

Program Structures and Algorithms
Spring 2025

NAME: Mingrui Yuan, Yuanchang Lee, Ronghao Yuan
NUID: 002089150, 002310611, 002086973
GITHUB LINK: <https://github.com/info6206final/finalProject6205>

Executive Summary

For this project, our team implemented a basic Monte Carlo Tree Search (MCTS) agent to play the game of Go. The idea was to create an AI that can play against a human player using the MCTS algorithm and integrate it into a graphical Go game. We used Java and added our code into the existing project structure, combining game logic with a user interface built using Swing. The final result is a playable Go game where a human can compete against an AI opponent. The AI calculates moves using MCTS and responds right after the player's move.

Introduction to The Game of Go

Go is a two-player board game that dates back over 2,500 years. It's usually played on a 19×19 grid—smaller 9×9 or 13×13 boards are common when learning the game.

Players take turns placing black or white stones on empty intersections. A stone or a connected group of stones is captured and removed if all its adjacent empty points (liberties) are occupied by the opponent.

The goal is to control more territory than your opponent. Territory consists of empty intersections you've surrounded with your stones. When both players pass in succession, play ends. Each scores by counting controlled territory plus captured stones; the “Ko” rule prevents infinite repetition by forbidding a move that would recreate the previous board position.

Project Structure

We built our project under this package:
`com.phasmidsoftware.dsaipg.projects.mcts.gogame.src`

Main files we worked on:

1. **Chess_B.java**: A class for storing individual piece data.
2. **ChessBoardPanel_B.java**: Handles game logic, the board state, and all the rules.
3. **GoGameGUI_B.java**: A graphical interface for interacting with the game.
4. **MCSTAgent.java**: Our main Monte Carlo Tree Search implementation.
5. **Memoryunit_B.java**: A helper class for storing board history (used for undo moves).

Go Game Implementation

We built a playable Go game using Java Swing. The board is 19x19, and users can place stones by clicking on the board. The game handles basic rules like:

- Valid move detection
- Capturing stones based on liberties
- Group formation and life-death detection

Key Implementation Details

1.State

Encapsulates the board (19×19 array) and current player.

- `getLegalMoves()` returns all empty positions.
- `applyMove()` clones the board, plays a stone, and switches turn.
- `evaluate()` compares black vs. white stone counts.

2.TreeNode

Holds a State, pointers to parent/children, counts (visits, wins), and untried moves.

- `getUCTValue()` computes the UCT score.
- `selectChild()` picks the child with maximum UCT.

3.Simulation Depth & Iterations

- **ITERATION_LIMIT** = 4000 playouts
- **MAX_SIMULATION_DEPTH** = 100 moves

Go's true win/loss determination and endgame require intricate territory scoring, capture tracking, and proper handling of ko fights. Implementing a full, rules-compliant scoring engine is a major undertaking. This project uses only a stone-count heuristic (black vs. white stones) and does *not* address ko rule enforcement or advanced endgame evaluation. As a result, the AI's playing strength remains very limited.

Algorithm Design and Implementation

In this project, we used the Monte Carlo Tree Search (MCTS) algorithm to build an AI that can play Go against a human. Since Go is a complex game, especially on a 19x19 board, we decided to start with a simplified version of the rules — we focus only on placing stones, capturing, and counting pieces at the end. We didn't include things like Ko or full territory scoring yet.

How the MCTS Agent Works

The idea behind MCTS is to simulate lots of random games and see which moves lead to better outcomes. For each AI move, the program goes through four steps:

1. **Selection:** Starting from the current board state, the algorithm selects child nodes using the UCT (Upper Confidence Bound for Trees) formula to balance exploration and exploitation. It continues down the tree until it reaches a node that hasn't been fully explored.

$$\text{UCT} = \frac{\text{wins}}{\text{visits}} + C \sqrt{\frac{\ln(\text{parent.visits})}{\text{visits}}}$$

2. **Expansion:** When it finds a move it hasn't tried yet, it adds it to the tree.
3. **Simulation:** From that new move, it plays out a random game until either the board is full or it reaches a maximum depth (we set it to 100 moves). It counts how many black and white pieces are on the board to see who would be ahead.
4. **Backpropagation:** The result of that simulation (black win, white win, or draw) is sent back up the tree, and all the nodes that led to that move get updated.

