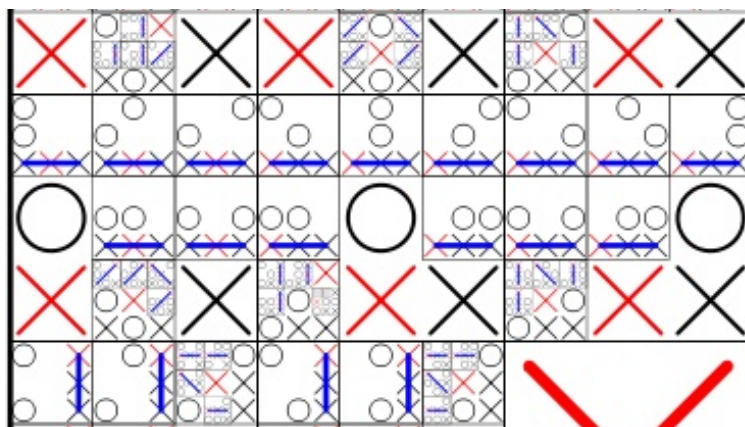## Machine Educable Noughts And Crosses Engine

The MENACE

A gorithmic demonstration of learning menace to play the Tic-Tac-Toe game and learn by reinforcement.

Guided By

Prof. Robin Hillar

Team Members:

Shweta Jose

Tushar Kurhekar

Aditya Mulik

Northeastern University

Index

Introduction

Donald Michie, a British computer scientist who helped break the German Tunny code during the Second World War, came up with Menace (the Machine Educable Noughts And Crosses Engine), a machine that was trained to play tic-tac-toe using 304 matchboxes filled with colored beads.

The 304 matchboxes that makeup Menace represent all the possible layouts a tic-tac-toe board might come across while playing which is reduced from a much larger number by only allowing Menace to play first and treating rotations and reflections as the same board. Each of these matchboxes contains colored beads representing a valid move that Menace could play for the corresponding layout of the board. The box corresponding to Menace's first turn is filled with 8 beads for each different move (each of the 9 colours). The 12 boxes representing the layouts of the board for Menace's second move consist of 4 beads of colors of spots that aren't taken, two beads each for Menace's third move, and one for its 4th move. Since there is only one spot left for its fifth move, there are no boxes used to represent it, as Menace is forced to take up that spot.

Our Aim:

To demonstrate the use of algorithms and mathematics in order to solve the real-time intellectual problem which is to win. It also gives a brief overview on a machine learning algorithm that is being used to train or educate the menace in the game of tic-tac-toe by the method of reinforcement learning (reward and punishment method) which is inspired by the very first video demonstration shown by the professor in the class and meetings. In the end, we have to show the logging and the learning of menace by the graph or report to back the demonstration.

Approach

Step 1

Menace has been implemented in JavaScript using a HashMap where the key is the state of the game (ELABORATE the technical details)

Step 2

When Menace makes its move, one must find the key rep representing the current state of the board layout from which (a bead is taken out randomly) change the bracket part to what is done in JavaScript code The number of beads in each matchbox is represented as *alpha* at the start of the game

Step 3

When MENACE wins the match, the *beta* phase, the MENACE is rewarded with beads. Note: Each matchbox from the 304 matchboxes that played a role in winning the game is awarded **3** beads.

Step 4

When MENACE loses the match, the *gamma* phase, the MENACE is punished by reducing the beads.

Note: Each matchbox from the 304 matchboxes that played a role in losing the game is punished by reducing **1** bead.

Step 5

When the game draws (neither MENACE/nor Human win), the *delta* phase, is where a bead is rewarded.

Note: Each matchbox from the 304 matchboxes which played a role in drawing the game is awarded by adding **1** bead.

The Program

Data Structure and Classes

- HashMap (to Store 304 unique Matchboxes)
- Array or ArrayList (to store incentives/ beads, start value of beads for each matchbox)

Algorithms

Logic Operation Algorithms :

- apply_rotation() - To apply rotation on the state of the game based on symmetry

- three() - To check if MENACE or Opponent has Won by validating the positions on the board either horizontally or vertically or diagonally.

