Computer Programming Final Project

Group 09

January 5, 2023

1 Introduction

This project presents Gomoku, also called Five in a Row or connect five (五子棋), which is a classical board game. It is traditionally played with Go pieces (black and white stones) on a 15×15 board.

Our program includes both PVP and PVC modes. For the later, we use Minimax algorithm as the decision rule for minimizing the possible loss for a worst case scenario. We also adopt alpha-beta pruning to decrease the number of nodes that are evaluated by the Minimax algorithm in its search tree.

The SDL library is included in order to implement GUI (graphical user interface).

2 Algorithms for PVC Mode

Minimax Algorithm with Alternate Moves

This is a recursive algorithm for choosing the next move in a 2-player game. A value is associated with each state of the game. This value is computed by means of a position evaluation function and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is A's turn to move, A gives a value to each of their legal moves.

Game Trees

This is a graph representing all possible game states within such a game. Such games include well-known ones such as chess, checkers, Go, and tic-tac-toe.

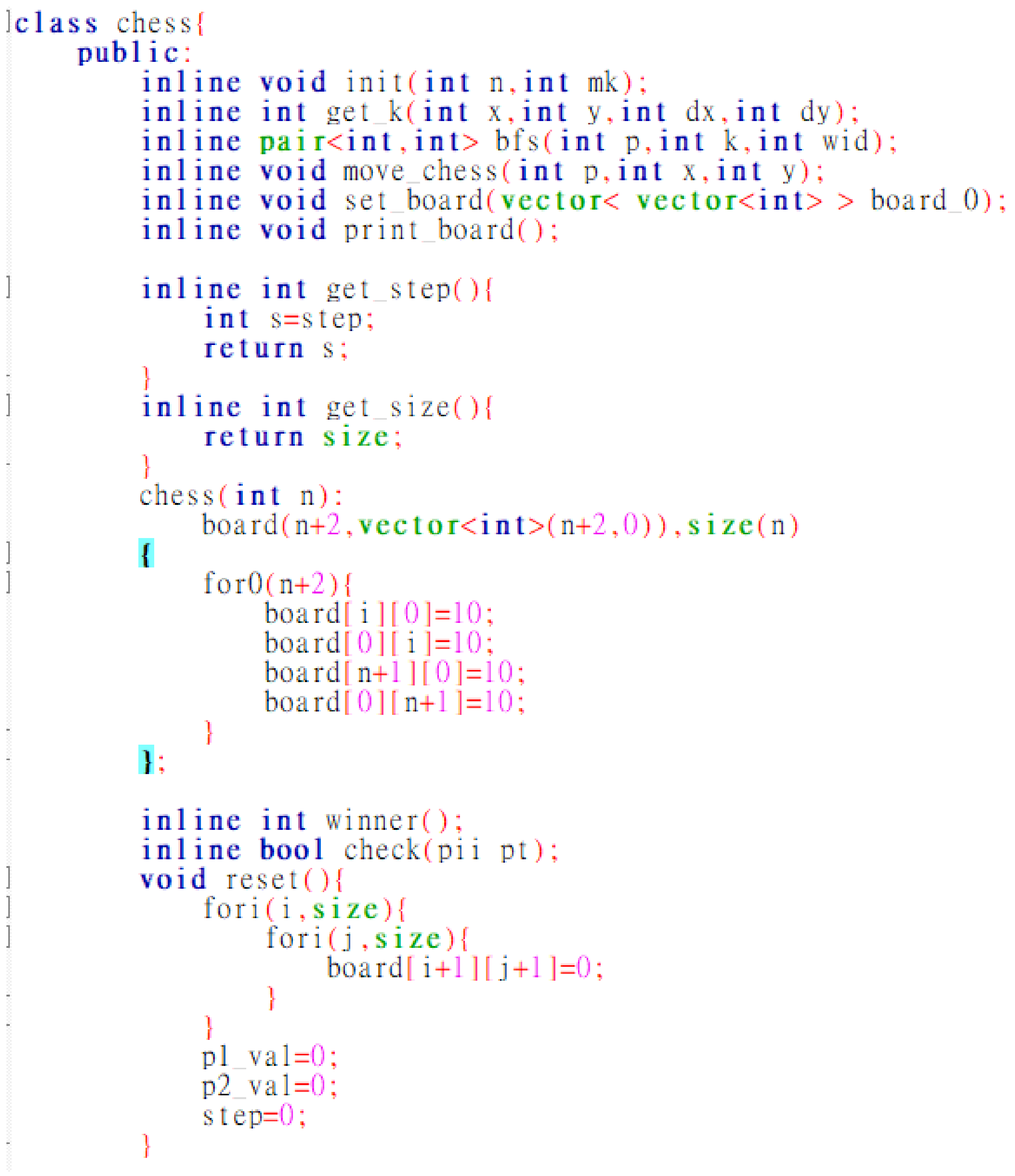
Alpha-beta Pruning

This algorithm maintains two values, α and β, which respectively represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of.

Initially, $\alpha = -\infty$ and $\beta = \infty$ (both players start with their worst possible score). Whenever the maximum score that the minimizing player (the β player) is assured of becomes less than the minimum score that the maximizing player (the α player) is assured of (i.e. $\beta < \alpha$), the maximizing player need not consider further descendants of this node, as they will never be reached in the actual play.

