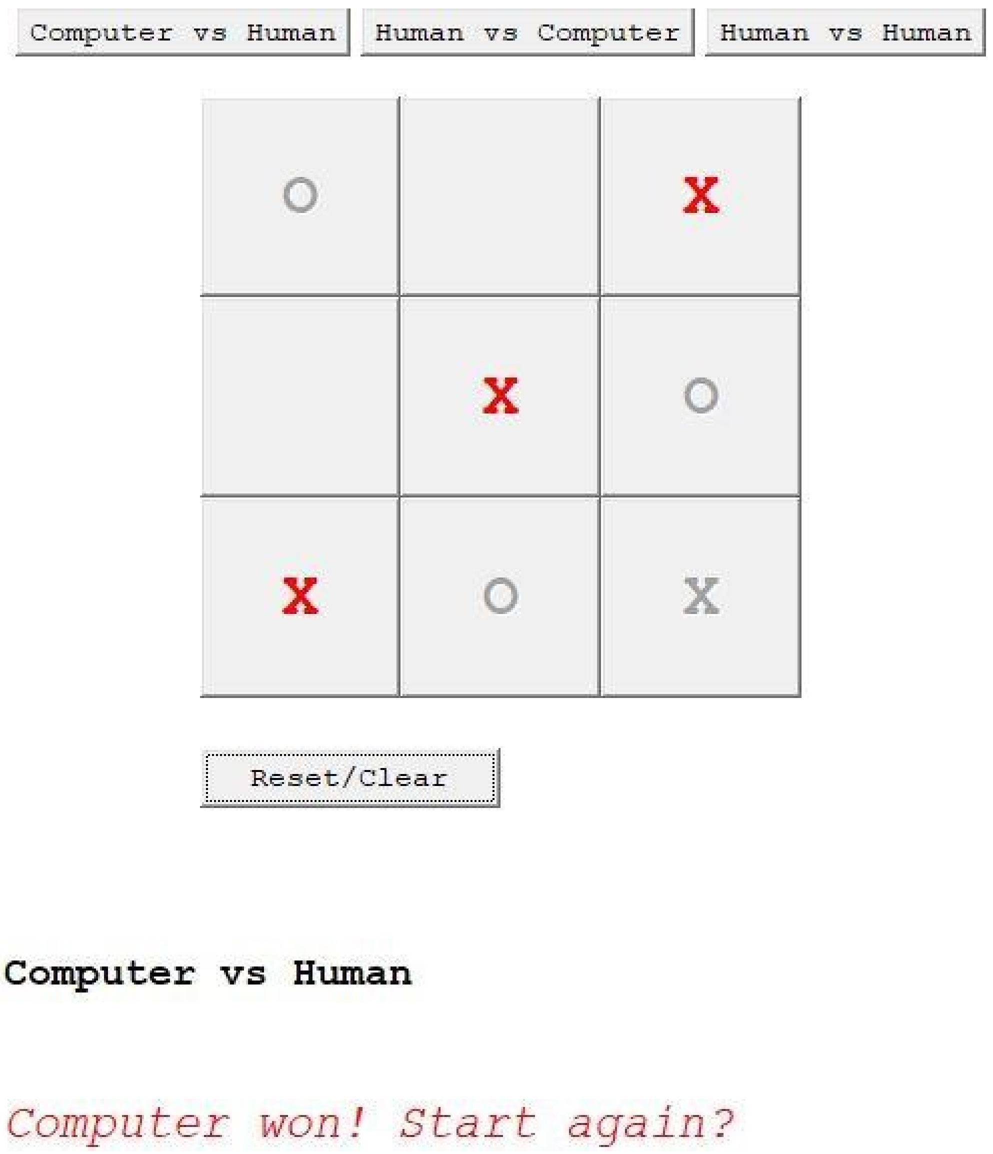
Bhagya Dhome

Unbeatable Tic-Tac-Toe



O		X
	X	O
X	O	X

Computer vs Human

Computer won! Start again?

Try to beat the computer here; and you can also check out the code here.

Im portant Note: https://cs.indstate.edu/~bdhome/tictactoe.html works perfectly assuming the browser is Internet Ex
Java is installed.

What's Tic-Tac-Toe?