- make_move() - Menace makes Move

- get_random_move() - Getting a random position on the game of board using the Math. random library and in a choice of Nine positions.

- menace_add_beads() - Beats are awarded or deducted (rewards & punishment) based on win, loss, or draw conditions

- reset_menace() - Board is reset to default position and beats are added with the initial number and probability is considered zero at this point.

Common UI Operations :

- getMenaceMove() - Gets the first, third, fifth and seventh move

- new_game() - reset board to null positions

- winner() - check whether menace has won lost or draw
- check_win() - check and log the winning or losing or draw of the opponent with the timestamp
- play_menace() - executed first and runs until a winner is found
- play_opponent() - opponent executes after menace plays and until the winner is found

- The "incentives" [1, 3, -1] used to award and punish Menace
- The winning positions on the board:
- The rotations used to optimize the state of the board

Common Steps on the Tic-Tac-Toe Board Game :

Player Position on The Game
[0,1,2],
[3,4,5],
[6,7,8],
[0,3,6],
[1,4,7],
[2,5,8],
[0,4,8],
[6,4,2]

Player Position on The Game
[0,1,2,3,4,5,6,7,8], [0,3,6,1,4,7,2,5,8],
[6,3,0,7,4,1,8,5,2],
[6,7,8,3,4,5,0,1,2],
[8,7,6,5,4,3,2,1,0],
[8,5,2,7,4,1,6,3,0],
[2,5,8,1,4,7,0,3,6],
[2,1,0,5,4,3,8,7,6]

Critical Points & Invariants

1.

MENACE moves mapped per HashMap; game state saved for each mapping in a JavaScript Object as per the below screenshot.

Below Mappings has ordered boxes that represent the 304 Matchboxes for each move which are symmetric in nature. Each state of the game is represented from these ordered boxes and is found using rotations of the 2d Matrix.

1 – Represents Menace (X)

2 – Represents Human (O)

0 – Represents Blank (Empty boxes on a tic-tac-toe game)

```
▼ {1: {…}, 2: {…}} ℹ
  ▼ 1:
    ▶ boxes: (221211001) [100002000: Array(9), 100012020: Array(9), 100020000: Array(9), 100022010: Array(9),
    ▶ incentives: (3) [1, 3, −1]
    ▶ moves: (4) [Array(2), Array(2), Array(2), Array(2)]
    ▼ orderedBoxes: Array(304)
      ▶ [0 … 99]
      ▶ [100 … 199]
      ▶ [200 … 299]
      ▼ [300 … 303]
          300: "202010121"
          301: "210020121"
          302: "210002121"
          303: "201002121"
        length: 304
      ▶ [[Prototype]]: Array(0)
      player: 1
      removesymm: true
    ▶ start: (4) [8, 4, 2, 1]
    ▶ [[Prototype]]: Object
```

2.

The number of beads in each matchbox represented as *alpha* at the start of the game is as below.

- **8 beads** in the first Matchbox (which is the starting representation of the game)

- Total 0 moves out of 9.

- Menace makes the 1st move

- **4 beads** in the next 12 Matchboxes (which is where Human and Menace both have played once)

- Total 2 Moves out of 9

- Menace makes the 3rd move

- **2 beads** in the next 108 Matchboxes (which is where Human and Menace both have played twice)

- Total 4 Moves out of 9

- Menace makes the 5th move

- **1 bead** in the next 183 Matchboxes (which is where Human and Menace both have played thrice)

