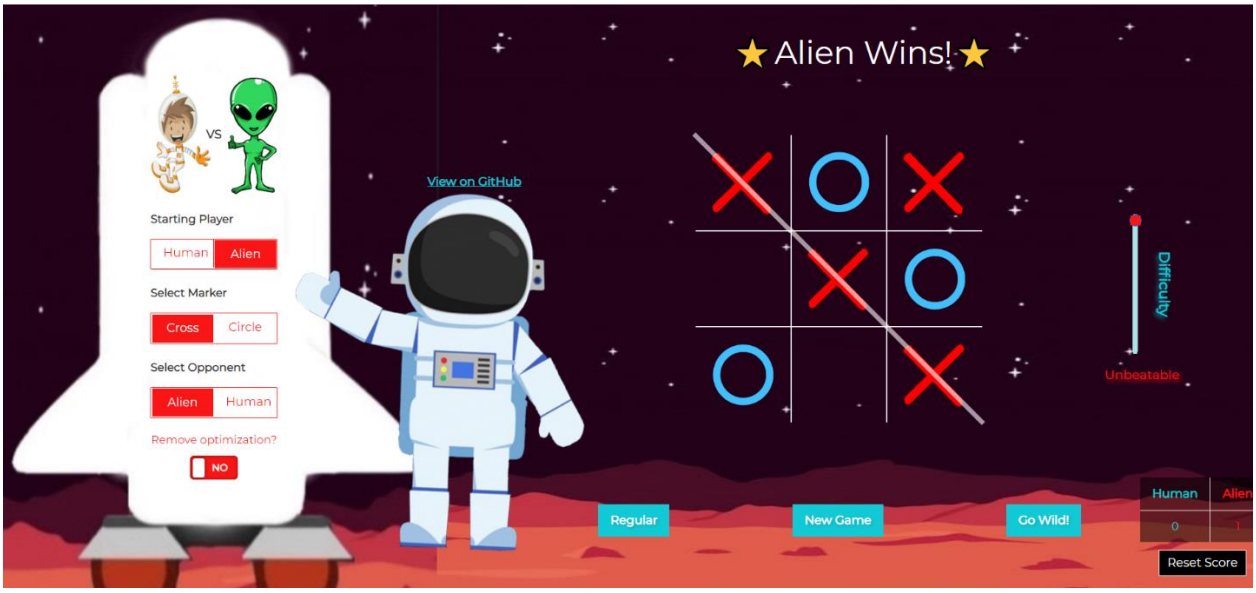
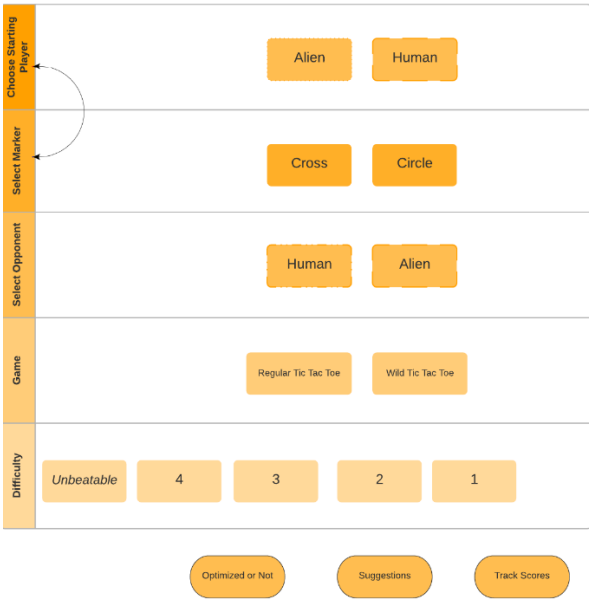
Microsoft-Codess Tic-Tac-Toe!

A 'Mars Colonization theme web application which makes use of the minimax algorithm to create an unbeatable version of the famous Noughts and Crosses game!



Have a Personalised Experience at Mars!

