School of Data and Computer Science, Sun Yat-sen University

E03 Othello Game ($\alpha - \beta$ pruning)

20214810 Suixin Ou 20214966 Yangkai Lin

September 14, 2020

Contents

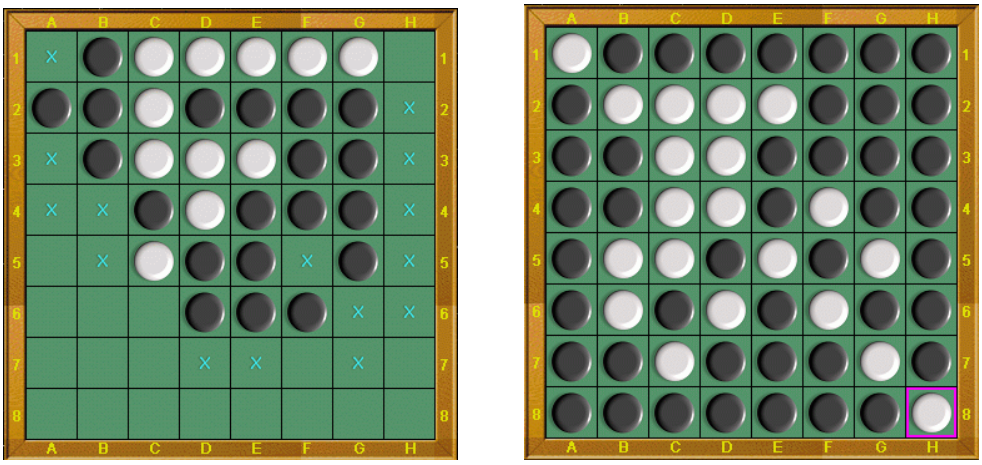

Figure 1: Othello Game

1 Othello

Othello (or Reversi) is a strategy board game for two players, played on an 8×8 uncheckered 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://www.cs.cornell.edu/~yuli/othello/othello.html` for help. In order to reduce the difficulty of the task, I have gaven 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 $\alpha - \beta$ prunning 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_StudentNumber.pdf, and send it to ai_2020@foxmail.com, the **deadline** is 2020.09.20 23:59:59.

3 Codes

```
#include <iostream>
#include <stdlib.h>

using namespace std;

int const MAX = 65534;

int  deepth = 10;          //最大搜索深度  （可调节）

int depth = 10;

//基本元素   棋子，颜色，数字变量

enum Option
{
    WHITE = -1, SPACE, BLACK  //是否能落子  黑子//
};

struct Do
{
    pair<int , int > pos;
    int score;
};

struct WinNum
{ enum Option color;
    int stable;                    // 此次落子赢棋个数
};


