Redmond Pie

TEAM MEMBERS

- Apoorv Pathak 202051033
- Aryan Gupta 202051038
- Tejas Joshi 202051091
- Kamal kumar 202051098

Learning Objectives :

- Basics of data structure needed for state-space search tasks and use of random numbers required for MDP and RL .

Title :

- Matchbox Educable Naughts and Crosses Engine.

State Space Search

- Process for modeling sequential decision making problems.
- Explore the Space of possible states and action to find optimal solution.

A search problem consists of

- A State Space A state is a set of properties that define the current conditions of the world our agent is in.
- A Start Space is the state the agent begins in.
- A Goal State is a state where the agent may end the search.
- An agent may take different actions that will lead the agent to new states.

Properties of Search Algorithms:

- Completeness
- Optimality
- Time Complexity
- Space Complexity

Type of Search Algorithm

Uninformed Search

- These methods have **no information** about which nodes are on promising paths to a solution. also called **blind search**.
- Uninformed Search Basic Fundamental Algorithm are:
    1) Depth First Search (DFS)
    2) Breadth First Search (BFS)

Informed Search

- The algorithms have information on the **goal state**, which helps in more efficient searching. This information is obtained by something called a **heuristic**.
- Informed Search Basic Fundamental Algorithm are:
    1) A* Search
    2) Greedy Search

Markov Decision Process (MDP)

Markov Decision Process (MDP) is a mathematical framework for modeling decision-making in situations where outcomes are partially random and partially under the control of a decision-maker.

- State space (S): a set of all possible states that the agent can be in. The agent interacts with the environment.
- Action space (A): a set of all possible actions that the agent can take in each state. The agent selects an action based on its current state.
- Transition probability function (T): a function that specifies the probability of transitioning from one state to another state when the agent takes a particular action. It is denoted by T(s, a, s') = P(s' | s, a), which gives the probability of transitioning to state s' from state s when the agent takes action a.
- Reward function (R): a function that specifies the reward the agent receives when it takes a particular action in a particular state. It is denoted by **R(s, a).**
- Discount factor (γ): a scalar value that determines the importance of future rewards relative to immediate rewards. (It is used to discount future rewards so that they have less influence on the agent's decision-making process.)

State space Search and MDP

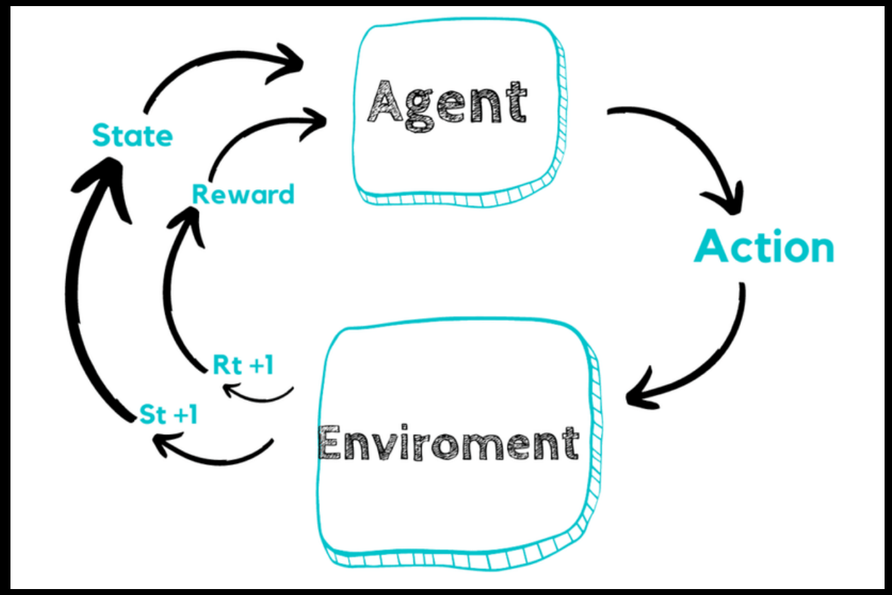
- Deterministic and non- deterministic.

- State space involves constructing a tree or graph that represents the state space of the problem, with nodes representing states and edges representing actions. The goal is to find a path through the state space that leads to a solution.
- MDP is a mathematical framework used to model decision-making problems with stochastic (random) outcomes over time.

- State space search is used to solve problems by searching through a space of possible states and actions.
- The goal is to find a policy (a rule for choosing actions) that maximizes the expected cumulative reward over time.

Rinforcement Learning

- Reinforcement learning (RL) is a subfield of machine learning that deals with how an agent can learn to make decisions based on feedback from the environment in the form of rewards or punishments.
- RL algorithms aim to maximize the expected cumulative reward over time by learning a policy that maps states to actions.
- In this sense, MENACE can be seen as a primitive form of RL algorithm, since it updates its behavior based on the reward signal obtained through winning or losing a game.



• Matchbox Educable Naughts and Crosses Engine.

MENACE (the Machine Educable Noughts And Crosses Engine) is a machine learning computer built from 304 matchboxes.

MENACE is a learning algorithm for playing the game of Tic Tac Toe. It uses a form of reinforcement learning to adjust its behavior based on the outcomes of previous games. MENACE can be seen as a primitive form of a reinforcement learning algorithm, since it updates its behavior based on the reward signal obtained through winning or losing a game.

Uninformed Search

Depth First Search(DFS)

