五子棋

靓仔语塞。jpg
林天皓 龙行健 陈逸铭 廖洋 高远 艾刚正

2019 年 6 月 30 日

目录

1 简介

本程序实现了五子棋人人对战，人机对战的功能，支持悔棋，切换皮肤，回合数显示，记录走棋过程，棋局复盘等功能。

主界面如下：

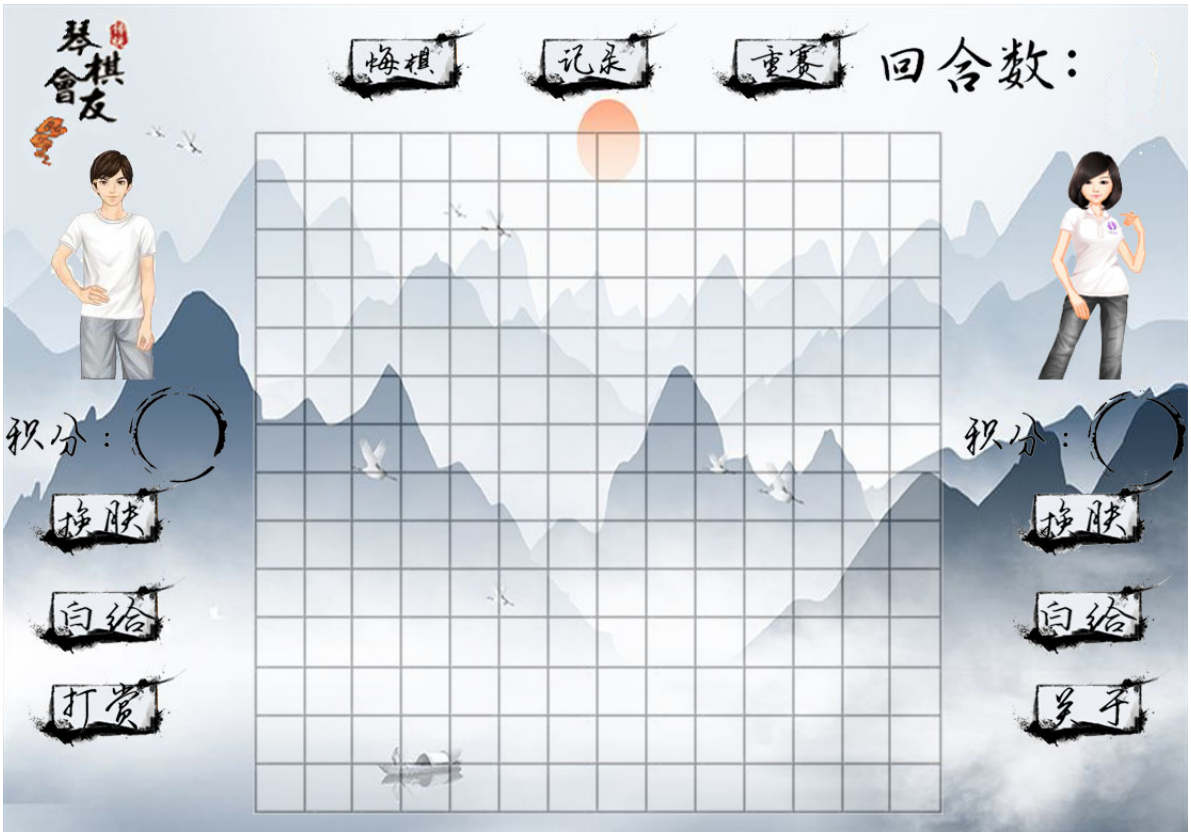

图 1: 棋盘

2 程序架构

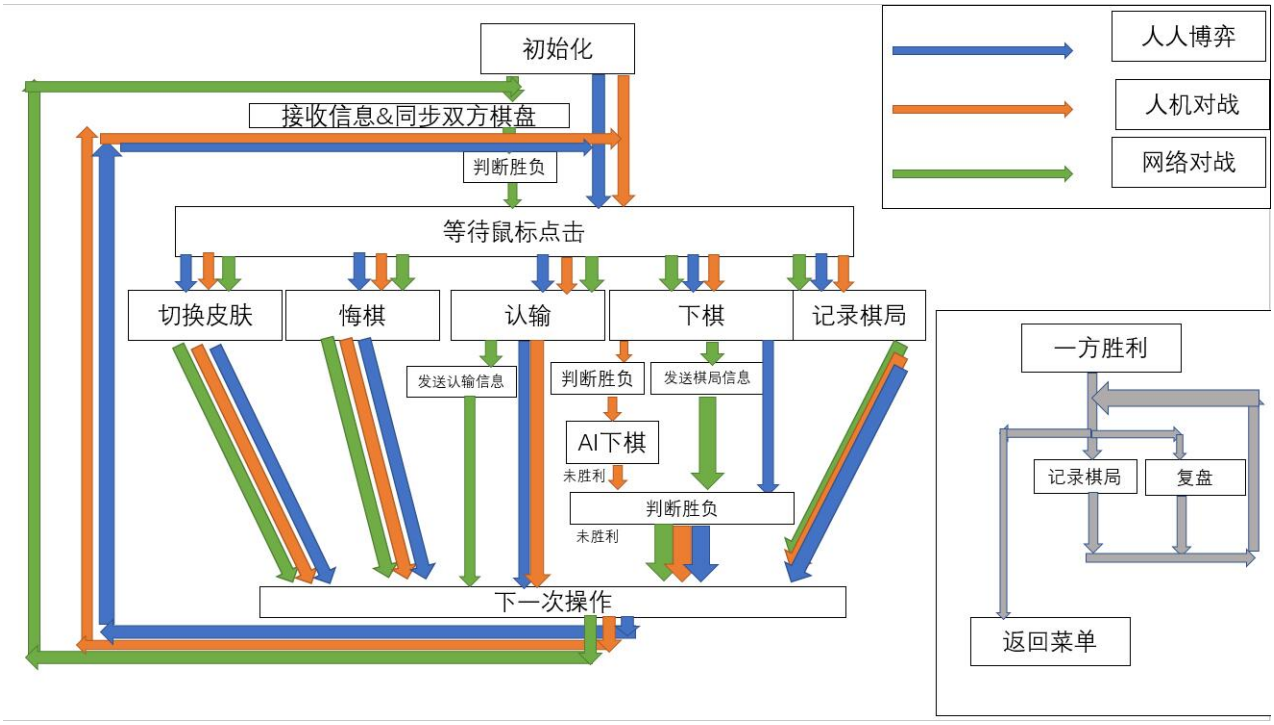

图 2: 整体架构流程

3 程序函数介绍

3.1 主函数

```
int main() {
    load_image();
    game_init();
    play();
}
```

图 3: 2

3.2 载入图片

```
void load_image() {
    loadimage(&pic_about, L"img\\about.jpg");
    loadimage(&pic_zan, L"img\\dashang.jpg");
    loadimage(&pic_chess_board, L"img\\qipan.jpg", 1110, 774);
    loadimage(&menu, L"img\\caidan.jpg");
    loadimage(&pic_skin_change, L"img\\pifuxuanze.jpg");
    loadimage(&skin_mask[0], L"img\\jian.jpg", chess_size, chess_size);
    loadimage(&skin[0], L"img\\jianjian.jpg", chess_size, chess_size);
    loadimage(&skin_mask[1], L"img\\kun.jpg", chess_size, chess_size);
    loadimage(&skin[1], L"img\\kunkun.jpg", chess_size, chess_size);
    loadimage(&skin_mask[2], L"img\\xiongmao1.jpg", chess_size, chess_size);
    loadimage(&skin[2], L"img\\xiongmao.jpg", chess_size, chess_size);
    loadimage(&skin_mask[3], L"img\\leishen1.jpg", chess_size, chess_size);
    loadimage(&skin[3], L"img\\leishen.jpg", chess_size, chess_size);
    loadimage(&skin_mask[4], L"img\\zongzi1.jpg", chess_size, chess_size);
    loadimage(&skin[4], L"img\\zongzi.jpg", chess_size, chess_size);
