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

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

17341175 zhicheng xu

September 16, 2019

Contents

1 Othello **2**

2 Tasks **2**

3 Codes **3**

4 Results **22**

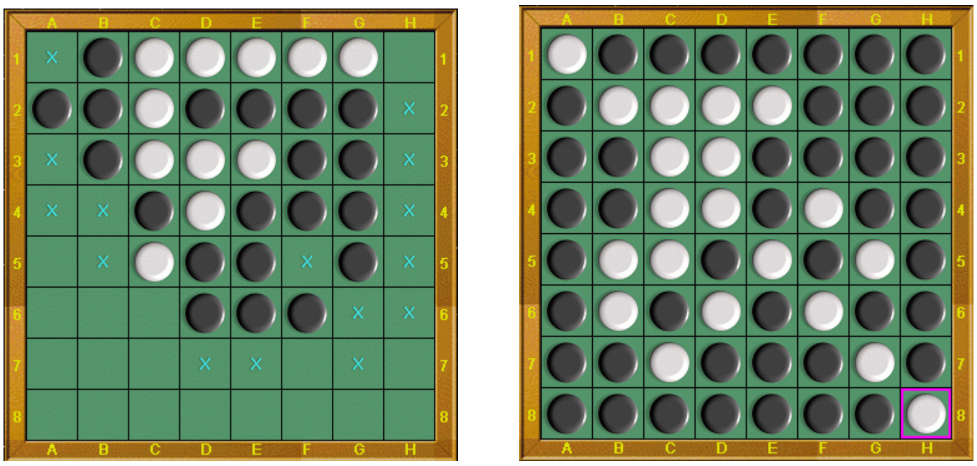


Figure 1: Othello Game

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

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

using namespace std;

int const MAX = 65534;

int  deepth = 10;          //

//

enum Option
{
        WHITE = -1, SPACE, BLACK        //                //
};

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

struct WinNum
{       enum Option color;
        int stable;                          //
};
```

```
//
struct Othello
{

        WinNum cell[6][6];
        int whiteNum;
        int blackNum;



        void Create(Othello *board);
        void Copy(Othello *boardDest, const  Othello *boardSource);                    //
        void Show(Othello *board);
        int Rule(Othello *board, enum Option player);
        int Action(Othello *board, Do *choice, enum Option player);                    //
        void Stable(Othello *board);
        int Judge(Othello *board, enum Option player);
};//
```

```
//                          -
Do * Find(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); //
        if (num == 0)      /*                */
        {
                if (board->Rule(board, (enum Option) - player))
/*                              ,                   .*/
                {
                        Othello tempBoard;
                        Do nextChoice;
                        Do *pNextChoice = &nextChoice;
                        board->Copy(&tempBoard, board);
                        pNextChoice = Find(&tempBoard, (enum Option) - player, step - 1, -
//
                        choice->score = -pNextChoice->score;
                        choice->pos.first = -1;
                        choice->pos.second = -1;
                        return choice;
                }
                else    /*                              ,                   . */
                {
                        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)     /*                  step     ,                           */
        {
                choice->score = board->Judge(board, player);
//
                return choice;
```

```
    }
//
    allChoices = (Do *)malloc(sizeof(Do)*num);   //
    k = 0;
    for (i = 0; i<6; i++)
    {
        for (j = 0; j<6; j++)  //
        {
            if (i == 0 || i == 5 || j == 0 || j == 5)
            {
                if (board->cell[i][j].color == SPACE && board->cell[i][j].
                {
                    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 &&
            {
                if (board->cell[i][j].color == SPACE && board->cell[i][j].
                {
                    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 &&
                        {
                                if (board->cell[i][j].color == SPACE && board->cell[i][j].
                                {
                                        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 = Find(&tempBoard, (enum Option) - player, step - 1, -max, -mi
                thisChoice.score = -pNextChoice->score;

                if (thisChoice.score>min && thisChoice.score<max)
/*                             */
                {
                        min = thisChoice.score;
                        choice->score = thisChoice.score;
                        choice->pos.first = thisChoice.pos.first;
                        choice->pos.second = thisChoice.pos.second;
                }
```

```
            else if (thisChoice.score >= max)      //         >MAX,
            {
                    choice->score = thisChoice.score;
                    choice->pos.first = thisChoice.pos.first;
                    choice->pos.second = thisChoice.pos.second;
                    break;
            }
            /*                                  */
    }
    free(allChoices);
    return choice;
}

int main()
{
    Othello board;
    Othello *pBoard = &board;
    enum Option player , present ;
    Do choice;
    Do *pChoice = &choice;
    int  num , result = 0;
    char restart = '␣';

start:
    player = SPACE;
    present = BLACK;
    num = 4;
    restart = '␣';

    cout << ">>>                              ␣\n";




            while (player != WHITE && player != BLACK)
            {
                    cout << ">>>                          (    ),                (    )           1
```

