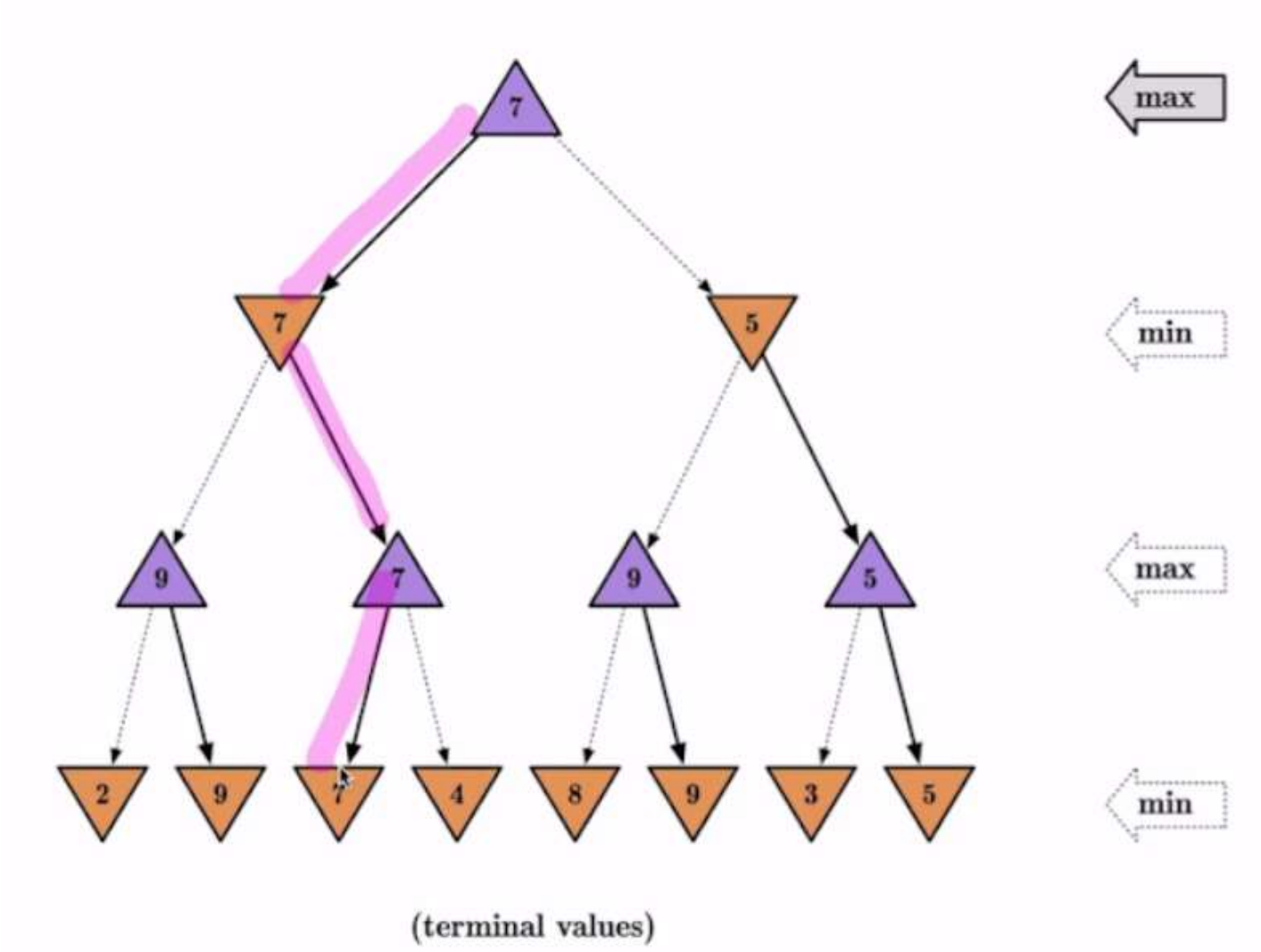
計算機程式設計 (二) MiniProject3

107062372 張凱博

Tree Search - Minimax



Tree Search Optimization - AlphaBeta pruning

我們知道root Max要做決定的話，要取max(3, Z, 2) 但是由於Z = min(2,X,Y) <= 2 < 3，所以我們不care Z多少了，因為對我們要取max來說沒貢獻了。 所以不必explore X Y就可以知道 root Max要取 max(3 , DONT_CARE, 2) = 3 這樣馬上就省略了一個subtree需要去explore!

