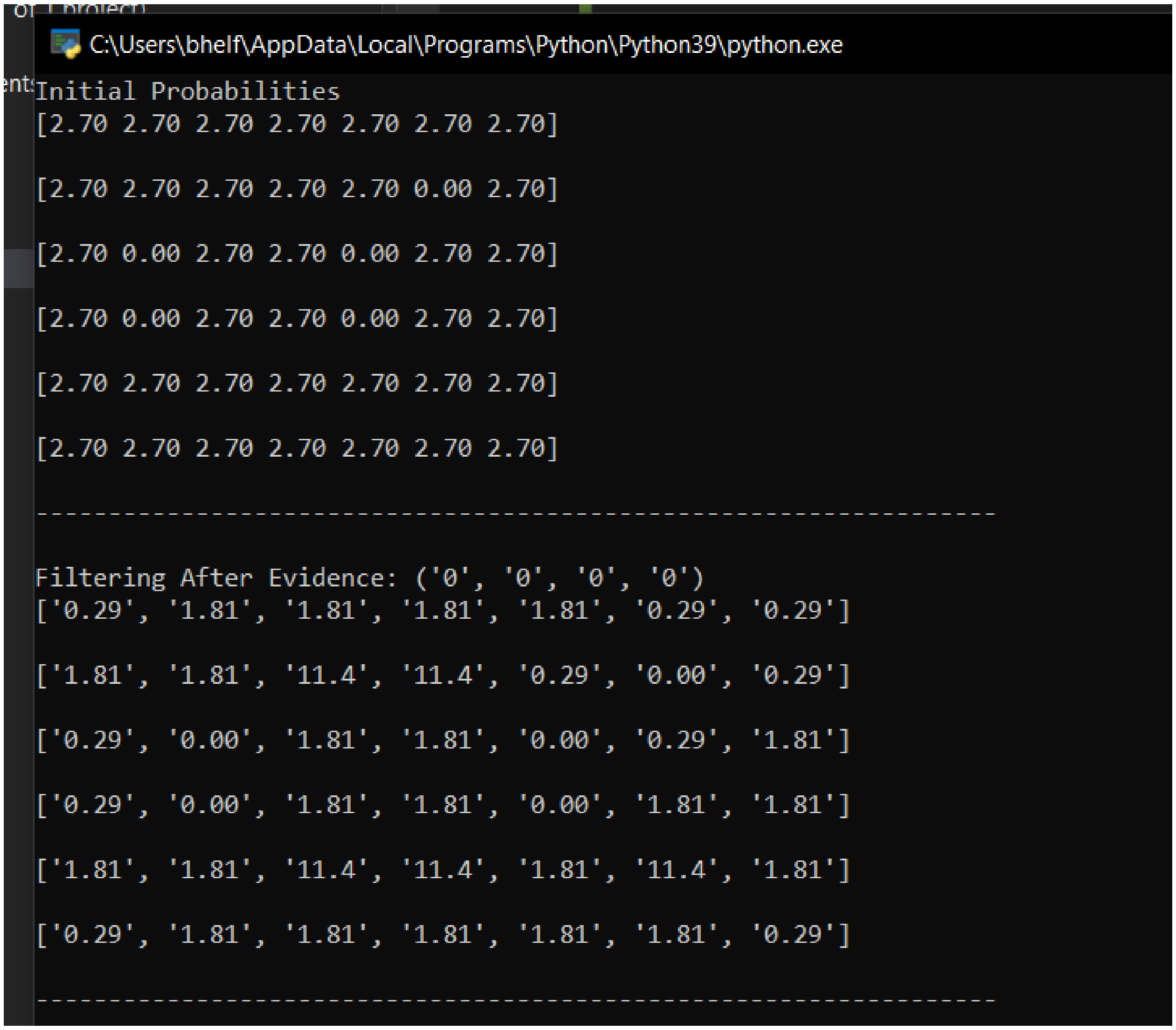
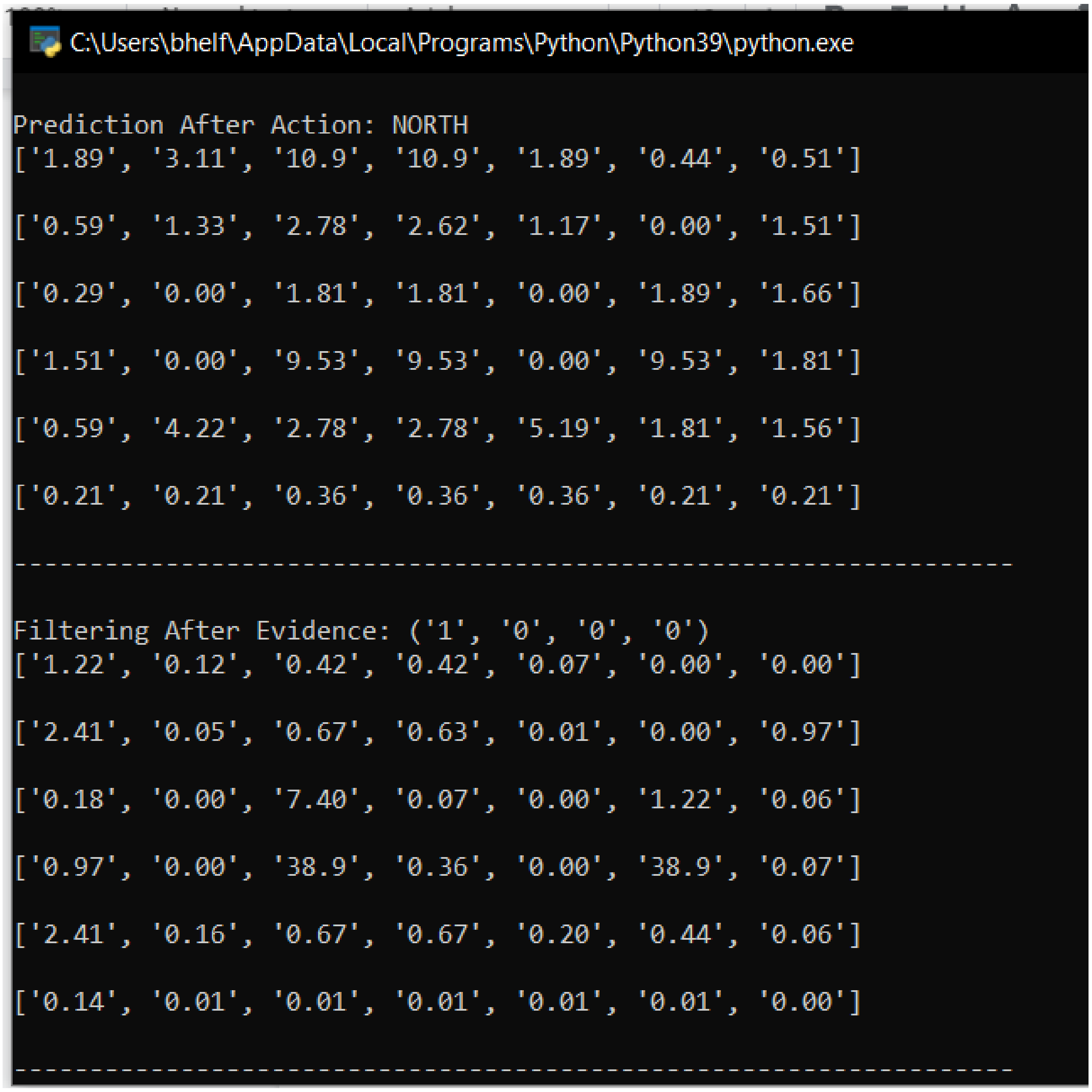
Project 2(HMM)

