Introduction to AI Group Game Project Report

Group ID: 42 Name: AED35

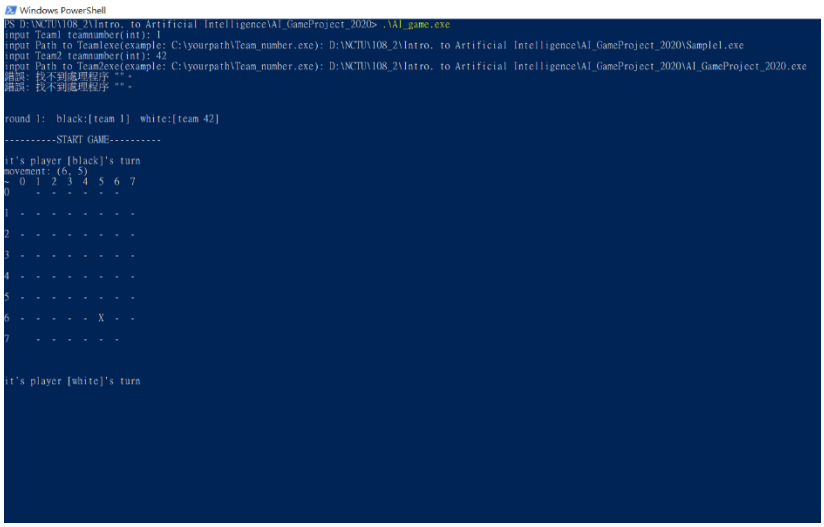
Member ID: 0716085 Name: 賴品樺

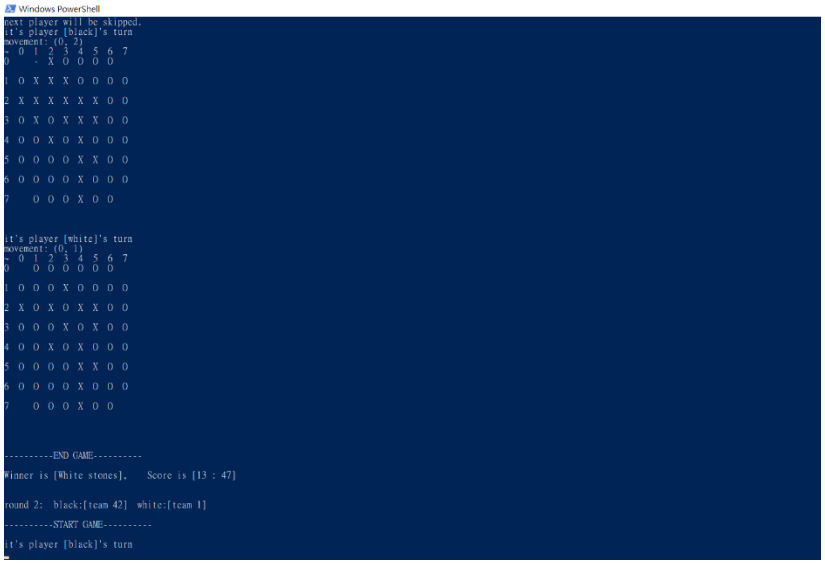
Description:

The program is based on minimax algorithm and α-β pruning. The program will expand possible steps and use minimax algorithm and α-β pruning mechanism to find the optimal step. To prevent time limit, at the beginning I set the depth of searching is 3. Later I change the depth becomes 2 if pieces on board is < 8, 3 if pieces on board is >= 8 and < 30, 4 if pieces on board is >= 30 and < 40, 5 if pieces on board is >= 40 and < 50, 6 if pieces on board is >= 50. This change can prevent time limit and can obtain a great result than the original.
The way how I determine the utility of step is according to the board state. As same as original Othello/Reversi game, I use the corner heuristic, which means the corner that the piece can't be reversed to opponent has the higher priority. And if there exist more than 1 optimal steps, I use a random function to choose one to increase some randomness in the program.

