

# Program Structures and Algorithms

Spring 2024

Group Members: Siddharth Dumbre, Gaurav Gunjal

NUID: 002247119, 0022820367

GitHub Link: [https://github.com/gunjalga/INFO6205\\_MCTS](https://github.com/gunjalga/INFO6205_MCTS)

Task: TicTacToe and DotsAndBoxes game implementation using MCTS

## Abstract

This project explores the implementation of the Monte Carlo Tree Search (MCTS) algorithm in two classic 2-player games: Tic-Tac-Toe and Dots and Boxes. These games, known for their simplicity yet strategic depth, pose interesting challenges for strategic decision-making. In this study, we present an overview of both games and delve into the implementation details of MCTS to create game-playing agents capable of making strategic decisions.

Tic-Tac-Toe, a familiar game with a 3x3 grid, provides an ideal testing ground for exploring various decision-making techniques. Our implementation focuses on developing a player that employs MCTS to strategically explore the game tree, aiming to maximize its chances of winning.

Dots and Boxes played on a grid of dots, offer a more intricate strategic landscape. Here, our implementation harnesses the power of MCTS to navigate through possible moves, with the goal of constructing and capturing strategic boxes efficiently.

Monte Carlo Tree Search is a versatile algorithm that combines random simulation with tree-based exploration to make decisions in environments with large state spaces and complex decision trees. By leveraging MCTS, our implementations demonstrate adaptive and strategic gameplay, capable of providing engaging gaming experiences.

Through this project, we showcase the versatility of Monte Carlo Tree Search in developing agents for diverse gaming environments. Our implementations not only offer entertaining gameplay experiences but also serve as educational tools for understanding and exploring the potential of MCTS in game development and strategic decision-making.

## TicTacToe Implementation

Tic-Tac-Toe is a classic 2-player game played on a 3x3 grid. The objective is to place three markers of the same kind (traditionally X or O) in a row, column, or diagonal.

### Implementation Details:

Our implementation begins with the user providing the first move, which initializes the game board position. From this position, we create a state representation, encapsulating the current state of the game. This state is then used to create a node in the Monte Carlo Tree.

### Monte Carlo Tree Search:

The MCTS algorithm is employed to make strategic decisions for the computer player. The process starts with expanding the current node by considering all possible single moves available to the computer. These moves are represented as child nodes of the current node.

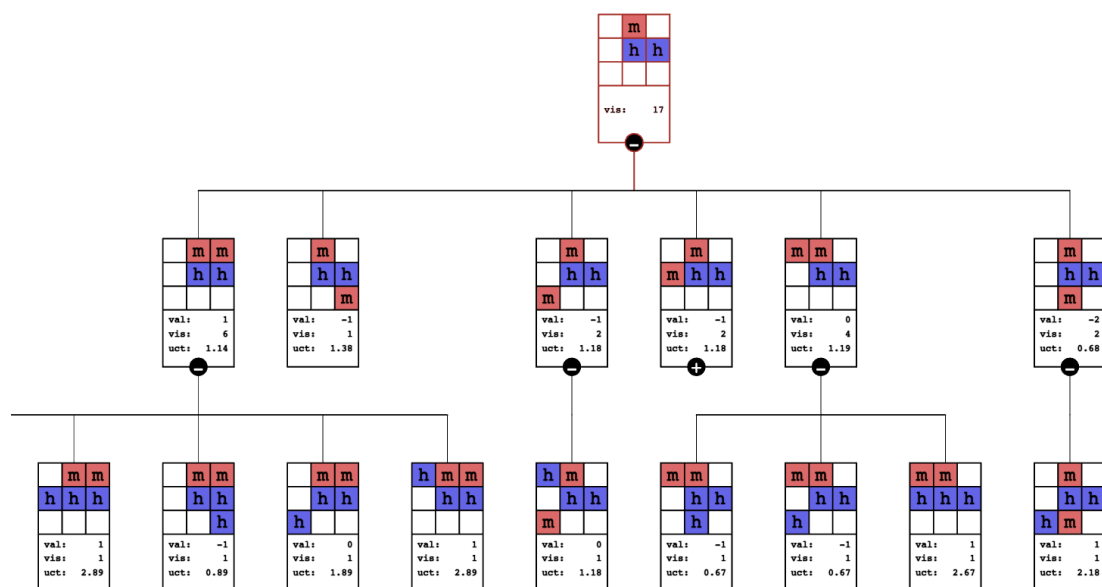
**Selection:** The algorithm begins at the root node of a tree and proceeds to select subsequent child nodes. The selection is based on optimizing a specific criterion, typically the node with the highest win rate. The selection process uses the Upper Confidence Bound (UCB) formula, commonly referred to as the UCT (Upper Confidence bounds applied to Trees) in tree search contexts. This method balances exploration of less visited nodes and exploitation of nodes known to perform well. The selection phase continues until a leaf node, which has not yet been fully expanded, is reached.

