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

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

18340149 孙新梦

September 14, 2020

目录

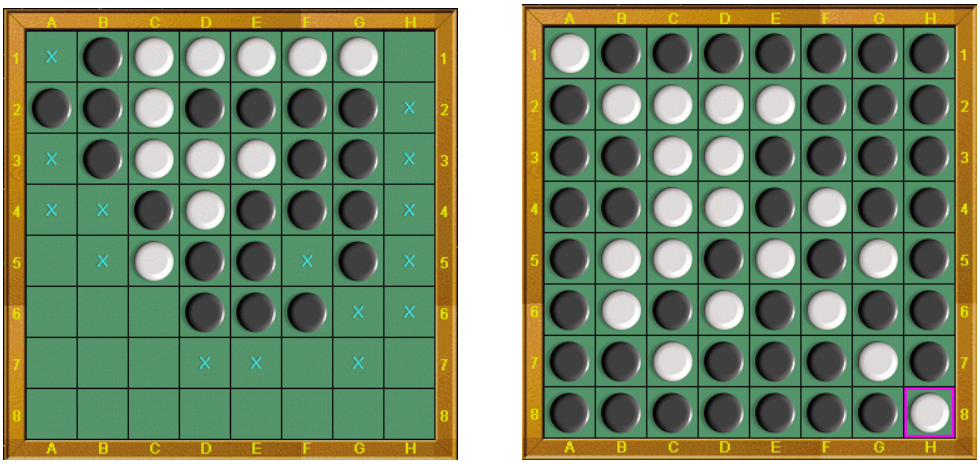

图 1: Othello Game

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 My Analasis

α β 剪枝算法介绍

1. 在很多下棋类AI，也就是代表性的零和博弈问题，如中国象棋和五子棋围棋等游戏，AI会使用对当前棋盘状态的观察，选择一个执行方式使得自己的利益最大化，对方的利益最小化。

2. 博弈游戏的基本元素有：两个玩家(min 和 max)，有穷状态空间，初始状态，终止状态的集合，后继节点，效益值等。

3. 首先是极大化极小的方法，假设双方的玩家都选择最优步，对于max玩家，会选择当前利益最大化的步，而min会选择使利益最小的步，这样一来，对于一个状态转换树，到了终止状态我们可以判断出得到的分值，我们通过对不同轮次，选择max下方最大的收益，min下方最小的收益来作为下一步，会得到一条获得最优解的决策路径。

4. α - β 剪枝算法的改进之处在于，可以在搜索的时候，提前判断出某些分支是不需要继续探测的，也就是对于明显不可能取这条路径的分支我们直接回溯，不再考虑之后的分支。

5. 对于每个节点，可以同时维持alpha和beta两个值(其实只关心max的 α 和min的 β)，如果一个节点的 α ≥祖先的 β ，则可以剪枝，同理，如果一个节点的 β 小于等于祖先的 α ，则剪枝，剪枝就是不再关心节点的其他后继，直接回溯。

形式化本问题为博弈树搜索问题

1. 玩家：假定MAX是我们，选择BALCK方，机器选择WHITE方。

2. 状态：定义状态是棋盘的一种落子情况

3. 初始状态：空棋盘

4. 结束状态：有一方无法继续落子的棋盘状态

5. 后继节点：对于一个状态，他的后继节点是下一步（对手）的不同下法，由于下子必须保证至少有一颗棋子被夹住，变为己方棋子，那么后继节点就一句这个规则产生。

6. 效益值：观察到给出的代码模板给出的Judge函数判定棋盘得分是通过不同位置的权重加和得到，我们修改了给出的另一个函数————dynamicheuristicevaluationfunction，它有更多的特征值来判断棋盘，应该可以得到更高的判断准确度。但是修改他为判断6*6棋盘之后参数也需要进一步调整。用这个函数可以判断终止状态的得分值，作为启发式函数。

具体实现思路

1. 首先修改启发式评价函数为支持6*6的棋盘，修改相关参数。

2. 再把原先手动输入坐标的模块修改为用MyFind函数寻找下一步的方法，实现MyFind函数即 α - β 剪枝函数的思路如下： 1.判断若没有可以落子的数量，则让对方走，若对方也没有地方落子，则游戏结束。 2.由于搜索深度限制，搜索到相应深度之后，直接返回得分，也就是我们搜索状态树的叶子节点。3.若不是以上两种情况，我们寻找可以加入的后继节点，从外到里加入到allChoice队列中 4.对每个后继选择进行遍历，在保存当前棋盘之后继续递归调用MyFind函数寻找下一个选择，用DFS一层层到底。的如果判断若当前玩家为MAX，则不断更新max，并赋值choice的位置和得分，之后进行alpha剪枝，如果这个选择的分数比beta大，则break不再继续便利后继节点，之后更新alpha。若为min，则相反，更新min，赋值choice总是为min对应的选项，beta剪枝，更新beta 5.得到的choice对应在这个深度下的最好选择，作为下一步落子的位置。