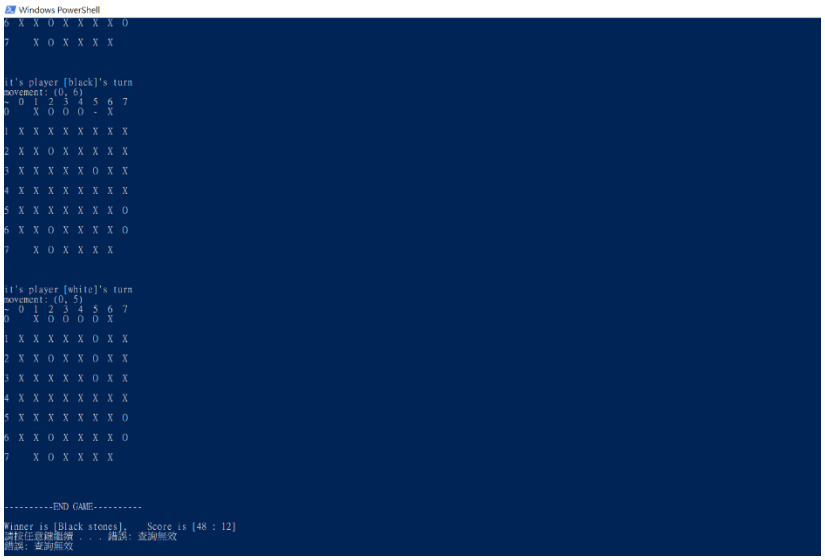
Result (depth = 3):

vs. Sample1.exe



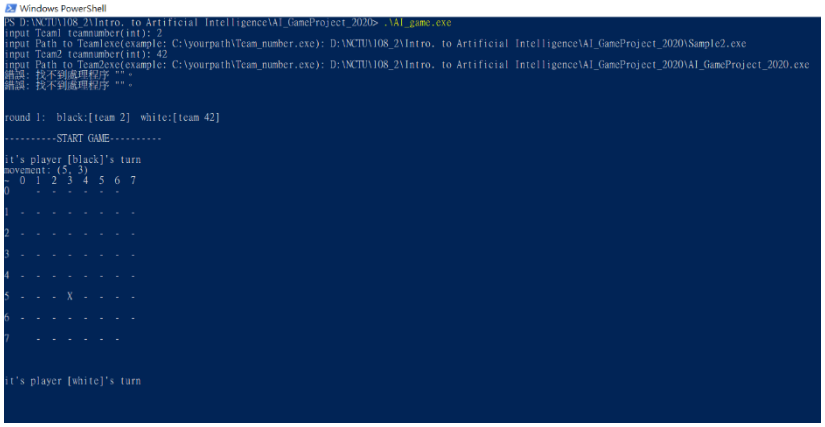


```
Windows PowerShell
next player will be skipped.
it's player [black]'s turn
movement: (0, 2)
-  0  1  2  3  4  5  6  7
0     -  X  O  O  O  O

1  O  X  X  X  O  O  O  O

2  X  X  X  X  X  X  O  O

3  O  X  O  X  X  X  O  O

4  O  O  X  O  X  O  O  O

5  O  O  O  O  X  X  O  O

6  O  O  O  O  X  O  O  O

7     O  O  O  X  O  O


it's player [white]'s turn
movement: (0, 1)
-  0  1  2  3  4  5  6  7
0     O  O  O  O  O  O

1  O  O  O  X  O  O  O  O

2  X  O  X  O  X  X  O  O

3  O  O  O  X  O  X  O  O

4  O  O  X  O  X  O  O  O

5  O  O  O  O  X  X  O  O

6  O  O  O  O  X  O  O  O

7     O  O  O  X  O  O



----------END GAME----------

Winner is [White stones],   Score is [13 : 47]

round 2:  black:[team 42]  white:[team 1]

----------START GAME----------

it's player [black]'s turn
```



```
Windows PowerShell
6  X  X  O  X  X  X  X  O

7     X  O  X  X  X  X


it's player [black]'s turn
movement: (0, 6)
-  0  1  2  3  4  5  6  7
0     X  O  O  O  -  X

1  X  X  X  X  X  X  X  X

2  X  X  O  X  X  X  X  X

3  X  X  X  X  X  O  X  X

4  X  X  X  X  X  X  X  X

5  X  X  X  X  X  X  X  O

6  X  X  O  X  X  X  X  O

7     X  O  X  X  X  X


it's player [white]'s turn
movement: (0, 5)
-  0  1  2  3  4  5  6  7
0     X  O  O  O  O  X

1  X  X  X  X  X  O  X  X

2  X  X  O  X  X  O  X  X

3  X  X  X  X  X  O  X  X

4  X  X  X  X  X  X  X  X

5  X  X  X  X  X  X  X  O

6  X  X  O  X  X  X  X  O

7     X  O  X  X  X  X


----------END GAME----------

Winner is [Black stones],   Score is [48 : 12]
請按任意鍵繼續 . . . 錯誤: 查詢無效
錯誤: 查詢無效
```

