答辩

淦饭不想排队（402）

NEVER-QUEUE

队长：刘豫吉
队员：任志晗
杨定华
韩颜泽
孟思奇

大纲

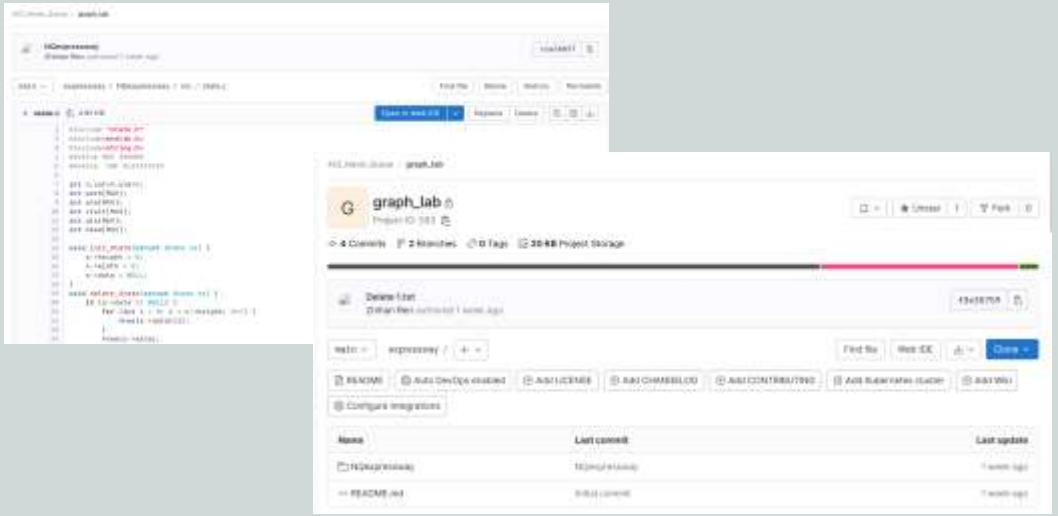
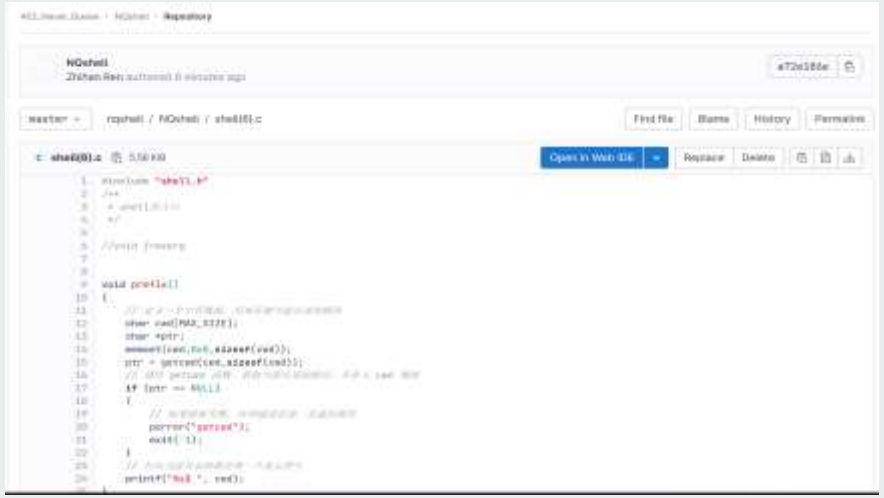
CONTENTS