alpha-beta pruning仍然是一個minimax演算法（DFS)，但是添加了pruning的機制。 我們需要多兩個輔助變數：

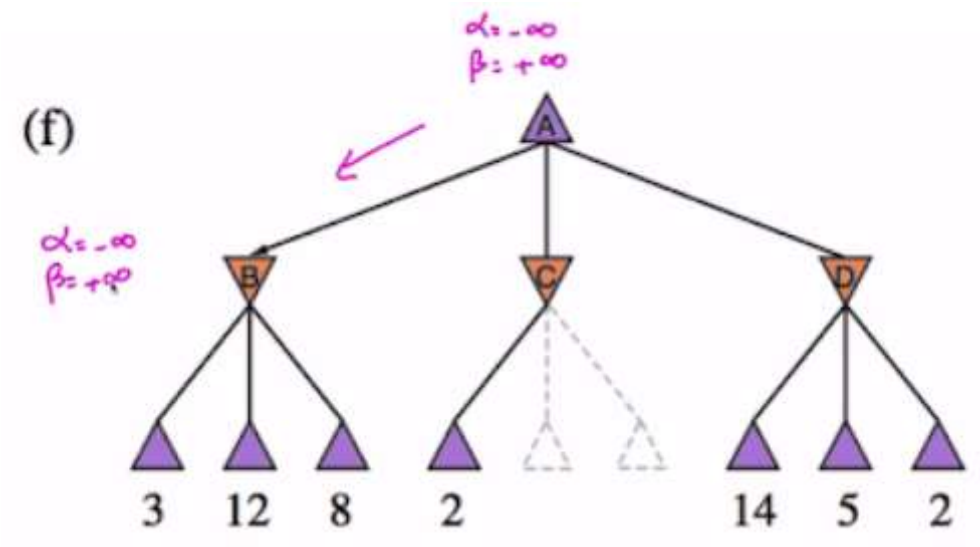
> alpha - 目前Max可做的move的最大utility score ，所以初始化要是 -INF，只能被Max node update
> beta - 目前Min可做的move的最小utility score，所以初始化要是 +INF，只能被Min node update

