TARS - The AI Bot for 3 Dimensional Tic-Tac-Toe

PROJECT REPORT - 2

R.Shyam Sundar (CS20B1029) & **Daathwinaagh** (CS20B1031)

Indian Institute of Information and Technology, Design & Manufacturing, Kancheepuram.

Course : Artificial Intelligence.

Instructor : Ram Prasad Padhy

Date : 14th, April, 2023

This Project report describes the aspects that are given below

- Problem Statement
- Implementation of User Interface.
- Algorithm Explanation
- Usage of AI

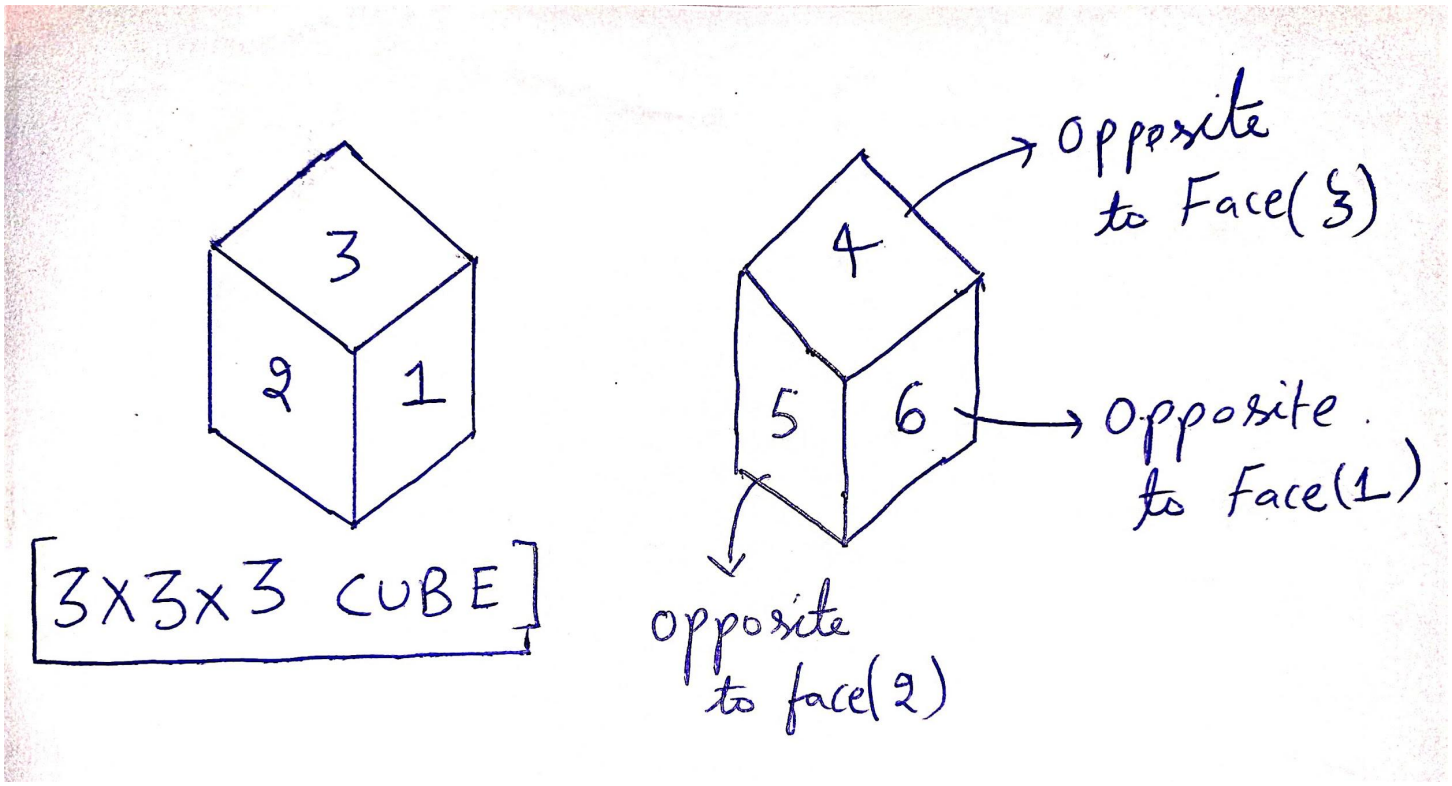
Problem Statement

Develop a 3D Tic Tac Toe game with a user interface that allows players to play against an AI bot. Such that

