Debugging Techniques for Q-Learning in C++

Introduction

If after making adjustments, the agent's behavior still does not change, the problem might be in the following areas:

- **Q-Table not updating:** The agent might be stuck in some loop state or strategy, preventing the Q-Table from effectively updating.
- **Insufficient exploration:** If the agent has adopted a "lazy" strategy and the epsilon value is too low, it may always choose the same action, preventing it from exploring better paths.
- **Action not updated:** Although the code updates the Q-Table and selects actions, the actual action selection may not be reflected in the agent's behavior.

Deep Debugging Suggestions

1. Print Q-Table

Print the contents of the Q-Table at the end of each episode to see if the Q-values for state-action pairs are being updated, especially those that cause the agent to stay in one place.

2. Increase Exploration Rate

Increase the epsilon value to raise the probability of exploration and see if the agent tries different actions. You can add a conditional check in the `choose_action()` function to force exploration.

3. Output Debugging Information

Add more debug information in the `step()` function and the Q-Table update logic to check the state, action, reward, and Q-value changes at each step.

Detailed Debugging Code

Modify `choose_action` Function

```
int choose_action(int state) {
    if ((double)rand() / RAND_MAX < EPSILON) {  // If random number is less than
        int random_action = rand() % NUM_ACTIONS;  // Randomly choose an action
        std::cout << "Exploring: Chose action " << random_action << " for state
        return random_action;
    } else {  // Otherwise exploit
        int best_action = std::max_element(Q_table[state].begin(), Q_table[state
// Choose action with max Q-value
        std::cout << "Exploiting: Chose action " << best_action << " for state "
        return best_action;
    }
}
```

Modify `update_Q` Function

```
void update_Q(int state, int action, int reward, int next_state) {
    double best_next_action = *std::max_element(Q_table[next_state].begin(), Q_
// Find max Q-value for next state
    double old_value = Q_table[state][action];
    Q_table[state][action] += ALPHA * (reward + GAMMA * best_next_action - Q_tab
// Update Q-value using Q-Learning formula
    std::cout << "Updated Q(" << state << "," << action << ") from " << old_valu
}
```

Print Q-Table Contents

```
void print_Q_table() {
    std::cout << "Q-table:" << std::endl;
    for (size_t i = 0; i < Q_table.size(); ++i) {
        std::cout << "State " << i << ": ";
        for (size_t j = 0; j < Q_table[i].size(); ++j) {
            std::cout << Q_table[i][j] << " ";
        }
        std::cout << std::endl;
    }
}
```

Call Debug Functions in Main Loop

```
for (int episode = 0; episode < 1000; ++episode) {
    env.reset();  // Reset environment
    int state = env.get_state();  // Get current state
    int total_reward = 0;  // Initialize total reward to 0

    while (!env.is_done()) {  // While task is not done
        int action = agent.choose_action(state);  // Agent chooses action
        int reward = env.step(action);  // Execute action, get reward
        int next_state = env.get_state();  // Get next state
        agent.update_Q(state, action, reward, next_state);
// Update Q-table

        state = next_state;  // Update current state to next state
        total_reward += reward;  // Accumulate reward for this episode
    }

    std::cout << "Episode " << episode << ", Total Reward: " << total_reward <<
    env.render();  // Render environment to show agent and goal

    if (episode % 10 == 0) {
        agent.print_Q_table();  // Print Q-table every 10 episodes
    }
}
```

Analyze Debug Output

- Check if the Q-Table is being updated during training, particularly for states leading the agent to stay in one place. - Observe the action selection output to see if there is too much "exploitation," causing the agent to always choose the same action. - Through these debug messages, identify and fix the issue preventing the agent from moving.

Other Considerations

- If some Q-values in the Q-Table are too high, they may cause the agent to always choose those actions. Consider introducing a mechanism to periodically reset or smooth the Q-Table to avoid overly confident decisions. - If the above methods are still ineffective, try step-by-step debugging to ensure each part is functioning as expected.