3 Key Objects and Functions

chess class public member



```
class chess{
    public:
        inline void init(int n,int mk);
        inline int get_k(int x,int y,int dx,int dy);
        inline pair<int,int> bfs(int p,int k,int wid);
        inline void move_chess(int p,int x,int y);
        inline void set_board(vector< vector<int> > board_0);
        inline void print_board();

        inline int get_step(){
            int s=step;
            return s;
        }
        inline int get_size(){
            return size;
        }
        chess(int n):
            board(n+2,vector<int>(n+2,0)),size(n)
        {
            for0(n+2){
                board[i][0]=10;
                board[0][i]=10;
                board[n+1][0]=10;
                board[0][n+1]=10;
            }
        };

        inline int winner();
        inline bool check(pii pt);
        void reset(){
            fori(i,size){
                fori(j,size){
                    board[i+1][j+1]=0;
                }
            }
            p1_val=0;
            p2_val=0;
            step=0;
        }
```

1. chess(n): constructor setting board size to be n×n

2. get_k(x,y,dx,dy):
   Return the maximal connected stones for a stone at coordinate (x,y) towards direction (dx,dy)
   Note that the maximal return is 5

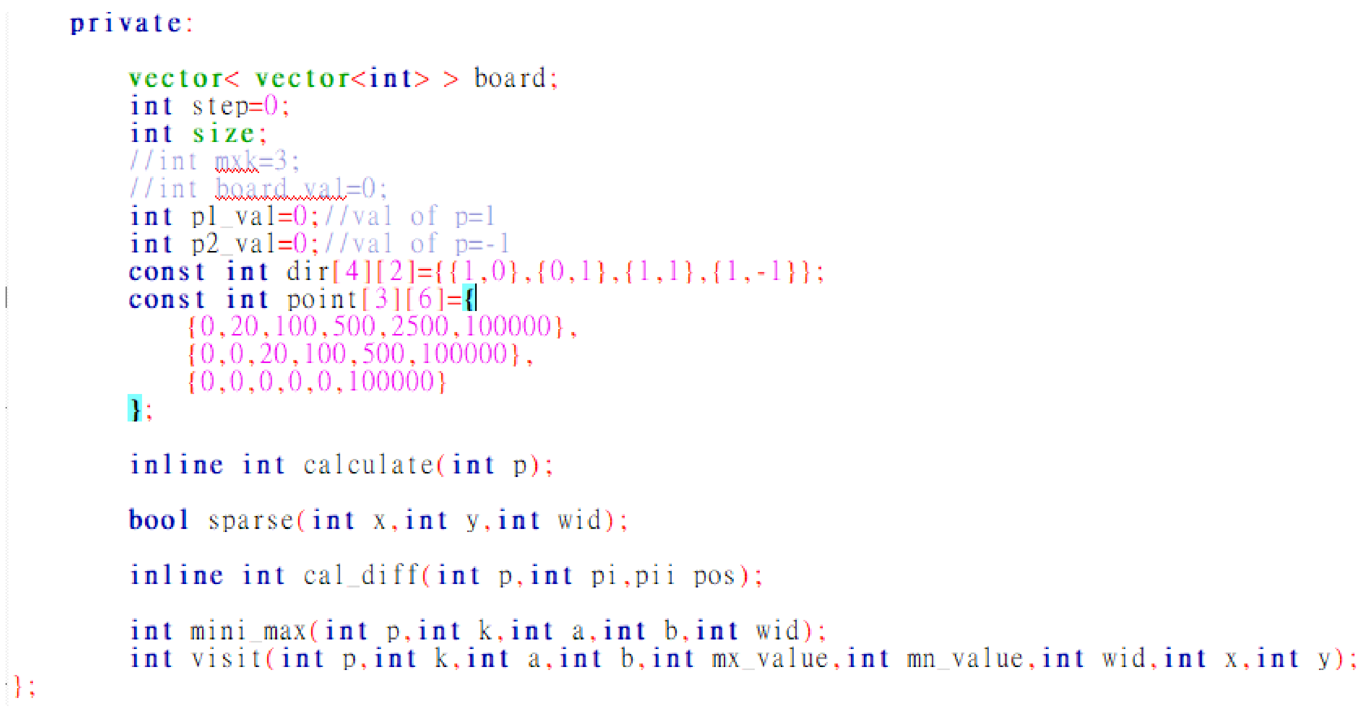
3. bfs(p): return the best choice of coordinate to land a stone for player p

4. move_chess(pi,x,y): player pi land a stone at coordinate (x,y)

5. set_board(board_0): initialize the board

6. print_board: print board
7. winner(): return winner (1 for black, -1 for white, 0 if game not yet finished)
8. get_step(): return current step (int)
9. check(pii pt): check if there is a stone at coordinate pt
10. reset(): reset board (start new game)

chess class private member

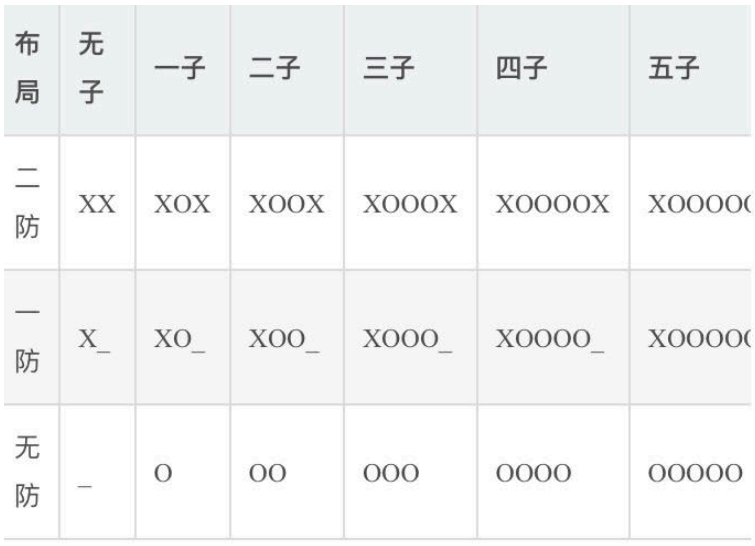


```
private:

    vector< vector<int> > board;
    int step=0;
    int size;
    //int mxk=3;
    //int board_val=0;
    int p1_val=0;//val of p=1
    int p2_val=0;//val of p=-1
    const int dir[4][2]={{1,0},{0,1},{1,1},{1,-1}};
    const int point[3][6]={
        {0,20,100,500,2500,100000},
        {0,0,20,100,500,100000},
        {0,0,0,0,0,100000}
    };

    inline int calculate(int p);

    bool sparse(int x,int y,int wid);

    inline int cal_diff(int p,int pi,pii pos);

    int mini_max(int p,int k,int a,int b,int wid);
    int visit(int p,int k,int a,int b,int mx_value,int mn_value,int wid,int x,int y);
};
```