    loadimage(&skin_mask[5], L"img\\angel1.jpg", chess_size, chess_size);
    loadimage(&skin[5], L"img\\angel.jpg", chess_size, chess_size);
    loadimage(&skin_mask_icon[0], L"img\\jian.jpg", chess_icon_size, chess_icon_size);
    loadimage(&skin_icon[0], L"img\\jianjian.jpg", chess_icon_size, chess_icon_size);
    loadimage(&skin_mask_icon[1], L"img\\kun.jpg", chess_icon_size, chess_icon_size);
    loadimage(&skin_icon[1], L"img\\kunkun.jpg", chess_icon_size, chess_icon_size);
    loadimage(&skin_mask_icon[2], L"img\\xiongmao1.jpg", chess_icon_size, chess_icon_size);
    loadimage(&skin_icon[2], L"img\\xiongmao.jpg", chess_icon_size, chess_icon_size);
    loadimage(&skin_mask_icon[3], L"img\\leishen1.jpg", chess_icon_size, chess_icon_size);
    loadimage(&skin_icon[3], L"img\\leishen.jpg", chess_icon_size, chess_icon_size);
    loadimage(&skin_mask_icon[4], L"img\\zongzi1.jpg", chess_icon_size, chess_icon_size);
    loadimage(&skin_icon[4], L"img\\zongzi.jpg", chess_icon_size, chess_icon_size);
    loadimage(&skin_mask_icon[5], L"img\\angel1.jpg", chess_icon_size, chess_icon_size);
    loadimage(&skin_icon[5], L"img\\angel.jpg", chess_icon_size, chess_icon_size);
    loadimage(&white_win, L"img\\whitewin.jpg");
    loadimage(&black_win, L"img\\blackwin.jpg");
```