402-NEVER-QUEUE

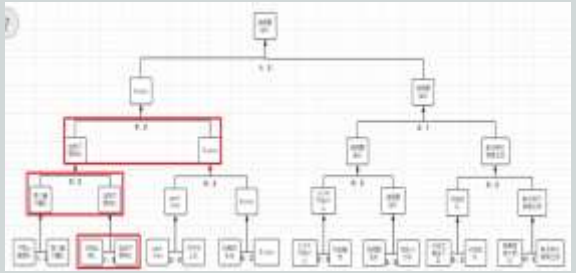
◎ 完成项目情况汇总

COMPLETE PROJECT SUMMARY

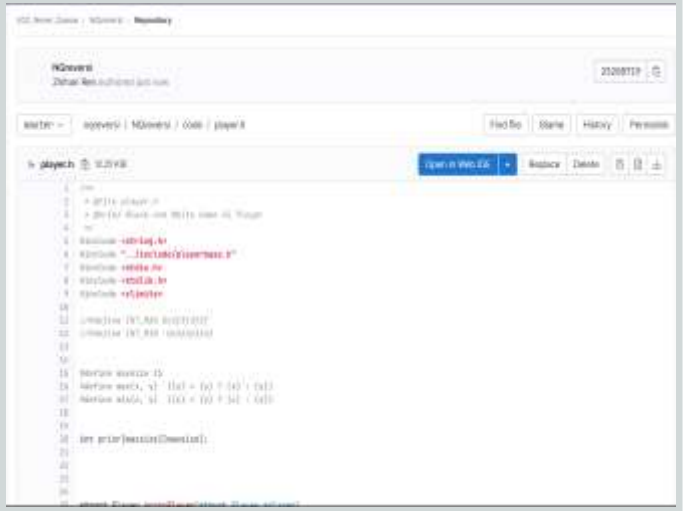
项目汇总

项目名称	完成情况及成绩	Git上传情况
高速路网设计实验	测评报告 50 / 50 得分情况 测例说明 分数 状态 Basic Test 1 10 已通过 Time Attack 1 20 已通过 Memory Test 1 20 已通过 Basic Test 2 10 已通过 Time Attack 2 20 已通过 Memory Test 2 20 已通过	
实现自己的Shell	测评报告 50 / 50 得分情况 测例说明 分数 状态 Basic Test 1 15 已通过 Basic Test 2 15 已通过 Basic Test 3 15 已通过 Valgrind Memory Test 20 已通过 Fork bomb Test 35 已通过	

项目汇总

项目名称	完成情况	成绩	Git上传情况
黑白棋	   	任务一： 4 | 433 | 402 | 淦饭不想排队 任务二： 9 | 799 | 402 | 淦饭不想排队 任务三： 2 | 2034 | 402 | 淦饭不想排队 任务四： 8 | 2704 | 402 | 淦饭不想排队 PK成绩：  在PK中取得了不错的成绩  	

