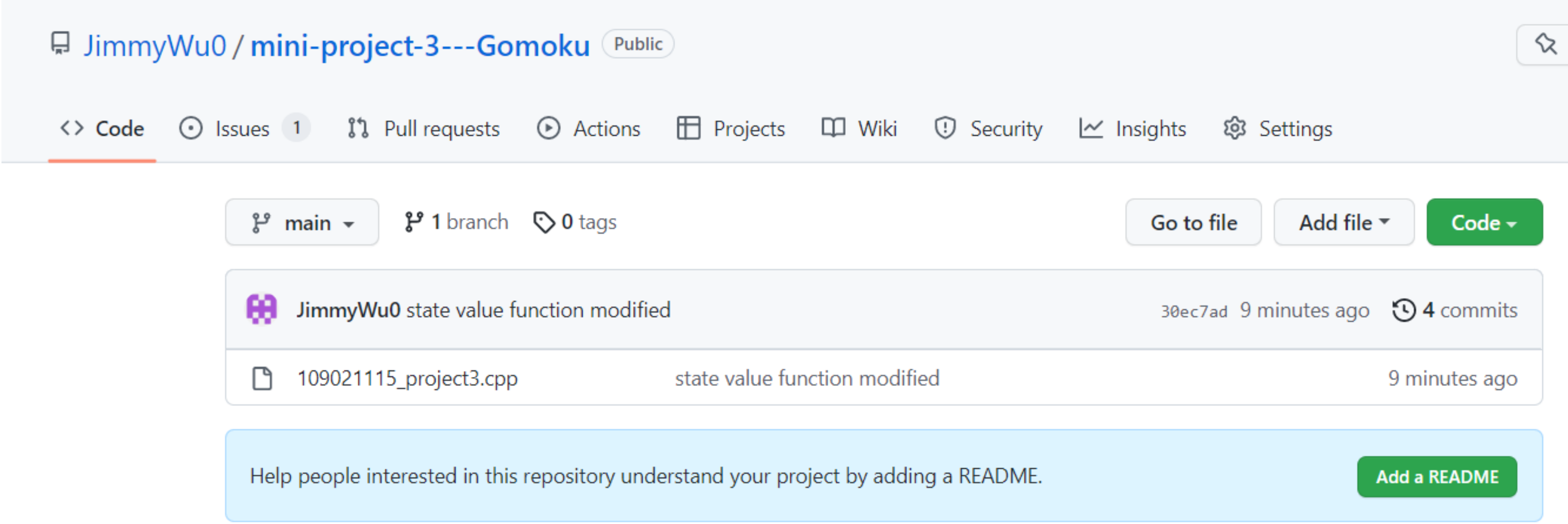
Version Control

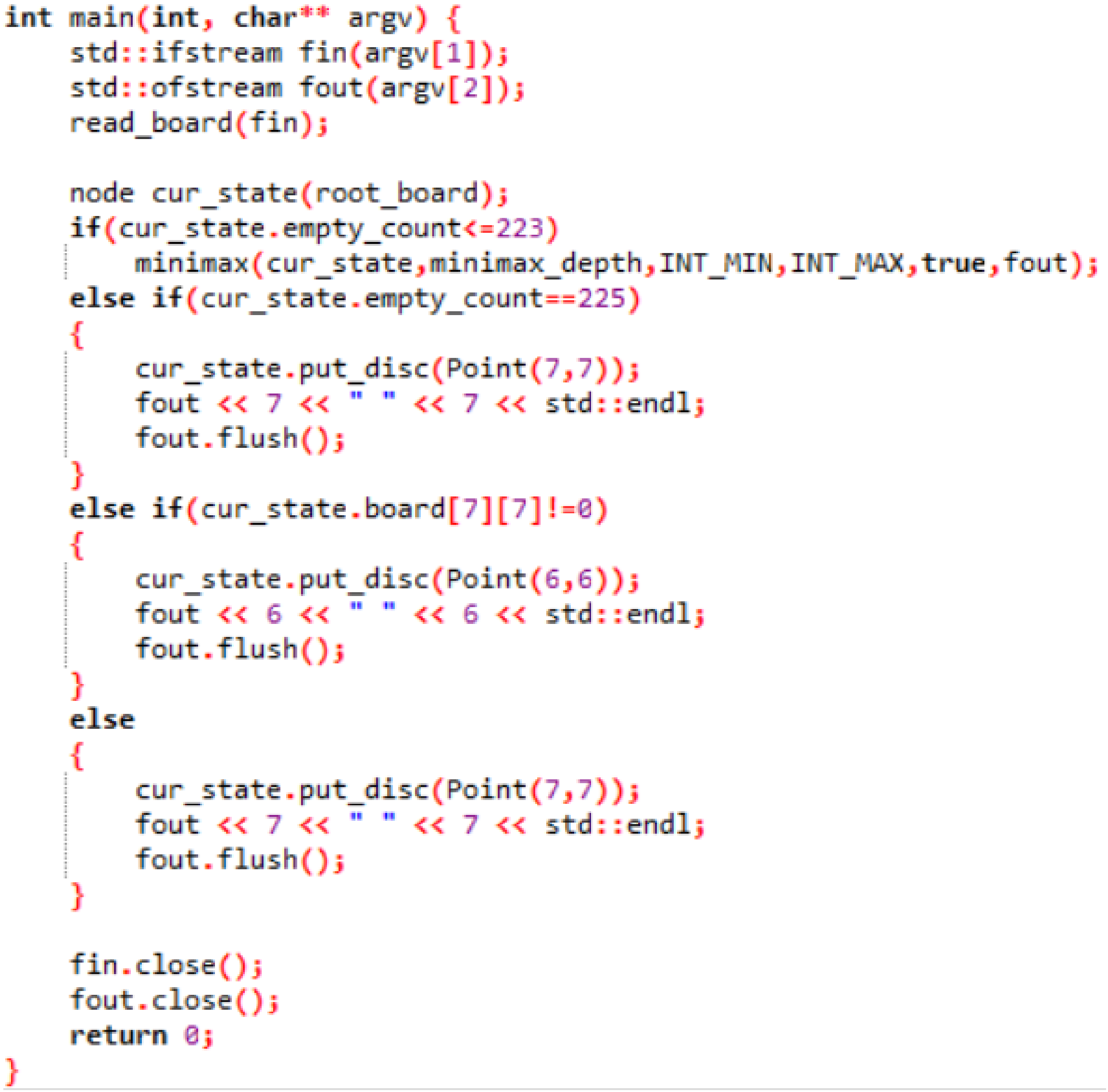


JimmyWu0 state value function modified 30ec7ad 9 minutes ago 4 commits