Tic-Tac-Toe is a two player's game played in 3 x 3 grid box usually using Xs and Os; a player can win, lose or draw.

Basic Approach (Block & Win):

Corner	Edge	Corner
Edge	Center	Edge
Corner	Edge	Corner

Assuming player A has to win, if A goes first; avoid using edge.

1. Corner:
   - 1.1. If B selects a Corner: A selects any available Corner
   - 1.2. If B selects an Edge: A selects Center
   - 1.3. If B selects Center: A selects opposite Corner
     - 1.3.1 B selects a corner
     - 1.3.2 B selects an edge; this leads to a draw.
2. Center:
   - 2.1. If B selects a Corner; leads to a draw.
   - 2.2. If B selects an Edge: A selects any available Corner

If player B goes first, (and player A has to win);

1. Center: A selects any Corner; leads to a draw.
2. Corner: A selects Center
   - 2.1. B selects a Corner: A selects an Edge; leads to draw.
   - 2.2. B selects an Edge: A selects Corner to block and draw.
3. Edge: A selects a Corner next to X
   - 3.1. A selects center (win) or blocks (draw)

MINIMAX Algorithm

It's a decision-making approach which finds the best next move; and it's being used here.

Two players: Maximizer tries to maximize the chances of winning and Minimizer tries to minimize the chances of Maximizer's winning.

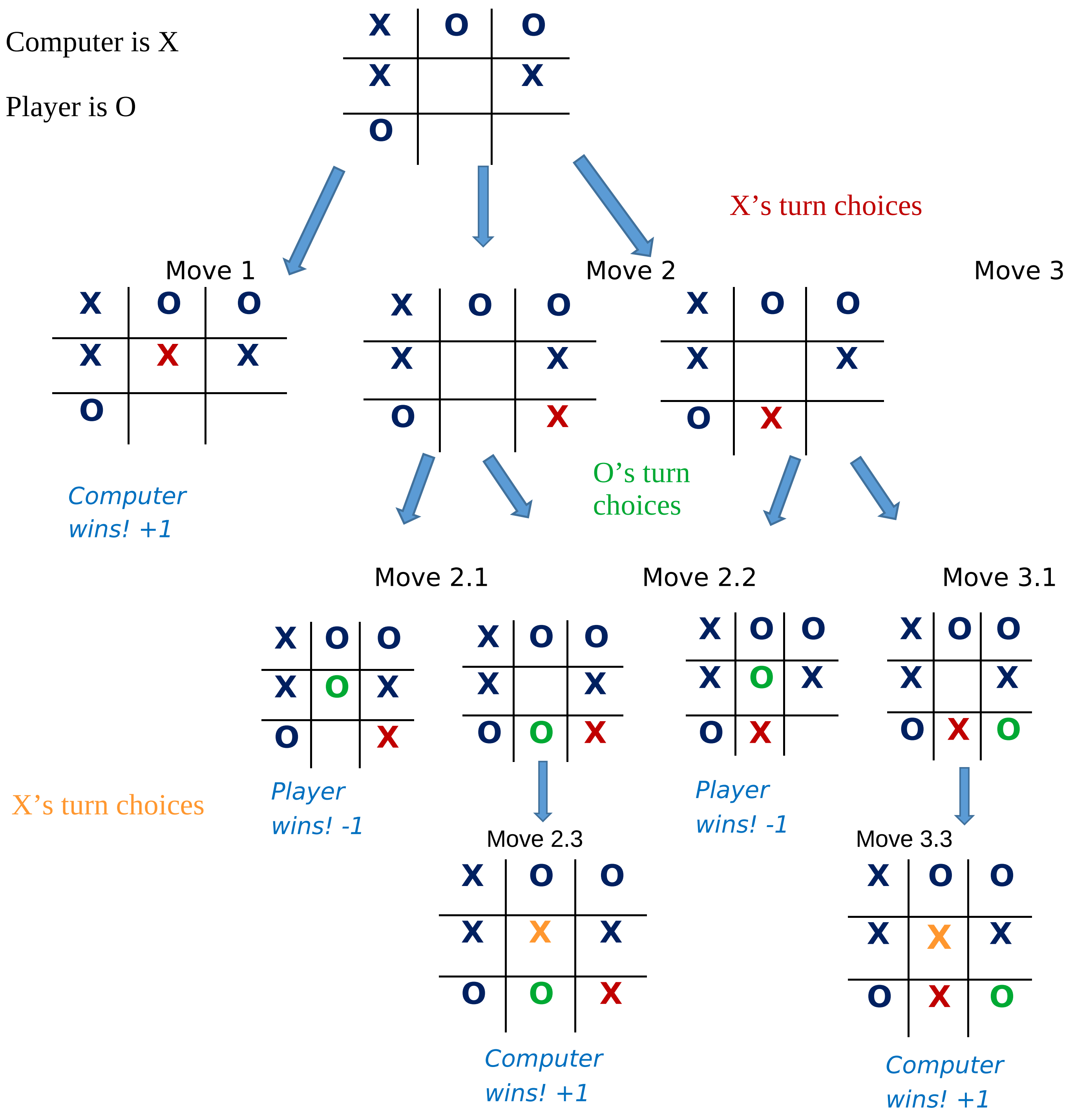
According to Wikipedia,*if player A can win in one move, their best move is that winning move. If player B knows that one move will lead to the situation where player A can win in one move, while another move will lead to the situation where player A can, at best, draw, then player B's best move is the one leading to a draw. Late in the game, it's easy to see what the "best" move is. The Minimax algorithm helps find the best move, by working backwards from the end of the game. At each step it assumes that player A is trying to* ***maximize*** *the chances of A winning, while on the next turn player B is trying to* ***minimize*** *the chances of A winning (i.e., to maximize B's own chances of winning).*