Reports and Testing:

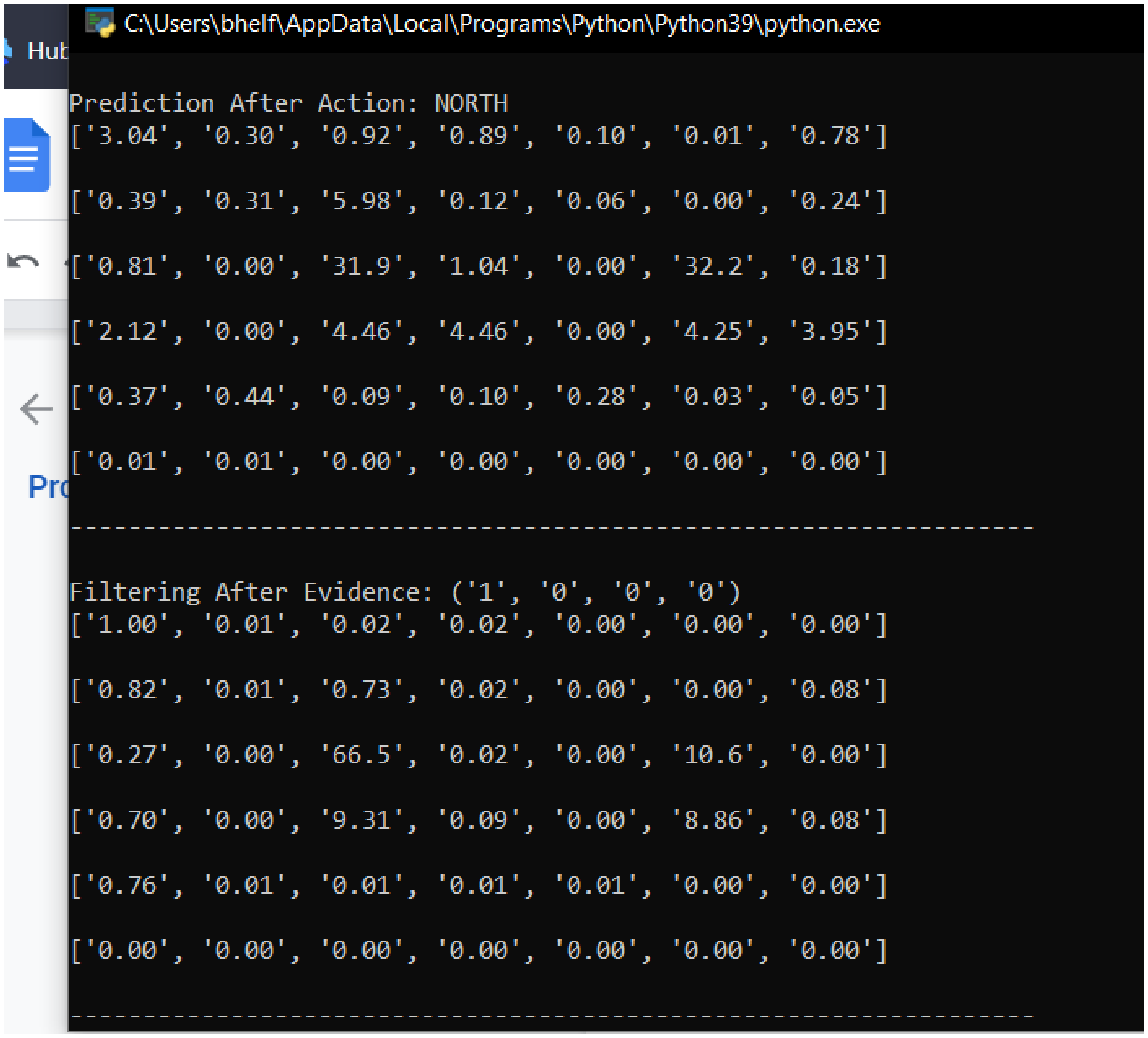


```
C:\Users\bhelf\AppData\Local\Programs\Python\Python39\python.exe
Initial Probabilities
[2.70 2.70 2.70 2.70 2.70 2.70 2.70]

[2.70 2.70 2.70 2.70 2.70 0.00 2.70]

[2.70 0.00 2.70 2.70 0.00 2.70 2.70]

[2.70 0.00 2.70 2.70 0.00 2.70 2.70]

[2.70 2.70 2.70 2.70 2.70 2.70 2.70]

[2.70 2.70 2.70 2.70 2.70 2.70 2.70]

------------------------------------------------------------------

Filtering After Evidence: ('0', '0', '0', '0')
['0.29', '1.81', '1.81', '1.81', '1.81', '0.29', '0.29']

['1.81', '1.81', '11.4', '11.4', '0.29', '0.00', '0.29']

['0.29', '0.00', '1.81', '1.81', '0.00', '0.29', '1.81']

['0.29', '0.00', '1.81', '1.81', '0.00', '1.81', '1.81']

['1.81', '1.81', '11.4', '11.4', '1.81', '11.4', '1.81']

['0.29', '1.81', '1.81', '1.81', '1.81', '1.81', '0.29']

------------------------------------------------------------------
```



