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

E03 Othello Game (min-max, $\alpha - \beta$ pruning and evaluation function)

17341111 Xuehai Liu

September 18, 2019

Contents

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.

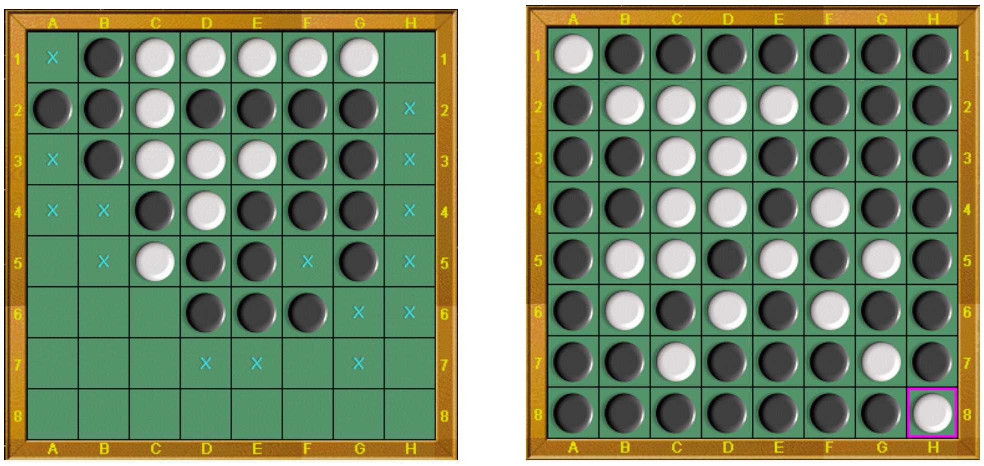


Figure 1: Othello Game

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 E06_YourNumber.pdf, and send it to ai_201901@formail.com