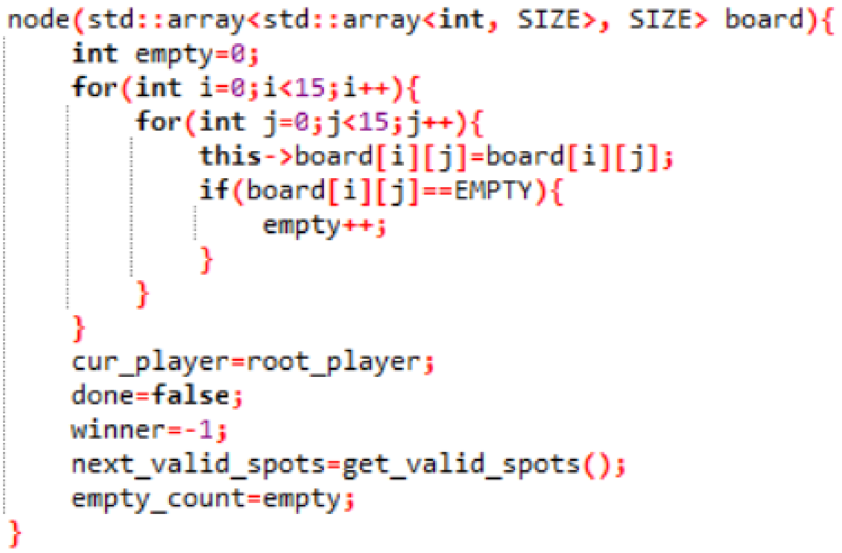
109021115_project3.cpp	state value function modified	9 minutes ago

Help people interested in this repository understand your project by adding a README.

