

**Artificial Intelligence Systems**

Lab Report # 07

**Submitted by:** Ali Hamza

**Registration No:** B23F0063AI106

**Submitted to:** Dr. Mohsin

Date: 20th March, 2025

# Lab Task:

# Consider a simple grid world with the following properties:

# The grid is 4x4 (4 rows and 4 columns).

# The agent can take four possible actions in each state: up, down, left, or right.

# The goal is located at position (3, 3), and the agent receives a reward of 1 for reaching the goal and 0 for all other positions.

# The agent's objective is to find the optimal policy that maximizes long-term rewards using Value Iteration.

# Given this scenario:

# What is the purpose of the get\_next\_state function in the grid world?

# What is the significance of the discount factor (gamma) set to 0.9 in this scenario?

# Explain how the Value Iteration process works and how it contributes to finding the optimal policy in this grid world example.

# What is the resulting value function (V), and what does it represent?

# Based on the final output of the optimal policy, how should the agent navigate the grid to reach the goal?

*from* typing *import* Tuple  
*import* numpy *as* np  
  
*# Gridworld Parameters*ROWS, COLS = 4, 4  
ACTIONS = ('UP', 'DOWN', 'LEFT', 'RIGHT')  
START, GOAL = (0, 0), (3, 3)  
REWARD\_GOAL, REWARD\_OTHER = 1.0, 0.0  
GAMMA, THRESHOLD = 0.9, 1e-4  
ACTION\_PROB = 1 / len(ACTIONS) *# Uniform action probability  
  
def* get\_next\_state(state: Tuple[int, int], action: str) -> Tuple[int, int]:  
 *"""Returns the next state given a current state and an action."""* row, col = state  
 transitions = {  
 'UP': (max(row - 1, 0), col),  
 'DOWN': (min(row + 1, ROWS - 1), col),  
 'LEFT': (row, max(col - 1, 0)),  
 'RIGHT': (row, min(col + 1, COLS - 1))  
 }  
 *return* transitions[action]  
  
*def* value\_iteration() -> np.ndarray:  
 *"""Performs Value Iteration to compute the optimal value function."""* V = np.zeros((ROWS, COLS)) *# Initialize values  
 while True*:  
 delta = 0  
 *for* row *in* range(ROWS):  
 *for* col *in* range(COLS):  
 *if* (row, col) == GOAL:  
 *continue # Goal state remains unchanged* v\_old = V[row, col]  
  
 *# Compute Bellman update* V[row, col] = max(  
 ACTION\_PROB \* (  
 REWARD\_GOAL *if* get\_next\_state((row, col), action) == GOAL *else* REWARD\_OTHER + GAMMA \* V[get\_next\_state((row, col), action)]  
 ) *for* action *in* ACTIONS  
 )  
  
 delta = max(delta, abs(v\_old - V[row, col])) *# Track convergence  
 if* delta < THRESHOLD:  
 *break # Stop when convergence threshold is met  
 return* V  
  
*def* extract\_policy(V: np.ndarray) -> np.ndarray:  
 *"""Extracts the optimal policy from the computed value function."""* policy = np.full((ROWS, COLS), ' ', dtype=str)  
 *for* row *in* range(ROWS):  
 *for* col *in* range(COLS):  
 *if* (row, col) == GOAL:  
 policy[row, col] = 'G' *# Goal state marker  
 continue  
  
 # Find the best action* best\_action = max(ACTIONS, key=*lambda* action: REWARD\_GOAL *if* get\_next\_state((row, col), action) == GOAL *else* REWARD\_OTHER + GAMMA \* V[get\_next\_state((row, col), action)])  
 policy[row, col] = best\_action[0] *# Store the first letter of the best action  
 return* policy  
  
*def* main() -> *None*:  
 *"""Runs Value Iteration and extracts the optimal policy."""* V = value\_iteration()  
 policy = extract\_policy(V)  
  
 print("\nOptimal Value Function\n")  
 print(np.round(V, 2))  
 print("\nOptimal Policy 🚀\n")  
 print(policy)  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 main()

# 