University of the People

CS4408 Artificial Inteligence

Unit 6 Written Assignment 6

Liang Xiao

Below is an illustrative **pseudo-code** (in a Python-like style) for solving the described grid-world problem as a Markov Decision Process (MDP). The pseudo-code includes:

1. A data structure (2D array) to represent the grid and the different states (Start, Fire, Diamond, Blocked, etc.).
2. A function to determine which action to take based on the MDP solution (i.e., an optimal policy).
3. A reward policy that considers step costs, fire penalties, diamond reward, etc.
4. Handling of blocked and fire states within the policy computation.
5. A function to check whether the goal (diamond) is reached.

### Explanation of Key Parts

1. **Grid Representation**  
   A 2D array grid holds cell types:
   * 'S' for **Start**
   * 'F' for **Fire**
   * 'D' for **Diamond**
   * 'B' for **Blocked**
   * '.' for normal/empty cells
2. **Transition Function** (get\_next\_state\_and\_reward)
   * Checks validity of the move (i.e., not out-of-bounds and not blocked).
   * Returns the next state and an immediate reward based on whether the cell is Fire, Diamond, etc.
3. **Reward Policy**
   * **Fire** cell: large negative reward (FIRE\_PENALTY).
   * **Diamond** cell: large positive reward (DIAMOND\_REWARD).
   * Normal step: negative step cost (STEP\_COST).
   * Blocked or invalid move: either no movement or treat it as step cost.
4. **Value Iteration**
   * Standard MDP algorithm that iterates until convergence.
   * Updates the value V[s] of each state by looking at all possible actions and choosing the one that yields the highest expected return.
   * Extracts a policy by choosing the action that maximizes the value at each state.
5. **Handling Blocked & Fire States**
   * **Blocked**: the agent cannot move into these cells; they are simply invalid transitions.
   * **Fire**: terminal state with a large negative reward.
6. **Goal Check**
   * is\_goal\_achieved(row, col) returns True if the agent is in the Diamond cell.

This pseudo-code meets the assignment requirements:

* **Data Structure**: Uses a 2D list (grid) to represent the environment.
* **Action Decision**: Uses an MDP approach (value iteration) to decide the best action from each state.
* **Reward Policy**: Incorporates step costs, fire penalty, diamond reward.
* **Blocked/Fire States**: Handled in transitions and terminal checks.
* **Goal Test**: Checks if the diamond cell is reached at the end of the simulation.

### Where Each Requirement is Addressed

* **(Requirement 1)**:  
  The 2D list grid is used to store the environment with states ('S', 'F', 'D', 'B', '.'). This is a suitable data structure for a small, static grid.
* **(Requirement 2)**:  
  The algorithm uses **Value Iteration**, a standard MDP approach, to derive an optimal policy.
* **(Requirement 3)**:  
  Blocked states ('B') are handled in is\_valid\_state (they’re deemed invalid moves), and the function handle\_blocked\_state can be extended for dynamic adjustments if needed.
* **(Requirement 4)**:  
  Fire states ('F') are recognized as terminal with a large negative reward (FIRE\_PENALTY). The function handle\_fire\_state is provided for further logic if the agent encounters fire.
* **(Requirement 5)**:  
  The **reward policy** is explicitly considered in get\_next\_state\_and\_reward via STEP\_COST, FIRE\_PENALTY, and DIAMOND\_REWARD. The step costs, penalties, and goal rewards guide the agent’s policy.