4 Codes

```
1  /*
2  *    程序名：Othello.cpp
3  *    姓名：孙新梦
4  *    学号：18340149
5  *    作业要求：实现alpha-beta剪枝算法
6  *    编译：g++ -o Othello Othello.cpp
7  *    运行: ./Othello.cpp
8  *    环境：windows10 64bits
9  */
10 #include <iostream>
11 #include <stdlib.h>
12
13 using namespace std;
14
```

```
15 int const MAX = 65534;
16
17 int  deepth = 12;          //最大搜索深度 （可调节）
18
19 //基本元素 棋子，颜色，数字变量
20 enum Option
21 {
22     WHITE = -1, SPACE, BLACK     //是否能落子 黑子 //=1
23 };
24
25 struct Do
26 {
27     pair<int, int > pos;
28     int score;
29 };
30
31 struct WinNum
32 {
33     enum Option color;
34     int stable;                  // 此次落子赢棋个数
35 };
36
37
38
39 //主要功能 棋盘及关于棋子的所有操作，功能
40 struct Othello
41 {
42
43     WinNum cell[6][6];
            //定义棋盘中有个格
        子6*6
44     int whiteNum;
            //白棋数
        目
```

```
45      int blackNum;
                //黑棋
            数
46
47
48      void Create(Othello* board);
                //初始化棋
            盘
49      void Copy(Othello* boardDest, const  Othello* boardSource);
                //复制棋
            盘
50      void Show(Othello* board);
                //显示棋
            盘
51      int Rule(Othello* board, enum Option player);
                //判断落子是否符合规
            则
52      int Action(Othello* board, Do* choice, enum Option player);
                //落子并修改棋
            盘,
53      void Stable(Othello* board);
                //计算赢棋个
            数
54      int Judge(Othello* board, enum Option player);
                //计算本次落子分
            数
55  };//主要功能
56
57  double dynamic_heuristic_evaluation_function(Othello* board, enum
        Option myColor);//新的启发式函
        数
58
59  /*函数说明：函数
60      MyFind输入：棋盘状态，玩家方，步数，
61      beta,alpha下一步选择,输出：下一步选择注：为玩家，为玩家
62
63      BLACKMAXWHITEMIN
64  */
```

```
65 Do* MyFind(Othello* board, enum Option player, int step, int alpha, int
       beta, Do* choice)
66 {
67     int i, j, k, num;
68
69     Do* allChoices;
70     choice->score = -MAX;
71     choice->pos.first = -1;
72     choice->pos.second = -1;
73
74     num = board->Rule(board, player);//可以落的数量
75     if (num == 0)     /* 无处落子 terminal state*/
76     {
77         if (board->Rule(board, (enum Option) - player))     /* 对方可以落
               子让对方下,.*/
78         {
79             Othello tempBoard;
80             Do nextChoice;
81             Do* pNextChoice = &nextChoice;
82             board->Copy(&tempBoard, board);
83             pNextChoice = MyFind(&tempBoard, (enum Option) - player,
                   step - 1, alpha, beta, pNextChoice);
84             choice->score = -pNextChoice->score;
85             choice->pos.first = -1;
86             choice->pos.second = -1;
87             return choice;
88         }
89         else     /* 对方也无处落子游戏结束,. */
90         {
91             int value = WHITE * (board->whiteNum) + BLACK * (board->
                   blackNum);
92             if (player * value > 0)
93             {
94                 choice->score = MAX - 1;
```

```
95              }
96              else if (player * value < 0)
97              {
98                  choice->score = -MAX + 1;
99              }
100             else
101             {
102                 choice->score = 0;
103             }
104             return choice;
105         }
106     }
107
108     if (step <= 0)      /* 已经考虑到步step直接返回得分, */
109     {
110         choice->score = (int)dynamic_heuristic_evaluation_function(
                board, player);
111         return choice;
112     }
113
114     //接下来开始寻找后继，加入到这个队列中去，一圈又一圈allChoice
115     allChoices = (Do*)malloc(sizeof(Do) * num);
116     k = 0;
117     for (i = 0; i < 6; i++)//外圈
118     {
119         for (j = 0; j < 6; j++)
120         {
121             if (i == 0 || i == 5 || j == 0 || j == 5)
122             {
123                 if (board->cell[i][j].color == SPACE && board->cell[i][
                        j].stable)
124                 {
125                     allChoices[k].score = -MAX;
```

```
126                     allChoices[k].pos.first = i;
127                     allChoices[k].pos.second = j;
128                     k++;
129                 }
130             }
131         }
132     }
133
134     for (i = 0; i < 6; i++)//中圈
135     {
136         for (j = 0; j < 6; j++)
137         {
138             if ((i == 2 || i == 3 || j == 2 || j == 3) && (i >= 2 && i
                    <= 3 && j >= 2 && j <= 3))
139             {
140                 if (board->cell[i][j].color == SPACE && board->cell[i][
                        j].stable)
141                 {
142                     allChoices[k].score = -MAX;
143                     allChoices[k].pos.first = i;
144                     allChoices[k].pos.second = j;
145                     k++;
146                 }
147             }
148         }
149     }
150
151     for (i = 0; i < 6; i++)//內圈
152     {
153         for (j = 0; j < 6; j++)
154         {
155             if ((i == 1 || i == 4 || j == 1 || j == 4) && (i >= 1 && i
                    <= 4 && j >= 1 && j <= 4))
```

```
156             {
157                 if (board->cell[i][j].color == SPACE && board->cell[i][
                        j].stable)
158                 {
159                     allChoices[k].score = -MAX;
160                     allChoices[k].pos.first = i;
161                     allChoices[k].pos.second = j;
162                     k++;
163                 }
164             }
165         }
166     }
167
168     int min = MAX;
169     int max = -MAX;
170
171     for (k = 0; k < num; k++)
172     {
173         Othello tempBoard;
174         Do thisChoice, nextChoice;
175         Do* pNextChoice = &nextChoice;
176
177         thisChoice.score = allChoices[k].score;
178         thisChoice.pos = allChoices[k].pos;
