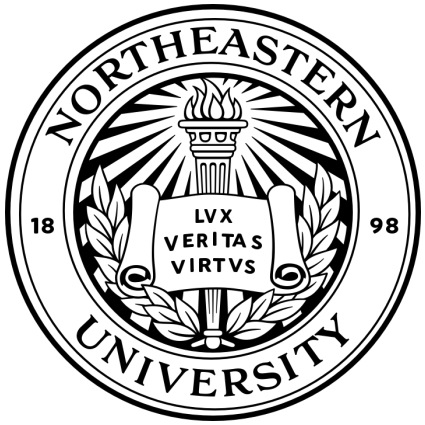
THE MENACE

PROJECT REPORT



INFO-6205
Program Structures and Algorithms

INSTRUCTOR
Prof. Robin Hillyard

1. Jayesh Kumar Khattar 001568947
2. Mayank Yadav 002193976
3. Sangram Vuppala 002958963

AIM

The aim of the project is to implement menace without matchboxes and by using hash table instead. The keys in the hash-table represents each type of the match boxes which is also the different states of the board.

Introduction

Tic-tac-toe, also known as nougats and crosses or Xs and Os, is a two-person paper and pencil game in which each player alternates marking squares in a three-by-three grid with an X or an O. The winner is the player who successfully places three of their markers in a horizontal, vertical, or diagonal row. It's a solved game with a forced draw if both players play their best.

Donald Michie created MENACE (Machine Educable Nougats And Crosses Engine) in 1961, a machine that could learn to play Nougats and Crosses better. MENACE was made from 304 matchboxes because computers were not readily available at the time.
MENACE starts with four beads of each color in the first move box, three beads in the third move box, two beads in the fifth move box, and one bead in the final move box. When a bead is removed from each box after a loss, it signifies that subsequent movements are more strongly discouraged. This allows MENACE to learn faster because the latter moves are more likely to have resulted in a loss.

APPROACH

The approach we took in this project is first to train the menace.
During the training, we train the menace by playing a large set of rounds.

- The first player “X” is random bot which picks its moves randomly from the list of available moves.
- The second player “O” our menace bot plays the move using the minimax algorithm.
- We are use 4 hash tables, one for every move which keeps all the possible states as key.
- As the simulation runs, we give certain reward and penalty for each round the menace wins or loses. If the match draws, we neither reward nor penalize the menace.
- After we train the bot for 500 rounds, our 4 hash tables store the keys which represent the moves taken and their score value.

The beads used to reward or punish the Menace are:

- Alpha = 0
- Beta = +1
- Gamma = -1
- Delta = +0.5

After this training, our menace is ready to be challenged. You can play with the menace after. The menace in this case, chooses its step based on hash table values, identifies which step to take.