vs. Sample2.exe



```
Windows PowerShell
PS D:\NCTU\108_2\Intro. to Artificial Intelligence\AI_GameProject_2020> .\AI_game.exe
input Team1 teamnumber(int): 2
input Path to Team1exe(example: C:\yourpath\Team_number.exe): D:\NCTU\108_2\Intro. to Artificial Intelligence\AI_GameProject_2020\Sample2.exe
input Team2 teamnumber(int): 42
input Path to Team2exe(example: C:\yourpath\Team_number.exe): D:\NCTU\108_2\Intro. to Artificial Intelligence\AI_GameProject_2020\AI_GameProject_2020.exe
錯誤: 找不到處理程序 "".
錯誤: 找不到處理程序 "".


round 1:  black:[team 2]  white:[team 42]

----------START GAME----------

it's player [black]'s turn
movement: (5, 3)
-  0  1  2  3  4  5  6  7
0     -  -  -  -  -  -

1  -  -  -  -  -  -  -  -

2  -  -  -  -  -  -  -  -

3  -  -  -  -  -  -  -  -

4  -  -  -  -  -  -  -  -

5  -  -  -  X  -  -  -  -

6  -  -  -  -  -  -  -  -

7     -  -  -  -  -


it's player [white]'s turn
```

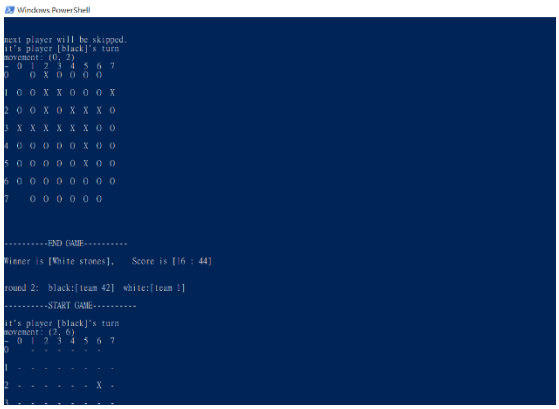
```
3  O  O  X  X  X  O  X  O

4  O  O  O  X  X  X  O  O

5  O  O  O  O  O  O  O  O

6  O  O  O  O  O  O  O  O

7     O  O  O  O  -  O


next player will be skipped.
it's player [black]'s turn
movement: (7, 5)
-  0  1  2  3  4  5  6  7
0     O  O  O  O  O  O

1  O  X  X  X  O  O  O  O

2  O  O  X  X  X  O  X  X

3  O  O  X  X  X  O  X  O

4  O  O  O  X  X  X  O  O

5  O  O  O  O  O  X  O  O

6  O  O  O  O  O  X  O  O

7     O  O  O  O  X  O


---------END GAME---------

Winner is [White stones],    Score is [18 : 42]

round 2:  black:[team 42]  white:[team 2]

---------START GAME---------

it's player [black]'s turn
```

