**Team Members:**

Vaishnavi Anne ( Build the episode score, episode path calculation and comparing shortest and longest path of the agent. Also, worked on project report )

Anjan Shrestha ( Build agent next move logic using 80, 10, 10 percentage rule and out put next move. Also, worked on project report )

Approach:

* Used two dimensional array to store grid layout of 6x6.
* Create a loop for episodes of 1000
* For each episode, generated a random number between one and the total number of item types and used that number to get random order items
* Created an array of size 2 named as “current\_robot\_position” to store agent position where first index determines the row and second determines the columns of the agent position
* Created episode\_score and episode\_path variable to store current episode score and path of the agent
* Created a list of available items in the available\_items list
* Created shortest\_path, longest\_path, shortest\_path\_score, longest\_path\_score, total\_score to store its respective values
* When the agent finds the order, then it will remove the item from the list and iterates after the order item list is empty
* For each item in the list, if robot finds the item in the grid, then it will increment episode score by 3 and remove the item from the order item list. If not, then it will deduct 1 from episode\_score
* Next, it will try to identify the next move using current robot position, order items list and warehouse grid.
* To identify next move, we find the available position for next move based on current robot position, filters the position that are outside the grid and sense moves based on the sensor accuracy.
* To sense the move based on accuracy, first we generated the random number from 1 to 10 where 1-8 represents 80% of the time robot is accurate, 9 represents 10% of the time where agent is false positive where it sense items even if its not there and 10 represents another 10% of the time where agent is false negative where it doesn’t sense item even it its there.
* After finding the possible next moves list , we get the random move using random number and assign it to current robot position and append the position as a tuple to episode path.
* Update the shortest\_path and shortest\_path\_score if length of shortest\_path is greater than length of current episode path
* Update the longest\_path and longest\_ path \_score if length of longest\_path is less than length of current episode path
* Add the current episode score to the total score and use it to calculate average score
* Print shortest\_path, longest\_path, shortest\_path\_score, longest\_path\_score and average\_score to show the output

Assumptions:

1. Warehouse grid is 6 by 6.
2. Agent can move in to the grid that contains item
3. Number of items in a given order can be taken as a random number between 1 and total number of item types
4. 80% of the time the robot will correctly sense whether a shelf exists in its neighborhood.
5. Assume that the neighborhood consists of a maximum of 4 grid positions which are to the left, right, up and down from its current position.
6. 10% of the time the sensor fails, and the robot thinks that a shelf is present when it is not the case
7. 10% of the time a shelf exists but the sensor fails to detect it until it lands on the grid position containing the shelf.
8. If any of the grid positions in its neighborhood refers to an item ordered, then it moves to that grid position. If two or more neighboring positions contain ordered items, then the tie is broken by making a random choice between the positions involved.
9. If none of the neighboring positions contain ordered items, then a random choice is made to move to the next node.
10. For each episode, the order item list would be random list with random number of items.
11. Agent has memory of previously visited grid, so it doesnot have to go back to visited one
12. Program stops after executing 1000 episodes

Questions:

1. The average score taken across 1000 episodes. - done
2. Display the shortest path (i.e., the sequence of grid positions visited) across the 1000 episodes and its corresponding score. - done
3. Display the longest path across the 1000 episodes and its corresponding score. - done
4. Demonstrate that your program is generic enough to cope with a new layout of shelving as shown below. This should be done by carrying out requirements 1, 2 and 3 above for the new layout.
5. Give two possible reasons why the average score differs from the average score computed across the old layout.

- old layout didnt have sensor to sense the neighbour grid and it uses brute force that calculates every scenario which increased the average score

1. What data structure(s) did you use to ensure that your program can cope with any given layout?

- used two dimensional array.