**Expansion:** Once a leaf node is identified during the selection phase, the MCTS algorithm expands this node by adding all potential future states originating from it. This expansion adds these new nodes to the tree, each representing a possible future move or state in the game.

**Simulation:** After expansion, the algorithm performs a simulation starting from the new nodes. This simulation involves playing out the game from the node's state to a game conclusion using random moves, a process often referred to as a "playout". This phase is designed to provide a quick, randomized evaluation of the node's potential without the need for deep computation.

**Backpropagation:** In the final phase, once the simulation reaches a conclusion (typically when the game ends), the results are used to update the tree. The algorithm backpropagates from the terminal state of the simulation up to the root node. During this process, it updates the statistics for each visited node, such as the win rate (if the simulation resulted in a win) and the visit count. This data helps in refining the selection strategy for future iterations of the algorithm.

MCTS iterates through these four phases repeatedly until a specified condition is met, such as a set number of iterations or a time limit, after which it selects the move leading to the node with the best performance metric as its decision.

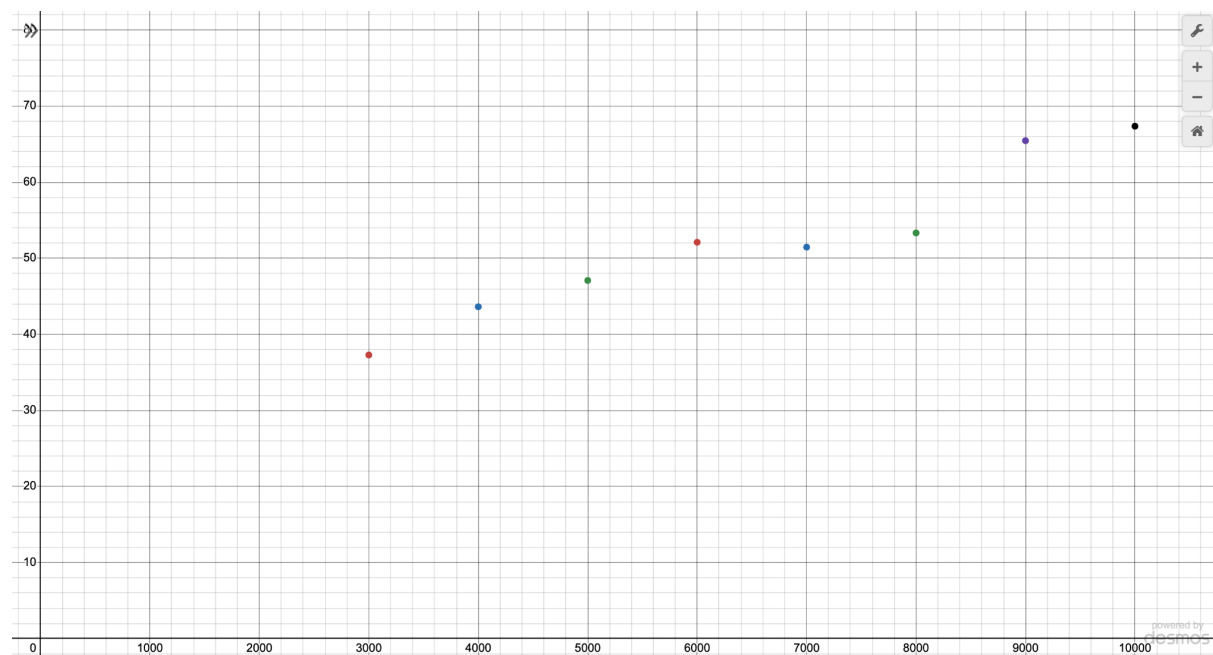


Tree formation based on the move made and all the available moves from that state

## Impact of number of iterations

Increasing the number of iterations in the Monte Carlo Tree Search (MCTS) algorithm for our Tic-Tac-Toe game profoundly impacts its strategic decision-making process. With more iterations, the algorithm exhibits enhanced exploration capabilities, thoroughly analyzing a larger portion of the game tree. This expanded exploration enables the algorithm to make more informed decisions by considering a greater number of potential moves and their consequences. Additionally, as the number of iterations rises, the algorithm refines its move selection process, leading to more accurate assessments of move values and improved decision-making. Furthermore, the increased number of iterations allows the algorithm to accumulate more experience through simulated gameplay, facilitating enhanced learning and adaptation over time. As a result, increasing the number of iterations in the MCTS algorithm leads to improved performance, stronger gameplay, and convergence towards optimal decision-making strategies. However, it's important to note that this enhanced performance comes at the cost of increased computational time, as reflected in the timing comparisons provided in the accompanying table.

