Pacman2 Bot for Botzone

> **By SEUCSE 09018316 黄开鸿 & 09018330 孙毅远** （Team 1）

github仓库链接 botzone官网

目录导航

Introduce

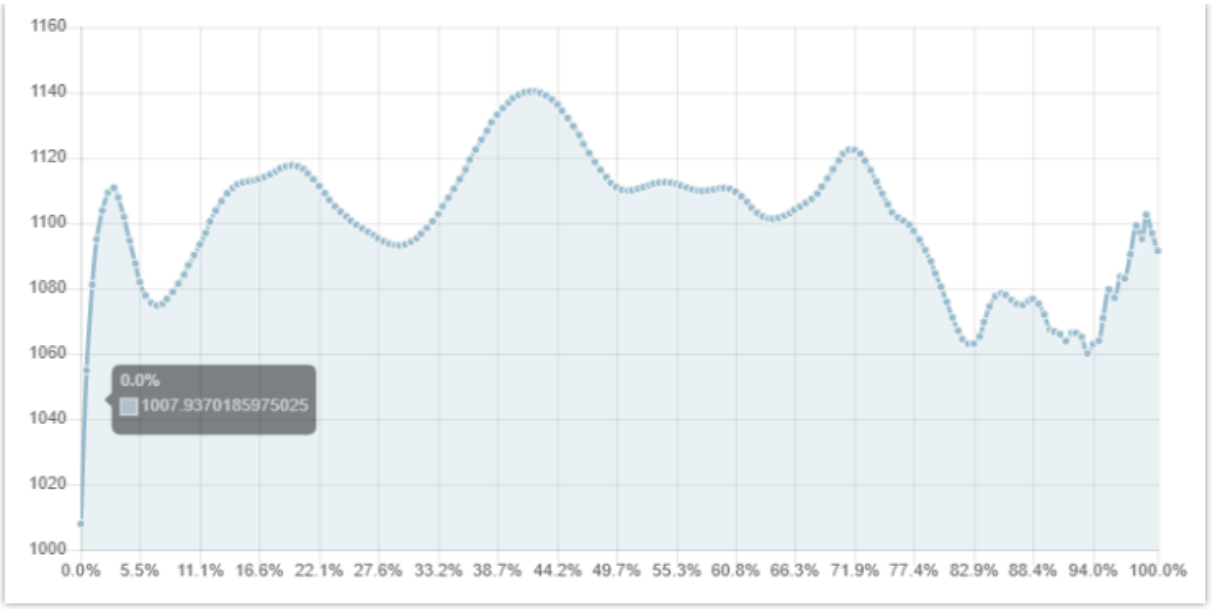
botzone 网站上 Pacman2 游戏的开发 BOT。旨在开发一个基于非机器学习思路的高适应性，高内聚低耦合的 Pacman2 游戏 Bot，将众多基础算法与游戏规则进行有机结合，并在开发中保证 Bot 代码的可扩展性和可读性，保证快速通过对局分析，迭代 Bot 的决策逻辑。

以下表格为当前在 botzone 网站上的天梯数据。

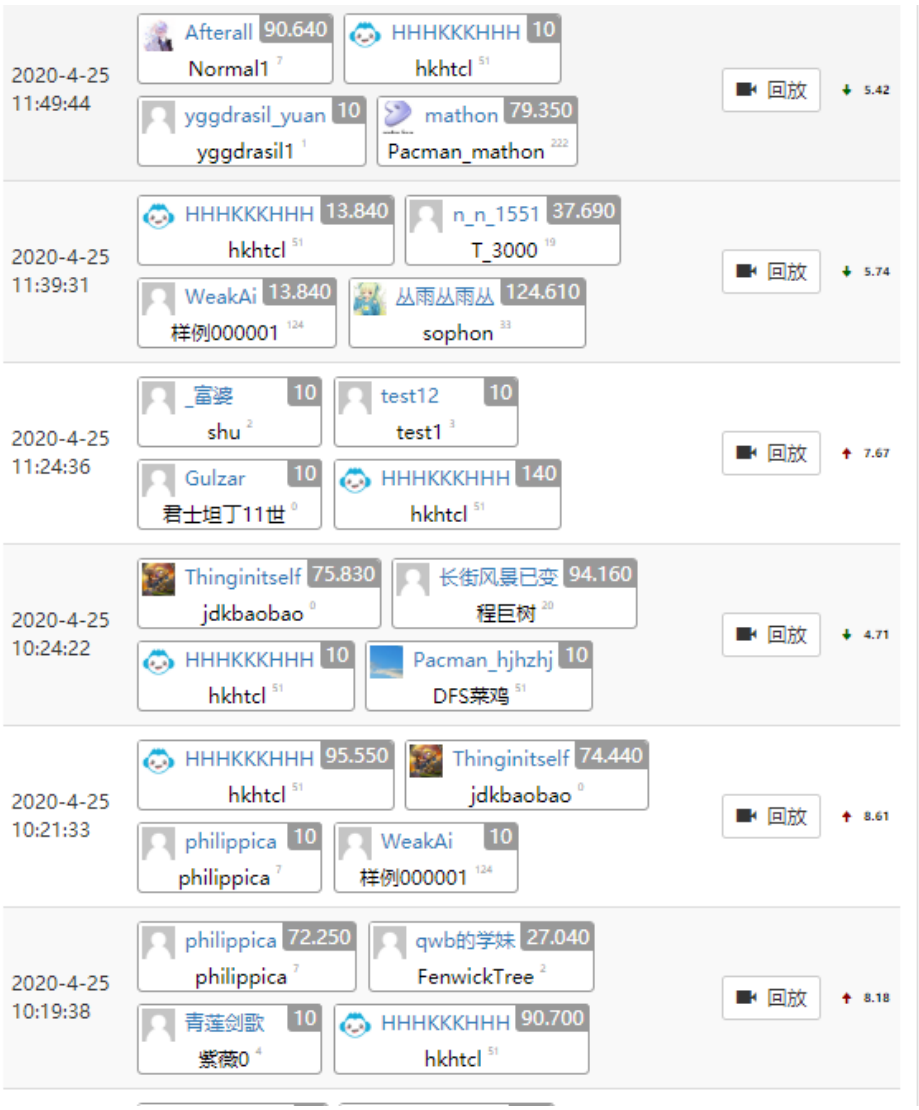
截止 2020-4-5 天梯数据

Bot 名	天梯分数	天梯排名
hkhtcl	1111.15	61
Normal1	1091.50	65

截止 2020-4-25 天梯分数波动



对局截图



Version 0.1.0

Review

无老版本。

Introduce

此版本为该 Bot 的初始版本，**目的在于完成对游戏规则决策基本的整体权衡框架**，使其在可接受的程度上完整完成游戏流程，并以**工程化，模块化**的思路为未来的功能改进铺平道路。

整体决策程序的设计思路参考我校机电中心推出的电脑鼠比赛中用以驱动电脑鼠的程序（团队中黄开鸿同学曾参与，对该程序有使用权），**以 BFS 算法为核心构建链式决策机制**，从工程角度大致流程如下：

1. 利用 BFS 的思想对整张地图做遍历，获取每一个可达点的关键信息（包含最短路径及其需求方向，价值等）。
2. 利用关键信息作出 “ 静态 ” 决策（即与其他玩家无关的决策）。
3. 检查其他玩家的信息（行为与位置），规避危险行为。
4. 检查自身有利决策（即金光法器）。
5. 传递必要全局信息。
6. 权衡2 ， 3 ， 4 步的价值，作出最优决策。

> 需要注意的是，此处仅仅是列出了在理想状态下，设计者意图实现的该程序从接收回合输入到作出该回合决策的流程，并不意味着此版本已经完成以上所有步骤的设计与实现。具体已实现部分详见下文。

