CS362 Artificial Intelligence Endsem Lab Report

TEAM : Code_Brigrade

Patel Bhavik
202051134

Suraj Poddar
202051186

Tejas Gundale
202051191

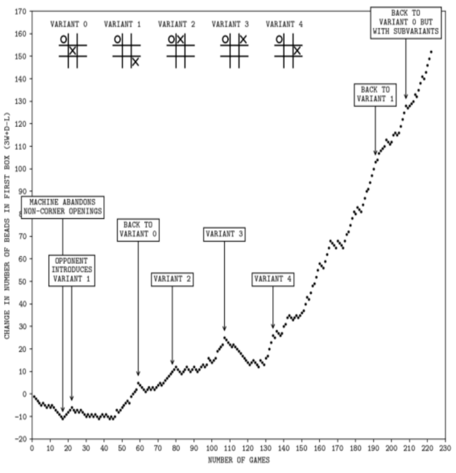
Tushar Maithani
202051194

***Abstract*—In this report, we completed lab assignment 8,9,10 and 11.Solution for each lab discussed in different sections, and their sub problems in respective subsection.**
Link to code repository

I. LAB ASSIGNMENT 8

A. Read the reference on MENACE by Michie and check for its implementations. Pick the one that you like the most and go through the code carefully. Highlight the parts that you feel are crucial. If possible, try to code the MENACE in any programming language of your liking.

Matchbox Educable Noughts And Crosses Engine, aka MENACE, was designed by Michie to play the game of tic-tac-toe against humans. It is trained by the means of reinforcement based computer learning method based on previously observed data i.e. the positive outcomes are fed back as positive reinforcements and negative outcomes are fed back in the form of negative reinforcements. Michie assigned a unique colour to every cell available on the board numbered the cells from 0-8(as shown in figure 8.1). He used matchboxes for labelling the different configurations of the board. In every matchbox, there were a certain number of beads of the colours of empty cells(the ones in which a move was possible). The computer made random moves in the training phase. After every move of the computer, he took out one bead of the same colour as that of the cell in which the computer had played. At the end of each game, there were 3 possible scenarios: computer wins, computer loses, draw. For each win of the computer, he added 3 beads of the same colour to those matchboxes from which he had removed one bead while playing. This increased the probability of achieving the same configuration later when the game was played again. For every game in which the computer lost, he removed 1 bead of the same colour from those matchboxes from which he had removed one bead while playing. This, on the other hand, decreased the probability of attaining the losing configuration. For draws, there was neither a reward nor a penalty.
One of the most critical stages in MENACE's training process is the function "menace-add-beads()". This function is responsible for building the foundation of the training model by adding beads to the relevant matchboxes, which ultimately enhances MENACE's performance as an opponent. Moreover, the functions "Update-plot()" and "redraw-plot()" aid in providing visual representation of the training model by plotting the curve at every step. These functions facilitate a better understanding of the training progress and make it easier to track the performance of MENACE throughout the training phase.



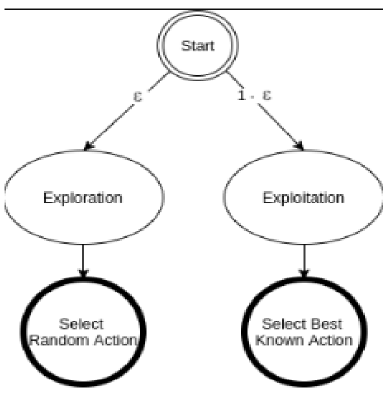
The implementation of MENACE by Matthew Scroggs is a good one to study the code.

- The functions play menace() and play opponent() are important for training the model as they check the validity of the current move made by MENACE and human respectively.
- The function menace add beads is the most crucial step of all as it helps in establishing the basic training model for the implementation by adding the beads to the corresponding matchboxes for increasing the model's efficiency as an opponent.
- The functions update plot() and redraw plot() help in plotting the curve at every step and helps in the visualization of the model.