In other words and with respect to the implementation of my project, when it's computer's turn, with every move it adds +1 if win, -1 if lose and 0 if draw, and selects the move with maximum score.

```
function minimax(count, depth, current_player)

    if computer won
        return +1;
    else if player won
        return -1;
    else if draw
        return 0;

    if current_player is computer  //Max block
        maxValue := −∞
        for each move left
            score := minimax(count, depth+1, player)
            maxValue := max(maxValue, score)
        return maxValue

    else  //player's turn; min block
        minValue := ∞
        for each move left
            score := minimax(count, depth+1, computer)
            minValue := min(minValue, score)
        return minValue
```

But, how does the algorithm work exactly?

Given the following **state of game** (first grid) and assuming its computer's turn (i.e. in this case, Computer is X), the best move according to the algorithm will be the first move (Move 1).



Let’s understand it further that **why** Move 1 is the best move;

> Given the state, moves 1, 2, and 3 are generated and function Minimax is called recursively further on those moves.
>
> As move 1 leads to computer’s win i.e. end of the game, giving +1(maximum score) as its score, on the other hand, move 2 and 3 generate 2.1, 2.2, 3.1 and 3.3 moves respectively and recursively call Minimax.
>
> Move 2.1 adds -1 to move 2’s score and move 3.1 adds -1 to move 3’s score as the opposite wins(minimum score).
>
> Moves 2.2 and 3.2 generate the last possible moves 2.3 and 3.3 respectively which adds +1 to moves 2 and 3 respectively (as computer wins in both cases).
>
> And since 2.1 and 2.2 are opposite player’s turn, it selects minimum score from (-1, +1) and same goes for 3.1 and 3.2.

So, ***the ultimate scores for 1, 2, and 3 are +1, -1, and -1 respectively; therefore, the best move for the given state of game is move 1.***

In simple words, a list of every possible moves and the ultimate score is created given a state of game like above; and the move with the maximum ultimate score is selected.

Functions **callMiniMax(), getMinMax()** and **getPosition()** are related to Minimax functionality of the application here.