现版本已实现游戏内功能描述为：

- 每回合寻找离自身可达路径最短的果实
  - 利用 BFS 算法计算出到达每一个果实的最短路径长，储存在每一个可达点上
- 在每一个可达点上还会储存此最短路径需求的第一步方向，方便作出决策时快速确定应前进方向。
- 每回合寻找离自身最近的果实生成器
- 当找不到可达果实时自动前往最近的果实生成器旁的随机位置
- 检查是否正处在被吃的位置
- 检查相邻是否有比自己强壮的玩家
- 检查是否有金光法器能击中的目标
- 沿着当前位置的四个方向遍历，检查是否有玩家
- 检查可击中目标的可能逃跑路线（即判断是否必中）
- 对可击中玩家的可能逃跑路径做判断
- 简单权衡部分决策价值

现版本已实现的功能性功能为：

- 本地自动截取输入 json 中对局初始化数据
- 本地进行调试并能读取必要信息，定义回合数等
- 模块化部分实现函数，规范化函数封装，便于以后链式扩展

已知的未来改进路线：

- 对最优果实的选择不能单纯以距离为判定方式，应该还权衡给每个果实周围的果实，考虑沿着哪一个路径能最快获取最多果实，同时权衡果实的竞争者（附近玩家）的多少
- 前往果实生成器位置的路径判定需要考虑其八个不同的相邻位置的价值，保证前往后能最快吃到果实
- 金光法器对必中目标的判断应该更加智能，考虑目标的可能动作。
- 避免被吃的算法需要再加改进，应考虑更多复杂情况
- 应对对战阶段进行决策划分，在对战开始初期应该以较保守的决策，保证初期生存而非得分（否则初期很容易死亡）。

Evaluate

对比其他同等级竞争程序，经过多次自我测试与对战游戏的实验表明，这样的链式决策逻辑保证了程序的稳定性和较小复杂度。尽管在决策的正确性，合理性和智能性上需要走的路还很长，但是程序从上线以来只在开发初期发生过一次动作错误，无崩溃或超时记录，版本迭代的测试数量不超过 2 次，团队对其未来的成长度持高度乐观态度。

回到游戏本身，本程序在极端情况下的某些表现并不尽人意，特别是在对战初期无法完全规避被吃的可能性。但是单单依赖能百分百抓住攻击机会这一点，让本程序在较长时间的对局中具有一定优势（因为较长时间的对局中出现开火的机会更多）

现阶段本程序的核心依然依赖 BFS 对地图进行遍历，采用链式的决策思路。优点在于 BFS 算法本身的确定性保证了可达点关键信息的绝对准确，相比 MCTS 等迭代算法来的更加稳定，也能附加更多的自定义信息。但现阶段的决策逻辑导致在程序角度，自身和其他三名玩家并不处在一个逻辑层上，而是以自身为主，外界（包括其他玩家信息）为外界附加输入，这样导致对其他玩家的行为考虑不足。

总体现阶段胜率并不尽人意，主要问题出在对战初期极其容易因为敌我同时抢夺同一果实，而力量较小导致死亡。

Code

- 游戏决策代码

> 仅贴出关键部分算法代码，略去判定程序本体和执行函数（main）

```
namespace Bot
{
    int vis[20][20] = {};//是否访问过
    int distance[20][20][2] = {};//距离，dis[x][y][0]表示当前点到(x,y)的距离，dis[x][y][1]表示当前点到(x,y)走的第一步的方向
    int shootData[4] = {0,0,0,0};//记录当前位置四个方向上能打到的人数，排序：上 右 下 左
    struct T
    {
        int x, y;
        T() {}
        T(int x, int y) :x(x), y(y) {}
    };
    queue <T>q;
    void BFSd(Pacman::GameField& gameField, int myID, int nowx, int nowy)//BFS遍历图上所有点，求出距离
    {
        while (!q.empty()) q.pop();
        memset(vis, 0, sizeof(vis));
        for (int i = 0; i < 20; i++)
            for (int j = 0; j < 20; j++)
            {
                distance[i][j][0] =  0x3fff;
                distance[i][j][1] = -1;
            }
        vis[nowx][nowy] = 1;
        q.push(T(nowx, nowy));
        distance[nowx][nowy][0] = 0;
        T u, v;
        while (!q.empty())
        {
            u = q.front();
            q.pop();
            for (int i = 0; i < 4; i++)
            {
                v.x = u.x + Pacman::dy[i];
                v.y = u.y + Pacman::dx[i];
                //边界判定
                if (v.x < 0) v.x += gameField.height;
                if (v.x >= gameField.height) v.x -= gameField.height;
                if (v.y < 0) v.y += gameField.width;
                if (v.y >= gameField.width) v.y -= gameField.width;

                //(u.x,u.y)到(v.x,v.y)方向上没有墙，(v.x,v.y)不是豆子产生器，(v.x,v.y)没走过
                if (!(gameField.fieldStatic[u.x][u.y] &
Pacman::direction2OpposingWall[(Pacman::Direction)(i)])
                    && !(gameField.fieldStatic[v.x][v.y] & Pacman::generator)
                    && !vis[v.x][v.y])

                {
                    //判断是不是有个坏家伙在这个方向上，有的话设成死路
                    bool hasBadguy = false;
                    if (gameField.fieldContent[v.x][v.y] & Pacman::playerMask)
                        for (int player = 0; player < MAX_PLAYER_COUNT; player++)
                            if (gameField.fieldContent[v.x][v.y] & Pacman::playerID2Mask[player])
                            {
                                if (gameField.players[player].strength >
gameField.players[myID].strength) {
                                    //有个坏家伙则往其他方向
```

```
                                hasBadguy = true;
                            }
                        }
                    if (hasBadguy) continue;
                    q.push(v);
                    vis[v.x][v.y] = 1;
                    distance[v.x][v.y][0] = distance[u.x][u.y][0] + 1;
                    if (u.x == nowx && u.y == nowy) distance[v.x][v.y][1] = i;//起始点方向记录
                    else distance[v.x][v.y][1] = distance[u.x][u.y][1];//传递方向
                }
            }
        }
    }
    int calc(Pacman::GameField& gameField, int myID)
    {
        int infDis = 0x3fff;
        int final = -1;
        Pacman::Player& x = gameField.players[myID];
        BFSd(gameField, myID, x.row, x.col);
        int douDis = infDis, douDir = -1;
        //枚举图上所有点找最近的豆子
        for (int i = 0; i < gameField.height; i++)
            for (int j = 0; j < gameField.width; j++)
            {
                if ((gameField.fieldContent[i][j] & Pacman::smallFruit)|| (gameField.fieldContent[i]
[j] & Pacman::largeFruit))
                    if (douDis > distance[i][j][0])
                    {
                        douDis = distance[i][j][0];
                        douDir = distance[i][j][1];
                    }
            }
        //暂时先决定去吃豆子
        final = douDir;
        int genDis = infDis, genDir = -1;
        //找最近的豆子产生器（先粗略找一下,只有场地上没豆子的时候会去找）
        if (douDis == infDis) {
            genDis = distance[gameField.generators[0].row-1][gameField.generators[0].col][0];
            for (int i = 1; i < 4; i++)
                {
                if (genDis >= distance[gameField.generators[i].row-1][gameField.generators[i].col][0])
{
                    genDis = distance[gameField.generators[i].row-1][gameField.generators[i].col][0];
                    genDir = distance[gameField.generators[i].row-1][gameField.generators[i].col][1];
                    }
                }
            //没豆子吃辣。那就去等着
            final = genDir;
        };
        //遍历四个方向看看有没有人可以打
        int shootDir = -1;
        int targetNum = 0;
        for (int dir = 0; dir < 4; dir++) {
            int r = x.row;
            int c = x.col;
            while (!(gameField.fieldStatic[r][c] & Pacman::direction2OpposingWall[dir]))
            {
                r = (r + Pacman::dy[dir] + gameField.height) % gameField.height;
                c = (c + Pacman::dx[dir] + gameField.width) % gameField.width;

                // 如果转了一圈回来……
                if (r == x.row && c == x.col)
                    break;
                if (gameField.fieldContent[r][c] & Pacman::playerMask)
                    for (int player = 0; player < MAX_PLAYER_COUNT; player++)
                        if (gameField.fieldContent[r][c] & Pacman::playerID2Mask[player])
                        {
                            //检测此玩家是不是跑不掉了，只对必中的目标开火
                            if (
```

```
                                (gameField.fieldStatic[r][c] &
Pacman::direction2OpposingWall[(Pacman::Direction)(dir + 1) % 4])
                                && (gameField.fieldStatic[r][c] &
Pacman::direction2OpposingWall[(Pacman::Direction)(dir - 1) % 4])
                                ) {
                                shootData[dir]++;
                            }
                        }
            }
            //挑目标最多的方向打
            if (shootData[dir] > targetNum ) {
                shootDir = dir + 4;
                targetNum = shootData[dir];
            }
        }
        //有人可以打当然要打啦
        if (shootDir!= -1&&(gameField.SKILL_COST<gameField.players[myID].strength)) final = shootDir;
        return final;
    }

}
```