179
180         board->Copy(&tempBoard, board);//保存当前的棋盘
181         board->Action(&tempBoard, &thisChoice, player);//改变棋局
182         pNextChoice = MyFind(&tempBoard, (enum Option) - player, step -
                1, alpha, beta, pNextChoice);
183
184         if (player == BLACK)
185         {
186             if (pNextChoice->score > max)
```

```
187             {
188                 max = pNextChoice->score;
189                 choice->score = max;
190                 choice->pos = thisChoice.pos;
191             }
192             //alpha pruning
193             if (pNextChoice->score >= beta)
194                 break;
195             //update alpha
196             if (pNextChoice->score > alpha)
197                 alpha = pNextChoice->score;
198         }
199         else if (player == WHITE)
200         {
201             if (pNextChoice->score < min)
202             {
203                 min = pNextChoice->score;
204                 choice->score = min;
205                 choice->pos = thisChoice.pos;
206             }
207             //beta pruning
208             if (pNextChoice->score <= beta)
209                 break;
210             //beta update
211             if (pNextChoice->score < beta)
212                 beta = pNextChoice->score;
213         }
214     }
215     free(allChoices);
216     return choice;
217 }
218
219
```

```
220
221
222 //给出的函数Find
223 Do* Find(Othello* board, enum Option player, int step, int min, int max
        , Do* choice)
224 {
225     int i, j, k, num;
226     Do* allChoices;
227     choice->score = -MAX;
228     choice->pos.first = -1;
229     choice->pos.second = -1;
230
231     num = board->Rule(board, player);
232     if (num == 0)     /* 无处落子 */
233     {
234         if (board->Rule(board, (enum Option) - player))     /* 对方可以落
                子让对方下,.*/
235         {
236             Othello tempBoard;
237             Do nextChoice;
238             Do* pNextChoice = &nextChoice;
239             board->Copy(&tempBoard, board);
240             pNextChoice = Find(&tempBoard, (enum Option) - player, step
                    - 1, -max, -min, pNextChoice);
241             choice->score = -pNextChoice->score;
242             choice->pos.first = -1;
243             choice->pos.second = -1;
244             return choice;
245         }
246         else     /* 对方也无处落子游戏结束,. */
247         {
248             int value = WHITE * (board->whiteNum) + BLACK * (board->
                    blackNum);
249             if (player * value > 0)
```

```
250             {
251                 choice->score = MAX - 1;
252             }
253             else if (player * value < 0)
254             {
255                 choice->score = -MAX + 1;
256             }
257             else
258             {
259                 choice->score = 0;
260             }
261             return choice;
262         }
263     }
264     if (step <= 0)     /* 已经考虑到步step直接返回得分, */
265     {
266         choice->score = board->Judge(board, player);
267         return choice;
268     }
269
270     allChoices = (Do*)malloc(sizeof(Do) * num);
271     k = 0;
272     for (i = 0; i < 6; i++)
273     {
274         for (j = 0; j < 6; j++)
275         {
276             if (i == 0 || i == 5 || j == 0 || j == 5)
277             {
278                 if (board->cell[i][j].color == SPACE && board->cell[i][
                        j].stable)
279                 {
280                     allChoices[k].score = -MAX;
281                     allChoices[k].pos.first = i;
```

```
282                     allChoices[k].pos.second = j;
283                     k++;
284                 }
285             }
286         }
287     }
288
289     for (i = 0; i < 6; i++)
290     {
291         for (j = 0; j < 6; j++)
292         {
293             if ((i == 2 || i == 3 || j == 2 || j == 3) && (i >= 2 && i
                    <= 3 && j >= 2 && j <= 3))
294             {
295                 if (board->cell[i][j].color == SPACE && board->cell[i][
                        j].stable)
296                 {
297                     allChoices[k].score = -MAX;
298                     allChoices[k].pos.first = i;
299                     allChoices[k].pos.second = j;
300                     k++;
301                 }
302             }
303         }
304     }
305
306     for (i = 0; i < 6; i++)
307     {
308         for (j = 0; j < 6; j++)
309         {
310             if ((i == 1 || i == 4 || j == 1 || j == 4) && (i >= 1 && i
                    <= 4 && j >= 1 && j <= 4))
311             {
```

```
312                 if (board->cell[i][j].color == SPACE && board->cell[i][
                        j].stable)
313                 {
314                     allChoices[k].score = -MAX;
315                     allChoices[k].pos.first = i;
316                     allChoices[k].pos.second = j;
317                     k++;
318                 }
319             }
320         }
321     }
322
323     for (k = 0; k < num; k++)
324     {
325         Othello tempBoard;
326         Do thisChoice, nextChoice;
327         Do* pNextChoice = &nextChoice;
328
329         thisChoice.score = allChoices[k].score;
330         thisChoice.pos = allChoices[k].pos;
331
332         board->Copy(&tempBoard, board);
333         board->Action(&tempBoard, &thisChoice, player);
334         pNextChoice = Find(&tempBoard, (enum Option) - player, step -
                1, -max, -min, pNextChoice);
335         thisChoice.score = -pNextChoice->score;
336
337         if (thisChoice.score > min && thisChoice.score < max)     /* 可
                以预计的更优值 */
338         {
339             min = thisChoice.score;
340             choice->score = thisChoice.score;