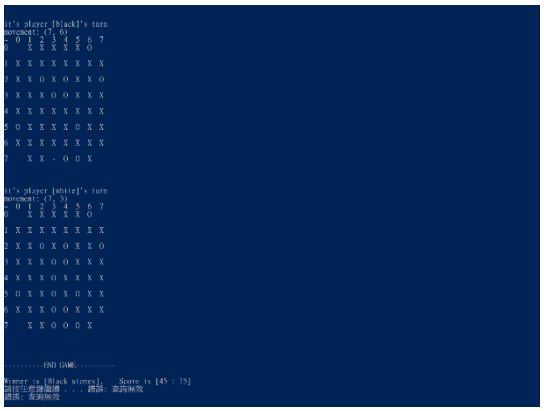
```
it's player [black]'s turn
movement: (4, 0)
-  0  1  2  3  4  5  6  7
0     O  X  X  X  X  X

1  X  X  O  X  O  X  X  X

2  X  X  O  X  X  X  X  X

3  X  X  O  O  O  X  X  X

4  X  X  O  O  O  X  X  X

5  X  X  O  X  O  X  X  X

6  -  X  O  O  X  X  X  X

7     X  O  X  X  X  X


it's player [white]'s turn
movement: (6, 0)
-  0  1  2  3  4  5  6  7
0     O  X  X  X  X  X

1  X  X  O  X  O  X  X  X

2  X  X  O  X  X  X  X  X

3  X  X  O  O  O  X  X  X

4  X  X  O  O  O  X  X  X

5  X  O  O  X  O  X  X  X

6  O  O  O  O  X  X  X  X

7     X  O  X  X  X  X


---------END GAME---------

Winner is [Black stones],    Score is [42 : 18]
請按任意鍵繼續 . . . 錯誤: 查詢無效
錯誤: 查詢無效
```

Result (non-constant depth):

