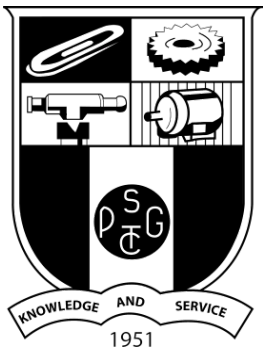
PLAYING THE GAME OF TWENTY-ONE AND PONTOON

DHIKSHITHA A 19PD09
JOSHIKA S 19PD16

REINFORCEMENT LEARNING

FIVE YEAR INTEGRATED
M.Sc. DATA SCIENCE



APRIL 2023

DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCES

PSG COLLEGE OF TECHNOLOGY
(Autonomous Institution)
COIMBATORE - 641004.

1. OBJECTIVE :

The objective of this package is to optimize playing the game of twenty-one and pontoon.

2. PROBLEM STATEMENT :

Equally well known as Twenty-One. The rules are simple, the play is thrilling, and there is opportunity for high strategy. In fact, for the expert player who mathematically plays a perfect game and is able to count cards, the odds are sometimes in that player's favor to win.

Obtain cards whose numerical value is as great as possible without exceeding 21. What is the optimal winning strategy (policy)?
i.e., When do you hit? When do you stick?

- Dealer vs players
- The player closest to the goal wins
- If a player exceeds 21 ,they lose
- Both player receive 2 cards in the beginning of the game - faceup
- One card face down - dealer
- Face cards (Kings, Queens, and Jacks) - 10 points.
- Aces - 1 or 11 points - 11(usable ace)
- Other cards - numeric value shown taken as points

3. METHODS :

- In Reinforcement learning, there are 2 kinds of approaches, model-based learning and model-free learning. Model-Based Learning can be applied if we have full information of the transition probabilities and rewards, but it would be too computationally expensive if the game gets too complex.
- Model-Free Learning is the more practical approach as it doesn't need to have information on the full transition probabilities and rewards as it focuses on figuring out value function directly from the interactions with the environment.

- We would attempt to train an agent to play blackjack using model-free learning approach
- Approaches:
  i. Basic Naive approach
  ii. Monte Carlo
  iii. Temporal Difference

4. ALGORITHM :

- **Monte Control - on policy**

Algorithm 9: First-Visit MC Prediction (*for action values*)

Input: policy π, positive integer *num_episodes*
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize $N(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Initialize $returns\_sum(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for $i \leftarrow 1$ *to* *num_episodes* **do**
  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
  for $t \leftarrow 0$ *to* $T-1$ **do**
    if (S_t, A_t) *is a first visit (with return* G_t*)* **then**
      $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
      $returns\_sum(S_t, A_t) \leftarrow returns\_sum(S_t, A_t) + G_t$
  end
end
$Q(s,a) \leftarrow returns\_sum(s,a)/N(s,a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
return Q

- **Monte Control - Off policy**

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$
  $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit For loop
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

- **Temporal Difference - SARSA**

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

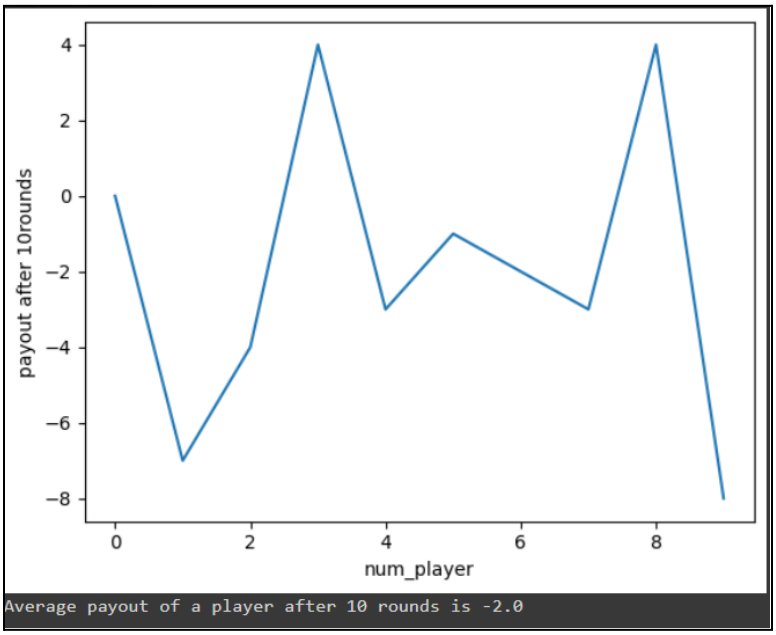
Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