- 工程部分调试代码

> 工程中便于本地调试的执行代码

```
namespace Helpers
{

    double actionScore[9] = {};
    void LocalPlay
    (Pacman::GameField& gameField, int myID,int countLimit)
    {
        int count = 0, myAct = -1;
        while (true)
        {
            for (int i = 0; i < MAX_PLAYER_COUNT; i++)
            {
                if (gameField.players[i].dead)
                    continue;
                gameField.actions[i] = (Pacman::Direction)Bot::calc(gameField, i);;
            }

            if (count == 0)
                myAct = gameField.actions[myID];

            // 演算一步局面变化
            gameField.DebugPrint();
            // NextTurn返回true表示游戏没有结束
            bool hasNext = gameField.NextTurn();
            count++;
            //限制下最大回合数
            if (count >= countLimit) {
                break;
            }
            if (!hasNext)
                break;
        }
        // 计算分数
        int total = 0;
        for (int _ = 0; _ < MAX_PLAYER_COUNT; _++)
            total += gameField.players[_].strength;

        if (total != 0)
            actionScore[myAct + 1] += (10000 * gameField.players[myID].strength / total) / 100.0;

        // 恢复游戏状态到最初（就是本回合）
        while (count-- > 0)
            gameField.PopState();
    }
```

```
}
```

Version 0.2.0

更新时间：2020-4-4

Review

0.1.0 已实现游戏内功能描述为：

- 每回合寻找离自身可达路径最短的果实
  - 利用 BFS 算法计算出到达每一个果实的最短路径长，储存在每一个可达点上
- 在每一个可达点上还会储存此最短路径需求的第一步方向，方便作出决策时快速确定应前进方向。
- 每回合寻找离自身最近的果实生成器
- 当找不到可达果实时自动前往最近的果实生成器旁的随机位置
- 检查是否正处在被吃的位置
- 检查相邻是否有比自己强壮的玩家
- 检查是否有金光法器能击中的目标
- 沿着当前位置的四个方向遍历，检查是否有玩家
- 检查可击中目标的可能逃跑路线（即判断是否必中）
- 对可击中玩家的可能逃跑路径做判断
- 简单权衡部分决策价值

0.1.0版本已实现的功能性功能为：

- 本地自动截取输入 json 中对局初始化数据
- 本地进行调试并能读取必要信息，定义回合数等
- 模块化部分实现函数，规范化函数封装，便于以后链式扩展

Introduce

此版本为该 Bot 的初始版本的进阶版本，在原有程序的基础上，增添了判断细节和程序优化，并加强了模块化的思路，便于后续的功能改进。

现版本增加的功能描述为：

- **储存上一回合所有玩家输入，并在这一回合由两回合的动作猜测其他玩家输入**
- **细化金光法器功能，对每个可达敌人的射击价值进行细化计算，考虑敌我双方力量对比，敌我双方处境，敌我双方预测的下一步行动等等。**
- 每回合在寻找最近的豆子果实的同时判断是否会与其他玩家重合
- 预处理计算出地图上从任意一点X到另一点Y的最短距离以及初始方向
- 进一步封装吃豆子判断和开枪判断，实现模块化，便于后续版本的修改扩展
- 判断当前点是否处于危险（可能被吃或开枪）的状态
- 判断危险时当前是否处于死路（没有方向可以逃跑），有路线则逃跑

已知的未来改进路线：

- 对最优果实的选择不能单纯以距离为判定方式，应该还权衡给每个果实周围的果实密度，考虑沿着哪一个路径能最快获取最多果实，同时权衡果实的竞争者（附近玩家）的多少，以及每个果实的危险性，被其他玩家提前吃掉的可能性
- 金光法器对目标的判断应该更加智能，考虑目标更多的可能动作以及攻击时机。
- 检查是否危险以及如何逃跑的算法需要改进，应考虑更多复杂情况

Evaluate

与上个版本相比，**本程序已经改善了大多数情况下的进攻策略**，在对战初期已经尽可能规避被吃的可能性，重点增强了攻击性。对于经由两回合输入判断的大概率可攻击到的玩家，激进的保证在自身体力不危险的情况下让本程序在豆子不十分稀疏的对局中具有一定优势。

