REPORT on PI and MCC implementations

Policy Iteration (PI) Implementation

1. **Initialization**:
    - The `PolicyIteration` class is initialized with an environment (`env`) and a discount factor (`gamma`), defaulting to 0.9.
    - It initializes the policy and value table. The policy assigns random actions and velocities to each state (represented by `(x, y, orientation)`), while the value table starts with zeros for each state.
2. **Policy Evaluation**:
    - This step iteratively updates the value table based on the current policy. For each state, it:
        - Checks if the state is the target state (goal), where the value is set to 0.
        - Computes the expected value by simulating the next state using the environment's `step` method and applying the Bellman equation.
    - It stops updating a state if it encounters an invalid state or if the action in the policy is `None`.
3. **Policy Improvement**:
    - For each state, it evaluates all possible actions and velocities to find the best action that maximizes the expected return.
    - The policy is updated with the action that gives the highest value for each state.
4. **Run Policy Iteration**:
    - The `run_policy_iteration` method runs a loop for a given number of iterations.
    - In each iteration, it performs policy evaluation followed by policy improvement.
    - It also plots the value function at the start and halfway through the iterations to visualize the policy improvement process.
5. **Getters**:
    - `get_policy()` and `get_value_table()` are utility methods to retrieve the final policy and value table after running the iterations.

Monte Carlo Control (MCC) Implementation

1. **Initialization**:
    - The `MonteCarloControl` class is initialized with an environment (`env`), a discount factor (`gamma`), and an exploration rate (`epsilon`), defaulting to 0.1.
    - It initializes the Q-values (action-value function) and returns dictionary. Q-values are initialized to 0 for all state-action pairs, and returns are stored as lists.
2. **Generate Episodes**:
    - This function simulates an episode by starting at a random state or a predefined starting state.

- It follows an epsilon-greedy policy to choose actions, where with a probability of `epsilon`, it selects a random action, and with probability `1 - epsilon`, it selects the best action based on current Q-values.
- The episode continues until it reaches a terminal state, an invalid state, or revisits a previously visited state.

3. **Update Q-values**:
   - After generating an episode, it computes the return `G` for each state-action pair by traversing the episode in reverse.
   - It updates the Q-value for each state-action pair by averaging the returns obtained from multiple episodes (Monte Carlo estimation).
4. **Run Monte Carlo Simulation**:
   - The `run_monte_carlo` method executes the Monte Carlo control algorithm for a specified number of episodes.
   - It generates an episode, updates the Q-values, and periodically plots the value function to visualize the learning progress.
5. **Getters**:
   - `get_policy()` returns the policy derived from the Q-values, selecting the best action for each state.
   - `get_q_values()` returns the Q-values for all states based on the current policy.

SAMPLE RESULTS

Results - Target: **(3, 5, 'NE')** (i.e., x, y, orientation)

Start Position	Policy Iteration	Monte Carlo Control
(31, 29, 'N')	15	22
(17, 42, 'SW')	15	20
(-31, 29, 'N')	13	17
(-17, 42, 'SW')	15	22
(-31, -29, 'N')	12	16
(-17, -42, 'SW')	17	20
(31, -29, 'N')	12	17
(17, -42, 'SW')	17	26

100 Cycles of POLICY ITERATION ~ 120 to 160 sec

100,000 Episodes of MONTE CARLO SIMULATION ~ 80 to 90 sec

OBSERVATIONS AND ANALYSIS

From the results, it's evident that **Policy Iteration (PI)** reaches the target state in fewer steps than **Monte Carlo Control (MCC)** across all starting positions.

Here are the reasons why PI tends to perform better than MCC in this context:

1. **Model-Based vs. Model-Free Learning**:
    - **Policy Iteration** is a **model-based method**. It uses a complete model of the environment, meaning it knows the transition dynamics (state transitions based on actions) and rewards for each state-action pair. This allows PI to make more informed updates during both policy evaluation and improvement steps.
    - **Monte Carlo Control**, on the other hand, is a **model-free method**. It learns solely from episodes generated through interaction with the environment. This process is inherently more stochastic and can result in less precise value estimates, especially in the early stages of learning or when the number of episodes is not sufficiently large.
2. **Convergence Speed**:
    - **PI converges faster** to the optimal policy because it directly computes the value function for all states using the environment's model. The value iteration step in PI uses the Bellman equation, providing accurate value updates and leading to quick convergence to the optimal policy within a few iterations.
    - **MCC converges more slowly** because it relies on sampling episodes. Even with 100,000 episodes, the learning can still be noisy and require more time to stabilize the Q-values. This is due to the variability in the experiences sampled, particularly when using an epsilon-greedy policy, which introduces additional exploration randomness.
3. **Deterministic vs. Stochastic Policy Updates**:
    - **PI updates are deterministic** because it directly calculates the value function for all possible actions at each state and selects the best action. This deterministic approach ensures consistent improvement toward the optimal policy.
    - **MCC updates are stochastic** since it relies on sampled episodes. The learning process is influenced by the random nature of episode generation and the epsilon-greedy exploration, which can lead to suboptimal actions being chosen frequently in the early phases of learning.
4. **State-Action Space Coverage**:
    - **PI evaluates the value of all states simultaneously**, leading to a comprehensive policy update. This exhaustive evaluation ensures that the policy improves across all states in each iteration.
    - **MCC may not explore all states and actions equally** within a finite number of episodes, especially in large state spaces. Some states might be visited frequently, while others less so, resulting in uneven value updates. This can cause suboptimal policy performance if the algorithm hasn't sufficiently explored all state-action pairs.

5. **Sample Efficiency**:
    - **PI is more sample-efficient** because it doesn't need to rely on repeated sampling to update its policy. It systematically computes the best possible actions from the value function and policy.
    - **MCC requires a large number of samples (episodes) to achieve good performance** because it estimates values based on returns from sampled episodes. Even with a high number of episodes, there can still be significant variance in value estimates.
6. **Exploration vs. Exploitation Trade-off**:
    - **PI inherently balances exploration and exploitation** through its policy improvement step, which always chooses the best-known action according to the current value function.
    - **MCC employs an epsilon-greedy exploration strategy**, which means it sometimes chooses random actions to explore the state space. While exploration is crucial for discovering optimal policies, it can also lead to suboptimal actions, affecting short-term performance.
7. **Impact of Initial Policy and Value Estimates**:
    - **PI starts with an arbitrary policy but quickly refines it**, leveraging the full environment model to compute accurate value estimates, leading to fast convergence to optimal actions.
    - **MCC's initial Q-values and exploration strategy might cause slower initial performance** until enough episodes have been sampled to provide reliable Q-value estimates.

Conclusion

Overall, **Policy Iteration (PI) performs better than Monte Carlo Control (MCC) in the results because it is a more direct, deterministic, and model-based approach**. PI leverages the full environment model for faster convergence to optimal policies, while MCC, being model-free and sample-based, requires more episodes to achieve a comparable level of performance due to its reliance on sampled experiences and the inherent randomness of its learning process.