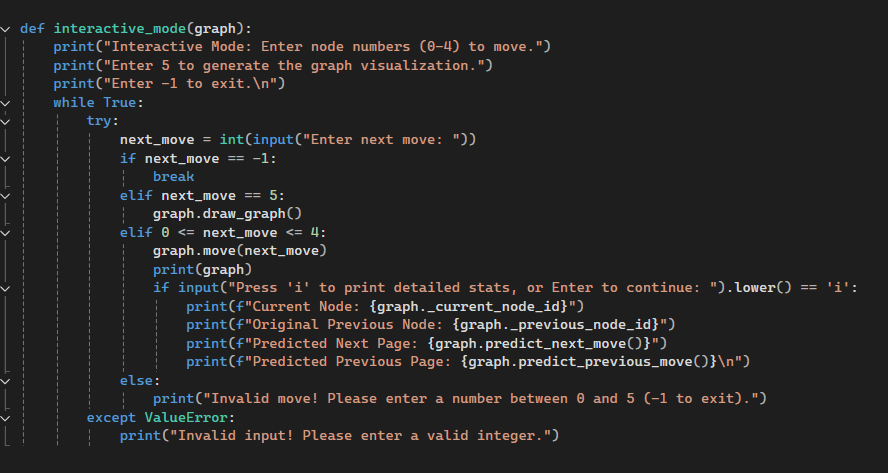
* Create a directed graph for navigation
* Add edges based on transition data by iterating over all nodes and check if transition exists then add a direct edge with its probability and store the percentage too ,as it will be displayed in the graph.
* Assign colors to edges: green for self loops,blue and red for bidirectional edges and black for one way transitions.
* Use spring layout to position nodes automatically
* Draw nodes, edges and add labels for probabilities

3.7 Display table



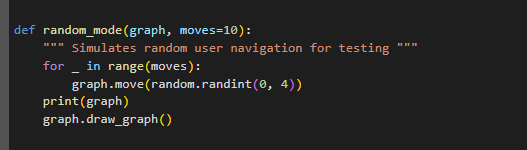
* Prints a readable table of movements.
* Displays move counts and percentages.

4.1 Interactive Mode user controlled navigation



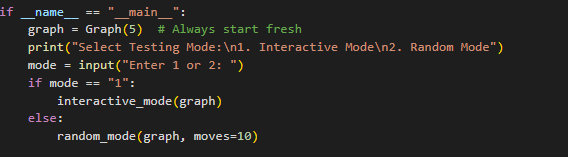
* User can enter a page number from 0-4
* Then code validates input:
  + If the input is -1, it exits.
  + If the input is valid (0-4), it updates the transition matrix.
  + If the input is invalid, it prints an error message.
* Updates the transition matrix .
* Displays the new matrix after each move.

4.2 Random mode (Automated Testing)



* Automatically selects a random page (0-4).
* Repeat this for moves=10 times (optionally can be changed).
* Updates the transition matrix
* After moving iterations, print the final transition table.

5 Main



* The program starts correctly.
* The user can choose between interactive\_mode() and random\_mode().
* The session is saved after execution.