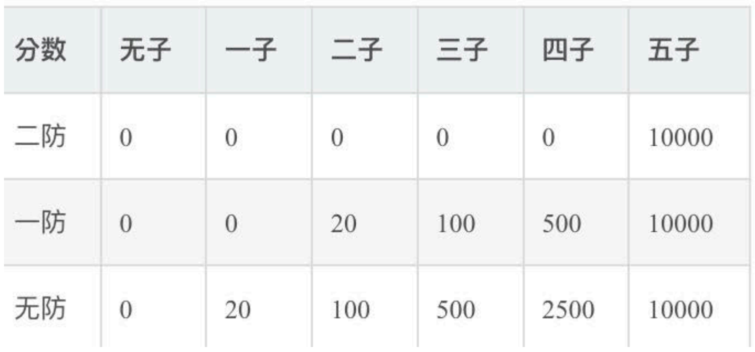
1. step: current step (int)
2. board: current board (vector<vector<int> where 1 for black, -1 for white , 0 for none and 10 for board boundary)
3. p1_val: score for black at current board (int)
4. p2_val: score for white at current board (int)
5. dir[4][2]: direction of 4 stones
6. point[3][6]: a chart of int indicating the score for a connected stone of length b with a stones blocking (see evaluation function below)
7. calculate(p): return score for player p
8. sparse(x,y): determine whether coordinate (x,y) is too far from the other stones
9. cal_diff(p,pi,pos): return the score variation for player p after player pi landing a stone at coordinate pos
10. mini_max(p,k): return board score for player p by expanding k layers of minimax algorithm
11. visit(): analyze coordinate (x,y) using minimax algorithm, which is constantly used in bfs()

Evaluation function

Reference: https://blog.csdn.net/jjwwwww/article/details/84593137

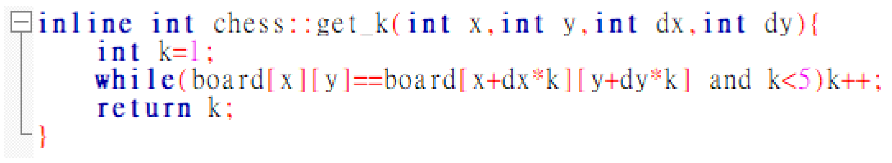


布局	无子	一子	二子	三子	四子	五子
二防	XX	XOX	XOOX	XOOOX	XOOOOX	XOOOO(
一防	X_	XO_	XOO_	XOOO_	XOOOO_	XOOOO(
无防	_	O	OO	OOO	OOOO	OOOOO

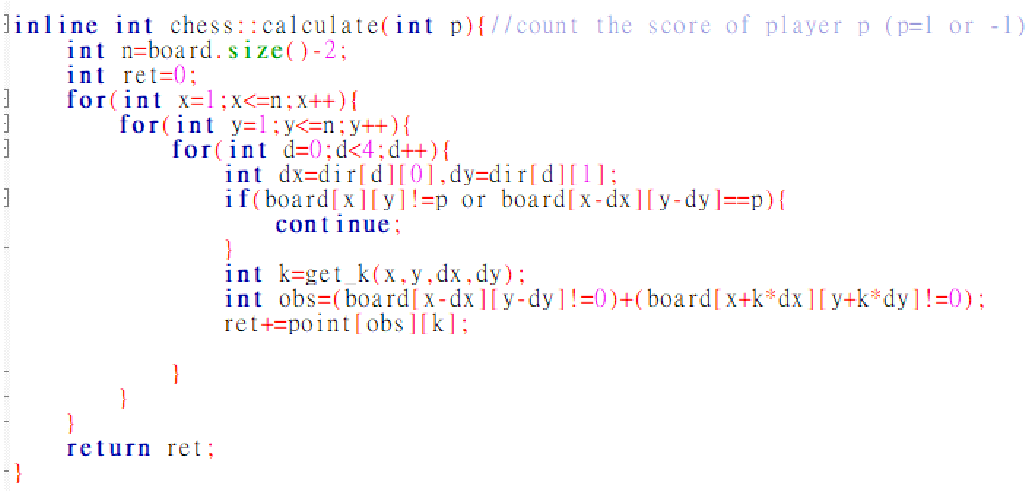


分数	无子	一子	二子	三子	四子	五子
二防	0	0	0	0	0	10000
一防	0	0	20	100	500	10000
无防	0	20	100	500	2500	10000

We use the get_k() function to inspect through all directions:

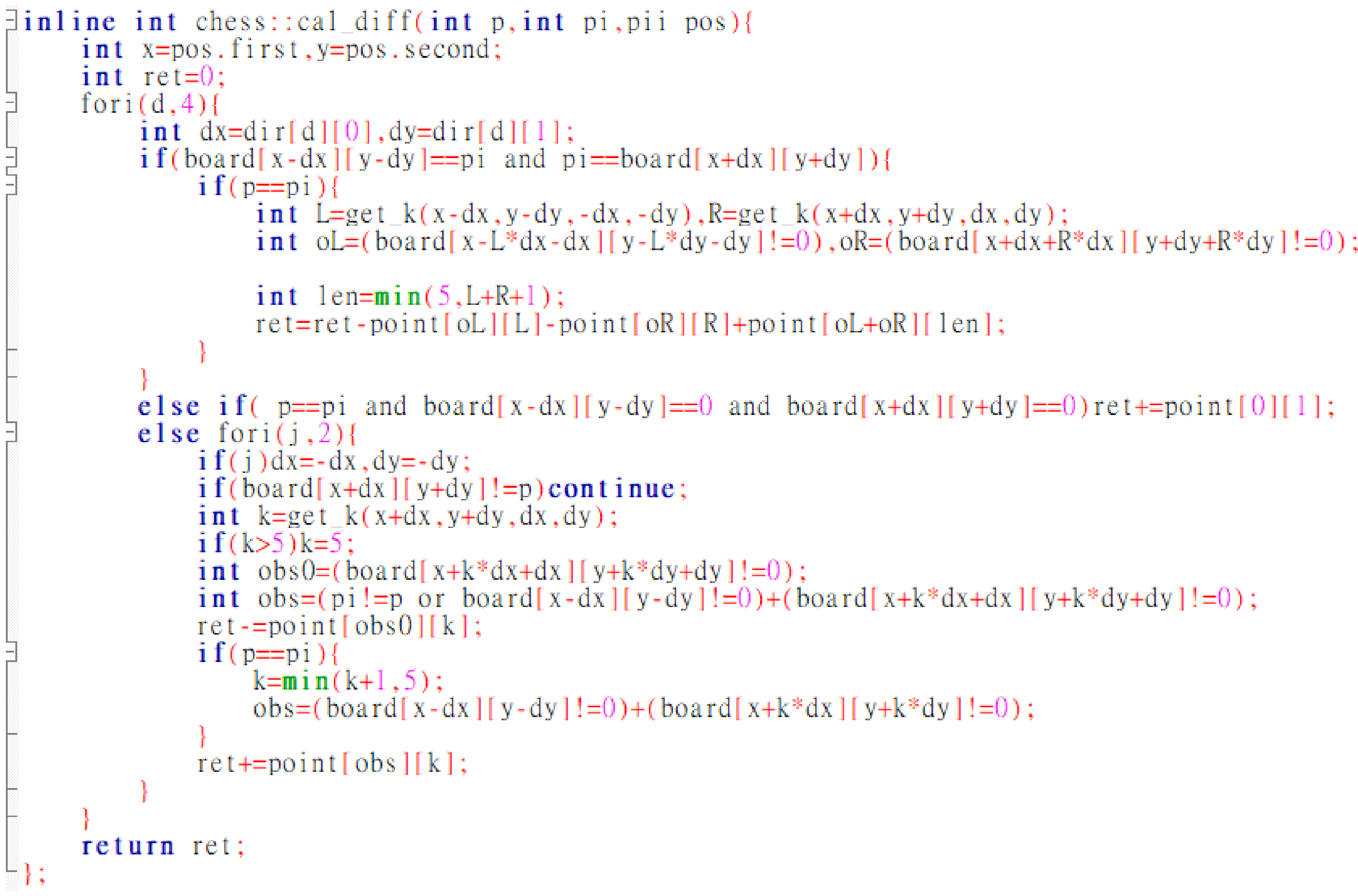


```
inline int chess::get_k(int x,int y,int dx,int dy){
    int k=1;
    while(board[x][y]==board[x+dx*k][y+dy*k] and k<5)k++;
    return k;
}
```



```
inline int chess::calculate(int p){//count the score of player p (p=1 or -1)
    int n=board.size()-2;
    int ret=0;
    for(int x=1;x<=n;x++){
        for(int y=1;y<=n;y++){
            for(int d=0;d<4;d++){
                int dx=dir[d][0],dy=dir[d][1];
                if(board[x][y]!=p or board[x-dx][y-dy]==p){
                    continue;
                }
                int k=get_k(x,y,dx,dy);
                int obs=(board[x-dx][y-dy]!=0)+(board[x+k*dx][y+k*dy]!=0);
                ret+=point[obs][k];

            }
        }
    }
    return ret;
}
```