```
                scanf("%d", &player);
                cout << ">>>                    :␣␣\n";



                if (player != WHITE && player != BLACK)
                {
                        cout << "                                                        \n";
                }
        }

        board.Create(pBoard);

        while (num<36)
//              3 6
        {
                char *Player = "";
                if (present == BLACK)
                {
                        Player = "          (     )";
                }
                else if (present == WHITE)
                {
                        Player = "          (     )";
                }

                if (board.Rule(pBoard, present) == 0)
                {
                        if (board.Rule(pBoard, (enum Option) - present) == 0)
                        {
                                break;
                        }

                        cout << Player << "GAME␣OVER!␣\n";
                }
                else
                {
                        int i, j;
```

```
board.Show(pBoard);

if (present == player)
{
        while (1)
        {
                cout << Player << "␣\n␣>>>

                cin >> i>> j;
                i--;
                j--;
                pChoice->pos.first = i;
                pChoice->pos.second = j;

                if (i<0 || i>5 || j<0 || j>5 || pBoard->c
                {
                        cout <<">>>
                        board.Show(pBoard);
                }
                else
                {
                        break;
                }
        }
        system("cls");
        cout << ">>>        ␣                        ␣␣␣␣␣" << pC
        system("pause");
        cout << ">>>                             " << pChoice->score
}
else    // A I
{
        cout << Player << "........................";

        pChoice = Find(pBoard, present, deepth, -MAX, MAX,
        i = pChoice->pos.first;
        j = pChoice->pos.second;
        system("cls");
```

```
                        cout << ">>>AI␣                          ␣␣␣␣␣" << pChoice
                }



                board.Action(pBoard, pChoice, present);
                num++;
                cout << Player << ">>> A I   " << i + 1 << "," << j + 1<<"
        }

        present = (enum Option) - present;     //
}



board.Show(pBoard);



result = pBoard->whiteNum - pBoard->blackNum;

if (result>0)
{
        cout << "\ n                               (     )                          \n";
}
else if (result<0)
{
        cout << "\ n                               (     )                          \n";
}
else
{
        cout << "\ n                                                                  \
}

cout << "\n␣                        G A M E ␣OVER!                         \n";
cout << "\n";

while (restart != 'Y' && restart != 'N')
{
        cout <<"|
                        cout <<"|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣\n";
                        cout <<"|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣\n";
                        cout <<"|>>>>>>>>>>>>>>>>Again?(Y,N)<<<<<<<<<<<<<<<<|\n";
                        cout <<"|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣\n";
                        cout <<"|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣\n";
                        cout <<"|
                        cout << "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣\n";
                        cout << "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣\n";
                        cout << "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣\n";
                        cout << "␣               ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣               ␣␣␣␣␣␣␣\n
                        cout << "␣|␣␣␣YES␣␣|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣␣␣NO␣␣␣|␣␣␣␣␣␣\n";
                        cout << "␣               ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣               ␣␣␣␣␣␣\n"

                        cin >> restart;
                        if (restart == 'Y')
                        {
                                goto start;
                        }
                }


        return 0;
}






void Othello::Create(Othello *board)
{
        int i, j;
        board->whiteNum = 2;
        board->blackNum = 2;
        for (i = 0; i<6; i++)
        {
                for (j = 0; j<6; j++)
```

```
            {
                board->cell[i][j].color = SPACE;
                board->cell[i][j].stable = 0;
            }
        }
        board->cell[2][2].color = board->cell[3][3].color = WHITE;
        board->cell[2][3].color = board->cell[3][2].color = BLACK;
}


void Othello::Copy(Othello *Fake, const  Othello *Source)
{
        int i, j;
        Fake->whiteNum = Source->whiteNum;
        Fake->blackNum = Source->blackNum;
        for (i = 0; i<6; i++)
        {
            for (j = 0; j<6; j++)
            {
                Fake->cell[i][j].color = Source->cell[i][j].color;
                Fake->cell[i][j].stable = Source->cell[i][j].stable;
            }
        }
}

void Othello::Show(Othello *board)
{
        int i, j;
        cout << "\n␣␣";
        for (i = 0; i<6; i++)
        {
            cout << "␣␣␣" << i + 1;
        }
        cout << "\n␣␣␣␣                                          \n";
        for (i = 0; i<6; i++)
        {
            cout << i + 1 << "—    ";
            for (j = 0; j<6; j++)
            {
                switch (board->cell[i][j].color)
                {
                case BLACK:
                    cout << "      ";
                    break;
                case WHITE:
                    cout << "      ";
                    break;
                case SPACE:
                    if (board->cell[i][j].stable)
                    {
                        cout << "␣+   ";
                    }
                    else
                    {
                        cout << "␣␣   ";
                    }
                    break;
                default:    /*                     */
                    cout << "*␣   ";
                }
            }
            cout << "\n␣␣␣␣                                  \n";
        }

        cout << ">>>      (    )         :" << board->whiteNum << "␣␣␣␣␣␣␣␣␣";
        cout << ">>>      (    )         :" << board->blackNum << endl << endl << endl;
}


int Othello::Rule(Othello *board, enum Option player)
{
        int i, j;
        unsigned num = 0;
        for (i = 0; i<6; i++)
```