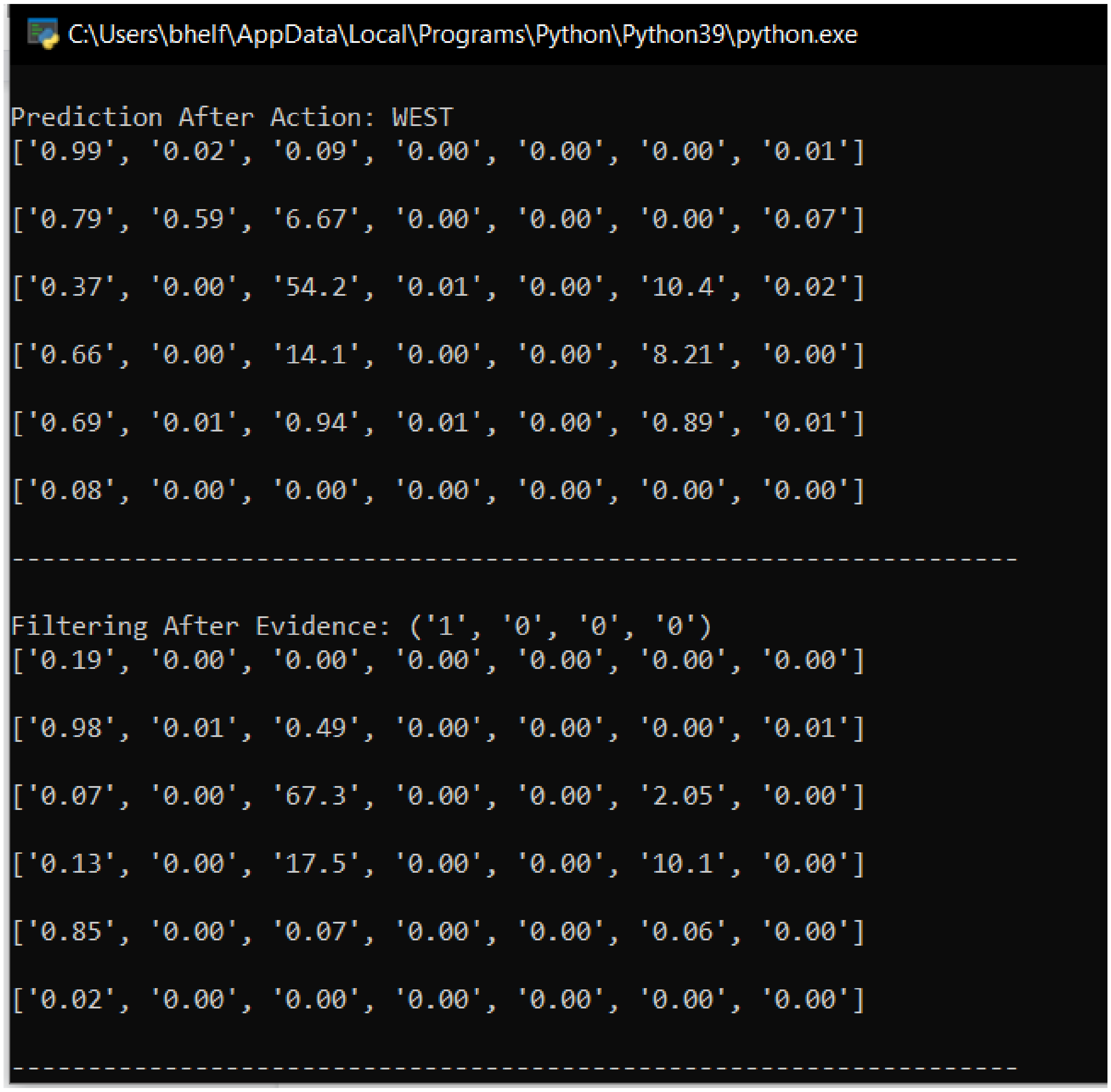
C:\Users\bhelf\AppData\Local\Programs\Python\Python39\python.exe

```
Prediction After Action: NORTH
['1.89', '3.11', '10.9', '10.9', '1.89', '0.44', '0.51']

['0.59', '1.33', '2.78', '2.62', '1.17', '0.00', '1.51']

['0.29', '0.00', '1.81', '1.81', '0.00', '1.89', '1.66']

['1.51', '0.00', '9.53', '9.53', '0.00', '9.53', '1.81']

['0.59', '4.22', '2.78', '2.78', '5.19', '1.81', '1.56']

['0.21', '0.21', '0.36', '0.36', '0.36', '0.21', '0.21']

------------------------------------------------------------------

Filtering After Evidence: ('1', '0', '0', '0')
['1.22', '0.12', '0.42', '0.42', '0.07', '0.00', '0.00']

['2.41', '0.05', '0.67', '0.63', '0.01', '0.00', '0.97']

['0.18', '0.00', '7.40', '0.07', '0.00', '1.22', '0.06']

['0.97', '0.00', '38.9', '0.36', '0.00', '38.9', '0.07']

['2.41', '0.16', '0.67', '0.67', '0.20', '0.44', '0.06']

['0.14', '0.01', '0.01', '0.01', '0.01', '0.01', '0.00']

------------------------------------------------------------------
```