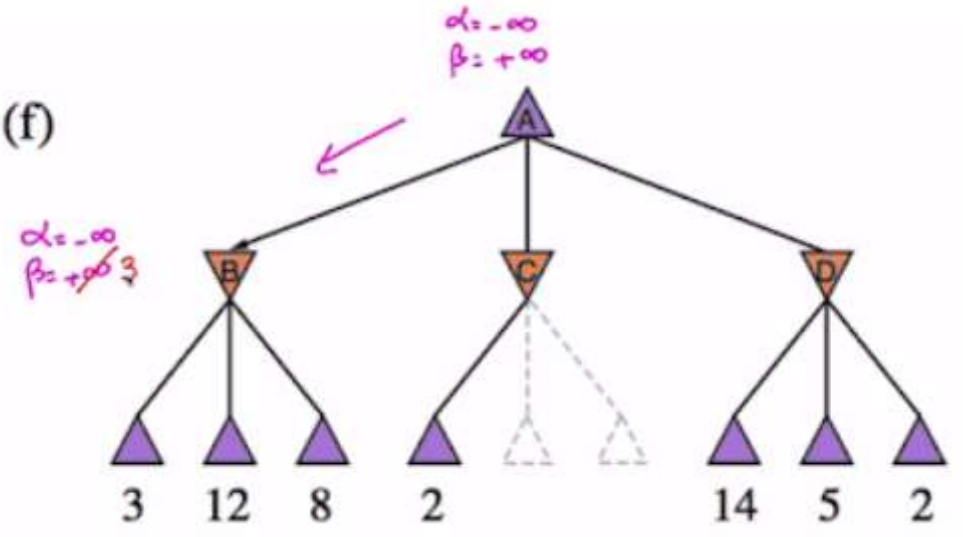
- 當 alpha >= beta 的時候，代表pruning發生

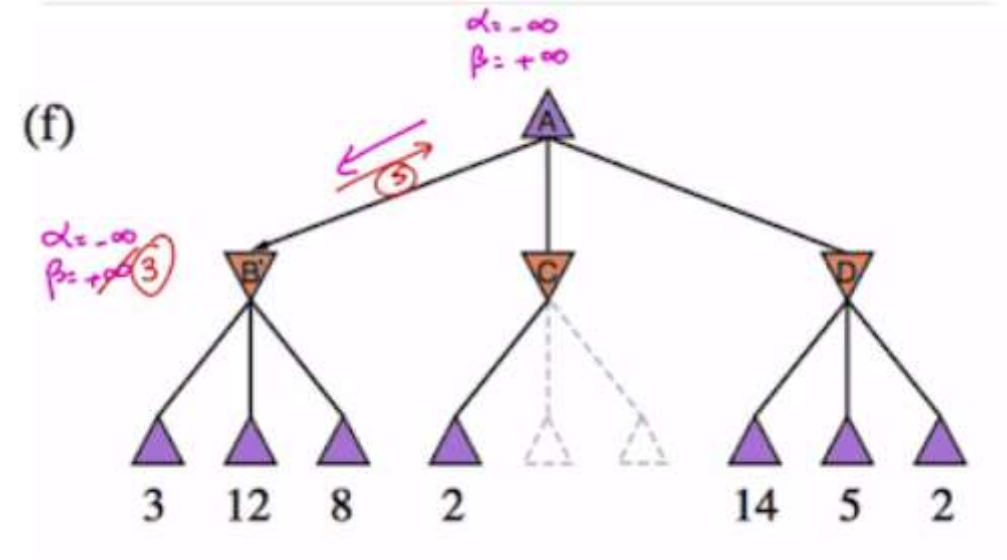
對一個min node來說，它看到目前的parent max node的alpha/beta值，而這個min node只能藉由其children的utility score來update beta值。如果它發現alpha >= beta，由於此min node能回傳的utility score最大也只能是目前的beta，意即此min node再也無法提供其parent max node大於目前alpha值的utility score，所以剩下的所unexplored的children都可以prune掉不看了。

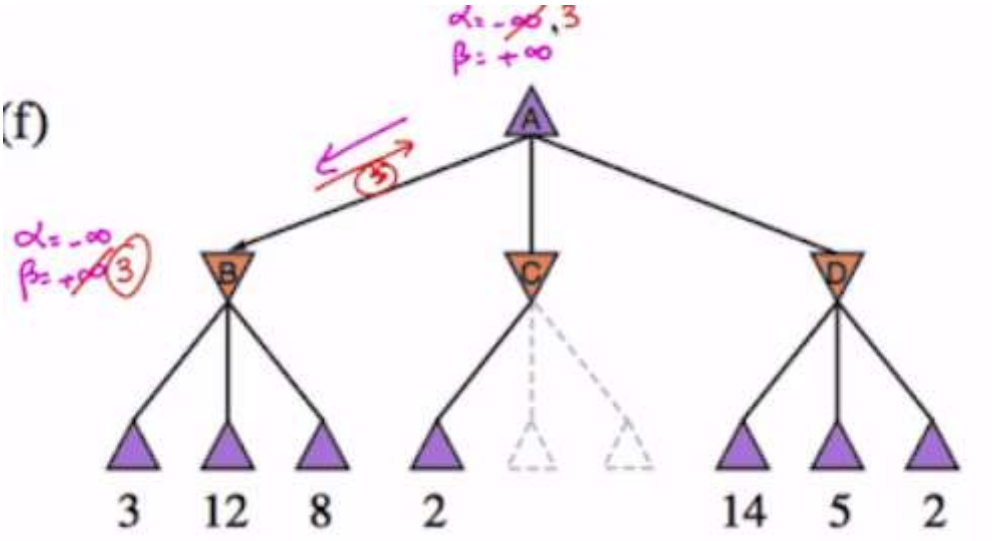
同理，對一個max node來說，它看到目前的parent min node的alpha/beta值，而這個max node只能藉由其children的utility score來update alpha值。如果它發現beta <= alpha，由於此max node能回傳的utility score最小也只能是目前的alpha，意即此max node再也無法提供其parent min node小於目前beta值的utility score，所以剩下的所unexplored的children都可以prune掉不看了。