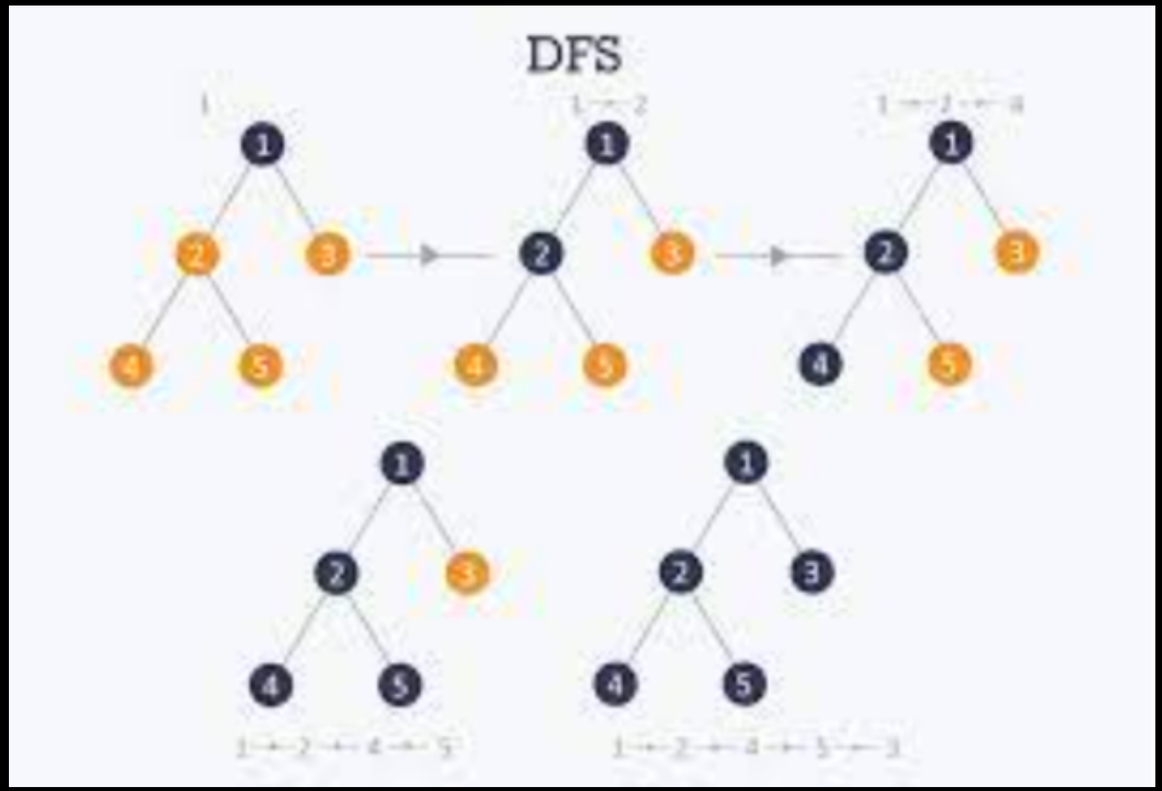
Data Structures

- **Stack :**

It uses last-in- first-out strategy and hence it is implemented using a stack.

- **Explored set :**

stores the collection of states we have already examined (and therefore don't need to look at again).

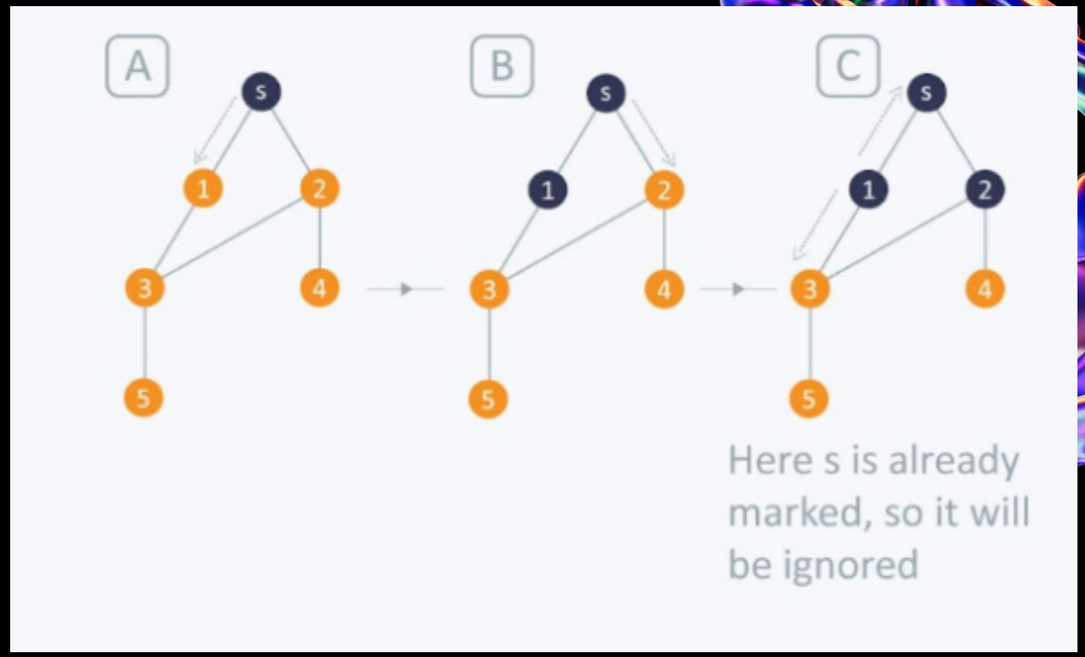


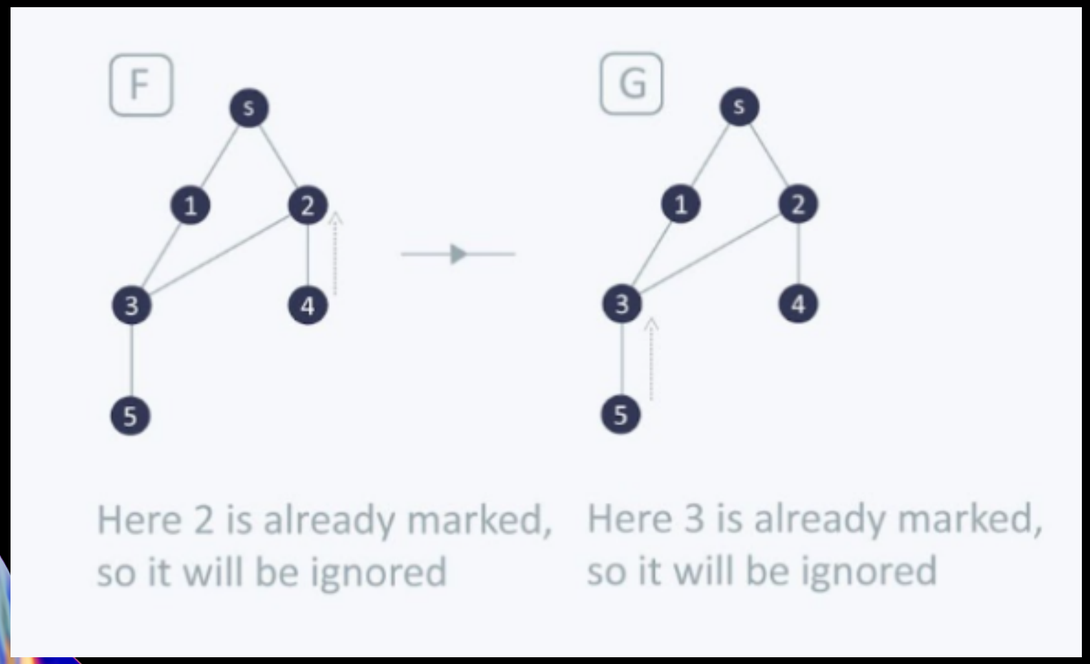
• Breadth first Search(BFS):