PROGRAM

Below are the program details -

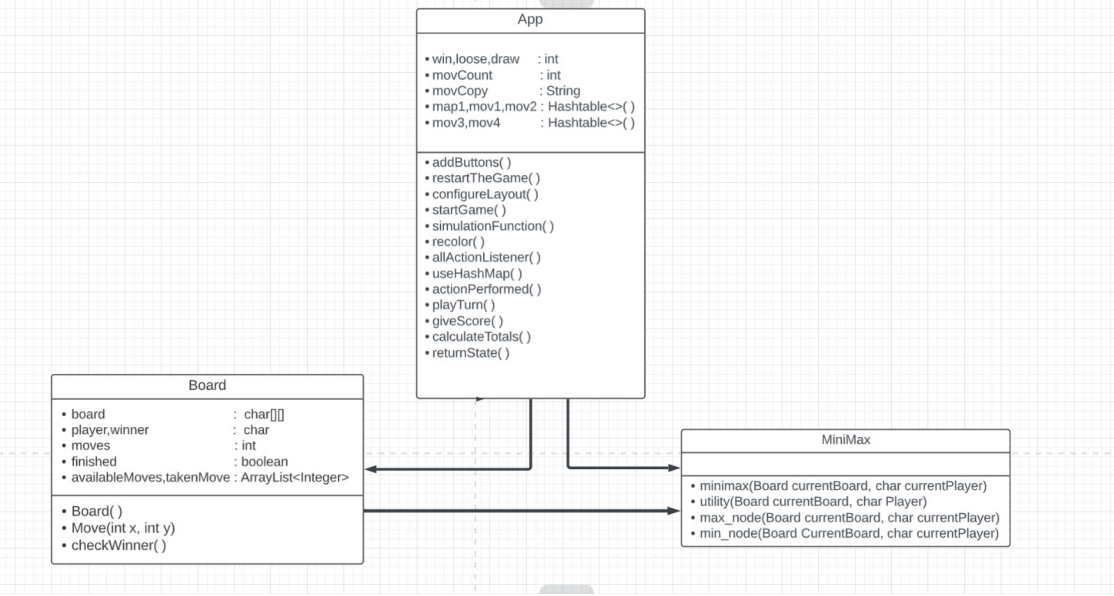
DATA STRUCTURES

We are using the following data structures for the purpose –

- Hash Tables – We are using hash tables to store the state of board and their score against the training rounds.
- 2D Char Array - We are using a 2D array to store the state of the game while training as well as playing the game.

CLASSES

We have the following classes and methods in our project –



App.java

- addButtons: Adding and enabling buttons to play TicTacToe.
- restartTheGame: Reinitializing the button panel after every training iteration.
- configureLayout: initializing GUI components in a constructor.
- recolor: Redrawing TicTacToe after every move.
- allActionListener: Function for ActionEvent on a click.
- startGame: Initializing board for player mode
- simulationFunction: Training menace against the random move.
- actionPerformed: Capturing player move and incrementing move
- playTurn: Marking player move on Board and checking if player wins.
- useHashMap: Selecting maxNode move from trained HashMap.
- giveScore: Scoring previous moves based on the outcome of game.
- calculateTotals: Calculating total wins, loses and draws during simulation.
- returnState: Returning next state from Hash Table based on current state.

Minimax.java

- Minimax: Constructor of class; Initializing class variables.
- Utility: Checking winner in the next move.
- max_node: Returning move to maximize the value at that node.
- min_node: Returning move to minimize the value at that node.

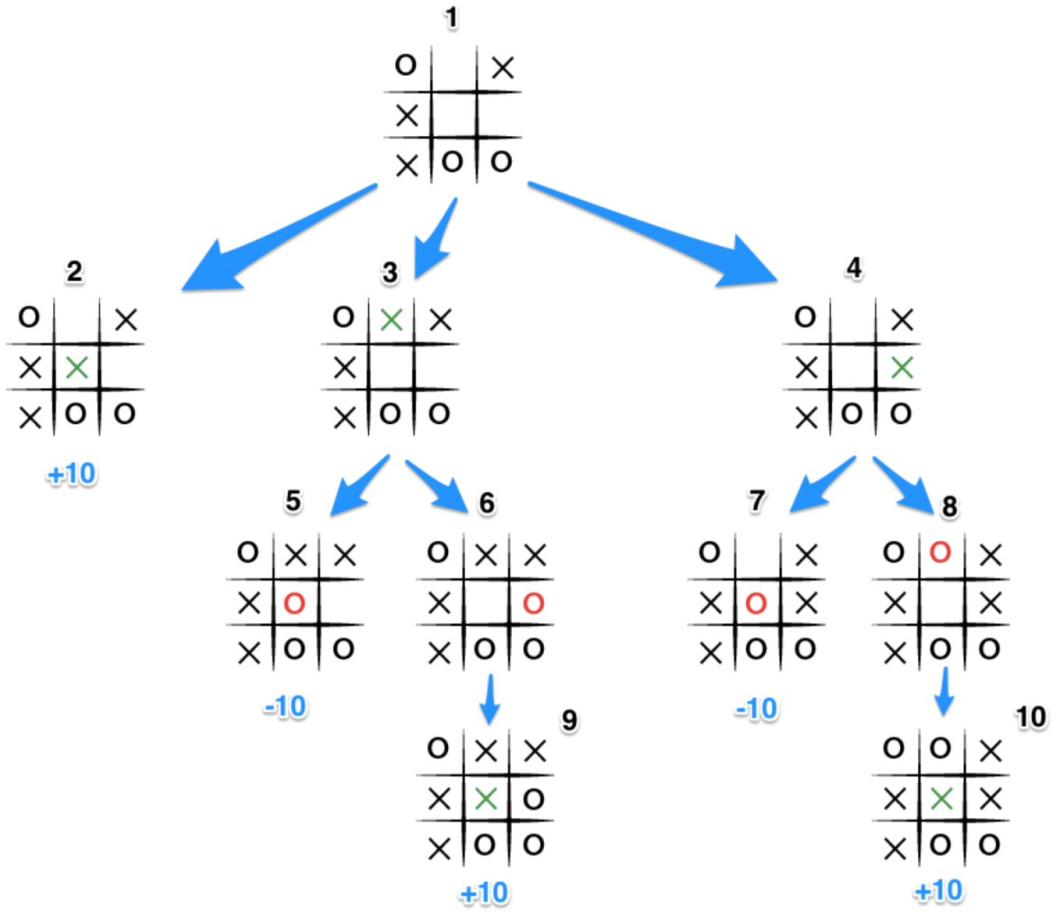
Board.java

- Board: Constructor of class; Initializing class variables.
- printBoard: Printing the current state of Board.
- setPlayer: Setting player for next move.
- Move: Marking move on the TicTacToe board.
- checkWinner: Checking winner by checking conditions on the Board.
- printWinner: Printing the winner of the current state.
- copyThis: Returning the copy of the current state.