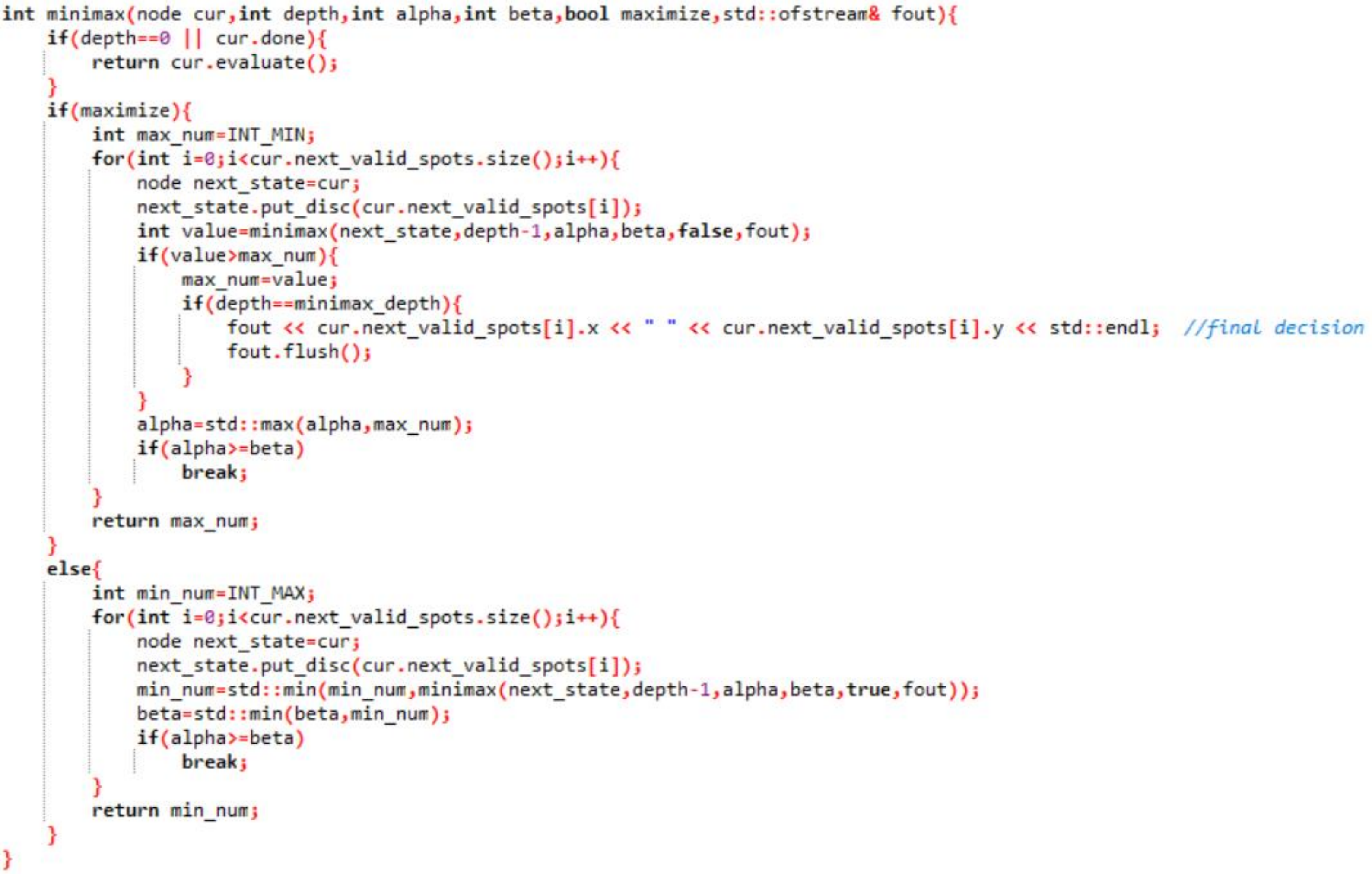
Minimax and alpha beta pruning



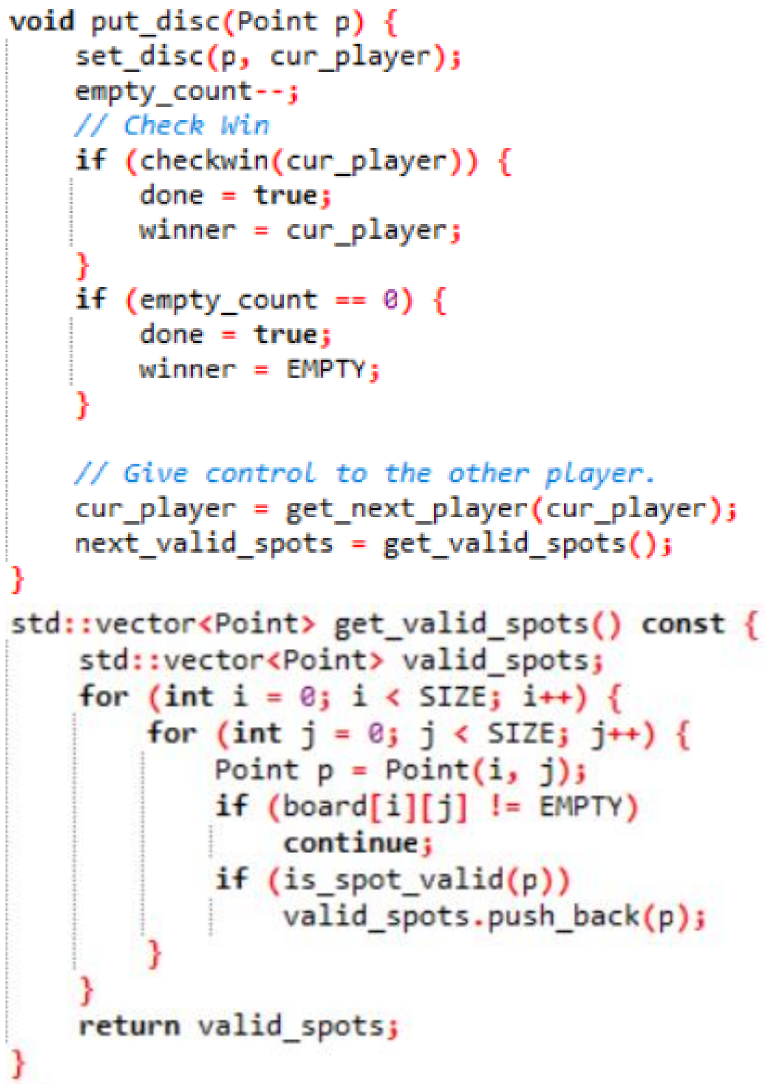
```
int main(int, char** argv) {
    std::ifstream fin(argv[1]);
    std::ofstream fout(argv[2]);
    read_board(fin);

    node cur_state(root_board);
    if(cur_state.empty_count<=223)
        minimax(cur_state,minimax_depth,INT_MIN,INT_MAX,true,fout);
    else if(cur_state.empty_count==225)
    {
        cur_state.put_disc(Point(7,7));
        fout << 7 << " " << 7 << std::endl;
        fout.flush();
    }
    else if(cur_state.board[7][7]!=0)
    {
        cur_state.put_disc(Point(6,6));
        fout << 6 << " " << 6 << std::endl;
        fout.flush();
    }
    else
    {
        cur_state.put_disc(Point(7,7));
        fout << 7 << " " << 7 << std::endl;
        fout.flush();
    }

    fin.close();
    fout.close();
    return 0;
}
```



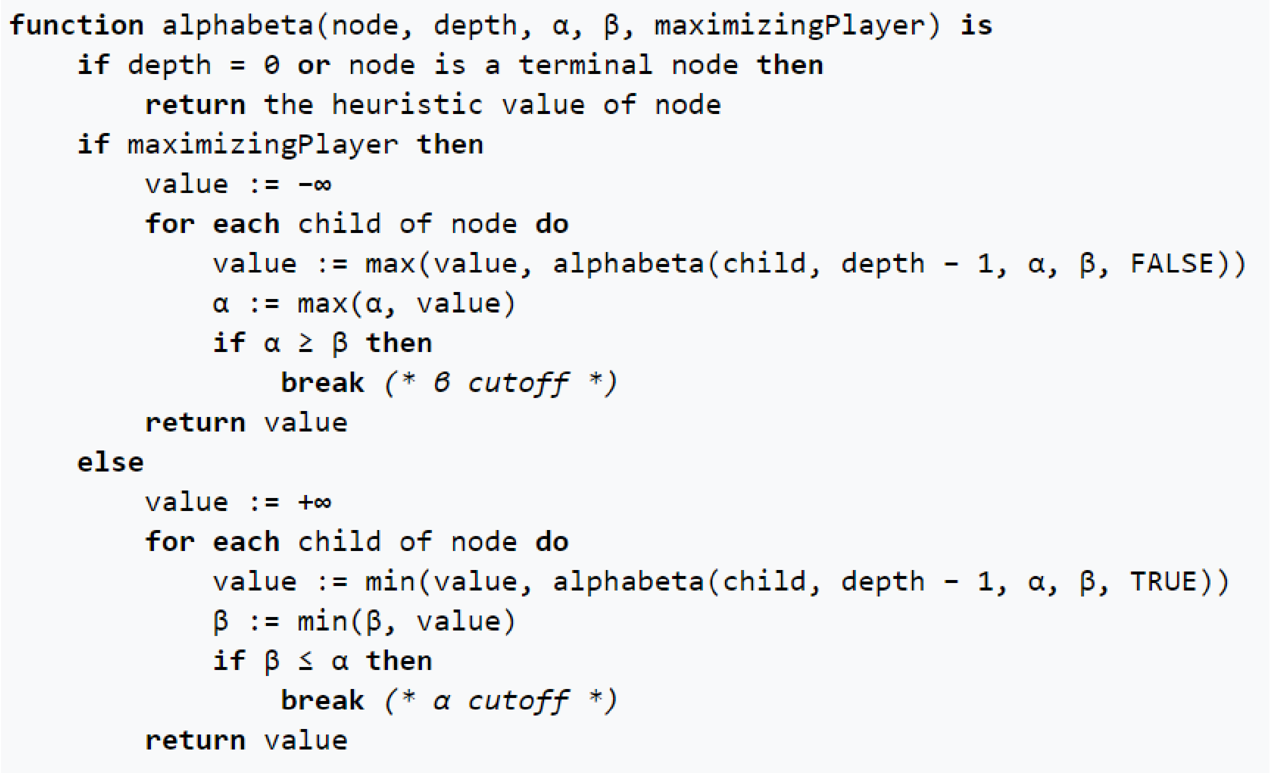
```
node(std::array<std::array<int, SIZE>, SIZE> board){