图 4: 3

3.3 鼠标按钮判断

```
bool in_range_vs_human(MOUSEMSG m_mouse){
    //判断是否选择人人对战
    if(m_mouse.x>=VS_HUMAN_LEFT && m_mouse.x<=VS_HUMAN_RIGHT && m_mouse.y>=VS_HUMAN_UP && m_mouse.y<=VS_HUMAN_DOWN){
        return true;
    }
    else return false;
}
bool in_range_vs_ai(MOUSEMSG m_mouse){
    //判断是否选择人机对战
    if(m_mouse.x>=VS_AI_LEFT && m_mouse.x<=VS_AI_RIGHT && m_mouse.y>=VS_AI_UP && m_mouse.y<=VS_AI_DOWN){
        return true;
    }
    else return false;
}
bool in_range_black_give_up(MOUSEMSG m_mouse){
    //判断是否在黑棋认输范围之内
    if(m_mouse.x>=BLACK_GIVE_UP_LEFT && m_mouse.x<=BLACK_GIVE_UP_RIGHT && m_mouse.y>=BLACK_GIVE_UP_UP && m_mouse.y<=BLACK_GIVE_UP_DOWN){
        return true;
    }
    else return false;
}
```

图 5: 4

3.4 初始化

```
void game_init(){
    //棋盘置零，棋局记录清空，双方使用0号皮肤，双方金钱清零
    initgraph(1110, 774);
    memset(chess_board,0,sizeof(chess_board));
    game_record.clear();
    play_side_now=BLACK_SIDE;
    chose_rival();
    white_using_skin_num = 5;
    black_using_skin_num = 0;
    white_money=0;
    black_money = 0;
    rounds = 0;
    draw_chess_board(chess_board);
}
```

图 6: 5

3.5 选择人人对战或者人机对战

```
void chose_rival(){
    //进入菜单选择对手
    int flag = 1, k = 0;
    putimage(0,0,&menu);
    while(flag){
        m_mouse = GetMouseMsg();
        switch (m_mouse.uMsg) {
            case WM_LBUTTONDOWN:{
                if(in_range_vs_human(m_mouse)){
                    rival_now=HUMAN;
                    flag=0;
                }
                else if(in_range_vs_ai(m_mouse)){
                    rival_now=AI;
                    flag=0;
                }
            }
        }
    }
}
```

图 7: 6

3.6 游戏中内容

```
1106  void play() {
1107      while (1) {
1108          m_mouse = GetMouseMsg();
1109          switch (m_mouse.uMsg) {
1110          case WM_LBUTTONDOWN: {
1111              if (in_range_set_chess(m_mouse)) {
1112                  set_chess(m_mouse, play_side_now);
1113                  check_win();
1114                  if (rival_now == AI && play_side_now == WHITE_SIDE)
1115                      ai_set_chess();
1116                      check_win();
1117                  }
1118              }
1119              else if (in_range_withdraw(m_mouse)) {
1120                  withdraw();
1121                  if (rival_now == AI) {
1122                      withdraw();
1123                  }
1124              }
1125              else if (in_range_record(m_mouse)) {
1126                  save_game_record();
1127              }
1128              else if (in_range_white_change_skin(m_mouse)) {
1129                  white_skin_change();
1130              }
1131              else if (in_range_black_change_skin(m_mouse)) {
1132                  black_skin_change();
1133              }
1134              else if (in_range_reset(m_mouse)) {
1135                  game_reset();
1136              }
1137              else if(in_range_white_give_up(m_mouse)){
1138                  win(BLACK_SIDE);
1139              }
1140              else if(in_range_black_give_up(m_mouse)){
1141                  win(WHITE_SIDE);
1142              }
1143              else if(in_range_zan(m_mouse)){
1144                  zan();
1145              }
1146              else if(in_range_about(m_mouse)){
1147                  about();
1148              }
1149              //这里以后加更多按钮的功能
1150          }
```