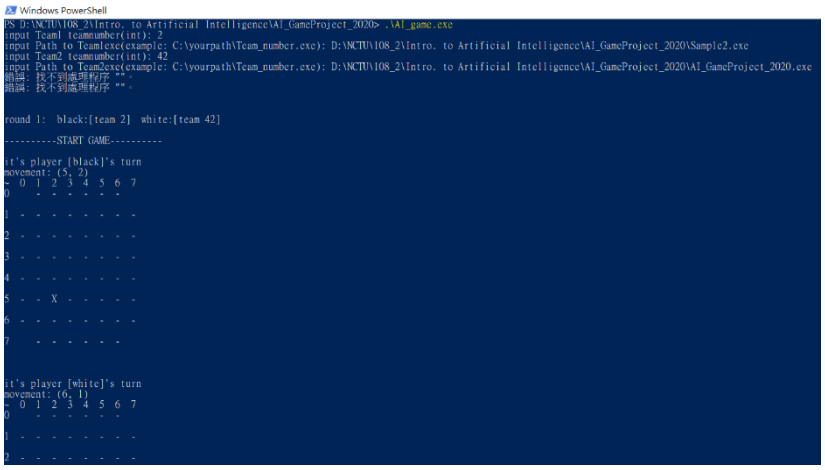
vs. Sample1.exe

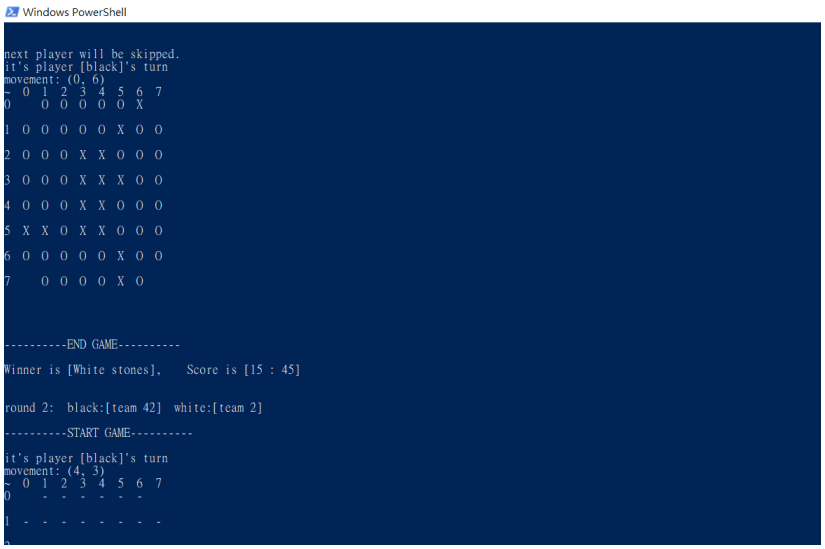
```
Windows PowerShell
PS D:\NCTU\108_2\Intro. to Artificial Intelligence\AI_GameProject_2020> .\AI_game.exe
input Team1 teamnumber(int): 1
input Path to Team1exe(example: C:\yourpath\Team_number.exe): D:\NCTU\108_2\Intro. to Artificial Intelligence\AI_GameProject_2020\Sample1.exe
input Team2 teamnumber(int): 42
input Path to Team2exe(example: C:\yourpath\Team_number.exe): D:\NCTU\108_2\Intro. to Artificial Intelligence\AI_GameProject_2020\AI_GameProject_2020.exe
錯誤: 找不到處理程序 "" 。
錯誤: 找不到處理程序 "" 。


round 1:  black:[team 1]  white:[team 42]

---------START GAME---------

it's player [black]'s turn
movement: (2, 3)
-  0  1  2  3  4  5  6  7
0     -  -  -  -  -  -

1  -  -  -  -  -  -  -  -

2  -  -  -  X  -  -  -  -

3  -  -  -  -  -  -  -  -

4  -  -  -  -  -  -  -  -

5  -  -  -  -  -  -  -  -

6  -  -  -  -  -  -  -  -

7     -  -  -  -  -


it's player [white]'s turn
movement: (4, 3)
-  0  1  2  3  4  5  6  7
0     -  -  -  -  -  -

1  -  -  -  -  -  -  -  -

2  -  -  -  X  -  -  -  -

3  -  -  -  -  -  -  -  -

4  -  -  -  O  -  -  -  -

5  -  -  -  -  -  -  -  -

6  -  -  -  -  -  -  -  -

7     -  -  -  -  -
```





vs. Sample2.exe





Experiment of depth:

To understand how the depth of search influence to the performance. I let computer play black and white by itself with different depth, and here is the result.

	2	3	4	non-const
2	4/6/0	2/8/0	2/8/0	0/10/0
3	6/4/0	3/7/0	1/9/0	0/10/0
4	6/4/0	5/5/0	2/8/0	1/9/0
non-const	10/0/0	9/1/0	9/1/0	3/5/2

The row represents to black piece depth, and the column represents to white piece depth. The non-const depth is the dynamic depth that has described in the Description section at the begin of this report. The data in every grid represents in black win/white win/draw.

By this experiment, we can see that the deeper search is, the performance is greater.

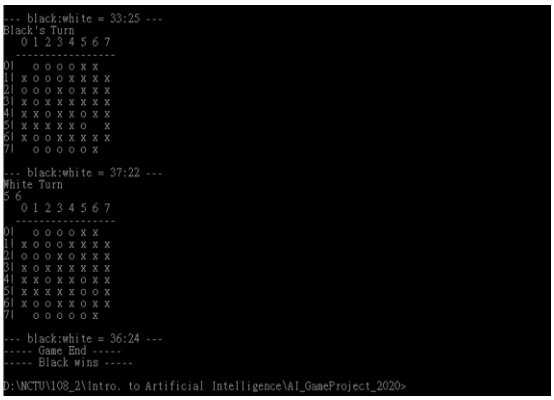
The reason why the non-const depth will has the highest performance is because around the end of game, the depth of searching will up to 6. It can cover almost all possible goal states, which means around the end of game, the program just finds the greatest step that leads to the greatest goal.

