

E3 Othello Game ($\alpha - \beta$ pruning)

18340013 Conghao Chen

September 17, 2020

Contents

1	Othello	2
2	Tasks	2
3	Codes	3
4	Results	20

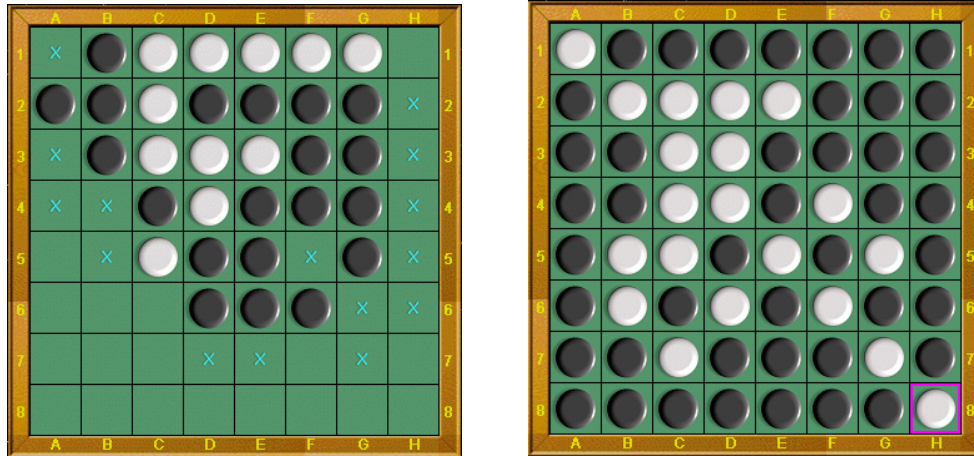


Figure 1: Othello Game

1 Othello

Othello (or Reversi) is a strategy board game for two players, played on an 8×8 unchecked board. There are sixty-four identical game pieces called disks (often spelled "discs"), which are light on one side and dark on the other. Please see figure 1.

Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color.

The object of the game is to have the majority of disks turned to display your color when the last playable empty square is filled.

You can refer to http://www.tothello.com/html/guideline_of_reversed_othello.html for more information of guideline, meanwhile, you can download the software to have a try from <http://www.tothello.com/html/download.html>. The game installer `tothello_trial_setup.exe` can also be found in the current folder.

2 Tasks

1. In order to reduce the complexity of the game, we think the board is 6×6 .
2. There are several evaluation functions that involve many aspects, you can turn to http://blog.sina.com.cn/s/blog_53ebdba00100cpy2.html for help. In order to reduce the difficulty of the task, I have given you some hints of evaluation function in the file `Heuristic Function for Reversi (Othello).cpp`.

3. Please choose an appropriate evaluation function and use min-max and $\alpha - \beta$ pruning to implement the Othello game. The framework file you can refer to is `Othello.cpp`. Of course, I wish your program can beat the computer.
4. Write the related codes and take a screenshot of the running results in the file named `E03_YourNumber.pdf`, and send it to `ai_2020@foxmail.com`.

3 Codes

```

1  #include <iostream>
2  #include <stdlib.h>
3  using namespace std;
4  int const MAX = 65534;
5  int depth = 12;          //最大搜索深度    (可调节)
6  //基本元素    棋子, 颜色, 数字变量
7  enum Option
8  {
9      WHITE = -1, SPACE, BLACK          //是否能落子    //黑子
10 };
11 struct Do
12 {
13     pair<int , int > pos;
14     int score;
15 };
16 struct WinNum
17 {
18     enum Option color;
19     int stable;
20     // 此次落子赢棋个数
21 };
22 //主要功能    棋盘及关于棋子的所有操作, 功能
23 struct Othello
24 {
25     WinNum cell[6][6];
26     //定义棋盘中有6*6个格子
27
28     int whiteNum;
29     //白棋数目
30
31     int blackNum;
32     //黑棋数目

