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

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

17341137 Zhenpeng Song

2019 年 9 月 17 日

目录

1 Othello **2**

2 Tasks **2**

3 Codes **3**

4 Results **8**

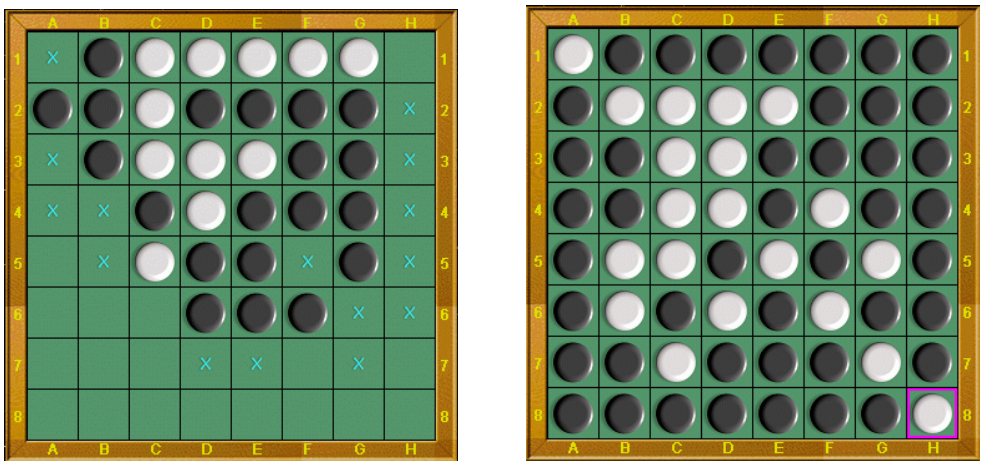

图 1: Othello Game

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://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 - 剪枝
Do * Find(Othello *board, enum Option player, int step, int min, int max, Do *choice)
/*  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;
        choice->pos.first = -1;
        choice->pos.second = -1;

        num = board->Rule(board, player);
        /*  找出player可以落子的数量, 对应于图像界面里面的‘+’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);
                        pNextChoice = Find(&tempBoard, (enum Option) - player, \\
                                            step - 1, -max, -min, pNextChoice);
                        choice->score = -pNextChoice->score;
                        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);
                        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step步,直接返回得分 */
        {
                choice->score = board->Judge(board, player);
                return choice;
        }

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


        /*
                下面三个两重for循环其实就是分区域寻找可落子的位置, 第67行代码 num = board->Rule(board, player)只
                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
                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: 1、该位置上没有棋子, 2、如果把棋子放在这个位置上
                                   棋子(可以夹住对方的棋子)。stable记录的是可以吃掉对方棋子的数量, 所以stable>0很
                                */
                                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 <= 3))
                        {
                            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 <= 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++)   /* 尝试在之前得到的num个可落子位置进行落子 */
                {
                    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, pNextChoice); // 递
thisChoice.score = -pNextChoice->score;


/* 下面的if条件和 - 剪枝有关，这里不解释，你们自己把注释写上去hh */

/* 使用Negamax算法代替minmax算法，实现 - 剪枝*/
// 其中，max 取上一层min的相反数，min取当前选择的score。
// 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小的；
// 因此，实际上只需要将每一层的max min调换并取反即可；
// 故假设根节点为第0层，beta层的数值为负。
// 剪枝条件：beta <= alpha，即score >= max。

if (player == WHITE) {
        int alpha = -max, beta = -min;
        if (thisChoice.score > -beta) {
                beta = -thisChoice.score;
                choice->score = thisChoice.score;
                choice->pos.first = thisChoice.pos.first;
                choice->pos.second = thisChoice.pos.second;
                min = -beta;
                if (beta <= alpha) break;
        }
}
else if(player == BLACK) {
        int alpha = min, beta = max;
        if (thisChoice.score > alpha) {
                alpha = thisChoice.score;
                choice->score = thisChoice.score;
                choice->pos.first = thisChoice.pos.first;
                choice->pos.second = thisChoice.pos.second;
                min = alpha;
                if (beta <= alpha) break;
        }
}