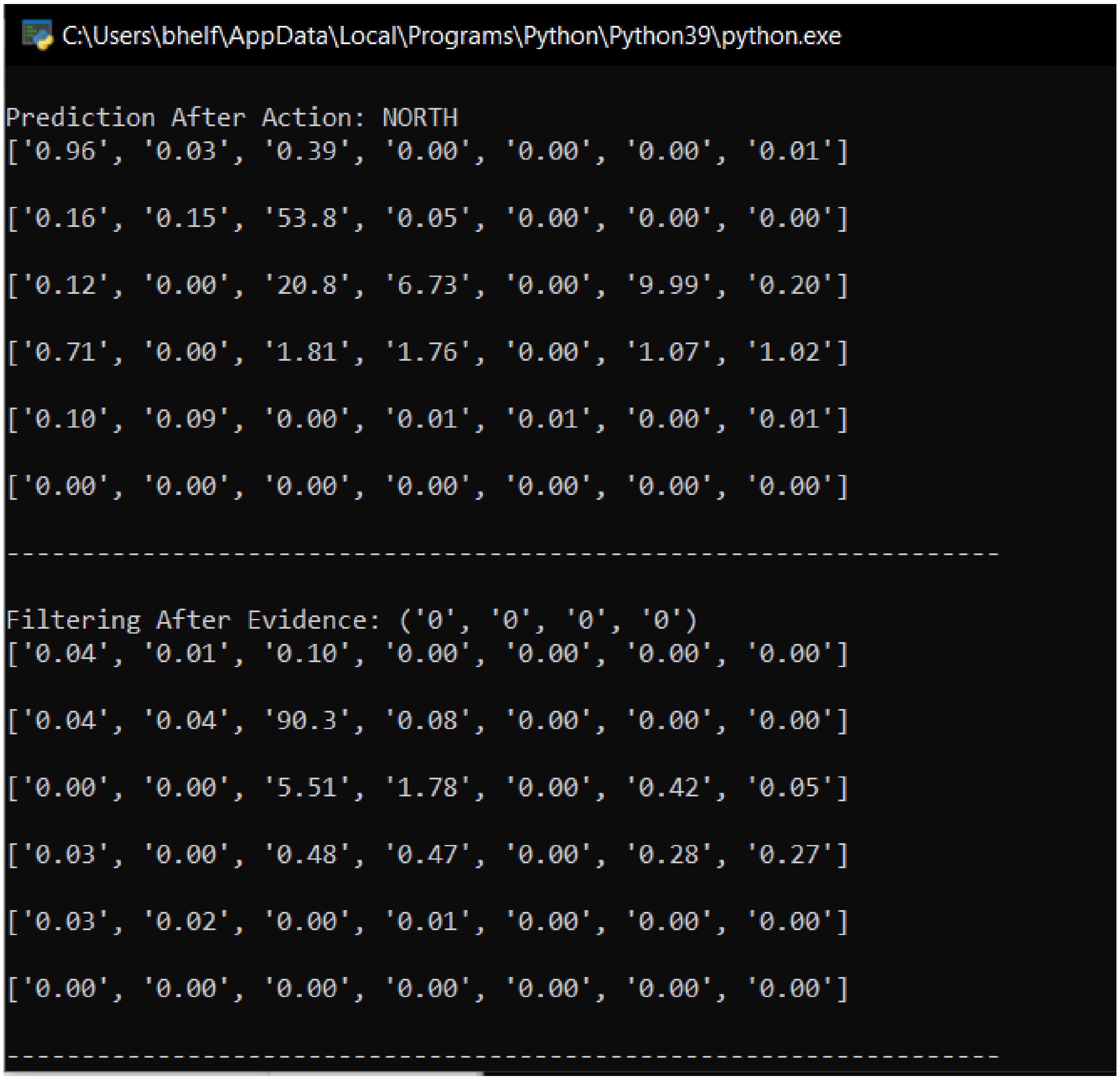
C:\Users\bhelf\AppData\Local\Programs\Python\Python39\python.exe

```
Prediction After Action: NORTH
['3.04', '0.30', '0.92', '0.89', '0.10', '0.01', '0.78']

['0.39', '0.31', '5.98', '0.12', '0.06', '0.00', '0.24']

['0.81', '0.00', '31.9', '1.04', '0.00', '32.2', '0.18']

['2.12', '0.00', '4.46', '4.46', '0.00', '4.25', '3.95']

['0.37', '0.44', '0.09', '0.10', '0.28', '0.03', '0.05']

['0.01', '0.01', '0.00', '0.00', '0.00', '0.00', '0.00']

------------------------------------------------------------------

Filtering After Evidence: ('1', '0', '0', '0')
['1.00', '0.01', '0.02', '0.02', '0.00', '0.00', '0.00']

['0.82', '0.01', '0.73', '0.02', '0.00', '0.00', '0.08']

['0.27', '0.00', '66.5', '0.02', '0.00', '10.6', '0.00']

['0.70', '0.00', '9.31', '0.09', '0.00', '8.86', '0.08']

['0.76', '0.01', '0.01', '0.01', '0.01', '0.00', '0.00']

['0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']

------------------------------------------------------------------
```