项目汇总

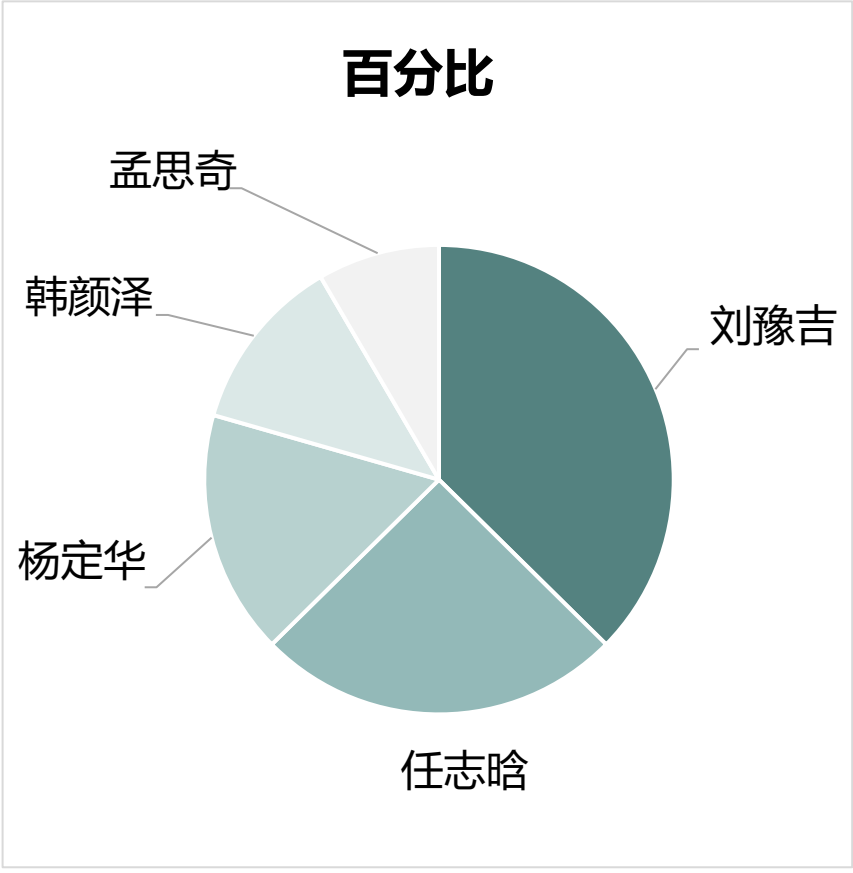
项目名称	完成情况	成绩	Git上传情况
蒜头抢分大作战			

◎ 团队分工

DIVISION OF LABOR OF THE TEAM

团队分工

成员	完成的任务
刘豫吉 (队长)	高速路网：state.c中的读入函数，solve1，solve2 shell：shell.c和log.c的debug调试 黑白棋：alpha-beta剪枝算法设计、评估函数 蒜头抢分大作战：算法设计与构思、min_max函数 debug调试、搜集网络资料、答辩资料收集
任志晗	高速路网：state.c的solve1 黑白棋：评估函数 蒜头抢分大作战：算法设计与构思、evaluate函数 搜集网络资料、答辩资料收集
杨定华	高速路网：state.cs的solve1 黑白棋：判断合法性函数 蒜头抢分大作战：算法设计与构思、BFS函数 搜集网络资料、答辩PPT制作
韩颜泽	高速路网：state.h shell:log.c、shell.c的编写 黑白棋：返回落子位置 搜索策略、搜集网络资料、答辩资料收集、实验报告
孟思奇	高速路网：state.c 黑白棋：返回落子位置 搜集网络资料、答辩资料收集