现阶段本程序的核心依然依赖 BFS 对地图进行遍历，采用链式的决策思路。

但是现阶段的决策对其他玩家的动作除了可能会被直接攻击或被吃之外未能进行有效预判，考虑的情况还不够多，而且当前所作出的决策还有提高的空间，并没有完全保证最优。

总体现阶段胜率处于中游，多数情况对局良好，但是在豆子较为稀疏的局面下吃豆子策略并不占优，以及整体较小的局面下容易与其他玩家同归于尽，这些情况仍有待改进。

Code

- 游戏决策代码

> 仅贴出关键部分算法代码，略去判定程序本体和执行函数（main）

```
namespace Bot
{
    const int infDis = 0x3fff;
    struct location
    {
        int x, y;
        location() { x = 0; y = 0; }
        location(int x, int y) :x(x), y(y) {}
    };
    struct justify
    {
        int dis;
        int dir;
        bool isDanger;
        int value;
        justify() { dis = infDis; dir = -1; isDanger = false; value = 0; }
        justify(int x, int y) :dis(x), dir(y) { isDanger = false; value = 0; }
    };

    struct valDou
    {
        int num;//当前豆子
        int sum;
    };

    int vis[20][20] = {};//是否访问过
    justify valueMap[20][20] = {};//记录BFS的结果
    int mustShootData[4] = {0,0,0,0};//记录当前位置四个方向上必定能打到的人数，排序：上 右 下 左
    int shootData[4] = {0,0,0,0};//记录当前位置四个方向上能打到的人数，排序：上 右 下 左
    //最优豆子信息
    int douDis = infDis, douDir = -1;
    //最优豆子产生器附近信息
    int genDis = infDis, genDir = -1;
    //最终决策
    int final = -1;
    //叫嚣语句
    string shoutString = "";


    justify valueALL[20][20][20][20] = {};//记录全图BFS结果
    int visALL[20][20] = {};


    //便于本地调试，每次都初始化
    void init(Pacman::GameField& gameField) {
        final = -1;
        douDis = infDis, douDir = -1;
        genDis = infDis, genDir = -1;
        for (int i = 0; i < 4; i++) {
            mustShootData[i] = 0;
            shootData[i] = 0;
        }
        for (int x = 0; x < gameField.height; x++)
        {
            for (int y = 0; y < gameField.width; y++)
            {
                vis[x][y] = 0;
                valueMap[x][y].dis=infDis;
                valueMap[x][y].dir = -1;

                valueMap[x][y].isDanger = false;
```

```
                valueMap[x][y].value = 0;
            }
        }
    }
    //判定危险范围
    void dangerJustify(Pacman::GameField& gameField, int myID)
    {
        for (int player = 0; player < MAX_PLAYER_COUNT; player++)
        {
            //自己当然不危险
            if (player == myID) continue;
            Pacman::Player& aPlayer = gameField.players[player];
            if (gameField.fieldContent[aPlayer.row][aPlayer.col] & Pacman::playerID2Mask[player])
                {
                if (gameField.players[player].strength > gameField.players[myID].strength)
                {
                    //不能直接走到人家嘴里去
                    valueMap[aPlayer.row][aPlayer.col].isDanger = true;
                    //把坏家伙旁边的可达点也设为危险，避免傻屌行为暴毙
                    for (int dir = 0; dir < 4; dir++) {
                        if (!(gameField.fieldStatic[aPlayer.row][aPlayer.col] &
Pacman::direction2OpposingWall[(Pacman::Direction)(dir)])) {
                            valueMap[(aPlayer.row + Pacman::dy[dir] + gameField.height) %
gameField.height][(aPlayer.col + Pacman::dx[dir]+ gameField.width) % gameField.width].isDanger = true;
                        }
                    }
                }

            }
        }
    }

    //预处理图上任意两点之间距离及第一步方向
    void BFSALL(Pacman::GameField& gameField)
    {
        for (int I = 0; I < gameField.height; I++)
        {
            for (int J = 0; J < gameField.width; J++)
            {
                queue <location>q;
                while (!q.empty()) q.pop();
                memset(visALL, 0, sizeof(visALL));
                visALL[I][J] = 1;
                q.push(location(I, J));
                if (!valueALL[I][J][I][J].isDanger)
                    valueALL[I][J][I][J].dis = 0;
                location u, v;
                while (!q.empty())
                {
                    u = q.front();
                    q.pop();
                    for (int i = 0; i < 4; i++)
                    {
                        v.x = u.x + Pacman::dy[i];
                        v.y = u.y + Pacman::dx[i];
                        //边界判定
                        if (v.x < 0) v.x += gameField.height;
                        if (v.x >= gameField.height) v.x -= gameField.height;
                        if (v.y < 0) v.y += gameField.width;
                        if (v.y >= gameField.width) v.y -= gameField.width;
                        //(u.x,u.y)到(v.x,v.y)方向上没有墙，(v.x,v.y)不是豆子产生器，(v.x,v.y)没走过
                        if (!(gameField.fieldStatic[u.x][u.y] &
Pacman::direction2OpposingWall[(Pacman::Direction)(i)])
                            && !visALL[v.x][v.y])
                        {
                            q.push(v);
                            visALL[v.x][v.y] = 1;
                            //对于初始点要特殊处理
                            if (u.x == I && u.y == J) valueALL[I][J][v.x][v.y].dis = 1;
                            else valueALL[I][J][v.x][v.y].dis = valueALL[I][J][u.x][u.y].dis + 1;
```

```
                            if (u.x == I && u.y == J) valueALL[I][J][v.x][v.y].dir = i;//起始点方向记录
                            else valueALL[I][J][v.x][v.y].dir = valueALL[I][J][u.x][u.y].dir;//传递方向
                        }
                    }
                }
            }
        }

    //BFS遍历图上所有点
    void BFSd(Pacman::GameField& gameField, int myID, int nowx, int nowy)
    {
        queue <location>q;
        while (!q.empty()) q.pop();
        memset(vis, 0, sizeof(vis));
        vis[nowx][nowy] = 1;
        q.push(location(nowx, nowy));
        if(!valueMap[nowx][nowy].isDanger)
        valueMap[nowx][nowy].dis = 0;
        location u, v;
        while (!q.empty())
        {
            u = q.front();
            q.pop();
            for (int i = 0; i < 4; i++)
            {
                v.x = u.x + Pacman::dy[i];
                v.y = u.y + Pacman::dx[i];
                //边界判定
                if (v.x < 0) v.x += gameField.height;
                if (v.x >= gameField.height) v.x -= gameField.height;
                if (v.y < 0) v.y += gameField.width;
                if (v.y >= gameField.width) v.y -= gameField.width;

                //(u.x,u.y)到(v.x,v.y)方向上没有墙，(v.x,v.y)不是豆子产生器，(v.x,v.y)没走过
                if (!(gameField.fieldStatic[u.x][u.y] &
Pacman::direction2OpposingWall[(Pacman::Direction)(i)])
                    && !vis[v.x][v.y])
                {
                    //危险地带当然不能走，直接跳过可以理解成让程序绕路
                    if (valueMap[v.x][v.y].isDanger) continue;
                    q.push(v);
                    vis[v.x][v.y] = 1;
                    //对于初始点要特殊处理，否则当所在点为危险时，其他所有点都是inf啦
                    if (u.x == nowx && u.y == nowy) valueMap[v.x][v.y].dis = 1;
                    else valueMap[v.x][v.y].dis = valueMap[u.x][u.y].dis + 1;

                    if (u.x == nowx && u.y == nowy) valueMap[v.x][v.y].dir = i;//起始点方向记录
                    else valueMap[v.x][v.y].dir = valueMap[u.x][u.y].dir;//传递方向
                }
            }
        }
    }

    //遍历四个方向看看有没有人可以打
    int Shot(Pacman::GameField& gameField, int myID, string& shoutString, Pacman::Player& me, int
final)
    {
        int shootDir = -1;
        int maxTargetNum = 0;
        for (int dir = 0; dir < 4; dir++) {
            int r = me.row;
            int c = me.col;
            int distance = 0;
            //如果不开枪下回合自己的位置
            int nextR = me.row;
            int nextC = me.col;
            if (final != -1) {
                nextR = (me.row + Pacman::dy[final] + gameField.height) % gameField.height;
                nextC = (me.col + Pacman::dx[final] + gameField.width) % gameField.width;
```

```
            }
            while (!(gameField.fieldStatic[r][c] & Pacman::direction2OpposingWall[dir]))
            {
                distance++;
                r = (r + Pacman::dy[dir] + gameField.height) % gameField.height;
                c = (c + Pacman::dx[dir] + gameField.width) % gameField.width;

                // 如果转了一圈回来……
                if (r == me.row && c == me.col)
                    break;
                if (gameField.fieldContent[r][c] & Pacman::playerMask)
                    for (int player = 0; player < MAX_PLAYER_COUNT; player++)
                        if (player != myID && (gameField.fieldContent[r][c] &
Pacman::playerID2Mask[player]))
                        {
                            //记录自己是不是躲不开
                            bool canNotHide = (gameField.fieldStatic[me.row][me.col] &
Pacman::direction2OpposingWall[(dir + 1) % 4])
                                && (gameField.fieldStatic[me.row][me.col] &
Pacman::direction2OpposingWall[(dir + 3) % 4]);

                            //记录下回合是不是也躲不开
                            bool nextCanNotHide = (final == -1)