Another interesting thing that I discover in the experiment is that white piece seems to have better advantage than the black piece (first hand).

Appendix:

Because testing by AI_game.exe is inconvenient and hard to debug, I write main.cpp to simulate the game. Hence, I can play with my program

and trace if there exist unreasonable move of the program.
Below is the latest version of the program, black is computer.



main.cpp:

```
#include <iostream>
#include <vector>
#include <climits> // INT_MAX, INT_MIN
#include <cstdlib> // random
#include <ctime> // time
// v612.20.30-beta_003
// [*] dynamic max_depth for different step condition

class Reversi{
// [0: unoccupied]; [1: occupied by Black]; [2: occupied by White]; [-1:
the four corners]
private:
	bool Blackturn;
	std::vector<std::vector<int> > Board;
public:
	Reversi();
	bool GetBoard(std::vector<std::vector<int> > & board);
	void SendStep(int player,std::vector<int> & step);
	void printBoard();
};
// AI, from here
int dx[] = {-1,-1,-1,0,0,1,1,1},dy[] = {-1,0,1,-1,1,-1,0,1}; // direction
int score[8][8] = {
	{0,100,1,1,1,1,100,0},
	{100,1,1,1,1,1,1,100},
	{1,1,1,1,1,1,1,1},
```

```
	{1,1,1,1,1,1,1,1},
	{1,1,1,1,1,1,1,1},
	{1,1,1,1,1,1,1,1},
	{100,1,1,1,1,1,1,100},
	{0,100,1,1,1,1,100,0}
};
std::vector<std::vector<int> > Filp_piece(std::vector<std::vector<int> >
board,std::vector<int> & step,bool is_black){
	int x = step[0],y = step[1];
	board[x][y] = (is_black?1:2);
	for(int di = 0;di < 8;di++){
		int nx = x + dx[di],ny = y + dy[di];
		while(0 <= nx && nx < 8 && 0 <= ny && ny < 8 &&
board[nx][ny] == (is_black?2:1)){
			nx += dx[di];
			ny += dy[di];
			if(0 <= nx && nx < 8 && 0 <= ny && ny < 8 &&
board[nx][ny] == (is_black?1:2)){
				while(nx != x || ny != y){
					board[nx][ny] = (is_black?1:2);
					nx -= dx[di];
					ny -= dy[di];
				}
				break;
			}
		}
	}
	return board;
}
int Utility(std::vector<std::vector<int> > board,bool is_black){
	// piece difference utility (can prevent nearby death move)
	int utility = 0;
	for(int i = 0;i < 8;i++)
		for(int j = 0;j < 8;j++){
			if(board[i][j] == (is_black?1:2))
				utility += score[i][j];
			else if(board[i][j] == (is_black?2:1))
				utility -= score[i][j];
```

```
        }
    return utility; // return the utility
}
bool isStepLegal(std::vector<std::vector<int> > & board,std::vector<int>
& step,bool is_black){
    int x = step[0],y = step[1];
    if(board[x][y] != 0)
        return false;
    if(x != 0 && x != 7 && y != 0 && y != 7)
        return true;
    for(int di = 0;di < 8;di++){
        int nx = x + dx[di],ny = y + dy[di];
        while(0 <= nx && nx < 8 && 0 <= ny && ny < 8 &&
board[nx][ny] == (is_black?2:1)){
            nx += dx[di];
            ny += dy[di];
            if(0 <= nx && nx < 8 && 0 <= ny && ny < 8 &&
board[nx][ny] == (is_black?1:2))
                return true;
        }
    }
    return false;
}
int Min_value(std::vector<std::vector<int> > board,bool is_black,int
alpha,int beta,int depth,int max_depth);
int Max_value(std::vector<std::vector<int> > board,bool is_black,int
alpha,int beta,int depth,int max_depth){
    if(depth == max_depth)
        return Utility(board,is_black);
    int max_val = INT_MIN;
    std::vector<int> step_n(2);
    for(int i = 0;i < 8;i++)
        for(int j = 0;j < 8;j++){
            step_n[0] = i;
            step_n[1] = j;