II. LAB ASSIGNMENT 9

A. (1)Consider a binary bandit with two rewards 1-success, 0-failure. The bandit returns 1 or 0 for the action that you select, i.e. 1 or 2. The rewards are stochastic (but stationary). Use an epsilon-greedy algorithm discussed in class and decide upon the action to take for maximizing the expected reward. There are two binary bandits named binaryBanditA.m and binaryBanditB.m are waiting for you.
(2)Develop a 10-armed bandit in which all ten mean-rewards start out equal and then take independent random walks (by adding a normally distributed increment with mean zero and standard deviation 0.01 to all mean-rewards on each time step).
$function[value] = bandit\_nonstat(action)$
(3)The 10-armed bandit that you developed ($bandit\_nonstat$) is difficult to crack with a standard epsilon-greedy algorithm since the rewards are non-stationary. We did discuss how to track non-stationary rewards in class. Write a modified epsilon-greedy agent and show whether it is able to latch onto correct actions or not. (Try at least 10000 time steps before commenting on results)

1) **Exploration-Exploitation in Epsilon Greedy Algorithm:**
Exploitation is when the agent knows all his options and chooses the best option based on the previous success rates. Whereas exploration is the concept where the agent is unaware of his opportunities and tries to explore other options to better predict and earn rewards. As in the example above, going to the same pizza parlor and ordering the best pizza is an example of exploitation; here, the user knows all his options and selects the best option. On the other hand, going to the new Chinese restaurant and trying new things would come under exploration. The user is exploring new things; it could be better or worse, the user is unaware of the result.
2) **Epsilon Greedy Action Selection:**
The epsilon greedy algorithm chooses between exploration and exploitation by estimating the highest rewards. It determines the optimal action. It takes advantage of previous knowledge to choose exploitation, looks for new options, and select exploration.



3) **Epsilon greedy policy improvement**
The policy improvement is a theorem that states For any epsilon greedy policy π, the epsilon greedy policy π concerning qπ is an improvement. Therefore, the reward for π will be more. The inequality is because the max function returns a more excellent value than the arbitrary weighted sum.

$$\text{Action at time(t)} \begin{cases} \max Q_t(a) & \text{with probability } 1-\epsilon \\ \text{any action (a)} & \text{with probability } \epsilon \end{cases}$$

Pseudocode of Epsilon Greedy

```
Algorithm 2: Epsilon-Greedy Action Selection
Data: Q: Q-table generated so far, : a small number, S: current
      state
Result: Selected action
Function SELECT-ACTION(Q, S, ε) is
    n ← uniform random number between 0 and 1;
    if n < ε then
        A ← random action from the action space;
    else
        A ← maxQ(S,.);
    end
    return selected action A;
end
```

4) **Multi-Armed Bandit Problem**
The multi-armed bandit problem is used in reinforcement learning to formalize the notion of decision-making under uncertainty. In a multi-armed bandit problem, an agent(learner) chooses between k different actions and receives a reward based on the chosen action. The multi-armed bandits are also used to describe fundamental concepts in reinforcement learning, such as rewards, timesteps, and values. For selecting an action by an agent, we assume that each action has a separate distribution of rewards and there is at least one action that generates maximum numerical reward. Thus, the probability distribution of the rewards corresponding to each action is different and is unknown to the agent(decision-maker). Hence, the goal of the agent is to identify which action to choose to get the maximum reward after a given set of trials.
5) **Solution of the Multi-Armed Bandit Problem :**

- To solve this problem, instead of dealing with distributions caused by certain actions, we need to deal with one number that will somehow quantify the distribution. Every distribution has an expected or a mean value. Consequently, expected or mean values are a good choice for tackling this problem. Let us first introduce a few definitions.
- The number of arms is denoted by N.
- The action is denoted by a_i, where i=1,2,. . . , N. The

action a_i is actually the selection of the arms i.

- The action selected at the time step k is denoted by A_k. Note that A_k is a random variable.
- The action A_k can take different numerical values a_1, a_2,..,$\ldots$,a_N. A numerical value of the action i is denoted by a_i.
- Every action A_k at the time step k, produces a reward that we denote by R_k. Note that R_k is a random variable. The expected or the mean reward of the action we take is referred to as the value of that action. Mathematically speaking, the value is defined by

$$v_i = v(a_i) = E[R_k|A_k = a_i]$$

Now, if we would know in advance the values

of actions v_i, i=1,2,3,$\ldots$, N, our task would be completed: we would simply select the action that produces the largest value!

