Birla Institute of Science and Technology

Artificial and Computational Intelligence

PS1: Game Playing in Python

Abstract:

In this problem, we will be dealing with the **Catch-Up with Numbers** game problem of decision-making and considering the Minmax optimization technique used for selecting the best move.

Task Environment:	Game State, Players, Set of Natural Numbers, Randomization, Strategy, Win/Lose Conditions, Freedom of state, Game Rules, Freedom of moment, Utility function
Algorithm:	Minmax with Alph-beta pruning
Fully vs Partially: **Observable:**	Fully observable
Single vs Multi-Agent:	Multi-Agent
Deterministic vs Stochastic:	Deterministic
Episodic vs Sequential:	Sequential
Static vs Dynamic:	Dynamic
Discrete vs Continuous:	Discrete

PEAS Environment

Performance:

- Every player aims to get a higher score than the opponent or secure a tie if winning is not feasible. The efficiency of the Minimax algorithm with Alpha-Beta pruning is crucial for larger game states. It ensures that the algorithm performs well, especially as the size of the number set increases.

Environment:

G. Ankur Vatsa 2023aa05727
Nidasanametla Sree Sitamahalakshmi 2023aa05716
Prasenjit Samantha 2023aa05256
Randhawane Santosh 2023aa05828
Vedagiri Sai Krishna 2023aa05348

- The environment consists of the set of numbers {1, 2, 3, ..., n} where **n** will be provided by the player as an input.
- Two player's game.
- It also includes the rules of the game, such as
    - players taking turns choosing numbers
    - numbers being removed from the set after being chosen
    - the condition for ending the game.

Actuators:

- It helps to prune unnecessary branches and evaluates different move possibilities for each player
- It is using the mechanism of Minimax with alpha-beta pruning for decision making.

Sensors:

- The sensors provide information about the state of the game, including the current set of available numbers, the sum chosen by the opponent in the previous turn, possibly the choices made by the opponent and player scores.

Task Environment

Players: Two players take turns making choices from a set of numbers. Each player aims are to maximize their own score or minimize their loss while considering others players moves.

Game State: Current state of the game where each player has a current score representing the sum of the numbers they have chosen so far. The current score is important for determining the available move for the opponent player.

Set of Natural Numbers: The initial set consists of numbers starting from 1 to a predetermined number n.

Game Rules: Players alternate turns and each player selects one or more numbers from the remaining set, aiming to either exceed or match their opponent player score.

Objective: Players aim to maximize their score or minimize their loss or secure a tie if winning is not feasible.

Randomization: When making moves, each player must randomize their selection from the available number set, considering the chosen move equal or exceed their opponent's last total. Which ensures the fairness and unpredictability in the game.

Strategy: Each player's plan is to anticipate their opponent's moves to maximize their own score and adapt the changing game states based on opponent's move.

Win/Lose Conditions: Winning or losing the game depends on,

- Game ends when all numbers have been chosen.
- If one player has a higher sum than the other, that player wins. If not the game ends in a tie.

Scoring System: Implement a scoring system based on factors such as the number chosen by each player, the efficiency of operations etc.

Designing of Catch-Up with Numbers game using Minmax algorithms.

The complete code has been segregated into three major classes,

- MinimaxStrategy
- Player
- CatchUpGame

MinimaxStrategy

Implements the Minimax algorithm with alpha-beta pruning. It helps to determines the best move for the current player by considering the opponent's potential moves and maximizing the player's score while minimizing the opponent's score.

Key Objectives:

- **MinimaxStrategy::move( )**

This method implements the core Minimax logic with alpha-beta pruning. It explores the game tree using recursion and evaluates different move likelihoods for each player.

- **MinimaxStrategy::_alpha_beta_pruning( )**

This method helps to prune unnecessary branches in the game tree based on alpha and beta values, refining the proficiency of the search.

This search algorithm helps to reduces the number of nodes evaluated in the minimax algorithm.

- It explores the game tree in a depth-first manner
- At each level, the algorithm applies the minimax principle in order to evaluate the potential moves.
- This algorithm maintains two values, alpha and beta, which represent the minimum score the maximizing player is secure of and the maximum score the minimizing player is secure of, individually, along the process of path exploration.

- During the search process if, the algorithm finds a move that leads to a score worse than what the other player already has available, it recognizes that further exploration of this branch is unnecessary. Hence, it starts prunes this branch of the game tree.

- **MinimaxStrategy::_minimax( )**

This private method recursively evaluates the game state from the perspective of both players, considering their potential moves and scores.

This algorithm is a decision-making algorithm.