            if(isStepLegal(board,step_n,is_black)){
                int utility =
Min_value(Filp_piece(board,step_n,is_black),is_black,alpha,beta,depth +
```

```
1,max_depth);
                    if(utility > max_val)
                        max_val = utility;
                    if(max_val >= beta)
                        return max_val;
                    if(max_val > alpha)
                        alpha = max_val;
                }
            }
        if(max_val == INT_MIN)
            return Utility(board,is_black);
        return max_val;
}
int Min_value(std::vector<std::vector<int> > board,bool is_black,int
alpha,int beta,int depth,int max_depth){
        if(depth == max_depth)
            return Utility(board,is_black);
        int min_val = INT_MAX;
        std::vector<int> step_n(2);
        for(int i = 0;i < 8;i++)
            for(int j = 0;j < 8;j++){
                step_n[0] = i;
                step_n[1] = j;
                if(isStepLegal(board,step_n,!is_black)){
                    int utility =
Max_value(Filp_piece(board,step_n,!is_black),is_black,alpha,beta,depth
+ 1,max_depth);
                    if(utility < min_val)
                        min_val = utility;
                    if(min_val <= alpha)
                        return min_val;
                    if(min_val < beta)
                        beta = min_val;
                }
            }
        if(min_val == INT_MAX)
            return Utility(board,is_black);
        return min_val;
```

```
}
std::vector<int> GetStep(std::vector<std::vector<int> >& board,bool
is_black){
	std::vector<int> step;
	step.resize(2);
	step[0] = step[1] = -1; // step[0]: row, step[1]: column
	if(!is_black){
		scanf("%d%d",&step[0],&step[1]);
		return step;
	}
	std::vector<std::vector<int> > step_arr,optimal_step_arr;
	int max_val = INT_MIN,pieces = 0;
	for(int i = 0;i < 8;i++)
		for(int j = 0;j < 8;j++){
			step[0] = i;
			step[1] = j;
			if(board[i][j] == 1 || board[i][j] == 2)
				pieces++;
			if(isStepLegal(board,step,is_black)){ // first step of
Max_val()
				if(score[step[0]][step[1]] == 100)
					return step;
				step_arr.push_back(step);
			}
		}

	int max_depth = 2;
	if(pieces >= 50)
		max_depth = 6;
	else if(pieces >= 40)
		max_depth = 5;
	else if(pieces >= 30)
		max_depth = 4;
	else if(pieces >= 8)
		max_depth = 3;
	for(int si = 0;si < step_arr.size();si++){
		step = step_arr[si];
		int utility =
```

```
Min_value(Filp_piece(board,step,is_black),is_black,INT_MIN,INT_MA
X,0,max_depth);
            if(utility > max_val){
                max_val = utility;
                optimal_step_arr.clear();
                optimal_step_arr.push_back(step);
            }
            else if(utility == max_val)
                optimal_step_arr.push_back(step);
        }
        srand(time(NULL));
        int index = rand() % optimal_step_arr.size();
        return optimal_step_arr[index];
}
// AI, to here
int main(int argc, char const *argv[]){
        Reversi game;
        // p1: black, p2: white
        std::vector<std::vector<int> > board;
        std::vector<int> step;
        while(true){
            bool is_black = game.GetBoard(board);
            printf(is_black?"Black's Turn\n":"White Turn\n");
            step = GetStep(board,is_black);
            game.SendStep(is_black?1:2,step);
        }
        return 0;
}
Reversi::Reversi(){