```

```
32
33
34 //主要功能     棋盘及关于棋子的所有操作，功能
35 struct Othello
36 {
37
38     WinNum cell[6][6];                          //定义棋盘中有个格子6*6
39     int whiteNum;                               //白棋数目
40     int blackNum;                               //黑棋数
41
42
43     void Create(Othello *board);                        //初始化棋盘
44     void Copy(Othello *boardDest, const  Othello *boardSource);      //复制棋盘
45     void Show(Othello *board);                          //显示棋盘
46     int Rule(Othello *board, enum Option player);             //判断落子是否符合规则
47     int Action(Othello *board, Do *choice, enum Option player);      //落子并修改棋盘,
48     void Stable(Othello *board);                        //计算赢棋个数
49     int Judge(Othello *board, enum Option player);              //计算本次落子分数
50     int Heuristic_Evaluation_Function(Othello *board, enum Option player); //计算本次落
       子分数
51 };//主要功能
52
53
54
55
56
57
58
59
60 //最大最小博弈与α β剪枝-
61 Do * Find(Othello *board, enum Option player, int step, int min, int max, Do *choice)
62 {
63     int i, j, k, num;
64     Do *allChoices;
65     choice->score = -MAX;
66     choice->pos.first = -1;
67     choice->pos.second = -1;
68
69     num = board->Rule(board, player);
70     if (num == 0)     /* 无处落子 */
71     {
```

```
72          if (board->Rule(board, (enum Option) - player))     /* 对方可以落子让对方下,.*/
73          {
74              Othello tempBoard;
75              Do nextChoice;
76              Do *pNextChoice = &nextChoice;
77              board->Copy(&tempBoard, board);
78              pNextChoice = Find(&tempBoard, (enum Option) - player, step - 1, -max, -
    min, pNextChoice);
79              choice->score = -pNextChoice->score;
80              choice->pos.first = -1;
81              choice->pos.second = -1;
82              return choice;
83          }
84          else     /* 对方也无处落子游戏结束,. */
85          {
86              int value = WHITE*(board->whiteNum) + BLACK*(board->blackNum);
87              if (player*value>0)
88              {
89                  choice->score = MAX - 1;
90              }
91              else if (player*value<0)
92              {
93                  choice->score = -MAX + 1;
94              }
95              else
96              {
97                  choice->score = 0;
98              }
99              return choice;
100         }
101     }
102     if (step <= 0)     /* 已经考虑到步step直接返回得分, */
103     {
104         choice->score = board->Judge(board, player);
105         return choice;
106     }
107
108     allChoices = (Do *)malloc(sizeof(Do)*num);
109     k = 0;
110     for (i = 0; i<6; i++)
111     {
```

```
112         for (j = 0; j<6; j++)
113         {
114             if (i == 0 || i == 5 || j == 0 || j == 5)
115             {
116                 if (board->cell[i][j].color == SPACE && board->cell[i][j].stable)
117                 {
118                     allChoices[k].score = -MAX;
119                     allChoices[k].pos.first = i;
120                     allChoices[k].pos.second = j;
121                     k++;
122                 }
123             }
124         }
125     }
126 
127     for (i = 0; i<6; i++)
128     {
129         for (j = 0; j<6; j++)
130         {
131             if ((i == 2 || i == 3 || j == 2 || j == 3) && (i >= 2 && i <= 3 && j >= 2
    && j <= 3))
132             {
133                 if (board->cell[i][j].color == SPACE && board->cell[i][j].stable)
134                 {
135                     allChoices[k].score = -MAX;
136                     allChoices[k].pos.first = i;
137                     allChoices[k].pos.second = j;
138                     k++;
139                 }
140             }
141         }
142     }
143 
144     for (i = 0; i<6; i++)
145     {
146         for (j = 0; j<6; j++)
147         {
148             if ((i == 1 || i == 4 || j == 1 || j == 4) && (i >= 1 && i <= 4 && j >= 1
    && j <= 4))
149             {
150                 if (board->cell[i][j].color == SPACE && board->cell[i][j].stable)
```

```
151                 {
152                     allChoices[k].score = -MAX;
153                     allChoices[k].pos.first = i;
154                     allChoices[k].pos.second = j;
155                     k++;
156                 }
157             }
158         }
159     }
160 
161     for (k = 0; k<num; k++)
162     {
163         Othello tempBoard;
164         Do thisChoice, nextChoice;
165         Do *pNextChoice = &nextChoice;
166         thisChoice = allChoices[k];
167         board->Copy(&tempBoard, board);
168         board->Action(&tempBoard, &thisChoice, player);
169         pNextChoice = Find(&tempBoard, (enum Option) - player, step - 1, -max, -min,
    pNextChoice);
170         thisChoice.score = -pNextChoice->score;
171 
172         if (thisChoice.score>min && thisChoice.score<max)     /* 可以预计的更优值 */
173         {
174             min = thisChoice.score;
175             choice->score = thisChoice.score;
176             choice->pos.first = thisChoice.pos.first;
177             choice->pos.second = thisChoice.pos.second;
178         }
179         else if (thisChoice.score >= max)     /* 好的超乎预计 */
180         {
181             choice->score = thisChoice.score;
182             choice->pos.first = thisChoice.pos.first;
183             choice->pos.second = thisChoice.pos.second;
184             break;
185         }
186         /* 不如已知最优值 */
187     }
188     free(allChoices);