- **Temporal Difference - Q -Learning**

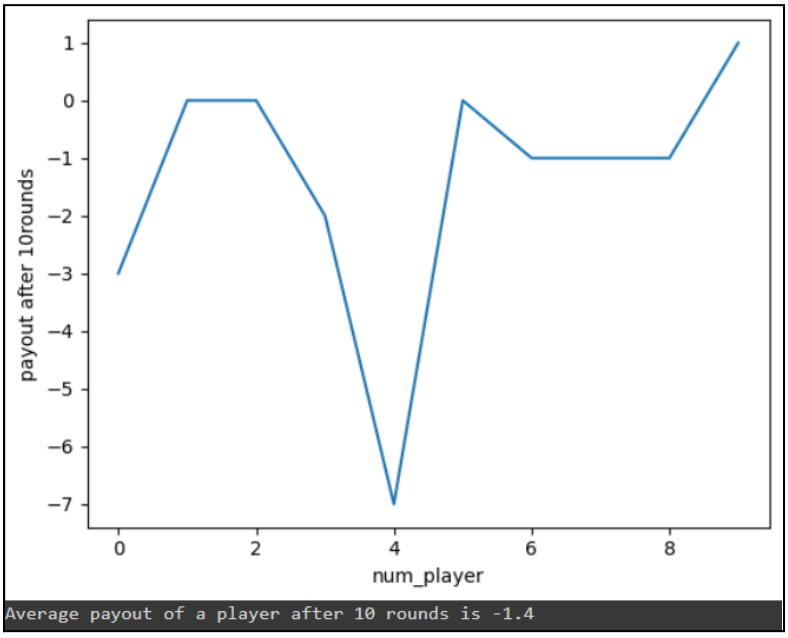
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big]$
    $s \leftarrow s'$;
  until s is terminal