3.7 下棋

```
void set_chess(MOUSEMSG m_mouse, int& play_side_now) {
    //根据鼠标位置和现在是哪一方在下棋来落子
    point temp;
    temp.x = max(0, (m_mouse.x - 234 + 23) / 46);
    temp.x = min(temp.x, 14);
    temp.y = max(0, (m_mouse.y - 113 + 23) / 46);
    temp.y = min(temp.y, 14);
    if (chess_board[temp.x][temp.y] == 0) {
        game_record.push_back(temp);
        rounds++;
        switch (play_side_now) {
        case WHITE_SIDE:
        {
            chess_board[temp.x][temp.y] = -1;
            play_side_now = BLACK_SIDE;
            break;
        }
        case BLACK_SIDE:
        {
            chess_board[temp.x][temp.y] = 1;
            play_side_now = WHITE_SIDE;
            break;
        }
        }
        draw_chess_board(chess_board);
    }
}
```

图 9: 8

3.8 悔棋

```
void withdraw() {
    //悔棋
    if (game_record.size() > 0) {
        point last_point = game_record.back();
        game_record.pop_back();
        chess_board[last_point.x][last_point.y] = 0;
        rounds--;
        switch (play_side_now) {
            case WHITE_SIDE:
            {
                play_side_now = BLACK_SIDE;
                break;
            }
            case BLACK_SIDE:
            {
                play_side_now = WHITE_SIDE;
                break;
            }
        }
        draw_chess_board(chess_board);
    }
}
```

图 10: 9

3.9 暂停，按鼠标左键继续

```
void wait_mouse_click() {
    int flag = 1;
    while (flag) {
        m_mouse = GetMouseMsg();
        switch (m_mouse.uMsg) {
        case WM_LBUTTONDOWN: {
            flag = 0;
        }
        }
    }
}
```

图 11: 10

3.10 画棋盘

```
void draw_chess_board(int chess_board[][15]) {
    //画棋盘，字面意思
    setbkcolor(RGB(245, 211, 155));
    cleardevice();
    putimage(0, 0, &pic_chess_board);
    points(WHITE_SIDE, white_money);
    points(BLACK_SIDE, black_money);
    round(rounds);
    for (int i = 0; i < 15; i++)
    {
        for (int j = 0; j < 15; j++) {
            if (chess_board[i][j] == 1) {
                putimage(234 - 23 + 46 * i, 113 - 23 + 46 * j, &skin_mask[black_using_skin_num], NOTSRCERASE);
                putimage(234 - 23 + 46 * i, 113 - 23 + 46 * j, &skin[black_using_skin_num], SRCINVERT);

            }
            else if (chess_board[i][j] == -1) {
                putimage(234 - 23 + 46 * i, 113 - 23 + 46 * j, &skin_mask[white_using_skin_num], NOTSRCERASE);
                putimage(234 - 23 + 46 * i, 113 - 23 + 46 * j, &skin[white_using_skin_num], SRCINVERT);
            }
        }
    }
}
```

图 12: 11

3.11 判断是否五连

```
int is_five(int row, int column, int which_side) {
    int i, j, k = 0, p[4] = { 0,-1,-1,-1 }, q[4] = { 1,1,0,-1, }, chess_count = 0, z = 0;
    if (which_side == WHITE_SIDE) {
        z = -1;
    }
    else if (which_side == BLACK_SIDE) {
        z = 1;
    }
    for (i = 0; i < 4; i++)
    {
        for (j = -4; j <= 4 && (row + p[i] * j) < 15 && (row + p[i] * j) >= 0 && (column + q[i] * j) < 15 && (column + q[i] * j) >= 0; j++)
        {
            if (chess_board[row + p[i] * j][column + q[i] * j] != z)chess_count = 0;
            else chess_count++;
            if (chess_count == 5)k = 1;
        }
    }
    return k;
}
```

图 13: 12

3.12 是否有一方胜利

```
bool is_one_side_win(int which_side) {
    if (which_side == WHITE_SIDE) {
        for (int i = 0; i < 15; i++) {
            for (int j = 0; j < 15; j++) {
                if (is_five(i, j, WHITE_SIDE)) {
                    return true;
                }
            }
        }
    }
    if (which_side == BLACK_SIDE) {
        for (int i = 0; i < 15; i++) {
            for (int j = 0; j < 15; j++) {
                if (is_five(i, j, BLACK_SIDE)) {
                    return true;
                }
            }
        }
    }
    return false;
}
```