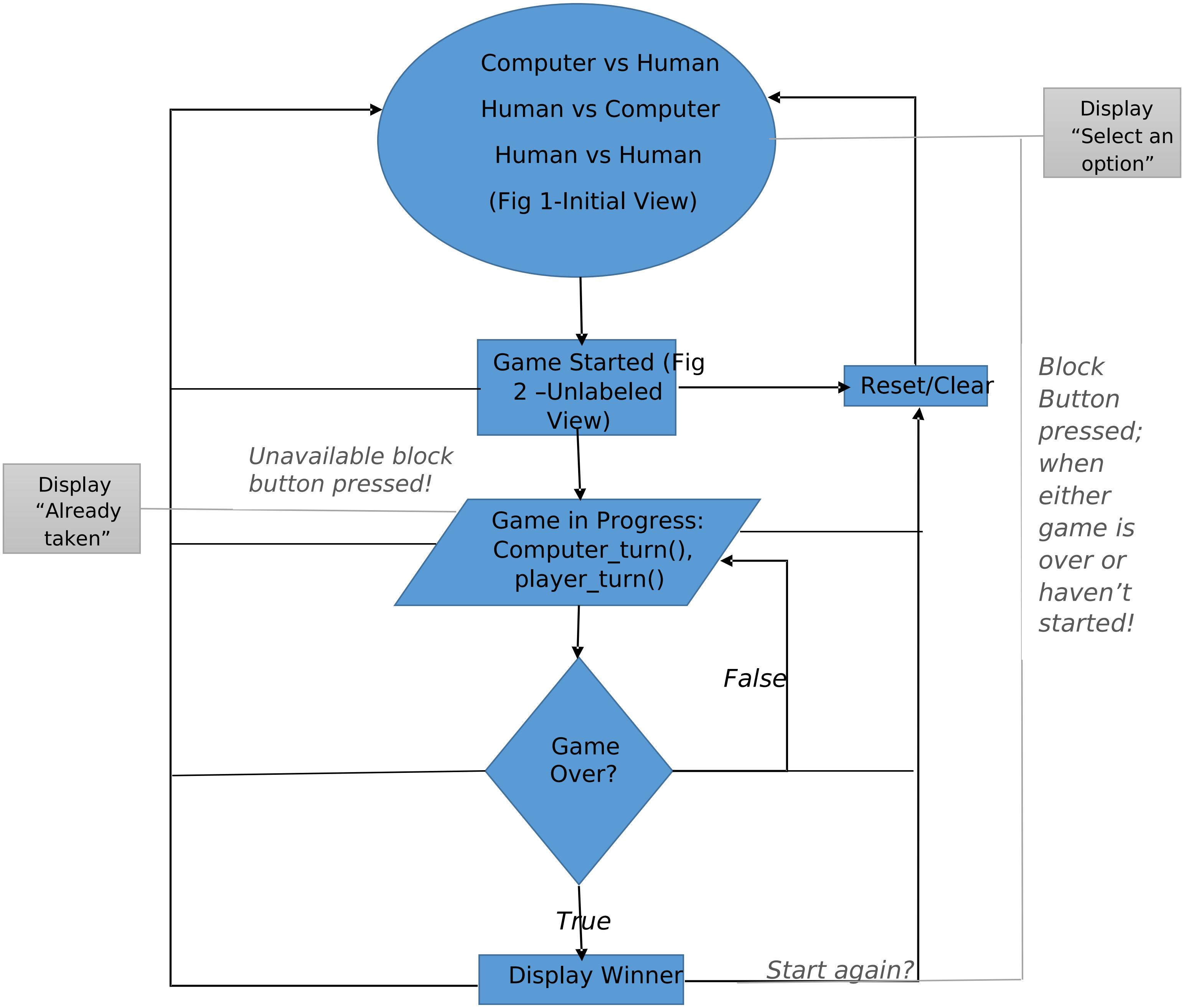
Also, this approach considers best move as the move which is unbeatable and while doing so, it might ignore current win for a definite future win. So, I have added **find_winning_position()** function for win-first approach.

Can it be improved further?

Yes, it can further be improved by adding **Alpha-Beta Pruning** to Minimax.

This approach is exactly same as Minimax with an exception of two extra values **alpha** and **beta**.

Flow Chart of the implementation:


Computer vs Human
Human vs Computer
Human vs Human
(Fig 1-Initial View)
Display "Select an option"
Game Started (Fig 2 –Unlabeled View)
Reset/Clear
Block Button pressed; when either game is over or haven't started!
Unavailable block button pressed!
Display "Already taken"
Game in Progress: Computer_turn(), player_turn()
False
Game Over?
True
Display Winner
Start again?

Please refer https://en.wikipedia.org/wiki/Flowchart#Software for flow chart and symbol standards.

Why Java Applet?