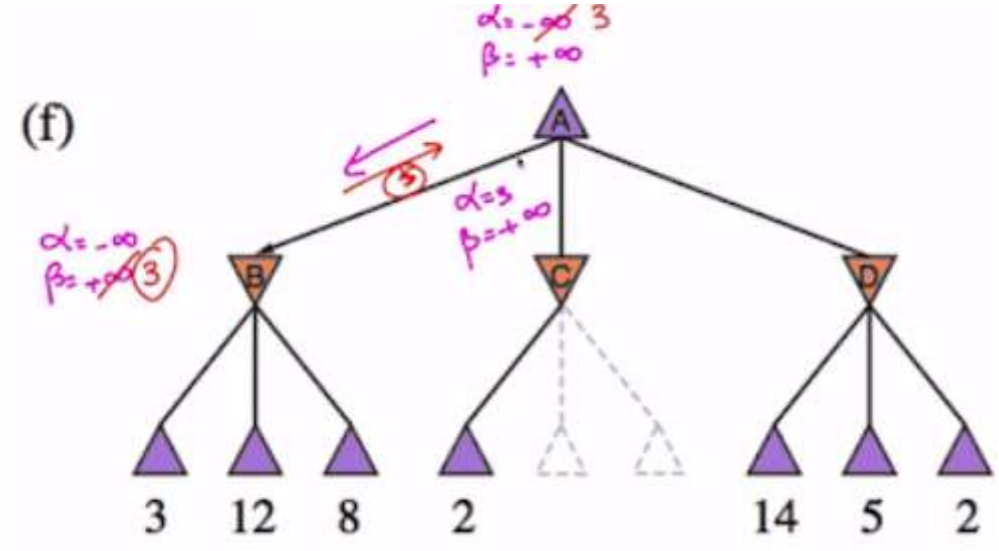
流程

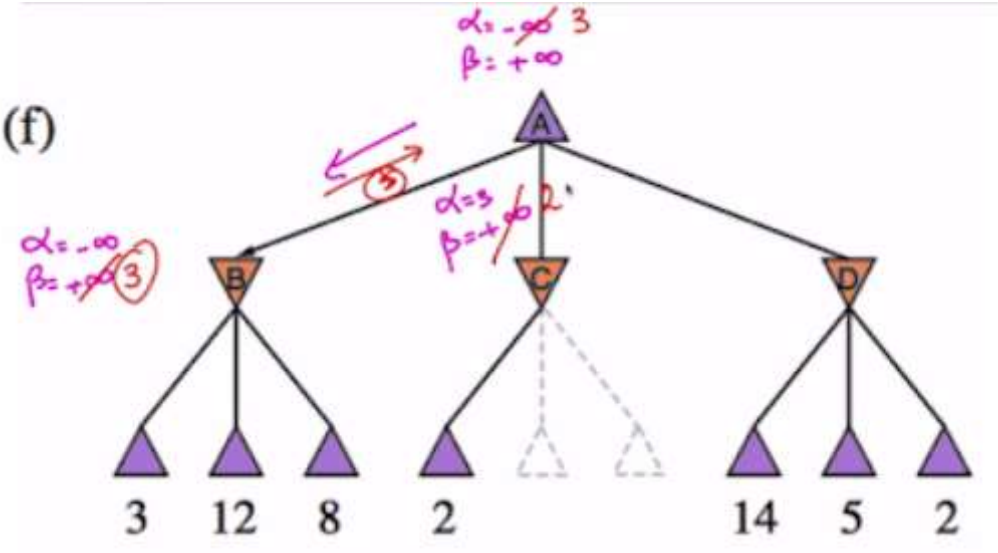




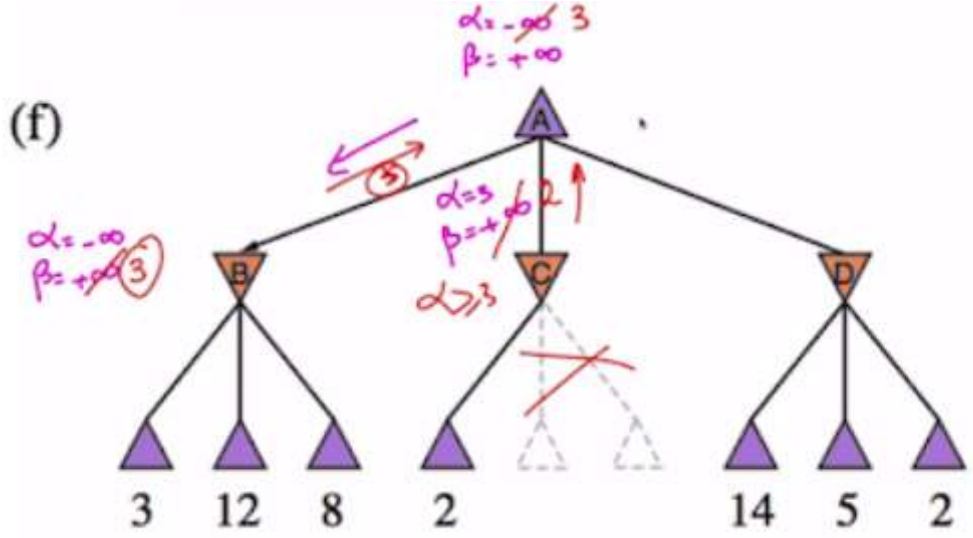

(f)
α = -∞
β = +∞
A
B
C
D
3
12
8
2
14
5
2


(f)
α = -∞, 3
β = +∞
A
B
C
D
3
12
8
2
14
5
2

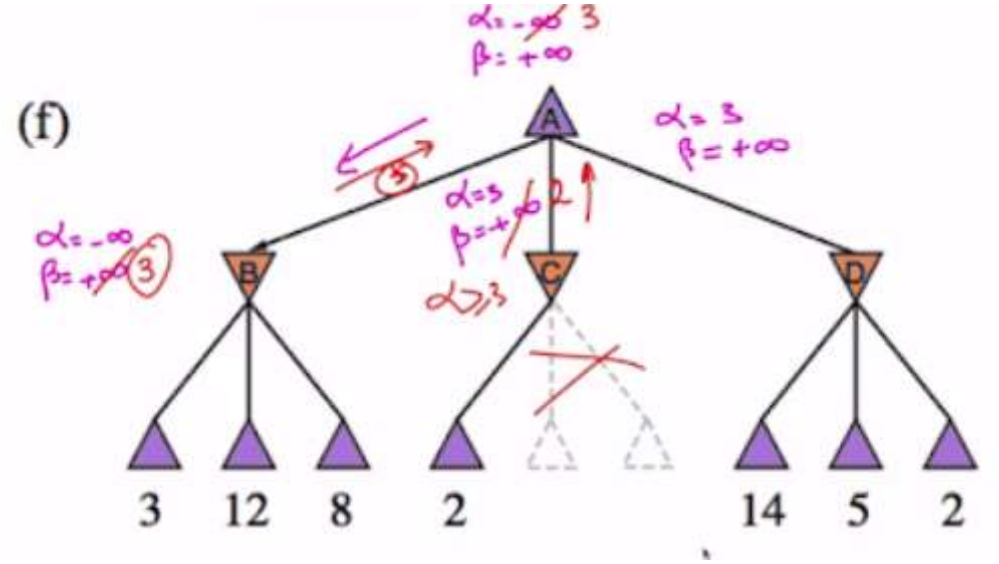


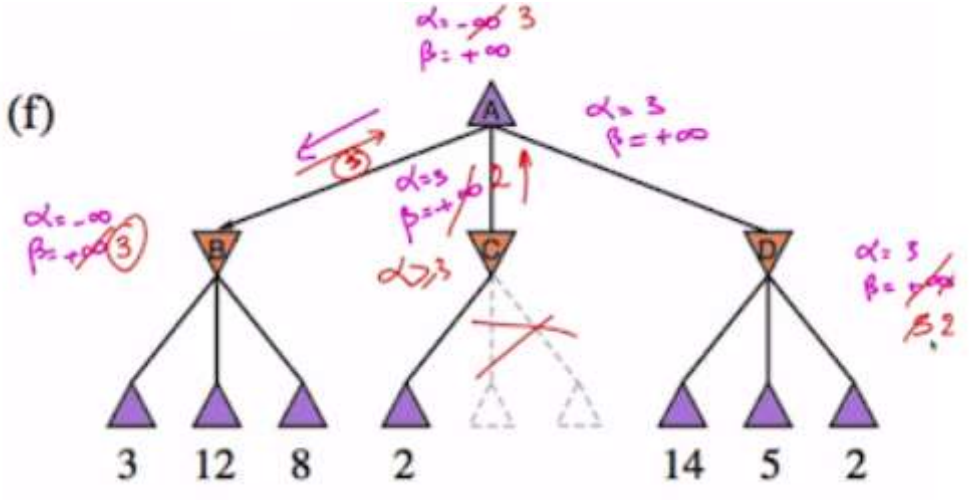


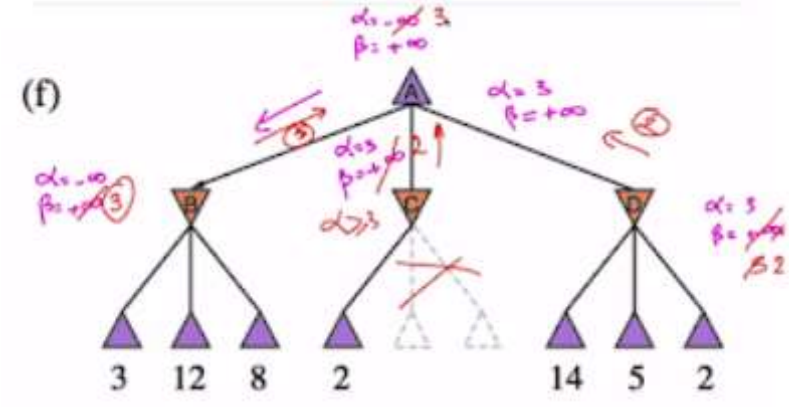
<下圖發生**Pruning**> 因為 min node C 發現了目前(alpha = 3) >= (beta = 2)，這表示Min node C最大的utility score只能是2（因為C只能從children中選擇 <= 2的utility score)，而對root Max A來說，既然child C最大只能回傳utility = 2，但是目前已經有找到一個utility score = 3 = alpha了，那C這個node是不會被 root Max A所選擇的，所以剩下的C的unexplored children也不會造成任何影響了，可以放心的



prune掉。








```
int minimax(Point p, array<array<int, SIZE>, SIZE> board,
    int player, int depth, bool isMaximizingPlayer, int alpha, int beta)
{
    if (depth == 3){
        int h = set_heuristic(board, player);
        return h;
    }
    if (isMaximizingPlayer){
        int bestValue = NEG_INF;
        vector<Point> valid_spots = get_valid_spots(board, player);
        for(Point c : valid_spots){
            array<array<int, SIZE>, SIZE> newBoard = copyBoard(board);
            put_disc(c, newBoard, player);
```

```
                int value = minimax(p, newBoard, player, depth+1, false, alpha, beta);
                bestValue = max(bestValue, value);
                alpha = max(alpha, bestValue);
                if (beta <= alpha)
                    break;
            }
            return bestValue;
        }
        else{
            int bestValue = POS_INF;
            vector<Point> valid_spots = get_valid_spots(board, player);
            for(Point c : valid_spots){
                array<array<int, SIZE>, SIZE> newBoard = copyBoard(board);
                put_disc(c, newBoard, player);

                int value = minimax(p, newBoard, player, depth+1, true, alpha, beta);
                bestValue = min(bestValue, value);
                beta = min(beta, bestValue);
                if (beta <= alpha)
                    break;
            }
            return bestValue;
        }
    }
```