```

```

27     void Create(Othello *board);
                                     //初始化棋盘
28     void Copy(Othello *boardDest, const Othello *boardSource);
                                     //复制棋盘
29     void Show(Othello *board);
                                     //显示棋盘
30     int Rule(Othello *board, enum Option player);
                                     //判断落子是否符合规则
31     int Action(Othello *board, Do *choice, enum Option player);
                                     //落子,并修改棋盘
32     void Stable(Othello *board);
                                     //计算赢棋个数
33     int Judge(Othello *board, enum Option player);
                                     //计算本次落子分数
34 };//主要功能
35
36
37 //我用的剪枝算法
38 Do * Find1(Othello *board, enum Option player, int step, int min, int max, Do *choice)
39 /* step: 极大极小树的深度, 从大往小递减 */
40 {
41     int i, j, k, num;
42     Do *allChoices;
43     choice->score = -MAX;
44     choice->pos.first = -1;
45     choice->pos.second = -1;
46
47     num = board->Rule(board, player);
48     /* 找出player可以落子的数量, 对应于图像界面里面的 '+' 的个数 */
49     if (num == 0) /* 无处落子 */
50     {
51         if (board->Rule(board, (enum Option) - player)) /* 对方可以落子, 让对方下.*/
52         {
53             Othello tempBoard;
54             Do nextChoice;
55             Do *pNextChoice = &nextChoice;
56             board->Copy(&tempBoard, board);
57             pNextChoice = Find1(&tempBoard, (enum Option) - player, step -
                                1, -max, -min, pNextChoice);
58             choice->score = -pNextChoice->score;

```

```

59         choice->pos.first = -1;
60         choice->pos.second = -1;
61         return choice;
62     }
63     else /* 对方也无处落子,游戏结束. */
64     {
65         int value = WHITE*(board->whiteNum) + BLACK*(board->blackNum);
66         if (player*value>0)
67         {
68             choice->score = MAX - 1;
69         }
70         else if (player*value<0)
71         {
72             choice->score = -MAX + 1;
73         }
74         else
75         {
76             choice->score = 0;
77         }
78         return choice;
79     }
80 }
81 if (step <= 0) /* 已经考虑到step步,直接返回得分 */
82 {
83     choice->score = board->Judge(board, player);
84     return choice;
85 }
86
87 /* 新建一个do*类型的数组,其中num即为玩家可落子的数量 */
88 allChoices = (Do *)malloc(sizeof(Do)*num);
89
90
91

```

92 下面三个两重for循环其实就是分区域寻找可落子的位置,代码num = board->Rule(board, player)只返回了可落子的数量,并没有返回可落子的位置,因此需要重新遍历整个棋盘去寻找可落子的位置。下面三个for循环分别按照最外一圈、最中间的四个位置、靠里的一圈这三个顺序来寻找可落子的位置,如下图所示(数字表示寻找的顺序)

```

93     1 1 1 1 1 1
94     1 3 3 3 3 1
95     1 3 2 2 3 1
96     1 3 2 2 3 1

```

```

97         1 3 3 3 3 1
98         1 1 1 1 1 1
99     */
100     k = 0;
101     for (i = 0; i < 6; i++) /* 在最外圈寻找可落子位置 */
102     {
103         for (j = 0; j < 6; j++)
104         {
105             if (i == 0 || i == 5 || j == 0 || j == 5)
106             {
107                 /* 可落子的位置需要满足两个条件：1、该位置上没有棋子，2
108                    、如果把棋子放在这个位置上可以吃掉对方的棋子(可以夹
109                    住对方的棋子)。stable记录的是可以吃掉对方棋子的数
110                    量，所以stable>0符合条件2*/
111                 if (board->cell[i][j].color == SPACE && board->cell[i][j]
112                     .stable)
113                 {
114                     allChoices[k].score = -MAX;
115                     allChoices[k].pos.first = i;
116                     allChoices[k].pos.second = j;
117                     k++;
118                 }
119             }
120         }
121     }
122
123     for (i = 0; i < 6; i++) // 分析同上
124     {
125         for (j = 0; j < 6; j++)
126         {
127             if ((i == 2 || i == 3 || j == 2 || j == 3) && (i >= 2 && i <= 3
128                 && j >= 2 && j <= 3))
129             {
130                 if (board->cell[i][j].color == SPACE && board->cell[i][j]
131                     .stable)
132                 {
133                     allChoices[k].score = -MAX;
134                     allChoices[k].pos.first = i;
135                     allChoices[k].pos.second = j;
136                     k++;
137                 }
138             }
139         }
140     }