Number of iterations	Time taken
3000	37.29
4000	43.625
5000	47.08
6000	52.10275
7000	51.463
8000	53.33
9000	65.45279
10000	67.36






















































Time taken for a move(expansion+simulation+backpropagation) vs Number of Iterations

## Test Cases

```
1 package edu.neu.coe.info6205.mcts.tictactoe;
2
3 > import ...
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1
```

```
1 package edu.neu.coe.info6205.mcts.tictactoe;
2
3 > import ...
4
5
6
7
8
9
10
11 public class TicTacToeTest {
12
13     /**
14      *
15      */
16     @Test
17     public void runGame() {
18         long seed = 0L;
19         TicTacToe target = new TicTacToe(seed); // games run here will all be deterministic.
20         State<TicTacToe> state = target.runGame();
21     }
22 }
```

Run  TicTacToeTest x

✓ TicTacToeTest (edu.neu.coe) 5ms ✓ Tests passed: 1 of 1 test - 5 ms

✓ runGame 5ms

/Users/siddharth/Library/Java/JavaVirtualMachines/openjdk-21.0.1/Contents/Home/bin/java ...

-1,-1,-1

-1,-1,-1

-1,-1,-1

Process finished with exit code 0



## Dots and Boxes Implementation

Data structure: We are using an  $n*n$  grid of Box objects, each of these box objects will have 4 booleans left, right, top, and bottom to represent the edges of a box, and x, and y variables to help us determine the position of the box on the grid.

This  $n*n$  grid is stored in a BoxPosition object which contains methods for operations on the grid like making a move, getting all the possible moves from a position, checking if the grid is full, getting a winner, etc.

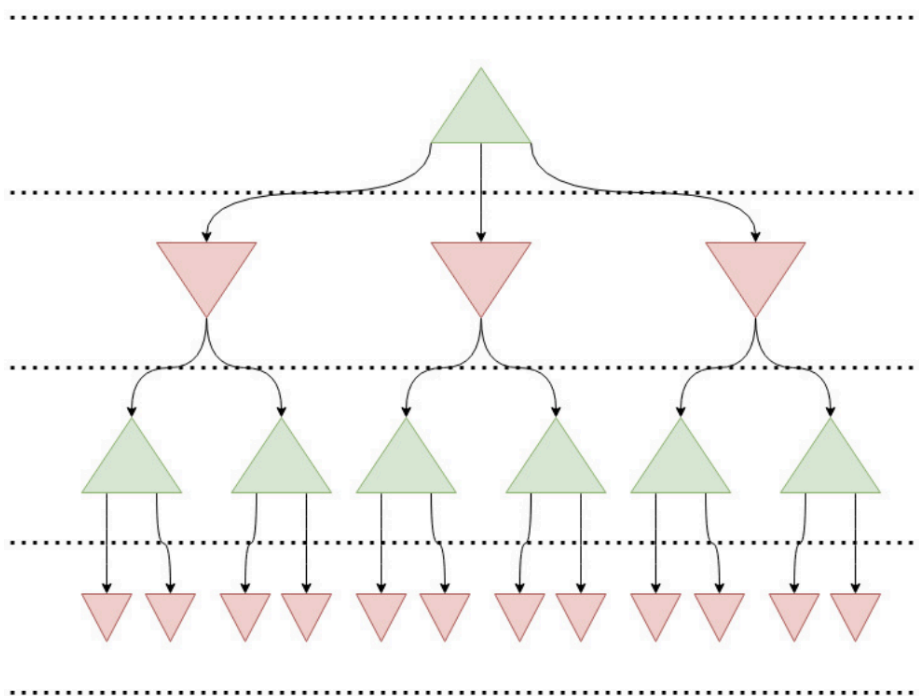
This data structure might be very simple to understand but it has its drawbacks

Keeping in mind that we cannot use the same references of box objects in different nodes as it would cause a clash of operations on the same references leading to the failure of the MCTS algorithm, we created a copy grid method which gives a new instance of every single box object in the position.