- Total 6 Moves out of 9

- Menace makes the 7th move

- **No Matchbox or beads** representation left, last (9th) and final move which would either result in the game won by MENACE or DRAW.
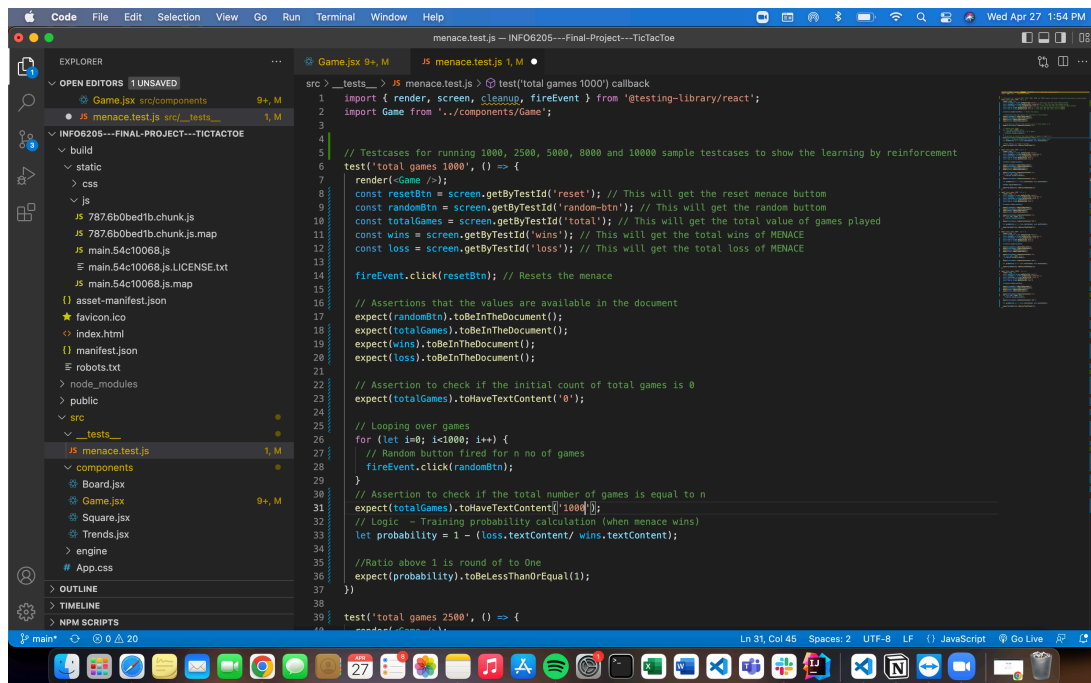
```
▼ 1:
  ▶ boxes: (221211001) [100002000: Array(9), 100012020: Array(9), 100020000: Array(9), 100022010:
  ▶ incentives: (3) [1, 3, -1]
  ▶ moves: (4) [Array(2), Array(2), Array(2), Array(2)]
  ▶ orderedBoxes: (304) ['000000000', '120000000', '100020000', '100002000', '210000000', '0100200
    player: 1
    removesymm: true
  ▼ start: Array(4)
      0: 8
      1: 4
      2: 2
      3: 1
      length: 4
    ▶ [[Prototype]]: Array(0)
```

3.

When MENACE wins the match, the **beta** phase, the MENACE is rewarded with beads.

Note: Each matchbox from the 304 matchboxes which played a role in winning the game is awarded **3** beads.

4.

When MENACE loses the match, the **gamma** phase, the MENACE is punished by reducing the beads.

Note: Each matchbox from the 304 matchboxes which played a role in losing the game is punished by reducing **1** bead.

5.

When the game draws (neither MENACE/nor Human win), the **delta** phase, is where a bead is rewarded.

Note: Each matchbox from the 304 matchboxes which played a role in drawing the game is awarded by adding **1** bead.

```
▼ 1:
  ▶ boxes: (221211001) [100002000: Array(9), 100012020: Array(9), 100020000: Array(9), 100022010:
  ▶ incentives: (3) [1, 3, -1]
  ▶ moves: (4) [Array(2), Array(2), Array(2), Array(2)]
  ▶ orderedBoxes: (304) ['000000000', '120000000', '100020000', '100002000', '210000000', '0100200
    player: 1
    removesymm: true
  ▼ start: Array(4)
      0: 8
      1: 4
      2: 2
      3: 1
      length: 4
    ▶ [[Prototype]]: Array(0)
```

Reference Image representing points 3, 4 & 5 (incentives)

Logging Mechanism and Methods

Each move of MENACE and OPPONENT has been recorded with the position it plays and at what time.