341             choice->pos.first = thisChoice.pos.first;
342             choice->pos.second = thisChoice.pos.second;
```

```
343             }
344             else if (thisChoice.score >= max)     /* 好的超乎预计 */
345             {
346                 choice->score = thisChoice.score;
347                 choice->pos.first = thisChoice.pos.first;
348                 choice->pos.second = thisChoice.pos.second;
349                 break;
350             }
351             /* 不如已知最优值 */
352         }
353         free(allChoices);
354         return choice;
355     }
356
357     int main()
358     {
359         Othello board;
360         Othello* pBoard = &board;
361         enum Option player, present;
362         Do choice;
363         Do* pChoice = &choice;
364         int  num, result = 0;
365         char restart = 'y';
366         int round = 0;
367
368     start:
369         player = SPACE;
370         present = BLACK;
371         num = 4;
372         restart = 'y';
373
374         cout << ">>>的和机器对战开始：IdaAI\n";
375
```

```
376
377
378     /*
379     while (player != WHITE && player != BLACK)
380     {
381         cout << 请选择执黑棋○">>>()或执白棋● ,(): 输入为黑棋，为白
                棋1-1" << endl;
382         //scanf("%d", &player);
383
384         cout << 黑棋行动">>>:  \n";
385
386
387         if (player != WHITE && player != BLACK)
388         {
389             cout << 输入不符合规范，请重新输入"\n";
390         }
391     }*/
392
393     player = BLACK;//直接选择玩家为黑子
394
395     board.Create(pBoard);
396
397     while (num < 36)
                // 棋盘上未下满
            子36
398     {
399         cout << endl<<endl<<"----------ROUND  " << ++round <<"----------"
                << endl;
400         string Player ;
401         if (present == BLACK)
402         {
403             Player = "黑棋○()";
404         }
405         else if (present == WHITE)
```

```
406         {
407             Player = "白棋●()";
408         }
409
410         if (board.Rule(pBoard, present) == 0)                    //
                未下满并且无子可
                下
411         {
412             if (board.Rule(pBoard, (enum Option) - present) == 0)
413             {
414                 break;
415             }
416
417             cout << Player << "GAME_OVER! \n";
418         }
419         else
420         {
421             int i, j;
422             board.Show(pBoard);
423
424             if (present == player)
425             {
426                 /*while (1)
427                 {
428                     cout << Player << " \n 请输入棋子坐标（空格相隔>>> 如
                        “3 ”代表第行第列）535:\n";
429
430                     cin >> i>> j;
431                     i--;
432                     j--;
433                     pChoice->pos.first = i;
434                     pChoice->pos.second = j;
435
```

```
436                 if (i<0 || i>5 || j<0 || j>5 || pBoard->cell[i][j].
                        color != SPACE || pBoard->cell[i][j].stable ==
                        0)
437                 {
438                     cout 此处落子不符合规则，请重新选择<<">>>    \n";
439                     board.Show(pBoard);
440                 }
441                 else
442                 {
443                     break;
444                 }
445             }
446             system("cls");
447             cout << 玩家">>> 本手棋得分
                    为      " << pChoice->score << endl;
448             system("pause");
449             cout << 按任意键继续">>>" << pChoice->score << endl;
450             */
451             cout << Player << "..........................";
452
453             pChoice = MyFind(pBoard, present, deepth, -MAX, MAX,
                    pChoice);
454             i = pChoice->pos.first;
455             j = pChoice->pos.second;
456             //system("cls");
457             cout << "玩家>>>Ida 本手棋得分
                    为     " << pChoice->score << endl;
458         }
459         else    //下棋AI
460         {
461             cout << Player << "..........................";
462
463             pChoice = Find(pBoard, present, deepth, -MAX, MAX,
                    pChoice);
```

```
464                 i = pChoice->pos.first;
465                 j = pChoice->pos.second;
466                 //system("cls");
467                 cout << ">>>AI 本手棋得分
                        为      " << pChoice->score << endl;
468             }
469
470
471             board.Action(pBoard, pChoice, present);
472             num++;
473             cout << Player << ">>>于AI" << i + 1 << "," << j + 1 << "落
                    子，该你了！";
474         }
475
476         present = (enum Option) - present;      //交换执棋者
477     }
478
479
480     board.Show(pBoard);
481
482
483     result = pBoard->whiteNum - pBoard->blackNum;
484
485     if (result > 0)
486     {
487         cout << "\————————白棋n●()胜————————\n";
488     }
489     else if (result < 0)
490     {
491         cout << "\————————黑棋n○()胜————————\n";
492     }
493     else
494     {
495         cout << "\————————————平局————————————n\n";
```

```
496     }
497
498     cout << "\n ----------GAME OVER----------!\n";
499     cout << "\n";
500
501     while (restart != 'Y' && restart != 'N')
502     {
503         cout << "-----------------------------------------||\n";
504         cout << "|                                         | \n";
505         cout << "|                                         |   \n";
506         cout << "|>>>>>>>>>>>>>>>>Again?(Y,N)<<<<<<<<<<<<<<<<|\n";
507         cout << "|                                         | \n";
508         cout << "|                                         |   \n";
509         cout << "-----------------------------------------||\n";
510         cout << "                                           \n";
511         cout << "                                           \n";
