Lab Assignment 7

LEARNING OBJECTIVE:

Learning Objective: Basics of data structure needed for state-space search tasks and use of random numbers required for MDP and RL, Understanding Exploitation - Exploration in simple n-arm bandit reinforcement learning task, epsilon-greedy algorithm for state-space search tasks and use of random numbers required for MDP and RL.

MENACE

A. Understanding MENACE:

MENACE stands for Machine Educable Noughts and Crosses Engine I. It was first explained by Donald Michie, who demonstrated how this system could learn to play the game of Noughts and Crosses (Tic-Tac-Toe). Michie used physical matchboxes to represent possible game moves. Each matchbox symbolized a specific state in the game, and MENACE learned from wins, losses, and draws by adjusting the beads in each box, which corresponded to move choices.

B. The Matchbox Machine

The matchboxes used in MENACE contained different colored beads. Each bead color represented a potential move that could be made in a given game state. By modifying the number of beads for each color, MENACE adapted its strategy over time, improving its performance based on prior games.

1 WHITE	2 LILAC	3 SILVER
8 BLACK	0 GOLD	4 GREEN
7 AMBER	6 RED	5 PINK

Fig. 1: Standard colour code.

C. MENACE's Learning Strategy

MENACE learns through a simple **reward and punishment** mechanism based on moves made during the game, represented by beads in matchboxes. Each bead color corresponds to a potential move in a specific game state.

Loss (Punishment)

- After a loss, MENACE **removes one bead** of the color corresponding to each move made during the game.
- **Effect on Learning:** This reduces the likelihood of choosing the same move in future games, as fewer beads mean a lower chance of selection.

Win (Reward)

- A win results in MENACE **adding three beads** of the winning move's color to the relevant boxes.
- **Effect on Learning:** This reinforces successful moves, increasing their chance of being selected again.

Draw (Neutral Outcome)

- In the event of a draw, MENACE **adds one bead** of the color for each move.
- **Effect on Learning:** This mildly encourages keeping draw-related moves in the strategy pool without favoring them heavily.

Long-Term Adaptation

Over time, losing moves are discarded as beads are removed, and successful moves are reinforced through added beads. If a box runs out of beads for a certain move, MENACE stops selecting that move entirely, indicating it has "given up" on that option.

Resetting MENACE

When early game boxes run out of beads, MENACE can be **reset** by adding more beads to allow further learning and exploration, especially when facing more skilled opponents.

STAGE OF PLAY	NUMBER OF TIMES EACH COLOUR IS REPLICATED
1	4
3	3
5	2
7	1

Fig. 2: Variation of the number of colour-replicates of a move according to the stage of play.

D. RESULTS

We tried to implement MENACE using python, it's code can be found here.

N-ARM BANDIT

Problem : 1

A binary bandit operates by offering two outcomes for each action: either a success (reward of 1) or a failure (reward of 0). The outcomes are determined by a stochastic process, meaning the rewards vary based on probability, but remain consistent in the long run. Using the epsilon-greedy algorithm (as covered in class), the goal is to select the action that maximizes the expected reward. This method balances exploration (trying new actions) with exploitation (choosing known actions that provide the highest reward). Two binary bandits, BinaryBanditA and BinaryBanditB, have been analyzed:

- Bandit A: Initially, with low probability of success, the expected rewards started at zero. As the trials continued, the rewards gradually increased.
- Bandit B: With a higher probability of success (0.8 and 0.9), the expected rewards were close to one from the start. Over time, the rewards stabilized as more trials were averaged.

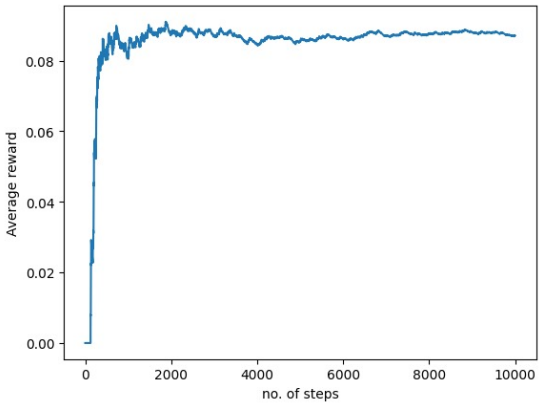

Fig. 3: Performance of Bandit A and Bandit B over trials.

Problem : 2

In each iteration of the problem, a ten-value array is generated, which follows a normal distribution with a mean of 0 and a standard deviation of 0.01. This array is added to the mean-reward array, modifying the rewards for each action. Since the rewards are non-stationary (changing over time), the reward policy for each action adjusts as the steps increase. Initially, all actions offer equal rewards (e.g., around 1), but as time progresses, the expected rewards begin to rise rapidly due to the non-stationary nature of the rewards.

Problem : 3

In this scenario, a 10-armed non-stationary bandit (bandit-nonstat) presents a unique challenge for the standard epsilon-greedy algorithm because the rewards vary over time. In class, we discussed handling non-stationary environments by updating the reward estimation differently.

To address this, a modified Epsilon-greedy agent is introduced, which uses an alpha parameter of 0.7. This agent

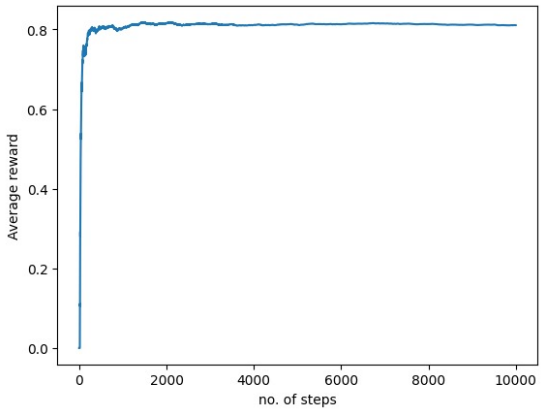

Fig. 4: Excepted Results we get when rewards are non-stationary(standard epsilon-greedy algorithm)

gives more weight to the current reward rather than relying on the traditional averaging method for updating the reward estimations. The use of the alpha parameter allows the agent to adapt more quickly to changes in the reward structure, which is crucial for non-stationary problems.

Compared to the previous case (Section A), where the expected rewards were lower, the modified agent shows a steeper rise in rewards. By placing more emphasis on recent rewards, the agent successfully adjusts to the dynamic environment and rapidly identifies the best actions. The graph reflecting this modified approach demonstrates a significantly higher and sharper increase in expected rewards than the stationary case.

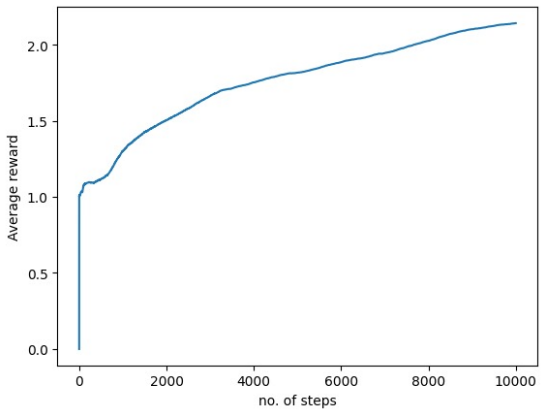

Fig. 5: Excepted Results we get when rewards are non-stationary(modified epsilon-greedy algorithm)

D. RESULTS

We have implemented N-arm problem(problem1, problem2, problem3) using python, it's code can be found here