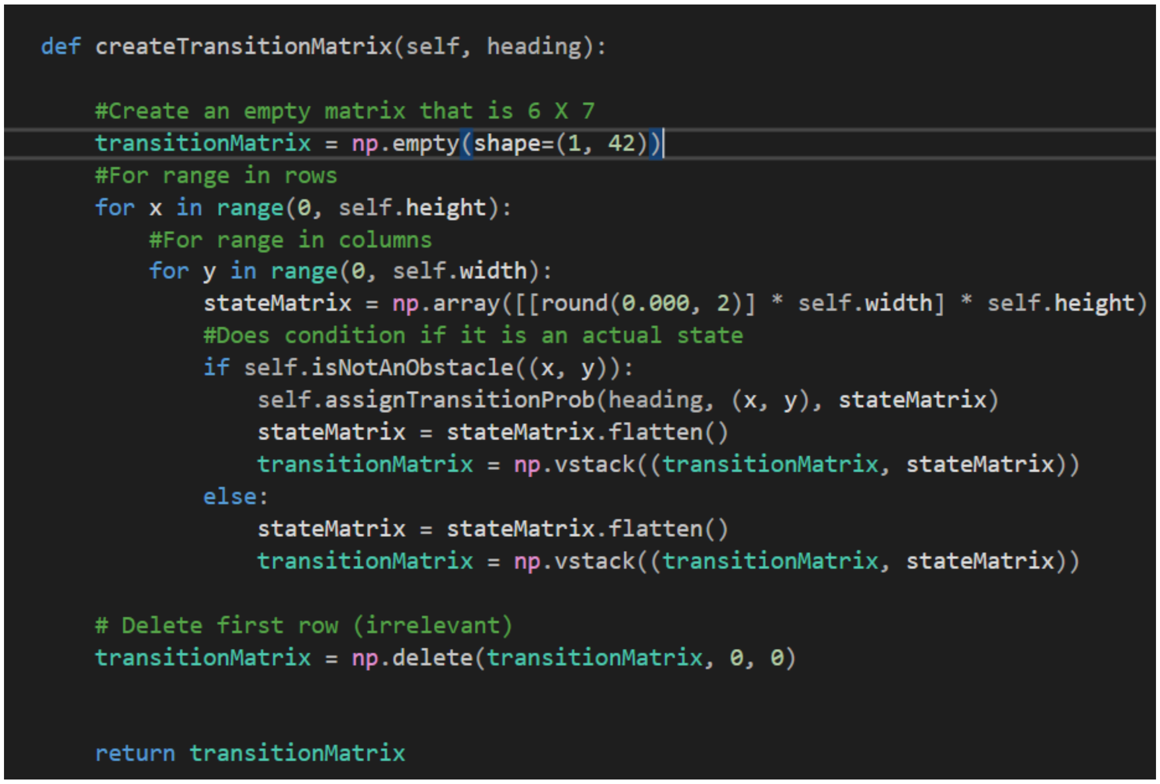
C:\Users\bhelf\AppData\Local\Programs\Python\Python39\python.exe

```
Prediction After Action: WEST
['0.99', '0.02', '0.09', '0.00', '0.00', '0.00', '0.01']

['0.79', '0.59', '6.67', '0.00', '0.00', '0.00', '0.07']

['0.37', '0.00', '54.2', '0.01', '0.00', '10.4', '0.02']

['0.66', '0.00', '14.1', '0.00', '0.00', '8.21', '0.00']

['0.69', '0.01', '0.94', '0.01', '0.00', '0.89', '0.01']

['0.08', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']

-----------------------------------------------------------------

Filtering After Evidence: ('1', '0', '0', '0')
['0.19', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']

['0.98', '0.01', '0.49', '0.00', '0.00', '0.00', '0.01']

['0.07', '0.00', '67.3', '0.00', '0.00', '2.05', '0.00']

['0.13', '0.00', '17.5', '0.00', '0.00', '10.1', '0.00']

['0.85', '0.00', '0.07', '0.00', '0.00', '0.06', '0.00']

['0.02', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']

-----------------------------------------------------------------
```