ALGORITHM

Minimax Algorithm

Here is a image which represents minimax algorithm.



- It's X's turn in state 1. X generates the states 2, 3, and 4 and calls minimax on those states.
- State 2 pushes the score of +10 to state 1's score list, because the game is in an end state.

- State 3 and 4 are not in end states, so 3 generates states 5 and 6 and calls minimax on them, while state 4 generates states 7 and 8 and calls minimax on them.
- State 5 pushes a score of -10 onto state 3's score list, while the same happens for state 7 which pushes a score of -10 onto state 4's score list.
- State 6 and 8 generate the only available moves, which are end states, and so both add the score of +10 to the move lists of states 3 and 4.
- Because it is O's turn in both state 3 and 4, O will seek to find the minimum score, and given the choice between -10 and +10, both states 3 and 4 will yield -10.
- Finally, the score list for states 2, 3, and 4 are populated with +10, -10 and -10 respectively, and state 1 seeking to maximize the score will chose the winning move with score +10, state 2.

Below is the algorithm to train the menace and store it steps in hash table -

```
//List of available moves
availableMove = list of available moves {0 to 8}

//4 Maps to store values for each step
HashTable <String, Integer>  mov1Map  //For step 1 of menace
HashTable <String, Integer>  mov2Map  //For step 2 of menace
HashTable <String, Integer>  mov3Map  //For step 3 of menace
HashTable <String, Integer>  mov4Map  //For step 4 of menace

For loop to train the menace
For i = 0 to 500:
        move= 0
        Map<Integer, String> boardStepMap;

        While (!game.finished)
                //Move 1 – Play move by random bot
                moveRandomMenace();
                availableMove.remove();
                printBoard();
                if(game.finished)
                        game.winner = "X won"
                //Move 2 – Play move by minimax bot
                moveAIMinimax();
                availableMove.remove();
                printBoard();
                if(game.finished)
                        game.winner = "O won"
                boardStepMap.put(move;boardState);
```

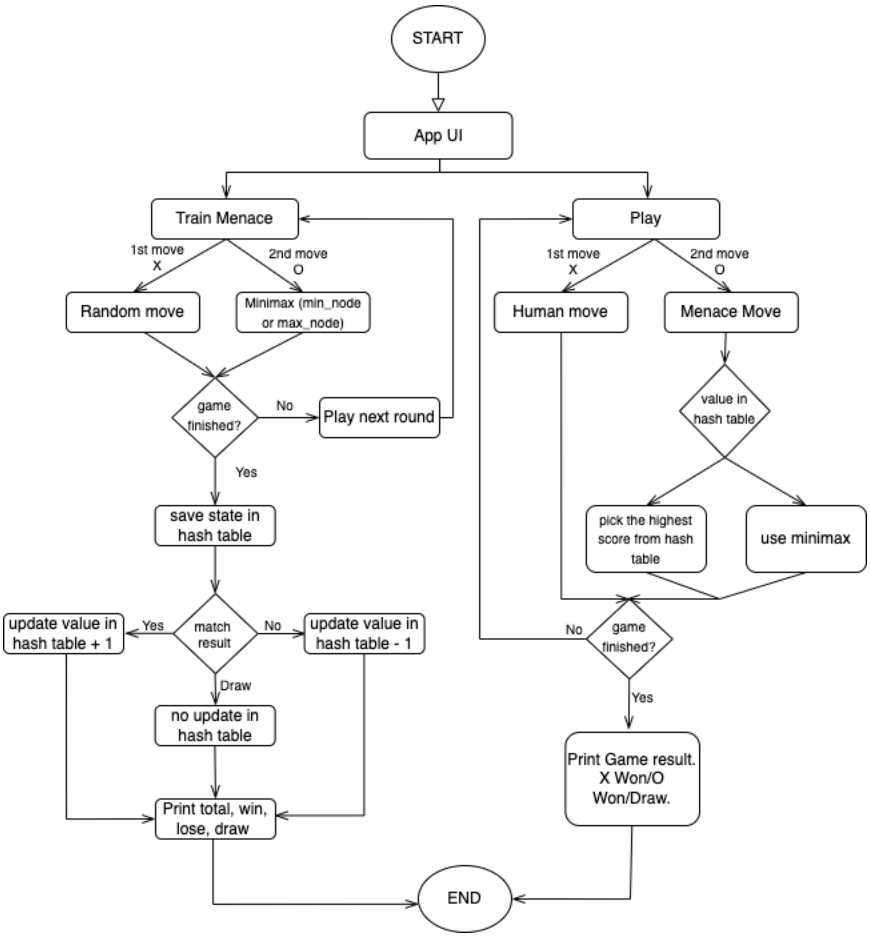
```
        move++

    //Update the four maps with alpha, beta and delta values for each step
k in boardStepMap map
        for k = 1, update mov1Map
        for k = 2, update mov2Map
        for k = 3, update mov3Map
        for k = 4, update mov4Map
        if menace won, add or update mov[k]map with +1,
        if lost, add or update mov[k]map with -1
        if draw, no update

print(total count);
print(win count);
print(loose count);
print(draw count);
```