        Board = std::vector<std::vector<int> >(8,std::vector<int>(8));
        Board[0][0] = Board[0][7] = Board[7][0] = Board[7][7] = -1; //
corner
        Blackturn = true;
}
bool Reversi::GetBoard(std::vector<std::vector<int> > & board){
        // return 1: black turns, 0: white turns
        // usually return Blackturn if player(Blackturn?1:2) has legal move
        printBoard();
```

```
	board = Board;
	for(int i = 0;i < 8;i++)
		for(int j = 0;j < 8;j++){
			if(i == 0 || i == 7 || j == 0 || j == 7){
				// edge case, legal if flip event exists
				if(Board[i][j] == 0)
					for(int di = 0;di < 8;di++){
						int nx = i + dx[di],ny = j + dy[di];
						while(0 <= nx && nx < 8 && 0 <= ny && ny <
8 && Board[nx][ny] == (Blackturn?2:1)){
							nx += dx[di];
							ny += dy[di];
							if(0 <= nx && nx < 8 && 0 <= ny && ny
< 8 && Board[nx][ny] == (Blackturn?1:2))
								return Blackturn;
						}
					}
			}
			else if(!Board[i][j])
				return Blackturn; // legal if empty square in central 6x6
exists
		}
	// No legal move for present player
	return Blackturn ^ true;
}
void Reversi::SendStep(int player,std::vector<int> & step){
	// check if the position player choose is legal
	int x = step[0],y = step[1];
	if(0 > x || x >= 8 || 0 > y || y >= 8){
		printf("(%d,%d) ILEGAL MOVE DETECT\n",x,y);
		return; // outside board
	}
	if(Board[x][y] != 0){
		printf("(%d,%d) ILEGAL MOVE DETECT\n",x,y);
		return; // has been occupied
	}
	bool legal = (x != 0 && x != 7 && y != 0 && y != 7);
	for(int di = 0;di < 8;di++){
```

```
            int nx = x + dx[di],ny = y + dy[di];
            while(0 <= nx && nx < 8 && 0 <= ny && ny < 8 &&
Board[nx][ny] == (player == 1?2:1)){
                nx += dx[di];
                ny += dy[di];
                if(0 <= nx && nx < 8 && 0 <= ny && ny < 8 &&
Board[nx][ny] == player){
                    legal = true;
                    while(nx != x || ny != y){
                        Board[nx][ny] = player;
                        nx -= dx[di];
                        ny -= dy[di];
                    }
                    break;
                }
            }
        }
        if(!legal){
            printf("(%d,%d) ILEGAL MOVE DETECT\n",x,y);
            return;
        }
        Board[x][y] = player;

        Blackturn ^= true; // change turn
}
void Reversi::printBoard(){
        int unoccupied = 60,white = 0,black = 0;
        printf("      ");
        for(int j = 0;j < 8;j++)
            printf("%d ",j % 10);
        printf("\n    -");
        for(int j = 0;j < 8;j++)
            printf("--");
        printf("\n");
        for(int i = 0;i < 8;i++){
            printf("%d| ",i % 10);
            for(int j = 0;j < 8;j++){
                if(Board[i][j] == 1){
```

```
                unoccupied--;
                black++;
                printf("x ");
            }
            else if(Board[i][j] == 2){
                unoccupied--;
                white++;
                printf("o ");
            }
            else
                printf("    ");
        }
        printf("\n");
    }
    printf("\n");
    printf("--- black:white = %d:%d ---\n",black,white);
    if(!unoccupied){
        printf("----- Game End -----\n");
        if(black == white)
            printf("------- Draw -------\n");
        else
            printf(black > white?"----- Black wins -----\n":"----- White
wins -----\n");
        exit(0);
    }
}
```