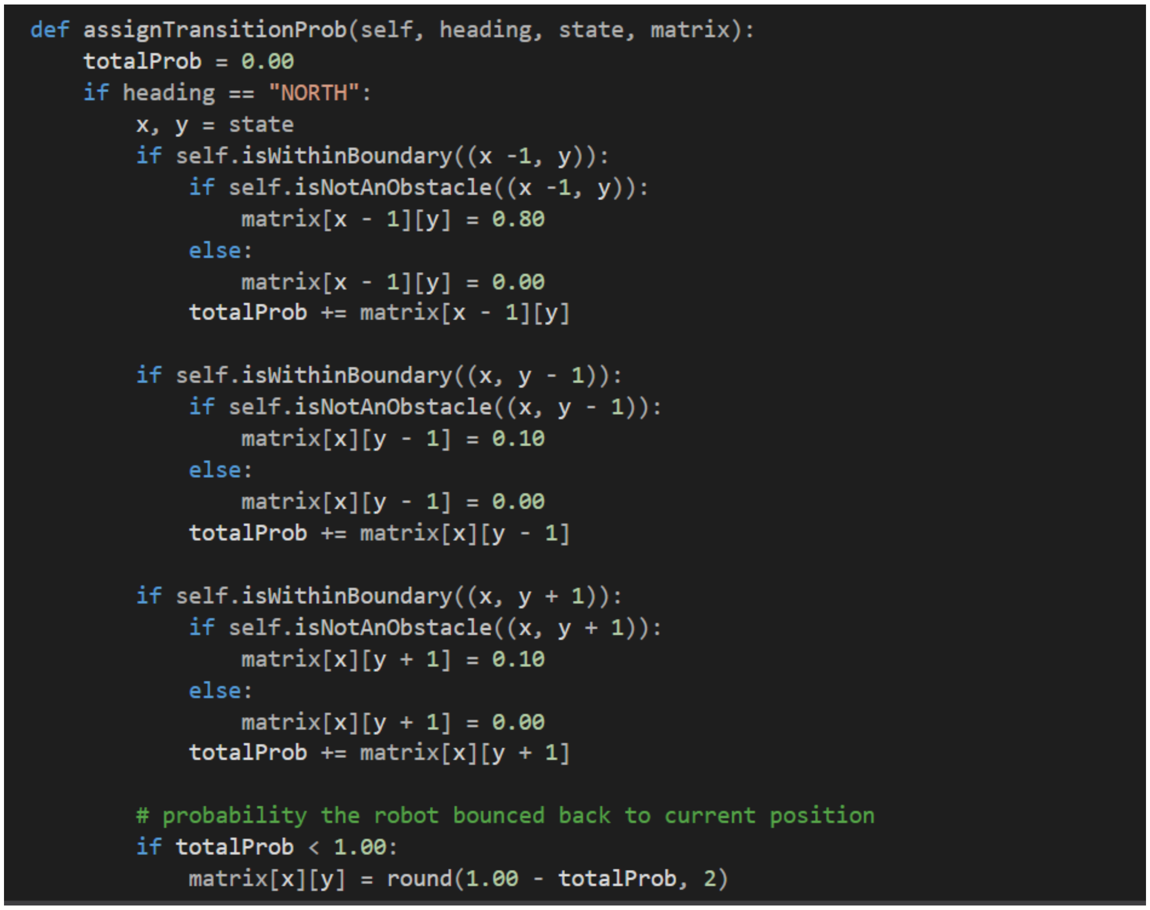

C:\Users\bhelf\AppData\Local\Programs\Python\Python39\python.exe

```
Prediction After Action: NORTH
['0.96', '0.03', '0.39', '0.00', '0.00', '0.00', '0.01']

['0.16', '0.15', '53.8', '0.05', '0.00', '0.00', '0.00']

['0.12', '0.00', '20.8', '6.73', '0.00', '9.99', '0.20']

['0.71', '0.00', '1.81', '1.76', '0.00', '1.07', '1.02']

['0.10', '0.09', '0.00', '0.01', '0.01', '0.00', '0.01']

['0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']

------------------------------------------------------------------

Filtering After Evidence: ('0', '0', '0', '0')
['0.04', '0.01', '0.10', '0.00', '0.00', '0.00', '0.00']

['0.04', '0.04', '90.3', '0.08', '0.00', '0.00', '0.00']

['0.00', '0.00', '5.51', '1.78', '0.00', '0.42', '0.05']

['0.03', '0.00', '0.48', '0.47', '0.00', '0.28', '0.27']

['0.03', '0.02', '0.00', '0.01', '0.00', '0.00', '0.00']

['0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00']

------------------------------------------------------------------
```

Transitional Probability



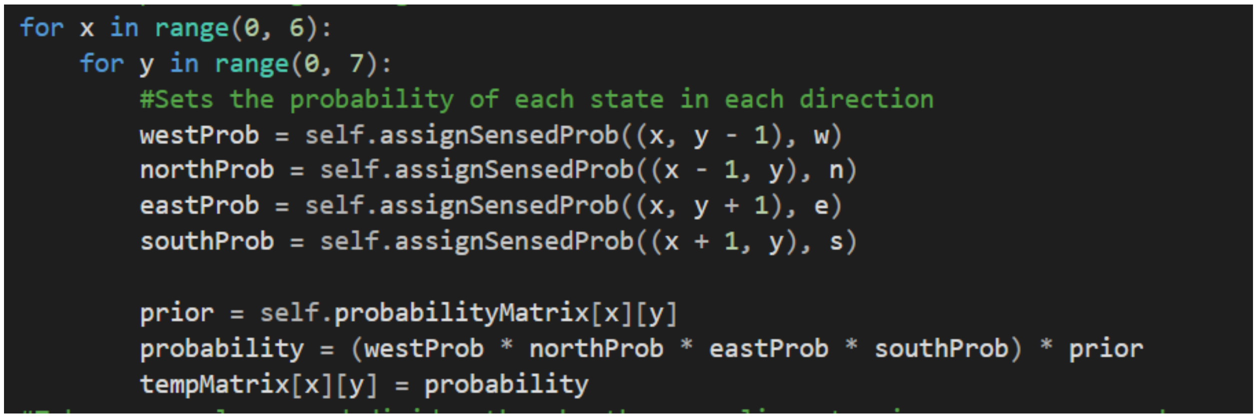
```python
def createTransitionMatrix(self, heading):

    #Create an empty matrix that is 6 X 7
    transitionMatrix = np.empty(shape=(1, 42))
    #For range in rows
    for x in range(0, self.height):
        #For range in columns
        for y in range(0, self.width):
            stateMatrix = np.array([[round(0.000, 2)] * self.width] * self.height)
            #Does condition if it is an actual state
            if self.isNotAnObstacle((x, y)):
                self.assignTransitionProb(heading, (x, y), stateMatrix)
                stateMatrix = stateMatrix.flatten()
                transitionMatrix = np.vstack((transitionMatrix, stateMatrix))
            else:
                stateMatrix = stateMatrix.flatten()
                transitionMatrix = np.vstack((transitionMatrix, stateMatrix))

    # Delete first row (irrelevant)
    transitionMatrix = np.delete(transitionMatrix, 0, 0)


    return transitionMatrix
```



```python
def assignTransitionProb(self, heading, state, matrix):
    totalProb = 0.00
    if heading == "NORTH":
        x, y = state
        if self.isWithinBoundary((x -1, y)):
            if self.isNotAnObstacle((x -1, y)):
                matrix[x - 1][y] = 0.80
            else:
                matrix[x - 1][y] = 0.00
            totalProb += matrix[x - 1][y]

        if self.isWithinBoundary((x, y - 1)):
            if self.isNotAnObstacle((x, y - 1)):
                matrix[x][y - 1] = 0.10
            else:
                matrix[x][y - 1] = 0.00
            totalProb += matrix[x][y - 1]

        if self.isWithinBoundary((x, y + 1)):
            if self.isNotAnObstacle((x, y + 1)):
                matrix[x][y + 1] = 0.10
            else:
                matrix[x][y + 1] = 0.00
            totalProb += matrix[x][y + 1]

        # probability the robot bounced back to current position
        if totalProb < 1.00:
            matrix[x][y] = round(1.00 - totalProb, 2)
```

I create the transitional matrix and then iterate through the amount of rows and columns to calculate the transitional probability of each.