```
{
    for (j = 0; j<6; j++)
    {
        if (board->cell[i][j].color == SPACE)
        {
            int x, y;
            board->cell[i][j].stable = 0;
            for (x = -1; x <= 1; x++)
            {
                for (y = -1; y <= 1; y++)
                {
                    if (x || y)    /* 8              */
                    {
                        int i2, j2;
                        unsigned num2 = 0;
                        for (i2 = i + x, j2 = j + y; i2 >=
                        {
                            if (board->cell[i2][j2].co
                            {
                                num2++;
                            }
                            else if (board->cell[i2][j
                            {
                                board->cell[i][j].
                                break;
                            }
                            else if (board->cell[i2][j
                            {
                                break;
                            }
                        }
                    }
                }
            }

            if (board->cell[i][j].stable)
            {
```

```
                                        num++;
                                }
                        }
                }
        }
        return num;
}



int Othello::Action(Othello *board, Do *choice, enum Option player)
{
        int i = choice->pos.first, j = choice->pos.second;
        int x, y;

        if (board->cell[i][j].color != SPACE || board->cell[i][j].stable == 0 || player =
        {
                return -1;
        }



        board->cell[i][j].color = player;
        board->cell[i][j].stable = 0;



        if (player == WHITE)
        {
                board->whiteNum++;
        }
        else if (player == BLACK)
        {
                board->blackNum++;
        }




        for (x = -1; x <= 1; x++)
        {
```

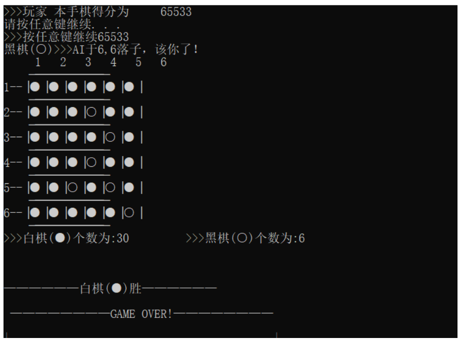
```
        for (y = -1; y <= 1; y++)
        {

            //                          8


            if (x || y)
            {
                int i2, j2;
                unsigned num = 0;
                for (i2 = i + x, j2 = j + y; i2 >= 0 && i2 <= 5 && j2 >= 0
                {
                    if (board->cell[i2][j2].color == (enum Option) - p
                    {
                        num++;
                    }
                    else if (board->cell[i2][j2].color == player)
                    {
                        board->whiteNum += (player*WHITE)*num;
                        board->blackNum += (player*BLACK)*num;

                        for (i2 -= x, j2 -= y; num>0; num--, i2 -=
                        {
                            board->cell[i2][j2].color = player
                            board->cell[i2][j2].stable = 0;
                        }
                        break;
                    }
                    else if (board->cell[i2][j2].color == SPACE)
                    {
                        break;
                    }
                }
            }
        }
    }
    return 0;
```

```
}


void Othello::Stable(Othello *board)
{
        int i, j;
        for (i = 0; i<6; i++)
        {
                for (j = 0; j<6; j++)
                {
                        if (board->cell[i][j].color != SPACE)
                        {
                                int x, y;
                                board->cell[i][j].stable = 1;

                                for (x = -1; x <= 1; x++)
                                {
                                        for (y = -1; y <= 1; y++)
                                        {
                                                /* 4              */
                                                if (x == 0 && y == 0)
                                                {
                                                        x = 2;
                                                        y = 2;
                                                }
                                                else
                                                {
                                                        int i2, j2, flag = 2;
                                                        for (i2 = i + x, j2 = j + y; i2 >=
                                                        {
                                                                if (board->cell[i2][j2].co
                                                                {
                                                                        flag--;
                                                                        break;
                                                                }
                                                        }
                                                        for (i2 = i - x, j2 = j - y; i2 >=
                                                        {
                                                                if (board->cell[i2][j2].co
                                                                {
                                                                        flag--;
                                                                        break;
                                                                }
                                                        }