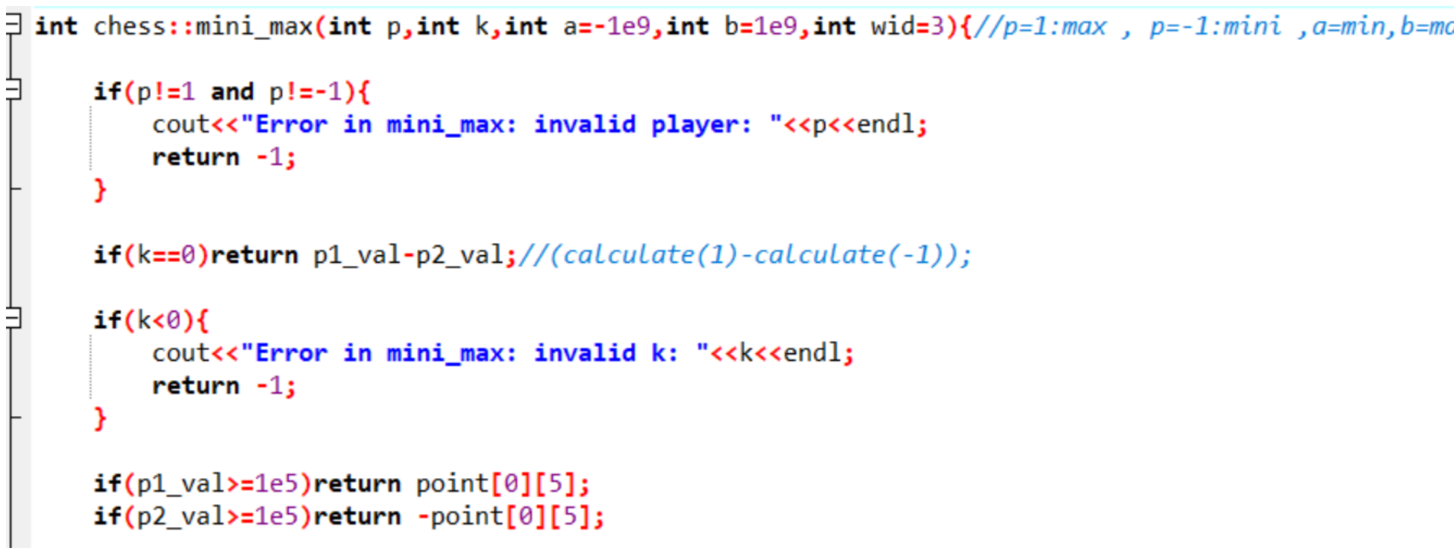
If two boards only differ by one move, we introduce the cal_diff() function to reduce computation:



```
inline int chess::cal_diff(int p,int pi,pii pos){
    int x=pos.first,y=pos.second;
    int ret=0;
    fori(d,4){
        int dx=dir[d][0],dy=dir[d][1];
        if(board[x-dx][y-dy]==pi and pi==board[x+dx][y+dy]){
            if(p==pi){
                int L=get_k(x-dx,y-dy,-dx,-dy),R=get_k(x+dx,y+dy,dx,dy);
                int oL=(board[x-L*dx-dx][y-L*dy-dy]!=0),oR=(board[x+dx+R*dx][y+dy+R*dy]!=0);

                int len=min(5,L+R+1);
                ret=ret-point[oL][L]-point[oR][R]+point[oL+oR][len];
            }
        }
        else if( p==pi and board[x-dx][y-dy]==0 and board[x+dx][y+dy]==0)ret+=point[0][1];
        else fori(j,2){
            if(j)dx=-dx,dy=-dy;
            if(board[x+dx][y+dy]!=p)continue;
            int k=get_k(x+dx,y+dy,dx,dy);
            if(k>5)k=5;
            int obs0=(board[x+k*dx+dx][y+k*dy+dy]!=0);
            int obs=(pi!=p or board[x-dx][y-dy]!=0)+(board[x+k*dx+dx][y+k*dy+dy]!=0);
            ret-=point[obs0][k];
            if(p==pi){
                k=min(k+1,5);
                obs=(board[x-dx][y-dy]!=0)+(board[x+k*dx][y+k*dy]!=0);
            }
            ret+=point[obs][k];
        }
    }
    return ret;
};
```

The minimax function

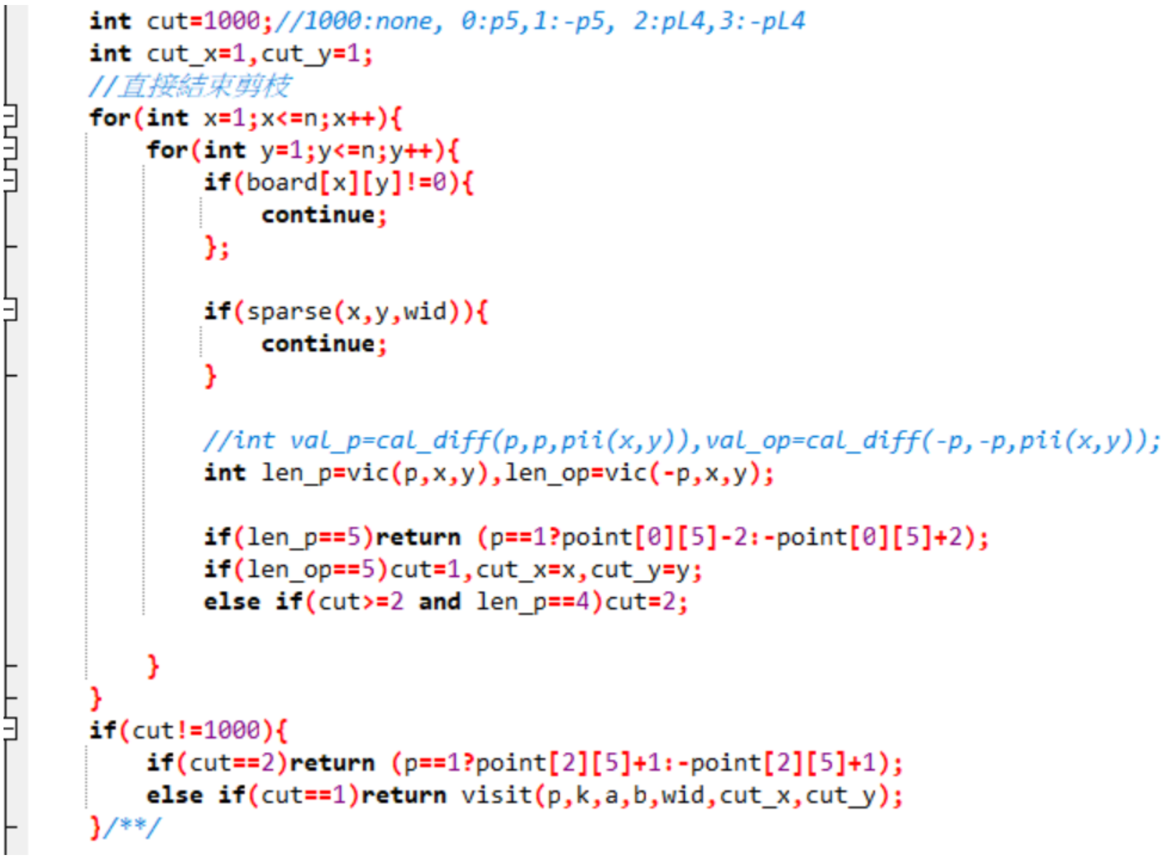
(a). < 終止狀態判斷 >



```
int chess::mini_max(int p,int k,int a=-1e9,int b=1e9,int wid=3){//p=1:max , p=-1:mini ,a=min,b=ma

    if(p!=1 and p!=-1){
        cout<<"Error in mini_max: invalid player: "<<p<<endl;
        return -1;
    }

    if(k==0)return p1_val-p2_val;//(calculate(1)-calculate(-1));

    if(k<0){
        cout<<"Error in mini_max: invalid k: "<<k<<endl;
        return -1;
    }

    if(p1_val>=1e5)return point[0][5];
    if(p2_val>=1e5)return -point[0][5];
```