INVARIANTS

- Menace plays second. It always moves after the random bot or human moves.
- Menace cannot play more than four states in a game.
- Rewarding and punishing the menace with +1 in case of winning and losing.

FLOW CHARTS

Here below is the flow chart for our application-

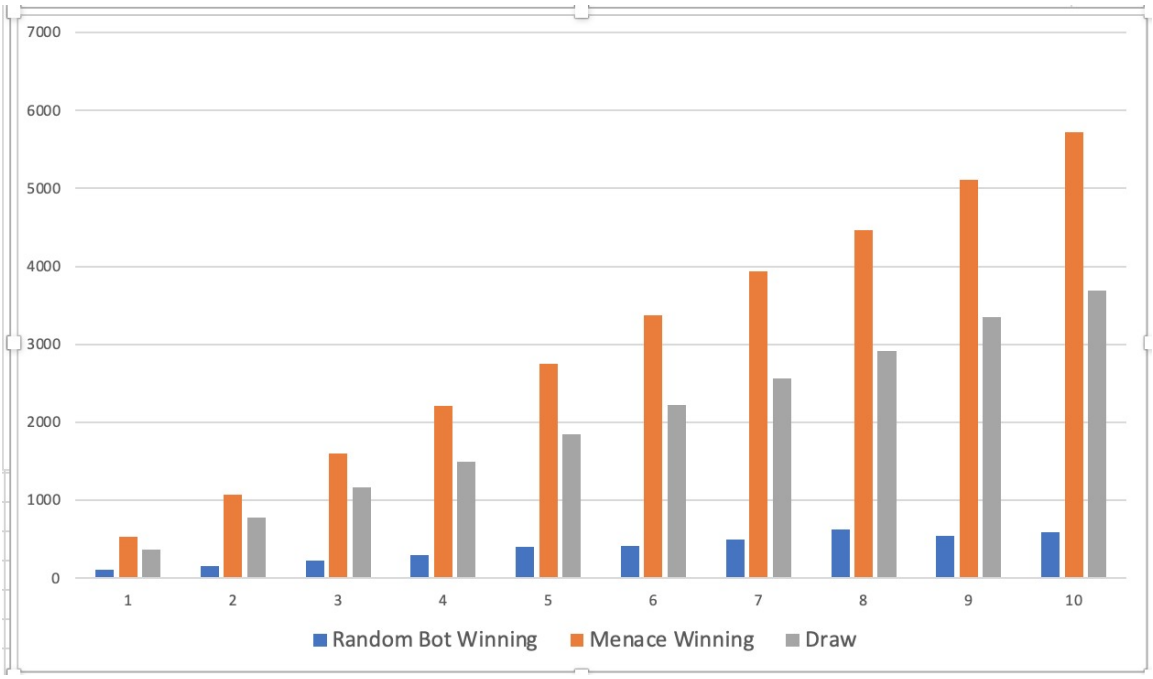


OBSERVATIONS AND GRAPHICAL ANALYSIS

From the below observations we can see that the menace got trained almost at the same rate every time with different executions –

Random Bot Winning	Menace Winning	Draw	Total	Menace Win Training Percentage	Draw Percentage	Random Bot percentage
107	527	366	1000	52.7	36.6	10.7
155	1065	780	2000	53.25	39	7.75
230	1603	1167	3000	53.43	38.9	7.67
301	2209	1490	4000	55.23	37.25	7.53
401	2749	1850	5000	54.98	37	8.02
410	3370	2220	6000	56.17	37	6.83
497	3936	2567	7000	56.23	36.67	7.1
624	4466	2910	8000	55.83	36.38	7.8
544	5110	3346	9000	56.78	37.18	6.04
589	5718	3693	10000	57.18	36.93	5.89

It could be observed from the graphical representation that the percentage of games won by menace and number of draws increases with the number of games played.