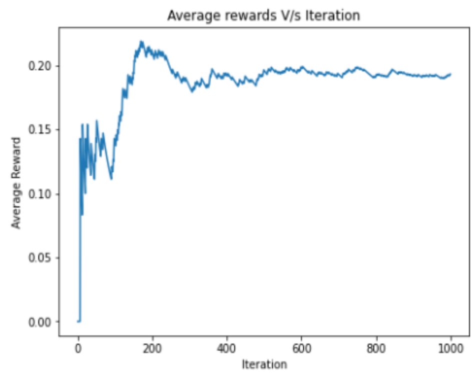
For that purpose, we will use an action value method. The action values method aims at estimating the values of actions and using these estimates to take proper actions that will maximize the sum or rewards.

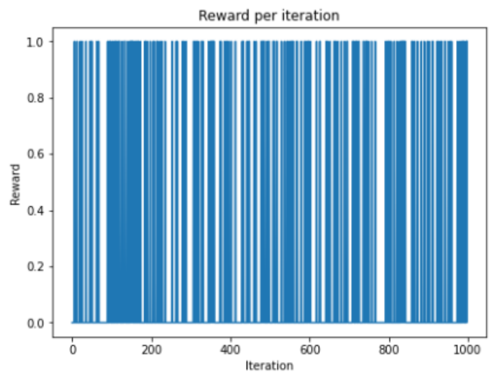
The story goes like this. At every time step k we can select any of the actions a_i. The action that is selected at the time step is denoted by A_k. The reward produced by this action is R_k. So, as the discrete-time k progresses, we will obtain a sequence of actions A_1, A_2,$\ldots$, and a sequence of produced rewards R_1, R_2,. From this data, we want to estimate a value of every action, that is denoted by $v_i = v(a_i)$. We can estimate these values at the time step k, as follows

$$v_{i,k} = v_i a_i = \sum_{j=1}^{k-1} R_{i,j}$$

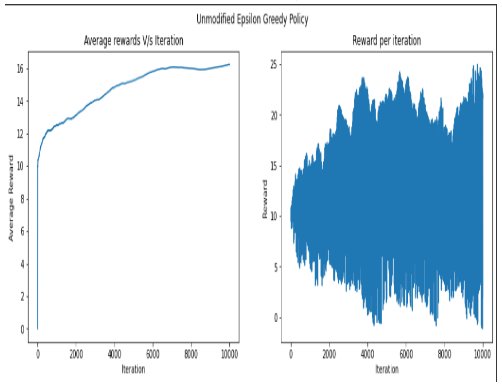
6) Results :

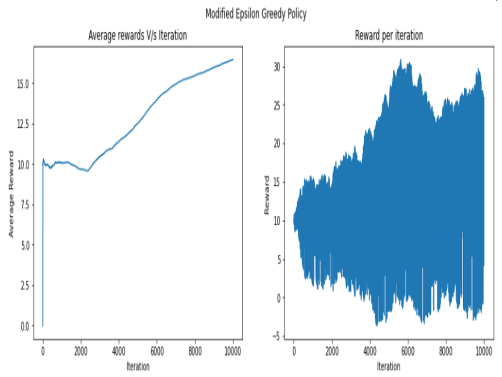
a) Result for o/1 bandit :





b) Result for N bandit :





III. LAB ASSIGNMENT 10

A. Suppose that an agent is situated in the 4x3 environment as shown in Figure 1. Beginning in the start state, it must choose an action at each time step. The interaction with the environment terminates when the agent reaches one of the goal states, marked +1 or -1. We assume that the environment is fully observable, so that the agent always knows where it is. You may decide to take the following four actions in every state: Up, Down, Left and Right. However, the environment is stochastic, that means the action that you take may not lead you to the desired state. Each action achieves the intended effect with probability 0.8, but the rest of the time, the action moves the agent at right angles to the intended direction with equal probabilities. Furthermore, if the agent bumps into a wall, it stays in the same square. The immediate reward for moving to any state (s) except for the terminal states S+ is r(s)= -0.04. And the reward for moving to terminal states is +1 and -1 respectively. Find the value function corresponding to the optimal policy using value iteration.
Find the value functions corresponding optimal policy for the following: (a)r(s)=-2 (b)r(s)=0.1 (c)r(s)=0.02 (d)r(s)=1.

