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

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

17341137 Zhenpeng Song

September 17, 2019

Contents

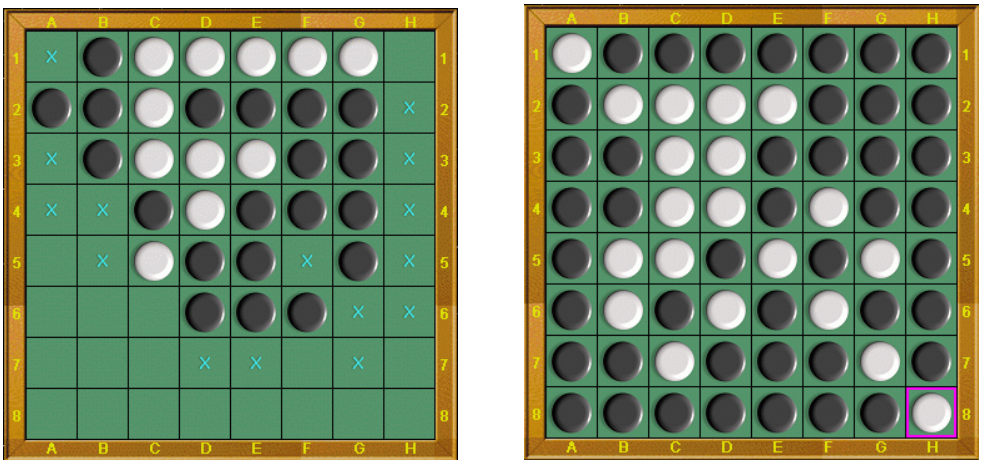

Figure 1: Othello Game

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

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

        num = board->Rule(board, player);  /*          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, ste
                        choice->score = -pNextChoice->score;
                        choice->pos.first = -1;
                        choice->pos.second = -1;
```

```
                return choice;
        }
        else    /*                          ,              . */
        {
                int value = WHITE*(board->whiteNum) + BLACK*(board->blackNu
                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)    /*              s t e p    ,                    */
{
        choice->score = board->Judge(board, player);
        return choice;
}

/*          d o *                        n u m
allChoices = (Do *)malloc(sizeof(Do)*num);


/*
                          f o r
```

```
                    f o r
                                )
        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++)    /*                                          */
{
        for (j = 0; j<6; j++)
        {
                if (i == 0 || i == 5 || j == 0 || j == 5)
                {
                        /*                                                  1
                                (                                )    s t a b l e
                        */
                        if (board->cell[i][j].color == SPACE && board->cel
                        {
                                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
                {
                        if (board->cell[i][j].color == SPACE && board->cel
                        {
                                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
                {
                        if (board->cell[i][j].color == SPACE && board->cel
                        {
                                allChoices[k].score = -MAX;
                                allChoices[k].pos.first = i;
                                allChoices[k].pos.second = j;
                                k++;
                        }
                }
        }
}

for (k = 0; k<num; k++)    /*                              n u m
{
```

```
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, -
thisChoice.score = -pNextChoice->score;


/*        if        -

/*     Negamax        minmax              -        */
//       max          min            min               s
//
//                                         max  min
//                         0       beta
//              beta <= alpha,   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)     /*
                // {
                //       min = thisChoice.score;
                //       choice->score = thisChoice.score;
                //       choice->pos.first = thisChoice.pos.first;
                //       choice->pos.second = thisChoice.pos.second;
                // }
                // else if (thisChoice.score >= max)     /*                    */
                // {
                //       choice->score = thisChoice.score;
                //       choice->pos.first = thisChoice.pos.first;
                //       choice->pos.second = thisChoice.pos.second;
                //       break;
                // }
                // /*                        */
        }
        free(allChoices);
        return choice;
}

...

int Othello::Judge(Othello *board, enum Option player)
{
```

```
int value = 0;
int i, j;
Stable(board);

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

 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;
        }
 }

//
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...

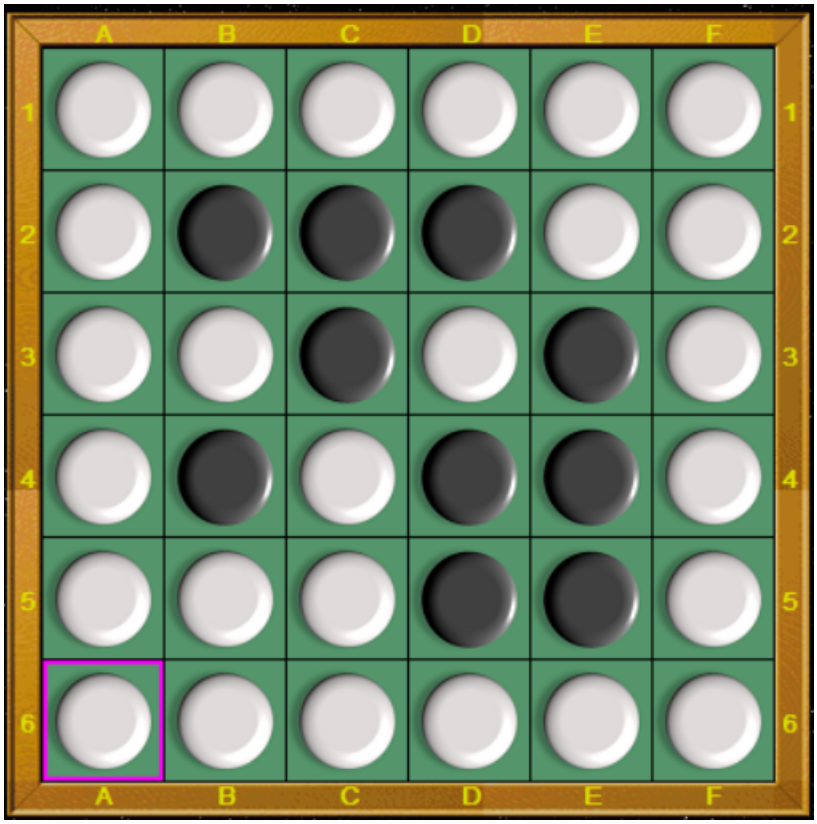

Figure 2: Result