

# INFO 6205 Spring 2022 Project

## The Menace

### Team members:

Shashank Siripragada, Section 8, NUID: 002193773

Mayannk Kumaar, Section 8, NUID: 001537115

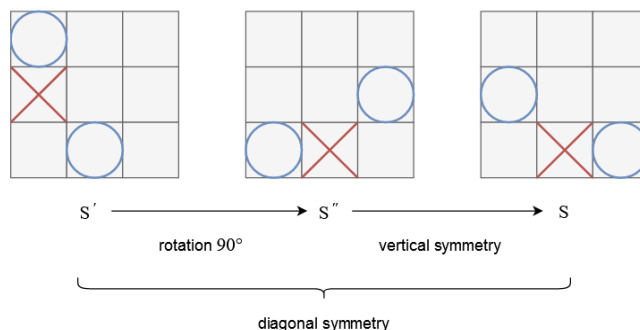
### Introduction

#### Aim:

- Implement “The Menace” by replacing matchboxes with values in a hash table (the key will be the state of the game)
- Train the Menace by running games played against “human” strategy, which is based upon the optimal strategy

#### Approach:

- We generate all the possible states of the tic-tac-toe game. We try to reduce the states by checking if one state is equal to some other state after performing rotational steps (flips/ mirror/ rotate) to obtain unique states



- We add indices (beads) which are equally distributed at the beginning to all states achieved after a reduction so that every move is equiprobable
- We train the menace agent against the human agent so that it gets better when it plays a real human
- To train the agent we take several iterations, the probability of ideal moves played by a human agent as input from the user
- At first, the menace agent picks a random index (bead) from the state matching with the empty board state and places 'X'
- The human agent matches the current state with the sample states and picks the best move (index/ bead) given the probability mentioned by the user and places 'O'
- All the moves by menace agents are tracked along with the states which led to these moves.
- If the menace wins the game, we reward the menace by adding the picked index (bead) for each state to the maintained dictionary. Which in return increases the probability of playing that move if this state is achieved later making our chances of winning higher
- If the menace wins the game, we punish the menace by removing the picked index (bead) for each state from the maintained dictionary. Which in return decreases the probability of playing that move if this state is achieved later, we might not play this move and end up losing again
- If the game is drawn, we neither add nor remove indices from the dictionary

Strategy:

When training a menace agent with a human agent, we sample a random value, if it is greater than the given epsilon (1-probability) value the human agent chooses the optimal strategy i.e. picks the best possible move. Else, the human agent picks a random move from the list of possible moves.

## **Program**

### Data Structures:

- The **list** is used to store the game state
- **Dictionary** is used to hash the value of the state to possible moves (beads)
- A **tuple** is used to convert a state list to hashing key value in a dictionary
- **Set** is used to find unique possible moves

### Functions:

We use the following functions to store and manipulate states.

**vflip** : Get a vertical flip of a state

**rotate90** : rotate a given state by 90 degrees right

**next\_state** : Obtain the next state given the current state and player

**remove\_duplicates** : Remove duplicate states after checking possible flips and rotations

**set\_possible\_moves** : Given a state set all possible moves

**get\_required\_action** : Given an action, the number of rotations and flips get action after performing given rotations and flips