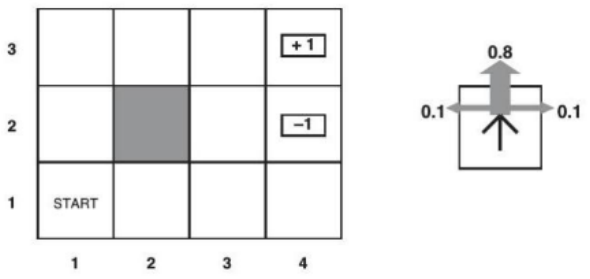


Fig. 1. 4 * 3 grid world with uncertain stage change.

To find the value function corresponding to the optimal policy using value iteration, we need to follow the below steps:
1.Initialization: Set the value of all states to 0 except for the terminal states, whose values are set to +1 and -1 respectively.
2.Iterative update: Update the value of each state as the sum of the immediate reward and the discounted value of the next state, where the discount factor is 0.99. We will perform this step iteratively until the values of all states converge.
3.Policy extraction: Extract the optimal policy from the value function by choosing the action that maximizes the expected value in each state.
Lets perform the above steps in detail,
The Bellman optimality equation for the value function of an MDP is given by:

$$V_\pi(s) \ = \sum_{r,s'} p(s', r|s, \pi(s)) * (r + \gamma V_\pi(s'))$$

where $V_\pi(s)$ is the optimal value function, a is the action taken in state s, $P(s'|s,a)$ is the probability of transitioning to state s' given action a in state s, r is the reward obtained from transitioning to state s' from s with action a, and γ is the discount factor.
To use value iteration, we start with an arbitrary initial value function V0, and iteratively improve it until it converges to the optimal value function V_π. The update rule for value iteration is given by:

$$V_\pi^{k+1}(s) \ = \ max(s) \sum_{r,s'} p(s', r|s, \pi(s)) * (r + \gamma V_\pi^k(s'))$$

where V_π^k is the value function at iteration k.
Using the above equations, we can compute the optimal value function and policy for the given MDP. We start by initializing the value function for all states as 0, except for the terminal states which have a value of +1 and -1 respectively. We choose a discount factor of gamma = 0.99, and a threshold of epsilon = 0.0001 for convergence.
Here is the pseudocode for value iteration:

```
V(s) = 0 for all s
while True:
-       delta = 0
-       for each state s:
-             v = V(s)
-             V(s) = $max_a(\sum_{s',r} P(s', r|s, a)[r + \gamma * V(s')])$
-             delta = $max(delta, |v - V(s)|)$
-       if delta ¡ epsilon:
-             break
```

B. [Gbike bicycle rental] You are managing two locations for Gbike. Each day, some number of customers arrive at each location to rent bicycles. If you have a bike available, you rent it out and earn INR 10 from Gbike. If you are out of bikes at that location, then the business is lost. Bikes become available for renting the day after they are returned. To help ensure that bicycles are available where they are needed, you can move them between the two locations overnight, at a cost of INR 2 per bike moved.
Assumptions: Assume that the number of bikes requested and returned at each location are Poisson random variables. Expected numbers of rental requests are 3 and 4 and returns are 3 and 2 at the first and second locations respectively. No more than 20 bikes can be parked at either of the locations. You may move a maximum of 5 bikes from one location to the other in one night. Consider the discount rate to be 0.9.
Formulate the continuing finite MDP, where time steps are days, the state is the number of bikes at each location at the end of the day, and the actions are the net number of bikes moved between the two locations overnight.