    int empty=0;
    for(int i=0;i<15;i++){
        for(int j=0;j<15;j++){
            this->board[i][j]=board[i][j];
            if(board[i][j]==EMPTY){
                empty++;
            }
        }
    }
    cur_player=root_player;
    done=false;
    winner=-1;
    next_valid_spots=get_valid_spots();
    empty_count=empty;
}
```



```
int minimax(node cur,int depth,int alpha,int beta,bool maximize,std::ofstream& fout){
    if(depth==0 || cur.done){
        return cur.evaluate();
    }
    if(maximize){
        int max_num=INT_MIN;
        for(int i=0;i<cur.next_valid_spots.size();i++){
            node next_state=cur;
            next_state.put_disc(cur.next_valid_spots[i]);
            int value=minimax(next_state,depth-1,alpha,beta,false,fout);
            if(value>max_num){
                max_num=value;
                if(depth==minimax_depth){
                    fout << cur.next_valid_spots[i].x << " " << cur.next_valid_spots[i].y << std::endl;  //final decision
                    fout.flush();
                }
            }
            alpha=std::max(alpha,max_num);
            if(alpha>=beta)
                break;
        }
        return max_num;
    }
    else{
        int min_num=INT_MAX;
        for(int i=0;i<cur.next_valid_spots.size();i++){
            node next_state=cur;
            next_state.put_disc(cur.next_valid_spots[i]);
            min_num=std::min(min_num,minimax(next_state,depth-1,alpha,beta,true,fout));
            beta=std::min(beta,min_num);
            if(alpha>=beta)
                break;
        }
        return min_num;
    }
}
```

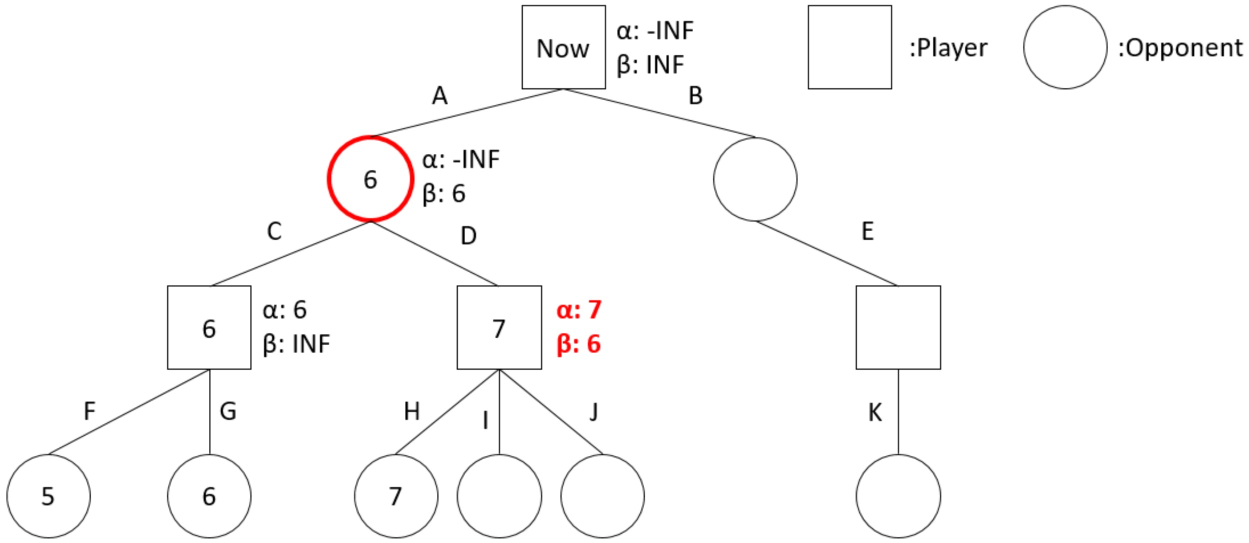


```cpp
void put_disc(Point p) {
    set_disc(p, cur_player);
    empty_count--;
    // Check Win
    if (checkwin(cur_player)) {
        done = true;
        winner = cur_player;
    }
    if (empty_count == 0) {
        done = true;
        winner = EMPTY;
    }

    // Give control to the other player.