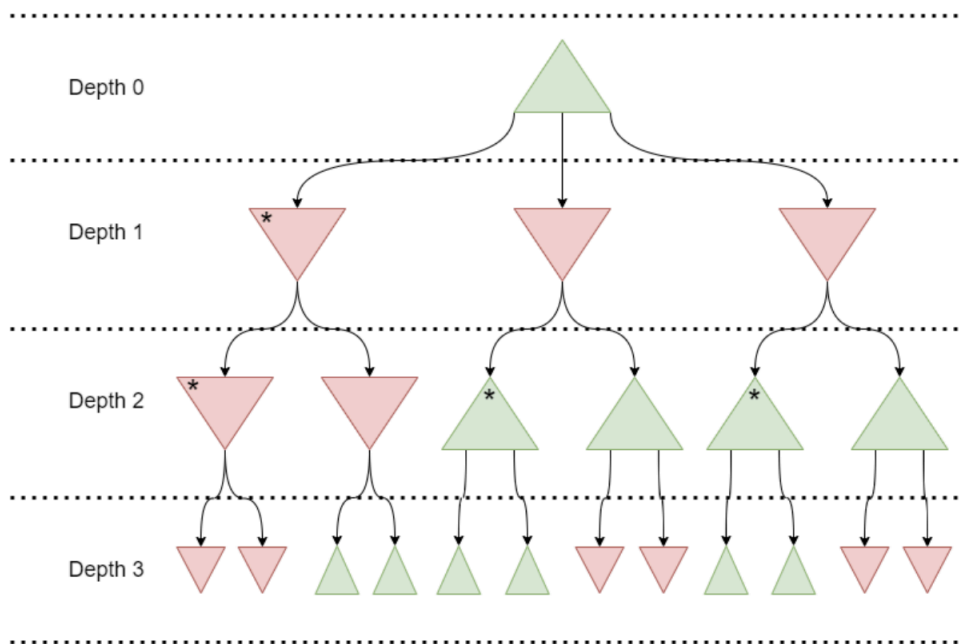
While making a move we need to handle the booleans of two neighboring box objects to make sure everything on the grid is consistent.

Moves: Unlike the Tic-tac-toe move, here we need to take three parameters from the user that is the x,y coordinate of the box and the direction(which will help us determine which edge in x,y box)

MCTS for Dots and Boxes: In the MCTS implementation for dots and boxes the selection, expansion, and simulation phase are quite similar to the tic-tac-toe game but the backpropagation phase is different which makes the algorithm quite complex. As we know in a tic-tac-toe game the players take turns in one after the other fashion but in the dots and boxes, a player can get a chain of turns if he continues to capture a box. So we have to keep track of the player to maximize at each node and the nodes below that node. In games like tic-tac-toe, we move in min-max-min-max fashion but here we might have to change to order to let's say min-max-max-max-min if a player captures 3 boxes in 3 consecutive moves.



### Min-max for tic-tac-toe



Min-max for dots and boxes. \* represents a state where box is captured

Finding balance between exploration and exploitation terms:

This is the formula to calculate UCT score for a particular node. There are two terms in this formula. The left hand side term is the exploitation term and the right hand side term is the exploration term. And  $c$  is UCB constant which will determine how much the algorithm will explore from all available nodes at a level in the tree.