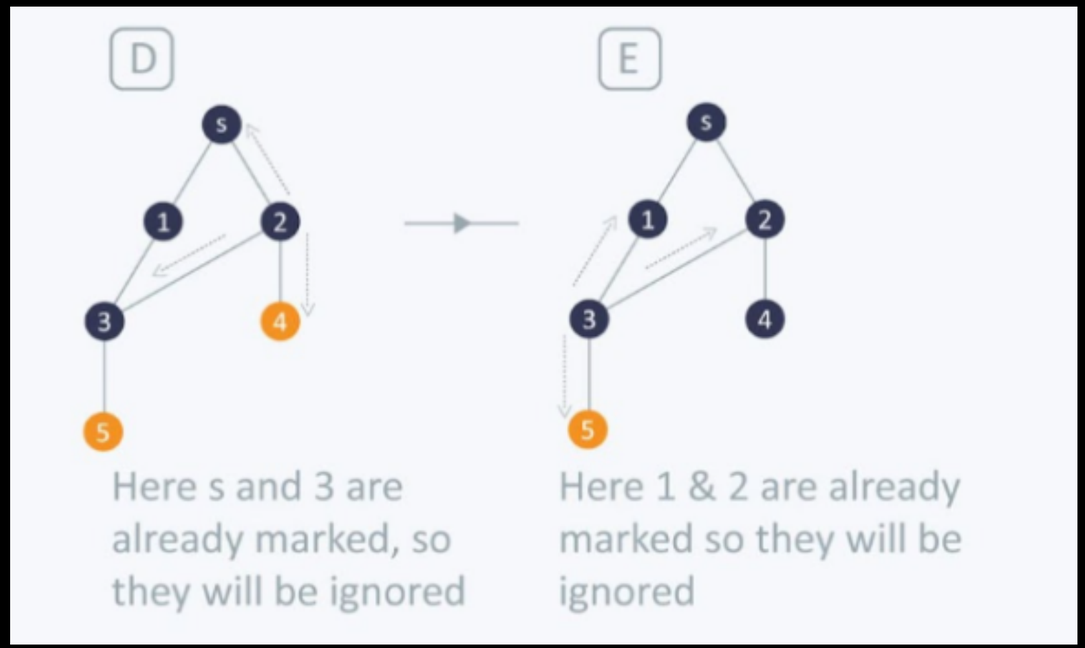
Data Structure:

- **Queue**: It uses a first-in- first-out strategy and hence it is implemented using a queue.

- **Explored set:** stores the collection of states we have already examined (and therefore don't need to look at again).

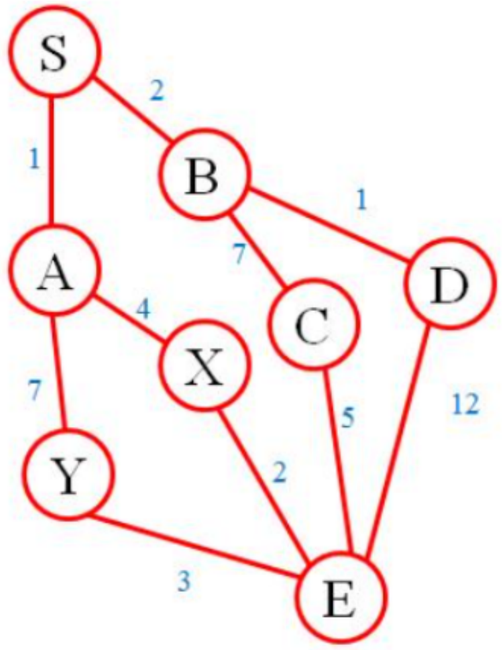






A* Search

It considers both the heuristic value and the actual cost from the start node to the current node.



- Values for h:

  A:5, B:6, C:4, D:15, X:5, Y:8

Expand S

{S,A} f=1+5=6

{S,B} f=2+6=8

Expand A

{S,B} f=2+6=8

{S,A,X} f=(1+4)+5=10

{S,A,Y} f=(1+7)+8=16

Expand B

{S,A,X} f=(1+4)+5=10

{S,B,C} f= (2+7)+4=13

{S,A,Y} f=(1+7)+8=16

{S,B,D} f= (2+1)+15=18

Expand X

{S,A,X,E} is the best path... (costing 7)

Data Structure in both Greedy and A*

- **Frontier:**

A data structure storing the collection of nodes that are available to be examined next in the algorithm.

- **Explored set:**

Stores the collection of states we have already examined (and therefore don’t need to look at again).

Markov Decision Process (MDP)

Markov Decision Process (MDP) is a mathematical framework for modeling decision-making in situations where outcomes are partially random and partially under the control of a decision-maker.

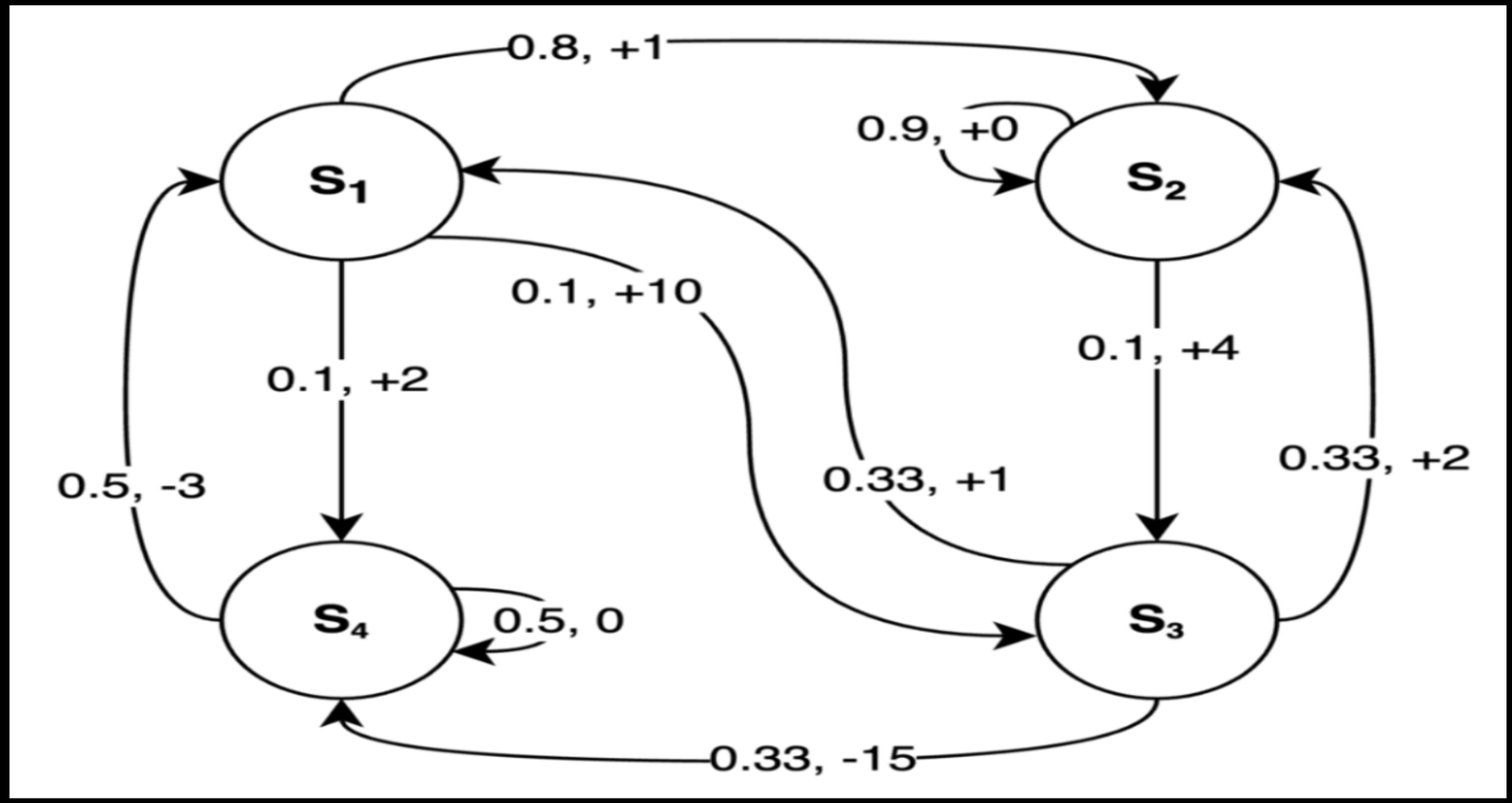
- State space (S): a set of all possible states that the agent can be in.
- Action space (A): a set of all possible actions that the agent can take in each state.
- Transition probability function (T): a function that specifies the probability of transitioning from one state to another state when the agent takes a particular action. It is denoted by $T(s, a, s') = P(s' \mid s, a)$,
- Reward function (R): a function that specifies the reward the agent receives when it takes a particular action in a particular state. It is denoted by **R(s, a).**
- Discount factor (γ): a scalar value that determines the importance of future rewards relative to immediate rewards.

