Card Playing meets Rock-Paper-Scissors

Roll: 20CS30062

Objective:

The goal of this assignment was to design a reinforcement learning (RL) agent that plays a card game based on the mechanics of rock-paper-scissors (RPS). The task involved implementing both Value Iteration (VI) and Temporal Difference (TD(0)) learning algorithms to train the agent.

Problem Definition:

The agent's task is to play a series of rounds against an opponent. Each round, both the agent and the opponent select a card from their decks, and the winner is determined by the one who shows a higher card value. Rock-paper-scissors is played before this, the loser of which has to show its selection of cards. The game continues until all cards have been played. In the report thereon, N (or n) will represent the total cards..

State Representation:

- Agent Deck: Cards remaining in the agent's deck.
- Opponent Card Shown: The card currently shown by the opponent.

Transitions:

- The agent selects a card from its deck.
- The opponent shows its card.
- The game transitions to the next state where the opponent's next card is unknown.

Reward Structure:

- +1 for winning a round.
- -1 for losing a round.

Training Algorithms:

1. Value Iteration (VI):

The VI algorithm estimates the optimal value function for each state by iterating over all possible state transitions and updating values based on the Bellman equation.

Pseudocode for VI:

—---

```
while True:
    for state in all_states:
        agent_deck, opponent_deck = state
        state_tuple = (
            tuple(agent_deck),
            opponent_deck[0] if len(opponent_deck) > 0 else -1
        )

        if len(agent_deck) == 0:
            V[state_tuple] = 0  # If no cards left, the value is 0
            continue

        v = V[state_tuple]
        best_value = float('-inf')

        for action in range(len(agent_deck)):
            # Initialize the environment state
            env.state = deepcopy({
                'agent_deck': agent_deck,
                'opponent_card_shown': opponent_deck[0]
            })
            env.opponent_deck = deepcopy(opponent_deck)

            # Take a step and get the next state, reward, and done flag
            new_state, reward, done = env.step(action)
            new_state_tuple = (
                tuple(new_state['agent_deck']),
                new_state['opponent_card_shown']
            )

            action_value = reward + gamma * V.get(new_state_tuple, 0)

            if action_value > best_value:
                best_value = action_value
                policy[state_tuple] = action

        V[state_tuple] = best_value
```

```
          delta = max(delta, abs(v - V[state_tuple]))

     if delta < theta:
         break
```

—--

2. TD(0):

TD(0) is a model-free RL algorithm that updates the value function based on actual transitions experienced during gameplay.

Pseudocode for TD(0):

—--

```
while not done:
    state = (tuple(agent_deck), opponent_card_shown)
    action = select_card(agent_deck)  # Random action
    new_state, reward, done = env.step(action)

    V[state] += alpha * (reward + gamma * V[new_state] - V[state])

    state = new_state
```

—--

The training involved generating random episodes where the agent's and opponent's decks were initialized randomly, and gameplay transitions continued until the game ended. Each transition updated the value function based on rewards received.

Training Setup:

Optimizations:

- State Reduction: The problem was reduced to a Partially Observable Markov Decision Process (POMDP) by removing the opponent's entire deck from the state and only considering the current opponent card shown. Given the opponent’s fixed strategy, this transformation simplified the state space.

- Blowup Control: Initially, the state space was large due to permutations of decks, but the reduction minimized unnecessary complexity. Still n=10 generates 133651 states.

1. Value Iteration (VI):

For VI, we generated all possible configurations of the agent's and the opponent's decks. Each configuration was treated as a unique state. To avoid revisiting states multiple times in a single iteration, we used a visitation array (vis).

2. TD(0):

For TD(0), we generated 400,000 random episodes for training. The algorithm was tested to ensure convergence on randomly selected episodes.

Hyperparameters:

- Gamma (VI):	0.8 (low discounting since all future rewards matter).
- Theta (VI):	0.1 (not essential; convergence happens quickly for small n).
- Gamma (TD(0)):	0.6 (higher discounting due to reward structure).
- Alpha (TD(0)):	0.1 (learning rate).
- Episodes (TD(0)):	400,000 episodes for robust training.