512         cout << "                                           \n";
513         cout << "  ---------                 ---------       \n";
514         cout << "  |   YES   |               |   NO    |       \n";
515         cout << "  ---------                 ---------       \n";
516
517         cin >> restart;
518         if (restart == 'Y')
519         {
520             goto start;
521         }
522     }
523
524
525     return 0;
526 }
527
528
```

```
529
530
531
532
533 void Othello::Create(Othello* board)
534 {
535     int i, j;
536     board->whiteNum = 2;
537     board->blackNum = 2;
538     for (i = 0; i < 6; i++)
539     {
540         for (j = 0; j < 6; j++)
541         {
542             board->cell[i][j].color = SPACE;
543             board->cell[i][j].stable = 0;
544         }
545     }
546     board->cell[2][2].color = board->cell[3][3].color = WHITE;
547     board->cell[2][3].color = board->cell[3][2].color = BLACK;
548 }
549
550
551 void Othello::Copy(Othello* Fake, const  Othello* Source)
552 {
553     int i, j;
554     Fake->whiteNum = Source->whiteNum;
555     Fake->blackNum = Source->blackNum;
556     for (i = 0; i < 6; i++)
557     {
558         for (j = 0; j < 6; j++)
559         {
560             Fake->cell[i][j].color = Source->cell[i][j].color;
561             Fake->cell[i][j].stable = Source->cell[i][j].stable;
```

```
562             }
563         }
564 }
565
566 void Othello::Show(Othello* board)
567 {
568     int i, j;
569     cout << "\n  ";
570     for (i = 0; i < 6; i++)
571     {
572         cout << "   " << i + 1;
573     }
574     cout << "\n    ----------------------\n";
575     for (i = 0; i < 6; i++)
576     {
577         cout << i + 1 << " |";
578         for (j = 0; j < 6; j++)
579         {
580             switch (board->cell[i][j].color)
581             {
582             case BLACK:
583                 cout << "○|";
584                 break;
585             case WHITE:
586                 cout << "●|";
587                 break;
588             case SPACE:
589                 if (board->cell[i][j].stable)
590                 {
591                     cout << " |+";
592                 }
593                 else
594                 {
```

```
595                         cout << "␣␣|";
596                     }
597                     break;
598                 default:    /* 棋子颜色错误 */
599                     cout << "*␣|";
600                 }
601             }
602             cout << "\n␣␣␣␣————————————————————\n";
603         }
604 
605         cout << "白棋●>>>()个数为:" << board->whiteNum << "␣␣␣␣␣␣␣␣␣";
606         cout << "黑棋○>>>()个数
                为:" << board->blackNum << endl << endl << endl;
607 }
608 
609 
610 int Othello::Rule(Othello* board, enum Option player)
611 {
612     int i, j;
613     unsigned num = 0;
614     for (i = 0; i < 6; i++)
615     {
616         for (j = 0; j < 6; j++)
617         {
618             if (board->cell[i][j].color == SPACE)
619             {
620                 int x, y;
621                 board->cell[i][j].stable = 0;
622                 for (x = -1; x <= 1; x++)
623                 {
624                     for (y = -1; y <= 1; y++)
625                     {
626                         if (x || y)     /* 个方向8 */
627                         {
```

```
628                             int i2, j2;
629                             unsigned num2 = 0;
630                             for (i2 = i + x, j2 = j + y; i2 >= 0 && i2
                                    <= 5 && j2 >= 0 && j2 <= 5; i2 += x, j2
                                     += y)
631                             {
632                                 if (board->cell[i2][j2].color == (enum
                                        Option) - player)
633                                 {
634                                     num2++;
635                                 }
636                                 else if (board->cell[i2][j2].color ==
                                        player)
637                                 {
638                                     board->cell[i][j].stable += player
                                            * num2;
639                                     break;
640                                 }
641                                 else if (board->cell[i2][j2].color ==
                                        SPACE)
642                                 {
643                                     break;
644                                 }
645                             }
646                         }
647                     }
648                 }
649
650                 if (board->cell[i][j].stable)
651                 {
652                     num++;
653                 }
654             }
```

```
655             }
656         }
657         return num;
658     }
659
660
661     int Othello::Action(Othello* board, Do* choice, enum Option player)
662     {
663         int i = choice->pos.first, j = choice->pos.second;
664         int x, y;
665
666         if (board->cell[i][j].color != SPACE || board->cell[i][j].stable ==
                0 || player == SPACE)
667         {
668             return -1;
669         }
670
671
672         board->cell[i][j].color = player;
673         board->cell[i][j].stable = 0;
674
675
676         if (player == WHITE)
677         {
678             board->whiteNum++;
679         }
680         else if (player == BLACK)
681         {
682             board->blackNum++;
683         }
684
685
686
```

```
687     for (x = -1; x <= 1; x++)
688     {
689         for (y = -1; y <= 1; y++)
690         {
691 
692             //需要在每个方向（个）上检测落子是否符合规则（能否吃子）8
693 
694 
695             if (x || y)
696             {