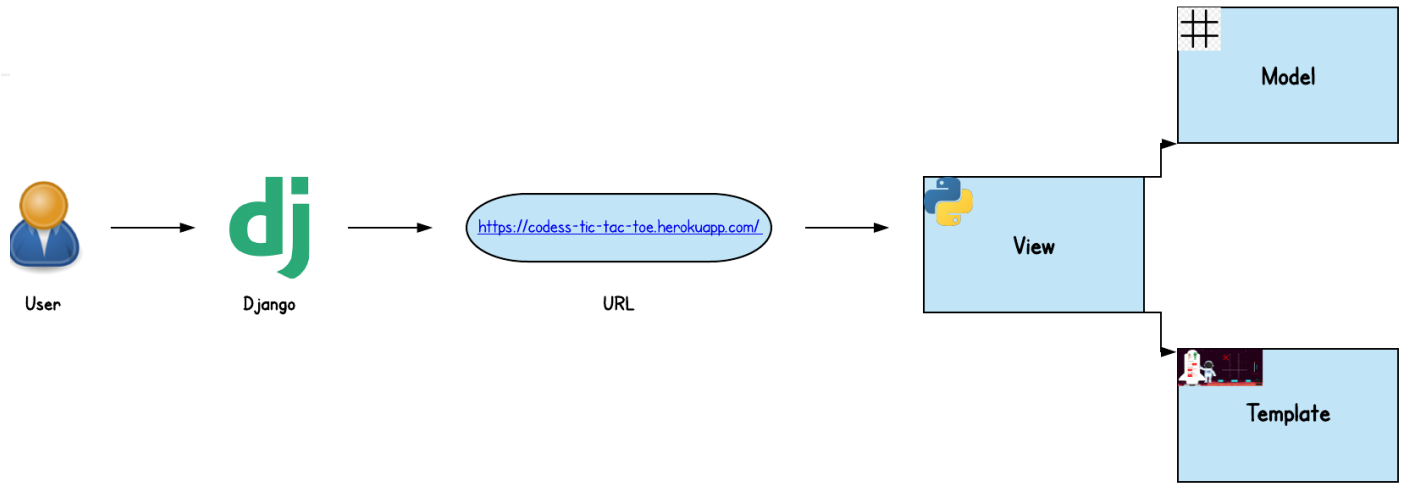
- **Alien**: AI
- **Human**: Opponent
- **Marker**: Select your own marker
- **First Move**: Don't wish to play the first move? No worries!
- **Score Tracker**: Keep a track of all the games played yet
- **Level**: Are you a beginner? Or an expert? You'll not be disappointed
- **Surprise Element**: Reverse Tic-Tac-Toe. Go wild!
- **Optimised Version**: You can check how the game works without optimisation and with optimisation
- **Suggestions**: Playing against another Human? Use suggestions to have an upper hand!



Languages Used:

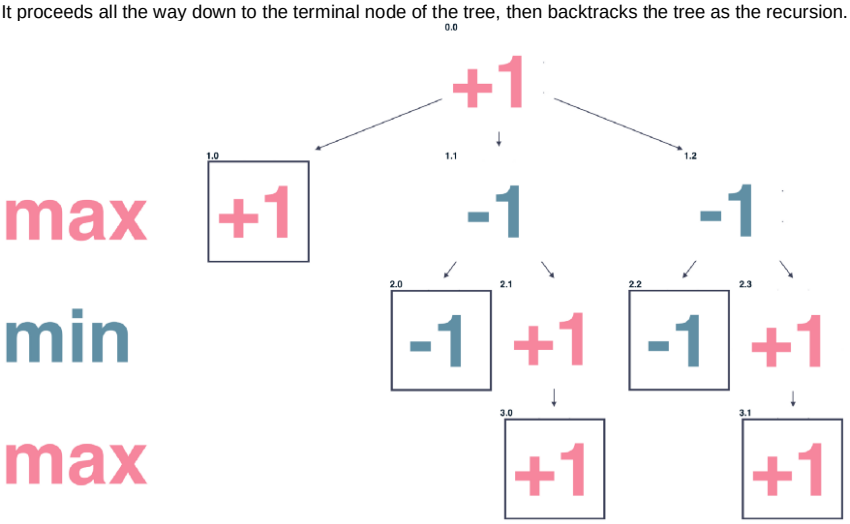
Algorithm- Python
Library used- random
Frontend- HTML, CSS, JavaScript

Model View Template Diagram:



MiniMax Algorithm:

MiniMax algorithm provides an optimal move for the player assuming that the opponent is also playing optimally. In this algorithm two players play the game, one is called MAX, and the other is called MIN. MAX will select the maximized value and MIN will select the minimized value.
It performs a depth-first search algorithm for the exploration of the complete game tree.
It proceeds all the way down to the terminal node of the tree, then backtracks the tree as the recursion.