                                ||
                                ((final == dir || final == (dir + 2) % 4)
                                    && (gameField.fieldStatic[nextR][nextC] &
Pacman::direction2OpposingWall[(dir + 1) % 4])
                                    && (gameField.fieldStatic[nextR][nextC] &
Pacman::direction2OpposingWall[(dir + 3) % 4]));
                            //先记录此方向有人
                            shootData[dir]++;
                            //检测此玩家是不是跑不掉了
                            if (
                                (gameField.fieldStatic[r][c] & Pacman::direction2OpposingWall[(dir +
1) % 4])
                                && (gameField.fieldStatic[r][c] & Pacman::direction2OpposingWall[(dir
+ 3) % 4])
                                ) {
                                //如果自己也没路跑，并且对方比自己强大或势均力敌，为避免互射致死，走为上计
                                //if ((gameField.fieldStatic[x.row][x.col] &
Pacman::direction2OpposingWall[(Pacman::Direction)(dir + 1) % 4])
                                //  && (gameField.fieldStatic[x.row][x.col] &
Pacman::direction2OpposingWall[(Pacman::Direction)(dir + 1) % 4])
                                //  && gameField.players[player].strength >=
gameField.players[myID].strength) {
                                //}
                                //else
                                mustShootData[dir]++;
                            }
                            //判断是否很大机会命中
                            //如果我们没得躲,且对方比较弱，那就对射
                            else if (canNotHide
                                && gameField.players[player].strength <=
gameField.players[myID].strength
                                ) {
                                mustShootData[dir]++;
                            }
                            //如果下回合也躲不开，那不管咋样都得打
                            else if (canNotHide && nextCanNotHide) {
                                mustShootData[dir]++;
                            }
                            //如果上回合没动，假定他这回合如果不生成果子也不动
                            else if (
                                (gameField.players[player].lastAction == -1 &&
gameField.generatorTurnLeft != 0)
                                ) {
                                mustShootData[dir]++;
                            }
                            //以下两个判断对方是否刚刚拐进小巷，拐进来的就是送上门啦（除了某些丧心病狂的程序还会躲
                            /*else if (
```

```
                                (gameField.fieldStatic[r][c] & Pacman::direction2OpposingWall[(dir +
1) % 4])
                                &&distance>1
                                && gameField.players[player].lastAction == (dir+1)%4
                                ) {
                                musrShootData[dir]++;
                            }
                            else if (
                                (gameField.fieldStatic[r][c] & Pacman::direction2OpposingWall[(dir +
3) % 4])
                                && distance > 1
                                && gameField.players[player].lastAction == (dir +3) % 4){
                                musrShootData[dir]++;
                            }*/
                        }
                }
                //挑出目标最多的方向打
                if (mustShootData[dir] > maxTargetNum) {
                    shootDir = dir + 4;
                    maxTargetNum = mustShootData[dir];
                }
            }
            //有人可以打当然要打啦(前提是不处在危险之中)
            if (shootDir != -1 && (gameField.SKILL_COST < gameField.players[myID].strength)) {
                if (!valueMap[me.row][me.col].isDanger) {
                    final = shootDir;
                    shoutString = "我射";
                }
            }
            //如果到了死路，路被封死了
            if (final == -1 && valueMap[me.row][me.col].isDanger) {
                //能跑先跑
                for (int dir = 0; dir < 4; dir++) {
                    if ((gameField.fieldStatic[me.row][me.col] & Pacman::direction2OpposingWall[dir]) &&
                        (valueMap[(me.row + Pacman::dy[dir] + gameField.height) % gameField.height]
[(me.col + Pacman::dx[dir] + gameField.width) % gameField.width].dis != infDis)) {
                        final = dir;
                    }
                }
                //实在不能跑就拼啦
                if (final == -1 && (gameField.SKILL_COST < gameField.players[myID].strength)) {
                    int maxShootNum = 0;
                    for (int dir = 0; dir < 4; dir++) {
                        if (maxShootNum < shootData[dir]) {
                            maxShootNum = shootData[dir];
                            shootDir = dir + 4;
                            shoutString = "跟你拼啦！";
                        }
                    }
                    final = shootDir;
                }
            }
            return final;
        }

        //吃豆子
        int Eat(Pacman::GameField& gameField, int myID, string& shoutString, int final, int& douDis, int&
douDir, int& genDis, int& genDir, int X, int Y)
        {
            //枚举图上所有点找最近的豆子
            for (int i = 0; i < gameField.height; i++)
                for (int j = 0; j < gameField.width; j++)
                {

                    //暂时大果子小果子都要，并且判定是否有人重叠于果子上
                    if (((gameField.fieldContent[i][j] & Pacman::smallFruit)
                        || (gameField.fieldContent[i][j] & Pacman::largeFruit))
                        && (!(gameField.fieldContent[i][j] & Pacman::playerMask)))
                        if ((douDis >= valueALL[X][Y][i][j].dis) && !(valueALL[X][Y][i][j].dis == infDis))
                        {
```

```
                        if (douDis > valueALL[X][Y][i][j].dis) {
                            douDis = valueALL[X][Y][i][j].dis;
                            douDir = valueALL[X][Y][i][j].dir;
                        }
                        //如果遇到一样近的，那么随机选择要不要选择此果实，防止出现两人一直重叠
                        else {
                            if (rand() % 2)
                            {
                                douDis = valueALL[X][Y][i][j].dis;
                                douDir = valueALL[X][Y][i][j].dir;
                            }
                        }
                    }
            }
        //暂时先决定去吃豆子
        if (douDir != infDis) {
            shoutString = "恰恰恰~";
            final = douDir;
        }
        if (douDis == infDis)
        {
            //找最近的豆子产生器（先粗略找一下,只有场地上没豆子的时候会去找）
            for (int i = 0; i < MAX_GENERATOR_COUNT; i++)
            {
                justify* generatorSide[4];
                generatorSide[0] = &(valueALL[X][Y][(gameField.generators[i].row + gameField.height) %
gameField.height][(gameField.generators[i].col + gameField.width - 1) % gameField.width]);
                generatorSide[1] = &(valueALL[X][Y][(gameField.generators[i].row + gameField.height) %
gameField.height][(gameField.generators[i].col + 1 + gameField.width) % gameField.width]);
                generatorSide[2] = &(valueALL[X][Y][(gameField.generators[i].row + 1 +
gameField.height) % gameField.height][(gameField.generators[i].col) % gameField.width]);
                generatorSide[3] = &(valueALL[X][Y][(gameField.generators[i].row - 1 +
gameField.height) % gameField.height][(gameField.generators[i].col) % gameField.width]);
                for (int pos = 0; pos < 4; pos++)
                {
                    if (genDis > (*generatorSide[pos]).dis)
                    {
                        genDis = (*generatorSide[pos]).dis;
                        genDir = (*generatorSide[pos]).dir;
                    }
                }
            }
            //没豆子吃辣。那就去等着
            if (genDis != infDis) {
                shoutString = "还有的吃";
                final = genDir;
            }
        }
        return final;
    }

    int calc(Pacman::GameField& gameField, int myID)
    {
        init(gameField);
        Pacman::Player& me = gameField.players[myID];

        dangerJustify(gameField, myID);
        BFSd(gameField, myID, me.row, me.col);
        //遍历全图预处理
        BFSALL(gameField);

        final = Eat(gameField, myID, shoutString, final,douDis,douDir,genDis,genDir,me.row,me.col);

        final = Shot(gameField, myID, shoutString, me, final);


        //最后一道防线，防止出现违规输出
        if (final >= -1 && final <= 7)
        {
            if (final == -1) shoutString = "敌不动我不动敌";
```

```
            return final;
        }
        else {
            shoutString = "bugggg了";
            return -1;
        }
    }


    string shout() {
        return shoutString;
    }
}
```