$$\frac{w_i}{n_i} + c \sqrt{\frac{\ln N_i}{n_i}}$$

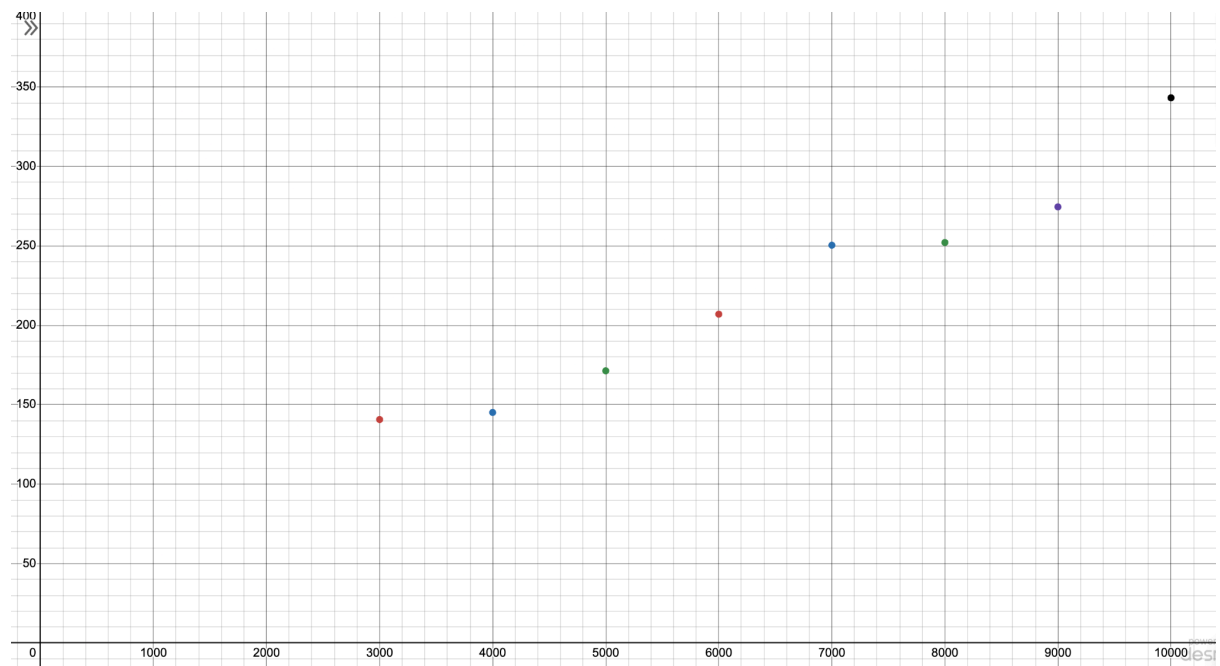
- $w_i$  stands for the number of wins for the node considered after the  $i$ -th move
- $n_i$  stands for the number of simulations for the node considered after the  $i$ -th move
- $N_i$  stands for the total number of simulations after the  $i$ -th move run by the parent node of the one considered
- $c$  is the exploration parameter—theoretically equal to  $\sqrt{2}$ ; in practice usually chosen empirically

In a 3\*3 grid for dots and boxes we have 9 total boxes and 24 different edges, the total nodes after a single move are around 4048. To ensure that the algorithm doesn't converge on one single node as the tree below that node reaches its leaf levels, we need to increase the UCB constant so that our algorithm goes down each path and tries to find an optimal path. Also we need to keep the number of iterations on the higher side for a game like dots and boxes as the number of possible nodes from one node are quite large.

What else could we do to optimise the algorithm? Instead of assigning the nodes with +1 or -1 for winning and losing simulation, could we just use the number of boxes captured as the value for wins? Well if you think about it carefully it doesn't really have any effect on the algorithm at all because it

will do the same for the levels of the opposite player as well. This makes it almost the same as adding just +1 and -1 to the nodes.

No of Iterations	Time taken(ms)
3000	140.7242
4000	145.19
5000	171.422
6000	207.0541
7000	250.48
8000	252.18
9000	274.61
10000	343.27



Time taken for a move(expansion+simulation+backpropagation+selection) vs Number of iterations



## Test Cases

```
1 package edu.neu.coe.info6205.mcts.DotsAndBoxes;
2
3 > import ...
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030