## 

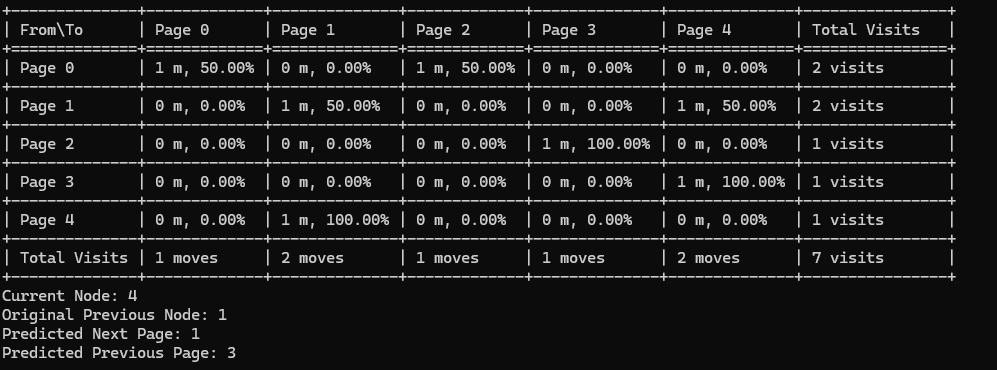
## 

## 

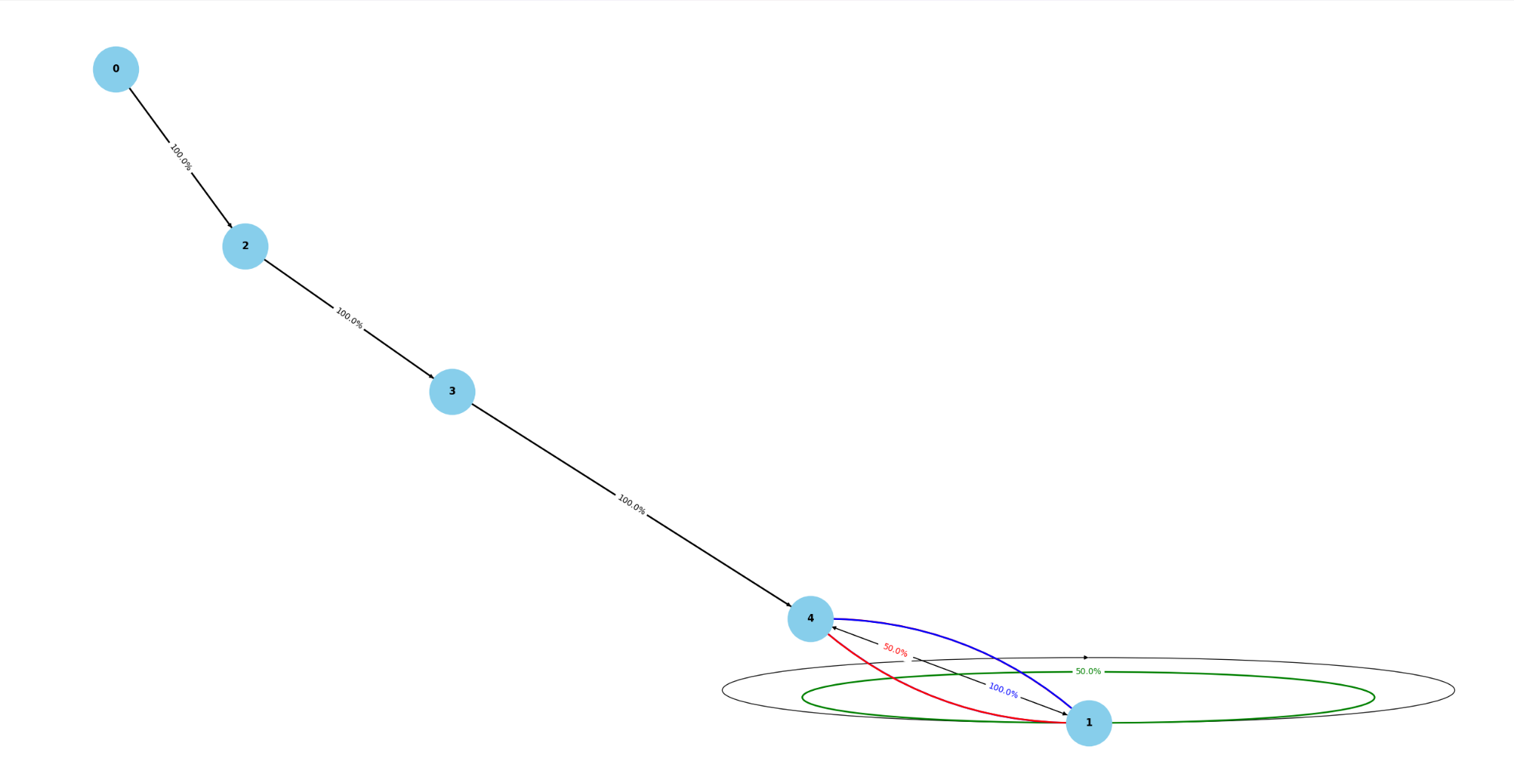
## 

## 

## Results from the code execution



* This table is using the same movement set we will use in the ‘Theoretical proof’ which is {0,2,3,4,1,1,4}., it shows the number of visits for each node and the predicted percentages of the next/previous move.

