POLICY DESIGN

The main objective here is that the ball should avoid the restart position as it unnecessary leads to increase in path cost. It then checks for the goto points to see where the path cost would be less to reach power point. Then once the agents goes to the power point then it checks the path to reach the goal point. In case the path cost using the goto points and power point is more than the cost of direct approach then it follows the direct approach avoiding all the goto and power points.

To get the path cost here I have used A* algorithm which is one of the popular path finding algorithm. The algorithm is:

1. First the open list is initialized.
2. Then the closed list is initialized.
3. While (open list is not empty)
   a) Find the node with the least **f** on
      Let it be "q"

   b) Then q is popped off the open list

   c) The q's 8 successors are generated and set their
      parents is to q

   d) For each of the successor generated:
      i) If successor generated is the goal itself, then the search is stopped.

      successor.**g** = q.**g** + distance between successor and q
      successor.**h** = distance from goal to successor (using Manhattan distance)
      successor.**f** = successor.**g** + successor.**h**

      ii) If we find a node with the same position as successor is in the open list which has a lower **f** than successor, then the successor is skipped.

      iii) If we find a node with the same position as successor is in the closed list which has a lower **f** than successor, then the successor is skipped.
      Otherwise, the node is added to the open list
      End (for loop)

   e) q is pushed on to the closed list
   End (while loop)

REWARD FUNCTION DESIGN

11	11	100	11	11	11	11	11
11	11	100	10	100	100	11	0
11	10	100	11	11	11	11	11
11	11	100	11	100	100	100	5
150	11	11	11	100	11	150	11
11	100	100	11	100	11	11	11
11	11	11	11	100	11	100	100
15	11	11	11	11	11	11	10

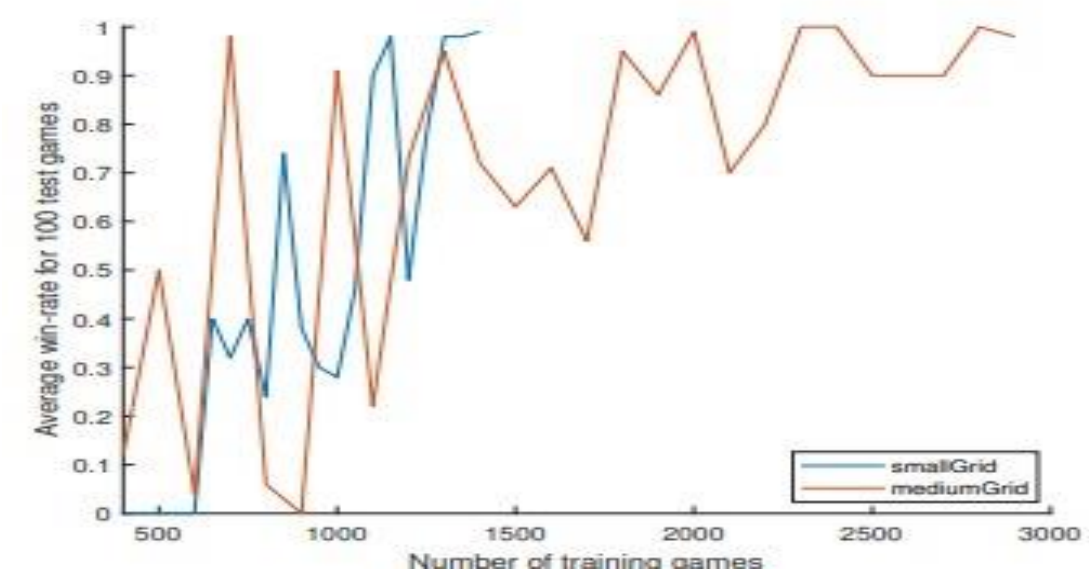
This is the main matrix according to which the reward of each state is defined. If the ball passes the certain state then respective reward is calculated. Where 100 is mentioned it is actually negative rewards which specifies us that it's the obstacle present there else other number present is positive in nature.

```
memory[current_state,action] = reward[current_state,action] + gamma*max_value
```

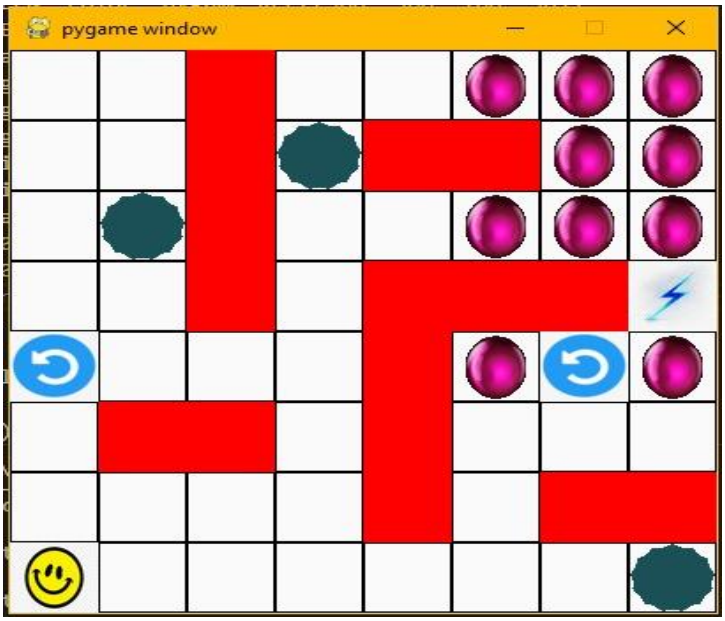
This is the main learning formula that has been used. Using this the agent learns the whole environment. If we increase the gamma value then learning increases but it's not advised to increase beyond certain limit else inaccurate results are generated.