    cur_player = get_next_player(cur_player);
    next_valid_spots = get_valid_spots();
}
std::vector<Point> get_valid_spots() const {
    std::vector<Point> valid_spots;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            Point p = Point(i, j);
            if (board[i][j] != EMPTY)
                continue;
            if (is_spot_valid(p))
                valid_spots.push_back(p);
        }
    }
    return valid_spots;
}
```

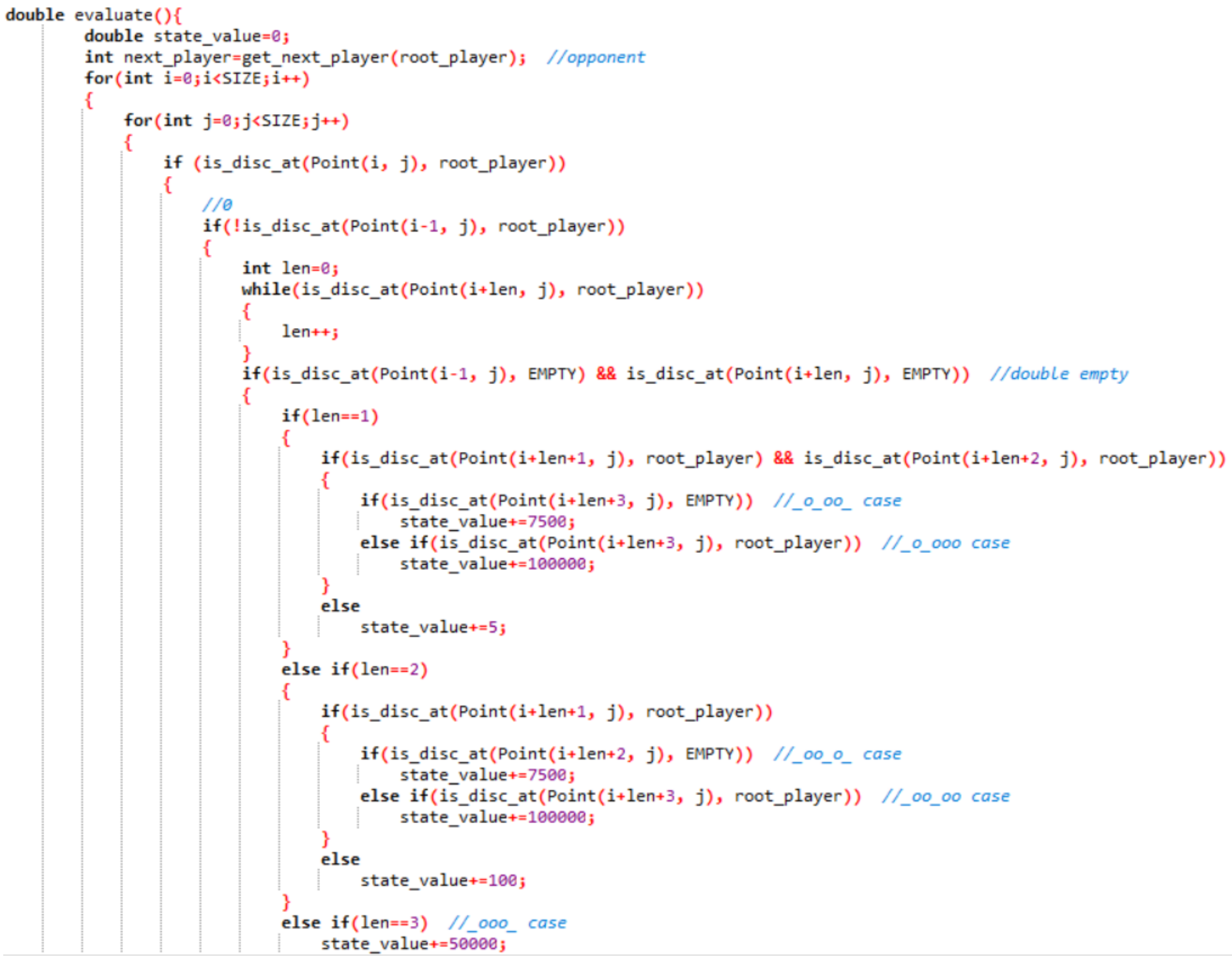


```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            α := max(α, value)
            if α ≥ β then
                break (* β cutoff *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, TRUE))
            β := min(β, value)
            if β ≤ α then
                break (* α cutoff *)
        return value
```

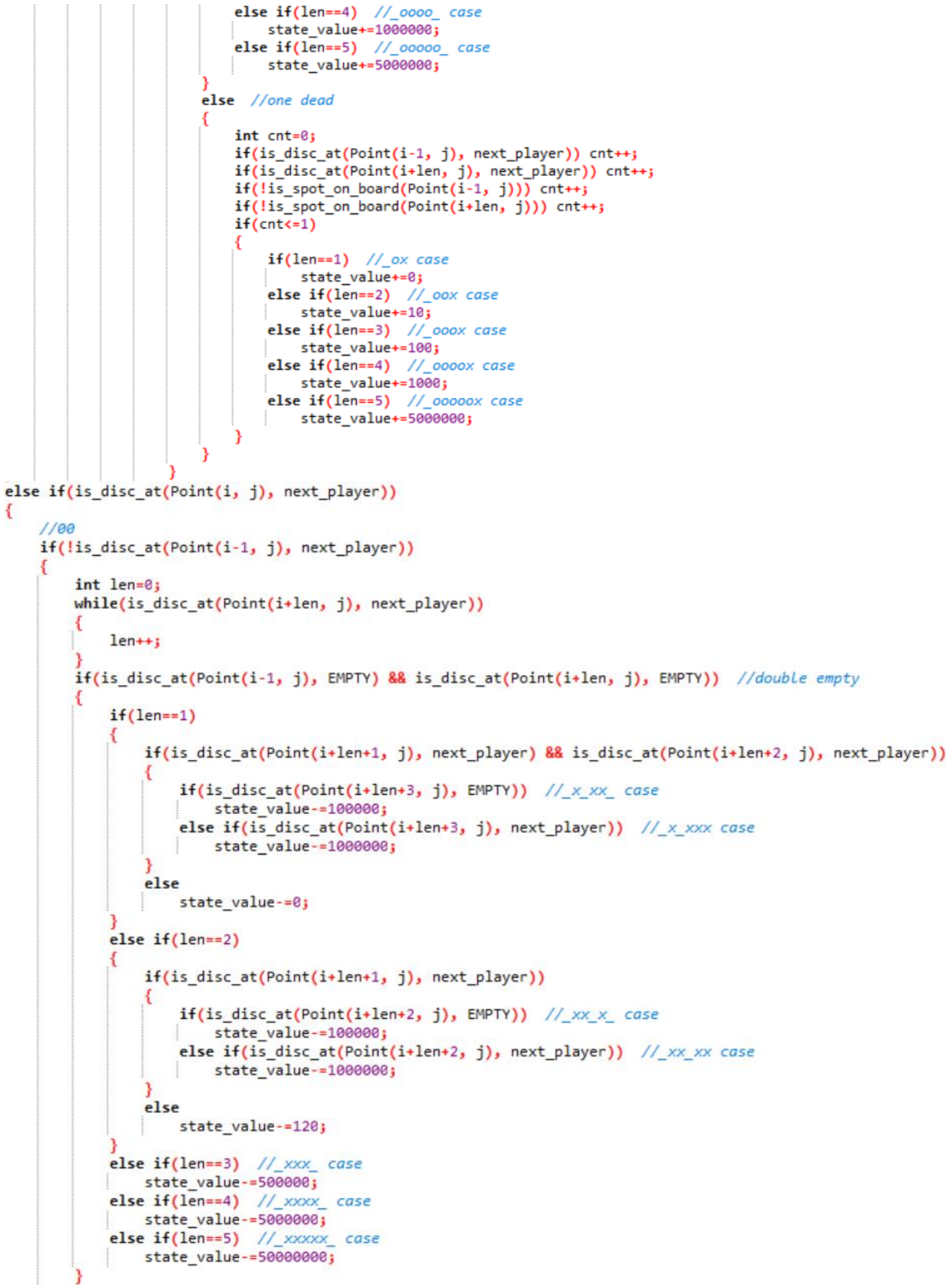
Opponent picks the smallest score



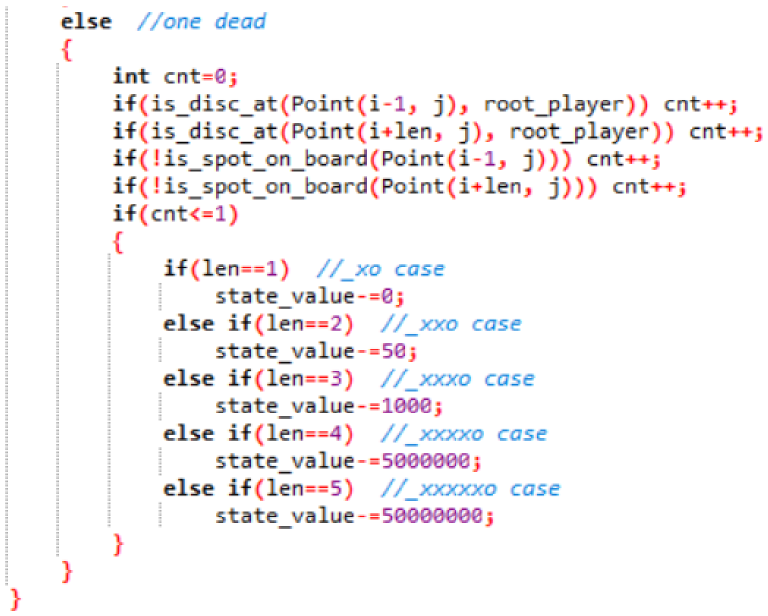
State Value Function Design



```
double evaluate(){
    double state_value=0;
    int next_player=get_next_player(root_player);  //opponent
    for(int i=0;i<SIZE;i++)
    {
        for(int j=0;j<SIZE;j++)
        {
            if (is_disc_at(Point(i, j), root_player))
            {
                //0
                if(!is_disc_at(Point(i-1, j), root_player))
                {
                    int len=0;
                    while(is_disc_at(Point(i+len, j), root_player))
                    {
                        len++;
                    }
                    if(is_disc_at(Point(i-1, j), EMPTY) && is_disc_at(Point(i+len, j), EMPTY))  //double empty
                    {
                        if(len==1)
                        {
                            if(is_disc_at(Point(i+len+1, j), root_player) && is_disc_at(Point(i+len+2, j), root_player))
                            {
                                if(is_disc_at(Point(i+len+3, j), EMPTY))  //_o_oo_ case
                                    state_value+=7500;
                                else if(is_disc_at(Point(i+len+3, j), root_player))  //_o_ooo case
                                    state_value+=100000;
                            }
                            else