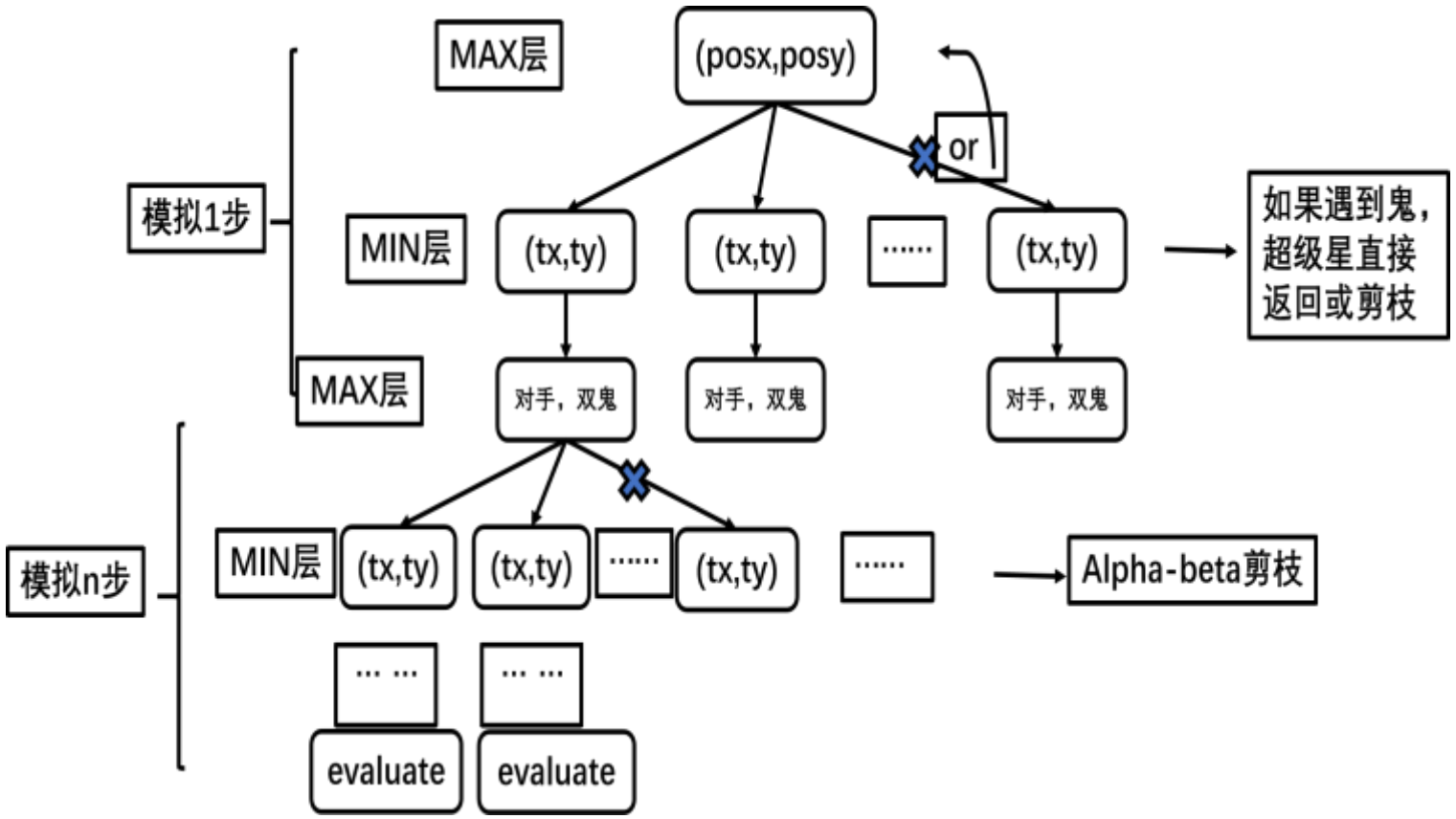


◎ 抢分大作战AI算法说明

ALGORITHM SPECIFICATION

算法说明

Minmax算法：




```
float alpha_beta(struct Player *player, int depth, float alpha, float beta, int maximizingPlayer)
{
    if(depth==2*foot-1)
    {
        float n=evaluate(player);
        return n;

    }

    if (maximizingPlayer==0)
    {
        for (int i = 0; i < 5; i++)
        {
            float score;
            int tx=player->your_posx+dir[i][0];
            int ty=player->your_posy+dir[i][1];
            if (in(tx,ty,player)&&player->mat[tx][ty]!='#')
            {

                if(depth==2&&player->your_status > 0&&(tx==player->ghost_posx[0]&&ty==player->ghost_posy[0]||tx==player->ghost_po
                return FLT_MAX;
                else if(depth==2&&player->your_status == 0&&(tx==player->ghost_posx[0]&&ty==player->ghost_posy[0]||tx==player->gh
                return -FLT_MAX;
                else
                {
                    struct Player *next_player = copy_move(player,tx,ty,maximizingPlayer);
                    score= alpha_beta(next_player, depth + 1, alpha, beta, 5);
                    freePlayer(next_player);
                }

                alpha = max(alpha, score);

                if (beta <= alpha)
                {
                    return beta;
                }
            }
        }
        return alpha;
    }
    else if(maximizingPlayer==5)
    {
        struct Player *next_player = copy_move(player,player->opponent_posx,player->opponent_posy,maximizingPlayer);
        ghost_move(next_player);
        float score = alpha_beta(next_player, depth + 1, alpha, beta,0);
        beta = min(beta, score);

        freePlayer(next_player);

        if (beta <= alpha)
        {
            return alpha;
        }
        return beta;
    }
}
```