189     return choice;
190 }
```

```
191
192 Do * My_Find(Othello *board, enum Option player, int step, int min, int max, Do *
    choice)
193 {
194     int i, j, k, num;
195     Do *allChoices;
196     choice->score = -MAX;
197     choice->pos.first = -1;
198     choice->pos.second = -1;
199
200     num = board->Rule(board, player);
201     if (num == 0)     /* 无处落子 */
202     {
203         if (board->Rule(board, (enum Option) - player))     /* 对方可以落子让对方下,.*/
204         {
205             Othello tempBoard;
206             Do nextChoice;
207             Do *pNextChoice = &nextChoice;
208             board->Copy(&tempBoard, board);
209             pNextChoice = My_Find(&tempBoard, (enum Option) - player, step - 1, -max,
    -min, pNextChoice);
210             choice->score = -pNextChoice->score;
211             choice->pos.first = -1;
212             choice->pos.second = -1;
213             return choice;
214         }
215         else     /* 对方也无处落子游戏结束,. */
216         {
217             int value = WHITE*(board->whiteNum) + BLACK*(board->blackNum);
218             if (player*value>0)
219             {
220                 choice->score = MAX - 1;
221             }
222             else if (player*value<0)
223             {
224                 choice->score = -MAX + 1;
225             }
226             else
227             {
228                 choice->score = 0;
229             }
```

```
230             return choice;
231         }
232     }
233     if (step <= 0)     /* 已经考虑到步step直接返回得分, */
234     {
235         choice->score = board->Heuristic_Evaluation_Function(board, player);
236         return choice;
237     }
238 
239     allChoices = (Do *)malloc(sizeof(Do)*num);
240     k = 0;
241     for (i = 0; i<6; i++)
242     {
243         for (j = 0; j<6; j++)
244         {
245             if (i == 0 || i == 5 || j == 0 || j == 5)
246             {
247                 if (board->cell[i][j].color == SPACE && board->cell[i][j].stable)
248                 {
249                     allChoices[k].score = -MAX;
250                     allChoices[k].pos.first = i;
251                     allChoices[k].pos.second = j;
252                     k++;
253                 }
254             }
255         }
256     }
257 
258     for (i = 0; i<6; i++)
259     {
260         for (j = 0; j<6; j++)
261         {
262             if ((i == 2 || i == 3 || j == 2 || j == 3) && (i >= 2 && i <= 3 && j >= 2
    && j <= 3))
263             {
264                 if (board->cell[i][j].color == SPACE && board->cell[i][j].stable)
265                 {
266                     allChoices[k].score = -MAX;
267                     allChoices[k].pos.first = i;
268                     allChoices[k].pos.second = j;
269                     k++;
```

```
                }
            }
        }
    }

    for (i = 0; i<6; i++)
    {
        for (j = 0; j<6; j++)
        {
            if ((i == 1 || i == 4 || j == 1 || j == 4) && (i >= 1 && i <= 4 && j >= 1
    && j <= 4))
            {
                if (board->cell[i][j].color == SPACE && board->cell[i][j].stable)
                {
                    allChoices[k].score = -MAX;
                    allChoices[k].pos.first = i;
                    allChoices[k].pos.second = j;
                    k++;
                }
            }
        }
    }

    for (k = 0; k<num; k++)
    {
        Othello tempBoard;
        Do thisChoice, nextChoice;
        Do *pNextChoice = &nextChoice;
        thisChoice = allChoices[k];
        board->Copy(&tempBoard, board);
        board->Action(&tempBoard, &thisChoice, player);
        pNextChoice = My_Find(&tempBoard, (enum Option) - player, step - 1, -max, -min
    , pNextChoice);
        thisChoice.score = -pNextChoice->score;

        if (thisChoice.score>min && thisChoice.score<max)     /* 可以预计的更优值 */
        {
            min = thisChoice.score;
            choice->score = thisChoice.score;
            choice->pos.first = thisChoice.pos.first;
            choice->pos.second = thisChoice.pos.second;
```

```
309         }
310         else if (thisChoice.score >= max)     /* 好的超乎预计 */
311         {
312             choice->score = thisChoice.score;
313             choice->pos.first = thisChoice.pos.first;
314             choice->pos.second = thisChoice.pos.second;
315             break;
316         }
317         /* 不如已知最优值 */
318     }
319     free(allChoices);
320     return choice;
321 }
322
323 int main()
324 {
325     Othello board;
326     Othello *pBoard = &board;
327     enum Option player , present ;
328     Do choice;
329     Do *pChoice = &choice;
330     int  num , result = 0;
331     char restart = ' ';
332
333 start:
334     player = SPACE;
335     present = BLACK;
336     num = 4;
337     restart = ' ';
338
339     cout << "人机对战开始: >>> \n";
340
341
342
343
344         while (player != WHITE && player != BLACK)
345         {
346             cout << "请选择执黑棋○>>>()或执白棋● ,(): 输入为黑棋, 为白棋1-1" << endl;
347             scanf("%d", &player);
348             cout << "黑棋行动>>>:  \n";
349
```

```
350
351             if (player != WHITE && player != BLACK)
352             {
353                 cout << "输入不符合规范，请重新输入\n";
354             }
355         }
356
357         board.Create(pBoard);
358
359         while (num<36)                              // 棋盘上未下满子36
360         {
361             char *Player = "";
362             if (present == BLACK)