Markov Decision Process (MDP)

- The transition probabilities are assumed to satisfy the Markov property, which means that the probability of moving from one state to another depends only on the current state and the action taken, and not on any past history.

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \ldots\ldots, S_t]$$

- In an MDP, the decision-maker tries to maximize a reward signal that is received for each action taken. The reward signal is a scalar value that reflects the desirability of the outcome associated with each action. The objective of the decision-maker is to find a policy that maximizes the expected cumulative reward over a finite or infinite time horizon.


0.8, +1
0.9, +0
S1
S2
0.1, +10
0.1, +2
0.1, +4
0.5, -3
0.33, +1
0.33, +2
S4
0.5, 0
S3
0.33, -15

- **Rewards** are the numerical values that the agent receives on performing some action at some state(s) in the environment. The numerical value can be positive or negative based on the actions of the agent. In Reinforcement learning, we care about maximizing the cumulative reward (all the rewards agent receives from the environment) instead of, the reward agent receives from the current state(also called immediate reward). This total sum of reward the agent receives from the environment is called returns.

$$G_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

- **Discount Factor (ɣ):** It determines how much importance is to be given to the immediate reward and future rewards

- **Policy (π):** In reinforcement learning, a policy in a Markov Decision Process (MDP) is a mapping from states to actions, which specifies the action to be taken in each state. Formally, a policy π is a function that maps each state s to an action a, such that a = π(s). A policy can be deterministic, where it maps each state to a single action, or stochastic, where it maps each state to a probability distribution over actions.

Value Function and Policy Function

- Value Function determines how good it is for the agent to be in a particular state.
- A policy is a simple function that defines a probability distribution over Actions (a∈ A) for each state (s ∈ S). If an agent at time t follows a policy π then π(a|s) is the probability that the agent with taking action (a) at a particular time step (t).In Reinforcement Learning the experience of the agent determines the change in policy. Mathematically, a policy is defined as follows :

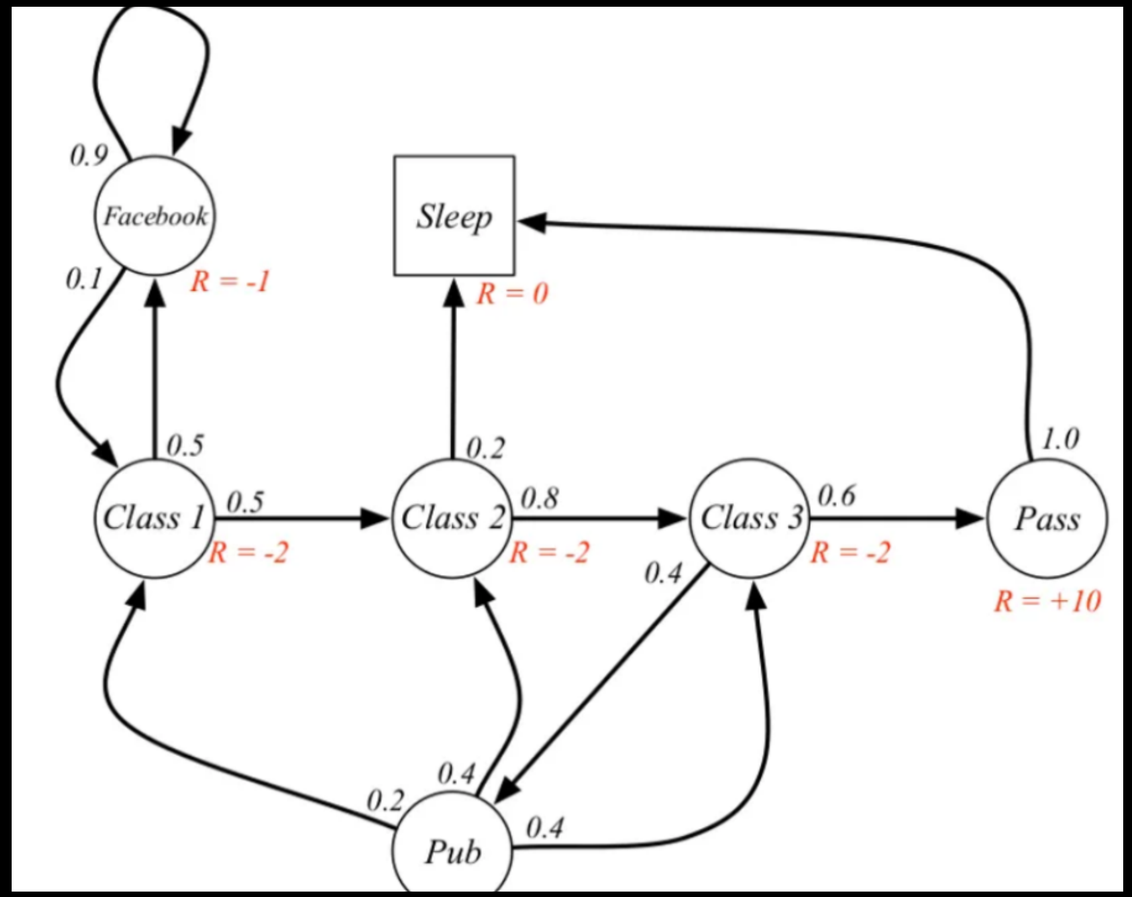
$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

- Now, how do we find the value of a state. The value of state s, when the agent is following a policy π which is denoted by vπ(s) is the expected return starting from s and following a policy π for the next states until we reach the terminal state. We can formulate this as :(This function is also called State-value Function)

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S},$$

- This equation gives us the expected returns starting from the state(s) and going to successor states thereafter, with the policy π.