图 14: 13

3.13 重来

```
void game_reset() {
    memset(chess_board, 0, sizeof(chess_board));
    rounds = 0;
    game_record.clear();
    play_side_now = BLACK_SIDE;
    draw_chess_board(chess_board);
}
```

图 15: 14

3.14 复盘

```
void game_replay() {
    in_replay = 1;
    memset(chess_board, 0, sizeof(chess_board));
    play_side_now = BLACK_SIDE;
    rounds = 0;
    for (int i = 0; i<int(game_record.size()); i++) {

        switch (play_side_now) {
        case WHITE_SIDE:
        {
            chess_board[game_record[i].x][game_record[i].y] = -1;
            play_side_now = BLACK_SIDE;
            break;
        }
        case BLACK_SIDE:
        {
            chess_board[game_record[i].x][game_record[i].y] = 1;
            play_side_now = WHITE_SIDE;
            break;
        }
        }
        rounds++;
        draw_chess_board(chess_board);
        Sleep(500);
    }
    wait_mouse_click();
    if (is_one_side_win(WHITE_SIDE)) {
        win(WHITE_SIDE);
    }
    else if (is_one_side_win(BLACK_SIDE)) {
        win(BLACK_SIDE);
    }
    in_replay = 0;
}
```

图 16: 15

3.15 胜利后的画面

```
void after_win_action() {
    now_page = WIN_PAGE;
    int flag = 1;
    while (flag) {
        m_mouse = GetMouseMsg();
        switch (m_mouse.uMsg) {
        case WM_LBUTTONDOWN: {
            if (in_range_return(m_mouse)) {
                chose_rival();
                flag = 0;
            }
            else if (in_range_record_end(m_mouse)) {
                save_game_record();
            }
            else if (in_range_replay(m_mouse)) {
                flag = 0;
                game_replay();
            }
        }
        }
    }
}
```

图 17: 16

3.16 画积分数

```
void points(int which_side_now, int value) {
    int num1, num2;
    num2 = value % 10;
    num1 = value / 10;
    if (which_side_now == BLACK_SIDE) {
        putimage(130, 380, &numbers_mask_s[num1], NOTSRCERASE);
        putimage(130, 380, &numbers_s[num1], SRCINVERT);
        putimage(166, 380, &numbers_mask_s[num2], NOTSRCERASE);
        putimage(166, 380, &numbers_s[num2], SRCINVERT);
    }
    else if (which_side_now == WHITE_SIDE) {
        putimage(1029, 380, &numbers_mask_s[num1], NOTSRCERASE);
        putimage(1029, 380, &numbers_s[num1], SRCINVERT);
        putimage(1060, 380, &numbers_mask_s[num2], NOTSRCERASE);
        putimage(1060, 380, &numbers_s[num2], SRCINVERT);
    }
}
```

图 18: 17

3.17 画回合数

```
void round(int value) {
    int num1, num2;
    num2 = value % 10;
    num1 = value / 10;
    putimage(1014, 20, &numbers_mask_b[num1], NOTSRCERASE);
    putimage(1014, 20, &numbers_b[num1], SRCINVERT);
    putimage(1060, 20, &numbers_mask_b[num2], NOTSRCERASE);
    putimage(1060, 20, &numbers_b[num2], SRCINVERT);
}
```

图 19: 18

3.18 一方胜利后的动作

```
void win(int which_side) {
    if (which_side == WHITE_SIDE) {
        if(!in_replay){
            white_money += 5;
        }
        putimage(0, 0, &white_win);
        after_win_action();
        game_reset();
    }
    else if (which_side == BLACK_SIDE) {
        if(!in_replay){
            black_money += 5;
        }
        putimage(0, 0, &black_win);
        after_win_action();
        game_reset();
    }

}
```

图 20: 19

3.19 保存下棋记录

```
void save_game_record() {
    FILE* fp = fopen("game_record.txt", "w");
    if (fp == NULL)printf("open file failed£¡\n");
    else {
        int i = 1;
        for (auto iter = game_record.begin(); iter != game_record.end(); ++iter) {
            fprintf(fp, "step:%d row=%d column=%d\n", i++, (*iter).x, (*iter).y);
        }
    }
    if (fp != NULL) {
        fclose(fp);
    }
    putimage(500, 350, &record_success);
    wait_mouse_click();
    if (now_page == GAME_PAGE) {
        draw_chess_board(chess_board);
    }
    else if (now_page == WIN_PAGE) {
        if (is_one_side_win(WHITE_SIDE)) {
            putimage(0, 0, &white_win);
        }
        else if (is_one_side_win(BLACK_SIDE)) {
            putimage(0, 0, &black_win);
        }
    }
}
```