We are logging the every win or loss with manace and the position played on the board with timestamp to show how manace is getting trained.
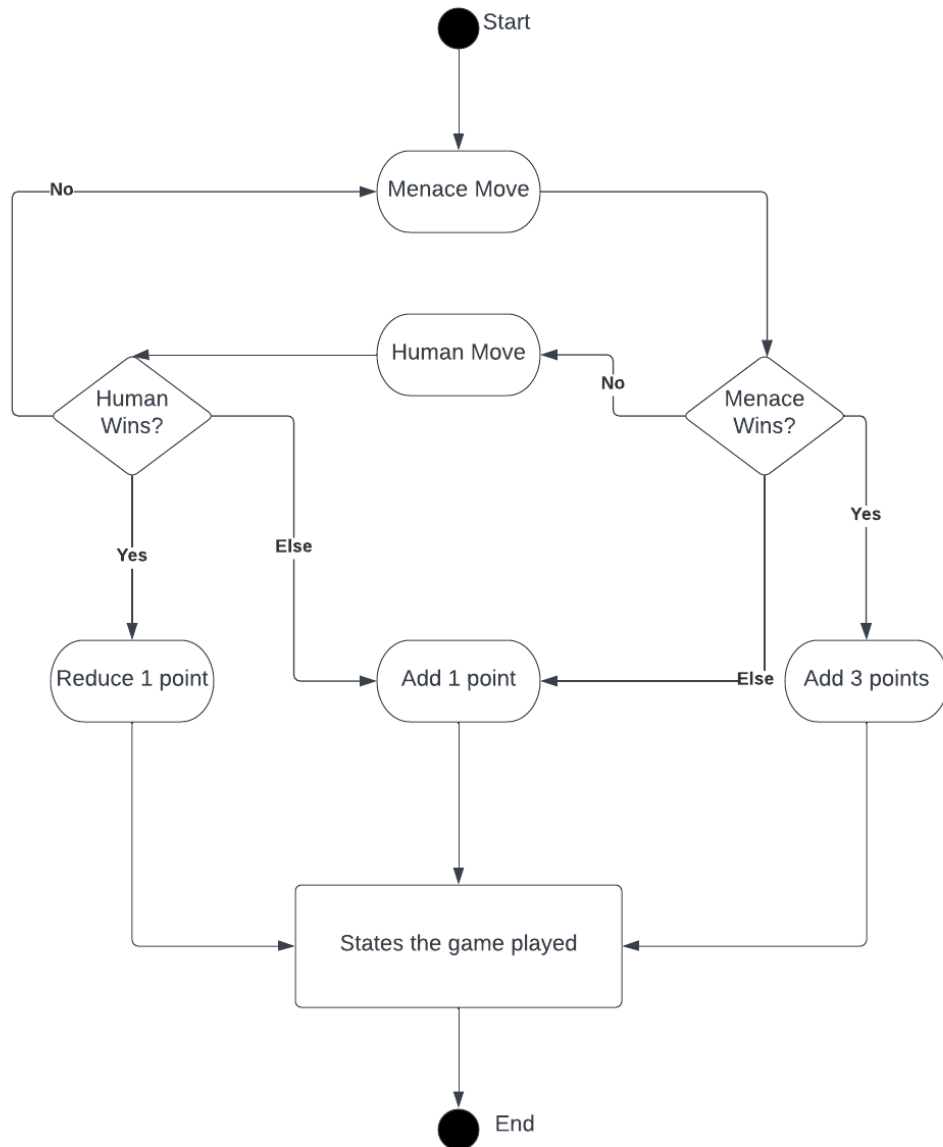