- 工程部分调试代码

  > 工程中便于本地调试的执行代码

```
namespace Helpers
{

    double actionScore[9] = {};


    void LocalPlay
    (Pacman::GameField& gameField, int myID,int countLimit)
    {
        int count = 0, myAct = -1;
        while (true)
        {
            for (int i = 0; i < MAX_PLAYER_COUNT; i++)
            {
                if (gameField.players[i].dead)
                    continue;
                gameField.actions[i] = (Pacman::Direction)Bot::calc(gameField, i);;
            }

            if (count == 0)
                myAct = gameField.actions[myID];

            // 演算一步局面变化
            gameField.DebugPrint();
            // NextTurn返回true表示游戏没有结束
            bool hasNext = gameField.NextTurn();
            count++;
            cout << count << endl;
            //限制下最大回合数
            if (count >= countLimit) {
                break;
            }
            if (!hasNext)
                break;
        }

        // 计算分数

        int total = 0;
        for (int _ = 0; _ < MAX_PLAYER_COUNT; _++)
            total += gameField.players[_].strength;

        if (total != 0)
            actionScore[myAct + 1] += (10000 * gameField.players[myID].strength / total) / 100.0;

        // 恢复游戏状态到最初（就是本回合）
        while (count-- > 0)
            gameField.PopState();
    }
}
```

Version 0.3.0（1.0.0）

Review

0.2.0 已实现游戏内功能描述为：

- **储存上一回合所有玩家输入，并在这一回合由两回合的动作猜测其他玩家输入**
- **细化金光法器功能，对每个可达敌人的射击价值进行细化计算，考虑敌我双方力量对比，敌我双方处境，敌我双方预测的下一步行动等等。**
- 每回合寻找离自身可达路径最短的果实
  - 利用 BFS 算法计算出到达每一个果实的最短路径长，储存在每一个可达点上
- 在每一个可达点上还会储存此最短路径需求的第一步方向，方便作出决策时快速确定应前进方向。
- 每回合寻找离自身最近的果实生成器
- 当找不到可达果实时自动前往最近的果实生成器旁的随机位置
- 检查是否正处在被吃的位置
- 检查相邻是否有比自己强壮的玩家
- 检查是否有金光法器能击中的目标
- 沿着当前位置的四个方向遍历，检查是否有玩家
- 检查可击中目标的可能逃跑路线（即判断是否必中）
- 对可击中玩家的可能逃跑路径做判断
- 简单权衡部分决策价值

Introduce

0.3.0（本版本）实现功能：

- 对地图中具有特征的坐标（如豆子，其他玩家等坐标）进行了以其为起点的BFS算法的更新，**使得Bot能获取对手的视角信息与进行下文的"豆子附加价值"计算。**
- **对当前最优豆子的价值判断，从距离最近转移到"豆子附加价值"。**计算公式如下：

$$Value[i] = CloseNum[i] - Distance[i]$$

$$i\text{豆子价值} = \text{豆子周围（暂定}3\text{格内）的豆子数} - \text{当前到}i\text{豆子的距离}$$

- 以对手视角考虑所有对手可能的下一步动作，规避与强大对手的交锋。
- DQN 训练的数据预处理与Python裁判程序完成
- DQN 模型building

Evaluate

本版本小组成员认为已经达到了有限开发时间内能做到的，对于对局中局部最优价值的完美诠释。

同时也预示着我们小组接下来的改进方向将进行转变，这后文再叙。

在这个版本中，我们在传统的BFS判断最近豆子中，**利用嵌套BFS与最小生成树算法，对每个豆子周围的豆子数与周围豆子到当前豆子的距离的总和进行了计算。**使得Bot会向豆子收益多的地区不断靠近，保证在交锋较少的僵局中（这样的局面广泛存在于1100分以上的10*10以上地图）能获得优势。

在整体算法上，我们加深了对0.2.0版本中实现的，对所有玩家上一回合输入的存取，对加入对对手下一步动作的预测（如进入拐角，死路等），对金光法器的命中率有了很大的提升，而这对于小地图的开局时的初期博弈非常重要。

在较长期的对局观察中，我们注意到，在1100分到1250分这个分段中，大量的玩家都注重于局部最优的价值判断算法，启发性算法或类学习算法（MCTS或Q-learn）因为其实现难度以及对于核心算法的极度依赖，导致其基本拥有较差的稳定性，这样的算法或在天梯底层（完成度不高，被更加稳定和可控的局部算法完虐），或就是20名以内（对算法的优化和设计很成功，超越了局部最优类算法）。

但与此同时，我们也意识到了局部算法的局限性，不管是对豆子价值的判断，或是对金光法器是否使用的局部博弈中，进一步的发展方向都是考虑更多的回合，考虑更多的前瞻后瞻因素，如被追杀的处理，对某一敌方的行为学习等等。如果还继续把所有考虑因素归一成局部价值，显然是一个1000MS内无法解决的问题。

所以在0.3.0完成后的测试时期，小组严肃的讨论了对于启发式与类学习算法的可行性。

1.0.0版本存在以下三种可行途径：

1. MCTS类算法：包括狭义蒙特卡洛树搜索，决策树等（Pass）

- 此类算法最大的问题来自于对节点树的构建，Pacman游戏本身决定了简单的MAXMIN算法是不可行的，且因为时间复杂度限制，对所有对手动作几乎只能进行纯随机模拟，显然这样对真实游戏环境拟合度非常低，且单局游戏可以真正可达的节点相比棋类博弈要少得多（因为单单一回合的随机模拟计算实测在平台上已经占用2ms）。

2. CNN神经网络（待定）
   - 卷积神经网络对于这类具有图深度的问题（Pacman中每个坐标有众多属性，可以转化为图深度）有一定拟合度，但是在理论阶段遇到的最大问题是，简单的CNN本身并不能区分图深度之间的关系，它更适合对于图像类灰度等像素进行训练，对于Pacman，CNN难以区分“玩家”维度和“豆子”维度以及“墙”维度，可以预见的是如果直接将所有图深度化为独热向量输入，CNN训练出的网络可能连简单的避障都做不到。
   - 当前设想下，CNN也只能是对局部最优的无限逼近拟合（因为训练数据本身可能只能输入一回合数据，标签为本回合排名前几的Bot作出的动作），就算能成功达到预想效果，也无法做到对全局的把握。
   - 但是的确CNN是对Pacman问题本身较为拟合的一种神经网络，但是可能需要在底层进行一部分约束。
3. DQN/Q-learn 强化学习（可能可行？）
   - 强化学习本身的实现难度大于CNN，但在pytorch强大的张量模块和损失函数计算模块的加持下，真正的实现难度其实体现在对超参的调整上。
   - 当前一种可行的设想是将DQN化作局部最优的一部分。
   -
   - 最有可能实现的一种无监督网络。