Heuristic function

1. 計算落子分數以及frontier分數

p = 計算地圖位置分數比後，在跟據落子點計算分數權重 f = 計算每一個空的位置，那個空位置旁邊的棋子是誰的，希望不要是自己的，因為這樣容易被別人吃掉，把空位置旁邊的棋子稱為 frontier_disc，所以評分函數，f = (-1)*(my_front_discs - opp_front_discs)

```
    int X1[] = {-1, -1, 0, 1, 1, 1, 0, -1};
        int Y1[] = {0, 1, 1, 1, 0, -1, -1, -1};
        int V[8][8] = {
            {2000, -300, 11, 8, 8, 11, -300, 2000},
            {-300, -700, -4, 1, 1, -4, -700, -300},
            {11, -4, 2, 2, 2, 2, -4, 11},
            {8, 1, 2, -3, -3, 2, 1, 8},
            {8, 1, 2, -3, -3, 2, 1, 8},
            {11, -4, 2, 2, 2, 2, -4, 11},
            {-300, -700, -4, 1, 1, -4, -700, -300},
            {2000, -300, 11, 8, 8, 11, -300, 2000}
        };

        // Piece difference, frontier disks and disk squares
        for (int i=0; i<8; i++){
            for (int j=0; j<8; j++){
                if (board[i][j] == player){
                    d += V[i][j];
```

```
                my_discs++;
            }else if (board[i][j] == get_next_player(player)){
                d -= V[i][j];
                opp_discs++;
            }
            if (board[i][j] != 0){
                for (int k=0; k<8; k++){
                    int x = i + X1[k];
                    int y = j + Y1[k];
                    if (x >= 0 && x < 8 && y >= 0 && y < 8 && board[x][y] == 0){
                        if (board[i][j] == player) my_front_discs++;
                        else opp_front_discs++;
                        break;
                    }
                }
            }
        }
    }
    p = my_discs - opp_discs;
    f = my_front_discs - opp_front_discs;
```

2. 四個角落的佔據

佔據四個角落很重要，所以 c 的權重很大

```
my_discs = opp_discs = 0;
if (board[0][0] == player) my_discs++;
else if (board[0][0] == get_next_player(player)) opp_discs++;
if (board[0][7] == player) my_discs++;
else if (board[0][7] == get_next_player(player)) opp_discs++;
if (board[7][0] == player) my_discs++;
else if (board[7][0] == get_next_player(player)) opp_discs++;
if (board[7][7] == player) my_discs++;
else if (board[7][7] == get_next_player(player)) opp_discs++;
c = 25*(my_discs - opp_discs);
```

3. 靠近四個角落，共12個點盡量不要下

```
my_discs = opp_discs = 0;
if (board[0][0] == 0){
    if (board[0][1] == player) my_discs++;
    else if (board[0][1] == get_next_player(player)) opp_discs++;
    if (board[1][1] == player) my_discs++;
    else if (board[1][1] == get_next_player(player)) opp_discs++;
    if (board[1][0] == player) my_discs++;
    else if (board[1][0] == get_next_player(player)) opp_discs++;
}
if (board[0][7] == 0){
    if (board[0][6] == player) my_discs++;
    else if (board[0][6] == get_next_player(player)) opp_discs++;
```

```
        if (board[1][6] == player) my_discs++;
        else if (board[1][6] == get_next_player(player)) opp_discs++;
        if (board[1][7] == player) my_discs++;
        else if (board[1][7] == get_next_player(player)) opp_discs++;
    }
    if (board[7][0] == 0){
        if (board[7][1] == player) my_discs++;
        else if (board[7][1] == get_next_player(player)) opp_discs++;
        if (board[6][1] == player) my_discs++;
        else if (board[6][1] == get_next_player(player)) opp_discs++;
        if (board[6][0] == player) my_discs++;
        else if (board[6][0] == get_next_player(player)) opp_discs++;
    }
    if (board[7][7] == 0){
        if (board[6][7] == player) my_discs++;
        else if (board[6][7] == get_next_player(player)) opp_discs++;
        if (board[6][6] == player) my_discs++;
        else if (board[6][6] == get_next_player(player)) opp_discs++;
        if (board[7][6] == player) my_discs++;
        else if (board[7][6] == get_next_player(player)) opp_discs++;
    }
    l = (-13) * (my_discs - opp_discs);
```

4. 限制對方可以下的步數，自己可以下越多步越好，對方能下的步數越少越好

```
vector<Point> my_discs_vec = get_valid_spots(board, player);
vector<Point> opp_discs_vec = get_valid_spots(board, get_next_player(player));
m = my_discs_vec.size() - opp_discs_vec.size();
```

5. Total heuristic score

```
score = (10 * p) + (2000* c) + (500 * l) + (60 * m) + (50 * f) + (10 * d);
```

Reference

1. https://fu-sheng-wang.blogspot.com/2017/02/ai-16-alpha-beta-pruning.html
2. https://fu-sheng-wang.blogspot.com/2017/02/ai-15-minimax.html
3. https://kartikkukreja.wordpress.com/2013/03/30/heuristic-function-for-reversiothello/