363             {
364                 Player = "黑棋○()";
365             }
366             else if (present == WHITE)
367             {
368                 Player = "白棋●()";
369             }
370
371             if (board.Rule(pBoard, present) == 0)           //未下满并且无子可下
372             {
373                 if (board.Rule(pBoard, (enum Option) - present) == 0)
374                 {
375                     break;
376                 }
377
378                 cout << Player << "GAME OVER! \n";
379             }
380             else
381             {
382                 int i, j;
383                 //board.Show(pBoard);
384
385                 if (present == player)
386                 {
387                     /*
388                     while (1)
389                     {
```

```
390                     cout << Player << " \n 请输入棋子坐标（空格相隔>>> 如“3 ”代表第行第
    列）535:\n";
391
392                     cin >> i>> j;
393                     i--;
394                     j--;
395                     pChoice->pos.first = i;
396                     pChoice->pos.second = j;
397
398                     if (i<0 || i>5 || j<0 || j>5 || pBoard->cell[i][j].color !=
    SPACE || pBoard->cell[i][j].stable == 0)
399                     {
400                         cout 此处落子不符合规则，请重新选择<<">>>    \n";
401                         board.Show(pBoard);
402                     }
403                     else
404                     {
405                         break;
406                     }
407                 }
408                 system("cls");
409                 cout << 玩家">>> 本手棋得分为      " << pChoice->score << endl;
410                 system("pause");
411                 cout << 按任意键继续">>>" << pChoice->score << endl;
412                 */
413                 //cout << Player << "........................";
414
415                 pChoice = My_Find(pBoard, present, depth, -MAX, MAX, pChoice);
416                 i = pChoice->pos.first;
417                 j = pChoice->pos.second;
418                 //system("cls");
419             }
420             else    //下棋AI
421             {
422                 //cout << Player << "........................";
423
424                 pChoice = Find(pBoard, present, deepth, -MAX, MAX, pChoice);
425                 i = pChoice->pos.first;
426                 j = pChoice->pos.second;
427                 //system("cls");
428                 //cout << ">>>AI 本手棋得分为      " << pChoice->score << endl;
```

```
429                 }
430
431
432                 board.Action(pBoard, pChoice, present);
433                 num++;
434                 //cout << Player << ">>>于AI" << i + 1 << "," << j + 落子1<<"";
435                 cout << Player << "于" << i + 1 << "," << j + 1<<"落子。黑棋数:
    "<<pBoard->blackNum<<", 白棋数:"<<pBoard->whiteNum<<endl;
436             }
437
438             present = (enum Option) - present;      //交换执棋者
439         }
440
441
442         //board.Show(pBoard);
443
444
445         result = pBoard->whiteNum - pBoard->blackNum;
446
447         if (result>0)
448         {
449             cout << "\n————————白棋n●()胜————————\n";
450         }
451         else if (result<0)
452         {
453             cout << "\n————————黑棋n○()胜————————\n";
454         }
455         else
456         {
457             cout << "\n————————————平局————————————n\n";
458         }
459
460         cout << "\n ————————————GAME OVER————————————!\n";
461         cout << "\n";
462
463         while (restart != 'Y' && restart != 'N')
464         {
465             cout <<"————————————————————————————————||\n";
466             cout <<"|                                                |\n";
467             cout <<"|                                                |    \n";
468             cout <<"|>>>>>>>>>>>>>>>>>Again?(Y,N)<<<<<<<<<<<<<<<<<|\n";
```

```
469             cout <<"|                                                  | \n";
470             cout <<"|                                                  |  \n";
471             cout <<"-------------------------------------||\n";
472             cout << "                                                    \n";
473             cout << "                                                    \n";
474             cout << "                                                    \n";
475             cout << " --------                  --------          \n";
476             cout << " |   YES  |                  |   NO   |          \n";
477             cout << " --------                  --------          \n";
478 
479             cin >> restart;
480             if (restart == 'Y')
481             {
482                 goto start;
483             }
484         }
485 
486 
487     return 0;
488 }
489 
490 
491 
492 
493 
494 
495 void Othello::Create(Othello *board)
496 {
497     int i, j;
498     board->whiteNum = 2;
499     board->blackNum = 2;
500     for (i = 0; i<6; i++)
501     {
502         for (j = 0; j<6; j++)
503         {
504             board->cell[i][j].color = SPACE;
505             board->cell[i][j].stable = 0;
506         }
507     }
508     board->cell[2][2].color = board->cell[3][3].color = WHITE;
509     board->cell[2][3].color = board->cell[3][2].color = BLACK;
```

```
510 }
511
512
513 void Othello::Copy(Othello *Fake, const  Othello *Source)
514 {
515     int i, j;
516     Fake->whiteNum = Source->whiteNum;
517     Fake->blackNum = Source->blackNum;