                                state_value+=5;
                        }
                        else if(len==2)
                        {
                            if(is_disc_at(Point(i+len+1, j), root_player))
                            {
                                if(is_disc_at(Point(i+len+2, j), EMPTY))  //_oo_o_ case
                                    state_value+=7500;
                                else if(is_disc_at(Point(i+len+3, j), root_player))  //_oo_oo case
                                    state_value+=100000;
                            }
                            else
                                state_value+=100;
                        }
                        else if(len==3)  //_ooo_ case
                            state_value+=50000;
```



```
                            else if(len==4)  //_oooo_ case
                                state_value+=1000000;
                            else if(len==5)  //_ooooo_ case
                                state_value+=5000000;
                        }
                        else  //one dead
                        {
                            int cnt=0;
                            if(is_disc_at(Point(i-1, j), next_player)) cnt++;
                            if(is_disc_at(Point(i+len, j), next_player)) cnt++;
                            if(!is_spot_on_board(Point(i-1, j))) cnt++;
                            if(!is_spot_on_board(Point(i+len, j))) cnt++;
                            if(cnt<=1)
                            {
                                if(len==1)  //_ox case
                                    state_value+=0;
                                else if(len==2)  //_oox case
                                    state_value+=10;
                                else if(len==3)  //_ooox case
                                    state_value+=100;
                                else if(len==4)  //_oooox case
                                    state_value+=1000;
                                else if(len==5)  //_ooooox case
                                    state_value+=5000000;
                            }
                        }
                    }
else if(is_disc_at(Point(i, j), next_player))
{
    //00