                                                        if (flag)
/*                                  */
                                                        {
                                                                board->cell[i][j].stable+
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }
}


int Othello::Judge(Othello *board, enum Option player)  //
{
    WinNum grid[6][6];
    for(int i=0;i<6;i++){
        for(int j=0;j<6;j++){
                grid[i][j]=board->cell[i][j];
                }
        }
    enum Option my_color=player;
    enum Option opp_color=(enum Option)-player;
        int my_tiles = 0, opp_tiles = 0, i, j, k, my_front_tiles = 0, opp_front_tiles = 0
        double p = 0, c = 0, l = 0, f = 0, d = 0;

        int X1[] = {-1, -1, 0, 1, 0, -1};
        int Y1[] = {0, 1, 1, -1, -1, -1};
        int V[6][6]={{20, -3, 11, 11, -3, 20}  //
        ,{-3, -7, -4, -4, -7, -3}
        ,{11, -4, 2, 2, -4, 11}

        ,{11, -4, 2, 2, -4, 11}
        ,{-3, -7, -4, -4, -7, -3}
        ,{20, -3, 11, 11, -3, 20}};



// Piece difference        , frontier disks      and disk squares

        for(i=0; i<6; i++)
                for(j=0; j<6; j++)  {
                        if(grid[i][j].color == my_color)  {
                                d += V[i][j];
                                my_tiles++;
                        } else if(grid[i][j].color == opp_color)  {
                                d -= V[i][j];
                                opp_tiles++;
                        }
                        if(grid[i][j].color != 0)   {
                                for(k=0; k<6; k++)  {
                                        x = i + X1[k]; y = j + Y1[k];
                                        if(x >= 0 && x < 6 && y >= 0 && y < 6 && grid[x][y
                                                if(grid[i][j].color == 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].color == my_color) my_tiles++;
        else if(grid[0][0].color == opp_color) opp_tiles++;
        if(grid[0][5].color == my_color) my_tiles++;
        else if(grid[0][5].color == opp_color) opp_tiles++;
        if(grid[5][0].color == my_color) my_tiles++;
        else if(grid[5][0].color == opp_color) opp_tiles++;
        if(grid[5][5].color == my_color) my_tiles++;
        else if(grid[5][5].color == opp_color) opp_tiles++;
        c = 25 * (my_tiles - opp_tiles);

// Corner closeness
        my_tiles = opp_tiles = 0;
        if(grid[0][0].color == SPACE)   {
                if(grid[0][1].color == my_color) my_tiles++;
                else if(grid[0][1].color == opp_color) opp_tiles++;
                if(grid[1][1].color == my_color) my_tiles++;
                else if(grid[1][1].color == opp_color) opp_tiles++;
                if(grid[1][0].color == my_color) my_tiles++;
                else if(grid[1][0].color == opp_color) opp_tiles++;
        }
        if(grid[0][5].color == SPACE)   {
                if(grid[0][4].color == my_color) my_tiles++;
                else if(grid[0][4].color == opp_color) opp_tiles++;
                if(grid[1][4].color == my_color) my_tiles++;
                else if(grid[1][4].color == opp_color) opp_tiles++;
                if(grid[1][5].color == my_color) my_tiles++;
```

```
            else if(grid[1][5].color == opp_color) opp_tiles++;
        }
        if(grid[5][0].color == SPACE)   {
            if(grid[5][1].color == my_color) my_tiles++;
            else if(grid[5][1].color == opp_color) opp_tiles++;
            if(grid[4][1].color == my_color) my_tiles++;
            else if(grid[4][1].color == opp_color) opp_tiles++;
            if(grid[4][0].color == my_color) my_tiles++;
            else if(grid[4][0].color == opp_color) opp_tiles++;
        }
        if(grid[5][5].color == SPACE)   {
            if(grid[4][5].color == my_color) my_tiles++;
            else if(grid[4][5].color == opp_color) opp_tiles++;
            if(grid[4][4].color == my_color) my_tiles++;
            else if(grid[4][4].color == opp_color) opp_tiles++;
            if(grid[5][4].color == my_color) my_tiles++;
            else if(grid[5][4].color == opp_color) opp_tiles++;
        }
        l = -12.5 * (my_tiles - opp_tiles);



// final weighted score
        double score = (64 * p) + (64 * c) + (128* l) + (64 * f) + (32 * d);

        return score*player;

}
```

4 Results



```
>>>玩家 本手棋得分为     65533
请按任意键继续. . .
>>>按任意键继续65533
黑棋(○)>>>AI于6,6落子, 该你了!
     1   2   3   4   5   6
    ________________
1-- |● |● |● |● |● |● |
    ________________
2-- |● |● |● |○ |● |● |
    ________________
3-- |● |● |● |● |○ |● |
    ________________
4-- |● |● |● |○ |● |● |
    ________________
5-- |● |● |○ |● |○ |● |
    ________________
6-- |● |● |● |● |● |○ |
    ________________
>>>白棋(●)个数为:30         >>>黑棋(○)个数为:6


————————白棋(●)胜————————

 ————————————GAME OVER!————————————
```