**get\_original\_action** : Given an action, the number of rotations and flips get the Original action

**is\_win** : check for the winner given the board

**match\_two\_states** : validate whether two given states are equal or not

**check\_equal** : validate two given states are equal or not after performing rotational steps (flip/mirror/ rotate)

```
menace_humanplay.py x menace_selfplay.py x states.py x utils.py x
1 def vflip(state):
2     """
3     Get Mirror image (flipped) of a state
4
5     Returns: Flipped state
6     """
7
8     flipped = list(state)
9
10    flipped[2], flipped[5], flipped[8] = state[0], state[3], state[6]
11    flipped[0], flipped[3], flipped[6] = state[2], state[5], state[8]
12
13    return flipped
14
15 def rotate90(state):
16     """
17     Rotate a given state by 90 degrees right
18
19     Returns: Rotated state
20     """
21
22    rotated = list(state)
23
24    rotated[0], rotated[2], rotated[8], rotated[6] = state[6], state[0], state[2], state[8]
25    rotated[1], rotated[5], rotated[7], rotated[3] = state[3], state[1], state[5], state[7]
26
27    return rotated
28
29 def next_state(cur, next, player):
30     """
31     Obtain next state given current state and player
32     """
33     # print(cur, '\n', 'len of state', len(cur))
34     for i in range(len(cur)):
35         for j in range(9):
36             next_s = []
37             if cur[i][j] == 0:
38                 next_s = list(cur[i])
39                 next_s[j] = player
```

```

menace_humanplay.py  menace_selfplay.py  states.py  utils.py
205 def check_equal(state_1, state_2):
206     '''
207     Given two states check whether they are equal
208     after all possible flips and rotations
209
210     Returns : bool - indicating whether two states are equal
211             nrotations - number of rotations to achieve equality
212             nflips - number of flips to achieve equality
213     '''
214
215     nflips, nrotations = 0, 0
216     s1, s2 = state_1, List(state_2)
217
218     if match_two_states(s1, s2) : return True, nrotations, nflips
219
220     ## one rotation
221     nrotations += 1
222     s2 = rotate90(s2)
223     if match_two_states(s1, s2) : return True, nrotations, nflips
224
225     ## two rotations
226     nrotations += 1
227     s2 = rotate90(s2)
228     if match_two_states(s1, s2) : return True, nrotations, nflips
229
230     ## three rotations
231     nrotations += 1
232     s2 = rotate90(s2)
233     if match_two_states(s1, s2) : return True, nrotations, nflips
234
235     ## Flip
236     s2 = vflip(List(state_2))
237     nflips += 1
238     nrotations = 0
239     if match_two_states(s1, s2) : return True, nrotations, nflips
240
241     ## one rotation
242     nrotations += 1
243     ...

```

```

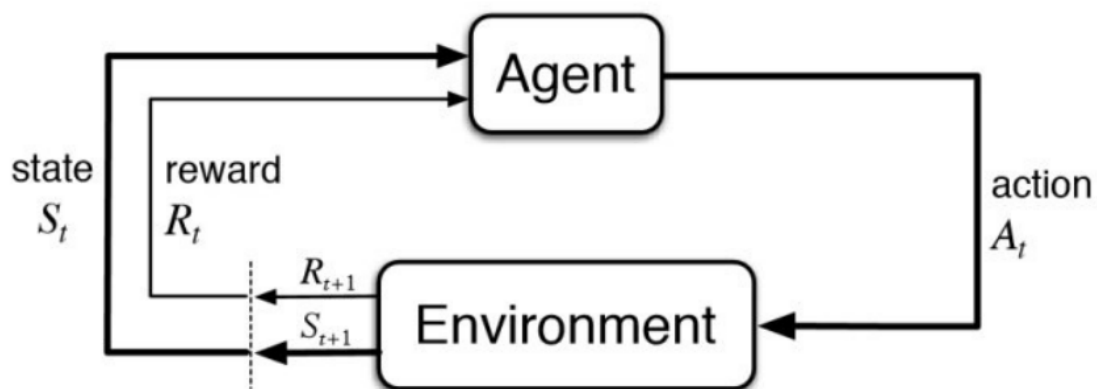
72
73 def set_possible_moves(states, alpha=2):
74     '''
75     Given a state set all possible moves
76
77     Returns : possible_moves (hashmap): possible moves for
78             each state in states
79     '''
80     n = len(states)
81     possible_moves = {}
82
83     for i in range(n):
84         actions = []
85         for j in range(len(states[i])):
86             if states[i][j] == 0:
87                 for alp in range(alpha):
88                     actions.append(j)
89                     #actions.append(j)
90
91         possible_moves[tuple(states[i])] = actions
92
93     return possible_moves
94

```

Above are the images of the selected functions to manipulate and store state information.

## Algorithm:

Ideally, 2 player zero-sum games (Tic-Tac-Toe, Chess, Go, etc.) use a min-max algorithm, in which it keeps on playing and exploring all states with outcomes win/ lose/ draw and selects the path with the maximum reward. But here we will be using reinforcement learning where our menace agent will be trained for a certain number of iterations against a human-like agent with probability as input from a user. If the probability is not given we take the default probability of human agent optimal moves as 0.7 ( $1 - \epsilon$  {0.3} )



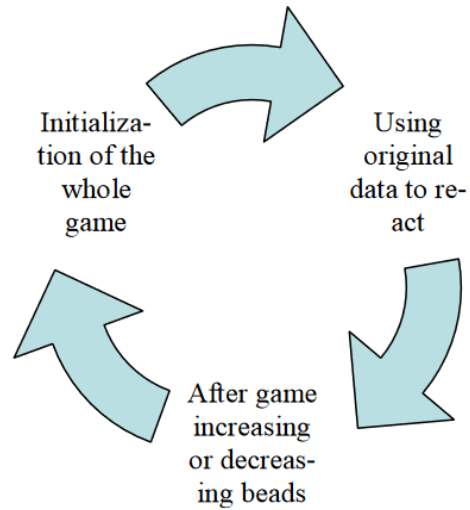


Figure 3. Process of the game

## Invariants:

- The state of the board (S) would remain unchanged by any rotation or symmetry
- At any given point in the game, the difference between the number of 'X' & 'O' on board will never be greater than 1.