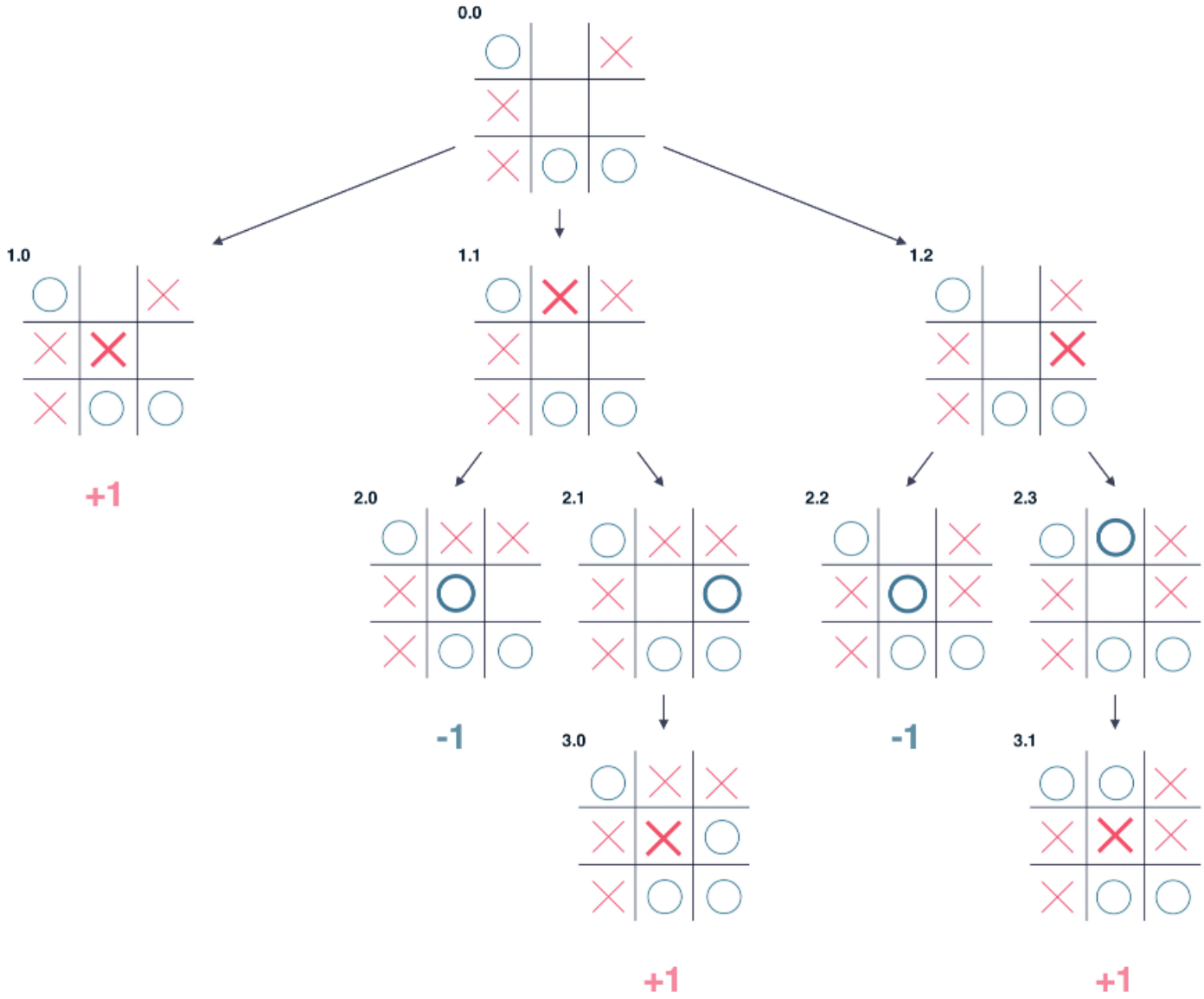
Img Source

Important Code Segments

Addition of Depth

Our Algorithm, makes use of 'maxdepth' to limit the difficulty level of the game against alien.
At each recursive call from one given state 'depth' is incremented by one; The variable maxdepth is responsible to keep check, whether the requested difficulty level has been reached or not and further limits the call to the MiniMax Algo.

```
def minimax(board, depth, isMaximizing,n):
# Check the depth condition
if depth==maxdepth:
#Do not call the further function and directly return from here
    return 0
```



Img Source

Do we have a winner yet?