So, we can clearly see that the menace is trying to learn against all possible states and moves it can make. Therefore, it can win with more and more training of the games.

RESULTS AND MATHEMATICAL ANALYSIS

- From the above graph, It is clear that Menace is trying to learn gradually against the random bot.

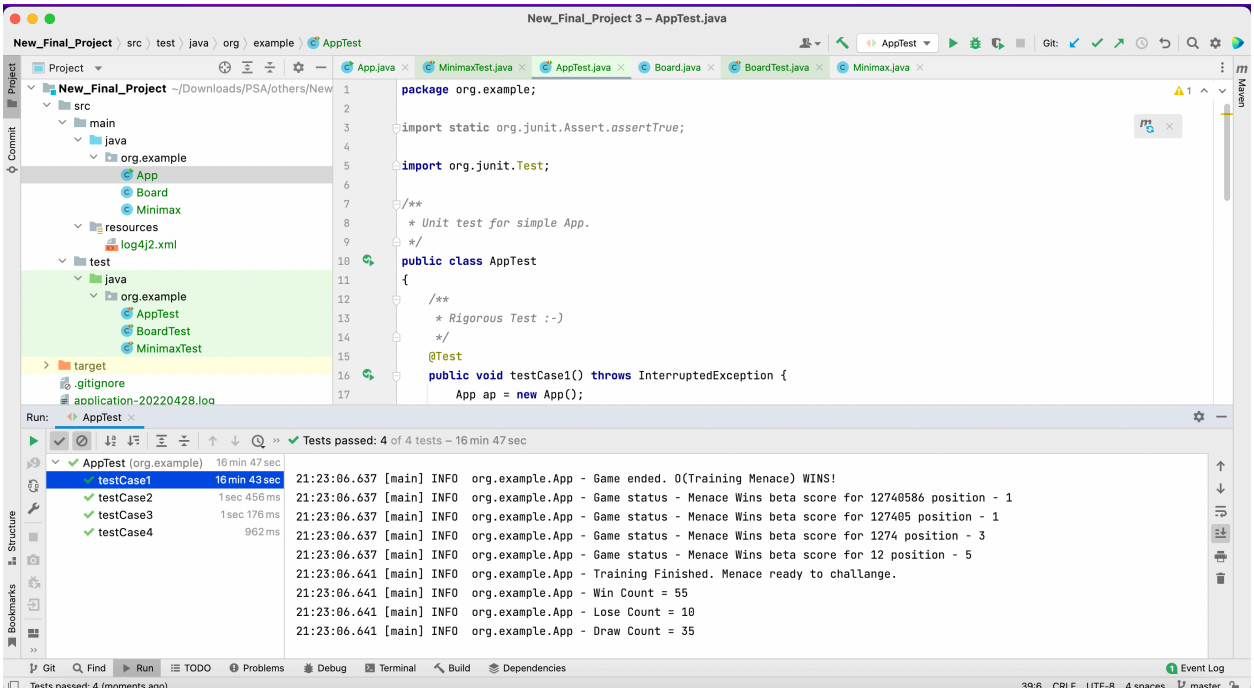
- The Random Bot winning percentage started dropping gradually. At the starting of the game, it won ~11% of the games but at the end of our iterations it won only ~6% of the games.

Random Bot Winning	Menace Winning	Draw	Total	Menace Win Training Percentage	Draw Percentage	Random Bot percentage
107	527	366	1000	52.7	36.6	10.7
155	1065	780	2000	53.25	39	7.75
230	1603	1167	3000	53.43	38.9	7.67
301	2209	1490	4000	55.23	37.25	7.53
401	2749	1850	5000	54.98	37	8.02
410	3370	2220	6000	56.17	37	6.83
497	3936	2567	7000	56.23	36.67	7.1
624	4466	2910	8000	55.83	36.38	7.8
544	5110	3346	9000	56.78	37.18	6.04
589	5718	3693	10000	57.18	36.93	5.89

