

Project 3: MDP-Based Reinforcement Learning for Parallel Parking

Salleh Sonko (SID: 11735637)

Yahya Awaad (SID: 11703756)

Group 33

CSCE 5210.4 — Fundamentals of Artificial Intelligence, Fall 2025

1 Introduction

The purpose of this project is to gain a deeper understanding of how an AI agent can be modeled and implemented using reinforcement learning in a Markov Decision Process (MDP) framework. The specific task is to enable an agent to learn how to safely parallel park a car in a designated parking space between two already parked cars on an 8×5 grid. Hazards in the grid correspond to collisions with parked cars and are assigned a large negative reward, while successfully reaching the parking slot yields a large positive reward. All other non-terminal states receive a live-in reward that is varied experimentally. The agent operates under stochastic dynamics with a noise factor of 0.1, meaning that it does not always move in the intended direction, and must learn a robust policy that maximizes expected cumulative reward under this uncertainty.

2 Requirement R1: Value Iteration and Optimal Policy

2.1 MDP Setup

The environment is defined as an 8×5 grid. States are indexed as (c, r) , where $c \in \{1, \dots, 8\}$ is the column (from left to right) and $r \in \{1, \dots, 5\}$ is the row (from bottom to top). Two hazard states are located at $(1, 1)$ and $(1, 5)$ and are assigned a reward of -1000 . The goal state (the desired parking position) is at $(1, 3)$ and is assigned a reward of $+1000$. All other non-terminal states share a live-in reward r which is varied over the range $[-20, 0]$.

Seven start states, S_1-S_7 , are positioned on the top row (row 5) along columns 2 through 8:

$$S_1 = (2, 5), S_2 = (3, 5), S_3 = (4, 5), S_4 = (5, 5), S_5 = (6, 5), S_6 = (7, 5), S_7 = (8, 5).$$

2.2 Transition Model

The agent can attempt to move in one of eight compass directions:

$$\{N, NE, E, SE, S, SW, W, NW\}.$$

Let I denote the intended action from state s . With probability 0.9 the agent moves in the intended direction (provided the resulting state is not blocked). The remaining probability 0.1 is distributed

uniformly among non-blocked neighboring states that are reachable by actions not opposite (i.e., not 180°) to the intended action. If the intended next state is blocked (off the grid, an obstacle, or otherwise invalid), the agent remains in the current state with probability 1.0. Hazard and goal states are terminal with fixed utilities equal to their rewards.

2.3 Experimental Procedure

For each live-in reward $r \in \{-20, -19, \dots, 0\}$, Value Iteration was applied to compute a utility function $U(s)$ and an induced greedy policy $\pi(s)$ on the original environment (no internal obstacles). To evaluate and compare the policies, a policy score was computed by simulating greedy rollouts starting from each start state S_1 – S_7 , summing the utilities of the states along each trajectory until reaching a terminal state or a fixed step limit. The policy with the highest total score across all start states is designated as the best policy P_1 , and the first value of r that achieves this maximal score is reported.

2.4 Results for R1

The policy score increased monotonically with the live-in reward r across the tested range. The best score was achieved at:

$$r^* = 0.$$

For this value, the resulting policy P_1 steers all starting positions toward the goal while avoiding both hazards. The policy grid for P_1 is shown in Table 1. In this table, rows are listed from top (row 5) to bottom (row 1), and columns from left (column 1) to right (column 8). “H” denotes a hazard, “G” denotes the goal, and action labels (e.g., N, SW, W) indicate the chosen action in non-terminal states.

Table 1: Optimal policy P_1 for the original environment (live-in reward $r = 0$).

Row	C1	C2	C3	C4	C5	C6	C7	C8
5	H	SW	SW	SW	SW	SW	W	SW
4	S	SW						
3	G	W	W	W	W	W	W	W
2	N	NW						
1	H	NW	NW	NW	NW	NW	W	NW

The pattern in Table 1 clearly shows that the agent moves diagonally downward and leftward from the top-row start states toward the goal corridor around column 1 and row 3. On the goal row (row 3), all non-terminal cells direct the agent west toward (1, 3), while the actions below and above the goal funnel trajectories toward this target while staying clear of the hazards at (1, 1) and (1, 5).

3 Requirement R2: Policy Robustness Under Environmental Change

3.1 Modified Environment

For Requirement R2, the environment is modified by introducing a new obstacle at position (4, 3) (row 3, column 4). This cell is now treated as a blocked position that the agent cannot enter. The same MDP configuration and Value Iteration procedure are used, with the live-in reward fixed at the optimal value from R1, $r = 0$. The resulting policy on the modified environment is denoted by P_2 .