This application is implemented using Java and Applet which extends java.applet.Applet class; Applets are used to add ***interactive components*** to the web page.

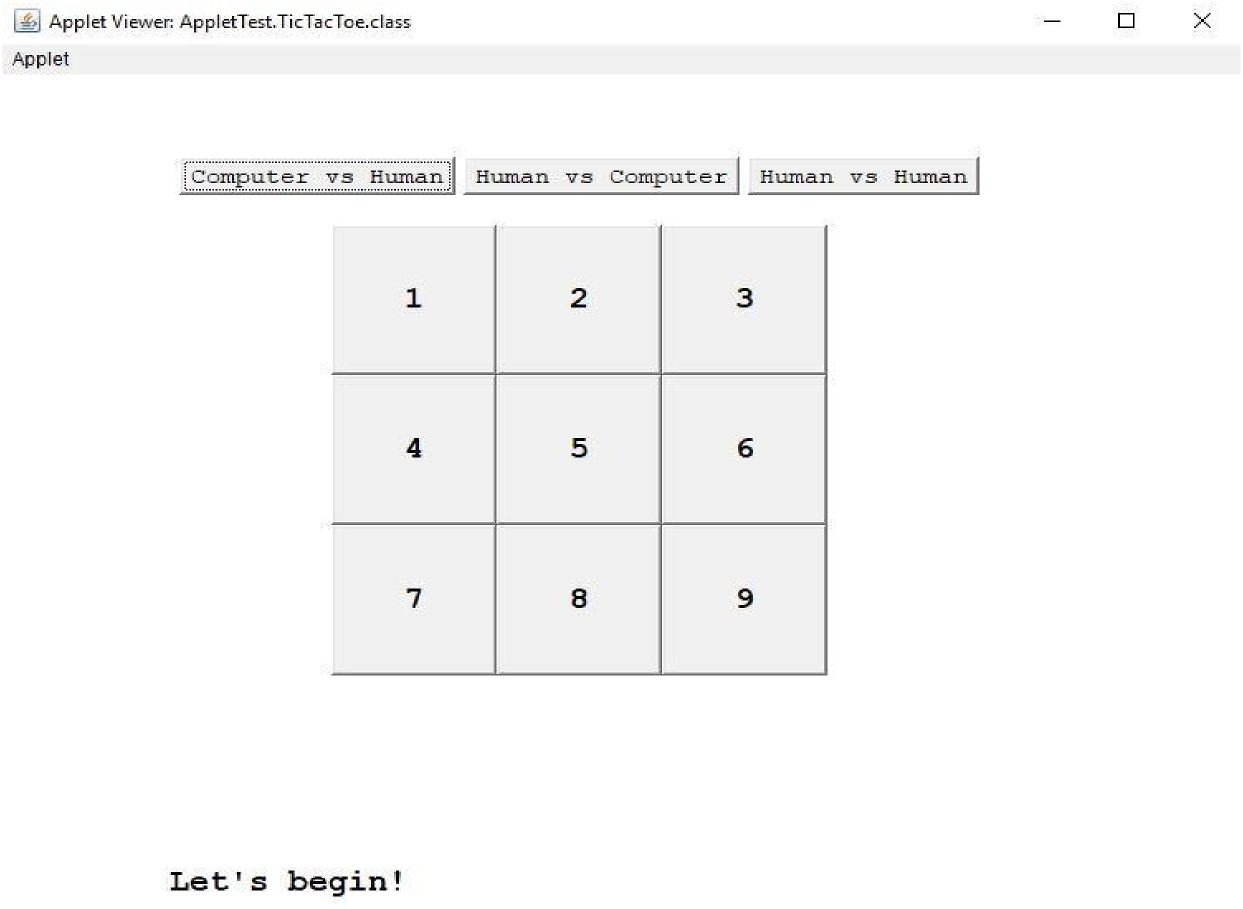
The main difference between a java program and a java applet application is that in a java applet, main() method is not invoked, instead init() is called.

It is embedded to the HTML page using **<applet>** tag.