Code

> 0.3.0局部算法方面的新代码（相比0.2.0不变部分不予展示）

```
//预处理图上任意两点之间距离及第一步方向
    void BFSALL(Pacman::GameField& gameField)
    {
        for (int I = 0; I < gameField.height; I++)
        {
            for (int J = 0; J < gameField.width; J++)
            {
                queue <location>q;
                while (!q.empty()) q.pop();
                memset(visALL, 0, sizeof(visALL));
                visALL[I][J] = 1;
                q.push(location(I, J));
                if (!valueALL[I][J][I][J].isDanger)
                    valueALL[I][J][I][J].dis = 0;
                location u, v;
                while (!q.empty())
                {
                    u = q.front();
                    q.pop();
                    for (int i = 0; i < 4; i++)
                    {
                        v.x = u.x + Pacman::dy[i];
                        v.y = u.y + Pacman::dx[i];
                        //边界判定
                        if (v.x < 0) v.x += gameField.height;
                        if (v.x >= gameField.height) v.x -= gameField.height;
                        if (v.y < 0) v.y += gameField.width;
                        if (v.y >= gameField.width) v.y -= gameField.width;
                        //(u.x,u.y)到(v.x,v.y)方向上没有墙，(v.x,v.y)不是豆子产生器，(v.x,v.y)没走过
                        if (!(gameField.fieldStatic[u.x][u.y] &
Pacman::direction2OpposingWall[(Pacman::Direction)(i)])
                            && !visALL[v.x][v.y])
                        {
                            q.push(v);
                            visALL[v.x][v.y] = 1;
                            //对于初始点要特殊处理
                            if (u.x == I && u.y == J) valueALL[I][J][v.x][v.y].dis = 1;
                            else valueALL[I][J][v.x][v.y].dis = valueALL[I][J][u.x][u.y].dis + 1;
                            if (u.x == I && u.y == J) valueALL[I][J][v.x][v.y].dir = i;//起始点方向记录
                            else valueALL[I][J][v.x][v.y].dir = valueALL[I][J][u.x][u.y].dir;//传递方向
                        }
                    }
                }
            }
        }
```

```
    }

    //BFS遍历图上所有点
    void BFSd(Pacman::GameField& gameField, int myID, int nowx, int nowy)
    {
        queue <location>q;
        while (!q.empty()) q.pop();
        memset(vis, 0, sizeof(vis));
        vis[nowx][nowy] = 1;
        q.push(location(nowx, nowy));
        if(!valueMap[nowx][nowy].isDanger)
        valueMap[nowx][nowy].dis = 0;
        location u, v;
        while (!q.empty())
        {
            u = q.front();
            q.pop();
            for (int i = 0; i < 4; i++)
            {
                v.x = u.x + Pacman::dy[i];
                v.y = u.y + Pacman::dx[i];
                //边界判定
                if (v.x < 0) v.x += gameField.height;
                if (v.x >= gameField.height) v.x -= gameField.height;
                if (v.y < 0) v.y += gameField.width;
                if (v.y >= gameField.width) v.y -= gameField.width;

                //(u.x,u.y)到(v.x,v.y)方向上没有墙，(v.x,v.y)不是豆子产生器，(v.x,v.y)没走过
                if (!(gameField.fieldStatic[u.x][u.y] &
Pacman::direction2OpposingWall[(Pacman::Direction)(i)])
                    && !vis[v.x][v.y])
                {
                    //危险地带当然不能走，直接跳过可以理解成让程序绕路
                    if (valueMap[v.x][v.y].isDanger) continue;
                    q.push(v);
                    vis[v.x][v.y] = 1;
                    //对于初始点要特殊处理，否则当所在点为危险时，其他所有点都是inf啦
                    if (u.x == nowx && u.y == nowy) valueMap[v.x][v.y].dis = 1;
                    else valueMap[v.x][v.y].dis = valueMap[u.x][u.y].dis + 1;

                    if (u.x == nowx && u.y == nowy) valueMap[v.x][v.y].dir = i;//起始点方向记录
                    else valueMap[v.x][v.y].dir = valueMap[u.x][u.y].dir;//传递方向
                }
            }
        }
    }
int Eat(Pacman::GameField& gameField, int myID, string& shoutString, int final, int& douDis, int&
douDir, int& genDis, int& genDir, int X, int Y)
    {
        valDou tot[405];
        int NumDou = 0; location AnsEat;//可达豆子总数，最终选择吃的豆子坐标
        for (int i = 1; i <= 400; i++) tot[i].x = tot[i].y = tot[i].big = tot[i].dist = tot[i].numNear
= tot[i].totNear = tot[i].sum = 0;

        //枚举图上所有点先找出所有可达的豆子
        for (int i = 0; i < gameField.height; i++)
            for (int j = 0; j < gameField.width; j++)
            {
                if (((gameField.fieldContent[i][j] & Pacman::smallFruit)
                    || (gameField.fieldContent[i][j] & Pacman::largeFruit))
                    && (!(gameField.fieldContent[i][j] & Pacman::playerMask)))
                {
                    ++NumDou;
                    if (gameField.fieldContent[i][j] & Pacman::largeFruit) tot[NumDou].big = 1;
                    tot[NumDou].x = i; tot[NumDou].y = j; tot[NumDou].dist = valueALL[X][Y][i][j].dis;
                }
            }
        //再对每个豆子进行价值计算，考虑的因素包括与当前点距离，豆子周围“接近”的豆子数量
        for (int i = 1; i < NumDou; i++)
        {
```

```
            for (int j = i+1; j <= NumDou; j++)
            {
                if (valueALL[tot[i].x][tot[i].y][tot[j].x][tot[j].y].dis <= Near)
                {
                    tot[i].numNear++; tot[j].numNear++;
                    tot[i].totNear += valueALL[tot[i].x][tot[i].y][tot[j].x][tot[j].y].dis;
                    tot[j].totNear += valueALL[tot[i].x][tot[i].y][tot[j].x][tot[j].y].dis;
                }
            }
        }

        //计算从目标豆子出发到达所有距离接近的豆子的最短路径（即求一个最小生成树）
        int Node[405], NumNode=0,fNode[405];
        for (int i = 1; i <= NumDou; i++)
        {
            memset(Edge, 0, sizeof(Edge));
            memset(Node, 0, sizeof(Node));
            int NumEdge = 0, tmp = 0;
            NumNode = 0;
            Node[++NumNode] = i;
            for (int j = 1; j <= NumNode; j++)
            {
                fNode[j] = j;//标记集合
            }
            for (int j = 1; j <= NumDou; j++)
            {
                if (i == j) continue;
                if (valueALL[tot[i].x][tot[i].y][tot[j].x][tot[j].y].dis <= Near) Node[++NumNode] = j;
            }

            for (int j = 1; j <= NumNode; j++)
            {
                for (int k = j + 1; k <= NumNode; k++)
                {
                    Edge[++NumEdge].u = Node[j];
                    Edge[NumEdge].v = Node[k];
                    Edge[NumEdge].w = valueALL[tot[Node[j]].x][tot[Node[j]].y][tot[Node[k]].x]
[tot[Node[k]].y].dis;
                }
            }

            sort(Edge + 1, Edge + NumEdge + 1, cmp);

            for (int j = 1; j <= NumEdge; j++)
            {
                int n1 = fNode[Edge[j].u];
                int n2 = fNode[Edge[j].v];//得到两点集合编号
                if (n1 != n2)//不同就加入边
                {
                    tot[i].Nearest += Edge[j].w;
                    for (int k = 1; k <= NumNode; k++)
                    {
                        if (fNode[k] == n2) fNode[k] = n1;//合并集合
                    }
                    tmp++;
                }
                if (tmp == NumNode - 1) break;
            }
        }


        for (int i = 1; i <= NumDou; i++)
        {
            tot[i].sum = tot[i].numNear - tot[i].dist + 0.01 * tot[i].big;//分值为距离接近的豆子数-距离，认
为同等情况下大豆子优于小豆子
        }
        for (int i = 1; i <= NumDou; i++)//对豆子价值进行排序
        {
            for (int j = i + 1; j <= NumDou; j++)
            {
```

```
                if (tot[i].sum < tot[j].sum)  swap(tot[i], tot[j]);
            }
        }
        if (tot[1].dist != infDis)
        {
            douDis = valueALL[X][Y][tot[1].x][tot[1].y].dis;
            douDir = valueALL[X][Y][tot[1].x][tot[1].y].dir;
        }

        //暂时先决定去吃豆子
        if (douDir != infDis) {
            shoutString = "恰恰恰~";
            final = douDir;
        }
        if (douDis == infDis)
        {
            //找最近的豆子产生器（先粗略找一下,只有场地上没豆子的时候会去找）
            for (int i = 0; i < MAX_GENERATOR_COUNT; i++)
            {
                justify* generatorSide[4];
                generatorSide[0] = &(valueALL[X][Y][gameField.generators[i].row]
[(gameField.generators[i].col + gameField.width - 1) % gameField.width]);
                generatorSide[1] = &(valueALL[X][Y][gameField.generators[i].row ]
[(gameField.generators[i].col + 1 + gameField.width) % gameField.width]);
                generatorSide[2] = &(valueALL[X][Y][(gameField.generators[i].row + 1 +
gameField.height) % gameField.height][(gameField.generators[i].col)]);
                generatorSide[3] = &(valueALL[X][Y][(gameField.generators[i].row - 1 +
gameField.height) % gameField.height][(gameField.generators[i].col)]);
                for (int pos = 0; pos < 4; pos++)
                {
                    if (genDis > (*generatorSide[pos]).dis)
                    {
                        genDis = (*generatorSide[pos]).dis;
                        genDir = (*generatorSide[pos]).dir;
                    }
                }
            }
            //没豆子吃辣。那就去等着
            if (genDis != infDis) {
                shoutString = "还有的吃";
                final = genDir;
            }
        }
        return final;
    }
```