Results and Comparisons:

The performance of the agent was evaluated using both the Value Iteration policy and the TD(0) learned value function. Below are some sample runs and comparisons:

Simulation Output:

```
Simulating the game using the Value Iteration policy:
Agent Deck: [1, 3, 7, 8, 9], Opponent Deck: [10, 6, 2, 4, 5], Opponent
Card Shown: 10
Opponent wins round!
Agent Deck: [3, 7, 8, 9], Opponent Deck: [6, 2, 4, 5], Opponent Card
Shown: 6
Agent wins round!
...
Agent wins the game!
Total Reward (Value Iteration): 3

Simulating the game using TD(0) learned value function:
Agent Deck: [1, 3, 7, 8, 9], Opponent Deck: [10, 6, 2, 4, 5], Opponent
Card Shown: 10
Opponent wins round!
...
```

```
Agent wins the game!
Total Reward (TD(0)): 1
```

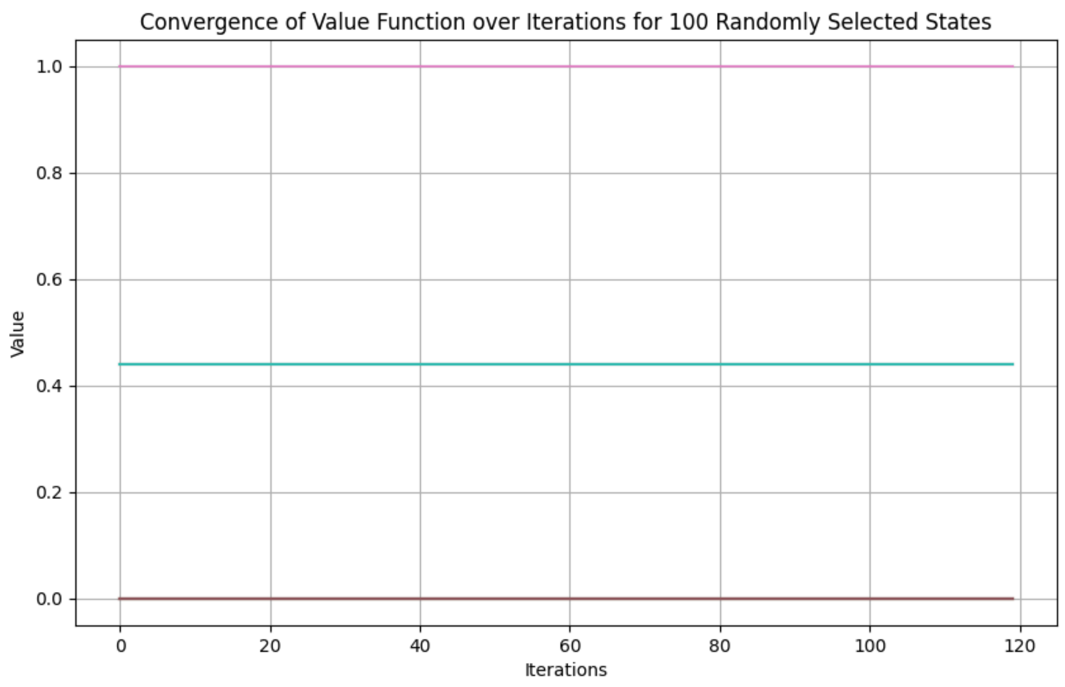
Comparison:

In 10 random runs, Value Iteration consistently outperformed TD(0), providing better strategies. For instance, in the simulation example provided, VI won the game with a total reward of 3, while TD(0) earned only 1.