This part checks whether we are at an ending position or not yet. Terminating position can be either of the three: AI loses, Human loses, Tie. If we are at any of these states, we send back the score mapped in the python dictionary named score. '+1' if AI wins, '-1' if Human wins, '0' if tie (opposite in the case of Wild Tic-Tac-Toe)

If these are not reached then we return None and continue.

```
 # Check the winning condition at every situation
result =checkWinner(board,n)
if result!=None:
    score=scores[result]
    return score
```

Maximizing the Score

In this segment of the code we are trying to Maximize the score for the AI bot.
High score is given to a move which will possibly lead us to a win, or in other words is the most optimal move. We check for the available moves at a particular depth and call the function recursively, alternating between isMaximizing and !isMaximizing.
!isMaximizing is used to check a move for the Human player, where the lowest score is given to the possibility of reaching to a state where Human wins.
bestScore tries to catch the score for the most optimal move.

```
 # Turn of Maximizing player(this player will always try to maximize the score)
# We will check the score of every move and will return the maximum one.
if(isMaximizing):
    bestScore = -1000
        for i in range(n):
            for j in range(n):
                #Is the spot available?
                if board[i][j]=='blank':
                    board[i][j] = ai
                    score = minimax(board,depth+1,False,n)
                    board[i][j]='blank'
                    bestScore = max(score, bestScore)

return bestScore
```

Minimizing the Score

In this segment of the code we are trying to Minimize the score for the opponent.
The catch here is, due to the marking scheme opted, a low score is assigned to the best move (the most optimal move gets the worst score), therefore choosing a low score
move means, we are assuming that Human is very smart and hence, we are not risking our chances of winning by picking a suboptimal move.
We check for the available moves at a particular depth and call the function recursively, alternating between !isMaximizing and isMaximizing.
!isMaximizing is used to check a move for the Human player, where the lowest score is given to the possibility of reaching to a state where Human wins.
bestScore tries to catch the score for the most optimal move.

```
 # Turn of Minimizing player(this player will always try to minimize the score)
# We will check the score of every move and will return the minimum one.
if(!isMaximizing):
    bestScore = 1000
    for i in range(n):
        for j in range(n):
            #Is the spot available?
            if board[i][j]=='blank':
            board[i][j] = human
            score = minimax(board,depth+1,True,n)
            board[i][j]='blank'
            bestScore = min(score, bestScore)

return bestScore
```

Wild Tic-Tac-Toe

In simple words, WTT is reverse tic-tac-toe! Your ultimate goal is to not let your marker appear consecutively either in a row, column or diagonal. Basically, all you have to do is to try to lose, to win!

```
scores={
    ai:-1 if isWild==1 else 1,
    human:+1 if isWild==1 else -1,
    "tie":0
            }
```

The above dictionary simply switches the scores of AI and Human if we are playing wild tic-tac-toe. If AI wins (conventionally) we assign the move a lower score and if Human wins we assign the move a higher score (which is reverse of what we were doing otherwise)

Optimization Technique:

- With the variation of letting the **AI agent play the first chance**, the algorithm was taking an **unusually high time to decide its move** .
    - This is due to the agent building all the possible cases and going till the terminal depth of the board. To eradicate this lag we made use of the game theory of tic-tac-toe.

- Most experienced tic-tac-toe players put the **first marker in a corner** when they get to play first. This gives the opponent the most opportunities to make a mistake.
    - Given the chance to play the first move, we asked our AI agent to pick a **random corner** from a list of all corners on the board.

Similarly, in Wild Tic-Tac-Toe the best move to be played, to give the opponent the most opportunities to make a mistake was the **centre block**.