```

```

132         }
133     }
134 }
135
136 for (i = 0; i < 6; i++) // 分析同上
137 {
138     for (j = 0; j < 6; j++)
139     {
140         if ((i == 1 || i == 4 || j == 1 || j == 4) && (i >= 1 && i <= 4
141             && j >= 1 && j <= 4))
142         {
143             if (board->cell[i][j].color == SPACE && board->cell[i][j]
144                 .stable)
145             {
146                 allChoices[k].score = -MAX;
147                 allChoices[k].pos.first = i;
148                 allChoices[k].pos.second = j;
149                 k++;
150             }
151         }
152     }
153 }
154
155 for (k = 0; k < num; k++) /* 尝试在之前得到的num个可落子位置进行落子 */
156 {
157     Othello tempBoard;
158     Do thisChoice, nextChoice;
159     Do *pNextChoice = &nextChoice;
160     thisChoice = allChoices[k];
161     board->Copy(&tempBoard, board); // 为了不影响当前棋盘，需要复制一份作为
162     虚拟棋盘
163     board->Action(&tempBoard, &thisChoice, player); // 在虚拟棋盘上落子
164     pNextChoice = Find1(&tempBoard, (enum Option) - player, step - 1, -max,
165         -min, pNextChoice); // 递归调用 - 剪枝，得到对手的落子评分
166     thisChoice.score = -pNextChoice->score;
167
168     /* 使用Negamax算法代替minmax算法，实现 - 剪枝 */
169     // 其中，max 取上一层min的相反数，min取当前选择的score。
170     // 对每一层，我方行棋选择我方获益分数最大的，对手行棋选择我方获益分数最
171     小的；
172     // 因此，实际上只需要将每一层的max min调换并取反即可；

```

```

168 // 故假设根节点为第0层, beta层的数值为负。
169 // 剪枝条件: beta <= alpha, 即 score >= max。
170
171 if (player == WHITE) {
172     int alpha = -max, beta = -min;
173     if (thisChoice.score > -beta) {
174         beta = -thisChoice.score;
175         choice->score = thisChoice.score;
176         choice->pos.first = thisChoice.pos.first;
177         choice->pos.second = thisChoice.pos.second;
178         min = -beta;
179         if (beta <= alpha) break;
180     }
181 }
182 else if (player == BLACK) {
183     int alpha = min, beta = max;
184     if (thisChoice.score > alpha) {
185         alpha = thisChoice.score;
186         choice->score = thisChoice.score;
187         choice->pos.first = thisChoice.pos.first;
188         choice->pos.second = thisChoice.pos.second;
189         min = alpha;
190         if (beta <= alpha) break;
191     }
192 }
193 }
194 free(allChoices);
195 return choice;
196 }
197
198 int main()
199 {
200     Othello board;
201     Othello *pBoard = &board;
202     enum Option player , present ;
203     Do choice;
204     Do *pChoice = &choice;
205     int num , result = 0;
206     char restart = ' ';
207
208 start:

```



```

209     player = SPACE;
210     present = BLACK;
211     num = 4;
212     restart = ' ';
213
214     cout << ">>>人机对战开始:  \n";
215     while (player != WHITE && player != BLACK)
216     {
217         cout << ">>>请选择执黑棋( ),或执白棋( ): 输入1为黑棋, -1为白棋"
218             << endl;
219         scanf("%d", &player);
220         cout << ">>>黑棋行动:  \n";
221         if (player != WHITE && player != BLACK)
222         {
223             cout << "输入不符合规范, 请重新输入\n";
224         }
225     }
226
227     board.Create(pBoard);
228     while (num<36)
229         // 棋盘上未下满36子
230     {
231         char *Player = "";
232         if (present == BLACK)
233         {
234             Player = "黑棋( )";
235         }
236         else if (present == WHITE)
237         {
238             Player = "白棋( )";
239         }
240
241         if (board.Rule(pBoard, present) == 0)
242             //未下满并且无子可下
243         {
244             if (board.Rule(pBoard, (enum Option) - present) == 0)
245             {
246                 break;
247             }
248
249             cout << Player << "GAME OVER!  \n";

```

```

247     }
248     else
249     {
250         int i, j;
251         board.Show(pBoard);
252
253         if (present == player)    //我的AI下棋
254         {
255             cout << Player << " ..... ";
256
257             pChoice = Find1(pBoard, present, depth, -MAX,
258                             MAX, pChoice);
259             i = pChoice->pos.first;
260             j = pChoice->pos.second;
261             system("cls");
262             cout << ">>>我的AI本手棋得分为      " << pChoice
263                 << ">>>";
264             board.Action(pBoard, pChoice, present);
265             num++;
266             cout << Player << ">>>我的AI于" << i + 1 << ", " << j +
267                 << "1<<落子, 该你了! ";
268         }
269         else    //电脑的AI下棋
270         {
271             cout << Player << " ..... ";
272
273             pChoice = Find1(pBoard, present, depth, -MAX,
274                             MAX, pChoice);
275             i = pChoice->pos.first;
276             j = pChoice->pos.second;
277             system("cls");
278             cout << ">>>电脑的AI本手棋得分为      " <<
279                 << pChoice->score << ">>>";
280             board.Action(pBoard, pChoice, present);
281             num++;
282             cout << Player << ">>>电脑的AI于" << i + 1 << ", " << j +
283                 << "1<<落子, 该你了! ";
284         }
285     }
286     present = (enum Option) - present;    //交换执棋者
287 }

```

[illegible]

```

323
324     return 0;
325 }
326
327
328 void Othello::Create(Othello *board)
329 {
330     int i, j;
331     board->whiteNum = 2;
332     board->blackNum = 2;
333     for (i = 0; i < 6; i++)
334     {
335         for (j = 0; j < 6; j++)
336         {
337             board->cell[i][j].color = SPACE;
338             board->cell[i][j].stable = 0;
339         }
340     }
341     board->cell[2][2].color = board->cell[3][3].color = WHITE;
342     board->cell[2][3].color = board->cell[3][2].color = BLACK;
343 }
344
345
346 void Othello::Copy(Othello *Fake, const Othello *Source)
347 {
348     int i, j;
349     Fake->whiteNum = Source->whiteNum;
350     Fake->blackNum = Source->blackNum;
351     for (i = 0; i < 6; i++)
352     {
353         for (j = 0; j < 6; j++)
354         {
355             Fake->cell[i][j].color = Source->cell[i][j].color;
356             Fake->cell[i][j].stable = Source->cell[i][j].stable;
357         }
358     }
359 }
360
361 void Othello::Show(Othello *board)
362 {
363     int i, j;

```

```

364     cout << "\n ";
365     for (i = 0; i<6; i++)
366     {
367         cout << "    " << i + 1;
368     }
369     cout << "\n          \n";
370     for (i = 0; i<6; i++)
371     {
372         cout << i + 1 << "-- ";
373         for (j = 0; j<6; j++)
374         {
375             switch (board->cell[i][j].color)
376             {
377                 case BLACK:
378                     cout << "    ";
379                     break;
380                 case WHITE:
381                     cout << "    ";
382                     break;
383                 case SPACE:
384                     if (board->cell[i][j].stable)
385                     {
386                         cout << " + ";
387                     }
388                     else
389                     {
390                         cout << "    ";
391                     }
392                     break;
393                 default: /* 棋子颜色错误 */
394                     cout << "*    ";
395             }
396         }
397         cout << "\n          \n";
398     }
399
400     cout << ">>>白棋 ( ) 个数为:" << board->whiteNum << "          ";
401     cout << ">>>黑棋 ( ) 个数为:" << board->blackNum << endl << endl << endl;
402 }
403
404 int Othello::Rule(Othello *board, enum Option player)