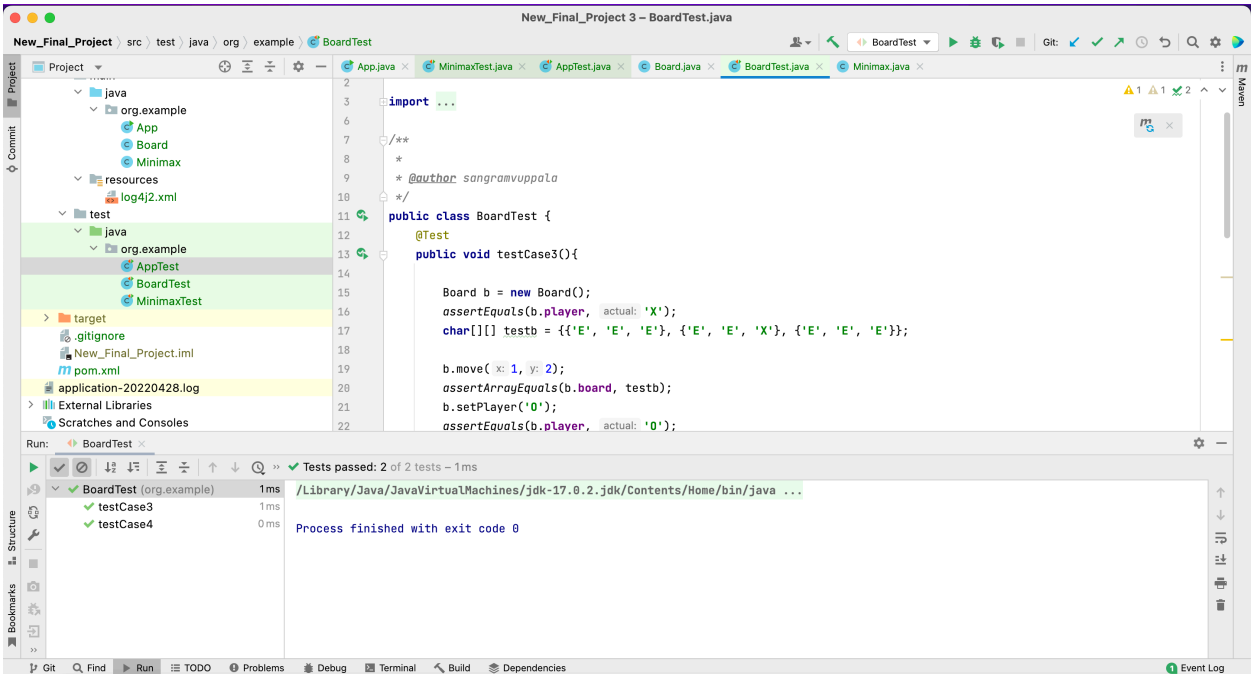
TESTCASES

We have three classes for test coverage–

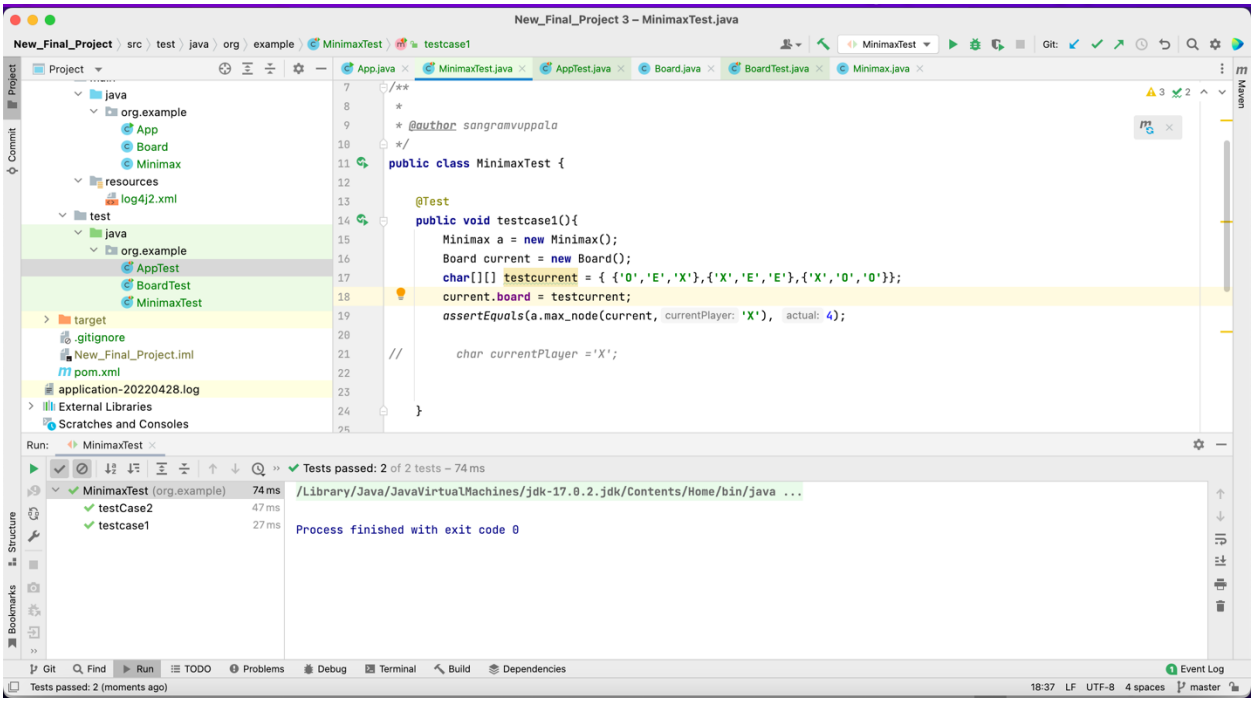
AppTest.java