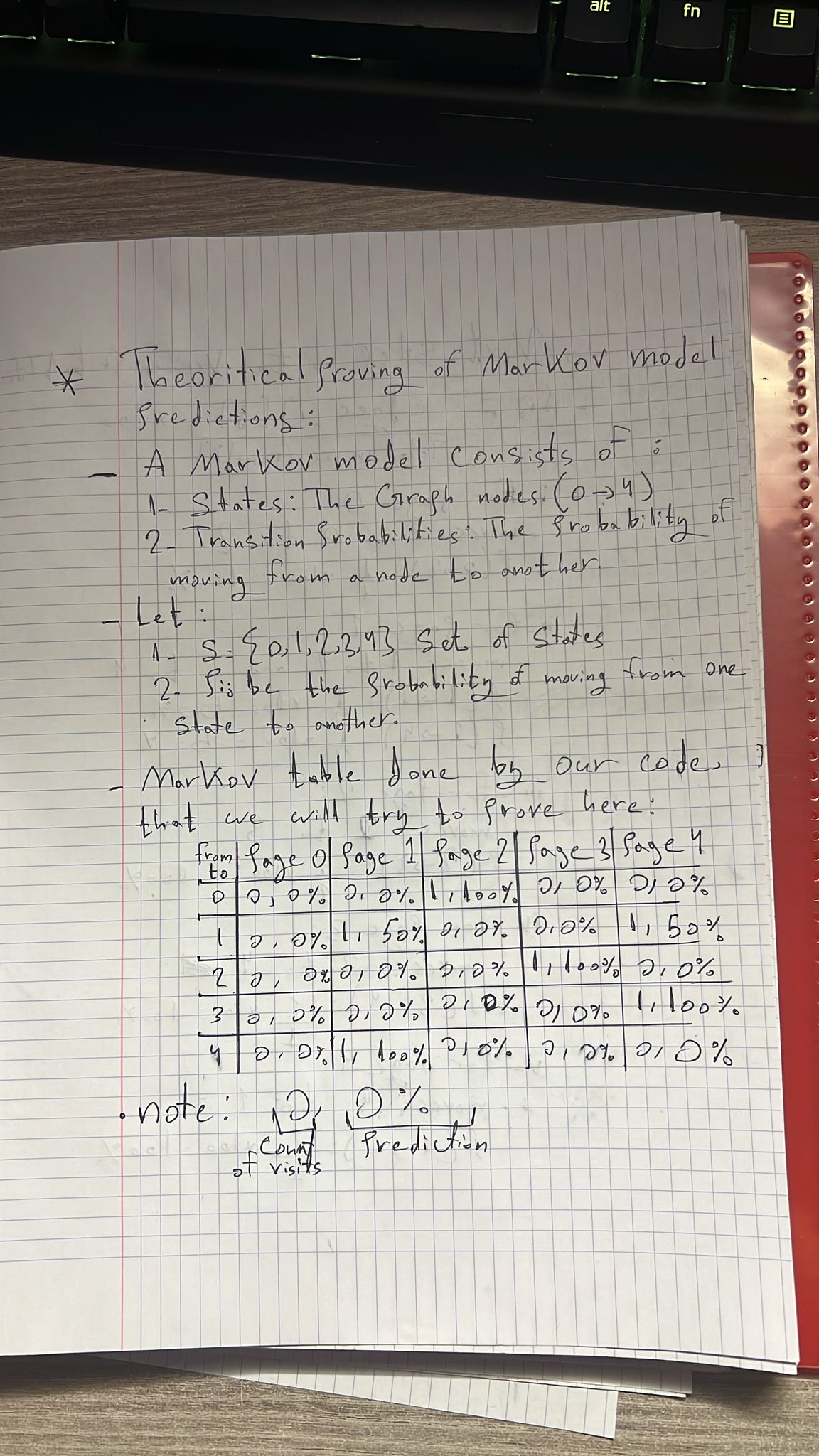


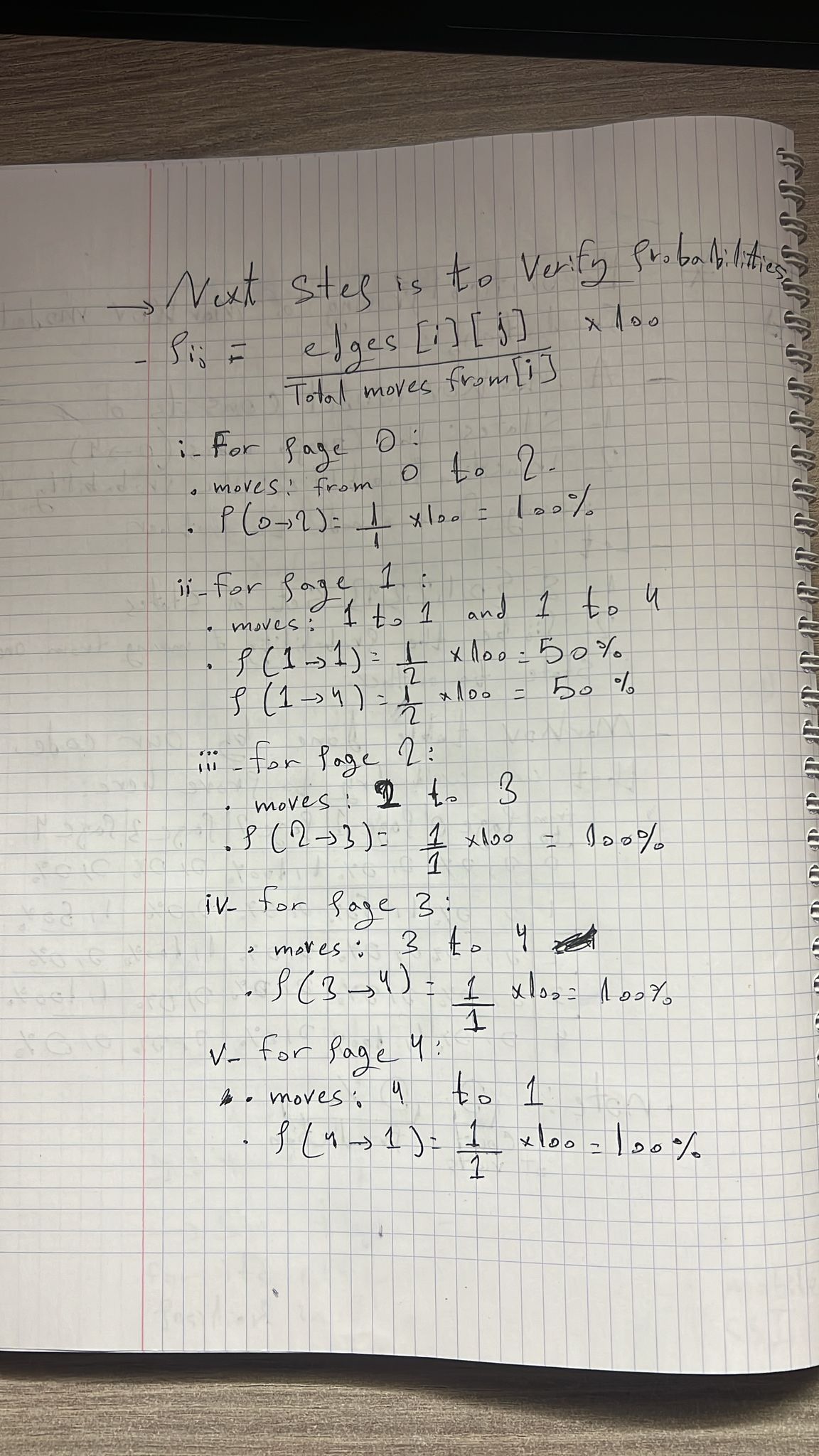
Here is a generated Markov Chain Transition Graph, which is generated through our program.  
It shows us the predicted movement from a node to another.