3 Codes

由于如果将代码全部贴上来显得有些过长，此处只给出我实现的部分。

首先是启发式函数的部分

```
 #define dim 6
bool canmove(char self, char opp, char *str)  {
        if (str[0] != opp) return false;
        for (int ctr = 1; ctr < dim; ctr++) {
                if (str[ctr] == '-') return false;
                if (str[ctr] == self) return true;
        }
        return false;
}

bool isLegalMove(char self, char opp, char grid[dim][dim], int startx, int starty)
{
        if (grid[startx][starty] != '-') return false;
        char str[10];
        int x, y, dx, dy, ctr;
        for (dy = -1; dy <= 1; dy++)
                for (dx = -1; dx <= 1; dx++)    {
        // keep going if both velocities are zero
                        if (!dy && !dx) continue;
                        str[0] = '\0';
                        for (ctr = 1; ctr < dim; ctr++)    {
                                x = startx + ctr*dx;
                        y = starty + ctr*dy;
                        if (x >= 0 && y >= 0 && x<dim && y<dim) str[ctr-1]
                        else str[ctr-1] = 0;
                    }
                    if (canmove(self, opp, str)) return true;
            }
        return false;
}

int num_valid_moves(char self, char opp, char grid[dim][dim])   {
        int count = 0, i, j;
        for(i=0; i<dim; i++)
                for(j=0; j<dim; j++)
                        if(isLegalMove(self, opp, grid, i, j)) count++;
        return count;
}

/*
 * Assuming my_color stores your color and opp_color stores opponent's color
 * '-' indicates an empty square on the board
 * 'b' indicates a black tile and 'w' indicates a white tile on the board
 */
double dynamic_heuristic_evaluation_function(char grid[dim][dim], enum Option play
{
        int my_tiles = 0, opp_tiles = 0, i, j, k, my_front_tiles = 0, opp_front_ti
        double p = 0, c = 0, l = 0, m = 0, f = 0, d = 0;

        int X1[] = {-1, -1, 0, 1, 1, 1, 0, -1};
        int Y1[] = {0, 1, 1, 1, 0, -1, -1, -1};
        int V[dim][dim] = {{20, -3, 11, 11, -3, 20},{-3, -7, -4, -4, -7, -3},{11,
        {11, -4, 2,  2, -4, 11},{-3, -7, -4, -4, -7, -3},{20, -3, 11, 11, -3, 20}}
;
```

```
    char my_color, opp_color;
        if(player == WHITE)
    {
        my_color = 'w';
        opp_color = 'b';
    }
    if(player == BLACK)
    {
        my_color = 'b';
        opp_color = 'w';
    }
        /*
        V[0] = {20, -3, 11, 8, 8, 11, -3, 20};
        V[1] = {-3, -7, -4, 1, 1, -4, -7, -3};
        V[2] = {11, -4, 2, 2, 2, 2, -4, 11};
        V[3] = {8, 1, 2, -3, -3, 2, 1, 8};
        V[4] = {8, 1, 2, -3, -3, 2, 1, 8};
        V[5] = {11, -4, 2, 2, 2, 2, -4, 11};
        V[6] = {-3, -7, -4, 1, 1, -4, -7, -3};
        V[7] = {20, -3, 11, 8, 8, 11, -3, 20};
        V[0] = {20, -3, 11, 11, -3, 20};
        V[1] = {-3, -7, -4, -4, -7, -3};
        V[2] = {11, -4, 2, 2, -4, 11};
        V[3] = {11, -4, 2,  2, -4, 11};
        V[4] = {-3, -7, -4, -4, -7, -3};
        V[5] = {20, -3, 11, 11, -3, 20};
        */
// Piece difference, frontier disks and disk squares
        for(i=0; i<dim; i++)
                for(j=0; j<dim; j++)  {
                        if(grid[i][j] == my_color)  {
                                d += V[i][j];
                                my_tiles++;
```

```
                    } else if(grid[i][j] == opp_color)  {
                            d -= V[i][j];
                            opp_tiles++;
                    }
                    if(grid[i][j] != '-')   {
                            for(k=0; k<dim; k++)  {
                                    x = i + X1[k]; y = j + Y1[k];
                                    if(x >= 0 && x < dim && y >= 0 && y < dim
                                            if(grid[i][j] == my_color)
my_front_tiles++;
                                            else opp_front_tiles++;
                                            break;
                                    }
                            }
                    }
            }
        if(my_tiles > opp_tiles)
                p = (100.0 * my_tiles)/(my_tiles + opp_tiles);
        else if(my_tiles < opp_tiles)
                p = -(100.0 * opp_tiles)/(my_tiles + opp_tiles);
        else p = 0;

        if(my_front_tiles > opp_front_tiles)
                f = -(100.0 * my_front_tiles)/(my_front_tiles + opp_front_tiles);
        else if(my_front_tiles < opp_front_tiles)
                f = (100.0 * opp_front_tiles)/(my_front_tiles + opp_front_tiles);
        else f = 0;

// Corner occupancy
        my_tiles = opp_tiles = 0;
        if(grid[0][0] == my_color) my_tiles++;
        else if(grid[0][0] == opp_color) opp_tiles++;
        if(grid[0][dim-1] == my_color) my_tiles++;
        else if(grid[0][dim-1] == opp_color) opp_tiles++;
        if(grid[dim-1][0] == my_color) my_tiles++;
        else if(grid[dim-1][0] == opp_color) opp_tiles++;
        if(grid[dim-1][dim-1] == my_color) my_tiles++;
        else if(grid[dim-1][dim-1] == opp_color) opp_tiles++;
        c = 25 * (my_tiles - opp_tiles);

// Corner closeness
        my_tiles = opp_tiles = 0;
        if(grid[0][0] == '-')   {
                if(grid[0][1] == my_color) my_tiles++;
                else if(grid[0][1] == opp_color) opp_tiles++;
                if(grid[1][1] == my_color) my_tiles++;
                else if(grid[1][1] == opp_color) opp_tiles++;
                if(grid[1][0] == my_color) my_tiles++;
                else if(grid[1][0] == opp_color) opp_tiles++;
        }
        if(grid[0][dim-1] == '-')   {
                if(grid[0][dim-2] == my_color) my_tiles++;
                else if(grid[0][dim-2] == opp_color) opp_tiles++;
                if(grid[1][dim-2] == my_color) my_tiles++;
                else if(grid[1][dim-2] == opp_color) opp_tiles++;
                if(grid[1][dim-1] == my_color) my_tiles++;
                else if(grid[1][dim-1] == opp_color) opp_tiles++;
        }
        if(grid[dim-1][0] == '-')   {
                if(grid[dim-1][1] == my_color) my_tiles++;
                else if(grid[dim-1][1] == opp_color) opp_tiles++;
                if(grid[dim-2][1] == my_color) my_tiles++;
                else if(grid[dim-2][1] == opp_color) opp_tiles++;
                if(grid[dim-2][0] == my_color) my_tiles++;
                else if(grid[dim-2][0] == opp_color) opp_tiles++;
        }
        if(grid[dim-1][dim-1] == '-')    {
                if(grid[dim-2][dim-1] == my_color) my_tiles++;
                else if(grid[dim-2][dim-1] == opp_color) opp_tiles++;
                if(grid[dim-2][dim-2] == my_color) my_tiles++;
                else if(grid[dim-2][dim-2] == opp_color) opp_tiles++;
                if(grid[dim-1][dim-2] == my_color) my_tiles++;
                else if(grid[dim-1][dim-2] == opp_color) opp_tiles++;
        }
        l = -12.5 * (my_tiles - opp_tiles);

// Mobility
        my_tiles = num_valid_moves(my_color, opp_color, grid);
        opp_tiles = num_valid_moves(opp_color, my_color, grid);
        if(my_tiles > opp_tiles)
                m = (100.0 * my_tiles)/(my_tiles + opp_tiles);
        else if(my_tiles < opp_tiles)
                m = -(100.0 * opp_tiles)/(my_tiles + opp_tiles);
        else m = 0;

// final weighted score
        double score = (10 * p) + (801.724 * c) + (382.026 * l) + (78.922 * m) + (
        return score;
}
```

