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

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

17341078 李焕成

September 18, 2019

Contents

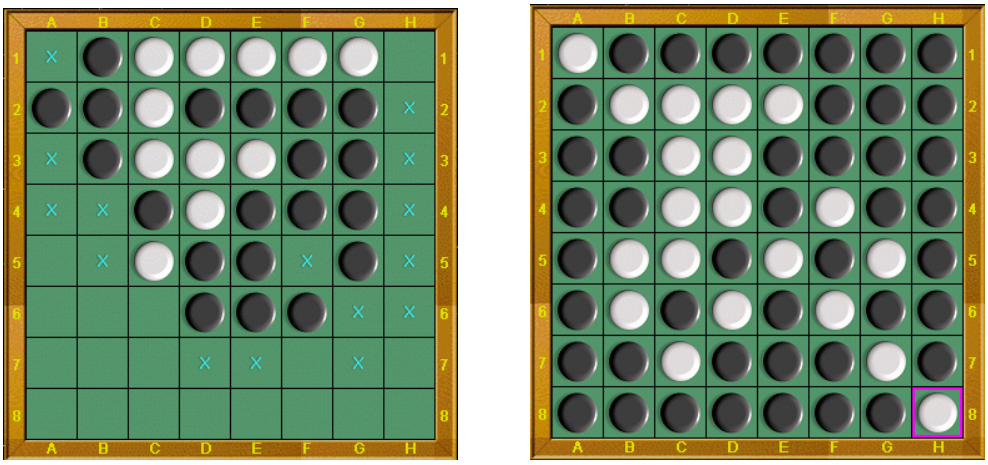

Figure 1: Othello Game

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://blog.sina.com.cn/s/blog_53ebdba00100cpy2.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_YourNumber.pdf, and send it to ai_201901@foxmail.com.

3 Codes

```
//最大最小博弈与α β剪枝—
Do * Find(Othello *board, enum Option player, int step, int min, int max, Do *choi
step: 极大极小树的深度，从大往小递减  */
{
int i, j, k, num;
Do *allChoices;
choice—>score = —MAX;                          //评估值设为最小值
choice—>pos.first = —1;
choice—>pos.second = —1;

num = board—>Rule(board, player);  /* 找出可以落子的数量，对应于图像界面里面的‘
player’的个数+  */
if (num == 0)    /* 无处落子 */
{
if (board—>Rule(board, (enum Option) — player))    /* 对方可以落子让对方下,.*/
{
Othello tempBoard;
Do nextChoice;
Do *pNextChoice = &nextChoice;
board—>Copy(&tempBoard, board);                    //复制下一步的棋盘
pNextChoice = Find(&tempBoard, (enum Option) — player, step — 1, —max, —min, pNext
//寻找下一步走法
choice—>score = —pNextChoice—>score;          //计算得分
choice—>pos.first = —1;
```

```
choice->pos.second = -1;
return choice;
}
else    /* 对方也无处落子游戏结束,. */
{
int value = WHITE*(board->whiteNum) + BLACK*(board->blackNum);    //计算
if (player*value>0)
{
choice->score = MAX - 1;
}
else if (player*value<0)
{
choice->score = -MAX + 1;
}
else
{
choice->score = 0;
}
return choice;
}
}
if (step <= 0)    /* 已经考虑到步step直接返回得分, */
{
choice->score = board->Judge(board, player);
return choice;
}

allChoices = (Do *)malloc(sizeof(Do)*num);  /* 新建一个do类型的数组, 其中*即为玩家可落子

/*下面三个两重循环其实就是分区域寻找可落子的位置, 第行代码
for67 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
```

```
for数字(表示寻找的顺序
)
1 1 1 1 1 1
1 3 3 3 3 1
1 3 2 2 3 1
1 3 2 2 3 1
1 3 3 3 3 1
1 1 1 1 1 1
*/
k = 0;
for (i = 0; i<6; i++)    /* 在最外圈寻找可落子位置 */
{
for (j = 0; j<6; j++)
{
if (i == 0 || i == 5 || j == 0 || j == 5)
{
/* 可落子的位置需要满足两个条件：、该位置上没有棋子1, 、如果把棋子放在这个位置上可以吃掉对方的2棋
()记录的是可以吃掉对方棋子的数量，所以stablestable符合条件>02
*/
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 (i = 0; i<6; i++)  // 分析同上
{
for (j = 0; j<6; j++)
```

```
{
if ((i == 2 || i == 3 || j == 2 || j == 3) && (i >= 2 && i <= 3 && j >= 2 && j <=
{
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 (i = 0; i<6; i++)  // 分析同上
{
for (j = 0; j<6; j++)
{
if ((i == 1 || i == 4 || j == 1 || j == 4) && (i >= 1 && i <= 4 && j >= 1 && j <=
{
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++)   /* 尝试在之前得到的个可落子位置进行落子num */
{
```

```
Othello tempBoard;
Do thisChoice, nextChoice;
Do *pNextChoice = &nextChoice;
thisChoice = allChoices[k];
board->Copy(&tempBoard, board);  // 为了不影响当前棋盘，需要复制一份作为虚拟棋盘
board->Action(&tempBoard, &thisChoice, player);  // 在虚拟棋盘上落子
pNextChoice = Find(&tempBoard, (enum Option) - player, step - 1, -max, -min, pNext
thisChoice.score = -pNextChoice->score;


/* 下面的条件和α if β剪枝有关，这里不解释，你们自己把注释写上去-hh */
/* max, min, 对应最大最小即α β值。-通过对比，赋值达到剪枝的目的α β之间就是可以更新的范围

-*/
if (thisChoice.score>min && thisChoice.score<max)     /* 可以预计的更优值 */
{
min = thisChoice.score;
//重置坐标以及得分
choice->score = thisChoice.score;
choice->pos.first = thisChoice.pos.first;
choice->pos.second = thisChoice.pos.second;
}
else if (thisChoice.score >= max)     /* 好的超乎预计 */
{
choice->score = thisChoice.score;                                    //得分比较高
choice->pos.first = thisChoice.pos.first;      //重置坐标以及得分
choice->pos.second = thisChoice.pos.second;
break;
}
/* 不如已知最优值 */

}
free(allChoices);
```

```
return choice;
}

int Othello::Judge(Othello *board, enum Option player)
{
        int value = 0;
        int i, j;
        Stable(board);

        // 对稳定子给予奖励
        for (i = 0; i<6; i++)
        {
                for (j = 0; j<6; j++)
                {
                        value += (board->cell[i][j].color)*(board->cell[i][j].stab
                }
        }

        // 对四个角给予额外奖励
        value += 64 * board->cell[0][0].color;
        value += 64 * board->cell[0][5].color;
        value += 64 * board->cell[5][0].color;
        value += 64 * board->cell[5][5].color;

        // 对X-给予惩罚square星位( 、、和的位置b2b7g2g7)
        value -= 32 * board->cell[1][1].color;
        value -= 32 * board->cell[1][4].color;
        value -= 32 * board->cell[4][1].color;
        value -= 32 * board->cell[4][4].color;

        //对行动力给予惩罚
        value -= board->Rule(board,player) * player;
```

```
        return value*player;
}
```

4 Results