After running this process 4000 times for each AI move, the AI picks the move that was the most successful during the simulations.

Game State and Move Logic

The board is represented as a **19x19** integer matrix. We use:

- 0 for empty spaces,

- 1 for black stones,
- -1 for white stones.

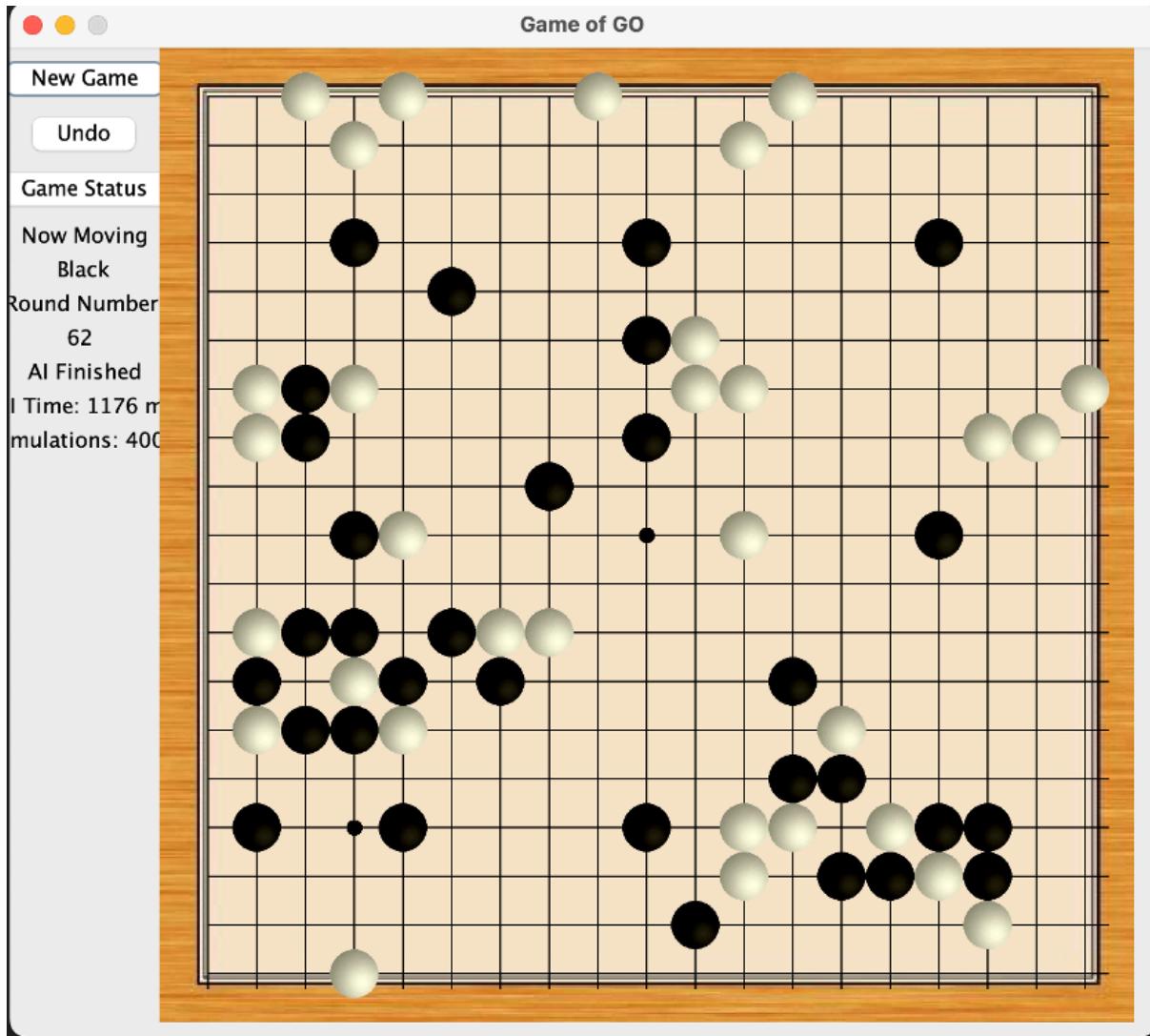
Each move creates a new board state and switches turns. For performance reasons, we simplified the simulation phase by skipping full liberty/capture checking, and instead just placing stones randomly. This makes it possible to run many simulations quickly while still getting reasonable results.