若有一方獲勝，則結束遞迴呼叫並回傳當前盤面分數

(b). < 直接結束剪枝 (敵我 5、我方活 4 剪枝)>

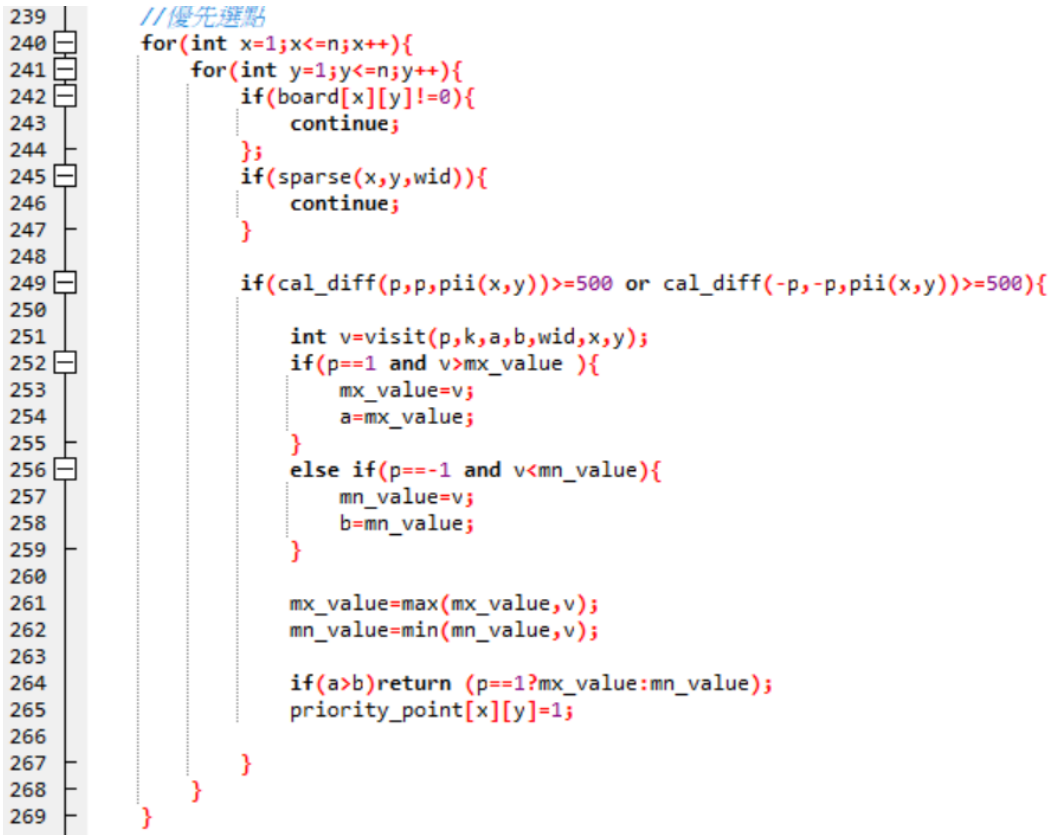


```
int cut=1000;//1000:none, 0:p5,1:-p5, 2:pL4,3:-pL4
int cut_x=1,cut_y=1;
//直接結束剪枝
for(int x=1;x<=n;x++){
    for(int y=1;y<=n;y++){
        if(board[x][y]!=0){
            continue;
        };

        if(sparse(x,y,wid)){
            continue;
        }

        //int val_p=cal_diff(p,p,pii(x,y)),val_op=cal_diff(-p,-p,pii(x,y));
        int len_p=vic(p,x,y),len_op=vic(-p,x,y);

        if(len_p==5)return (p==1?point[0][5]-2:-point[0][5]+2);
        if(len_op==5)cut=1,cut_x=x,cut_y=y;
        else if(cut>=2 and len_p==4)cut=2;

    }
}
if(cut!=1000){
    if(cut==2)return (p==1?point[2][5]+1:-point[2][5]+1);
    else if(cut==1)return visit(p,k,a,b,wid,cut_x,cut_y);
}/**/
```