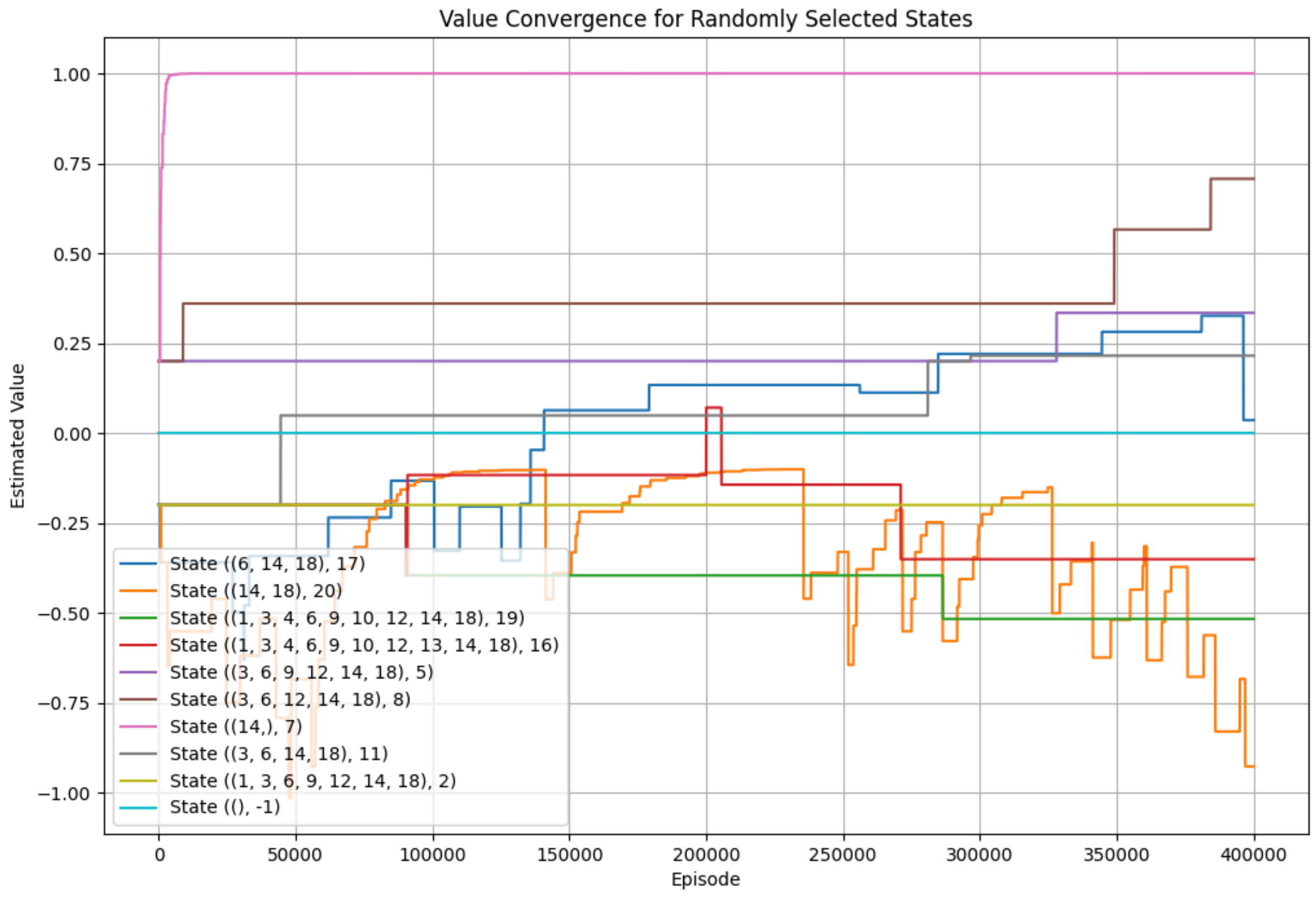
```
Example Scenario (VI won, TD lost):
Agent Deck: [2, 4, 6, 7, 10], Opponent Deck: [1, 3, 9, 8, 5], Opponent
Card Shown: 1
Agent wins the game with a reward of 3 using Value Iteration.
TD(0) lost the game with a reward of -1.
```

Value Function Comparison:

- For Value Iteration, the value function converges after 2 iterations for small n (e.g., n <= 10).