To formulate the continuing finite MDP we need following attributes:

1.State space: The state space is the set of all possible combinations of bikes at the two locations at the end of the day. Since we cannot have more than 20 bikes at each location, the state space can be defined as {0, 1, 2, ..., 20} x {0, 1, 2, ..., 20}.
2.Action space: The action space is the set of all possible net number of bikes that can be moved between the two locations overnight. Since we can move a maximum of 5 bikes from one location to the other in one night, the action space can be defined as {-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5}.
3.Reward function: The reward function is defined as the immediate reward that the agent receives for taking a particular action in a particular state. In this case, if we have a bike available, we rent it out and earn INR 10 from Gbike. If we are out of bikes at that location, then the business is lost. Hence, the reward function can be defined as follows:
If number of bikes is positive, then reward is 10 times the number of bikes.
If number of bikes is negative, then reward is 0.
If number of bike is zero, then reward is 0.
4.State transition probabilities: The state transition probabilities describe the probability of moving from one state to another given an action. In this case, we assume that the number of bikes requested and returned at each location are Poisson random variables. Expected numbers of rental requests are 3 and 4 and returns are 3 and 2 at the first and second locations respectively. Bikes become available for renting the day after they are returned. Hence, the state transition probabilities can be defined as follows:
- For each possible state and action, we calculate the expected number of bikes at each location the next day, based on the Poisson distribution.
- We then calculate the probability of moving from the current state to each possible next state, given the action and the expected number of bikes at each location the next day.
Once we have defined the above components, we can use value iteration or policy iteration to find the optimal policy for this MDP. The optimal policy will tell us, for each possible state, what action to take in order to maximize the expected discounted future rewards.
You can compare this formulation with the code provided in the gbike.zip file to better understand how the MDP is solved using policy iteration. Policy iteration is an iterative algorithm that alternates between the following two steps:
1.Policy evaluation: Given a policy, we calculate the value function for each possible state. The value function is the expected discounted future reward if we start in a particular state and follow the policy thereafter.
2.Policy improvement: Given the value function for each possible state, we update the policy to choose the action that maximizes the expected discounted future reward.

We repeat these two steps until the policy converges to the optimal policy.

C. Write a program for policy iteration and resolve the Gbike bicycle rental problem with the following changes. One of your employees at the first location rides a bus home each night and lives near the second location. She is happy to shuttle one bike to the second location for free. Each additional bike still costs INR 2, as do all bikes moved in the other direction. In addition, you have limited parking space at each location. If more than 10 bikes are kept overnight at a location (after any moving of cars), then an additional cost of INR 4 must be incurred to use a second parking lot (independent of how many cars are kept there)

To incorporate the changes in the Gbike bicycle rental problem, we need to modify the policy evaluation step to take into account the additional costs of moving bikes and using a second parking lot:
1.Initialize the policy arbitrarily.
2.Repeat until convergence:
a. Policy Evaluation: Given the current policy, calculate the state-value function V(s) for all states s in the MDP as follows:
i. For each state s, calculate the expected immediate reward and the expected next state value for each possible action a.
ii. Use the Bellman equation to update the state-value function V(s) for each state s as the maximum expected value over all possible actions a
iii. Repeat steps i-ii until V(s) converges
b. Policy Improvement: Given the current state-value function V(s), update the policy to be greedy with respect to V(s) as follows:
i. For each state s, calculate the expected immediate reward and the expected next state value for each possible action a
ii. Update the policy to select the action that maximizes the expected value over all possible actions a
c. Check if the policy has converged. If it has, return the optimal policy and state-value function
3.Return the optimal policy and state-value function

IV. LAB ASSIGNMENT 11

A. We all have seen, at least in advertisements, washing machines work. Some of us have used them also. Don't feel surprised if I tell you that it was one of the initial devices to have used fuzzy logic. In this laboratory, we will try to identify the appropriate time needed to wash the load of clothes, given the dirtiness and the volume of the load. This is a simplified design, in practice a lot more variables are involved including: water level, amount of detergent to be dispensed, and temperature of water (recent development) and variables that you and I may imagine in future.