The following are methods that use applet functions in the code:

```
void winningBlocks() //Highlights the winning blocks
void clearWinningBlocks() //Clear winning blocks
void updateGridBlock(int button_no) //Update Computer's move
void setComponents() //Arrange or set swing components wrt applet window
void init() //When applet
void displayPanel() //Display the X-O grid
void resetPanel() //Reset the Grid boxes label to numbers
void setButtonLabels() //Change the Grid Boxes label to NULL" "
void actionPerformed(ActionEvent e) //Called when each time a button is pressed
```

How does the implementation look and work?



Let's begin!

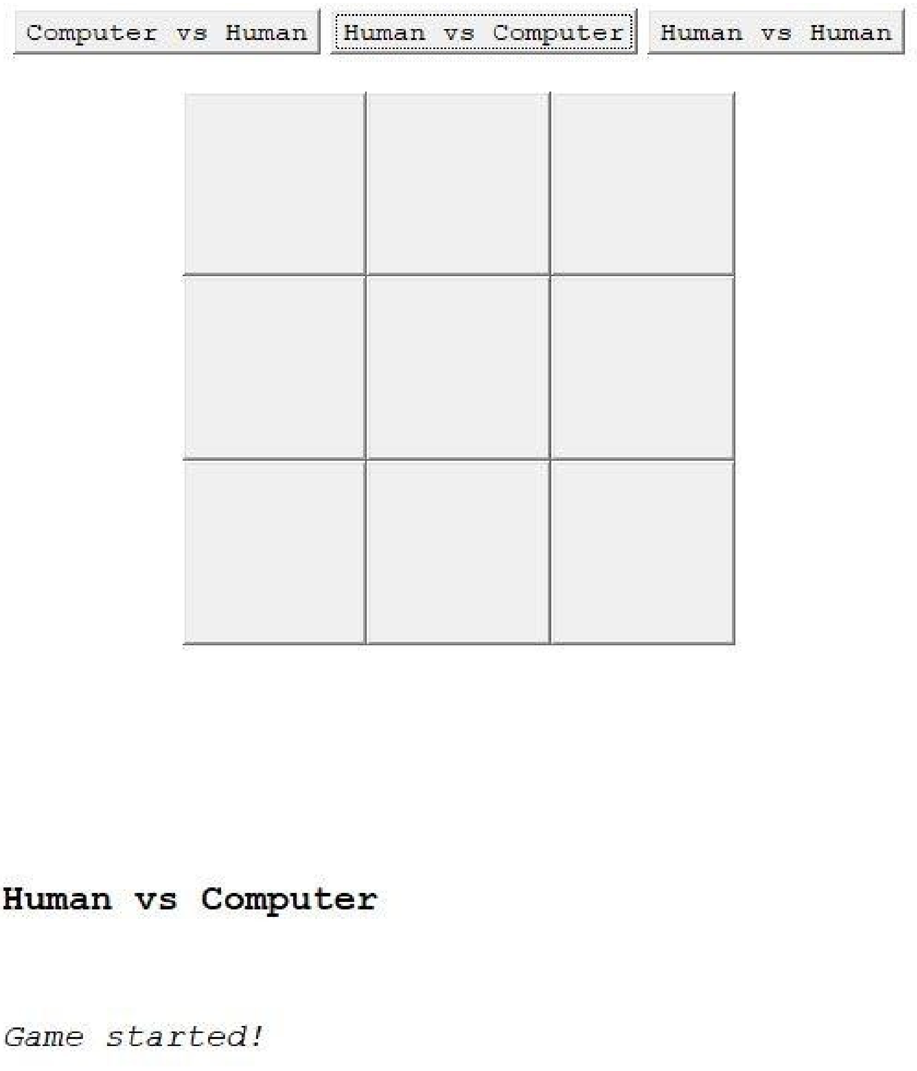
FIGURE 1: INITIAL VIEW

Game doesn’t start unless when one of the three buttons is pressed. (Refer figure 1)

If a numbered button is pressed, it displays a message asking to select one of the top options.