697                 int i2, j2;
698                 unsigned num = 0;
699                 for (i2 = i + x, j2 = j + y; i2 >= 0 && i2 <= 5 && j2
                        >= 0 && j2 <= 5; i2 += x, j2 += y)
700                 {
701                     if (board->cell[i2][j2].color == (enum Option) -
                            player)
702                     {
703                         num++;
704                     }
705                     else if (board->cell[i2][j2].color == player)
706                     {
707                         board->whiteNum += (player * WHITE) * num;
708                         board->blackNum += (player * BLACK) * num;
709 
710                         for (i2 -= x, j2 -= y; num > 0; num--, i2 -= x,
                                j2 -= y)
711                         {
712                             board->cell[i2][j2].color = player;
713                             board->cell[i2][j2].stable = 0;
714                         }
715                         break;
716                     }
```

```
717                     else if (board->cell[i2][j2].color == SPACE)
718                     {
719                         break;
720                     }
721                 }
722             }
723         }
724     }
725     return 0;
726 }
727
728
729 void Othello::Stable(Othello* board)
730 {
731     int i, j;
732     for (i = 0; i < 6; i++)
733     {
734         for (j = 0; j < 6; j++)
735         {
736             if (board->cell[i][j].color != SPACE)
737             {
738                 int x, y;
739                 board->cell[i][j].stable = 1;
740
741                 for (x = -1; x <= 1; x++)
742                 {
743                     for (y = -1; y <= 1; y++)
744                     {
745                         /* 个方向4 */
746                         if (x == 0 && y == 0)
747                         {
748                             x = 2;
749                             y = 2;
```

```
750                     }
751                     else
752                     {
753                         int i2, j2, flag = 2;
754                         for (i2 = i + x, j2 = j + y; i2 >= 0 && i2
                               <= 5 && j2 >= 0 && j2 <= 5; i2 += x, j2
                                += y)
755                         {
756                             if (board->cell[i2][j2].color != board
                                   ->cell[i][j].color)
757                             {
758                                 flag--;
759                                 break;
760                             }
761                         }
762
763                         for (i2 = i - x, j2 = j - y; i2 >= 0 && i2
                               <= 5 && j2 >= 0 && j2 <= 5; i2 -= x, j2
                                -= y)
764                         {
765                             if (board->cell[i2][j2].color != board
                                   ->cell[i][j].color)
766                             {
767                                 flag--;
768                                 break;
769                             }
770                         }
771
772                         if (flag)     /* 在某一条线上稳定 */
773                         {
774                             board->cell[i][j].stable++;
775                         }
776                     }
```

```
777                         }
778                     }
779                 }
780             }
781         }
782 }
783
784 int Othello::Judge(Othello* board, enum Option player)//函数得分多少Judge
785 {
786     int value = 0;
787     int i, j;
788     Stable(board);
789     for (i = 0; i < 6; i++)
790     {
791         for (j = 0; j < 6; j++)
792         {
793             value += (board->cell[i][j].color) * (board->cell[i][j].
                stable);
794         }
795     }
796
797     value += 64 * board->cell[0][0].color;
798     value += 64 * board->cell[0][5].color;
799     value += 64 * board->cell[5][0].color;
800     value += 64 * board->cell[5][5].color;
801     value -= 32 * board->cell[1][1].color;
802     value -= 32 * board->cell[1][4].color;
803     value -= 32 * board->cell[4][1].color;
804     value -= 32 * board->cell[4][4].color;
805
806     return value * player;
807 }
808
```

```
809 bool canmove(char self, char opp, char* str) {
810     if (str[0] != opp) return false;
811     for (int ctr = 1; ctr < 6; ctr++) {
812         if (str[ctr] == '-') return false;
813         if (str[ctr] == self) return true;
814     }
815     return false;
816 }
817 
818 bool isLegalMove(char self, char opp, char grid[6][6], int startx, int
        starty) {
819     if (grid[startx][starty] != '-') return false;
820     char str[10];
821     int x, y, dx, dy, ctr;
822     for (dy = -1; dy <= 1; dy++)
823         for (dx = -1; dx <= 1; dx++) {
824             // keep going if both velocities are zero
825             if (!dy && !dx) continue;
826             str[0] = '\0';
827             for (ctr = 1; ctr < 6; ctr++) {
828                 x = startx + ctr * dx;
829                 y = starty + ctr * dy;
830                 if (x >= 0 && y >= 0 && x < 6 && y < 6) str[ctr - 1] =
                        grid[x][y];
831                 else str[ctr - 1] = 0;
832             }
833             if (canmove(self, opp, str)) return true;
834         }
835     return false;
836 }
837 
838 int num_valid_moves(char self, char opp, char grid[6][6]) {
839     int count = 0, i, j;
```

```
840     for (i = 0; i < 6; i++)
841         for (j = 0; j < 6; j++)
842             if (isLegalMove(self, opp, grid, i, j)) count++;
843     return count;
844 }
845
846 /*
847  * Assuming my_color stores your color and opp_color stores opponent's
        color