## Logs

**Game Log:** The game log contains information regarding the initialization of the states and also state reduction information.

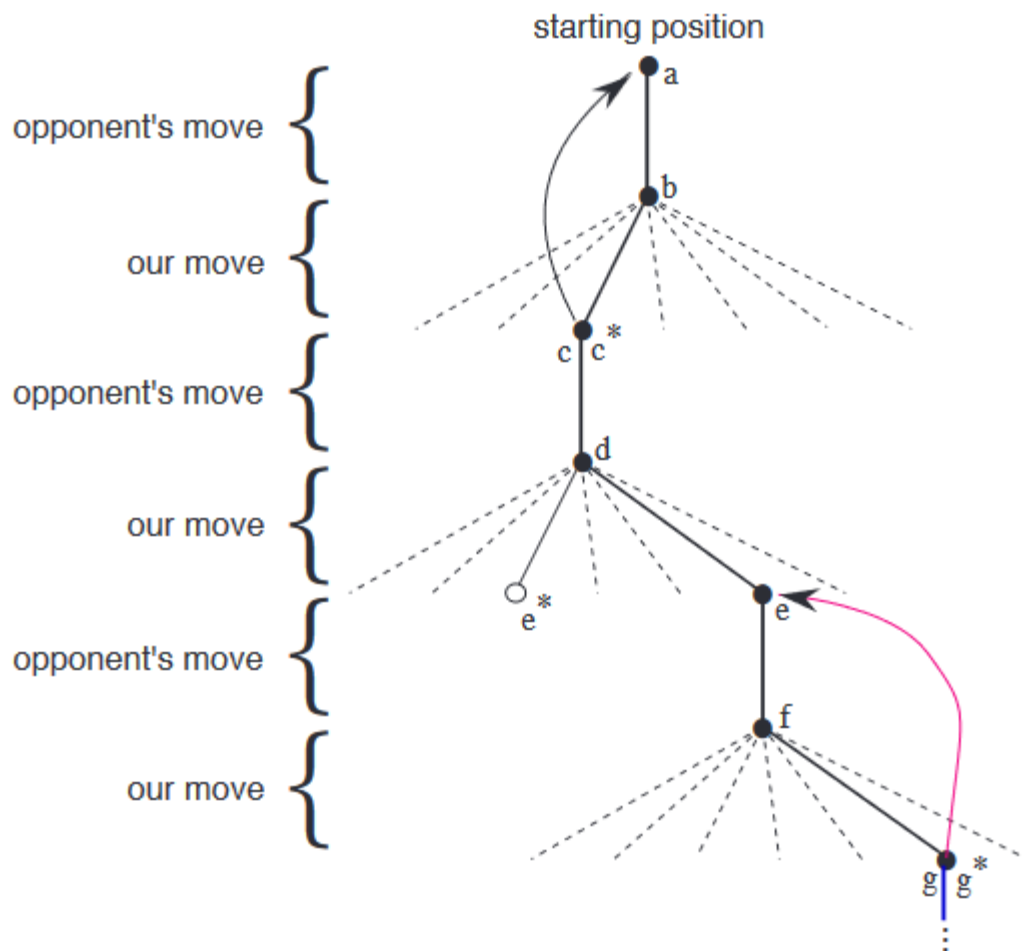
```
1 2022-04-27 20:40:00,923 INFO No of first possible move states: 3
2 2022-04-27 20:40:00,923 INFO #####
3 2022-04-27 20:40:00,923 INFO No. of next possible move states: 24
4 2022-04-27 20:40:00,923 INFO Reduced no. of states after eliminating duplicates: 12
5 2022-04-27 20:40:00,923 INFO #####
6 2022-04-27 20:40:00,923 INFO No. of next possible move states: 84
7 2022-04-27 20:40:00,955 INFO Reduced no. of states after eliminating duplicates: 38
8 2022-04-27 20:40:00,955 INFO #####
9 2022-04-27 20:40:00,955 INFO No. of next possible move states: 228
10 2022-04-27 20:40:01,111 INFO Reduced no. of states after eliminating duplicates: 108
11 2022-04-27 20:40:01,111 INFO #####
12 2022-04-27 20:40:01,111 INFO No. of next possible move states: 540
13 2022-04-27 20:40:02,023 INFO Reduced no. of states after eliminating duplicates: 174
14 2022-04-27 20:40:02,023 INFO #####
15 2022-04-27 20:40:02,023 INFO No. of next possible move states: 696
16 2022-04-27 20:40:03,545 INFO Reduced no. of states after eliminating duplicates: 228
17 2022-04-27 20:40:03,545 INFO #####
18 2022-04-27 20:40:03,545 INFO No. of next possible move states: 684
19 2022-04-27 20:40:05,036 INFO Reduced no. of states after eliminating duplicates: 174
20 2022-04-27 20:40:05,036 INFO #####
21 2022-04-27 20:40:05,036 INFO No. of next possible move states: 348
22 2022-04-27 20:40:05,427 INFO Reduced no. of states after eliminating duplicates: 89
23 2022-04-27 20:40:05,427 INFO #####
24 2022-04-27 20:40:05,427 INFO No. of next possible move states: 89
25 2022-04-27 20:40:05,443 INFO Reduced no. of states after eliminating duplicates: 23
26 2022-04-27 20:40:05,458 INFO #####
```

