Playing the game of Twenty-One and Pontoon

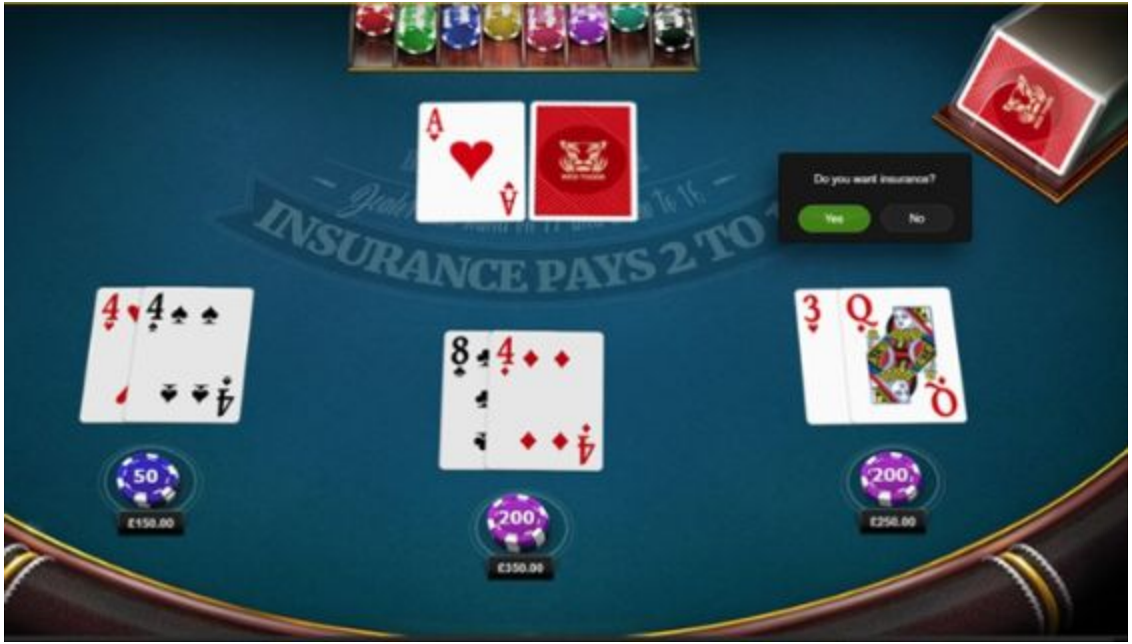
Dhikshitha A - 19PD09

Joshika S - 19PD16

What is Twenty-One and Pontoon ?

- 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.
- Today, Twenty - One is the one card game that can be found in every American casino.

How to Play it ?



Goal of the problem : 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.
- Other cards - numeric value shown aken as points

Using Reinforcement Learning :

- The problem can be solved as a RL problem
  - States - Players cards sum, the dealer’s showing card, usable ace
  - Reward - +1, -1, 0 - given at the end of episode
  - Actions -   Stick (end your turn), Hit (ask for more cards)
- The different approaches to solve the Twenty one and pontoon problem includes
  - Monte Carlo
  - Temporal Difference

Monte-Carlo Approach :

- Monte- Carlo is based on averaging sample returns
- Each game of twenty one is an episode
- Agent/player plays thousands of games using the policy defined
- Value of current state - Each time the agent carries out an action A in state S for the first time in the game and it will calculate the reward from that point onwards - > First Visit MC

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

Algorithm 11: First-Visit Constant-α (GLIE) MC Control

Input: positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: policy π ($\approx \pi_*$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
    $\epsilon \leftarrow \epsilon_i$
    $\pi \leftarrow \epsilon$-greedy(Q)
    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**
            $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$
    end
end
return π

Temporal Difference Approach

- Temporal difference (TD) learning is a model-free reinforcement learning algorithm that aims to learn the value function of a policy, without needing a model of the environment.
- The value function is an estimate of the expected cumulative reward that an agent can obtain by following a given policy in a given state.

- The agent updates its estimates of the value function after each time step, based on the observed reward and the next state.
- The updates are based on the difference between the predicted value of the current state and the predicted value of the next state.
- This difference is known as the TD error, and it is used to update the value estimate of the current state.

Q-LEARNING

Q-learning is a model-free reinforcement learning algorithm that aims to learn the optimal action-value function for the agent. In the case of twenty one, the action-value function Q(s, a) maps each state-action pair (s, a) to the expected cumulative reward of taking action a in state s.

Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode):

  Choose a from s using policy derived from Q

  Take action a, observe r, and s'

  $Q(s, a) \Leftarrow Q(s, a) + \alpha\, [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

  $s \Leftarrow s'$

until s is terminal

Figure 1: The Q-learning algorithm

Thank you!!

Thank you!!

In the case of blackjack, we can define the state space as the player's hand value, the dealer's face-up card, and whether or not the player has an ace. The action space is simply to hit or stand. The reward function can be defined as follows:

- If the player wins, the reward is +1.
- If the player loses, the reward is -1.
- If the game ends in a tie, the reward is 0.

1. Define the state space: In Blackjack, the state space consists of the player's current hand, the dealer's up-card, and whether or not the player has a usable ace. For example, a state could be represented as (12, 5, False), meaning the player has a hand value of 12, the dealer's up-card is a 5, and the player does not have a usable ace.
2. Define the action space: In Blackjack, the action space consists of the player's choices: hit or stand. The player can hit (i.e., request another card) or stand (i.e., end their turn).
3. Initialize the Q-table: The Q-table is a table that contains the Q-values for each state-action pair. The Q-value represents the expected reward that the agent will receive by taking a particular action in a particular state. The Q-table is initialized to some arbitrary values.

4. Implement the Q-learning algorithm: The Q-learning algorithm updates the Q-table by iterating over episodes of the game. In each episode, the agent starts with an initial state, and takes actions according to an exploration strategy (e.g., epsilon-greedy). After each action, the agent observes the reward and the new state, and updates the Q-value for the previous state-action pair based on the Bellman equation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \mathrm{alpha} * (r_{t+1} + \mathrm{gamma} * \max(Q(s_{t+1}, a)) - Q(s_t, a_t))$$

where alpha is the learning rate, gamma is the discount factor, r_{t+1} is the reward received after taking action a_t in state s_t, and $\max(Q(s_{t+1}, a))$ is the maximum Q-value over all possible actions in the next state s_{t+1}.

5. Repeat step 4 for a large number of episodes (e.g., thousands or millions) until the Q-table converges to the optimal values.

6. Once the Q-table is learned, the agent can use it to play Blackjack by selecting the action with the highest Q-value for a given state.