Parameters We Used

- UCT exploration constant: **1.41**
- Simulations per move: **4000**
- Max moves per simulation: **100**

How AI Plays the Game

We modified the GUI code so that when the player makes a move, the AI will immediately take its turn using the MCTS algorithm. The steps are: Get the current board state. Run MCSTAgent.nextMove() to find the best move. Use playChessAI() to make the move on the board. Update the interface with the new move.



Test Cases

In the “GoGameTestCases.java” class, we have several test cases in order to test the functionality of our game. To be more specific, we used `setUp()` to initialize a new game, resetting the board and step count. `TestNewGameState1()` calls `newGame()`, and then verifies whether the initial state of the game is correct. The `testInvalidMove()` class tries to access an invalid location of the board that should be returning a 0. And the `testMoveSurround()` class tests an intentionally suicidal move where it would be surrounded by the opponent. The expected output of `testMoveSurround()` is a -1. Last but not least, `testCapture()` is trying to surround a piece to capture it. Then we verify if the piece at position [1][1] becomes 0.

Code Screenshot

This screenshot shows a Java code editor with the file `Chess_B.java` open. The code implements a `Serializable` interface and defines a class `Chess_B` with various methods for managing piece coordinates and liberties. The editor interface includes tabs for other files like `MCTS.java`, `GoGameGUI_B.java`, and `ChessBoardPanel_B.java`. A status bar at the bottom shows the file path and current time.

```
public class Chess_B implements Serializable {
    private int Color = 0; // Color of the piece (1 = Black, -1 = White, 0 = Empty)
    private int x_Map=1,y_Map=-1; // Coordinates on the board (index in board_array), -1 = not placed
    private int x_Point, y_Point; // Pixel coordinates for drawing
    private Vector vCurrentGroup=null; // The group this stone currently belongs to
    public Vector getCurrentGroup() { return vCurrentGroup; }

    public void setvCurrentGroup(Vector vBelongto) { this.vCurrentGroup = vBelongto; }

    private int Qi=4; // Number of liberties (initially 4)
    private int alive=0; // Life status: 0 = dead, 1 = alive (optional usage)

    public int getX_Map() { return x_Map; }

    public void setX_Map(int x_Map) { this.x_Map = x_Map; }

    public int getY_Map() { return y_Map; }

    public void setY_Map(int y_Map) { this.y_Map = y_Map; }

    Chess_B() { no usages }
    Chess_B(int x, int y) { 1 usage
        this.x_Point = x;
        this.y_Point = y;
        //this.vBelongto=new Vector();
    }

    Chess_B(int x, int y, int qi) { no usages
        this.x_Point = x;
    }
}
```

(Chess_B.java)

This screenshot shows a Java code editor with the file `ChessBoardPanel_B.java` open. The code extends `JPanel` and contains numerous fields for managing a chessboard, including vectors for memory, coordinates, and piece counts. It also includes methods for determining the winner and handling turn flags. The editor interface includes tabs for other files like `MCTS.java`, `GoGameGUI_B.java`, and `Chess_B.java`. A status bar at the bottom shows the file path and current time.

```
public class ChessBoardPanel_B extends JPanel {
    private Vector MapMemory = new Vector(); // Memory array for storing states
    private Vector vAllGroup = new Vector(); // All groups
    private Vector CurrentRemovedGroup = new Vector(); // Currently removed neighbor group

    private int[][] ChessMap; // Board coordinates
    private Chess_B[][] ChessPoint = new Chess_B[ROWS][ROWS]; // Board coordinate objects
    private int blackSize = 0, whiteSize = 0; // Black and white territory counts

    private int step = 0; // Step counter
    private int Turnflag = 1; // Turn flag
    private int CurrentMapPoint_X, CurrentMapPoint_Y; // Mouse-click grid position, -1 = invalid

    public int Winner() { //AY
        blackSize=0;
        whiteSize=0;
    }
}
```