518     for (i = 0; i<6; i++)
519     {
520         for (j = 0; j<6; j++)
521         {
522             Fake->cell[i][j].color = Source->cell[i][j].color;
523             Fake->cell[i][j].stable = Source->cell[i][j].stable;
524         }
525     }
526 }
527
528 void Othello::Show(Othello *board)
529 {
530     int i, j;
531     cout << "\n  ";
532     for (i = 0; i<6; i++)
533     {
534         cout << "   " << i + 1;
535     }
536     cout << "\n    ——————————————————\n";
537     for (i = 0; i<6; i++)
538     {
539         cout << i + 1 << " |—";
540         for (j = 0; j<6; j++)
541         {
542             switch (board->cell[i][j].color)
543             {
544             case BLACK:
545                 cout << "○|";
546                 break;
547             case WHITE:
548                 cout << "●|";
549                 break;
550             case SPACE:
```

```
551                 if (board->cell[i][j].stable)
552                 {
553                     cout << "  |+";
554                 }
555                 else
556                 {
557                     cout << "   |";
558                 }
559                 break;
560             default:     /* 棋子颜色错误 */
561                 cout << "* |";
562             }
563         }
564         cout << "\n    ——————————————————\n";
565     }
566
567     cout << "白棋●>>>()个数为:" << board->whiteNum << "          ";
568     cout << "黑棋○>>>()个数为:" << board->blackNum << endl << endl << endl;
569 }
570
571
572 int Othello::Rule(Othello *board, enum Option player)
573 {
574     int i, j;
575     unsigned num = 0;
576     for (i = 0; i<6; i++)
577     {
578         for (j = 0; j<6; j++)
579         {
580             if (board->cell[i][j].color == SPACE)
581             {
582                 int x, y;
583                 board->cell[i][j].stable = 0;
584                 for (x = -1; x <= 1; x++)
585                 {
586                     for (y = -1; y <= 1; y++)
587                     {
588                         if (x || y)     /* 个方向8 */
589                         {
590                             int i2, j2;
591                             unsigned num2 = 0;
```

```
592                             for (i2 = i + x, j2 = j + y; i2 >= 0 && i2 <= 5 && j2 >= 0
    && j2 <= 5; i2 += x, j2 += y)
593                             {
594                                 if (board->cell[i2][j2].color == (enum Option) -
    player)
595                                 {
596                                     num2++;
597                                 }
598                                 else if (board->cell[i2][j2].color == player)
599                                 {
600                                     board->cell[i][j].stable += player*num2;
601                                     break;
602                                 }
603                                 else if (board->cell[i2][j2].color == SPACE)
604                                 {
605                                     break;
606                                 }
607                             }
608                         }
609                     }
610                 }
611
612                 if (board->cell[i][j].stable)
613                 {
614                     num++;
615                 }
616             }
617         }
618     }
619     return num;
620 }
621
622
623 int Othello::Action(Othello *board, Do *choice, enum Option player)
624 {
625     int i = choice->pos.first, j = choice->pos.second;
626     int x, y;
627
628     if (board->cell[i][j].color != SPACE || board->cell[i][j].stable == 0 || player ==
    SPACE)
629     {
```

```
630         return -1;
631     }
632
633
634     board->cell[i][j].color = player;
635     board->cell[i][j].stable = 0;
636
637
638     if (player == WHITE)
639     {
640         board->whiteNum++;
641     }
642     else if (player == BLACK)
643     {
644         board->blackNum++;
645     }
646
647
648
649     for (x = -1; x <= 1; x++)
650     {
651         for (y = -1; y <= 1; y++)
652         {
653
654             //需要在每个方向（个）上检测落子是否符合规则（能否吃子）8
655
656
657             if (x || y)
658             {
659                 int i2, j2;
660                 unsigned num = 0;
661                 for (i2 = i + x, j2 = j + y; i2 >= 0 && i2 <= 5 && j2 >= 0 && j2 <= 5;
    i2 += x, j2 += y)
662                 {
663                     if (board->cell[i2][j2].color == (enum Option) - player)
664                     {
665                         num++;
666                     }
667                     else if (board->cell[i2][j2].color == player)
668                     {
669                         board->whiteNum += (player*WHITE)*num;
```

```
670                         board->blackNum += (player*BLACK)*num;
671
672                         for (i2 -= x, j2 -= y; num>0; num--, i2 -= x, j2 -= y)
673                         {
674                             board->cell[i2][j2].color = player;
675                             board->cell[i2][j2].stable = 0;
676                         }
677                         break;
678                     }
679                     else if (board->cell[i2][j2].color == SPACE)
680                     {
681                         break;
682                     }
683                 }
684             }
685         }
686     }
687     return 0;
688 }
689
690
691 void Othello::Stable(Othello *board)
692 {