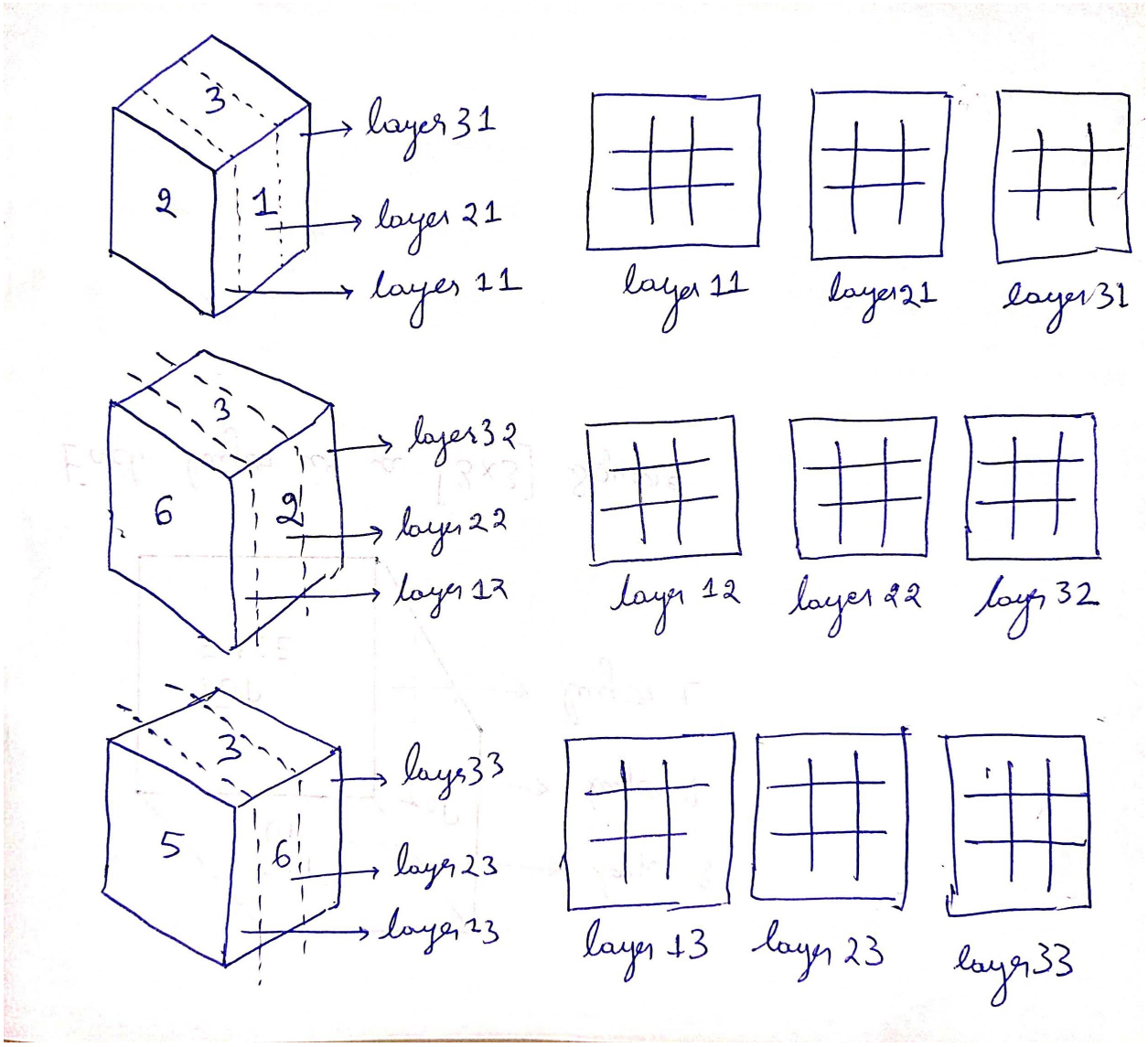
- The game has an option to choose the player who is going to make the first move.
- The bot makes strategic moves to win the game or stop the player from winning. The user interface is visually appealing and easy to use.
- The application is developed using Python and PyQt5 - GUI development.