算法说明

Minmax算法：evaluate函数

第一版评估函数：
缺点：分条件的评估公式太过繁琐，且不能考虑全部情况

```
int evaluate(struct Player *player)
{
    int distance[6];//距离策略
    //0:己方  1: 到鬼1  2: 到鬼2  3: 到普通星  4: 到超级星  5: 到对方AI
    for (int i = 1;i < 6;i++)
    {
        distance[i] = bfs_distance(player,0,i);

        printf("%d",distance[i]);
    }

    if (distance[1] > 3 && distance[2] > 3)
    {
        if (player->your_status == 0 && player->opponent_status == 0)
        {
            return 20 / distance[4] + 10 / distance[3] - 500 / (distance[1] + distance[2]);
        }

        if (player->your_status == 0 && player->opponent_status > 0)
        {
            return 20 / distance[4] + 10 / distance[3] - (player->your_score / 2) / distance[5] - 500 / (distance[1] + distance[2]);
        }

        if (player->your_status > 0 && player->opponent_status == 0)
        {
            return 20 / distance[4] + 10 / distance[3] + 200 / (distance[1] + distance[2]) + (player->opponent_score / 2) / distance[5];
        }

        if (player->your_status > 0 && player->opponent_status > 0)
        {
            return 20 / distance[4] + 10 / distance[3] + 200 / (distance[1] + distance[2]);
        }
    }

    else
    {
        return (-500) / (distance[1] + distance[2]);
    }
}
```

第二版评估函数：
在第一版的基础上改进了评估公式，只需写一个通式，为不同情况赋权值即可

```
float evaluate(struct Player *player)
{
    int distance[6];//距离策略
    //0:己方  1: 到鬼1  2: 到鬼2  3: 到普通星  4: 到超级星  5: 到对方AI

    distance[1] =1 + manhattan(player,0,1);
    distance[2] =1 + manhattan(player,0,2);
    distance[3] =1 + bfs_distance(player,0,3);
    distance[4] =1 + bfs_distance(player,0,4);

    distance[5] =1 + manhattan(player,0,5);
    //printf("%d",distance[1]);

    float ghost ,star = 100,superstar = 200,opponent = -(player->your_score / 2);

    if (distance[1] >= 2 && distance[2] >= 2)
    {
        if (player->your_status == 0 && player->opponent_status == 0)
        {
            opponent = 0;
            ghost = 0;
        }

        else if (player->your_status > 0 && player->opponent_status == 0)
        {
            opponent = (player->opponent_score / 2);
            ghost = 500;
        }

        else if (player->your_status > 0 && player->opponent_status > 0)
        {
            opponent = 0;
            ghost = 300;
        }

        else if (player->your_status == 0 && player->opponent_status == -1)
        {
            opponent = 0;
            ghost = 0;
        }

        if (distance[4] == -1)
        {
            superstar = 0;
        }

        if (distance[3] == -1)
        {
            star = 0;
        }
        return player->your_score +  star / (distance[3]) + superstar / (distance[4
    }

    else
    {
        if(player->your_status == 0)
        {
            ghost = -500;
        }
        else
        {
            ghost = 500;
        }
        return player->your_score + ghost/(distance[1] + distance[2]);
    }
```