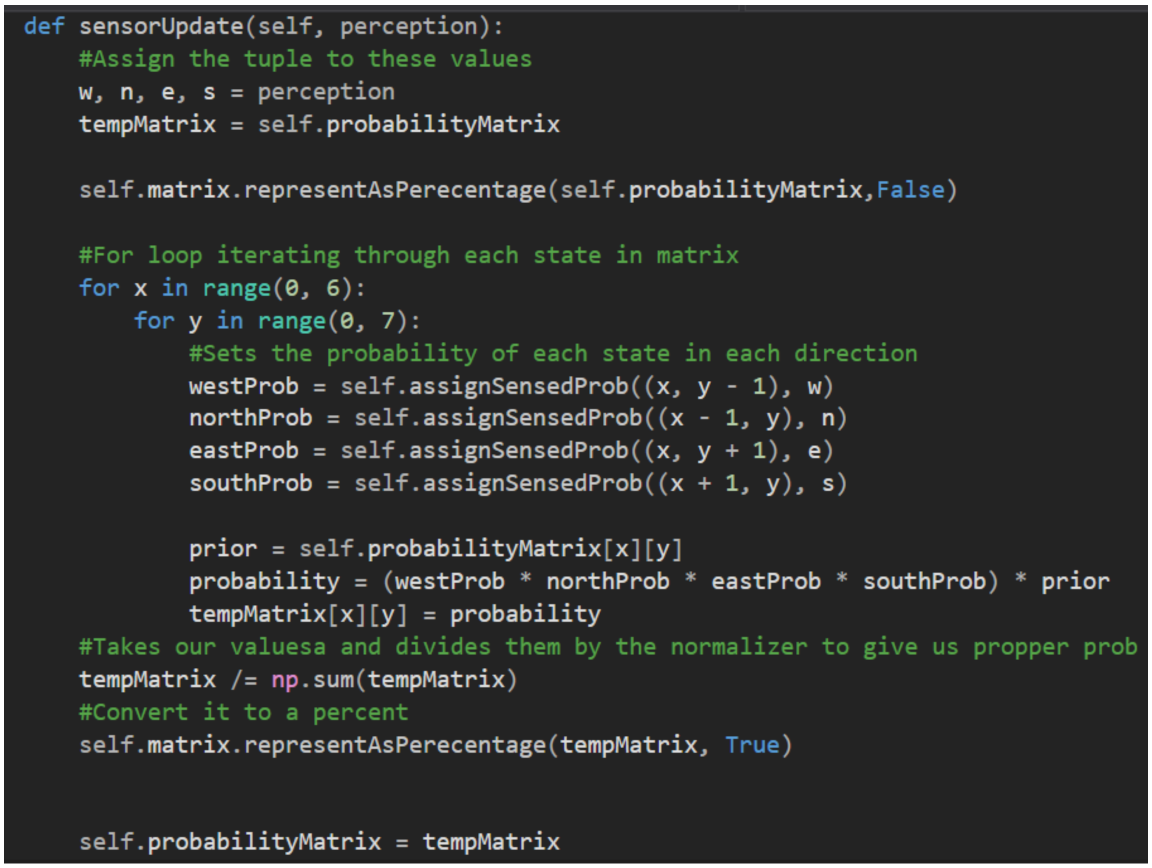
Evidence Conditional Probability



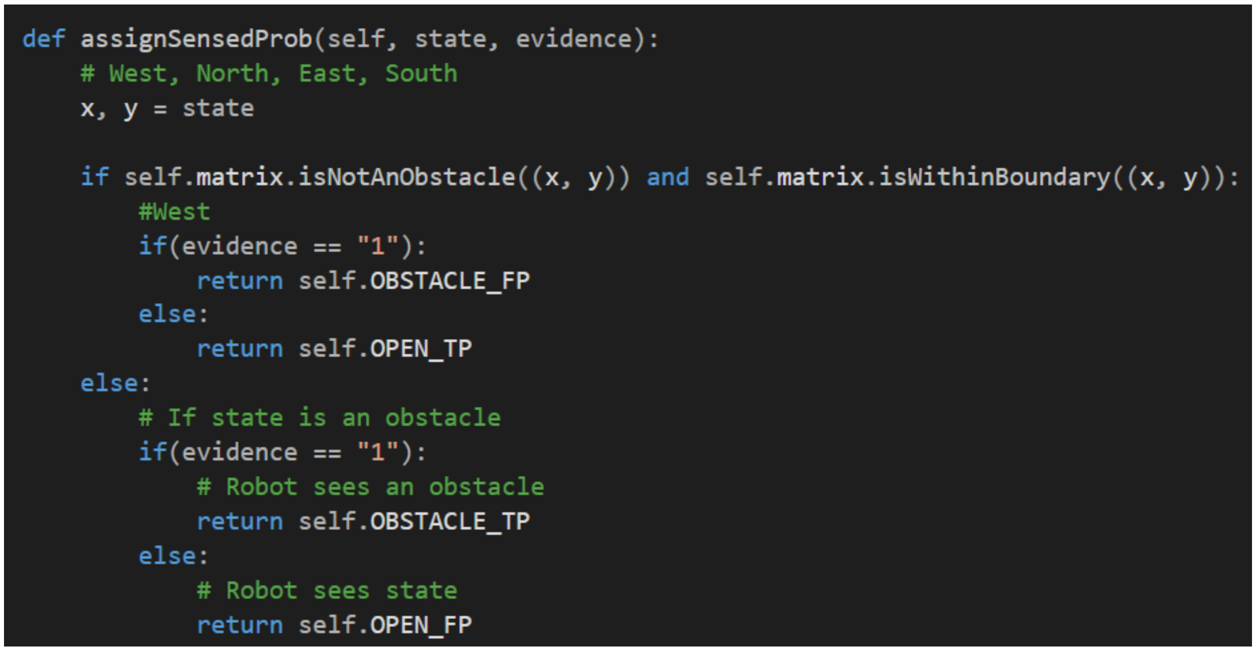
```
for x in range(0, 6):
    for y in range(0, 7):
        #Sets the probability of each state in each direction
        westProb = self.assignSensedProb((x, y - 1), w)
        northProb = self.assignSensedProb((x - 1, y), n)
        eastProb = self.assignSensedProb((x, y + 1), e)
        southProb = self.assignSensedProb((x + 1, y), s)

        prior = self.probabilityMatrix[x][y]
        probability = (westProb * northProb * eastProb * southProb) * prior
        tempMatrix[x][y] = probability
```