848  * '-' indicates an empty square on the board
849  * 'b' indicates a black tile and 'w' indicates a white tile on the
        board
850  */
851 double dynamic_heuristic_evaluation_function(Othello* board, enum
        Option myColor)
852 {
853     int my_tiles = 0, opp_tiles = 0, i, j, k, my_front_tiles = 0,
            opp_front_tiles = 0, x, y;
854     double p = 0, c = 0, l = 0, m = 0, f = 0, d = 0;
855
856     char my_color = (myColor == 1) ? 'b' : 'w';
857     char opp_color = (myColor == 1) ? 'w' : 'b';
858
859     int X1[] = { -1, -1, 0, 1, 1, 1, 0, -1 };
860     int Y1[] = { 0, 1, 1, 1, 0, -1, -1, -1 };
861
862     int V[6][6] =
863     {
864         {20, -3, 8, 8,  -3, 20},
865         {-3, -7, 1, 1,  -7, -3},
866         {8, 1, -3, -3, 1, 8},
867         {8, 1, -3, -3, 1, 8},
868         {-3, -7,  1, 1,  -7, -3},
```

```
869             {20, -3,  8, 8,  -3, 20}
870         };
871
872         char grid[6][6];
873         for (int i = 0; i < 6; i++)
874         {
875             for (int j = 0; j < 6; j++)
876             {
877                 if (board->cell[i][j].color == BLACK)
878                     grid[i][j] = 'b';
879                 else if (board->cell[i][j].color == WHITE)
880                     grid[i][j] = 'w';
881                 else
882                     grid[i][j] = '-';
883             }
884         }
885
886
887         // Piece difference, frontier disks and disk squares
888         for (i = 0; i < 6; i++)
889             for (j = 0; j < 6; j++)
890             {
891                 if (grid[i][j] == my_color)
892                 {
893                     d += V[i][j];
894                     my_tiles++;
895                 }
896                 else if (grid[i][j] == opp_color) {
897                     d -= V[i][j];
898                     opp_tiles++;
899                 }
900                 if (grid[i][j] != '-')
901                 {
```

```
902                 for (k = 0; k < 6; k++)
903                 {
904                     x = i + X1[k]; y = j + Y1[k];
905                     if (x >= 0 && x < 6 && y >= 0 && y < 6 && grid[x][y
                            ] == '-') {
906                         if (grid[i][j] == my_color)  my_front_tiles++;
907                         else opp_front_tiles++;
908                         break;
909                     }
910                 }
911             }
912         }
913     if (my_tiles > opp_tiles)
914         p = (100.0 * my_tiles) / (my_tiles + opp_tiles);
915     else if (my_tiles < opp_tiles)
916         p = -(100.0 * opp_tiles) / (my_tiles + opp_tiles);
917     else p = 0;
918
919     if (my_front_tiles > opp_front_tiles)
920         f = -(100.0 * my_front_tiles) / (my_front_tiles +
               opp_front_tiles);
921     else if (my_front_tiles < opp_front_tiles)
922         f = (100.0 * opp_front_tiles) / (my_front_tiles +
               opp_front_tiles);
923     else f = 0;
924
925     // Corner occupancy
926     my_tiles = opp_tiles = 0;
927     if (grid[0][0] == my_color) my_tiles++;
928     else if (grid[0][0] == opp_color) opp_tiles++;
929     if (grid[0][5] == my_color) my_tiles++;
930     else if (grid[0][5] == opp_color) opp_tiles++;
931     if (grid[5][0] == my_color) my_tiles++;
```

```
932     else if (grid[5][0] == opp_color) opp_tiles++;
933     if (grid[5][5] == my_color) my_tiles++;
934     else if (grid[5][5] == opp_color) opp_tiles++;
935     c = 25 * (my_tiles - opp_tiles);
936
937     // Corner closeness
938     my_tiles = opp_tiles = 0;
939     if (grid[0][0] == '-') {
940         if (grid[0][1] == my_color) my_tiles++;
941         else if (grid[0][1] == opp_color) opp_tiles++;
942         if (grid[1][1] == my_color) my_tiles++;
943         else if (grid[1][1] == opp_color) opp_tiles++;
944         if (grid[1][0] == my_color) my_tiles++;
945         else if (grid[1][0] == opp_color) opp_tiles++;
946     }
947     if (grid[0][5] == '-') {
948         if (grid[0][4] == my_color) my_tiles++;
949         else if (grid[0][4] == opp_color) opp_tiles++;
950         if (grid[1][4] == my_color) my_tiles++;
951         else if (grid[1][4] == opp_color) opp_tiles++;
952         if (grid[1][5] == my_color) my_tiles++;
953         else if (grid[1][5] == opp_color) opp_tiles++;
954     }
955     if (grid[5][0] == '-') {
956         if (grid[5][1] == my_color) my_tiles++;
957         else if (grid[5][1] == opp_color) opp_tiles++;
958         if (grid[4][1] == my_color) my_tiles++;
959         else if (grid[4][1] == opp_color) opp_tiles++;
960         if (grid[4][0] == my_color) my_tiles++;
961         else if (grid[4][0] == opp_color) opp_tiles++;
962     }
963     if (grid[5][5] == '-') {
964         if (grid[4][5] == my_color) my_tiles++;
```

```
965             else if (grid[4][5] == opp_color) opp_tiles++;
966             if (grid[4][4] == my_color) my_tiles++;
967             else if (grid[4][4] == opp_color) opp_tiles++;
968             if (grid[5][4] == my_color) my_tiles++;
969             else if (grid[5][4] == opp_color) opp_tiles++;
970         }
971         l = -12.5 * (my_tiles - opp_tiles);
972 
973         // Mobility
974         my_tiles = num_valid_moves(my_color, opp_color, grid);
975         opp_tiles = num_valid_moves(opp_color, my_color, grid);
976         if (my_tiles > opp_tiles)
977             m = (100.0 * my_tiles) / (my_tiles + opp_tiles);
978         else if (my_tiles < opp_tiles)
979             m = -(100.0 * opp_tiles) / (my_tiles + opp_tiles);
980         else m = 0;
981 
982         // final weighted score
983         double score = (10 * p) + (801.724 * c) + (382.026 * l) + (78.922 *
                m) + (74.396 * f) + (10 * d);
984         return score;
985     }
```