(ChessBoardPanel_B.java)

The screenshot shows a Java code editor with the file `GoGameGUI_B.java` open. The code implements a GUI for a Go game. It creates a main frame, initializes buttons and labels for game status, and adds them to a side panel. The code uses standard Java Swing components like `JFrame`, `JButton`, and `JLabel`.

```
15
16  public class GoGameGUI_B implements ActionListener { AY +2
17      private ChessBoardPanel_B CBPanel; 19 usages
18      JButton bt_Back; 4 usages
19      JButton bt_newGame; 4 usages
20      JButton bt_Win; 4 usages
21      JLabel jl_Turn1,jl_Turn2,jl_Step1,jl_Step2,jl_Message1,jl_Message2; 2 usages
22
23      GoGameGUI_B() { 1 usage AY +2
24          JFrame jf = new JFrame( title: "Game of GO");
25          // Create game_panel
26          this.CBPanel =new ChessBoardPanel_B();
27          // Initialize buttons and labels
28          bt_newGame = new JButton( text: "New Game");
29          bt_Back = new JButton( text: "Undo");
30          bt_Win=new JButton( text: "Game Status");
31          jl_Turn1=new JLabel( text: "Now Moving");
32          jl_Turn2=new JLabel( text: "Black");
33          jl_Step1=new JLabel( text: "Round Number\n");
34          jl_Step2=new JLabel( text: "0");
35          jl_Message1=new JLabel( text: "");
36          jl_Message2=new JLabel( text: "");
37
38          // Side panel
39          JPanel jp = new JPanel();
40          //jp.setLayout(new GridLayout(2, 1, 3, 3));
41          jp.setPreferredSize(new Dimension( width: 92, height: 600));
42
43          //jp.setLayout(new GridLayout(8,1,5,10));
44          jp.add(bt_newGame);
45          jp.add(bt_Back);
46          jp.add(bt_Win);
47          jp.add(jl_Turn1);
```

(GoGameGUI_B.java)

The screenshot shows a Java code editor with the file `MCTS.java` open. The code implements the Monte Carlo Tree Search algorithm for a Tic Tac Toe game. It includes methods for initializing the search tree, performing iterations, simulating moves, and determining the best move based on win rates and playout counts.

```
12  public class MCTS { AY
13
14      public static void main(String[] args) { AY
15          MCTS mcts = new MCTS(new TicTacToeNode(new TicTacToe().new TicTacToeState()));
16          Node<TicTacToe> root = mcts.root;
17
18          int iterations = 1000;
19
20          for (int i = 0; i < iterations; i++) {
21              simulate(root);
22          }
23
24
25          for (Node<TicTacToe> child : root.children()) {
26              System.out.println("Move:\n" + child.state());
27              System.out.printf("Wins: %d, Playouts: %d, Win Rate: %.2f\n",
28                  child.wins(), child.playouts(), child.playouts() > 0 ?
29                      (double) child.wins() / (2 * child.playouts()) : 0.0);
30          }
31
32
33          Node<TicTacToe> best = root.children().stream() Stream<Node<...>>
34              .max(( Node<TicTacToe> a, Node<TicTacToe> b) -> Double.compare(
35                  (double) a.wins() / a.playouts(),
36                  (double) b.wins() / b.playouts())) Optional<Node<...>>
37              .orElse( other: null);
38
39          if (best != null) {
40              System.out.println("Best move chosen:");
41              System.out.println(best.state());
42          } else {
43              System.out.println("No best move found.");
44          }
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
236
237
237
238
239
239
240
241
242
243
244
245
245
246
247
247
248
249
249
250
251
252
253
254
255
255
256
257
257
258
259
259
260
261
262
263
264
265
265
266
267
267
268
269
269
270
271
272
273
274
275
275
276
277
277
278
279
279
280
281
282
283
284
285
285
286
287
287
288
289
289
290
291
292
293
294
295
295
296
297
297
298
299
299
300
301
302
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
13
```

```

7  public class MCSTAgent { 1 usage  ↳ AY +1
8
9
10 private static final int ITERATION_LIMIT = 4000; // Number of simulation iterations 1 usage
11 private static final int MAX_SIMULATION_DEPTH = 100; // Maximum depth of each simulation 1 usage
12 private static final double UCT_CONSTANT = 1.41; // UCT exploration constant 1 usage
...
13 // Represents a game state: the board and the current player (1 for black, -1 for white)
14 public static class State {  ↳ AY +1
15     int[][] board;  9 usages
16     int turn;  3 usages
17
18     public State(int[][] board, int turn) {  ↳ AY
19         this.board = cloneBoard(board);
20         this.turn = turn;
21     }
22
23     @
24     private int[][] cloneBoard(int[][] board) {  2 usages  ↳ AY
25         int rows = board.length;
26         int cols = board[0].length;
27         int[][] newBoard = new int[rows][cols];
28         for (int i = 0; i < rows; i++) {
29             System.arraycopy(board[i], 0, newBoard[i], 0, cols);
30         }
31         return newBoard;
32     }
33
34     // Get all legal moves: positions with 0 (empty)
35     public List<int[]> getLegalMoves() {  4 usages  ↳ AY
36         List<int[]> moves = new ArrayList<>();
37         for (int i = 0; i < board.length; i++) {
38             for (int j = 0; j < board[i].length; j++) {
39                 if (board[i][j] == 0) {
40                     moves.add(new int[]{i, j});
41                 }
42             }
43         }
44     }
45 }