```

```
1 package edu.neu.coe.info6205.mcts.DotsAndBoxes;
2
3 import edu.neu.coe.info6205.mcts.core.State;
4 import edu.neu.coe.info6205.mcts.dotsandboxes.DotsAndBoxes;
5 import edu.neu.coe.info6205.mcts.tictactoe.TicTacToe;
6 import org.junit.Test;
7
8 import java.util.Optional;
9
10 import static org.junit.Assert.assertEquals;
11 import static org.junit.Assert.fail;
12
13 // Gaurav Popat Gunjal +1
14 public class DotsAndBoxesTest {
15
16     // Gaurav Ponnat Gunjal
17
18     @Test
19     public void runGame() {
20         TicTacToe ttt = new TicTacToe(3, 3);
21         State s = ttt.newGame();
22         assertEquals("Initial game state", s, ttt.getState());
23         DotsAndBoxes db = new DotsAndBoxes(s);
24         assertEquals("Initial game state", db.getState(), s);
25         db.move(0, 0, 1);
26         assertEquals("After move 0,0,1", db.getState(), ttt.getState());
27         db.move(1, 1, 1);
28         assertEquals("After move 1,1,1", db.getState(), ttt.getState());
29         db.move(2, 2, 1);
30         assertEquals("After move 2,2,1", db.getState(), ttt.getState());
31         db.move(0, 1, 2);
32         assertEquals("After move 0,1,2", db.getState(), ttt.getState());
33         db.move(1, 2, 2);
34         assertEquals("After move 1,2,2", db.getState(), ttt.getState());
35         db.move(2, 1, 2);
36         assertEquals("After move 2,1,2", db.getState(), ttt.getState());
37         db.move(0, 2, 2);
38         assertEquals("After move 0,2,2", db.getState(), ttt.getState());
39         db.move(1, 0, 2);
40         assertEquals("After move 1,0,2", db.getState(), ttt.getState());
41         db.move(2, 0, 2);
42         assertEquals("After move 2,0,2", db.getState(), ttt.getState());
43         db.move(0, 0, 2);
44         assertEquals("After move 0,0,2", db.getState(), ttt.getState());
45         db.move(1, 0, 2);
46         assertEquals("After move 1,0,2", db.getState(), ttt.getState());
47         db.move(2, 0, 2);
48         assertEquals("After move 2,0,2", db.getState(), ttt.getState());
49         db.move(0, 1, 2);
50         assertEquals("After move 0,1,2", db.getState(), ttt.getState());
51         db.move(1, 1, 2);
52         assertEquals("After move 1,1,2", db.getState(), ttt.getState());
53         db.move(2, 1, 2);
54         assertEquals("After move 2,1,2", db.getState(), ttt.getState());
55         db.move(0, 2, 2);
56         assertEquals("After move 0,2,2", db.getState(), ttt.getState());
57         db.move(1, 2, 2);
58         assertEquals("After move 1,2,2", db.getState(), ttt.getState());
59         db.move(2, 2, 2);
60         assertEquals("After move 2,2,2", db.getState(), ttt.getState());
61         db.move(0, 0, 2);
62         assertEquals("After move 0,0,2", db.getState(), ttt.getState());
63         db.move(1, 0, 2);
64         assertEquals("After move 1,0,2", db.getState(), ttt.getState());
65         db.move(2, 0, 2);
66         assertEquals("After move 2,0,2", db.getState(), ttt.getState());
67         db.move(0, 1, 2);
68         assertEquals("After move 0,1,2", db.getState(), ttt.getState());
69         db.move(1, 1, 2);
70         assertEquals("After move 1,1,2", db.getState(), ttt.getState());
71         db.move(2, 1, 2);
72         assertEquals("After move 2,1,2", db.getState(), ttt.getState());
73         db.move(0, 2, 2);
74         assertEquals("After move 0,2,2", db.getState(), ttt.getState());
75         db.move(1, 2, 2);
76         assertEquals("After move 1,2,2", db.getState(), ttt.getState());
77         db.move(2, 2, 2);
78         assertEquals("After move 2,2,2", db.getState(), ttt.getState());
79         db.move(0, 0, 2);
80         assertEquals("After move 0,0,2", db.getState(), ttt.getState());
81         db.move(1, 0, 2);
82         assertEquals("After move 1,0,2", db.getState(), ttt.getState());
83         db.move(2, 0, 2);
84         assertEquals("After move 2,0,2", db.getState(), ttt.getState());
85         db.move(0, 1, 2);
86         assertEquals("After move 0,1,2", db.getState(), ttt.getState());
87         db.move(1, 1, 2);
88         assertEquals("After move 1,1,2", db.getState(), ttt.getState());
89         db.move(2, 1, 2);
90         assertEquals("After move 2,1,2", db.getState(), ttt.getState());
91         db.move(0, 2, 2);
92         assertEquals("After move 0,2,2", db.getState(), ttt.getState());
93         db.move(1, 2, 2);
94         assertEquals("After move 1,2,2", db.getState(), ttt.getState());
95         db.move(2, 2, 2);
96         assertEquals("After move 2,2,2", db.getState(), ttt.getState());
97         db.move(0, 0, 2);
98         assertEquals("After move 0,0,2", db.getState(), ttt.getState());
99         db.move(1, 0, 2);
100         assertEquals("After move 1,0,2", db.getState(), ttt.getState());
101         db.move(2, 0, 2);
102         assertEquals("After move 2,0,2", db.getState(), ttt.getState());
103         db.move(0, 1, 2);
104         assertEquals("After move 0,1,2", db.getState(), ttt.getState());
105         db.move(1, 1, 2);
106         assertEquals("After move 1,1,2", db.getState(), ttt.getState());
107         db.move(2, 1, 2);
108         assertEquals("After move 2,1,2", db.getState(), ttt.getState());
109         db.move(0, 2, 2);
110         assertEquals("After move 0,2,2", db.getState(), ttt.getState());
111         db.move(1, 2, 2);
112         assertEquals("After move 1,2,2", db.getState(), ttt.getState());
113         db.move(2, 2, 2);
114         assertEquals("After move 2,2,2", db.getState(), ttt.getState());
115         db.move(0, 0, 2);
116         assertEquals("After move 0,0,2", db.getState(), ttt.getState());
117         db.move(1, 0, 2);
118         assertEquals("After move 1,0,2", db.getState(), ttt.getState());
119         db.move(2, 0, 2);
120         assertEquals("After move 2,0,2", db.getState(), ttt.getState());
121         db.move(0, 1, 2);
122         assertEquals("After move 0,1,2", db.getState(), ttt.getState());
123         db.move(1, 1, 2);
124         assertEquals("After move 1,1,2", db.getState(), ttt.getState());
125         db.move(2, 1, 2);
126         assertEquals("After move 2,1,2", db.getState(), ttt.getState());