BoardTest.java



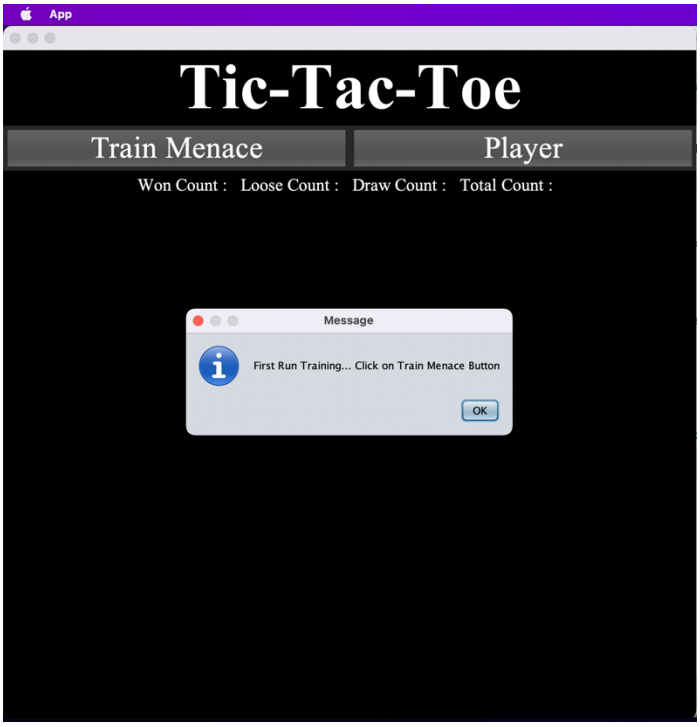
MinimaxTest.java



OUTPUT SCREENS

Here below are our screenshots of our application.