```

```

405 {
406     int i, j;
407     unsigned num = 0;
408     for (i = 0; i < 6; i++)
409     {
410         for (j = 0; j < 6; j++)
411         {
412             if (board->cell[i][j].color == SPACE)
413             {
414                 int x, y;
415                 board->cell[i][j].stable = 0;
416                 for (x = -1; x <= 1; x++)
417                 {
418                     for (y = -1; y <= 1; y++)
419                     {
420                         if (x || y) /* 8个方向 */
421                         {
422                             int i2, j2;
423                             unsigned num2 = 0;
424                             for (i2 = i + x, j2 = j + y; i2
425                                 >= 0 && i2 <= 5 && j2 >= 0
426                                 && j2 <= 5; i2 += x, j2 += y
427                             )
428                             {
429                                 if (board->cell[i2][j2].
430                                     color == (enum
431                                     Option) - player)
432                                 {
433                                     num2++;
434                                 }
435                                 else if (board->cell[i2
436                                     ][j2].color ==
437                                     player)
438                                 {
439                                     board->cell[i][j
440                                     ].stable +=
441                                     player*num2;
442                                     break;
443                                 }
444                                 else if (board->cell[i2
445                                     ][j2].color == SPACE

```

```

436         )
437         {
438             break;
439         }
440     }
441 }
442 }
443
444     if (board->cell[i][j].stable)
445     {
446         num++;
447     }
448 }
449 }
450 }
451 return num;
452 }
453
454
455 int Othello::Action(Othello *board, Do *choice, enum Option player)
456 {
457     int i = choice->pos.first, j = choice->pos.second;
458     int x, y;
459
460     /* 要准备落子的位置上已经有棋子，或者在这个位置落子不能吃掉对方任何棋子的话，说明这个 action 不合理，直接返回 */
461     if (board->cell[i][j].color != SPACE || board->cell[i][j].stable == 0 || player == SPACE)
462     {
463         return -1;
464     }
465
466     board->cell[i][j].color = player;
467     board->cell[i][j].stable = 0;
468
469
470
471     if (player == WHITE)
472     {
473         board->whiteNum++;

```

```

474     }
475     else if (player == BLACK)
476     {
477         board->blackNum++;
478     }
479
480
481
482     for (x = -1; x <= 1; x++)
483     {
484         for (y = -1; y <= 1; y++)
485         {
486
487             //需要在每个方向（8个）上检测落子是否符合规则（能否吃子）
488
489
490             if (x || y)
491             {
492                 int i2, j2;
493                 unsigned num = 0;
494                 for (i2 = i + x, j2 = j + y; i2 >= 0 && i2 <= 5 && j2 >=
495                     0 && j2 <= 5; i2 += x, j2 += y)
496                 {
497                     if (board->cell[i2][j2].color == (enum Option) -
498                         player)
499                     {
500                         num++;
501                     }
502                     else if (board->cell[i2][j2].color == player)
503                     {
504                         board->whiteNum += (player*WHITE)*num;
505                         board->blackNum += (player*BLACK)*num;
506
507                         for (i2 -= x, j2 -= y; num>0; num--, i2
508                             -= x, j2 -= y)
509                         {
510                             board->cell[i2][j2].color =
511                                 player;
512                             board->cell[i2][j2].stable = 0;
513                         }
514                         break;