然后是自己实现的Othello::myJudge.

```
    int Othello::myJudge(Othello *board, enum Option player)
{
    char grid[6][6];
    for (int i=0;i<6;i++)
        for (int j=0;j<6;j++)
    {
        if(board->cell[i][j].color == BLACK)
            grid[i][j] = 'b';
        if(board->cell[i][j].color == WHITE)
```

```
                grid[i][j] = 'w';
            if(board->cell[i][j].color == SPACE)
                grid[i][j] = '-';
        }
        return (int)dynamic_heuristic_evaluation_function(grid,player);
}
```

最后，根据上述启发式函数，构筑用于最大最小博弈与alpha-beta剪枝的Find函数：

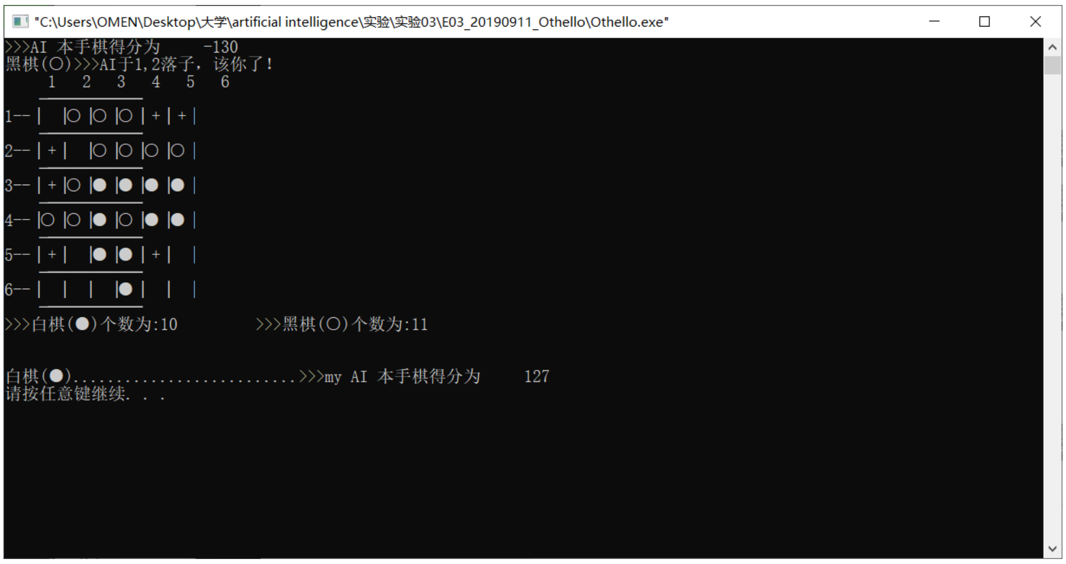
```
    //最大最小博弈与α-β剪枝

Do * myFind(Othello *board, enum Option player, int step, int min, int max, Do *choice)
{
        int i, j, k, num;
        Do *allChoices;
        choice->score = -MAX;
        choice->pos.first = -1;
        choice->pos.second = -1;

        num = board->Rule(board, player);
        // 首先获取可以下的位置的数量
        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, pNex
                        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->myJudge(board, player);
    return choice;
}

allChoices = (Do *)malloc(sizeof(Do)*num);
k = 0;

//第一步获取各个可行的决策（从外圈往内圈）
//将决策的值初始化为-无穷保证公平。
for (i = 0; i<6; i++)
{
    for (j = 0; j<6; j++)
    {
        if (i == 0 || i == 5 || j == 0 || j == 5)
        {
          //如果此位置没有被下且可以吃掉对方的子
            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++)
{
    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++)
    {
        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++;
                }
            }
        }
    }

//alpha beta 剪枝与 深搜策略

//对于每一个可以选择的策略进行分析
    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);
        //按照此决策行动后使用最大最小博弈算法找出下一个可能的决策
        //其中，此处交换了max和min的负值，作为另一方玩家的决策依据。
        pNextChoice = Find(&tempBoard, (enum Option) - player, step - 1, -max, -min, pNextChoice);
    //从而这次决策的分数就是下一次对方决策的分数的相反数
        thisChoice.score = -pNextChoice->score;

        //接下来作α-β剪枝。

        //由于黑白棋是零和游戏，我们可以将双方的操作进行合并。

        //这个if进行更优值的比较。对于极大节点，他选择评价值最大的子节点;
```

```
                //对于极小节点，他选择评价值最小的子节点
                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;
                }

                //此处进行alpha-beta剪枝，判定beta值小于等于祖先的alpha值，则不再对此节点进行拓展，直接返回beta。
                else if (thisChoice.score >= max)     /* 好的超乎预计 */
                {
                    //如果这个孩子节点的决策值优的爆表了，就将此节点的决策直接判定为此孩子节点。
                        choice->score = thisChoice.score;
                        choice->pos.first = thisChoice.pos.first;
                        choice->pos.second = thisChoice.pos.second;
                        break;
                }
                /* 不如已知最优值 */
        //不如已知最优值的情况下暂不做处理。仍需遍历完这个节点的所有子节点，挑选出最优解。
        }
        free(allChoices);
        return choice;
}
```