- Game state and possible moves are represented as a tree
- Each level of the tree represents a player's turn, and each node represents a particular game state.
- Evaluation function assigns a score to each game state at the terminal level. This also helps to determines how good a particular game state is for the maximizing player and how bad it is for the minimizing player.
- This algorithm recursively explores the game tree, considering all possible moves that can be made by both players.
- As soon as the algorithm travels deeper into the tree, it propagates the scores back up to the root node. This process continues until the algorithm find the optimal move based on the obtained scores.
- Once the entire tree evaluation completed, the algorithm identifies the best move that takes to the best possible result for the maximizing player.

Player

Depicts a player in the game, including their name, score, and chosen strategy.

Key Objectives:

- **Player::choose_strategy( )**

This method allows a player to select their approach (maximiser or minimizer) using user input.

- **Player::make_move( )**

This method calls the player's chosen approach to determine the best move considering the current game state.

CatchUpGame

Manages the overall game flow, including determining the starting player, checking the game state, identifying the winner, and handling turns for each player.

Key Objectives:

- **CatchUpGame::is_game_over( )**

  This method helps to evaluate the game ending condition by confirming, there are no more available numbers or neither player can make a valid move.

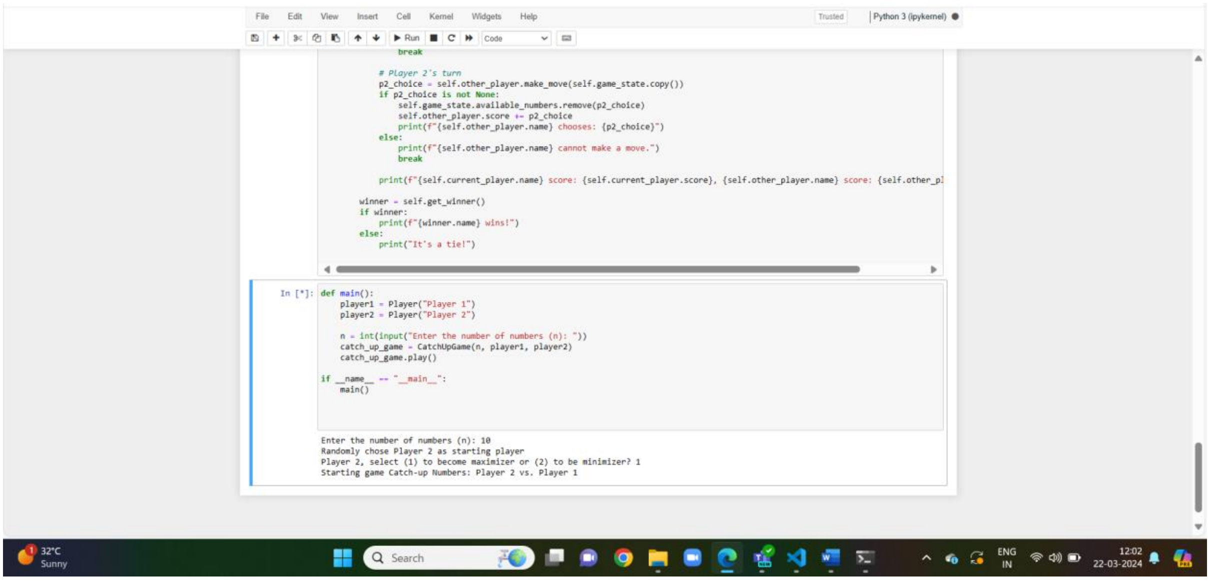
- **CatchUpGame::get_winner( )**

  This method helps to governs the winner based on the player's scores.

- **CatchUpGame::play( )**

  This method handles the entire game loop, counting, exposing available numbers, prompting players for moves, scores pupation, and declaring the game output.

Output:



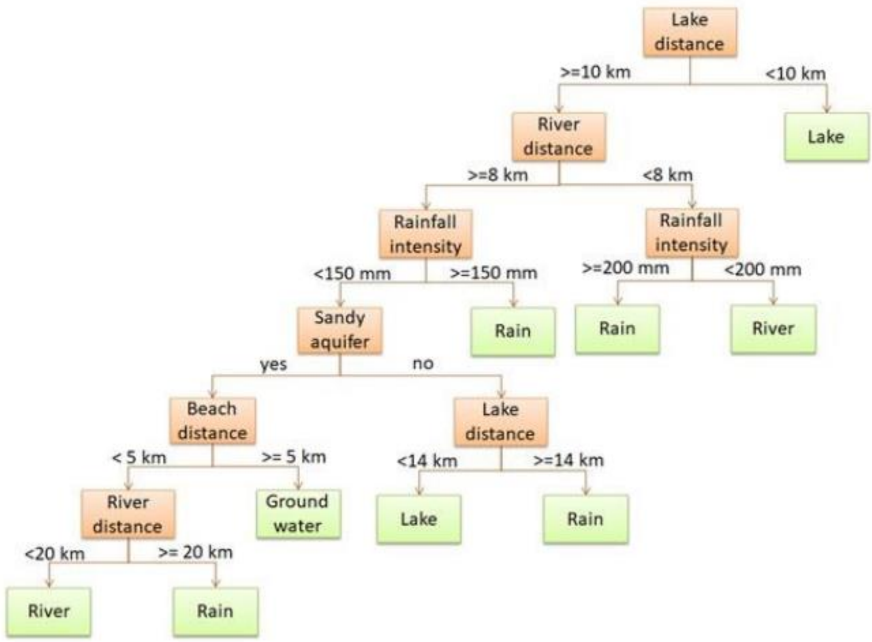
PS2: Logic design and inferencing in Prolog.

Abstract:

A decision-making process to determine the source of water based on various parameters like distance from a lake or river, rainfall intensity, and others.

In this problem, we are determining the best water source for a community using logic programming considering various parameters like distance from a lake or river, rainfall intensity, and others.

Decision tree provided for creation of Prolog rules:



G. Ankur Vatsa 2023aa05727
Nidasanametla Sree Sitamahalakshmi 2023aa05716
Prasenjit Samantha 2023aa05256
Randhawane Santosh 2023aa05828
Vedagiri Sai Krishna 2023aa05348

Designing logic and prolog inferencing

Logic implementation:

- **Prolog Rules:**

Program

```prolog
predict_water_source(Distance_from_Lake, Distance_from_River, Rainfall_intensity, Sandy_aquifer, Distance_from_Beach, WaterSource) :-
(
    Distance_from_Lake < 10,
    WaterSource = lake
);
(
    Distance_from_Lake >= 10,
    Distance_from_River < 8,
    Rainfall_intensity < 200,
    WaterSource = river
);
(
    Distance_from_Lake >= 10,
    Distance_from_River < 8,
    Rainfall_intensity >= 200,
    WaterSource = rain
);
(
    Distance_from_Lake >= 10,
    Distance_from_River >= 8,
    Rainfall_intensity >= 150,
    WaterSource = rain
);
(
    Distance_from_Lake < 14,
    Distance_from_River >= 8,
    Rainfall_intensity < 150,
    Sandy_aquifer == no,
    WaterSource = lake
);
```

Program

```prolog
(
    Distance_from_Lake >= 14,
    Distance_from_River >= 8,
    Rainfall_intensity < 150,
    Sandy_aquifer == no,
    WaterSource = rain
);
(
    Distance_from_Lake >= 10,
    Distance_from_River >= 8,
    Rainfall_intensity < 150,
    Sandy_aquifer == yes,
    Distance_from_Beach >= 5,
    WaterSource = groundwater
);
(
    Distance_from_Lake >= 10,
    Distance_from_River >= 20,
    Rainfall_intensity < 150,
    Sandy_aquifer == yes,
    Distance_from_Beach < 5,
    WaterSource = rain
);
(
    Distance_from_Lake >= 10,
    Distance_from_River < 20,
    Rainfall_intensity < 150,
    Sandy_aquifer == yes,
    Distance_from_Beach < 5,
    WaterSource = river
);
% default to groundwater
WaterSource = groundwater.
```


G. Ankur Vatsa	2023aa05727
Nidasanametla Sree Sitamahalakshmi	2023aa05716
Prasenjit Samantha	2023aa05256
Randhawane Santosh	2023aa05828
Vedagiri Sai Krishna	2023aa05348

- **Get_user_input**:
  Deals with user provided input at runtime

```
get_user_input :-
    write('Distance from Lake (km): '),
    read(Distance_from_Lake),
    write('Distance from River (km): '),
    read(Distance_from_River),
    write('Enter Rainfall_intensity (mm/month): '),
    read(Rainfall_intensity),
    write('Is the Sandy_aquifer Sandy? (yes/no): '),
    read(Sandy_aquifer),
    write('Distance from Beach (km): '),
    read(Distance_from_Beach),
    predict_water_source(Distance_from_Lake, Distance_from_River, Rainfall_intensity, Sandy_aquifer, Distance_from_Beach, WaterSource),
    write('Recommended Water Source: '),
    write(WaterSource).
```

- **Start**:
  Act as a main function or initiate rule execution program

```
% Start the program
start :-
    write('Welcome to the Water Source Prediction System!'), nl,
    get_user_input,
    nl,
    write('Thank you!').
```

Example: if there is a case where a location has features: 120 mm/month rainfall, sandy aquifer, 10 km away from the perennial river, 20 km away from the lake, and 2 km away from the beach, could you decide which water resource suitable for the community to take from, rainfall, river water, lake, or groundwater?