693     int i, j;
694     for (i = 0; i<6; i++)
695     {
696         for (j = 0; j<6; j++)
697         {
698             if (board->cell[i][j].color != SPACE)
699             {
700                 int x, y;
701                 board->cell[i][j].stable = 1;
702
703                 for (x = -1; x <= 1; x++)
704                 {
705                     for (y = -1; y <= 1; y++)
706                     {
707                         /* 个方向4 */
708                         if (x == 0 && y == 0)
709                         {
710                             x = 2;
```

```
711                             y = 2;
712                         }
713                         else
714                         {
715                             int i2, j2, flag = 2;
716                             for (i2 = i + x, j2 = j + y; i2 >= 0 && i2 <= 5 && j2 >= 0
    && j2 <= 5; i2 += x, j2 += y)
717                             {
718                                 if (board->cell[i2][j2].color != board->cell[i][j].
    color)
719                                 {
720                                     flag--;
721                                     break;
722                                 }
723                             }
724 
725                             for (i2 = i - x, j2 = j - y; i2 >= 0 && i2 <= 5 && j2 >= 0
    && j2 <= 5; i2 -= x, j2 -= y)
726                             {
727                                 if (board->cell[i2][j2].color != board->cell[i][j].
    color)
728                                 {
729                                     flag--;
730                                     break;
731                                 }
732                             }
733 
734                             if (flag)     /* 在某一条线上稳定 */
735                             {
736                                 board->cell[i][j].stable++;
737                             }
738                         }
739                     }
740                 }
741             }
742         }
743     }
744 }
745 
746 int Othello::Judge(Othello *board, enum Option player)
747 {
```

```
748     int value = 0;
749     int i, j;
750     Stable(board);
751     for (i = 0; i<6; i++)
752     {
753         for (j = 0; j<6; j++)
754         {
755             value += (board->cell[i][j].color)*(board->cell[i][j].stable);
756         }
757     }
758 
759     value += 64 * board->cell[0][0].color;
760     value += 64 * board->cell[0][5].color;
761     value += 64 * board->cell[5][0].color;
762     value += 64 * board->cell[5][5].color;
763     value -= 32 * board->cell[1][1].color;
764     value -= 32 * board->cell[1][4].color;
765     value -= 32 * board->cell[4][1].color;
766     value -= 32 * board->cell[4][4].color;
767 
768     return value*player;
769 }
770 
771 bool canmove(Option self, Option opp, Option* str) {
772     if (str[0] != opp) return false;
773     for (int ctr = 1; ctr < 8; ctr++) {
774         if (str[ctr] == SPACE) return false;
775         if (str[ctr] == self) return true;
776     }
777     return false;
778 }
779 
780 bool isLegalMove(Option self, Option opp, Othello *board, int startx, int starty) {
781     if (board->cell[startx][starty].color != SPACE) return false;
782     Option str[10];
783     int x, y, dx, dy, ctr;
784     for (dy = -1; dy <= 1; dy++)
785         for (dx = -1; dx <= 1; dx++) {
786             // keep going if both velocities are zero
787             if (!dy && !dx) continue;
788             str[0] = SPACE;
```

```
789             for (ctr = 1; ctr < 8; ctr++) {
790                 x = startx + ctr * dx;
791                 y = starty + ctr * dy;
792                 if (x >= 0 && y >= 0 && x < 8 && y < 8) str[ctr - 1] = board->cell[x][
    y].color;
793                 else str[ctr - 1] = SPACE;
794             }
795             if (canmove(self, opp, str)) return true;
796         }
797     return false;
798 }
799
800 int num_valid_moves(Option self, Option opp, Othello *board) {
801     int count = 0, i, j;
802     for (i = 0; i < 8; i++)
803         for (j = 0; j < 8; j++)
804             if (isLegalMove(self, opp, board, i, j)) count++;
805     return count;
806 }
807
808 int Othello::Heuristic_Evaluation_Function(Othello *board, enum Option player)
809 {
810     /*
811     int value = 0;
812     int i, j;
813     for (i = 0; i<6; i++)
814     {
815         for (j = 0; j<6; j++)
816         {
817             value += board->cell[i][j].color == player;
818         }
819     }
820
821     value += 100 * board->Rule(board, player);
822
823     value += 1000 * (board->cell[0][0].color == player);
824     value += 1000 * (board->cell[0][5].color == player);
825     value += 1000 * (board->cell[5][0].color == player);
826     value += 1000 * (board->cell[5][5].color == player);
827
828     return value;
```

```
829     */
830     int my_tiles = 0, opp_tiles = 0, i, j, k, my_front_tiles = 0, opp_front_tiles = 0,
        x, y;
831     double p = 0, c = 0, l = 0, m = 0, f = 0, d = 0;
832
833     int X1[] = {-1, -1, 0, 1, 1, 1, 0, -1};
834     int Y1[] = {0, 1, 1, 1, 0, -1, -1, -1};
835     int V[6][6] = {{20, -3, 8, 8, -3, 20},
836                    {-3, -7, 1, 1, -7, -3},
837                    {8, 1, -3, -3, 1, 8},
838                    {8, 1, -3, -3, 1, 8},
839                    {-3, -7, 1, 1, -7, -3},
840                    {20, -3, 8, 8, -3, 20}};
841
