Title:
Poker Playing Agent Project: 1st Assignment
Description:
As mentioned in class, this assignment is basically "part 1" of your final project assignment. The idea is to implement a simple(r) version of the problem that can be solved with "exact" methods (i.e., no approximations based on neural networks or other methods) that we learned in the first lectures abour MDP and Q-learning methods. I provide below my suggestions for implementing this first version of your project:

Environment (Simplified Poker Game):

- Assume 4-5 cards are used only: e,g, 10, J,Q,K,1 (adjust according to the solution time needed on colab or your PC) - In the final version this will be the entire deck.
- Each player is dealt, only one card in hand which is private (in Texas Hold'em this will be two cards per player)
- Only two cards will be set on the table (in Hold'em there'll be 5).
- The betting rounds and how the table cards are uncovered will be as follows:
    - Round 1: Each player sees their own card, but the 2 table cards are still unknown; Player 1 can check or bet 1 token.
        - In case of check: Player 2 can also check, or raise one token. In the latter case, Player 1 can fold or bet 1 token too.
        - In case of betting: Player 2 can fold, bet 1 token also, or raise 1 token (i..e bet a total of 2). In the latter case, player 1 can fold, or bet that extra token (this is not a "re-raise", player 1 just "meets" the raise of player 2)
    - Round 2: Both table cards are Tηrevealed. Another betting round follows with the same rules as Round 1.
- The winning rules are the following:
    - best combination is 3 of a kind
    - then 1 pair: highest pair wins (with order as usual: 1,K,Q,J,10). In case of same pair (but different suit), highest 3rd card wins.
    - If no pair by any player, highest card wins.
    - ignore "flush" (3 of same suit) for this simpler setup.
- Ante: Assume that before even being dealt the initial card, each player must always bet 0.5 tokens. (You should think what would happen if there is no such "ante" and what the optimal policy is that your agent will - hopefully - learn).
- Note: for implementing this environment (as well as the full fledged Hold'em environment for the final submission), you might find this python library useful: https://rlcard.org/

Environment (Opponent):

You will need to assume an opponent, as part of your environment. For this 1st part, you don't have to assume an "adversarial" opponent who also tries to play the "best respone" to your action. Instead, you can assume a "static" opponent for now. NOTE:

"static" does not mean that your opponent is deterministic. I propose to implement the following type of opponents:

- **Random** Opponent: You opponent will pick randomly among the available actions at each phase of the game, regardless of the card she owns, and the cards on the deck.
- **Threshold** Opponent: Your opponent will bet the maximum amount allowed in each round, provided that she has at least a high enough "combination":
    - e.g., for Round 1 you could assume your opponent will always bet/raise with a K or 1.
    - and for Round 2 you could assyme your opponent will always bet/raise with any pair.
    - you can adjust the above to also model "loose/tight" opponents.

Control variables and objective:

- Controls: the actions (fold,check,bet,raise) that your agent can take at each round.
- Objective: Your algorithm must learn to maximize the expected earned tokens per game.

Algorithms to implement:

- Policy or Value Iteration assuming the environment (game rules) and the specific opponent type is known.
- Q-learning, assuming the game rules are again known, but the opponent type is not known (you will still assume of course one of the above opponent types in the environment....possibly even a probabilistic mix of the two)

Experiments to try and report:

1. Demonstrate that Policy Iteration indeed solves the problem optimally .
2. Show the average reward achieved for different opponent types.
3. Prove that your Q-learning algorithm correctly learns the optimal policy as well (when the opponent is initially unknown). Demonstrate the convergence speed to the optimal.
4. Implement a simple environment (can be GUI or command line based) to be able to play against your agent.
5. (Not mandatory) experiment a bit by increasing the deck of cards used (e.g., add 9, then 8, etc.) until convergence becomes too slow.