INFO 6205 Spring 2022 Project

Menace

Created by :

Swapnil Vise, vise.s@northeastern.edu, NUID – 002110285

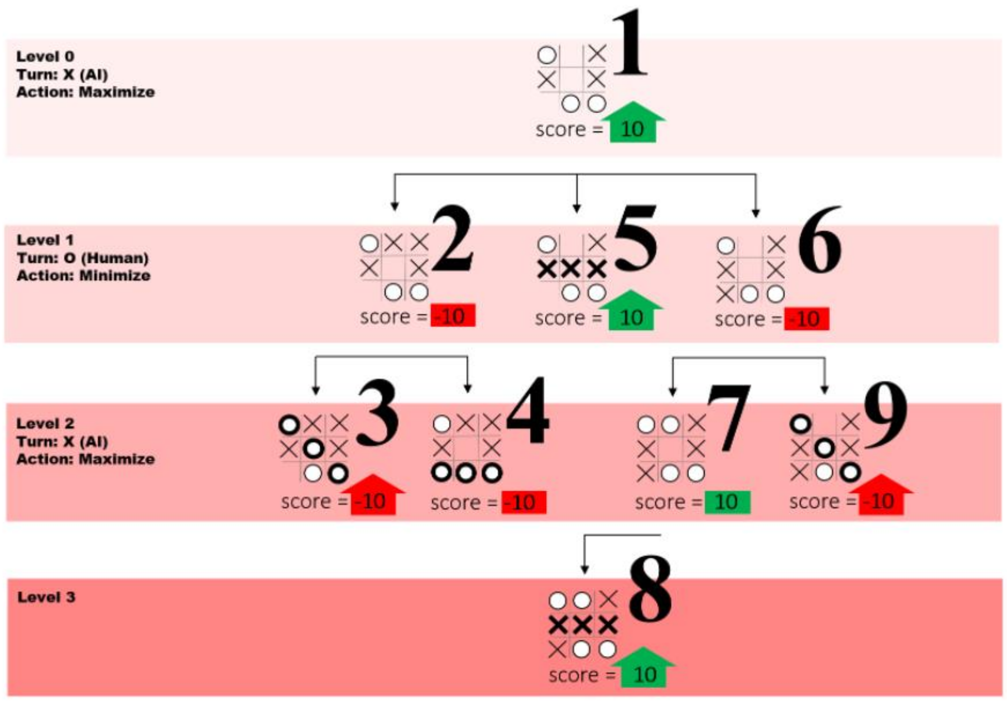
Shrishti Diggikar, diggikar.s@northeastern.edu, NUID – 002125179

Introduction:

Aim: Implement the "MENACE" by replacing matchboxes with values in a hash table (key will be state of the game)

Approach:

As observed from the video, Menace is a Tic tac toe game where in Menace is a system that learns from every move the user plays, and improvises itself as and how it goes on playing. The task of the project was to implement this Menace and replace the originally included matchsticks with hash tables. The approach that we have chosen, is to make use of the Minimax Algorithm which will train the Menace AI with every move the human plays. For the hashtable requirement we have created a separate class named Dictionary, which contains the hash table. To get the best optimal move the Menace uses the backtracking approach and evaluation function. It evaluates the state of the board for every move possible and then backtracks it to choose the best move possible. We have assigned the Menace as the maximizer and human to be the minimizer. The minimax algorithm helps us in evaluating every possible way in which the game can progress and returns us with a value. Now the Menace being the maximizer chooses from a value which will maximize our chances of winning. To check whether the game is over, we have checked if there are any moves available. If there are moves available a true value will be returned or else it will return false. To make the AI smarter we have made use of Alpha Beta Pruning method. This is basically an optimization to the minimax algorithm, where along with the board, depth and toWin we also pass alpha and beta. Alpha is the maximum value the maximizer can give at that moment whereas beta is the minimum value that the minimizer can give at that moment.



To explain the Minimax algorithm along with alpha beta pruning we can consider above image as an example. Just like a human, this algorithm sees a few steps ahead and puts itself in the shoes of its opponent. It continues to play until the board reaches a terminal state, ending in a tie, a win, or a loss. The AI will assign an arbitrary positive score- alpha (+10) for a win, a negative score beta (-10) for a defeat, or a neutral score (0) for a tie until it reaches the terminal state.

At the same time, based on the players' turns, the algorithm assesses the movements that lead to a terminal condition. When it is Menace's turn, it will choose the move with the highest score, and when it is the human player's turn, it will choose the move with the lowest score. Menace avoids losing to the human player by employing this approach.