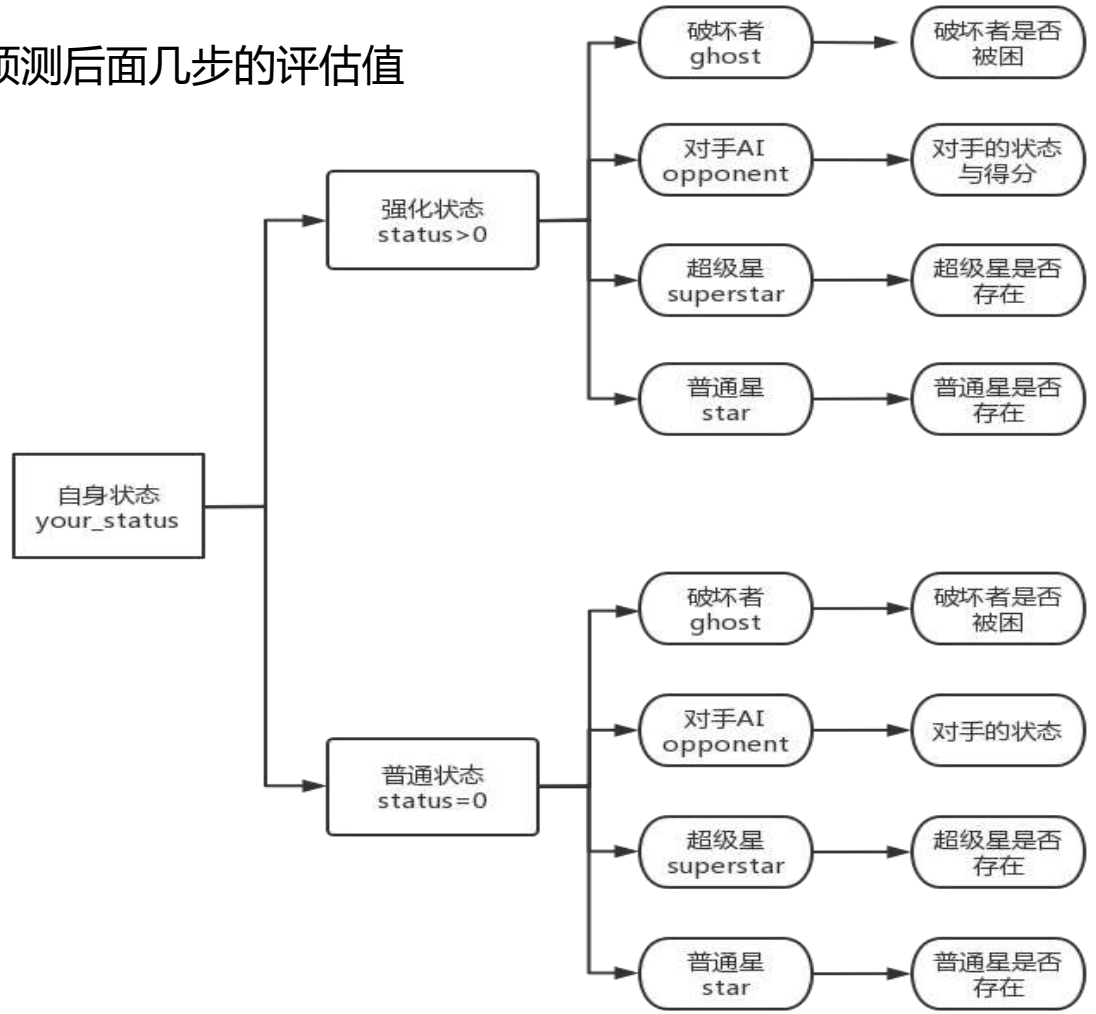
算法说明

Minmax算法：evaluate函数

第三版在计算距离时加上了foot（模拟走的步数），可以更好地预测后面几步的评估值
同时细化了不同情况下权值的赋用

```
float evaluate(struct Player *player)
{
    int distance[6];//距离策略
    //0:己方  1：到鬼1  2：到鬼2  3：到普通星  4：到超级星  5：到对方AI
    struct Point a;
    distance[1] =foot+ bfs_distance(player,0,1,&a);
    distance[2] =foot+ bfs_distance(player,0,2,&a);
    distance[3] =foot+ bfs_distance(player,0,3,&a);
    distance[4] =foot+ bfs_distance(player,0,4,&a);
    distance[5] =foot+ manhattan(player,0,5);

    float ghost_score=0,star_score,superstar_score,opponent_score;
    float ghost=500 ,star=20 ,superstar=1000 ,opponent=0;
```



算法说明

Minmax算法：evaluate函数

代码实现

```
if(player->your_status > 0)
{
    if(distance[1] == foot-1 && distance[2] == foot-1 )
    {
        ghost_score=0;
    }
    else if(distance[1] == foot-1)
    {
        ghost_score=ghost/distance[2];
    }
    else if(distance[2] == foot-1)
    {
        ghost_score=ghost/distance[1];
    }
    else
    {
        ghost_score=(1.0/2)*(ghost/distance[1]+ghost/distance[2]);
    }

    if (player->opponent_status == 0&&player->opponent_score>0)
    {
        opponent_score = player->opponent_score/2;
    }
    else
    {
        opponent_score=0;
    }

    if (distance[4] == foot-1)
    {
        superstar_score = player->your_status*50;
    }
    else
    {
        superstar_score = player->your_status*50+superstar/distance[4];
    }

    if (distance[3] == foot-1)
    {
        star_score = 0;
    }
    else
    {
        star_score = star/distance[3];
    }

}
else
{
    if(distance[1] == foot-1 &&distance[2] == foot-1 )
    {
        ghost_score=0;
    }
    else if(distance[1] == foot-1)
    {
        ghost_score=-ghost/distance[2];
    }
    else if(distance[2] == foot-1)
    {
        ghost_score=-ghost/distance[1];
    }
    else
    {
        ghost_score=(1.0/2)*(-ghost/distance[1]-ghost/distance[2]);
    }

    if (player->opponent_status > 0&&player->your_score>0)
    {
        opponent_score = -player->your_score/2;
    }
    else
    {
        opponent_score=0;
    }

    if (distance[4] == foot-1)
    {
        superstar_score =0;
    }
    else
    {
        superstar_score = superstar/distance[4];
    }

    if (distance[3] == foot-1)
    {
        star_score = 0;
    }
    else
    {
        star_score = star/distance[3];
    }

}
return player->your_score + ghost_score+star_score+superstar_score+oppo
```