Probability based off of the information received from WNES

Filtering



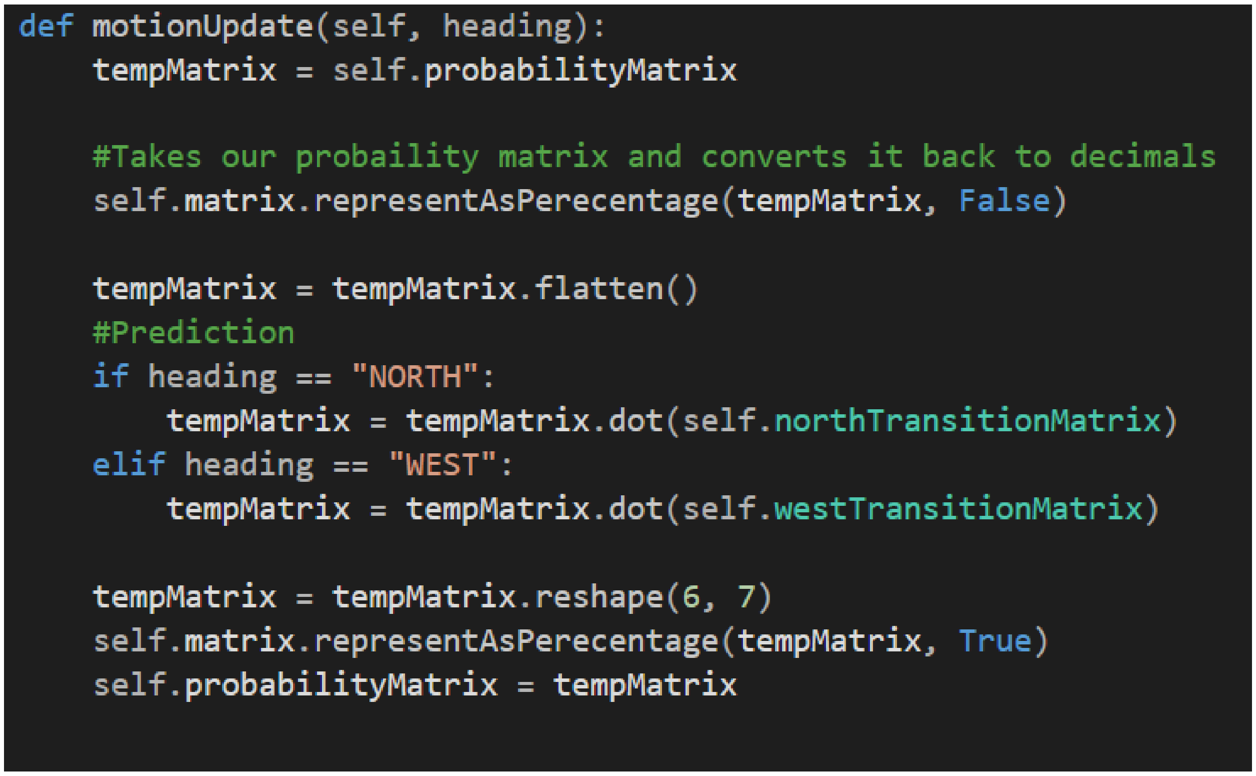
```
def sensorUpdate(self, perception):
    #Assign the tuple to these values
    w, n, e, s = perception
    tempMatrix = self.probabilityMatrix

    self.matrix.representAsPerecentage(self.probabilityMatrix,False)

    #For loop iterating through each state in matrix
    for x in range(0, 6):
        for y in range(0, 7):
            #Sets the probability of each state in each direction
            westProb = self.assignSensedProb((x, y - 1), w)
            northProb = self.assignSensedProb((x - 1, y), n)
            eastProb = self.assignSensedProb((x, y + 1), e)
            southProb = self.assignSensedProb((x + 1, y), s)

            prior = self.probabilityMatrix[x][y]
            probability = (westProb * northProb * eastProb * southProb) * prior
            tempMatrix[x][y] = probability
    #Takes our valuesa and divides them by the normalizer to give us propper prob
    tempMatrix /= np.sum(tempMatrix)
    #Convert it to a percent
    self.matrix.representAsPerecentage(tempMatrix, True)


    self.probabilityMatrix = tempMatrix
```



```python
def assignSensedProb(self, state, evidence):
    # West, North, East, South
    x, y = state

    if self.matrix.isNotAnObstacle((x, y)) and self.matrix.isWithinBoundary((x, y)):
        #West
        if(evidence == "1"):
            return self.OBSTACLE_FP
        else:
            return self.OPEN_TP
    else:
        # If state is an obstacle
        if(evidence == "1"):
            # Robot sees an obstacle
            return self.OBSTACLE_TP
        else:
            # Robot sees state
            return self.OPEN_FP
```

This step is the filtering step where we use the product rule

Prediction



```python
def motionUpdate(self, heading):
    tempMatrix = self.probabilityMatrix

    #Takes our probability matrix and converts it back to decimals
    self.matrix.representAsPerecentage(tempMatrix, False)

    tempMatrix = tempMatrix.flatten()
    #Prediction
    if heading == "NORTH":
        tempMatrix = tempMatrix.dot(self.northTransitionMatrix)
    elif heading == "WEST":
        tempMatrix = tempMatrix.dot(self.westTransitionMatrix)

    tempMatrix = tempMatrix.reshape(6, 7)
    self.matrix.representAsPerecentage(tempMatrix, True)
    self.probabilityMatrix = tempMatrix
```

Our prediction happens at our motion update function