Value Function and Policy Function



GT = -2 + -2 * 0.5 + 10*0.5*0.5
= -0.5

- Suppose our start state is Class 2, and we move to Class 3 then Pass then Sleep.In short, Class 2 > Class 3 > Pass > Sleep. Our expected return is with a discount factor of 0.5:

Bellman Equation for Value Function

- The Bellman equation is a powerful tool in reinforcement learning because it allows us to compute the value function for a given policy without explicitly modeling the transition probabilities and rewards of the environment. This makes it particularly useful in situations where the environment is complex and difficult to model.

- **Bellman expectation function**

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$$

The above equation tells us that the value of a particular state is determined by the immediate reward plus the value of successor states when we are following a certain policy(π).

Bellman Expectation Equation for State-Action Value Function (Q-Function)

$$q_\pi(s,a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right]$$

From the above equation, we can see that the State-Action Value of a state can be decomposed into the immediate reward we get on performing a certain action in state(s) and moving to another state(s') plus the discounted value of the state-action value of the state(s') with respect to the some action(a) our agent will take from that state on-wards.

- Overall, the Q-function is a critical concept in reinforcement learning, as it enables an agent to make optimal decisions by evaluating the expected cumulative reward of different actions in a given state. It Enables model-free learning.

Optimal State-Value Function

$$v_*(s) = \max_\pi v_\pi(s)$$

It is the maximum Value function over all policies.

Optimal State-Action-Value Function

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

It is the maximum action-value function over all policies.

Optimal Policy

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

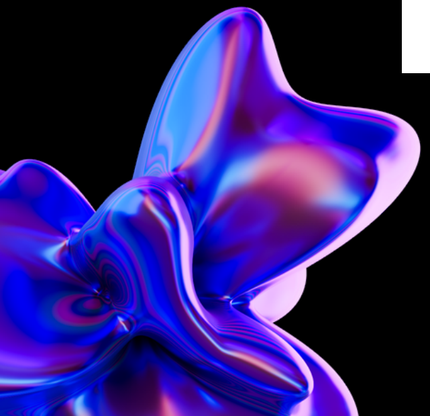
Finding an Optimal policy :

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_*(s,a) \\ 0 & \textit{otherwise} \end{cases}$$

State-action value function or Q-Function

- This function specifies how good it is for the agent to take action (a) in a state (s) with a policy π.

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t=s, A_t=a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t=s, A_t=a\right].$$



Policy iteration algorithm (Pseudocode)

Data: θ: a small number
Result: V: a value function s.t. $V \approx v_*$, π: a deterministic policy s.t. $\pi \approx \pi_*$
Function *PolicyIteration* **is**
    /* Initialization */
    Initialize $V(s)$ arbitrarily;
    Randomly initialize policy $\pi(s)$;
    /* Policy Evaluation */
    $\Delta \leftarrow 0$;
    while $\Delta < \theta$ **do**
        for *each* $s \in S$ **do**
            $v \leftarrow V(s)$;
            $V(s) \leftarrow \sum_{s',r'} p(s', r|s, \pi(s))[r + \gamma V(s')]$;
            $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
        end
    end
    /* Policy Improvement */
    policy-stable $\leftarrow$ *true*;
    for *each* $s \in S$ **do**
        old-action $\leftarrow \pi(s)$;
        $\pi(s) \leftarrow arg\max_a \sum_{s',r'} p(s', r|s, a)[r + \gamma V(s')]$;
        if *old-action* $!= \pi(s)$ **then**
            policy-stable $\leftarrow$ *false*;
        end
    end
    if *policy* $-$ *stable* **then**
        return $V \approx v_*$ and $\pi \approx \pi_*$;
    else
        go to Policy Evaluation;
    end
end

Value iteration algorithm (Pseudocode)

Algorithm 2: Value Iteration Algorithm

Data: θ: a small number
Result: π: a deterministic policy s.t. $\pi \approx \pi_*$
Function *ValueIteration* **is**
    /* Initialization */
    Initialize $V(s)$ arbitrarily, except $V(terminal)$;
    $V(terminal) \leftarrow 0$;
    /* Loop until convergence */
    $\Delta \leftarrow 0$;
    while $\Delta < \theta$ **do**
        for *each* $s \in S$ **do**
            $v \leftarrow V(s)$;
            $V(s) \leftarrow \max_a \sum_{s',r'} p(s', r|s, a)[r + \gamma V(s')]$;
            $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
        end
    end
    /* Return optimal policy */
    return π s.t. $\pi(s) = arg\max_a \sum_{s',r'} p(s', r|s, a)[r + \gamma V(s')]$;
end

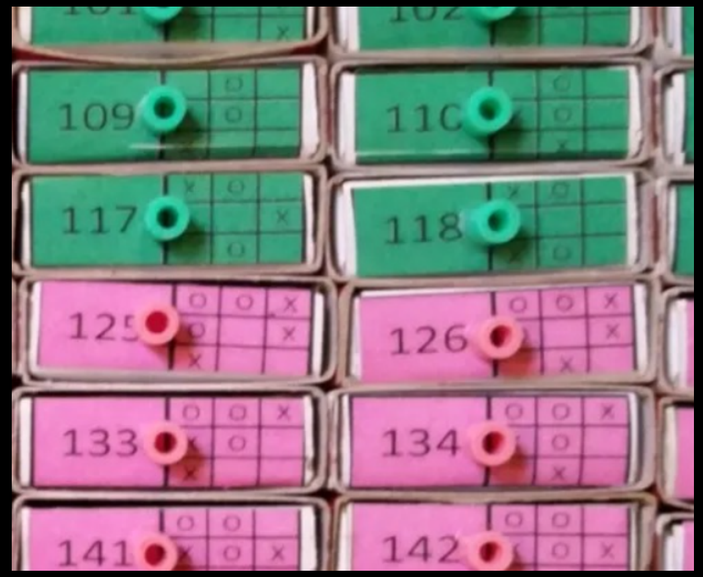
USE OF RANDOM NUMBERS REQUIRED FOR MDP & RL

1. MDPs typically involve probabilistic transitions between states, which means that the agent may not always end up in the same state even if it takes the same action multiple times. To model these probabilistic transitions, random numbers are often used to sample from a probability distribution that determines the next state and reward.
2. In RL algorithms, the agent's behavior is often guided by an exploration strategy that encourages it to try different actions and explore new parts of the state space. One common exploration strategy is to select actions randomly with some probability, which helps the agent discover new states and learn a good policy.
3. Some RL algorithms, such as Q-learning and SARSA, use a "greedy" policy that always selects the action with the highest expected reward. However, this can lead to the agent getting stuck in a local optimum and failing to explore other parts of the state space.

Matchbox Educable Naughts And Crosses Engine

Origin :

MENACE(Machine Educable Noughts And Crosses Engine) is a machine-learning computer built from 304 matchboxes. MENACE was first built by Donald Michie in 1960. It was a mechanical device designed to compete against a human opponent in the game of tic-tac-toe (also known as naughts and crosses).