KNOWLEDGE BASED FORMAT

The knowledge about the world is stored in memory matrix. The memory matrix is formed using the reward function as mentioned earlier. Here I have used 10000 iterations to form the whole memory matrix from scratch. Initially the matrix is zero and then it keeps on updating with every iteration.



The above graph depicts the win-rate against the trained iterations.



The above diagram shows us the available goal positions in the entire game screen among which it randomizes. Here the purple ball gives us all the possible goal positions as per the requirement.

Output of various iterations

memory matrix:

[[25. 50. 25.]

 [25. 50.75 100.]

 [25.75 50. 25.]

 [25. 50. 0.]

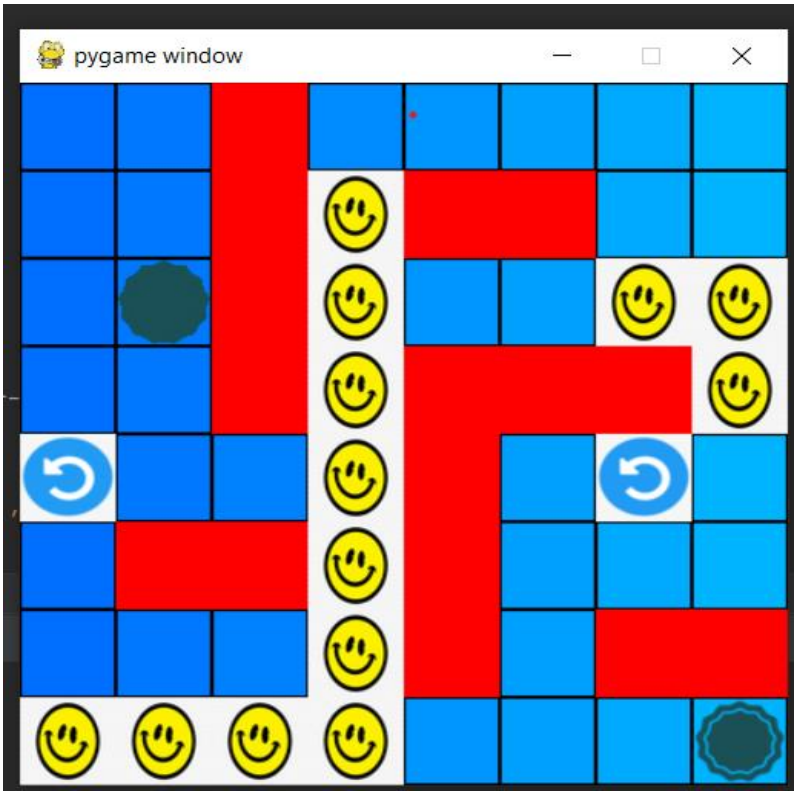
 [36.25 50. 40.]]

1st iteration

300 100

Path found: ['S', 'D2', 'H4', 'H3', 'G3']

path cost: 11



2nd iteration

350 200

Path found: ['S', 'H8', 'H4', 'H5']

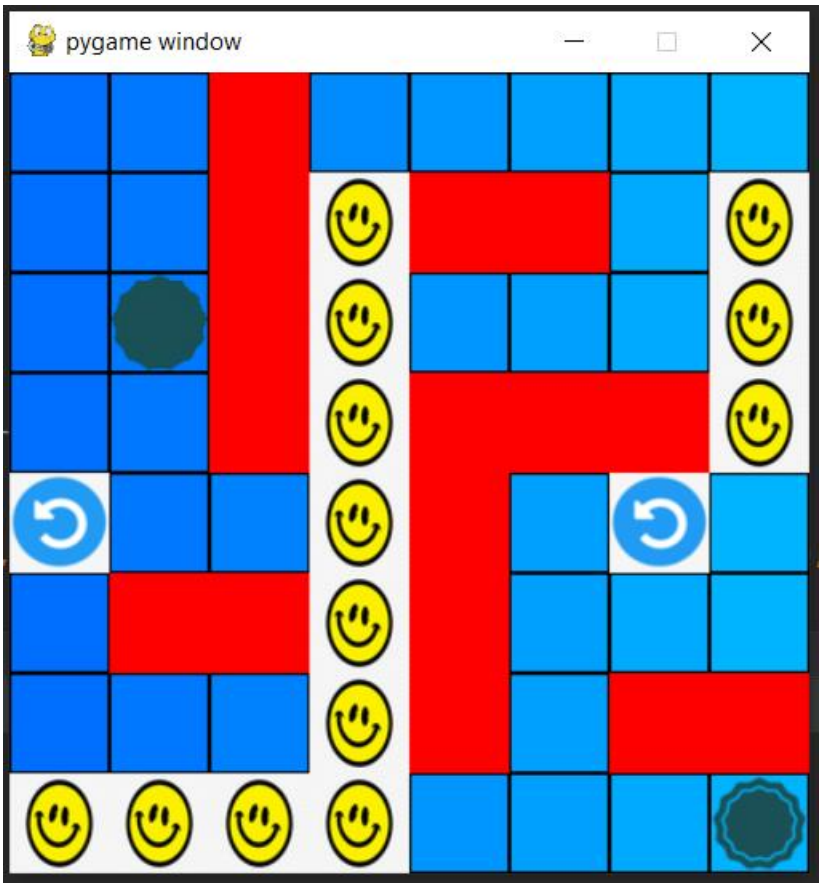
path cost: 8

3rd iteration

350 50

Path found: ['S', 'D2', 'H4', 'H3', 'H2']

path cost: 11



In the game purple ball appearing in each iteration is the final goal that the agent has to reach avoiding all obstacles. While the agent interacts with the environment it turns blue and then we get to know the exact path it follows to the goal.