图 21: 20

3.20 展示皮肤

```
void show_skin() {
    putimage(pic_x, pic_y, &pic_skin_change);
    putimage(pic_x + 38, pic_y + 47, &skin_mask_icon[0], NOTSRCERASE);
    putimage(pic_x + 38, pic_y + 47, &skin_icon[0], SRCINVERT);
    putimage(pic_x + 189, pic_y + 47, &skin_mask_icon[1], NOTSRCERASE);
    putimage(pic_x + 189, pic_y + 47, &skin_icon[1], SRCINVERT);
    putimage(pic_x + 337, pic_y + 47, &skin_mask_icon[2], NOTSRCERASE);
    putimage(pic_x + 337, pic_y + 47, &skin_icon[2], SRCINVERT);
    putimage(pic_x + 38, pic_y + 182, &skin_mask_icon[3], NOTSRCERASE);
    putimage(pic_x + 38, pic_y + 182, &skin_icon[3], SRCINVERT);
    putimage(pic_x + 187, pic_y + 182, &skin_mask_icon[4], NOTSRCERASE);
    putimage(pic_x + 187, pic_y + 182, &skin_icon[4], SRCINVERT);
    putimage(pic_x + 343, pic_y + 182, &skin_mask_icon[5], NOTSRCERASE);
    putimage(pic_x + 343, pic_y + 182, &skin_icon[5], SRCINVERT);
}
```

图 22: 21

3.21 选择皮肤

```
int chose_skin() {
    //不知道为什么要点很多次才有作用
    int flag = 1, k = 0;
    while (flag) {
        m_mouse = GetMouseMsg();
            switch (m_mouse.uMsg) {
            case WM_LBUTTONDOWN: {
                if (in_range_change_skin_1(m_mouse)) {
                    k = 0;
                    flag = 0;
                }
                else if (in_range_change_skin_2(m_mouse)) {
                    k = 1;
                    flag = 0;
                }
                else if (in_range_change_skin_3(m_mouse)) {
                    k = 2;
                    flag = 0;
                }
                else if (in_range_change_skin_4(m_mouse)) {
                    k = 3;
                    flag = 0;
                }
                else if (in_range_change_skin_5(m_mouse)) {
                    k = 4;
                    flag = 0;
                }
                else if (in_range_change_skin_6(m_mouse)) {
                    k = 5;
                    flag = 0;
                }
                else if(in_range_close(m_mouse)){
                    flag = 0;
                }
            }
        }
    }
    return k;
}
```

图 23: 22

3.22 白色方换皮肤

```
void white_skin_change() {
    show_skin();
    int new_skin = chose_skin();
    if (new_skin != black_using_skin_num) {
        white_using_skin_num = new_skin;
    }
    draw_chess_board(chess_board);
}
```

图 24: 23

3.23 黑色方换皮肤

```
void black_skin_change() {
    show_skin();
    int new_skin = chose_skin();
    if (new_skin != white_using_skin_num) {
        black_using_skin_num = new_skin;
    }
    draw_chess_board(chess_board);
}
```

图 25: 24

3.24 判断两头周围是否有棋子

```
int Around(int x, int y) // 空子只算旁边有子的
{
    int i, j;
    for (i = (x - 2 > 0 ? x - 2 : 0); i <= x + 2 && i < 15; i++)
        for (j = (y - 2 > 0 ? y - 2 : 0); j <= y + 2 && j < 15; j++)
            if (Map[i][j] != EMPTY) return 1;
    return 0;
}
```

图 26: 25

3.25 输入几联，输出得分

```
/*
    按照成五 100000、活四 10000、活三 1000、活二 100、活一 10、死四 1000、死三 100、死二 10 的规则给棋盘上的所有棋子打分，
    然后把所有 AI 和对手棋子的得分分别相加，ScoreBLACK_CHESS_ 为 AI 得分，ScoreWHITE_CHESS 为对手得分，
    ScoreBLACK_CHESS_ - ScoreWHITE_CHESS 即为当前局势的总分数
*/

int ScoreTable(int Number, int Empty) // 计分板
{
    if (Number >= 5) return 100000;
    else if (Number == 4)
    {
        if (Empty == 2) return 10000;
        else if (Empty == 1) return 1000;
    }
    else if (Number == 3)
    {
        if (Empty == 2) return 1000;
        else if (Empty == 1) return 100;
    }
    else if (Number == 2)
    {
        if (Empty == 2) return 100;
        else if (Empty == 1) return 10;
    }
    else if (Number == 1 && Empty == 2) return 10;
    return 0;
}
```