```

(MCSTAgent.java)

```

5  public class Memeryunit_B { 4 usages  ↳ AY +1
6
7
8     private final int ROWS = 19;// Board size (19x19) 4 usages
9     private int M_Turnflag;// Turn flag (1 for Black, -1 for White) 3 usages
10    private int[][] M_ChessMap=new int[ROWS][ROWS];// Board state 3 usages
11    private int M_step=0 ;// Number of steps/moves 3 usages
...
12
13    // Default constructor
14    Memeryunit_B() { no usages  ↳ AY
15
16    }
17
18    // Constructor with state copy
19    Memeryunit_B(int[][] m_ChessMap,int m_step,int m_Turnflag){ 2 usages  ↳ AY
20        for(int i=0;i<ROWS;i++) {
21            for(int j=0;j<ROWS;j++) {
22                M_ChessMap[i][j]=m_ChessMap[i][j];
23            }
24        }
25
26        this.M_Turnflag=m_Turnflag;
27        this.M_step=m_step;
28    }
29
30    // Getters and setters
31    public int getM_Turnflag() { return M_Turnflag; }
32
33    public void setM_Turnflag(int m_Turnflag) { M_Turnflag = m_Turnflag; }
34
35    public int[][] getM_ChessMap() { return M_ChessMap; }
36
37    public void setM_ChessMap(int[][] m_ChessMap) { M_ChessMap = m_ChessMap; }
38
39
40
41
42
43

```

(Memeryunit_B.java)

Test Case Screenshot

The screenshot shows an IDE interface with the following details:

- Left Panel:** Shows the file structure and code editor for `GoGameTestCases.java`. The code defines a class `GoGameTestCases` with methods `setUp()`, `TestNewGameState1()`, `testInvalidMove()`, and `testMoveSurround()`.
- Top Bar:** Shows tabs for `GoGameTestCases.java`, `GoGameGUIL_B.java`, and `ChessBoardPanel_B.java`. The `Run` tab is selected.
- Right Panel:** Displays the test results for `GoGameTestCases`. It shows 4 tests passed in 488 ms. The tests are:
 - `testInvalidMove`: 478 ms
 - `TestNewGameState1`: 2 ms
 - `testMoveSurround`: 6 ms
 - `testCapture`: 2 ms
- Bottom Status Bar:** Shows the file path `i5_Final > finalProject6205 > Java > src > test > java > com > phasmidsoftware > dsaipg > projects > mcts > core > GoGameTestCases.java`, line count `58:1`, character count `LF`, encoding `UTF-8`, and code style `4 spaces`.

(GoGameTestCases.java)