**MADANAPALLE INSTITUTE OF TECHNOLOGY AND SCIENCE**

**DEPARTMENT OF COMPUTER APPLICATIONS**

**ASSIGNMENT-1**

**Subject Code:** 24MCAP107

**Subject Name**: ARTIFICIAL INTELLIGENCE

**Faculty Name:** Dr. T . Saravanan  **Due Date:** 20-05-2025

**Name of the Student: G.prathima**

**Section:** MCA-C

**Roll No:**24691F00I5 **Assignment No:**1

**Year/Semester:**1 year-2 Semester **Submission Date:**23-05-2025

**1.Use knowledge representation techniques to model the rules for traffic light control an intersection.**

To model the rules for traffic light control at an intersection using knowledge representation techniques, we can use a rule-based system combined with a semantic network or frame-based representation. Below is a structured approach using different knowledge representation techniques:

**1. Entities and Relationships –** Semantic Network or Ontology

Use a semantic network to represent key entities and their relationships.

**Entities:**

Intersection

Traffic Light

Direction (North, South, East, West)

Vehicle

Pedestrian

Signal Phase (Green, Yellow, Red)

**Relationships:**

has\_light (intersection, traffic\_light)

controls (traffic\_light, direction)

has\_phase (traffic\_light, signal\_phase)

is\_waiting (vehicle, direction)

is\_crossing (pedestrian, crosswalk)

Example (in Prolog-style logic or RDF triples):

has\_light(intersection1, light\_NS).

has\_light(intersection1, light\_EW).

controls(light\_NS, north).

controls(light\_NS, south).

controls(light\_EW, east).

controls(light\_EW, west).

**2. Rules (If-Then) –** Rule-Based System

**Rule 1:** Basic Phase Switching

IF phase(light\_NS) = green AND time\_elapsed >= t\_green

THEN phase(light\_NS) := yellow

**Rule 2:** Yellow to Red

IF phase(light\_NS) = yellow AND time\_elapsed >= t\_yellow

THEN phase(light\_NS) := red AND phase(light\_EW) := green

**Rule 3:** Pedestrian Crossing Request

IF pedestrian\_request(crosswalk\_NS) = true AND phase(light\_NS) = red

THEN allow\_pedestrian\_crossing(crosswalk\_NS)

**Rule 4:** Emergency Vehicle Detection

IF detect\_emergency\_vehicle(direction)

THEN override\_normal\_cycle AND phase(direction\_light) := green

**3. Frame-Based Representation**

Frame: Traffic\_Light

Traffic\_Light:

ID: light\_NS

Directions\_Controlled: [north, south]

Current\_Phase: green

Timings:

green: 30s

yellow: 5s

red: 35s

Frame: Intersection\_Control\_System

Intersection\_Control\_System:

Intersection\_ID: intersection1

Lights: [light\_NS, light\_EW]

Sensors: [vehicle\_detector, pedestrian\_button]

Functions:

- cycle\_lights

- detect\_vehicles

- handle\_pedestrian\_requests

**4. Temporal Logic (Optional)**

You can use temporal logic to express sequential constraints:

G (phase(light\_NS) = green → F phase(light\_NS) = yellow)

(Globally: if light\_NS is green, then eventually it must turn yellow)

phase(light\_NS) = green U time\_elapsed > t\_green

(Until: NS stays green until time elapses)

**Summary**

**Technique Role**

Semantic Network Defines structure of entities and relationships

Rule-Based Logic Models control behavior with IF-THEN rules

Frames Encapsulates data for objects like lights and systems

Temporal Logic Manages time-based transitions and guarantees

2. **Demonstrate how the Best First Search algorithm combined with a heuristic function can improve the AI’s performance in Tic-Tac-Toe by pruning unnecessary branches.**

Certainly! the combining Best First Search (BFS) with a heuristic function can enhance AI performance in Tic-Tac-Toe by effectively pruning unnecessary branches:

Best First Search with Heuristic in Tic-Tac-Toe

**What It Is**

\* Best First Search (BFS) is an informed search algorithm that selects the most promising node based on a heuristic evaluation function.

\*In the context of Tic-Tac-Toe, BFS can prioritize exploring game states that are more likely to lead to a win or a favorable outcome, thereby reducing the number of nodes explored.

**How It Works**

**Heuristic Evaluation Function:** Assigns a score to each game state. For example:

\* +10 for a win.

\*-10 for a loss.

\*0 for a draw or ongoing game.

**Priority Queue:** Stores game states, with the most promising ones (based on the heuristic score) given higher priority.

**Search Process:**

Start with the initial game state.

Generate all possible next moves.

Evaluate each resulting game state using the heuristic function.

Add the evaluated states to the priority queue.

Select the state with the highest priority (best score) for further exploration.

Repeat the process until a terminal state (win, loss, or draw) is reached.

**Pruning Unnecessary Branches**

**Selective Exploration:** By focusing on the most promising moves, BFS avoids exploring less favorable branches of the game tree.

**Efficiency:** This selective approach reduces the computational complexity, allowing the AI to make decisions more quickly and effectively.

**Benefits**

**Improved Performance:** The AI can make optimal or near-optimal moves without exhaustively searching all possible game states.

**Faster Decision-Making:** By pruning less promising branches, the AI can evaluate potential moves more efficiently.

**Scalability:** The approach can be extended to more complex games with larger state spaces.