Program

Data Structures included:

Hash Table**,** Arrays

Classes:

1. Dictionary
2. DictionaryADT
3. DuplicateKeyException
4. InexistentDuplicateKeyException
5. Node
6. Play
7. PossiblePlay

8. Record
9. TicTacToe

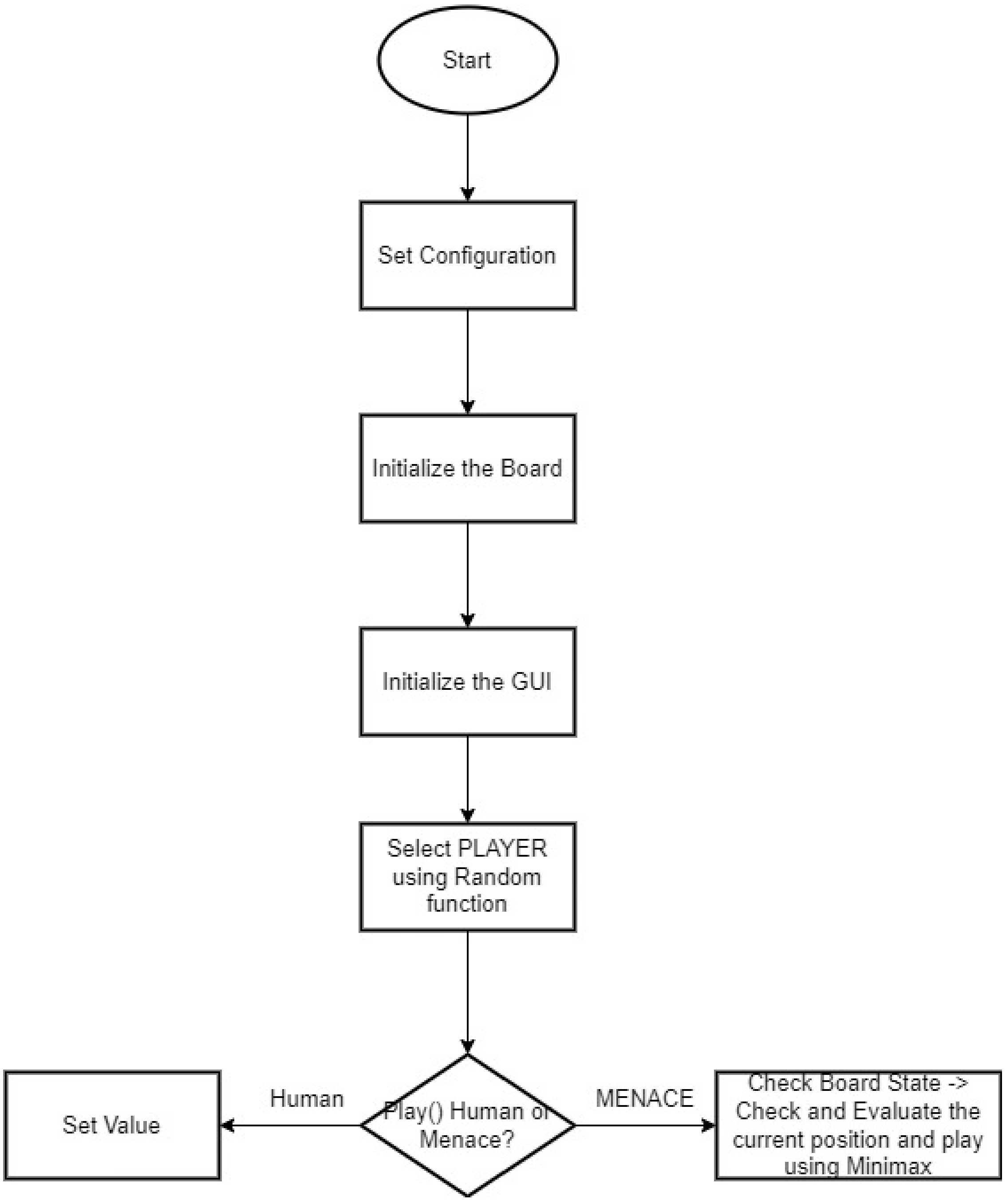
Algorithm

Algorithm used to implement MENACE is Minimax algorithm with alpha-beta pruning for optimization. Our objective is to identify the best possible move for the player. To accomplish so, we just select the node with the highest assessment score. We may also look ahead and analyze possible opponents' movements to make the process smarter. We can look forward to as many movements as our processing capability allows for each move. The algorithm assumes that the opponent is playing at his or her best. Technically, we begin with the root node and select the best node feasible. We assess nodes based on their evaluation ratings. In our scenario, the evaluation function can only award scores to result nodes (leaves). As a result, we recursively reach leaves with scores and backpropagate the scores.