Implementation In the code base

We are logging in the console of the JavaScript to how the logging step and training of the menace machine. Following is the output of the code in the console.

```
Reset Menace                                                        menace.js:445

INFO: MENACE played it's move (O) at position 3 on the board at Tue    menace.js:208
Apr 26 2022 10:44:39 GMT-0400 (Eastern Daylight Time)

INFO: Opponent played it's move (X) at position 2 on the board at Tue  menace.js:234
Apr 26 2022 10:44:39 GMT-0400 (Eastern Daylight Time)

INFO: MENACE played it's move (O) at position 8 on the board at Tue    menace.js:208
Apr 26 2022 10:44:39 GMT-0400 (Eastern Daylight Time)

INFO: Opponent played it's move (X) at position 6 on the board at Tue  menace.js:234
Apr 26 2022 10:44:39 GMT-0400 (Eastern Daylight Time)

INFO: MENACE played it's move (O) at position 0 on the board at Tue    menace.js:208
Apr 26 2022 10:44:39 GMT-0400 (Eastern Daylight Time)

INFO: Opponent played it's move (X) at position 7 on the board at Tue  menace.js:234
Apr 26 2022 10:44:39 GMT-0400 (Eastern Daylight Time)

INFO: MENACE played it's move (O) at position 4 on the board at Tue    menace.js:208
Apr 26 2022 10:44:39 GMT-0400 (Eastern Daylight Time)

INFO: MEANCE won at Tue Apr 26 2022 10:44:39 GMT-0400 (Eastern         menace.js:160
Daylight Time)
```

Flow Chart

```
                          ● Start
                          │
                          ▼
   No ─────────────→ ( Menace Move ) ─────────────┐
   │                                               │
   │                                               ▼
   │         ( Human Move ) ◄──── No ──── ◇ Menace
   │              │                            Wins?
   ▼              ▼                          /      \
  ◇ Human                                 Else      Yes
   Wins?                                    │         │
  /    \                                    ▼         ▼
Yes    Else                          ( Add 1 point ) ( Add 3 points )
 │       │
 ▼       ▼
( Reduce 1 point )  ( Add 1 point )

        States the game played
                │
                ▼
               ● End
```

Flow diagram of the tic-tac-toe game played by
Menace

Observation & Graphical Analysis
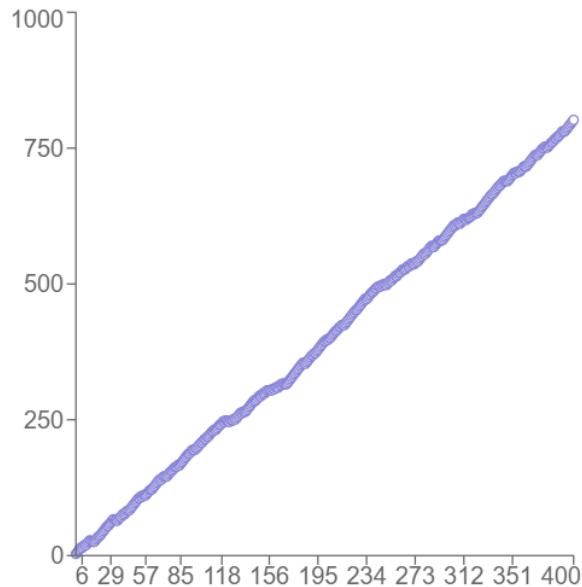

Menace playing againts random player

The graph shows Menace playing against a "Random" player. The x-axis represents the number of games played by Menace and y-axis represents the cumulative reward given to Menace. Three points are awarded for the states representing the winning moves, one for a draw game and one is taken away from each of the relevant states when Menace loses. This means that if Menace played badly, they would have a smaller chance of playing the same game next time. However, if Menace played well, it is more likely to follow the same route the next time and win again.

From the graph above which shows Menace's performance against a "Random" player, out of a total of 400 games, Menace wins 271 times, the "Random" player wins 70 times, and draws 59 times.

Total Games 254   Menace Wins 0   Human Wins 42   Draw 212
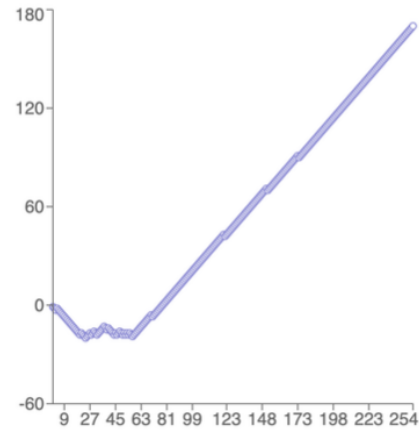
Menace's performance when it plays with the "perfect strategy" Human player

The graph above represents Menace's performance when it plays with the "perfect strategy" Human player. The plot dips initially after a few losses, Menace learns and ends up with 212 draws in 254 games because of the reinforcement provided to it which makes it more likely to make those same routes again.