图 27: 26

3.26 统计一行一列或一斜的得分

```
int CountScore(int n[], int turn) // 正斜线、反斜线、横、竖，均转成一维数组来计算
{
    int Scoretmp = 0, L = n[0], Empty = 0, Number = 0;
    if (n[1] == 0) Empty++;
    else if (n[1] == turn) Number++;

    int i;
    for (i = 2; i <= L; i++)
    {
        if (n[i] == turn) Number++;
        else if (n[i] == 0)
        {
            if (Number == 0) Empty = 1;
            else
            {
                Scoretmp += ScoreTable(Number, Empty + 1);
                Empty = 1;
                Number = 0;
            }
        }
        else
        {
            Scoretmp += ScoreTable(Number, Empty);
            Empty = 0;
            Number = 0;
        }
    }
    Scoretmp += ScoreTable(Number, Empty);
    return Scoretmp;
}
```

图 28: 27

3.27 评估整个棋局的得分

```
int Evaluate()//评估函数，评估局势
{
    int Score_BLACK_CHESS = 0, Score_WHITE_CHESS = 0;
    //横排
    int n[505] = { 0 }, i, j;
    for (i = 0; i < 15; i++)
    {
        //vector<int> n;
        for (j = 0; j < 15; j++)n[++n[0]] = Map[i][j];
        Score_BLACK_CHESS += CountScore(n, BLACK_CHESS);
        Score_WHITE_CHESS += CountScore(n, WHITE_CHESS);
        n[0] = 0;
    }

    //竖排
    for (j = 0; j < 15; j++)
    {
        for (i = 0; i < 15; i++)n[++n[0]] = Map[i][j];
        Score_BLACK_CHESS += CountScore(n, BLACK_CHESS);
        Score_WHITE_CHESS += CountScore(n, WHITE_CHESS);
        n[0] = 0;
    }

    //上半正斜线
    for (i = 0; i < 15; i++)
    {
        int x, y;
        for (x = i, y = 0; x < 15 && y < 15; x++, y++) n[++n[0]] = Map[y][x];
        Score_BLACK_CHESS += CountScore(n, BLACK_CHESS);
        Score_WHITE_CHESS += CountScore(n, WHITE_CHESS);
        n[0] = 0;
    }

    //下半正斜线
    for (j = 1; j < 15; j++)
    {
        int x, y;
        for (x = 0, y = j; y < 15 && x < 15; x++, y++) n[++n[0]] = Map[y][x];
        Score_BLACK_CHESS += CountScore(n, BLACK_CHESS);
        Score_WHITE_CHESS += CountScore(n, WHITE_CHESS);
        n[0] = 0;
    }
```

图 29: 28

3.28 最小值搜索 (假设人按最优方法下棋)

```
int Min_AlphaBeta(int Dep, int alpha, int beta) // 当 min（对手）走步时，对手的最好情况
{
    int res = Evaluate();
    if (Dep == 0)return res;

    struct point v[505];
    //EmptyPoint(v);
    v[0].x = 0;
    int i, j;
    for (i = 0; i < 15; i++)
        for (j = 0; j < 15; j++)
            if (Map[i][j] == EMPTY && Around(i, j))
            {
                v[++v[0].x].x = i;
                v[v[0].x].y = j;
            }

    int L = v[0].x;
    int best = INT_MAX;
    for (i = 1; i <= L; i++)
    {
        Map[v[i].x][v[i].y] = WHITE_CHESS;
        // 我是极小层，我要的是更小的数。我找过的孩子中，目前为止找到的最小的数是best，如果best小于了
        // 前辈们之前找到的最小值，那么我将更新它，并且告诉下面未遍历过的孩子，比alpha大的数就不要再给我了
        int tmp = Max_AlphaBeta(Dep - 1, best < alpha ? best : alpha, beta);
        Map[v[i].x][v[i].y] = EMPTY;
        if (tmp < best)best = tmp;
        if (tmp < beta)break;
    }
    return best;
}
```