4 Results

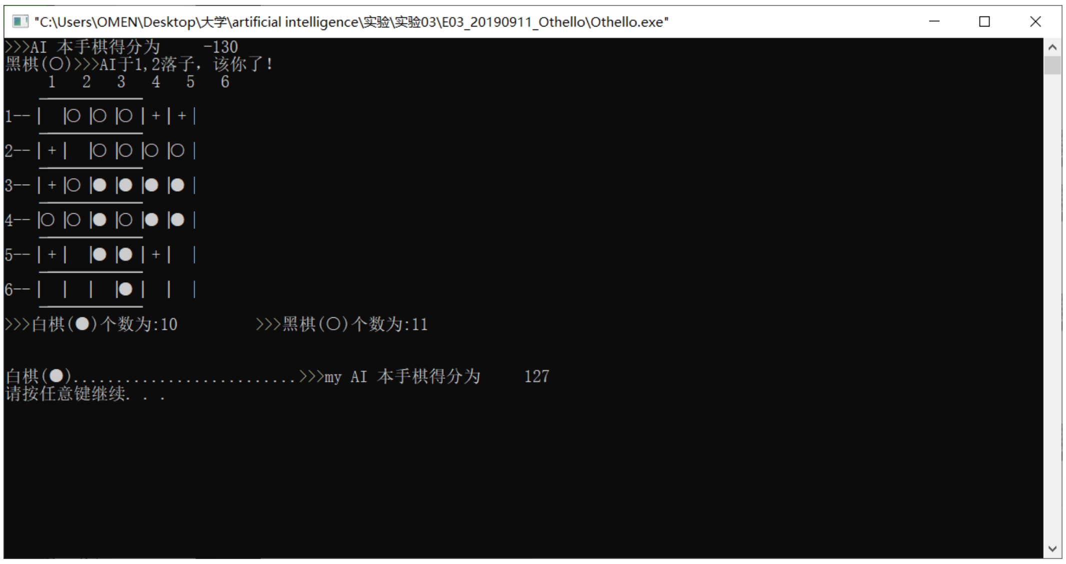
我令我写的AI与本来的AI对弈，下面截取了一些结果。（我是白棋）



"C:\Users\OMEN\Desktop\大学\artificial intelligence\实验\实验03\E03_20190911_Othello\Othello.exe"

```
>>>AI 本手棋得分为     -130
黑棋(○)>>>AI于1,2落子，该你了！
     1   2   3   4   5   6

1-- |   |○ |○ |○ | + | + |

2-- | + |   |○ |○ |○ |○ |

3-- | + |○ |● |● |● |● |

4-- |○ |○ |● |○ |● |● |

5-- | + |   |● |● | + |   |

6-- |   |   |   |● |   |   |

>>>白棋(●)个数为:10          >>>黑棋(○)个数为:11


白棋(●)...........................>>>my AI 本手棋得分为     127
请按任意键继续. . .
```

Figure 2: My AI vs computer - 1



```
"C:\Users\OMEN\Desktop\大学\artificial intelligence\实验\实验03\E03_20190911_Othello\Othello.exe"
>>>AI 本手棋得分为     -130
黑棋(○)>>>AI于1,2落子, 该你了!
     1   2   3   4   5   6
    ────────────
1-- |   |○ |○ |○ | + | + |
    ────────────
2-- | + |   |○ |○ |○ |○ |
    ────────────
3-- | + |○ |● |● |● |● |
    ────────────
4-- |○ |○ |● |○ |● |● |
    ────────────
5-- | + |   |● |● | + |   |
    ────────────
6-- |   |   |   |● |   |   |
    ────────────
>>>白棋(●)个数为:10         >>>黑棋(○)个数为:11


白棋(●)..........................>>>my AI 本手棋得分为     127
请按任意键继续. . .
```

Figure 3: My AI vs computer - 2