In Figure 2, the human vs computer option is select, and one of the blank blocks in grids are pressed, the following sequence of functions is called;

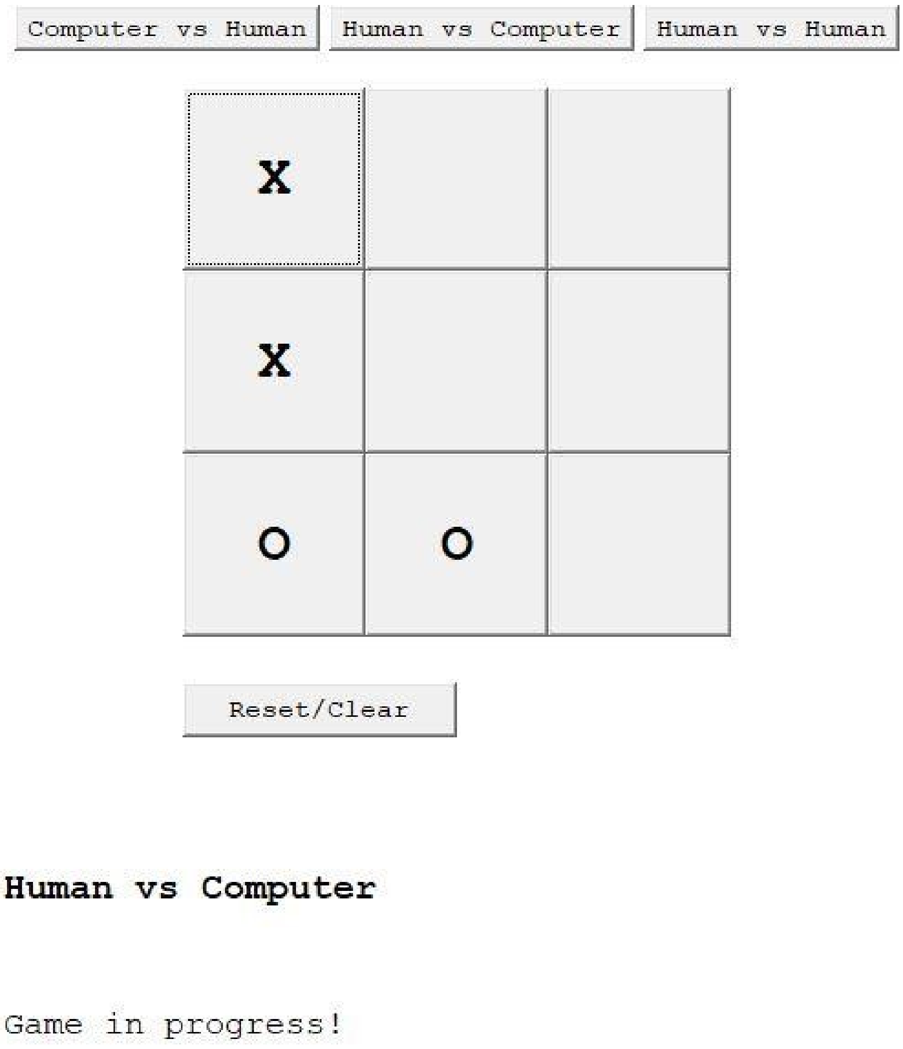
actionPerformed(ActionEvent e)
setButtonLabels();
clearWinningBlocks();
initialize_game();
player_turn(block_no)
computer_turn();



Human vs Computer

Game started!