842     // Piece difference, frontier disks and disk squares
843     for(i=0; i<6; i++)
844         for(j=0; j<6; j++)  {
845             if(board->cell[i][j].color == player)  {
846                 d += V[i][j];
847                 my_tiles++;
848             } else if(board->cell[i][j].color == (enum Option)-player)  {
849                 d -= V[i][j];
850                 opp_tiles++;
851             }
852             if(board->cell[i][j].color != SPACE)  {
853                 for(k=0; k<8; k++)  {
854                     x = i + X1[k]; y = j + Y1[k];
855                     if(x >= 0 && x < 6 && y >= 0 && y < 6 && board->cell[x][y].color
    == SPACE) {
856                         if(board->cell[i][j].color == player)  my_front_tiles++;
857                         else opp_front_tiles++;
858                         break;
859                     }
860                 }
861             }
862         }
863     if(my_tiles > opp_tiles)
864         p = (100.0 * my_tiles)/(my_tiles + opp_tiles);
865     else if(my_tiles < opp_tiles)
866         p = -(100.0 * opp_tiles)/(my_tiles + opp_tiles);
867     else p = 0;
```

```
868
869     if(my_front_tiles > opp_front_tiles)
870         f = -(100.0 * my_front_tiles)/(my_front_tiles + opp_front_tiles);
871     else if(my_front_tiles < opp_front_tiles)
872         f = (100.0 * opp_front_tiles)/(my_front_tiles + opp_front_tiles);
873     else f = 0;
874
875 // Corner occupancy
876     my_tiles = opp_tiles = 0;
877     if(board->cell[0][0].color == player) my_tiles++;
878     else if(board->cell[0][0].color == (enum Option)-player) opp_tiles++;
879     if(board->cell[0][5].color == player) my_tiles++;
880     else if(board->cell[0][5].color == (enum Option)-player) opp_tiles++;
881     if(board->cell[5][0].color == player) my_tiles++;
882     else if(board->cell[5][0].color == (enum Option)-player) opp_tiles++;
883     if(board->cell[5][5].color == player) my_tiles++;
884     else if(board->cell[5][5].color == (enum Option)-player) opp_tiles++;
885     c = 25 * (my_tiles - opp_tiles);
886
887     // Corner occupancy
888     my_tiles = opp_tiles = 0;
889     if (board->cell[0][0].color == player) my_tiles++;
890     else if (board->cell[0][0].color == (enum Option) - player) opp_tiles++;
891     if (board->cell[0][5].color == player) my_tiles++;
892     else if (board->cell[0][5].color == (enum Option) - player) opp_tiles++;
893     if (board->cell[5][0].color == player) my_tiles++;
894     else if (board->cell[5][0].color == (enum Option) - player) opp_tiles++;
895     if (board->cell[5][5].color == player) my_tiles++;
896     else if (board->cell[5][5].color == (enum Option) - player) opp_tiles++;
897     c = 25 * (my_tiles - opp_tiles);
898
899     // Corner closeness
900     my_tiles = opp_tiles = 0;
901     if (board->cell[0][0].color != SPACE) {
902         if (board->cell[0][1].color == player) my_tiles++;
903         else if (board->cell[0][1].color == (enum Option) - player) opp_tiles++;
904         if (board->cell[1][1].color == player) my_tiles++;
905         else if (board->cell[1][1].color == (enum Option) - player) opp_tiles++;
906         if (board->cell[1][0].color == player) my_tiles++;
907         else if (board->cell[1][0].color == (enum Option) - player) opp_tiles++;
908     }
```

```
909     if (board->cell[0][5].color != SPACE) {
910         if (board->cell[0][4].color == player) my_tiles++;
911         else if (board->cell[0][4].color == (enum Option) - player) opp_tiles++;
912         if (board->cell[1][5].color == player) my_tiles++;
913         else if (board->cell[1][5].color == (enum Option) - player) opp_tiles++;
914         if (board->cell[1][4].color == player) my_tiles++;
915         else if (board->cell[1][4].color == (enum Option) - player) opp_tiles++;
916     }
917     if (board->cell[5][0].color != SPACE) {
918         if (board->cell[5][1].color == player) my_tiles++;
919         else if (board->cell[5][1].color == (enum Option) - player) opp_tiles++;
920         if (board->cell[1][5].color == player) my_tiles++;
921         else if (board->cell[4][1].color == (enum Option) - player) opp_tiles++;
922         if (board->cell[1][4].color == player) my_tiles++;
923         else if (board->cell[4][0].color == (enum Option) - player) opp_tiles++;
924     }
925     if (board->cell[5][5].color != SPACE) {
926         if (board->cell[4][5].color == player) my_tiles++;
927         else if (board->cell[4][5].color == (enum Option) - player) opp_tiles++;
928         if (board->cell[4][4].color == player) my_tiles++;
929         else if (board->cell[4][4].color == (enum Option) - player) opp_tiles++;
930         if (board->cell[5][4].color == player) my_tiles++;
931         else if (board->cell[5][4].color == (enum Option) - player) opp_tiles++;
932     }
933     l = -12.5 * (my_tiles - opp_tiles);
934 
935 // Mobility
936     my_tiles = num_valid_moves(player, (enum Option)-player, board);
937     opp_tiles = num_valid_moves((enum Option)-player, player, board);
938     if(my_tiles > opp_tiles)
939         m = (100.0 * my_tiles)/(my_tiles + opp_tiles);
940     else if(my_tiles < opp_tiles)
941         m = -(100.0 * opp_tiles)/(my_tiles + opp_tiles);
942     else m = 0;
943 
944 // final weighted score
945     double score = (10 * p) + (801.724 * c) + (38.2026 * l) + (78.922 * m) + (74.396 *
      f) + (10 * d);
946     return (int)score;
947 }
```