5 Results

这里省略了中间的步骤，并且加入了轮次的输出，可以看到在三十多轮的对决之后，游戏结束。

```
>>>Ida的AI和机器对战开始:

--------ROUND  1---------
     1   2   3   4   5   6
   ┌───┬───┬───┬───┬───┬───┐
1--│   │   │   │   │   │   │
   ├───┼───┼───┼───┼───┼───┤
2--│   │   │ + │   │   │   │
   ├───┼───┼───┼───┼───┼───┤
3--│   │ + │● │○ │   │   │
   ├───┼───┼───┼───┼───┼───┤
4--│   │   │○ │● │ + │   │
   ├───┼───┼───┼───┼───┼───┤
5--│   │   │   │ + │   │   │
   ├───┼───┼───┼───┼───┼───┤
6--│   │   │   │   │   │   │
   └───┴───┴───┴───┴───┴───┘
>>>白棋(●)个数为:2          >>>黑棋(○)个数为:2

黑棋(○).........................>>>玩家Ida 本手棋得分为      49456
黑棋(○)>>>AI于2,3落子，该你了！

--------ROUND  2---------
     1   2   3   4   5   6
   ┌───┬───┬───┬───┬───┬───┐
1--│   │   │   │   │   │   │
   ├───┼───┼───┼───┼───┼───┤
2--│   │ + │○ │ + │   │   │
   ├───┼───┼───┼───┼───┼───┤
3--│   │   │○ │○ │   │   │
   └───┴───┴───┴───┴───┴───┘
```

```
--------ROUND  34---------
     1   2   3   4   5   6
   ┌───┬───┬───┬───┬───┬───┐
1--│● │● │● │● │● │● │
   ├───┼───┼───┼───┼───┼───┤
2--│● │● │● │○ │● │● │
   ├───┼───┼───┼───┼───┼───┤
3--│● │● │○ │○ │○ │● │
   ├───┼───┼───┼───┼───┼───┤
4--│● │ + │○ │○ │● │● │
   ├───┼───┼───┼───┼───┼───┤
5--│● │○ │● │● │● │● │
   ├───┼───┼───┼───┼───┼───┤
6--│○ │● │● │● │● │● │
   └───┴───┴───┴───┴───┴───┘
>>>白棋(●)个数为:27          >>>黑棋(○)个数为:8

白棋(●).........................>>>AI 本手棋得分为      65533
白棋(●)>>>AI于4,2落子，该你了！
     1   2   3   4   5   6
   ┌───┬───┬───┬───┬───┬───┐
1--│● │● │● │● │● │● │
   ├───┼───┼───┼───┼───┼───┤
2--│● │● │● │● │● │● │
   ├───┼───┼───┼───┼───┼───┤
3--│● │● │● │○ │○ │● │
   ├───┼───┼───┼───┼───┼───┤
4--│● │● │● │● │● │● │
   ├───┼───┼───┼───┼───┼───┤
5--│● │● │● │● │● │● │
   ├───┼───┼───┼───┼───┼───┤
6--│○ │● │● │● │● │● │
   └───┴───┴───┴───┴───┴───┘
>>>白棋(●)个数为:33          >>>黑棋(○)个数为:3
```

```
————————白棋(●)胜————————

  ——————————GAME OVER!——————————

┌──────────────────────────────────┐
│                                  │
│ >>>>>>>>>>>>>>>Again?(Y,N)<<<<<<<<<<<<<<< │
│                                  │
└──────────────────────────────────┘
```

6 What I learnt

1. 首先是对于0和博弈的minmax和alpha-beta剪枝算法有了更深的理解，在双方都是聪明人的情况下，我们是能够通过一层一层推理对方的步数和我们自己的步数来得到让自己利益最大化的一步的

2. 剪枝可以节省很多的时间，如果没有剪枝的话对于下方很多的后继我们仍需要把其一样探索，虽然最后结果根本通过剪枝算法是可以判断不可能取到的，就很浪费时间。

3. 对于权重矩阵的设置是需要慢慢调整的，并不是我们按照自己的思路就可以达到很优的效果。