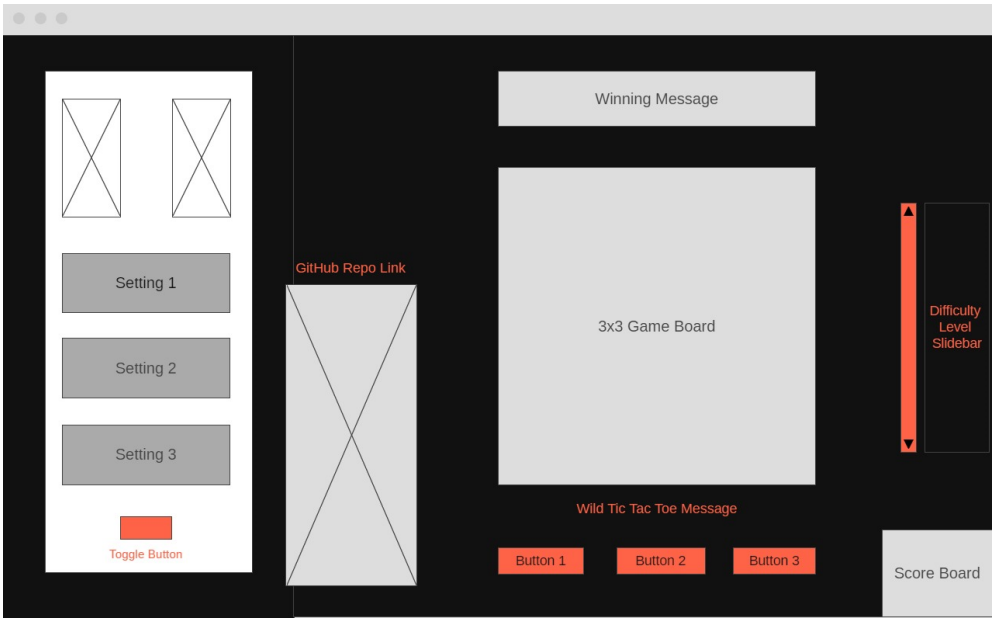
This technique, however small, reduced the initial time of picking a move from **9.01 seconds to 0.0001 seconds!**

```
if(all(ele=='blank' for ele in arr)):
    if(iswild==1):
    #each element on the board is indexed as a number 0-8 (sequentially, 0 being board[0][0].....8 being board[2][2])
    #return central block
        return 4
    else:
    #corner elements are marked by 0,2,6,8 boxes on the board
        return (random.choice([0,2,6,8]))
```

Frontend

The web app has the following components for the front-end functionality:

- **3x3 game board**: This component displays the board on which game is played and handles the functionality of placing markers wherever clicked
- **Sidebar for settings**: This component contains various settings and options like selecting the opponent, starting player and marker. It also displays customizations like suggestions and optimization
- **Difficulty level slider**: This component sets the difficulty level of the game for the user ie the depth for the minimax algorithm
- **Controller buttons**: This component contains buttons which control the type of gameplay like ‘regular’ or ‘wild’ and start a new game
- **Score Board**: This component keeps a track of scores of each player



WireFrame

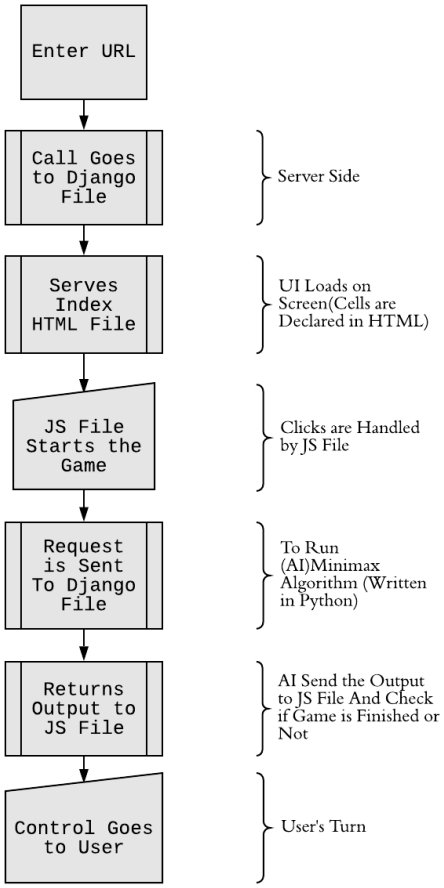
Frameworks Used

Django:

Django is a free and open-source Python web application framework. Django has the advantages of rapid development and clean, pragmatic design. The web app uses Django for the back-end implementation of the minimax algorithm. AJAX has been used to communicate with the server-side written in Python from the client-side on the web app.

Bootstrap UI:

Bootstrap is a free and open-source front-end library for creating websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. It aims to ease the development of dynamic websites and web applications. It has been used mainly for creating rows and columns for proper positioning of elements.



Future Improvements:

- n x n board
- Online Integration
- Alpha Beta Pruning
- 3-D matrix
- Mobile compatibility

Contributors:

Akanksha
Anima
Shreoshi

References:

AI Tic-Tac-Toe
Minimax Algorithm-I
Minimax Algorithm-II
Frontend-I
Frontend-II