We have referred [19][20] for this lab exercise.
Input : Dirtiness of clothes and Load Volume

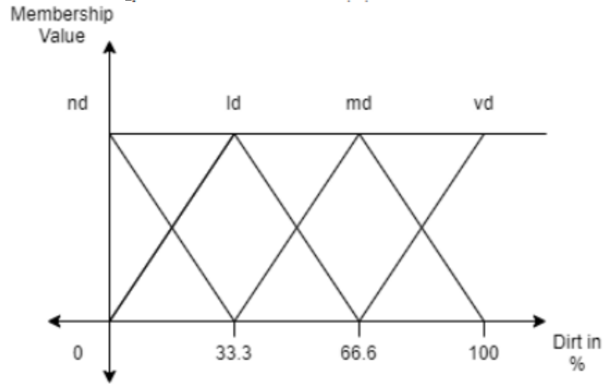
Output : Time
Steps to solve the problem -

– List out input and output variables
– Defining membership functions for each input and output variable
– Forming Rule Base
– Rule Evaluation
– Defuzzification

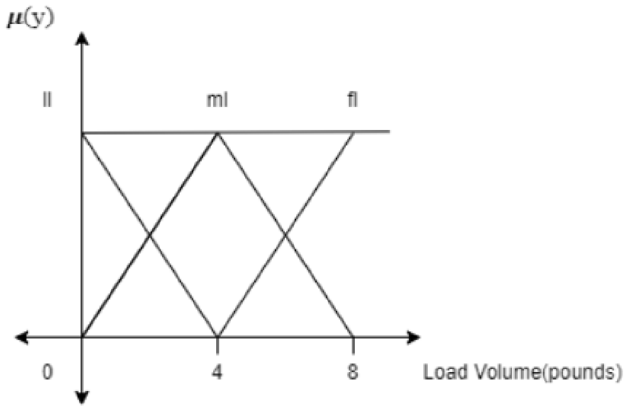
Let input for the dirtiness of clothes be in the form of percentage, load volume be in form of weight in pounds and wash time be measured in minutes.
Descriptors for the input
Dirtiness:
vd : very dirty
md : medium dirty
ld : light dirty
nd : not dirty
vd, md, nd

Load Volume:
fl : full load
ml : medium load
ll : low load
fl, ml, ll

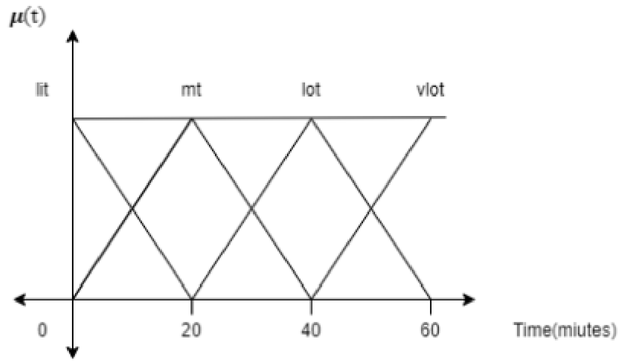
Descriptors for outputTime:
vlot : very long time
lot : long time
mt : medium time
lit : little time
vlot, lot, mt, lit
We will be using triangular membership functions for this problem.
Membership functions for Dirt(x) -



Membership value(μ(x))
μ(x) for nd : (33.3-x)/(33.3)
μ(x) for ld :
if 0¡x ¡33.3, then : (x)/(33.3)
if 33.3¡x¡66.6, then : (66.6-x)/(33.3)
μ(x) for md :
if 33.3¡x¡66.6, then : (x-33.3)/(33.3)
if 66.6¡x¡100, then : (100-x)/(33.3)
μ(x) for vd : (x-66.6)/(33.3)
Membership Functions for Load Volume(y)-
Assuming 8 pounds is the maximum weight that the washing machine can take at once.



μ(y) for ll : (4-y)/4
μ(y) for ml :
if 0¡y¡4, then : y/4
if 4¡y¡8, then : (8-y)/4
μ(y) for fl : (y-4)/4
Membership Functions for Time Taken(t)-
Assuming that the maximum time taken to clean the clothes is 40 minutes.