Result & Mathematical Analysis

Increasing or decreasing the probability of the decisive steps in the tic-tac-toe games results in regeneration of rewards or punishment to the decision making processing with backtracking and positively affects the decision making process of the menace playing the game either to make it a win or draw after a certain iteration of games.

Our inspiration mainly comes from Markov's Decision making process. This action or methodology helps create the best decision or action with the help of reinforcement learning.

Our Approach and Critical Points section dive deep into the steps we follow in the code and in this section we will discuss the underlying mathematics involved while calculating the probability.
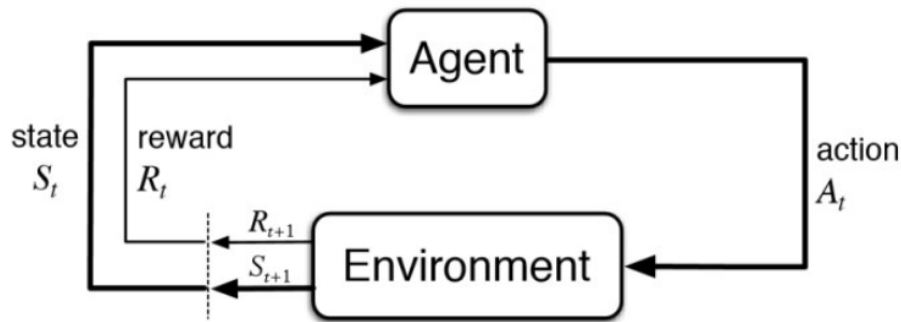
Reinforcement Learning Conceptualization : Source Internet

Donald Machice's Win Loss and Draw Correlation :

Depending on the strategy employed by the human player, MENACE produces a different trend on wins or losses. Using a random turn from the human player results in an almost-perfect positive trend. Playing the optimal strategy returns a slightly slower increase.The reinforcement does not create a perfect standard of wins; the algorithm will draw random uncertain conclusions each time. After the *n-th* round, the correlation of near-perfect play runs:

$$\frac{1-D}{D-D^{(j+2)}} \sum_{i=0}^{j} D^{(ji+1)} V_i$$

Caption

Where V_i is the outcome (+1 wins, 0 draws and -1 is loss) and D is the decay factor (average of past values of wins and losses). Below, M_n is the multiplier for the n-th round of the game.

Following is the training probability which is one of the properties or invariants responsible for reinforcement in the menace machine.

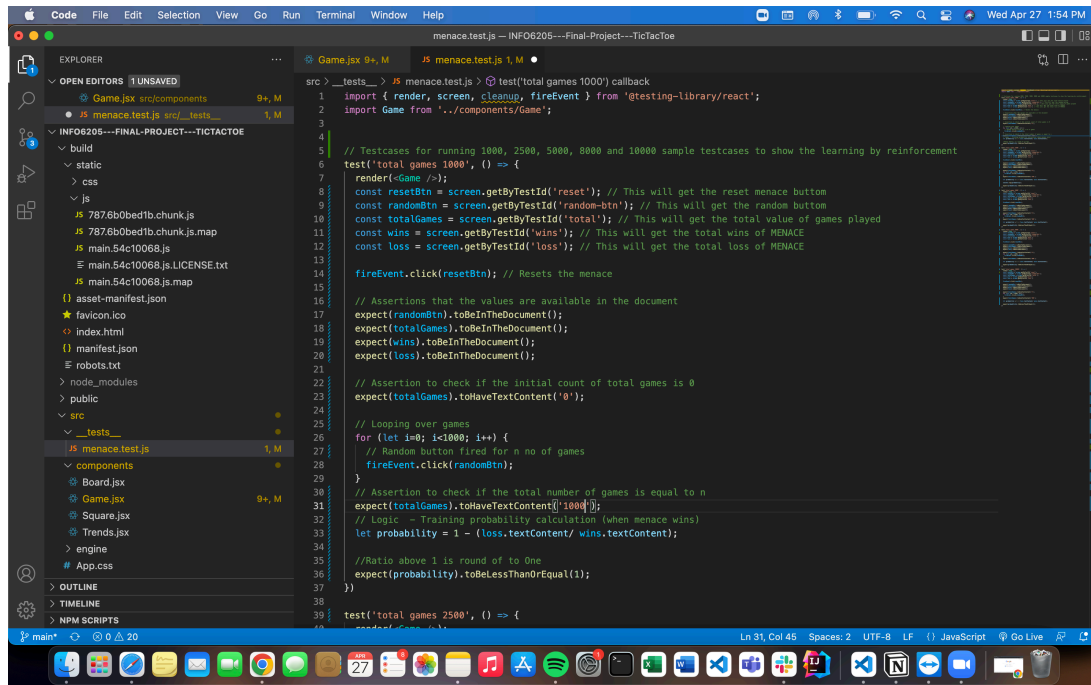| Number of games	Menace win	Human win	Training Probability (p)
50	25	17	0.32
100	61	23	0.55
150	97	30	0.70
200	137	37	0.70
250	174	47	0.71
351	247	67	0.72
501	362	89	0.75
800	598	124	0.76
1000	752	147	0.79
1503	1151	194	0.87