5. RESULTS :

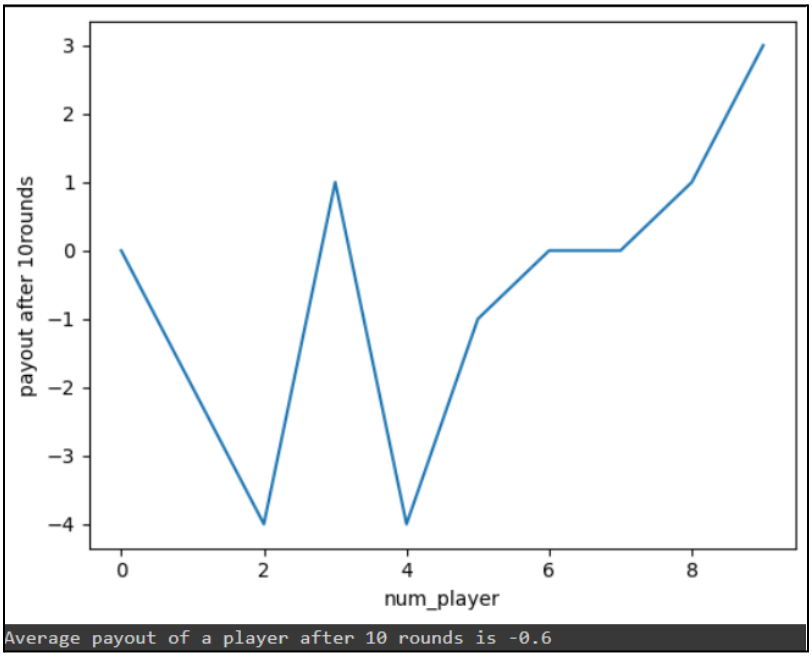
- **Naive Strategy**



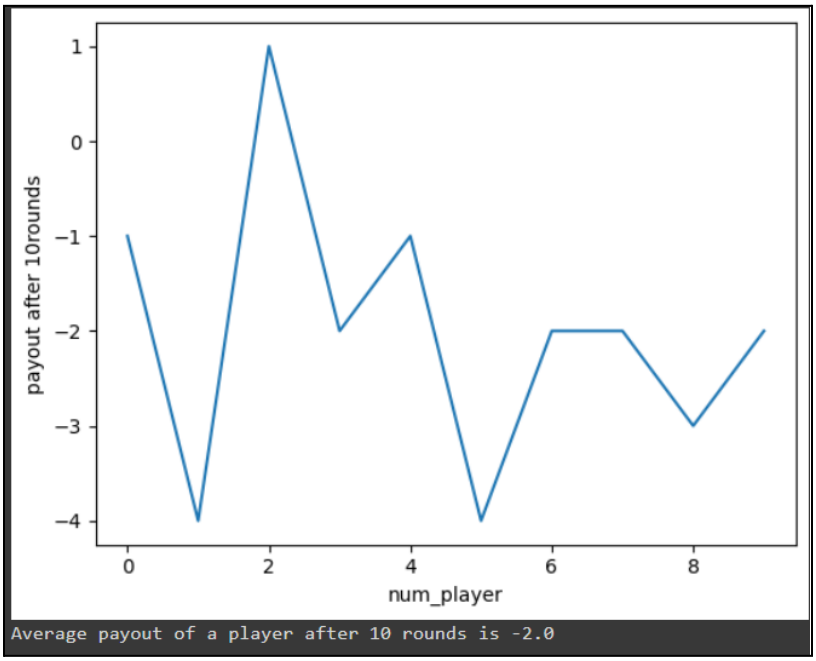
- **Monte Control - on policy**



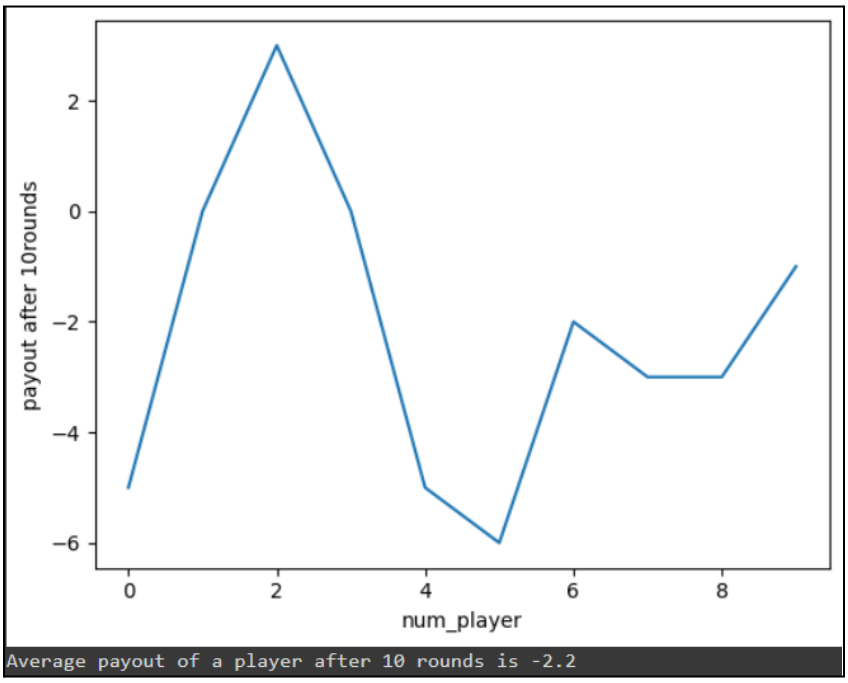
- **Monte Control - Off policy**



- **Temporal Difference - SARSA**



- **Temporal Difference - Q -Learning**



6. IMPROVISATION :

- We've improvised using TD Learning - SARSA And Q Learning.

7. TOOLS USED :

- Python
- OpenAi - Gym environment

8. SUMMARY :

We can see the different policies and payoffs for the basic naive strategy, the Monte-Carlo On-Policy trained policy, Monte-Carlo Off-Policy trained policy, TD SARSA (On-Policy) trained policy and TD Q-Learning (Off-Policy) trained policy.
The "best" policy we trained would be Monte-Carlo's policies, with an average amount improving for each player.