Features identified:

Rainfall → 120 mm/month

Sandy aquifer → yes

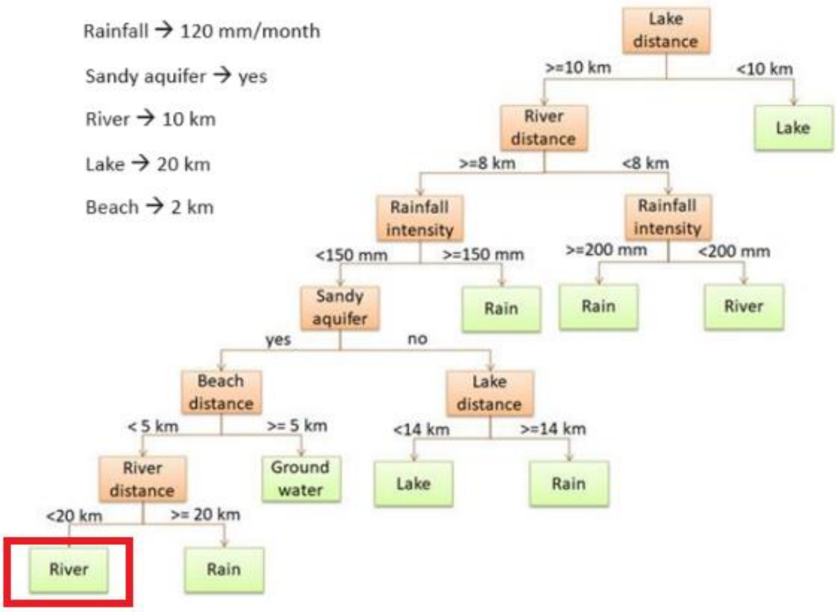
River → 10 km

Lake → 20 km

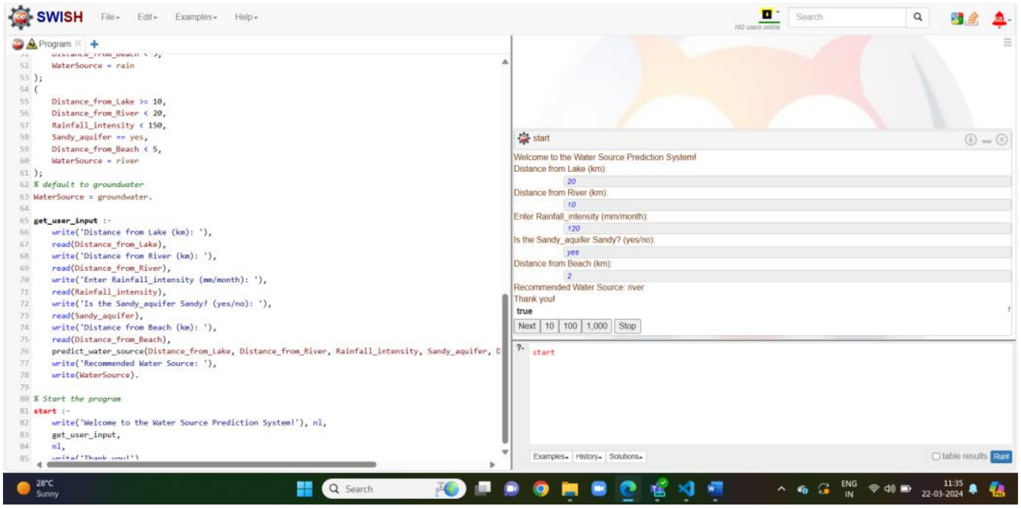
Beach → 2 km

G. Ankur Vatsa 2023aa05727
Nidasanametla Sree Sitamahalakshmi 2023aa05716
Prasenjit Samantha 2023aa05256
Randhawane Santosh 2023aa05828
Vedagiri Sai Krishna 2023aa05348

Output: River is the expected output considering the given decision tree.



User Input/Output:



GIT Reference: [mtech/semester_1/03_assignments/aci/ASSIGNMENT_2 at main · vatsaaa/mtech · GitHub]()

G. Ankur Vatsa	2023aa05727
Nidasanametla Sree Sitamahalakshmi	2023aa05716
Prasenjit Samantha	2023aa05256
Randhawane Santosh	2023aa05828
Vedagiri Sai Krishna	2023aa05348