```



```

511         }
512         else if (board->cell[i2][j2].color == SPACE)
513         {
514             break;
515         }
516     }
517 }
518 }
519 }
520 return 0;
521 }
522
523
524 void Othello::Stable(Othello *board)
525 {
526     int i, j;
527     for (i = 0; i < 6; i++)
528     {
529         for (j = 0; j < 6; j++)
530         {
531             if (board->cell[i][j].color != SPACE)
532             {
533                 int x, y;
534                 board->cell[i][j].stable = 1;
535
536                 for (x = -1; x <= 1; x++)
537                 {
538                     for (y = -1; y <= 1; y++)
539                     {
540                         /* 4个方向 */
541                         if (x == 0 && y == 0)
542                         {
543                             x = 2;
544                             y = 2;
545                         }
546                         else
547                         {
548                             int i2, j2, flag = 2;
549                             for (i2 = i + x, j2 = j + y; i2

```

```

550         )
551     {
552         if (board->cell[i2][j2].
553             color != board->cell
554             [i][j].color)
555         {
556             flag--;
557             break;
558         }
559     }
560     for (i2 = i - x, j2 = j - y; i2
561         >= 0 && i2 <= 5 && j2 >= 0
562         && j2 <= 5; i2 += x, j2 += y
563         )
564     {
565         if (board->cell[i2][j2].
566             color != board->cell
567             [i][j].color)
568         {
569             flag--;
570             break;
571         }
572     }
573     if (flag) /* 在某一条线上稳定
574                */
575     {
576         board->cell[i][j].stable
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 }

```

```

581     int value = 0;
582     int i, j;
583     Stable(board);
584
585     // 对稳定子给予奖励
586     for (i = 0; i < 6; i++)
587     {
588         for (j = 0; j < 6; j++)
589         {
590             value += (board->cell[i][j].color)*(board->cell[i][j].stable);
591         }
592     }
593
594     int V[6][6] = {{ 20,  -8,  11,  11,  -8,  20},
595                    { -8, -15,  -4,  -4, -15,  -8},
596                    { 11,  -4,   2,   2,  -4,  11},
597                    { 11,  -4,   2,   2,  -4,  11},
598                    { -8, -15,  -4,  -4, -15,  -8},
599                    { 20,  -8,  11,  11,  -8,  20}};
600
601     for (int i = 0; i < 6; ++i)
602     {
603         for (int j = 0; j < 6; ++j)
604         {
605             value += V[i][j] * board->cell[i][j].color;
606         }
607     }
608
609     // 行动力计算
610     int my_mov, opp_mov, mov = 0;
611     my_mov = Rule(board, player);
612     opp_mov = Rule(board, (enum Option) - player);
613     if(my_mov > opp_mov)
614         value += 78.922 * (100.0 * my_mov)/(my_mov + opp_mov);
615     else if(my_mov < opp_mov)
616         value += 78.922 * -(100.0 * opp_mov)/(my_mov + opp_mov);
617
618     return value*player;
619 }

```

4 Results

从上面的图可以看到，最终是执黑棋的我的 AI 击败了执白棋的电脑的 AI。代码主要修改了两部分，一部分是 Judge 函数里的 Evaluation function，还有就是剪枝算法做了修改：使用自己写的 Negamax 算法，而不是原文件内的算法。不过这个程序要执行好久，大约 7-10min 左右才能看到结果。运行后不用进行任何操作，自己会交替显示双方的情况。

C:\Users\czh\Desktop\AI\AI-master\Experiment\Exp3\E03_17341137\src\Othello_revised.exe

```
>>>人机对战开始:
>>>请选择执黑棋(O),或执白棋(●): 输入1为黑棋, -1为白棋
1
```

C:\Users\czh\Desktop\AI\AI-master\Experiment\Exp3\E03_17341137\src\Othello_revised.exe

```
>>>AI 本手棋得分为 -65533
白棋(●)>>>AI于2,5落子, 该你了!
  1  2  3  4  5  6
1-- |O|O|O|O|●|●|
2-- |O|●|●|●|●|●|
3-- |O|O|●|●|O|●|
4-- |O|O|O|●|O|●|
5-- |O|●|●|O|O|●|
6-- |O|O|O|O|O|●|
>>>白棋(●)个数为:16      >>>黑棋(O)个数为:20

-----黑棋(O)胜-----
-----GAME OVER!-----
```