變數 cut 代表目前剪枝狀態，1000 時代表沒發生，1 時代表敵方有連 5 點，2 代表敵方無連 5 點且我方有連活 4 點 (我方可連 5 時會在發現的那一刻直接回傳，因此不用 cut 標記)

搜索完所有點後，若 cut 不爲 1000，則根據其種類回傳該點數值。

(c). < 優先選點 (若 a,b 都沒發生才會執行 c,d,e)>

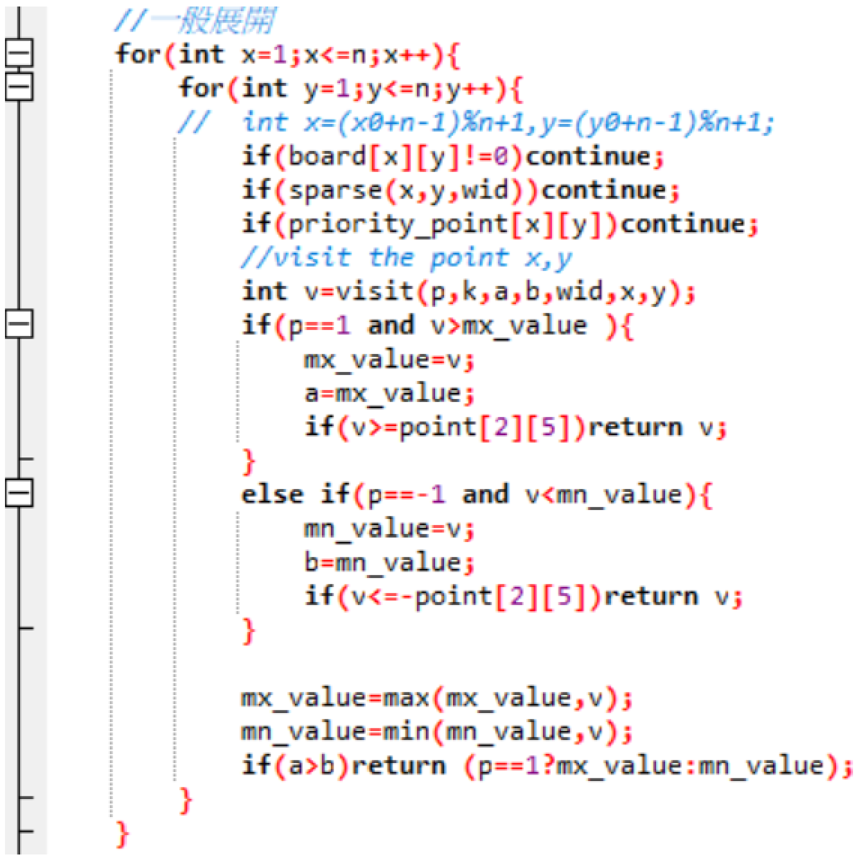


```
//優先選點
for(int x=1;x<=n;x++){
    for(int y=1;y<=n;y++){
        if(board[x][y]!=0){
            continue;
        };
        if(sparse(x,y,wid)){
            continue;
        }

        if(cal_diff(p,p,pii(x,y))>=500 or cal_diff(-p,-p,pii(x,y))>=500){

            int v=visit(p,k,a,b,wid,x,y);
            if(p==1 and v>mx_value ){
                mx_value=v;
                a=mx_value;
            }
            else if(p==-1 and v<mn_value){
                mn_value=v;
                b=mn_value;
            }

            mx_value=max(mx_value,v);
            mn_value=min(mn_value,v);

            if(a>b)return (p==1?mx_value:mn_value);
            priority_point[x][y]=1;

        }
    }
}
```

c 與 d 做的是相同的 minimax 展開，差別只在 c 是優先挑出”有可能較好的點”展開，也就是使用 cal_diff 對該點計算後，發現下在該點盤面分數較高的點。

當展開過較好的點，在展開較差的點時就能愈快將其淘汰，因此可加速程式

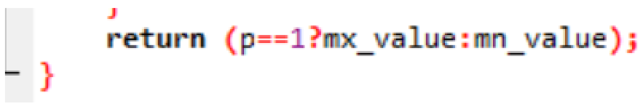
(d). < 一般展開 >



```
//一般展開
for(int x=1;x<=n;x++){
    for(int y=1;y<=n;y++){
    //  int x=(x0+n-1)%n+1,y=(y0+n-1)%n+1;
        if(board[x][y]!=0)continue;
        if(sparse(x,y,wid))continue;
        if(priority_point[x][y])continue;
        //visit the point x,y
        int v=visit(p,k,a,b,wid,x,y);
        if(p==1 and v>mx_value ){
            mx_value=v;
            a=mx_value;
            if(v>=point[2][5])return v;
        }
        else if(p==-1 and v<mn_value){
            mn_value=v;
            b=mn_value;
            if(v<=-point[2][5])return v;
        }

        mx_value=max(mx_value,v);
        mn_value=min(mn_value,v);
        if(a>b)return (p==1?mx_value:mn_value);
    }
}
```