127         db.move(0, 2, 2);
128         assertEquals("After move 0,2,2", db.getState(), ttt.getState());
129         db.move(1, 2, 2);
130         assertEquals("After move 1,2,2", db.getState(), ttt.getState());
131         db.move(2, 2, 2);
132         assertEquals("After move 2,2,2", db.getState(), ttt.getState());
133         db.move(0, 0, 2);
134         assertEquals("After move 0,0,2", db.getState(), ttt.getState());
135         db.move(1, 0, 2);
136         assertEquals("After move 1,0,2", db.getState(), ttt.getState());
137         db.move(2, 0, 2);
138         assertEquals("After move 2,0,2", db.getState(), ttt.getState());
139         db.move(0, 1, 2);
140         assertEquals("After move 0,1,2", db.getState(), ttt.getState());
141         db.move(1, 1, 2);
142         assertEquals("After move 1,1,2", db.getState(), ttt.getState());
143         db.move(2, 1, 2);
144         assertEquals("After move 2,1,2", db.getState(), ttt.getState());
145         db.move(0, 2, 2);
146         assertEquals("After move 0,2,2", db.getState(), ttt.getState());
147         db.move(1, 2, 2);
148         assertEquals("After move 1,2,2", db.getState(), ttt.getState());
149         db.move(2, 2, 2);
150         assertEquals("After move 2,2,2", db.getState(), ttt.getState());
151         db.move(0, 0, 2);
152         assertEquals("After move 0,0,2", db.getState(), ttt.getState());
153         db.move(1, 0, 2);
154         assertEquals("After move 1,0,2", db.getState(), ttt.getState());
155         db.move(2, 0, 2);
156         assertEquals("After move 2,0,2", db.getState(), ttt.getState());
157         db.move(0, 1, 2);
158         assertEquals("After move 0,1,2", db.getState(), ttt.getState());
159         db.move(1, 1, 2);
160         assertEquals("After move 1,1,2", db.getState(), ttt.getState());
161         db.move(2, 1, 2);
162         assertEquals("After move 2,1,2", db.getState(), ttt.getState());
163         db.move(0, 2, 2);
164         assertEquals("After move 0,2,2", db.getState(), ttt.getState());
165         db.move(1, 2, 2);
166         assertEquals("After move 1,2,2", db.getState(), ttt.getState());
167         db.move(2, 2, 2);
168         assertEquals("After move 2,2,2", db.getState(), ttt.getState());
169         db.move(0, 0, 2);
170         assertEquals("After move 0,0,2", db.getState(), ttt.getState());
171         db.move(1, 0, 2);
172         assertEquals("After move 1,0,2", db.getState(), ttt.getState());
173         db.move(2, 0, 2);
174         assertEquals("After move 2,0,2", db.getState(), ttt.getState());
175         db.move(0, 1, 2);
176         assertEquals("After move 0,1,2", db.getState(), ttt.getState());
177         db.move(1, 1, 2);
178         assertEquals("After move 1,1,2", db.getState(), ttt.getState());
179         db.move(2, 1, 2);
180         assertEquals("After move 2,1,2", db.getState(), ttt.getState());
181         db.move(0, 2, 2);
182         assertEquals("After move 0,2,2", db.getState(), ttt.getState());
183         db.move(1, 2, 2);
184         assertEquals("After move 1,2,2", db.getState(), ttt.getState());
185         db.move(2, 2, 2);
186         assertEquals("After move 2,2,2", db.getState(), ttt.getState());
187         db.move(0, 0, 2);
188         assertEquals("After move 0,0,2", db.getState(), ttt.getState());
189         db.move(1, 0, 2);
190         assertEquals("After move 1,0,2", db.getState(), ttt.getState());
191         db.move(2, 0, 2);
192         assertEquals("After move 2,0,2", db.getState(), ttt.getState());
193         db.move(0, 1, 2);
194         assertEquals("After move 0,1,2", db.getState(), ttt.getState());
195         db.move(1, 1, 2);
196         assertEquals("After move 1,1,2", db.getState(), ttt.getState());
197         db.move(2, 1, 2);
198         assertEquals("After move 2,1,2", db.getState(), ttt.getState());
199         db.move(0, 2, 2);
200         assertEquals("After move 0,2,2", db.getState(), ttt.getState());
201         db.move(1, 2, 2);
202         assertEquals("After move 1,2,2", db.getState(), ttt.getState());
203         db.move(2, 2, 2);
204         assertEquals("After move 2,2,2", db.getState(), ttt.getState());
205         db.move(0, 0, 2);
206         assertEquals("After move 0,0,2", db.getState(), ttt.getState());
207         db.move(1, 0, 2);
208         assertEquals("After move 1,0,2", db.getState(), ttt.getState());
209         db.move(2, 0, 2);
210         assertEquals("After move 2,0,2", db.getState(), ttt.getState());
211         db.move(0, 1, 2);
212         assertEquals("After move 0,1,2", db.getState(), ttt.getState());
213         db.move(1, 1, 2);
214         assertEquals("After move 1,1,2", db.getState(), ttt.getState());
215         db.move(2, 1, 2);
216         assertEquals("After move 2,1,2", db.getState(), ttt.getState());
217         db.move(0, 2, 2);
218         assertEquals("After move 0,2,2", db.getState(), ttt.getState());
219         db.move(1, 2, 2);
220         assertEquals("After move 1,2,2", db.getState(), ttt.getState());
221         db.move(2, 2, 2);
222         assertEquals("After move 2,2,2", db.getState(), ttt.getState());
223         db.move(0, 0,
```

```
1 package edu.neu.coe.info6205.mcts.DotsAndBoxes;
2
3 import edu.neu.coe.info6205.mcts.core.State;
4 import edu.neu.coe.info6205.mcts.dotsandboxes.BoxPosition;
5 import edu.neu.coe.info6205.mcts.dotsandboxes.DotsAndBoxes;
6 import edu.neu.coe.info6205.mcts.dotsandboxes.DotsAndBoxesNode;
7 import edu.neu.coe.info6205.mcts.tictactoe.TicTacToeNode;
8 import io.cucumber.java.en.Do;
9 import org.junit.Test;
10
11 import static org.junit.Assert.assertEquals;
12 import static org.junit.Assert.assertTrue;
13
14 // siddharth_dumbre
15 public class DotsAndBoxesNodeTest {
16     // siddharth_dumbre
17
18     @Test
19     @Do
20     public void addChild() {
21         // ...
22     }
23
24     @Test
25     @Do
26     public void state() {
27         // ...
28     }
29
30     @Test
31     @Do
32     public void children() {
33         // ...
34     }
35
36     @Test
37     @Do
38     public void winsAndPlayouts() {
39         // ...
40     }
41 }
```