**Training Log:** The training log contains information on training runs when the MENACE agent plays against Human-agent where MENACE\_1 is the menace agent and MENACE\_2 is a human agent. We log the two opponents, the epsilon value, and the outcome of the game.



1	2022-04-27 19:27:12,182	INFO	Training Game 0 Time: 2022-04-27 19:27:12.182141
2	2022-04-27 19:27:12,182	INFO	menace_agent_prob: 1 human_agent_prob: 0
3	2022-04-27 19:27:12,197	INFO	Won: MENACE_1
4	2022-04-27 19:27:12,226	INFO	Training Game 1 Time: 2022-04-27 19:27:12.226488
5	2022-04-27 19:27:12,226	INFO	menace_agent_prob: 1 human_agent_prob: 0
6	2022-04-27 19:27:12,232	INFO	Draw
7	2022-04-27 19:27:12,232	INFO	Training Game 2 Time: 2022-04-27 19:27:12.232591
8	2022-04-27 19:27:12,232	INFO	menace_agent_prob: 1 human_agent_prob: 0
9	2022-04-27 19:27:12,263	INFO	Draw
10	2022-04-27 19:27:12,263	INFO	Training Game 3 Time: 2022-04-27 19:27:12.263852
11	2022-04-27 19:27:12,263	INFO	menace_agent_prob: 1 human_agent_prob: 0
12	2022-04-27 19:27:12,263	INFO	Won: MENACE_1
13	2022-04-27 19:27:12,297	INFO	Training Game 4 Time: 2022-04-27 19:27:12.297534
14	2022-04-27 19:27:12,297	INFO	menace_agent_prob: 1 human_agent_prob: 0
15	2022-04-27 19:27:12,318	INFO	Won: MENACE_1
16	2022-04-27 19:27:12,362	INFO	Training Game 5 Time: 2022-04-27 19:27:12.362840
17	2022-04-27 19:27:12,362	INFO	menace_agent_prob: 1 human_agent_prob: 0
18	2022-04-27 19:27:12,383	INFO	Draw
19	2022-04-27 19:27:12,383	INFO	Training Game 6 Time: 2022-04-27 19:27:12.383791
20	2022-04-27 19:27:12,383	INFO	menace_agent_prob: 1 human_agent_prob: 0
21	2022-04-27 19:27:12,396	INFO	Draw
22	2022-04-27 19:27:12,396	INFO	Training Game 7 Time: 2022-04-27 19:27:12.396831
23	2022-04-27 19:27:12,396	INFO	menace_agent_prob: 1 human_agent_prob: 0
24	2022-04-27 19:27:12,396	INFO	Won: MENACE_1
25	2022-04-27 19:27:12,428	INFO	Training Game 8 Time: 2022-04-27 19:27:12.428092
26	2022-04-27 19:27:12,428	INFO	menace_agent_prob: 1 human_agent_prob: 0
27	2022-04-27 19:27:12,445	INFO	Won: MENACE_1
28	2022-04-27 19:27:12,478	INFO	Training Game 9 Time: 2022-04-27 19:27:12.478756
29	2022-04-27 19:27:12,478	INFO	menace_agent_prob: 1 human_agent_prob: 0
30	2022-04-27 19:27:12,494	INFO	Won: MENACE_2
31	2022-04-27 19:27:12,529	INFO	Training Game 10 Time: 2022-04-27 19:27:12.529369
32	2022-04-27 19:27:12,529	INFO	menace_agent_prob: 1 human_agent_prob: 0
33	2022-04-27 19:27:12,545	INFO	Won: MENACE_1
34	2022-04-27 19:27:12,586	INFO	Training Game 11 Time: 2022-04-27 19:27:12.586654
35	2022-04-27 19:27:12,586	INFO	menace_agent_prob: 1 human_agent_prob: 0
36	2022-04-27 19:27:12,596	INFO	Won: MENACE_1
37	2022-04-27 19:27:12,627	INFO	Training Game 12 Time: 2022-04-27 19:27:12.627805
38	2022-04-27 19:27:12,627	INFO	menace_agent_prob: 1 human_agent_prob: 0