3.2 Comparison Between P_1 and P_2

A state-by-state comparison of P_1 and P_2 shows that the policies are identical for most states, but differ in a small set of states near the new obstacle. These include states such as (4, 3) (which now has no outgoing action because it is an obstacle), as well as neighboring positions whose optimal paths originally traversed (4, 3) in the obstacle-free environment.

Table 2 shows the policy grid for P_2 .

Table 2: Optimal policy P_2 for the environment with an obstacle at (4, 3).

Row	C1	C2	C3	C4	C5	C6	C7	C8
5	H	SW	SW	SW	SW	W	W	SW
4	S	SW	SW	SW	W	SW	SW	SW
3	G	W	W	X	SW	W	W	W
2	N	NW	NW	NW	W	NW	NW	NW
1	H	NW	NW	NW	NW	W	W	NW

In Table 2, the cell (4, 3) is marked as “X” to indicate the obstacle. The agent now routes around this blocked position, for example by using different westward or diagonal moves in states near (4, 3). However, the overall structure of the policy remains similar: start states still move generally toward the goal corridor, and the agent continues to avoid the hazards. Thus, $P_2 \neq P_1$, but the differences are localized and attributable to the new obstacle’s effect on possible trajectories.

4 Requirement R3: Extension to Multi-Agent Navigation

In the original MDP formulation, the environment is static: obstacles and hazards do not move, and transition probabilities depend only on the agent’s actions and fixed grid constraints. In a multi-agent scenario, however, other agents act as dynamic obstacles: cells that are free at one time step may be occupied at the next. To handle this, we can extend the MDP by introducing a time-varying occupancy probability $P_{\text{occ}}(s, t)$ for each state s , representing the probability that some other agent is occupying s at time t .

The transition function is then modified so that the probability of safely entering a state s' becomes proportional to $(1 - P_{\text{occ}}(s', t))$, while a collision outcome is associated with probability

$P_{\text{occ}}(s', t)$ and a large negative reward. In this way, each transition encompasses both a safe and a collision branch, and the expected utility calculation accounts for the risk of entering a potentially occupied state. This approach preserves the MDP structure while allowing the agent to reason about mobile obstacles.

5 Pseudocode

In this section, we provide pseudocode for the Value Iteration algorithm and the modified transition probability function.

5.1 Value Iteration

Listing 1: Pseudocode for Value Iteration

```

# Input: MDP with states S, actions A(s), rewards R(s),
#        transition function P(s' | s, a),
#        discount factor gamma, threshold theta
# Output: Utility function U and policy pi

Initialize U[s] = 0 for all states s
repeat:
    delta = 0
    for each state s in S:
        if s is terminal:
            U_new = R(s)
        else:
            best_value = -infinity
            best_action = None
            for each action a in A(s):
                expected_utility = 0
                for each next state s_next in S:
                    p = P(s_next | s, a)
                    expected_utility = expected_utility + \
                        p * ( R(s_next) + gamma * U[s_next] )
                if expected_utility > best_value:
                    best_value = expected_utility
                    best_action = a
            U_new = best_value
            pi[s] = best_action
    delta = max(delta, abs(U_new - U[s]))
    U[s] = U_new
until delta < theta

return U, pi

```

5.2 Transition Function with Dynamic Occupancy (Conceptual Extension)

Listing 2: Pseudocode for modified transition with occupancy

```
# TransProbDynamic(s, a, t) returns the possible next states
# and their probabilities, considering occupancy risk.

def TransProbDynamic(s, a, t):
    transitions = {}

    # Base transitions from static model (no moving agents)
    for (next_state, base_prob) in TransProbStatic(s, a):

        # Occupancy probability of next_state at time t
        occ = P_occ[next_state][t]      # value between 0 and 1

        # Safe move to next_state with probability base_prob * (1 - occ)
        safe_prob = base_prob * (1 - occ)

        # Collision outcome with probability base_prob * occ
        collision_prob = base_prob * occ

        # Add safe transition
        if safe_prob > 0:
            transitions[("safe", next_state)] = safe_prob

        # Add collision transition (treated as separate outcome)
        if collision_prob > 0:
            transitions[("collision", next_state)] = collision_prob

    return transitions
```

Code Repository

The full implementation, including the Jupyter notebook and all source files used in this project, is publicly available on GitHub at the link below:

<https://github.com/Salboy06/mdp-parking-agent/blob/main/CSCE5210Proj3.ipynb>