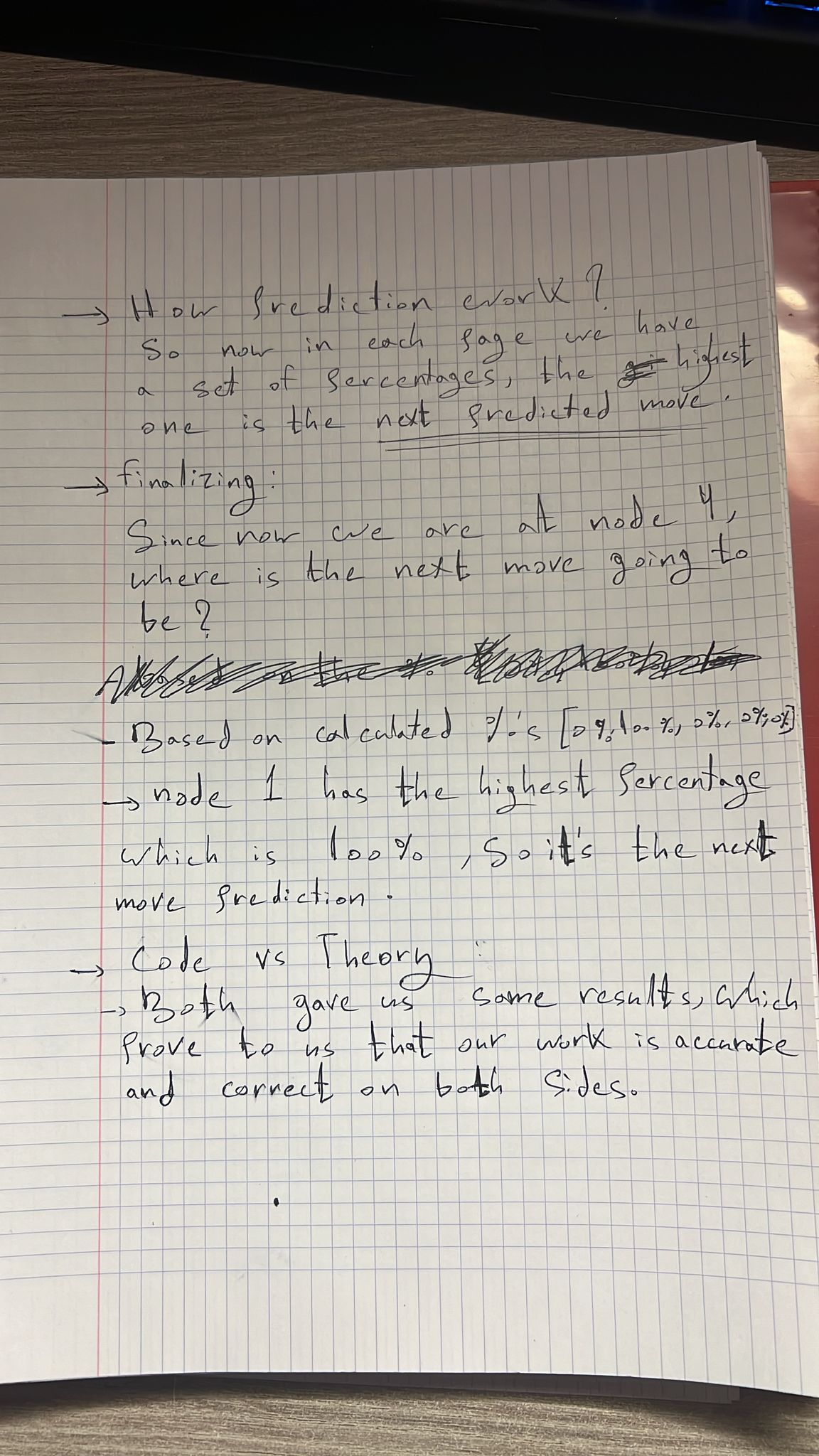
Legend:

* Blue and red line indicated bidirectional connection between nodes
* Green line indicates a loop
* Value shown is the predicted movement percentage

## Comparison between theoretical and code In this part we will prove our code results using theories and applying the rules that we used in our code:





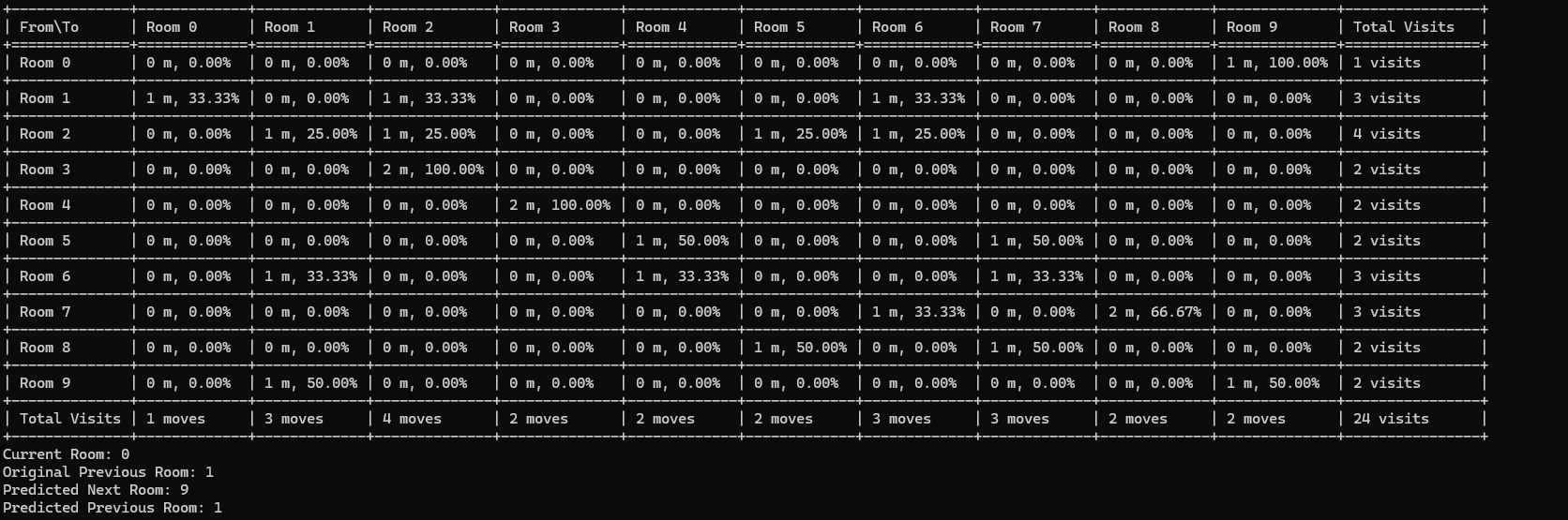


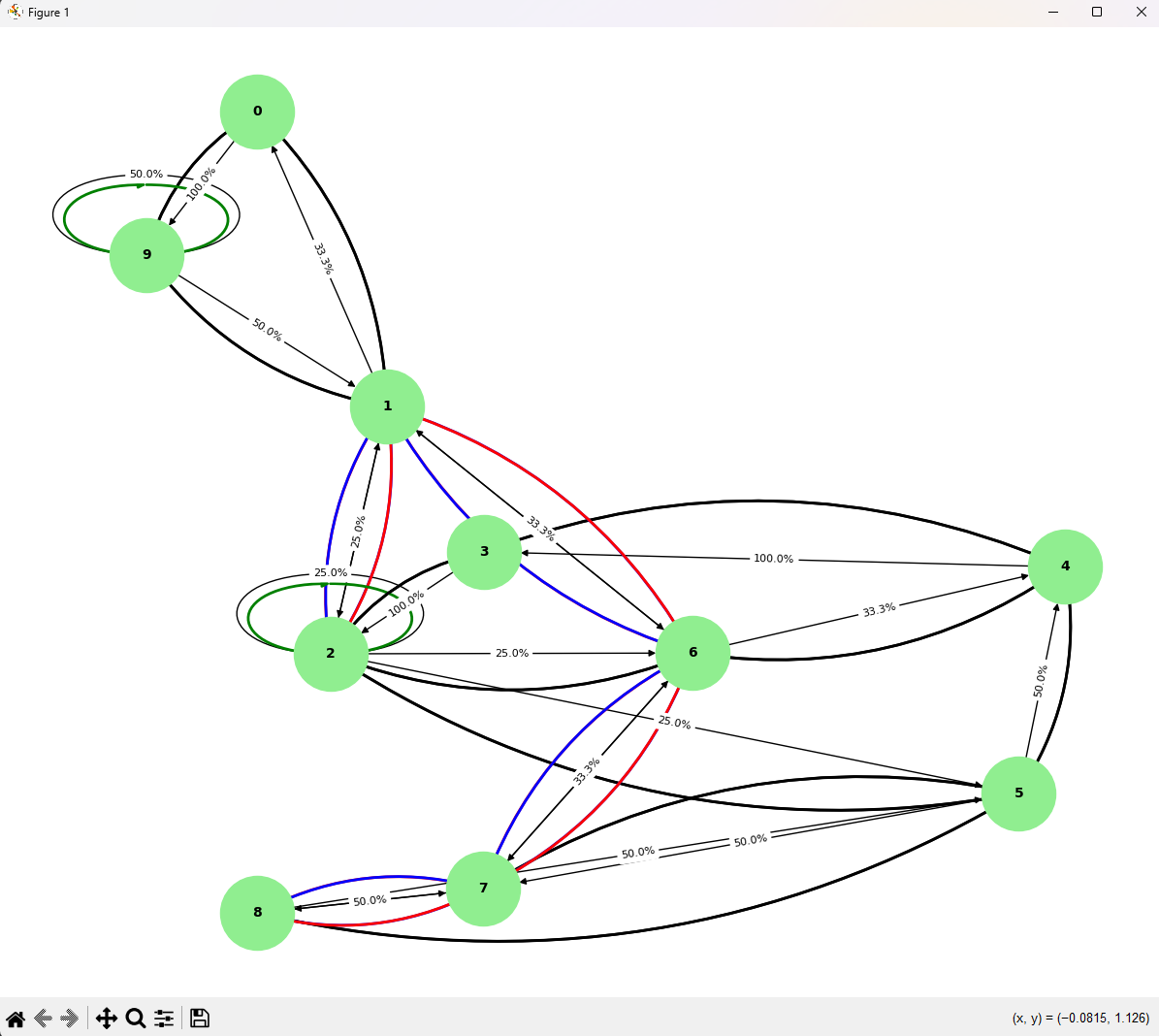
* To compare the results obtained from the **markov model prediction code** with the theoretical manual calculation, we chose a set of moves of {0,2,3,4,1,1,4}.  
    
  In the code it gave a prediction of current node 4 is expected to **move to node 1 with 100% probability**.
* While for the theoretical proof, we used the same probability rule we used in our code and it gave us the same results in theory which is that current node 4 is expected to **move to node 1 with 100% probability**.
* Keep in mind that we didn't prove the previous prediction in hand, because it's **already proved in code**, by giving the **actual previous** and **predicted previous** with the same values.
* Note that: The probability rule we used is percentage = (moves \* 100) / total\_moves.
* We provided only one general example to illustrate that code works in an accurate way. The same idea and way would be if we choose other results that we get from our code too.

### **A Complex scenario:**

Here we will try to run the same code, but using a more complex scenario to make sure ourcode is efficient and dynamic.

The scenario is that we have a building of 10 rooms. Track how many times the rooms are getting visited.





### **Conclusion:**

* In conclusion, mobility prediction with Hidden Markov Models is a useful and versatile technique for modeling and predicting the movement patterns of individuals or groups of people over time. HMMs allow for the incorporation of multiple sources of data, including geographic information, social network data, and sensor data, which can lead to more accurate and personalized models.
* However, there are some limitations to using HMMs, including the assumption of a stationary underlying process and the need for significant amounts of training data to accurately estimate model parameters. Despite these limitations, HMMs remain a valuable tool for mobility prediction, with potential applications in various domains such as transportation, healthcare, and sports.
* The results obtained from the Markov Model prediction code and the theoretical manual calculations are identical, confirming that the implemented probability rule is correctly applied in the code. By using the transition probability formula:
* percentage = (moves \* 100) / total\_moves.
* We were able to manually verify that when the current node is 4, the predicted next node is 1 with 100% probability, matching the output of our program.
* This validation confirms that the Markov Model implementation is accurate and reliable, and that the transition calculations correctly reflect the observed movement patterns.

**Therefore, we can confidently rely on the model for predicting future transitions based on past behavior.**