# Flow



## UI

After selection mode **'play'** the code initializes a command-line based game where MENACE (**X**) goes first. Then we are prompted to choose the remaining indices on the board (0 to 8). Our move is represented by (**O**). Our selected index is then displayed on the board.

```
Begin game 0
MENACE's Turn

  X |   |
  ---|---
    |   |
  ---|---
    |   |

Indices
  0 | 1 | 2
  ---|---
  3 | 4 | 5
  ---|---
  6 | 7 | 8

You choose
Select index:

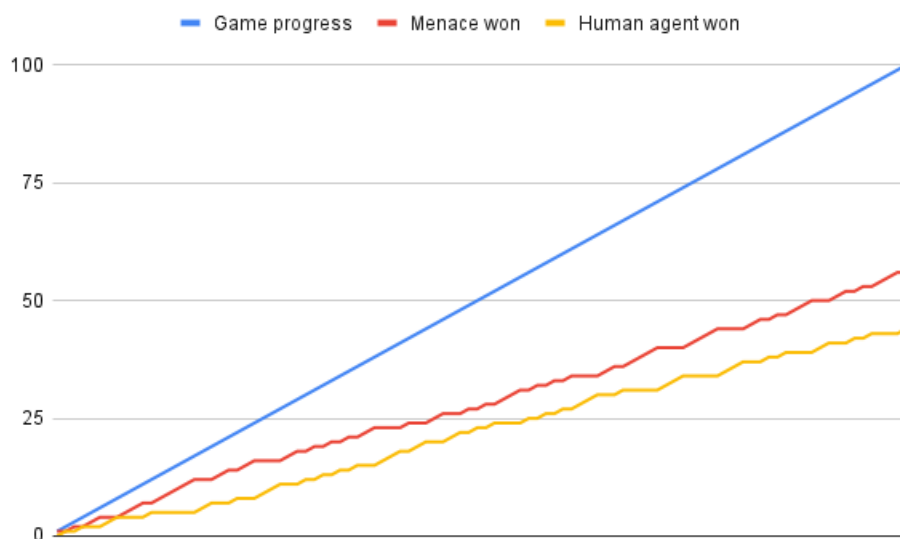
  X |   |
  ---|---
    | 0 |
  ---|---
  X |   |

Indices
  0 | 1 | 2
  ---|---
  3 | 4 | 5
  ---|---
  6 | 7 | 8

You choose
Select index:
```

## Observations & Graphical Analysis

As we pick an optimal human strategy with a probability of  $P^*$  and plot the number of games where the menace agent played against the human agent. We observe that the chances of menace winning increase as we play more games