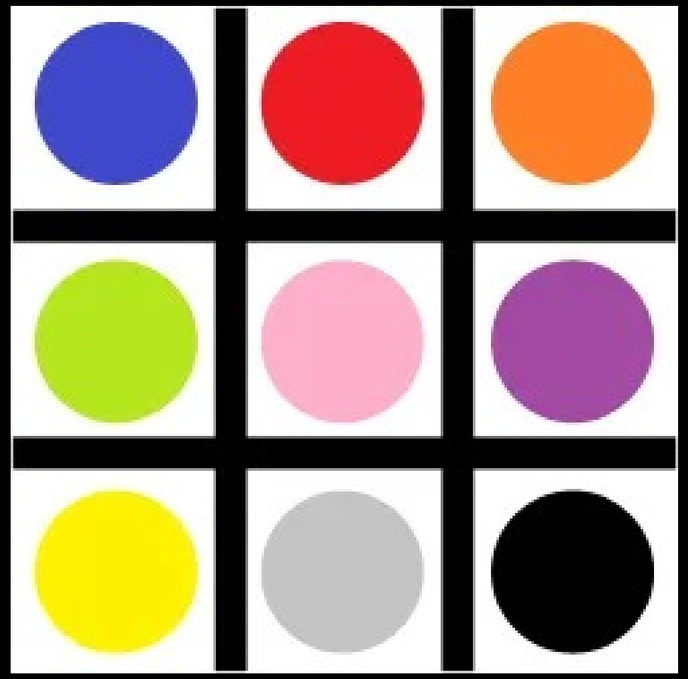


The toy was a small plastic box with a nine-square grid on top. A series of gears and switches inside the box allowed the toy to "think" and decide where to place its next move. The user would take their turn by pressing down on one of the box's nine squares, and the toy would respond with its own move shortly afterwards.

• Algorithm

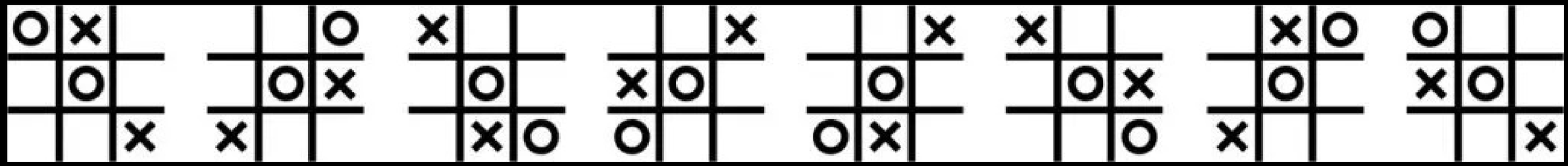
1. MENACE is made up of a series of matchboxes, each of which represents a different game board configuration. Each matchbox contains a variety of coloured beads, each of which represents a different move that MENACE could make.
2. When MENACE plays Tic-Tac-Toe, it selects a matchbox that represents the current game board configuration.
3. MENACE then draws a bead at random from the matchbox to represent its move for that turn.
4. After the game, MENACE is given a reward or a penalty based on the outcome. If MENACE wins, it is rewarded (in the form of additional beads added to the matchbox). It is penalised if it loses or draws (in the form of beads removed from the matchbox).
5. Based on the feedback it receives from of the rewards and penalties, MENACE learns which moves are more likely to result in a win over time. It adjusts the number of beads in each matchbox accordingly, increasing the chances of selecting the best moves in subsequent games.
6. MENACE gradually improves its game-playing ability and becomes increasingly difficult to defeat through this trial-and-error process.

Important Color Code



Symmetric Layouts

- The tic-tac-toe game board is symmetric, so many of the states are actually similar states if rotated or taken mirror image of the board. So we remove all the duplicate states that we generated. After this we are left with only around 900 states out of 255,168 states.



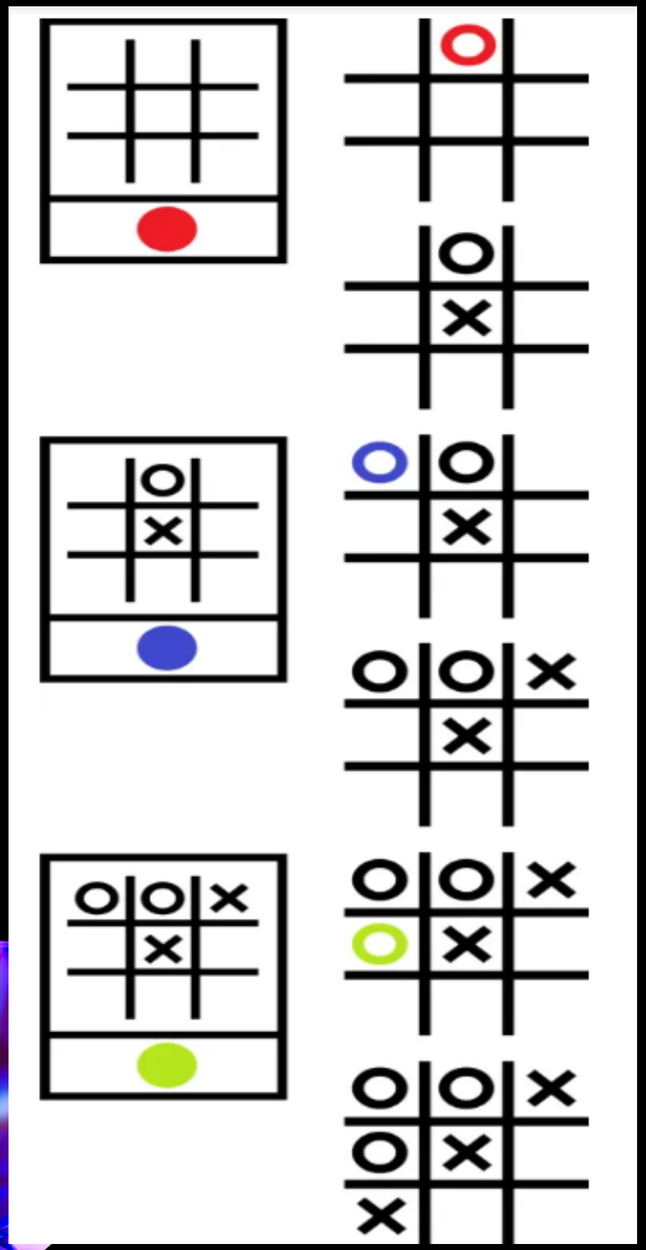
• How to play MENACE:

When playing against Menace, Menace always starts, otherwise the number of matchboxes needed would be greatly increased.

- 1st move, Menace’s turn: The operator finds the matchbox that displays the empty board, opens it and takes out a random bead. In this case the random bead is red, which means that Menace wants to place its counter in the centre-top space.
- 2nd move, player’s turn: The human player places the counter in the desired spot, which is in the centre.
- 3rd move, Menace’s turn: The operator finds the matchbox that displays the current board layout, opens it and takes out a random bead. The random bead is now blue, which means that Menace wants to place its counter in the top-left corner.