μ(t) for lit : (20-t)/(20)
μ(t) for mt :
if 0¡t¡20, then : (t)/(20)
if 20¡t¡40, then : (40-t)/(20)
μ(t) for lot :
if 20¡t¡40, then : (t-20)/(20)
if 40¡t¡60, then : (60-t)/(20)
μ(t) for vlot : (t-40)/(20)

Rule Base:

Load Dirtiness/ Load Volume	**vd**	**md**	**ld**	**nd**
fl	vlot	vlot	lot	lit
ml	vlot	mt	mt	lit
ll	lot	lot	lit	lit

Let us take an example now :
Dirt = 60 Load Volume = 5 pounds

Time Required : ?
Rule Evaluation :
60 % dirt maps to two membership functions for dirtμ(x) for ld :
if 33.3¡x¡66.6, then : (66.6-x)/(33.3)
μ(x) for md :
if 33.3¡x¡66.6, then : (x-33.3)/(33.3)
μ(60) for ld = (66.6 - 60)/(33.3) = 0.198 =¿ 1/5 (approx)
μ(60) for md = (60-33.3)/(33.3) = 0.801 =¿ 4/5 (approx)
5 pounds of Load Volume maps to two membership functions for Load Volumeμ(y) for ml :
if 4¡y¡8, then : (8-y)/4
μ(y) for fl : (y-4)/4
μ(5) for ml = (8-5)/4 = 3/4
μ(5) for fl = (5-4)/4 = 1/4

Therefore we need to evaluate four rules from the Rule Base:Dirt is md and Load Volume is ml
• Dirt is md and Load Volume is fl
• Dirt is ld and Load Volume is ml
• Dirt is ld and Load Volume is f
Since each rule is connected by ‘and’ operator, we will use min value to find the strength of each rule. Let S be strength.
Rule 1(md, ml): S1 = min(4/5, 3/4) = 4/5
Rule 2(md, fl): S2 = min(4/5, 1/4) = 1/4
Rule 3(ld, ml): S3 = min(1/5, 3/4) = 1/5
Rule 4(ld, fl): S4 = min(1/5, 1/4) = 1/5
Among these strength values we need to find the maximum strength to find the rule that is to be applied : Max(4/5, 1/4, 1/5, 1/5) = 4/5
Rule 1 has the maximum strength so we need to follow this rule which implies :
Dirt is md(medium dirty) and Load Volume is ml(medium load)
Corresponding to the above result, we need to call ‘mt’ membership function to calculate the time required. 4/5 is the membership value for the time taken function.
μ(t) for mt :
if 0¡t¡20, then : (t)/(20)
if 20¡t¡40, then : (40-t)/(20)
Now, equating each of the conditions with the strength we get,
4/5 = t/20
So, t = 16
And also,
4/5 = (40-t)/20
So, t = 24
Lastly, we need to take the average to find the time required : (16+24)/2 = 20
Therefore, 20 minutes is the time required to clean 5 pounds of clothes which are 60 % dirty.
Similarly, we can find time required for any input of dirt percentage and weight of the clothes.

V. Conclusion

To conclude, over the period of last four months our team has explored, analysed and studied the nature of problems that are all in the domain of (or can be approached as) Artificially Intelligent Systems. We learnt seeing a problem and identifying it into simpler components (the environment, the agent, sub-routines, etc.). It has given us a lingering taste of the subject, which we want more of, and therefore we will keep expanding our ”frontier” in this subject till our appetite allows.

VI. Acknowledgment

We would like to thank our professor, Dr. Pratik Shah for introducing us to these problems to explore with and helping us throughout the course of solving them. Also, our peers who helped wherever necessary.

References

[1] Khemani, D., 2013. A first course in artificial intelligence. McGraw-Hill Education.
[2] Reinforcement Learning: an introduction by R Sutton and A Barto (Second Edition)
[3] GitHub Repository for MENACE by Matthew Scroggs
[4] Sutton, R. S., Barto, A. G. (2018). Reinforcement Learning: An Introduction. The MIT Press.