# ACI Assignment- II Problem Statement- II

# Group 250

Participants:

|  |  |  |  |
| --- | --- | --- | --- |
| **S.I.** | **Name** | **ID** | **Contribution %** |
| 1 | Rahul | 2024ad05284 | 100 |
| 2 | Brijesh | 2024ad05270 | 100 |
| 3 | Mandan | 2024ad05331 | 100 |
| 4 | Surya Gayathri | 2024ad05359 | 100 |
| 5 | Pranav Deep | 2024ad05376 | 100 |

Part A:

1. **You are free to choose your own static evaluation function. Justify your choice of static evaluation value design and explain with a sample game state. Do not use any machine learning model for the evaluation function.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

EVALUATION FUNCTION

A static evaluation function is required for non-terminal states (since Minimax will expand possible moves up to some depth).

The key idea

* Reward AI (T) for creating sequences of 2, 3, or 4 in a row / column / diagonal
* Penalize Human (X) for creating similar sequences.
* The closer a player is to 4 in a row, the higher the score.

Design

* **Winning State:**  
  If the AI wins → +∞ (large positive, e.g., +1000)  
  If the human wins → -∞ (large negative, e.g., -1000)
* **Intermediate State:**  
  Assign scores based on sequences of consecutive marks:
  + 3 in a row (open on both ends): +100 if AI, -100 if human
  + 2 in a row (open): +10 if AI, -10 if human

This ensures:

* AI prioritizes winning.
* AI blocks human’s winning threats.
* AI strategically builds up to 4-in-a-row.

Sample Game State

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **X** |  | **T** | **T** |  |
| **X** | **T** | **T** |  |  |
|  | **X** |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Step 1 : Check sequences

1. Row 0 -> TT -> AI gets +10
2. Row 1 -> TT -> AI gets +10
3. Column 0 -> XX -> Human gets -10
4. Diagonal (1,0 -> 2,1) -> XX -> Human gets -10

Diagonal (1,1 -> 0,2) -> TT -> AI gets +10

Diagonal (1,2 -> 0,3) -> TT -> AI gets +10

Step 2 : Total Evaluation

E = +10 + (+10) + (-10) + (-10) + (+10) + (+10)

= +20

So this state slightly favors the AI

**B. Similar to the virtual lab example, one of the players must be a human ie., it must get dynamic inputs from us. The other player must be simulated using the program.**

1. Human to provides input (row, col for their move).

2. AI uses Minimax (with alpha-beta pruning for efficiency).

3. The program alternates turns until a win/draw.

Part B:

## Rules:

We can convert given decision tree into an **if-then** rule for Prolog programming as follows:

1. **If** *MajorAxisLength* is less than equal to 422.279133 and *Perimeter* is less than equal to 1006.375 **then** type is **Kecimen**.
2. **If** *MajorAxisLength* is less than equal to 422.279133 and *Perimeter* is greater than 1006.375 and *Extent* is less than equal to 0.7476 and *Perimeter* is less than equal to 1122.831 and *Area* is less than equal to 62835 and *Extent* is less than equal to 0.666255 **then** type is **Besni**.
3. **If** *MajorAxisLength* is less than equal to 422.279133 and *Perimeter* is greater than 1006.375 and *Extent* is less than equal to 0.7476 and *perimeter* is less than equal to 1122.831 and *Area* is less than equal to 62835 and *Extent* is less than equal to 0.701678 but greater than 0.666255 **then** type is **Kecimen**.
4. **If** *MajorAxisLength* is less than equal to 422.279133 and *Perimeter* is greater than 1006.375 and *Extent* is less than equal to 0.7476 and *perimeter* is less than equal to 1122.831 and *Area* is less than equal to 62835 and *Extent* is greater than 0.701678 **then** type is **Besni**.
5. **If** *MajorAxisLength* is less than equal to 422.279133 and *Perimeter* is greater than 1006.375 and *Extent* is less than equal to 0.7476 and *Perimeter* is less than equal to 1122.831 and *Area* is greater than 62835 **then** type is **Kecimen**.
6. **If** *MajorAxisLength* is less than equal to 422.279133 and *Perimeter* is greater than 1006.375 and *Extent* is less than equal to 0.7476 and *Perimeter* is greater than 1122.831 and *Extent* is less than equal to 0.671309 **then** type is **Besni**.
7. **If** *MajorAxisLength* is less than equal to 422.279133 and *Perimeter* is greater than 1006.375 and *Extent* is less than equal to 0.7476 and *Perimeter* is greater than 1122.831 and *Extent* is greater than 0.671309 and *Eccentricity* is less than equal to 0.75951 **then** type is **Besni**.
8. **If** *MajorAxisLength* is less than equal to 422.279133 and *Perimeter* is greater than 1006.375 and *Extent* is less than equal to 0.7476 and *Perimeter* is greater than 1122.831 and *Extent* is greater than 0.671309 and *Eccentricity* is greater than 0.75951 **then** type is **Kecimen**.
9. **If** *MajorAxisLength* is less than equal to 422.279133 and *Perimeter* is greater than 1006.375 and *Extent* is greater than 0.7476 **then** type is **Kecimen**.
10. **if** *MajorAxisLength* is greater than 422.279133 **then** type is **Besni**.

Please see attached **raisin\_classification\_g250.pl** for complete implementation of above rules in Prolog.

## Execution Screenshots:

Execution screen shot for below test data verified against implemented Prolog login for given decision tree.

1. classify\_raisin(422.279133, 1010.0, 0.68, 62000, 0.69951, Type).
2. classify\_raisin(422.279133, 1010.0, 0.65, 62000, 0.69951, Type).
3. classify\_raisin(422.279133, 1010.0, 0.72, 62900, 0.69951, Type).
4. classify\_raisin(422.279133, 1210.0, 0.72, 62900, 0.69951, Type).
5. classify\_raisin(422.279133, 1210.0, 0.72, 62900, 0.76951, Type).
6. classify\_raisin(422.279133, 1210.0, 0.62, 62900, 0.76951, Type).
7. classify\_raisin(422.289133, 1210.0, 0.62, 62900, 0.76951, Type).
8. classify\_raisin(422.289133, 1210.0, 0.62, 62900, \_, Type).