# Testcases

```
tests.py x
1 import unittest
2
3 from states import vflip, rotate90, next_state, remove_duplicates
4 from states import set_possible_moves, get_required_action, get_original_action
5 from states import is_win, match_two_states, check_equal
6
7 class TestMenaceStateMethods(unittest.TestCase):
8
9     def test_invariant(self):
10         """
11         Test for checking invariant that difference in
12         count of X's and O's on board should be <= 1
13         """
14         state = [1,2,2,0,0,1,0,0,0]
15         self.assertEqual(abs(state.count(1) - state.count(2))<=1)
16
17     def test_vflip(self):
18         """
19         Test for Vertical flip of a state
20         """
21         self.assertEqual(vflip([1,0,0,0,0,0,0,0,0]), [0,0,1,0,0,0,0,0,0])
22
23     def test_rotate90(self):
24         """
25         Test for rotate state by 90 degrees right
26         """
27         self.assertEqual(rotate90([1,2,2,0,0,1,0,0,0]), [0, 0, 1, 0, 0, 2, 0, 1, 2])
28
29     def test_next_state(self):
30         """
31         Test for getting next state given current state and a move
32         """
33         self.assertEqual(next_state([[1,2,1,2,1,2,1,0,0]], [1, 2]), [[1, 2, 1, 2, 1, 2, 1, 2, 0], [1, 2, 1, 2, 1, 2, 1, 2, 0]])
34
35     def test_remove_duplicates(self):
36         """
37         Test for removing duplicate states
38         """
```

```
tests.py x
34 self.assertEqual(next_state([[1,2,1,2,1,2,1,0,0]], [1, 2]), [[1, 2, 1, 2, 1, 2, 1, 2, 0], [1, 2, 1, 2, 1, 2, 1, 2, 0]])
35
36     def test_remove_duplicates(self):
37         """
38         Test for removing duplicate states
39         from states list after flip and rotation
40         """
41         self.assertEqual(remove_duplicates([[1,0,0,0,0,0,0,0,0], [0,0,1,0,0,0,0,0,0]]), [[1,0,0,0,0,0,0,0,0]])
42
43     def test_set_possible_moves(self):
44         """
45         Test for obtaining all possible moves
46         """
47         self.assertEqual(set_possible_moves([[1,0,0,0,0,0,0,0,0]], 2),
48             {(1, 0, 0, 0, 0, 0, 0, 0, 0):
49              [1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8]})
50
51     def test_get_required_action(self):
52         """
53         Test for getting required state given current state,
54         number of rotations and flips
55         """
56         self.assertEqual(get_required_action(0, 3, 0), 6)
57
58     def test_get_original_action(self):
59         """
60         Test for getting back the original state,
61         number of rotations and flips done and current state
62         """
63         self.assertEqual(get_original_action(6, 3, 0), 0)
64
65     def test_is_win(self):
66         """
67         Test for checking who won given a board state
68         """
69         self.assertEqual(is_win([1,2,2,2,1,1,2,2,1]), 1)
70
71     def test_match_two_states(self):
72         """
73         Test for matching two states index by index
74         """
75         self.assertEqual(match_two_states([1,2,2,0,0,1,0,0,0], [1,2,2,0,0,1,0,0,0]), True)
76
77     def test_check_equal(self):
78         """
79         Test to check whether two states are equal
80         after all possible flips and rotations
81         """
82         self.assertEqual(check_equal([1,0,0,0,0,0,0,0,0], [0,0,1,0,0,0,0,0,0]), (True, 3, 0))
83
84 if __name__ == '__main__':
85     unittest.main()
```

**test\_invariant (self)** : Test for invariant where the difference between the number of 'X' & 'O' should not be greater than 1

**test\_vlip (self)** : Test the vertical flip of a state

**test\_rotate90 (self)** : Test for rotate state by 90°

**test\_next\_state (self)** : Test whether the next state can be achieved from the current state after giving a defined move

**test\_remove\_duplicates (self)** : Test for excluding duplicates after performing rotational steps (flip/ mirror/ rotate)

**test\_set\_possible\_moves (self)** : Test for getting all possible moves for that state

**test\_get\_required\_action (self)** : Test for a particular index to get the position after a certain number of flips and rotations

**test\_get\_original\_action (self)** : Test for a particular index to get the position before certain number of flips and rotations

**test\_is\_win (self)** : Test for checking the winner by providing the final board

**test\_match\_two\_states (self)** : Test for validating whether two given states are equal or not

**test\_check\_equal (self)** : Test for validating whether two given states are equal or not after performing rotational steps (flip/ mirror/ rotate)

Testcase Output:

```
.....
-----
Ran 11 tests in 0.001s

OK
```

## Conclusion

- The accuracy of moves menace makes which leads to win or draw improves as we increase the number of our iterations while training the menace agent.
- Another factor on which the accuracy of menace depends is the start number of beads along with the change in the quantity of reward/ punishment beads (alpha, beta, gamma).
- If we want better results we can use an **epsilon-decreasing strategy** which is exploiting the epsilon instead of exploring and decreasing it until it becomes 0, As epsilon decreases our probability of menace winning increases

NOTE:  $\epsilon = 1 - \text{probability}$

## References

<https://towardsdatascience.com/lets-beat-games-using-a-bunch-of-code-part-1-tic-tac-toe-1543e981fec1>

<https://twice22.github.io/tictactoe/>

<http://incompleteideas.net/book/bookdraft2018mar16.pdf>