• How to play MENACE:

- 4th move, player’s turn: The human player places the counter in the desired spot, blocking Menace from getting three in a row.
- 5th move, Menace’s turn: The operator finds the matchbox that displays the current board layout, opens it, and takes out a random bead. The random bead is green, which means that Menace wants to place its counter in the middle row and leftmost column.
- 6th move, player’s turn: The human player places the counter in the bottom-left corner and obtains three in a row. The human player wins!

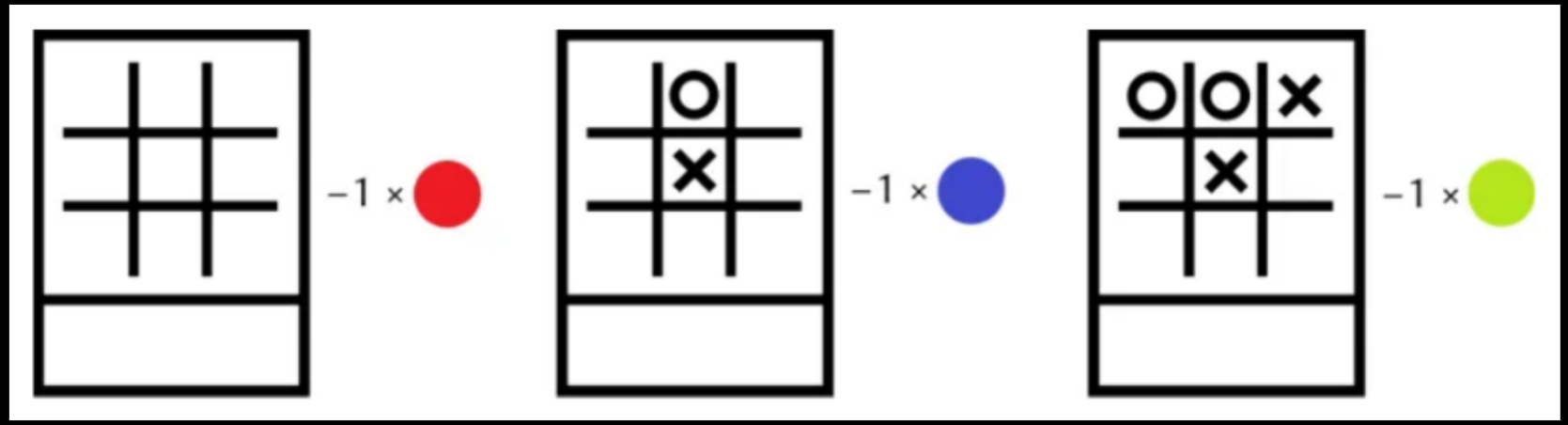


- The human player then plays bottom right. Again MENACE's operator finds the box for the current position, it gives an orange bead and MENACE plays in the left middle. Finally, the human player wins by playing top right.
- MENACE has been beaten, but all is not lost. MENACE can now learn from its mistakes to stop them from happening again.

- Initially, MENACE begins with four beads of each color in the first move box, three in the third move boxes, two in the fifth move boxes, and one in the final move boxes. Removing one bead from each box on losing means that later moves are more heavily discouraged. This helps MENACE learn more quickly, as the later moves are more likely to have led to the loss.
- After a few games have been played, it is possible that some boxes may end up empty. If one of these boxes is to be used, then MENACE resigns. When playing against skilled players, it is possible that the first move box runs out of beads. In this case, MENACE should be reset with more beads in the earlier boxes to give it more time to learn before it starts resigning.

At the end of each game:

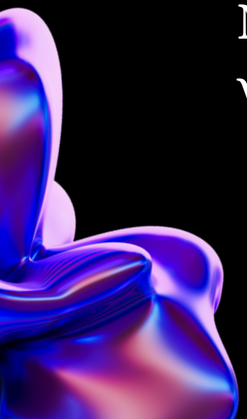
- If MENACE loses, then the beads used in that game are removed.
- If MENACE won, then three extra copies of each bead used in that game are added.
- If the game was a draw, then one extra copy of each bead used in that game is added.

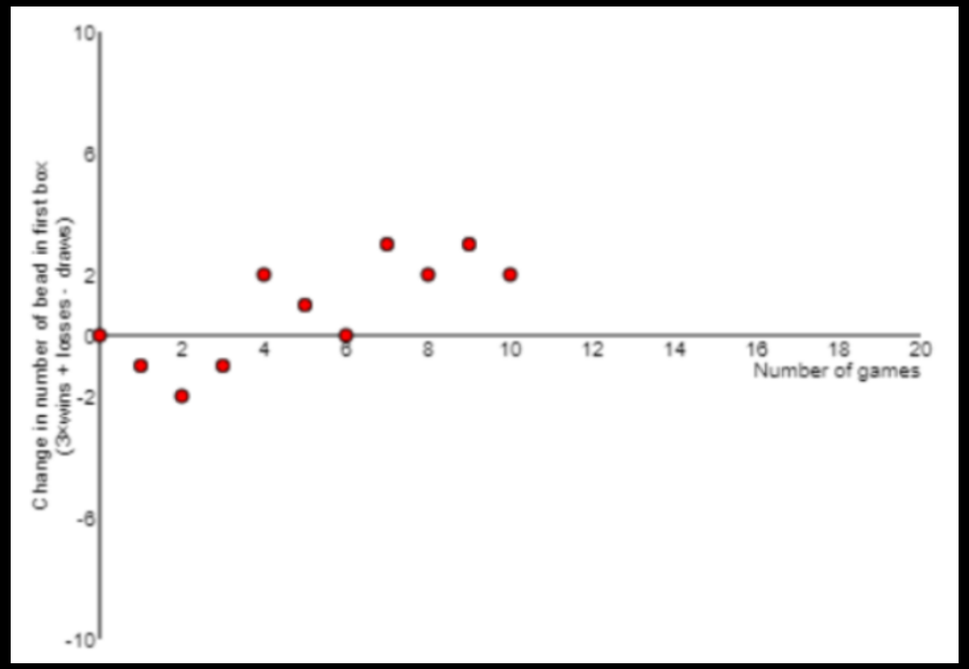


Learning:

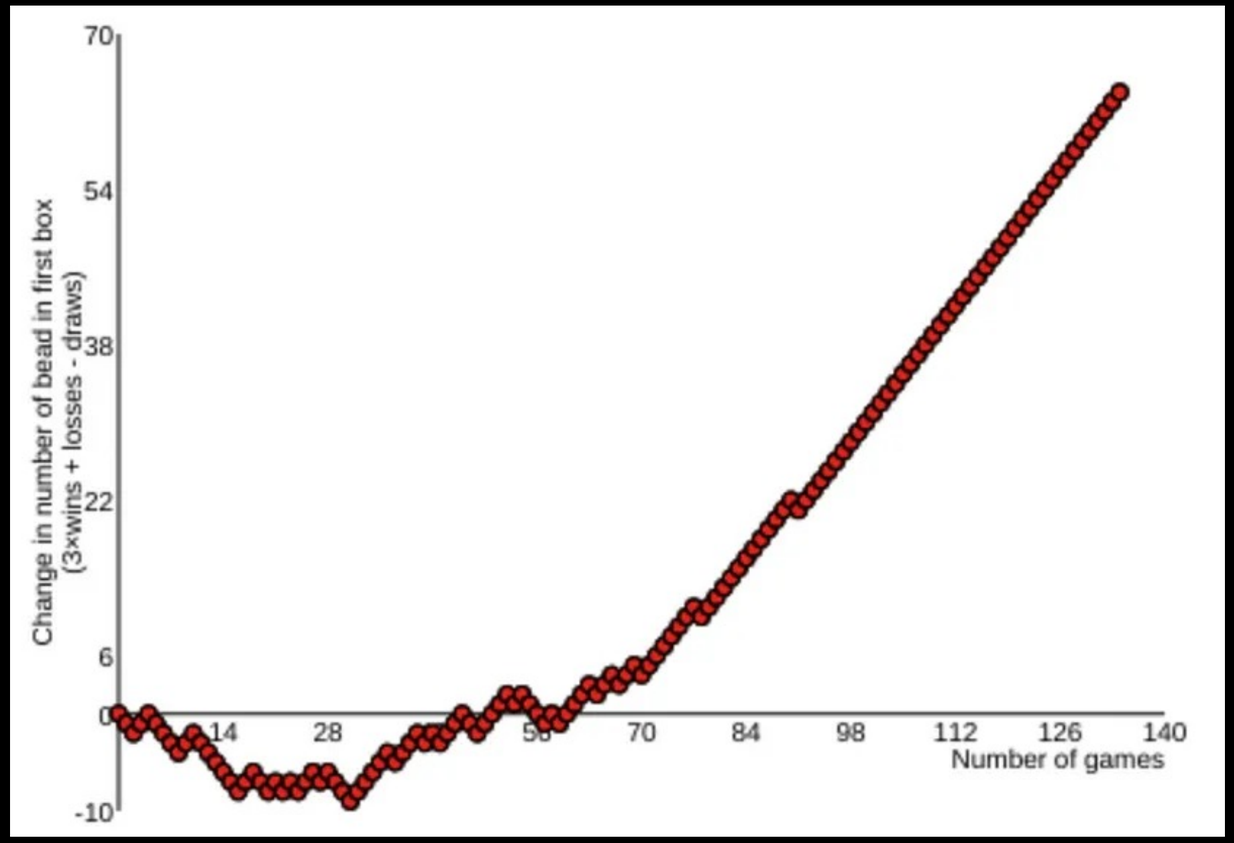
- When training begins, all boxes contain colour-coded beads, where each colour represents a move (or position) on a board.

- This graph shows the change in the number of beads in the first move box: this decreases when MENACE loses and increases when MENACE wins or draws

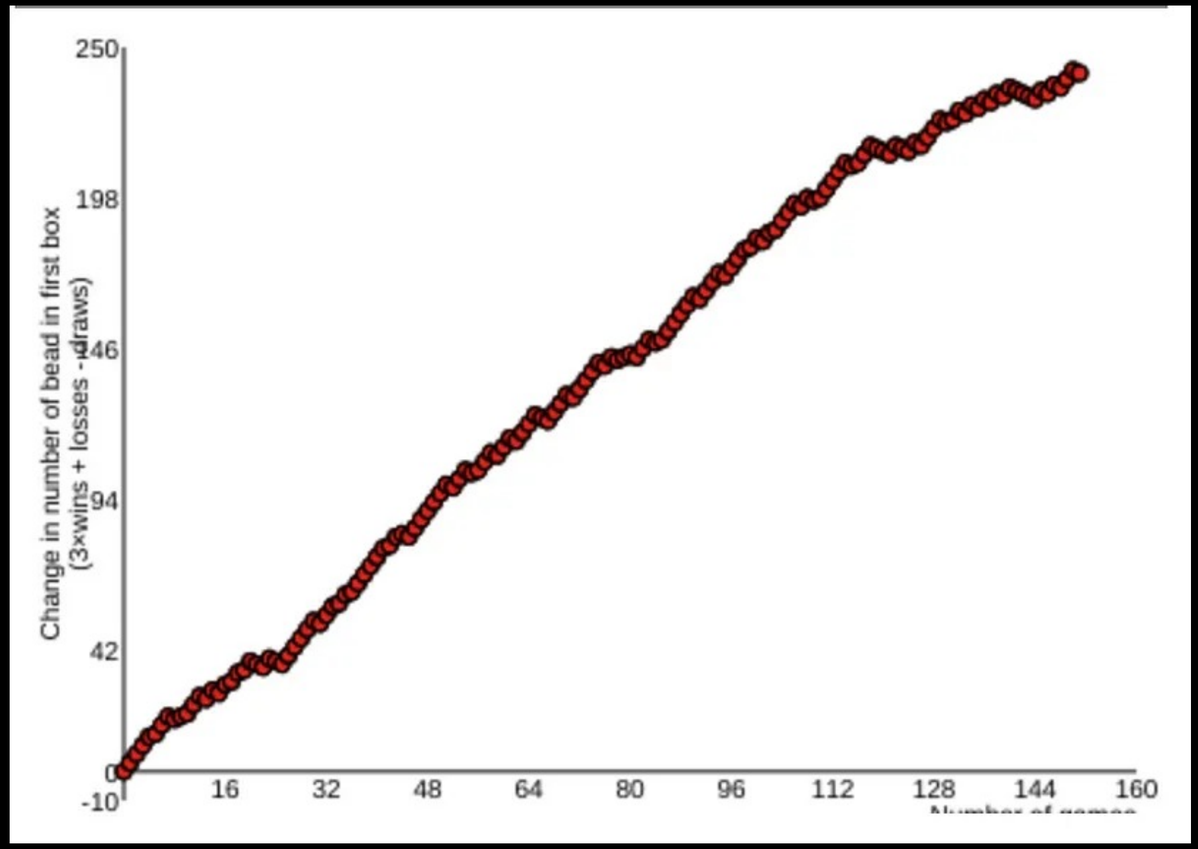




- **MENACE score when played against a perfect-playing opponent:**



- **MENACE score when played against a random opponent:**



References

- https://stanford-cs221.github.io/autumn2019/#schedule
- https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da
- https://www.baeldung.com/cs/ml-value-iteration-vs-policy-iteration
- https://www.geeksforgeeks.org/search-algorithms-in-ai/
- https://theuniverseofdatascience.quora.com/

• Github Link

kamalkumar7/
RedmondPie_AI_Lab_Co...

Contains code of AI lab submission

4 Contributors | 0 Issues | 0 Stars | 1 Fork

RedmondPie_AI_Lab_Codes/Week Assignment 10 - Lab 8 at main · kamalkumar7/RedmondPie_AI_Lab_Codes

Contains code of AI lab submission. Contribute to kamalkumar7/RedmondPie_AI_Lab_Codes development by creating an account on GitHub.

GitHub

THANK YOU