    if(!is_disc_at(Point(i-1, j), next_player))
    {
        int len=0;
        while(is_disc_at(Point(i+len, j), next_player))
        {
            len++;
        }
        if(is_disc_at(Point(i-1, j), EMPTY) && is_disc_at(Point(i+len, j), EMPTY))  //double empty
        {
            if(len==1)
            {
                if(is_disc_at(Point(i+len+1, j), next_player) && is_disc_at(Point(i+len+2, j), next_player))
                {
                    if(is_disc_at(Point(i+len+3, j), EMPTY))  //_x_xx_ case
                        state_value-=100000;
                    else if(is_disc_at(Point(i+len+3, j), next_player))  //_x_xxx case
                        state_value-=1000000;
                }
                else
                    state_value-=0;
            }
            else if(len==2)
            {
                if(is_disc_at(Point(i+len+1, j), next_player))
                {
                    if(is_disc_at(Point(i+len+2, j), EMPTY))  //_xx_x_ case
                        state_value-=100000;
                    else if(is_disc_at(Point(i+len+2, j), next_player))  //_xx_xx case
                        state_value-=1000000;
                }
                else
                    state_value-=120;
            }
            else if(len==3)  //_xxx_ case
                state_value-=500000;
            else if(len==4)  //_xxxx_ case
                state_value-=5000000;
            else if(len==5)  //_xxxxx_ case
                state_value-=50000000;
        }
```



```
    else  //one dead
    {
        int cnt=0;
        if(is_disc_at(Point(i-1, j), root_player)) cnt++;
        if(is_disc_at(Point(i+len, j), root_player)) cnt++;
        if(!is_spot_on_board(Point(i-1, j))) cnt++;
        if(!is_spot_on_board(Point(i+len, j))) cnt++;
        if(cnt<=1)
        {
            if(len==1)  //_xo case
                state_value-=0;
            else if(len==2)  //_xxo case
                state_value-=50;
            else if(len==3)  //_xxxo case
                state_value-=1000;
            else if(len==4)  //_xxxxo case
                state_value-=5000000;
            else if(len==5)  //_xxxxxo case
                state_value-=50000000;
        }
    }
}
```