**Microsoft Mars Colonization Program**

*Team: EPCDECRY*

*Rajendraprasath and Rohit*

**Problem chosen:**

***(Project 2)Entertain the Crew***: Engage your Crew by Using Minimax algorithm to build an unbeatable Tic-Tac-Toe game powered by AI.

**Our Ideas:**

* *3x3 and 4x4 board on a single webpage.*
* *Options of Player vs Player (PvP) and Player vs Computer (PvE).*
* *Score feature.*
* *Make the AI unbeatable.*
* *Include option for changing the level of AI so that the player has a chance of winning.*
* *Include a guiding feature which helps the human player to make an optimal move.*
* *Timeline feature to review the moves made in a game.*

**Implementation:**

The tic-tac-toe game is set up on a **3 X 3** board. Underneath the 3x3 board, the crew member has a choice to either to play with his fellow crewmate **(PvP)** or to play against the AI for which the levels (depth) can be modified **( PvE )**. If the player is in need of a help, he can click on the **Guided Mode** which gives suggestions for the next move. The option of **Guided Mode** is available in both PvP and PvE modes. After a Win or a Tie, the scores are updated and they can start a new game either by clicking **Replay** or reset the scores using **Reset score**. After every game the player can analyze the moves by clicking **View Timeline**.

The AI uses a minmax algorithm to find out the best possible move and with some extra tweaks in the algorithm, we have made the AI more competitive and less time consuming.

The View Timeline feature was added to analyze the sequence of moves in a game, and hence would enable the player to play a better game henceforth.

For a Bonus we have added **4 X 4** Board which could be played if the player thinks that **3 X 3** is getting repetitive.

Underneath the **4x4** board, most of the features from the **3 X 3** Board like playing against fellow crew **(PvP)** or against the AI **(PvE)** and **view** **Timeline** are also included.

**The working of the unbeatable AI algorithm**:

When it is the AI’s turn to play, the **bestSPOT()** function is called. This function takes the current status of the board and the current players symbol as its arguments and returns the suitable index (position on the board) for the AI to make its move.

Either one of the two functions are called when the AI executes the bestSPOT() function in accordance to the conditions satisfied.

The **Minimax()** function which is a kind of *backtracking algorithm* is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. The **Minimax()** function is recursively called and once it reaches the terminal nodes it assigns scores of +10 if the AI wins, 0 if it’s a tie and -10 if the human wins. The AI plays as the maximizer, who tries to maximize its score while the human is the minimizer who tries to minimize the score. Every board state has a value associated with it. In a given state if the maximizer **(AI)** has a upper hand then, the score of the board will tend to be some positive value. If the minimizer  **(Human)** has the upper hand in that board state then it will tend to be some negative value.

The other function **checkFinal()** takes the current boards status and the current players symbols as its arguments and checks if at least two spots in any one of the winning combinations is filled by the current player and returns the index of the nearest Winning Position. If the number of moves exceeds 5 and if the index returned is not a null value, then the index of the 3rd spot pertaining to the specific winning combination is returned by the **bestSPOT()** function, otherwise the **Minimax()** function is called.

The function **checkFinal()** was added to make the AI analyze the current status of the board and make a immediate winning move if possible.

By changing the **depthLimit**  variable, which has been initially set to Infinity makes the AI unbeatable but can be changed to 1 or 2 or 3 or 4 or back to Infinity. Each time the Minimax algorithm is called recursively, the depth is incremented and once it is greater than the value of **depthLimit** variable, the **heuristicfunc()** function is called. The **heuristicfunc()** function checks every row, column and diagonal. If one spot in any row, column or diagonal is filled by a human it assigns a score of -10 + depth, if filled by AI it assigns a score of +10 - depth, otherwise assigns a score of 0.

**Hosting the Project:**

The completed web app was hosted in both,

### *AWS s3 service -* <http://marscolepcdecry.s3-website.ap-south-1.amazonaws.com/>

### *Microsoft Azure storage service-* <https://marscolepcdecry.z29.web.core.windows.net/>