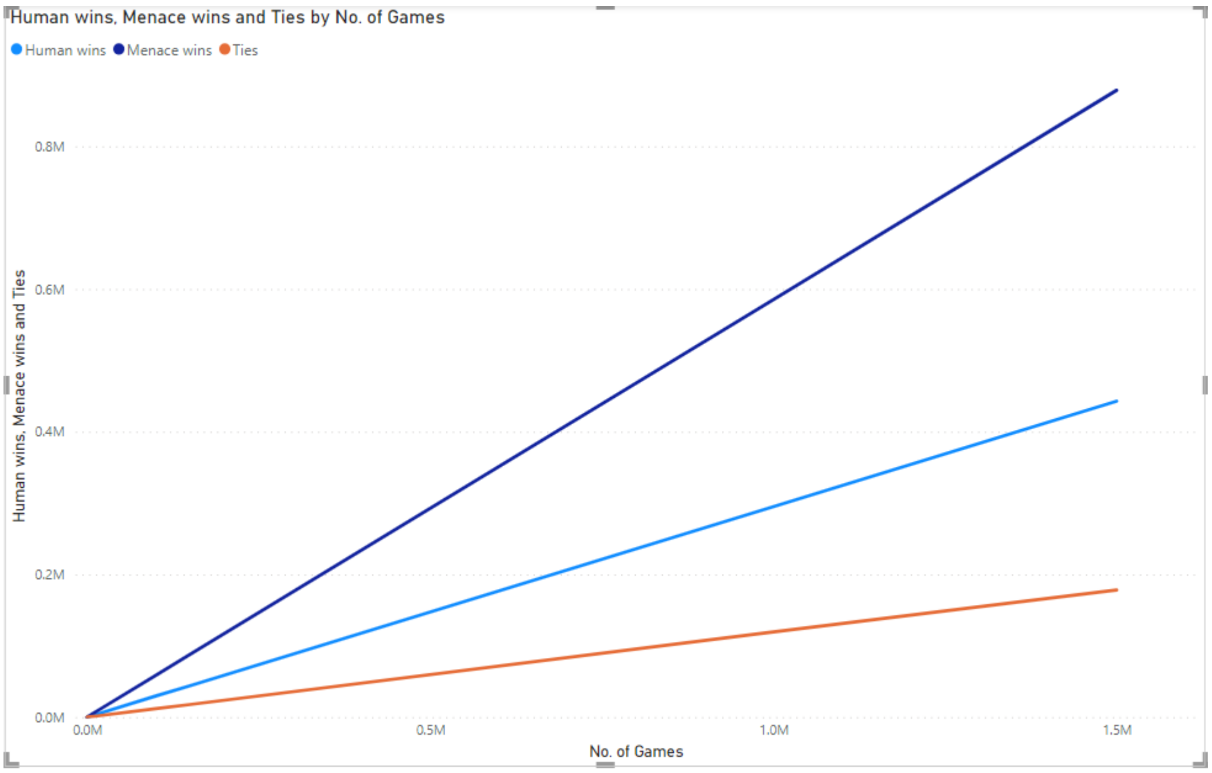
Invariants

The property of invariant defines the state of the board which is always true. So in a Tic Tac Toe game the condition which can be considered to be always true is the winning conditions. That is, getting three X's or O's in any of the three rows, getting three X's or O's in any of the three columns or getting a diagonal line of X's or O's

Flow Charts (inc. UI Flow)


Start
Set Configuration
Initialize the Board
Initialize the GUI
Select PLAYER using Random function
Play() Human or Menace?
Human
Set Value
MENACE
Check Board State -> Check and Evaluate the current position and play using Minimax

Observations & Graphical Analysis



As the number of games played increases it can be observed from the above graphical representation that Menace gets trained on the way and the chances of human winning a game decreases. The chances of the game getting tied also increases which is a result of both human and Menace playing in an optimized manner. This helps in meeting the main motive of our project that is to optimize the hash table in such a way that it can analyze the best possible move and win in less number of moves.

Results & Mathematical Analysis

The upper bound for number of moves that can be played in in a 3 by 3 board can be denoted by

$$b^d$$

where, b is branching factor, which denotes the number of possible moves available at that particular node.

And the total number of ways in which a space can be filled is denoted by factorial of number of spaces available in the board so for the 3 by 3 board that would be 9! = 362,880

Testcases

We have logged in the number of times a game gets tied or either the human or Menace wins the game. Depending on the gathered data we have reached the said conclusions. Sample logger files have been added to the repo in the logger folder.

Below is the description of the test cases that were written to check proper functionality.

testPlay1:

This test case is written to check if PossiblePlay class is returning proper row, column and score as it will be used by Menace when it is evaluating its chances and playing its optimized move.

testPlay2:

This test case is written to check if squareIsEmpty method in the TicTacToe class is returning proper Boolean value, so that if a move has already been played at the selected position there is no overlapping.

testPlay3:

This test case is written to check the win situation of a particular symbol. For this check wins method from TicTacToe class has been called which returns true when there is win situation else it returns false.

testPlay4:

This test case is written to check if the method evalBoard from TicTacToe evaluates the board efficiently.

Conclusion

Algorithms based on reinforcement learning are extremely useful and efficient in refining strategies and decision-making processes. As we progress ahead with the optimized minimax algorithm, the system AI in our case the Menace gets more and more trained and understands the human moves efficiently. It is then able to guess the best optimal move and win in less number of moves rather than taking more moves which it used to do initially. We can conclude it by saying more the number of runs with the Menace increases its chances of winning

References

https://www.youtube.com/watch?v=VrrnjYgDBEk

https://www.youtube.com/watch?v=STjW3eH0Cik