Implementation of User Interface

Description of board :



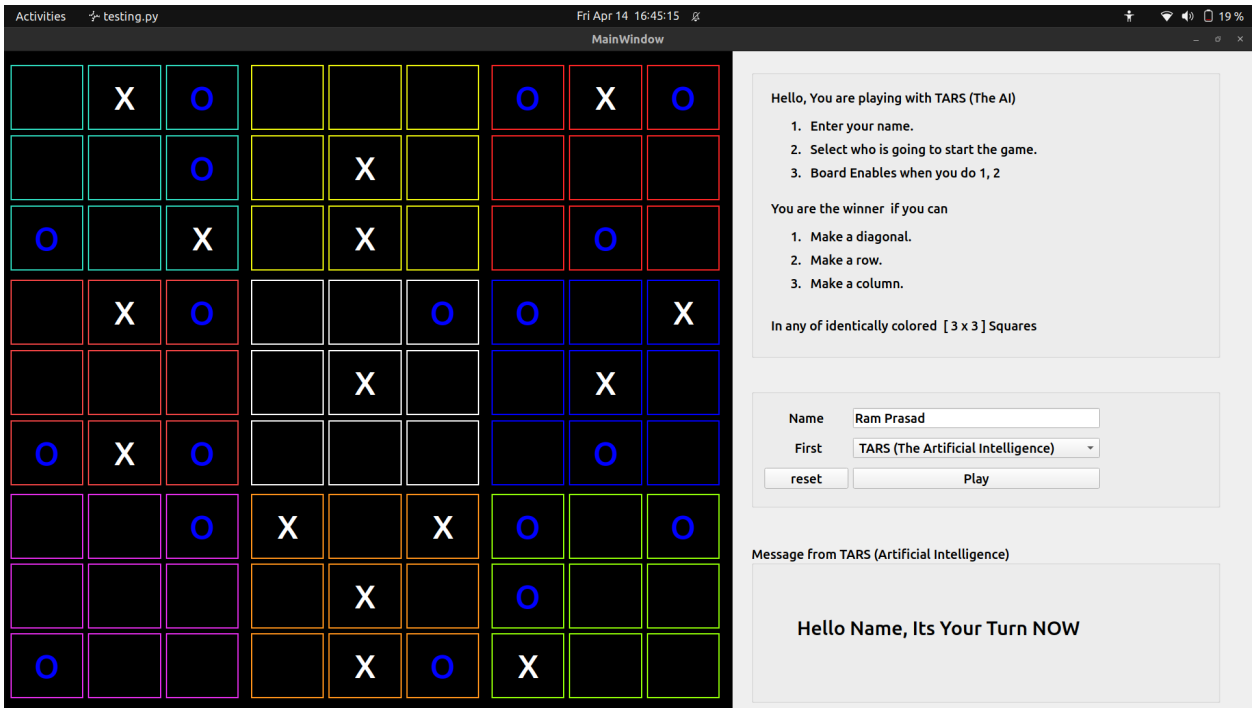
Intuition of cutting the 3 x 3 x 3 cube into three layers from 3 adjacent faces such that we obtain **Nine** [3 x 3] squares.



So presenting these 9 squares to the user will solve the problem. i,e user has to play the 2 Dimensional Tic-Tac-Toe and should find the solution to any one of these 3 x 3 squares to be the winner of the game. The total number of cells in all 3 x 3 squares is more than the cells in 3 x 3 x

3 board. So, UI can be developed to select the extra cells when a cell is selected. Because each cell is repeated thrice here.