Run DotsAndBoxesNodeTest x

Tests passed: 4 of 4 tests - 9 ms

Test	Time
addChild	7 ms
state	1 ms
children	1 ms
winsAndPlayouts	0 ms

Process finished with exit code 0

## Conclusion

After implementation of the MCTS algorithm for the TicTacToe and DotsAndBoxes games, it was observed that the time taken for the computer to make a move in the DotsAndBoxes game was much higher compared to TicTacToe. The reason for this is that the number of possible moves in the DotsAndBoxes game is much higher than tictactoe because it comprises all the edges and if a box is captured then the same player gets a new chance. For this reason, the computer evaluates the most optimal move based on the consecutive and non-consecutive moves depending on whether the box is captured by that move or not. Also, it is observed that as the number of iterations increases by a step of 1000 then the timing for the computer to make a move increases. Increasing the number of iterations in the Monte Carlo Tree Search algorithm increases computational time because each iteration involves additional simulations and evaluations of potential moves, leading to a proportional increase in the overall computational workload.

## References

<https://vgarciasc.github.io/mcts-viz/>  
<https://www.youtube.com/watch?v=UXW2yZndI7U&t=412s>  
[https://pats.cs.cf.ac.uk/@archive\\_file?p=1686&n=final&f=1-report.pdf&SIG=4eec909fce4fb2da94c4b05d1906f909face6aa1dc12dd5def36b33ff165eace](https://pats.cs.cf.ac.uk/@archive_file?p=1686&n=final&f=1-report.pdf&SIG=4eec909fce4fb2da94c4b05d1906f909face6aa1dc12dd5def36b33ff165eace)