Screenshot 1 – App Load

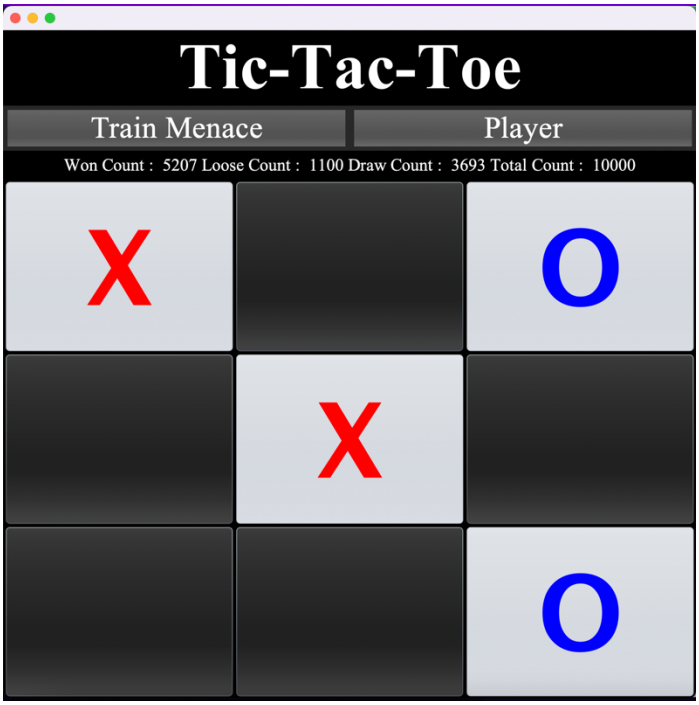


Screenshot 2 – App Load

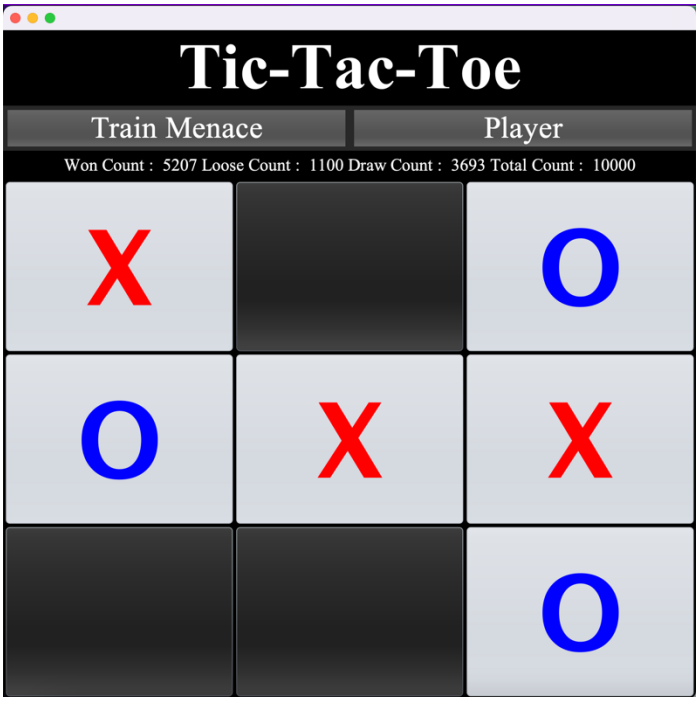
Screenshot 3 – Player button clicked

Screenshot 4 – Player game: move 1

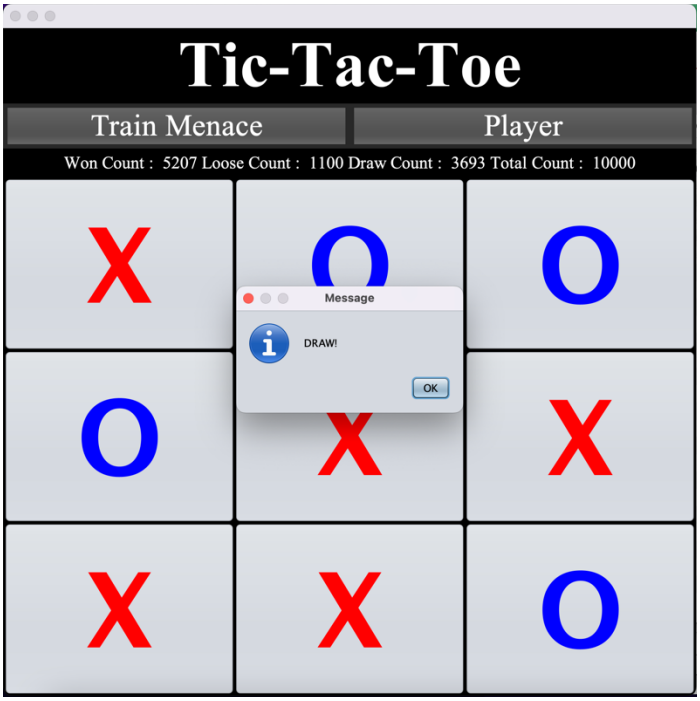
Screenshot 5 – Player game: move 2



Screenshot 6 – Player game: move 3



Screenshot 7 – Player game: move 4. Game finish



CONCLUSION

Every game menace plays four moves or fewer than four moves. The menace's likelihood of winning or drawing improves by 5 percent for every 10,000 games. The proportion of games that finish in a tie was 36 percent, whereas the number of games in which the menace wins increase dramatically as the number of trainings increases. This is because the menace becomes increasingly intelligent after many trainings, resulting in the player winning less.

REFERENCES

-Min Max

- https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/#:~:text=Minimax%20is%20a%20kind%20of,%2C%20Mancala%2C%20Chess%2C%20etc.
- https://en.wikipedia.org/wiki/Burnside%27s_lemma
- https://en.wikipedia.org/wiki/Symmetry_group

Menace

- https://www.mscroggs.co.uk/blog/tags/menace
- https://en.wikipedia.org/wiki/Matchbox_Educable_Noughts_and_Crosses_E
- https://www.mscroggs.co.uk/menace/