FIGURE 2: UNLABELED VIEW



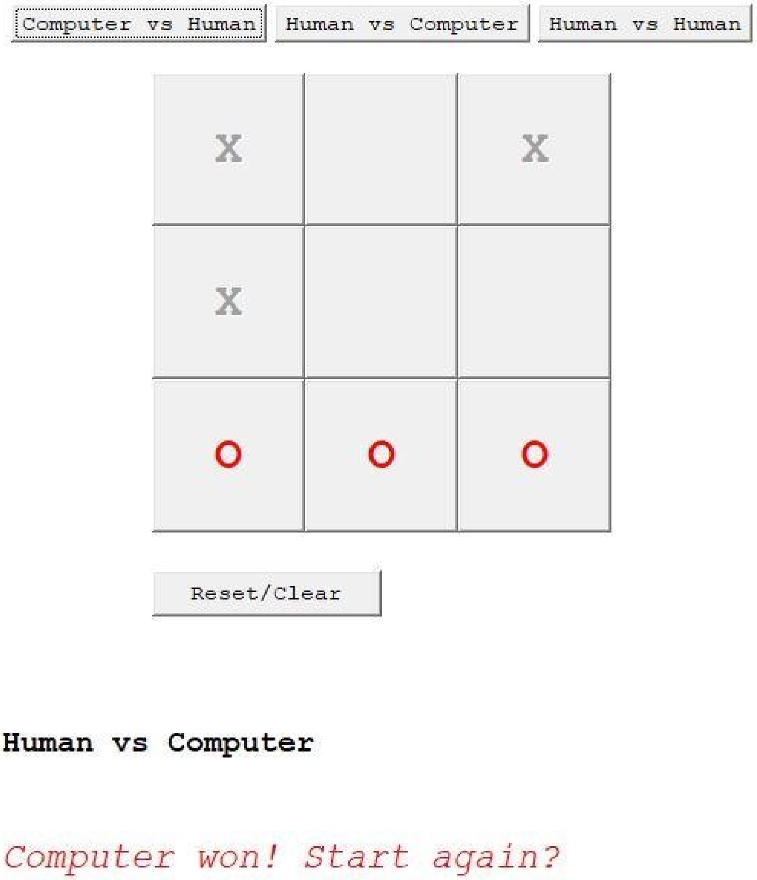
X		
X		
O	O	

Human vs Computer

Game in progress!

FIGURE 3: GAME IN PROGRESS

Funtion computer_turn() calls the minimax algorithms callMiniMax(), getMinMax() and getPosition(); waits for human’s turn.



X		X
X		
O	O	O

Human vs Computer

Computer won! Start again?

FIGURE 4 WINNING VIEW

Reset/Clear button appears after Initial View, and when pressed, takes you back to the Initial View(Figure 1); other options when pressed (anytime) leads to Unlabeled View(Figure 2) i.e start of the game.

When an unavailable button is pressed; it displays “Already taken!” message.

When the game is over, the buttons(except winning button which are high-lightened) are disabled using setEnable(false);

Other functions in the application:

```
void initialize_game() //Initialize all the variables and arrays
void start_game() //USED WHEN ON CONSOLE; instead of Applet
int find_winning_position() //Computer does not wait for a best move if there is a winning move
int check_win(int turn) //Keep track of the end game
void increment_rounds(int block_no, int current_player) //Gotta keep count of moves
void player_turn(int block_no) //Human tries
void computer_turn() //Computer destroys
int check_draw () //Checks for draw
void display() //Displays on console
```

Important **NOTE:**

https://cs.indstate.edu/~bdhome/tictactoe.html works perfectly assuming the browser is Internet Explorer and Java is installed.

If it still does not work; either lower your java security settings or add the above address as it is to the exception list.

You can read further on this here - https://www.java.com/en/download/help/java_blocked.xml

REFERENCES:

[1] http://www.eng.uerj.br/~fariasol/disciplinas/Topicos_B/AGEN TS/books/Stuart%20Russell,%20Peter%20Norvig-Artificial%20Intelligence_%20A%20Modern%20Approach-Prentice%20Hall%20(2002)-2nd-ed.pdf

[2] https://www.neverstopbuilding.com/blog/2013/12/13/tic-tac-toe-understanding-the-minimax-algorithm13

[3] https://medium.com/@victorcatalintorac/tic-tac-toe-with-ai-the-ultimate-beginner-guide-part-4-142b6ea534df

[4] https://web.eecs.umich.edu/~akamil/teaching/sp03/minimax.pdf

[5] https://en.wikipedia.org/wiki/Minimax#Pseudocode

[6] https://www.javatpoint.com/java-applet

[7] https://docs.oracle.com/javase/tutorial/uiswing/layout/flow.html

[8] https://classes.soe.ucsc.edu/cmps112/Winter16/presentations/ttt.pdf

[9] https://www.hackerearth.com/blog/artificial-intelligence/minimax-algorithm-alpha-beta-pruning/

[10] https://www.cs.cornell.edu/courses/cs312/2002sp/lectures/rec21.htm

[11] http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html

[12] https://athena.ecs.csus.edu/~gordonvs/Beijing/Minimax.pdf

[13] https://www.researchgate.net/figure/MiniMax-Algorithm-Pseduo-Code-In-Fig-3-there-is-a-pseudo-code-for-NegaMax-algorithm_fig2_262672371

[14] https://en.wikipedia.org/wiki/Flowchart#Software