图 30: 29

3.29 最大值搜索 (AI 用最优的走法)

```
int Max_AlphaBeta(int Dep, int alpha, int beta)//AI走步时应该考虑最好的情况
{
    int res = Evaluate();
    if (Dep == 0)return res;

    struct point v[505];
    //EmptyPoint(v);
    int i, j;
    v[0].x = 0;
    for (i = 0; i < 15; i++)
        for (j = 0; j < 15; j++)
            if (Map[i][j] == EMPTY && Around(i, j))
            {
                v[++v[0].x].x = i;
                v[v[0].x].y = j;
            }

    int L = v[0].x;
    int best = INT_MIN;
    for (i = 1; i <= L; i++)
    {
        Map[v[i].x][v[i].y] = BLACK_CHESS;
        int tmp = Min_AlphaBeta(Dep - 1, alpha, best > beta ? best : beta);
        Map[v[i].x][v[i].y] = EMPTY;
        if (tmp > best)best = tmp;
        // 这是极大层，电脑选取最大值点。到目前为止(i=0)，已知tmp。那么该层向上返回的值将不小于tmp
        if (tmp > alpha)break;

        // 我的上一层告诉我，它找到的最小的数是alpha，它是极小层，他需要更小的数。
        // 如果我找到的tmp比alpha大，那么就不要找了，因为我是极大层，我只会返回更大的数给上一层
    }
    return best;
}
```

图 31: 30

3.30 alphabeta 剪枝算法

```
struct point MinMax_AlphaBeta(int Dep)//极大极小值算法搜索n步后的最优解
{
    struct point v[505], v2[505];
    //EmptyPoint(v);
    v[0].x = 0;
    int i, j;
    for (i = 0; i < 15; i++)
        for (j = 0; j < 15; j++)
            if (Map[i][j] == EMPTY && Around(i, j))
            {
                v[++v[0].x].x = i;
                v[v[0].x].y = j;
            }

    int best = INT_MIN;
    int L = v[0].x;
    v2[0].x = 0; // v2[0].x 表示 v2 中的元素数量

    for (i = 1; i <= L; i++)
    {
        Map[v[i].x][v[i].y] = WHITE_CHESS;//选该子，将该子置白，防止后面递归时，再递归到
        int tmp = Min_AlphaBeta(Dep, INT_MAX, INT_MIN);
        if (tmp == best)v2[++v2[0].x] = v[i];
        if (tmp > best)
        {
            best = tmp;
            v2[0].x = 0;
            v2[++v2[0].x] = v[i];
        }
        Map[v[i].x][v[i].y] = EMPTY;    //假设完之后，该子需要重新置空，恢复原来的样子
    }

    L = v2[0].x;

    int k = (int)(rand() % L) + 1;
    return v2[k];
}
```

图 32: 31

3.31 返回 ai 计算出来下棋的点

```
struct point aiTurn(int chess_board[15][15])
{
    int i, j;
    for (i = 0; i < 15; i++)
        for (j = 0; j < 15; j++) Map[i][j] = chess_board[i][j];

    struct point preferedPos = MinMax_AlphaBeta(2);
    return preferedPos;
}
```

图 33: 32

3.32 ai 放置棋子

```
void ai_set_chess() {
    // AI
    point temp = game_record.back();
    int i;
    for (i = 0; i < Count; i++)
    {
        if (OTtable[temp.x][temp.y][i] && win_s[0][i] != 7) win_s[0][i]++;
        if (BLACK_CHESS_table[temp.x][temp.y][i])
        {
            BLACK_CHESS_table[temp.x][temp.y][i] = 0;
            win_s[1][i] = 7;
        }
    }
    struct point pos = aiTurn(chess_board);
    if (chess_board[pos.x][pos.y] == 0) {
        game_record.push_back(pos);
        chess_board[pos.x][pos.y] = -1;
        rounds++;
        play_side_now = BLACK_SIDE;
        draw_chess_board(chess_board);
    }
}
```

图 34: 33

3.33 检测是否胜利

```
void check_win() {
    if (is_one_side_win(WHITE_SIDE)) {
        win(WHITE_SIDE);
    }
    else if (is_one_side_win(BLACK_SIDE)) {
        win(BLACK_SIDE);
    }
}
```

图 35: 34

3.34 展示点赞图片

```
void zan(){
    putimage(450, 300, &pic_zan);
    wait_mouse_click();
    draw_chess_board(chess_board);
}
```

图 36: 35

3.35 展示关于信息

```
void about(){
    putimage(pic_x, pic_y, &pic_about);
    wait_mouse_click();
    draw_chess_board(chess_board);
}
```

图 37: 36