4 Results

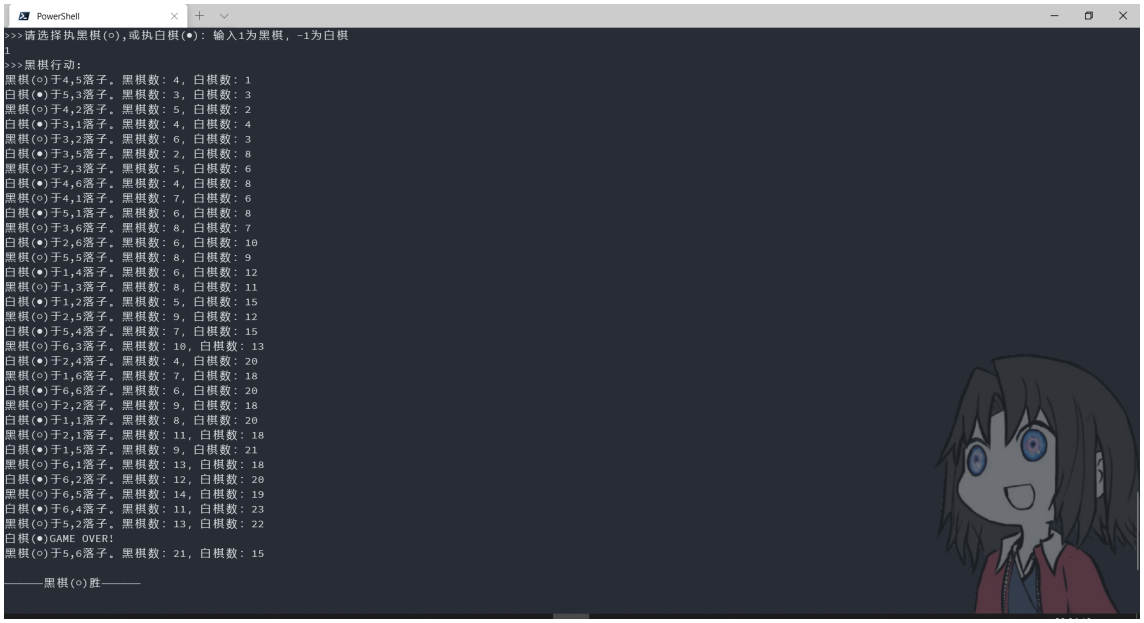

Figure 2: result1

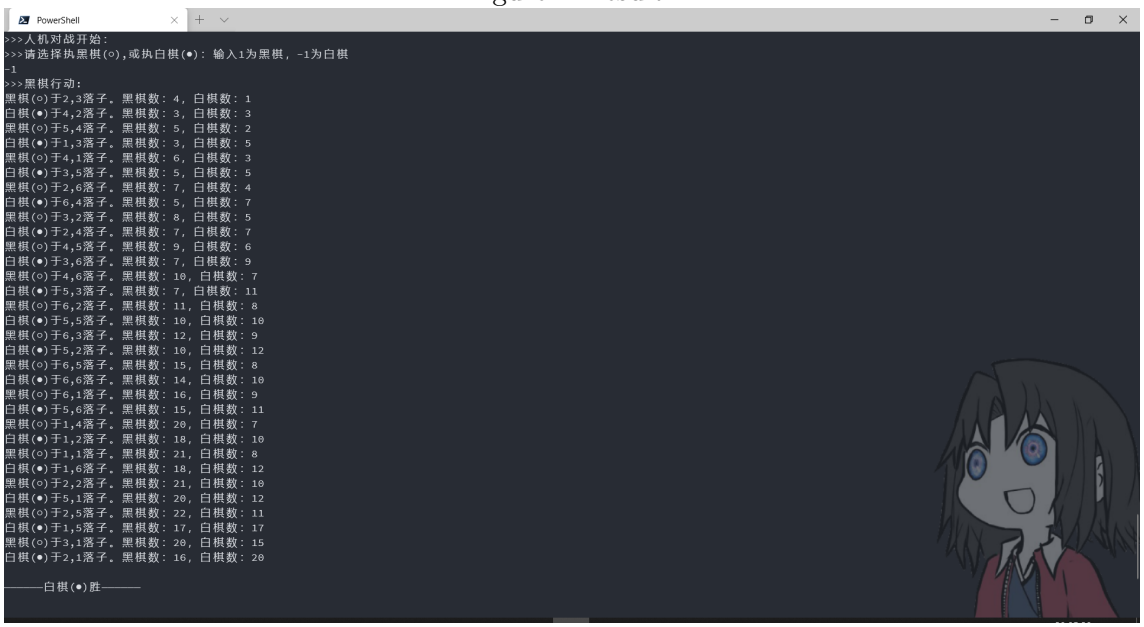

Figure 3: result2