The implemented User Interface is



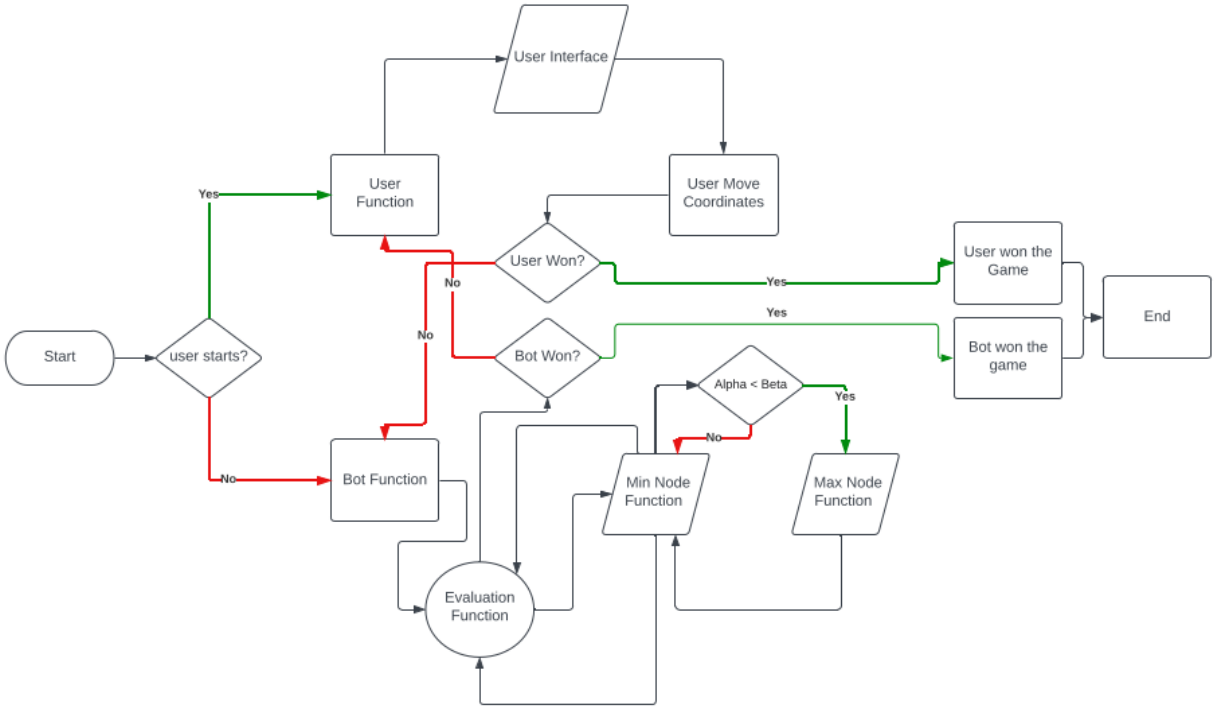
Algorithm Explanation:

To find the best possible move at each instant the Bot uses Minimax Algorithm along with Alpha beta pruning to optimize the search process.

user()	Function responsible for getting input from the user.
pc()	Function responsible for making the bot move. The move coordinates are received from evaluate() function.
turn()	To alternate between user() and pc().

evaluate()	Starts the branching of Game Tree from the root state(present state). Initially ALPHA* = -1000. This function creates a temporary board for modification and search process. This function makes all possible moves for the Bot, in the current state and calls the min_node(). Which would further continue the search. ALPHA = maximum value obtained from search till now from its successors.
min_node()	This function keeps all possible moves for the user and calls the max_node at each stage to continue the game tree. OPTIMIZATION: if ALPHA>BETA* , the the search would be stopped and the best coordinate till now will be returned. BETA = minimum value obtained till now from search from its successors.
max_node()	This function makes all possible moves for the Bot and calls the min_node(). (Min node and Max node call each other recursively until the entire state space is searched with pruning). ALPHA value is updated at this point so that its immediate successors can use alpha-beta pruning.
check_won()	An helper function to check is whether USER/BOT is won.
moves_left()	An helper function to check if there is any possible square to make a move. (If no such square is present, the game ends in a draw).

Complete Algorithm Flow Chart:



Usage of AI:

The number of **possibilities** of playing a **3 x 3 x 3 Tic Tac Toe game** is ~**108888694504183521607680000000** ways (27!). A naive unbeatable Tic-Tac-Toe bot using brute force would take years to compute the best possible move at a particular instant while playing against a player.

Since we have used Game Trees and Alpha beta pruning we have reduced the time complexity of search exponentially but removing unwanted branch searches. Thus it is assured that the Bot plays very fast and also the strongest move at each turn.

Thank you!