// 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;
// }
// else if (thisChoice.score >= max)     /* 好的超乎预计 */
// {
//      choice->score = thisChoice.score;
```

```
        //        choice->pos.first = thisChoice.pos.first;
        //        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].stable);
        }
    }

     int V[6][6] = {{ 20,  -8,  11,  11,  -8,  20},
                    { -8, -15,  -4,  -4, -15,  -8},
                    { 11,  -4,   2,   2,  -4,  11},
                    { 11,  -4,   2,   2,  -4,  11},
                    { -8, -15,  -4,  -4, -15,  -8},
                    { 20,  -8,  11,  11,  -8,  20}};

     for (int i = 0; i < 6; ++i)
     {
        for (int j = 0; j < 6; ++j)
        {
            value += V[i][j] * board->cell[i][j].color;
        }
     }

    // 行动力计算
    int my_mov, opp_mov, mov = 0;
    my_mov = Rule(board, player);
    opp_mov = Rule(board, (enum Option) - player);
    if(my_mov > opp_mov)
```

```
            value += 78.922 * (100.0 * my_mov)/(my_mov + opp_mov);
        else if(my_mov < opp_mov)
            value += 78.922 * -(100.0 * opp_mov)/(my_mov + opp_mov);

        return value*player;
}
```

4 Results

Actually, I didn't win the Tothello...

Instead, I tried to modified the judge function to lose with less gap between my proj. and Tothello...

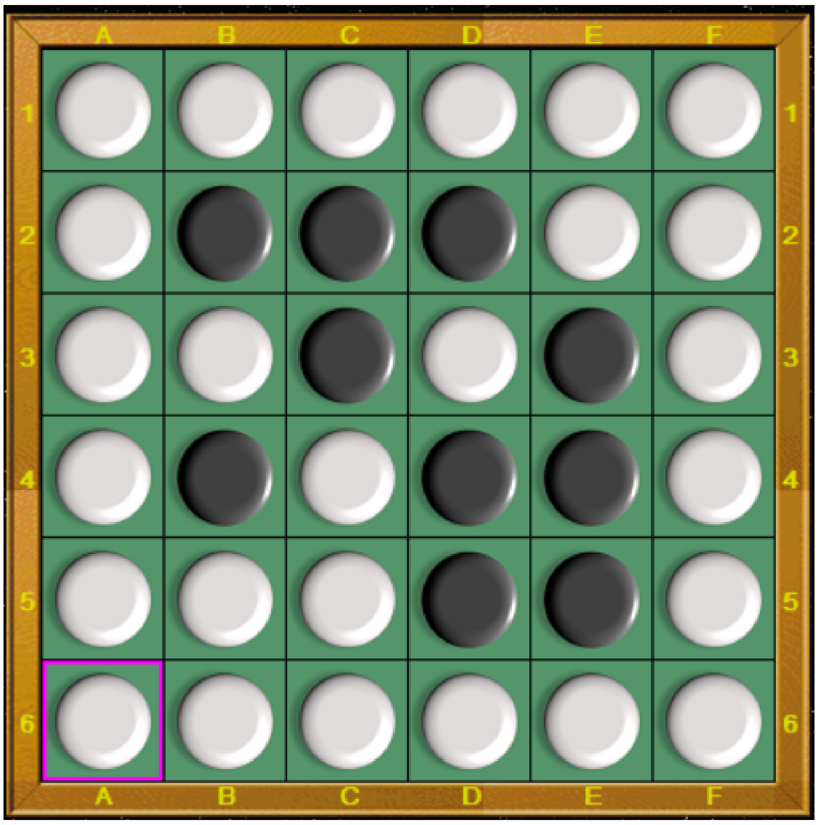

图 2: Result