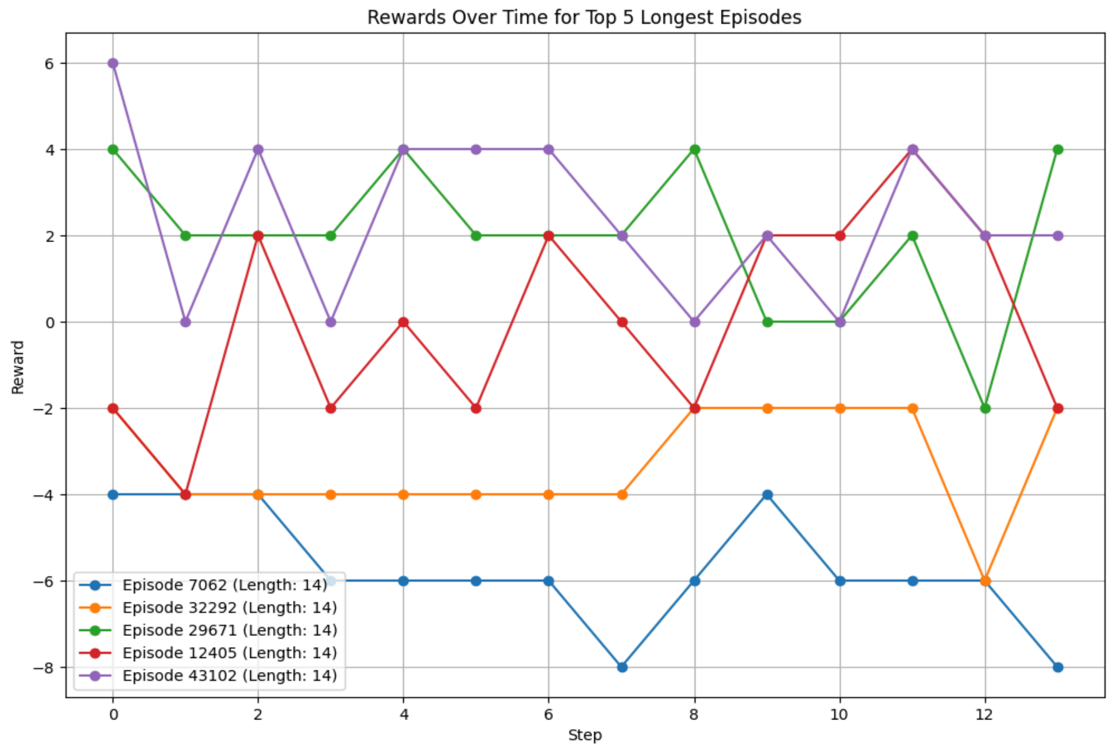


- For TD(0), the value function convergence was slower and dependent on the number of episodes and exploration.


Value Convergence for Randomly Selected States
Estimated Value
Episode
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50000
100000
150000
200000
250000
300000
350000
400000
State ((6, 14, 18), 17)
State ((14, 18), 20)
State ((1, 3, 4, 6, 9, 10, 12, 14, 18), 19)
State ((1, 3, 4, 6, 9, 10, 12, 13, 14, 18), 16)
State ((3, 6, 9, 12, 14, 18), 5)
State ((3, 6, 12, 14, 18), 8)
State ((14,), 7)
State ((3, 6, 14, 18), 11)
State ((1, 3, 6, 9, 12, 14, 18), 2)
State ((), -1)

Reward Convergence:

The reward convergence for TD(0) showed oscillations due to the limited number of iterations/occurences per episode. Episodes are indexed by a random seed.



Discussion:

Performance:

- Value Iteration outperformed TD(0) due to the deterministic nature of the environment. Since transitions were fixed (except for the initial random configuration), VI was able to exhaustively explore all states and determine an optimal strategy.
- The opponent’s strategy being fixed and the reduced state space allowed for efficient training in VI, at least for n <= 10.

Hardware Limitations:

Due to computational constraints, it was difficult to run Value Iteration for larger values of n. However, TD(0) could handle larger values of n due to its model-free approach and ability to learn from random episodes.

State Space Reduction:

A key optimization was representing the state space without considering the opponent's entire deck and only using the opponent’s current card. This effectively reduced the state complexity and transformed the problem into a POMDP.

Optimizations and Future Work:

Possible Optimizations:

1. State Representation: Representing the state as a binary list (e.g., -1 for unavailable cards and 0-1 for agent's deck) would further reduce complexity.
2. State Generation for VI: Instead of generating all states, VI could explore states based on transitions with random opponent actions. This would minimize redundant computations and make VI more scalable for larger values of n.

Max Possible Value of n:

- TD(0): Dependent on the number of episodes, larger n values can be handled.
- VI: For n <= 10, state space blowup is manageable, but higher values of n would require state reduction strategies or alternative methods.

Conclusion:

Value Iteration clearly outperformed TD(0) in this card game setting due to the deterministic nature of the environment. Optimizations such as state reduction and fixed opponent strategies allowed the VI algorithm to converge quickly and efficiently. TD(0), while more scalable, struggled to achieve the same level of performance due to its reliance on sample episodes and the need for extensive exploration.

Previous experiments (Appendix):

State: agent_deck, opponent_deck, rps_result
Issue: state blowup making training impossible for VI (for n>8)
Performance: VI outperforms TD(0) as it is an exhaustive and complete trained model.
Gamma (VI): 1, no discounting of transition state value
Hyperparameters for TD(0) remain the same.