As you can see from the table every iteration of tests that we perform i.e number of games performed will give us an increasing probability of winning.

We have derived it from the real-time results from the code and can be formulated as

Training Probability (p) = 1- (Human win / Menace win)*

*Note- * we are considering DRAWs as a no-effect co-efficient as it won't affect the probability -ve ly. Hence we are rounding the results based on wins or losses in order to demonstrate the effectiveness considering base i.e Menace is only learning (at least) for the win and avoiding those moves.*

Test Cases

Test cases implementation In the code base

Testcases passed for various configurations

Conclusion

To conclude, MENACE always plays the first move. MENACE represents the **'O'** on the Tic-tac-toe board. There are 304 Matchbox representations of each move played by the MENACE & the opponent. Each matchbox contains beads of 9 assorted colors for 9 distinct positions of a Tic-tac-toe Board. Considering symmetric or identical moves played is correlated with a matchbox, every win, loss, and draw in the game has rewards or punishment associated with MENACE by reducing or adding beads to the matchboxes.

Mathematically, We can see the code "learning" from its wrong move and increating the winning probability. To put it Simply after understanding the Menace Machine, Reinforcement Learning, and Mathematics, The number of beats getting added acts as a reward or punishment, and next time, The box will have more beats of that particular color to pick which made us win. Repeating the same process there will be the point where we know Menace wins because matchboxes will have the same color beats.

This project serves as a great example of Code and Mathematics generating amazing intellectual for the betterment of the World.

References

- https://www.mscroggs.co.uk/blog/19
- https://en.wikipedia.org/wiki/Matchbox_Educable_Noughts_and_Crosses_Engine
- https://odsc.medium.com/how-300-matchboxes-learned-to-play-tic-tac-toe-using-menace-35e0e4c29fc
- https://www.atarimagazines.com/v3n1/matchboxttt.html
- https://www.belloflostsouls.net/2019/03/teaching-304-matchboxes-to-beat-you-at-tic-tac-toe.html
- https://wikitia.com/wiki/Matchbox_Educable_Noughts_And_Crosses_Engine

Bibliography

- Menace - **something that threatens to cause evil, harm, injury, etc.; a threat**: Air pollution is a menace to health. a person whose actions, attitudes, or ideas are considered dangerous or harmful

- Reinforcement learning - **a machine learning training method based on rewarding desired behaviors and/or punishing undesired ones**. In general, a reinforcement learning agent is able to perceive and interpret its environment, take actions and learn through trial and error.

- **Artificial neural networks** (**ANNs**) :usually simply called **neural networks** (**NNs**), are computing systems inspired by the biological neural networks that constitute animal brains.

- Cumulative reward : an incentive mechanism that tells the agent what is correct and what is wrong using reward and punishment. The goal of agents in RL is to maximize the total rewards.

Thank You