算法说明

Minmax算法：BFS算法

```
struct node {
    int x, y, d;
} ;
int bfs_distance(struct Player *player, int start, int end,struct Point *front)
{
    int vis[110][110];
    memset(vis,0,sizeof(vis));
    struct node q[10010];
    int l = 0, r = 0;
    int sx=-1,sy=-1;
    int ex=-1,ey=-1;
    switch (start)
    {
    case 0:
        sx = player->your_posx;
        sy = player->your_posy;
        break;

    case 1:
        sx = player->opponent_posx;
        sy = player->opponent_posy;
        break;
    default:
        return -1;
        break;
    }
```

```
switch (end)
{
case 1:
    ex = player->ghost_posx[0];
    ey = player->ghost_posy[0];
    break;
case 2:
    ex = player->ghost_posx[1];
    ey = player->ghost_posy[1];
    break;
case 5:
    ex = player->opponent_posx;
    ey = player->opponent_posy;
default:
    break;
}
if (sx == ex && sy==ey)
{
    front->X=sx;
    front->Y=sy;
    return 0;
}
if(player->mat[sx][sy] == 'o' && end==3)
{
    return 0;
}
if(player->mat[sx][sy] == 'O' && end==4)
{
    return 0;
}
```

```
struct node t = {sx, sy, 0};
q[r++] = t;
vis[sx][sy] = 1;
while (l < r) {
    struct node now = q[l++];

    for (int i = 0; i < 4; i++)
    {
        int tx = now.x + dir[i][0];
        int ty = now.y + dir[i][1];
        if (in(tx, ty,player) && player->mat[tx][ty] != '#' && !vis[tx][ty])
        {
            if (tx == ex && ty==ey)
            {
                front->X=now.x;
                front->Y=now.y;
                return now.d+1;
            }
            if(player->mat[tx][ty] == 'o' && end==3)
            {
                return now.d+1;
            }
            if(player->mat[tx][ty] == 'O' && end==4)
            {
                return now.d+1;
            }
            vis[tx][ty] = 1;
            struct node t = {tx, ty, now.d + 1};
            q[r++] = t;
        }
    }
}
return -1;
}
```

编号	代表
0	己方AI
1	第一个鬼
2	第二个鬼
3	普通星星
4	超级星星
5	对方AI

◎ 收获与反思

HARVEST AND REFLECTION

收获反思

收获：

1、巩固C语言基础知识，能够写出更高质量的代码，规范了代码风格

2、学习了多种数据结构与算法，并能应用到不同的实际问题中

3、学会应用dijkstra算法、alpha-beta剪枝等

4、熟悉了小组合作的流程（讨论->分工->合并->调试测试->项目完成），思维的碰撞让我们感受到团队协作的魅力

5、学习了实际工作时开发工具的使用，如git、Docker、虚拟机的使用等

反思：

1、git的使用不够熟练，没有将git应用到实践中

2、尚不擅长本地的gdb调试，调试项目效率较为低下

3、不能熟练虚拟机本地测试，更多的是采用线上测试

4、不同成员分工写的函数合在一起时会有冲突导致达不到较好的效果


THANK YOU FOR YOUR WATCHING
感谢您的观看