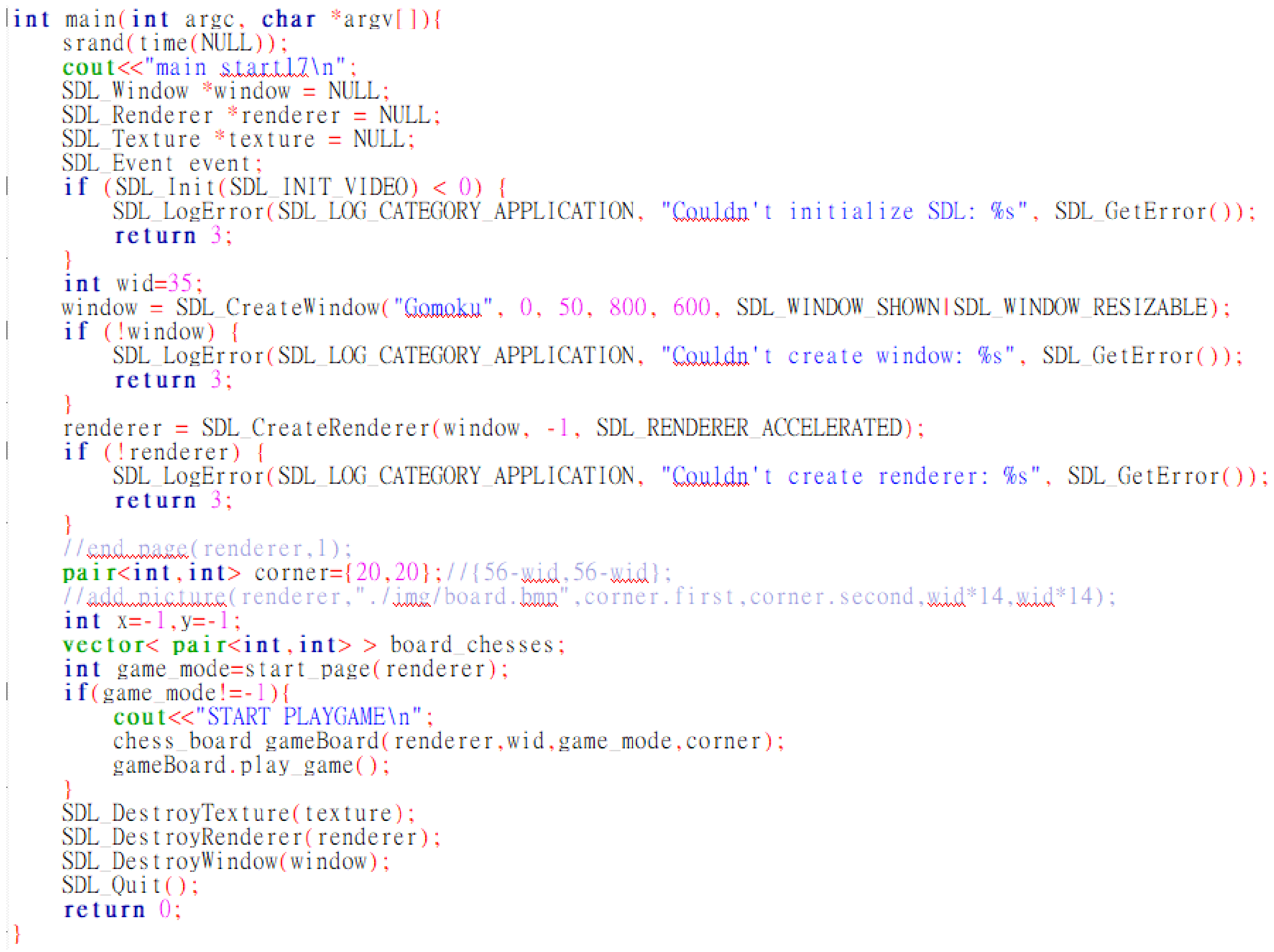
對沒有在 c 展開的點展開

(e). < 回傳 >



```
    return (p==1?mx_value:mn_value);
}
```

The main function



```
int main(int argc, char *argv[]){
    srand(time(NULL));
    cout<<"main start17\n";
    SDL_Window *window = NULL;
    SDL_Renderer *renderer = NULL;
    SDL_Texture *texture = NULL;
    SDL_Event event;
    if (SDL_Init(SDL_INIT_VIDEO) < 0) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't initialize SDL: %s", SDL_GetError());
        return 3;
    }
    int wid=35;
    window = SDL_CreateWindow("Gomoku", 0, 50, 800, 600, SDL_WINDOW_SHOWN|SDL_WINDOW_RESIZABLE);
    if (!window) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't create window: %s", SDL_GetError());
        return 3;
    }
    renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
    if (!renderer) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't create renderer: %s", SDL_GetError());
        return 3;
    }
    //end_page(renderer,1);
    pair<int,int> corner={20,20};//{56-wid,56-wid};
    //add_picture(renderer,"./img/board.bmp",corner.first,corner.second,wid*14,wid*14);
    int x=-1,y=-1;
    vector< pair<int,int> > board_chesses;
    int game_mode=start_page(renderer);
    if(game_mode!=-1){
        cout<<"START PLAYGAME\n";
        chess_board gameBoard(renderer,wid,game_mode,corner);
        gameBoard.play_game();
    }
    SDL_DestroyTexture(texture);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
    return 0;
}
```

4 Program Implementation

After opening the Gomoku.exe file, the user first clicks the [與電腦對戰] or [雙人對戰] button regarding either he/she wants to play with the computer or another player. If the player choose the first mode, he/she can then choose which (black/white) stone to play. If the second mode is chosen, the first player to move is defaulted to be the black side.

During game, the location of the cursor on board is shadowed so as to preview where the stone is to land. Also, the last move on board is marked for the player to read more easily.

After game, the user can choose whether to start a new game by clicking [Yes] or [No] button. If the former is chosen, the user can also pick which game mode to restart.

5 Group Members and Work Distribution

B11201024 蔡平樂: Core codes and SDL library
B08201024 張力仁: Proposal, video demo and PPT presentation
B08202062 廖崧蒼: Picture material of Objects and report