> 1.0.0方面已经完成部分

```
//torch模型引用
struct Net : torch::nn::Module {
    Net() {
        // Construct and register two Linear submodules.
        fc1 = register_module("fc1", torch::nn::Linear(2433, 2048));
        fc2 = register_module("fc2", torch::nn::Linear(2048, 1024));
        fc3 = register_module("fc3", torch::nn::Linear(1024, 128));
        fc4 = register_module("fc4", torch::nn::Linear(128,9))
    }

    // Implement the Net's algorithm.
    torch::Tensor forward(torch::Tensor x) {
        // Use one of many tensor manipulation functions.
        x = torch::relu(fc1->forward(x.reshape({ x.size(0), 784 })));
        x = torch::dropout(x, /*p=*/0.5, /*train=*/is_training());
        x = torch::relu(fc2->forward(x));
        x = torch::log_softmax(fc3->forward(x), /*dim=*/1);
        return x;
    }

    // Use one of many "standard library" modules.
```

```
    torch::nn::Linear fc1{ nullptr }, fc2{ nullptr }, fc3{ nullptr }, fc4{ nullptr };
};
```

> 裁判程序复刻，用于 DQN获取当前动作得分

```
# 裁判程序复刻

import json
import math


class Player:
    def __init__(self, idi=0, x=0, y=0):
        self.id = idi
        self.strength = 1
        self.x = x
        self.y = y
        self.enhance = 0
        self.dead = False
class Generator:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
class Pacman:
    state_cnt = 0
    players = [Player() for i in range(4)]
    generators = [Generator() for j in range(4)]
    generateTurnCount = 0
    dx = [-1, 0, 1, 0, -1, 1, 1, -1]
    dy = [0, 1, 0, -1, 1, 1, -1, -1]
    width = 1
    height = 1
    actions = [-1] * 4
    aliveNum = 4

    smallScore = 0
    turnId = 0
    skillCost = 4
    gameStatic = []
    gameContent = []
    myId = 0
    generatorLeftTurns = 20
    def actionValid(self,pl, move):
        if move == -1:
            return True
        if move >= 4:
            return move < 8 and players[pl].strength > skillCost
        return move >= 0 and move < 4 and not(gameStatic[players[pl].x][players[pl].y] & (1 << move))
    def nextTurn(self):
        # invalid input
        for i in range(4):
            if not(players[i].dead):
                ac = actions[i]
                if ac == -1:
                    continue
                if not(actionValid(i, ac)):
                    gameContent[players[i].x][players[i].y] &= (~(1 << players[i].idi))

                    players[i].strength = 0
                    players[i].dead = True

                    aliveNum -= 1
                elif ac < 4:
                    nx = (players[i].x + dx[ac] + height) % height
                    ny = (players[i].y + dy[ac] + width) % width
                    nxtp = gameContent[nx][ny]
                    if nxtp & (1 | 2 | 4 | 8):
                        for j in range(4):
```

```
                        if (nxtp & (1 << j)) and players[j].strength > players[i].strength:

                            actions[i] = -1

        for i in range(4):
            if players[i].dead or actions[i] == -1 or actions[i] >= 4:
                continue
            ac = actions[i]
            gameContent[players[i].x][players[i].y] &= (~(1 << i))
            npx = (players[i].x + dx[ac] + height) % height
            npy = (players[i].y + dy[ac] + width) % width
            players[i].x, players[i].y = npx, npy
            gameContent[npx][npy] |= (1 << i)
        for i in range(4):
            if players[i].dead:
                continue
            px, py = players[i].x, players[i].y
            player, cnt = 0, 0
            con_players = [0] * 4
            for player in range(4):
                if gameContent[px][py] & (1 << player):
                    con_players[cnt] = player
                    cnt += 1
            if cnt > 1:
                for k in range(cnt):
                    for j in range(cnt - k - 1):
                        if players[con_players[j]].strength < players[con_players[j + 1]].strength:

                            con_players[j], con_players[j + 1] = con_players[j + 1], con_players[j]

                beg = 0
                for beg in range(1, cnt):
                    if players[con_players[beg - 1]].strength > players[con_players[beg]].strength:

                        break
                    ltstren = 0
                    for k in range(beg, cnt):
                        pid = con_players[k]
                        gameContent[players[pid].x][players[pid].y] &= (~(1 << pid))
                        players[pid].dead = True
                        drop = players[pid].strength / 2
                        ltstren += drop
                        players[pid].strength = ceil(players[pid].strength - drop)
                        aliveNum -= 1
                    inc = int(ltstren / beg)
                    for k in range(beg):
                        pid = con_players[k]
                        players[pid].strength += inc
            for i in range(4):
                if players[i].dead or actions[i] < 4:
                    continue
                players[i].strength -= skillCost
                dirc = actions[i] - 4
                r, c, pl = players[i].x, players[i].y, 0
                while not(gameStatic[r][c] & (1 << dirc)):
                    r = (r + dx[dirc] + height) % height
                    c = (c + dy[dirc] + width) % width
                    if r == players[i].x and c == players[i].y:
                        break
                    if (gameContent[r][c] & 15):
                        for pl in range(4):
                            if (gameContent[r][c] & (1 << pl)):
                                players[pl].strength -= skillCost * 1.5
                                players[i].strength += skillCost * 1.5
        for i in range(4):
            if players[i].dead or players[i].strength > 0:
                continue
            gameContent[players[i].x][players[i].y] &= ~(1 << i)
            players[i].dead = True
            players[i].strength = 0
```

```
            aliveNum -= 1
        generatorLeftTurns -= 1
        if generatorLeftTurns == 0:
            generatorLeftTurns += 20
            for i in range(4):
                for d in range(8):
                    r = (generators[i].x + dx[d] + height) % height
                    c = (generators[i].y + dy[d] + width) % width
                    if (gameStatic[r][c] & 16) or (gameContent[r][c] & (16 | 32)):
                        continue
                    gameContent[r][c] |= 16
                    smallScore += 1
        for i in range(4):
            if players[i].dead:
                continue
            pr, pc = players[i].x, players[i].y
            if gameContent[pr][pc] & 15 & (~(1 << i)):
                continue
            if (gameContent[pr][pc] & 16):
                gameContent[pr][pc] &= (~16)
                players[i].strength += 1
                smallScore -= 1
            elif gameContent[pr][pc] & 32:
                gameContent[pr][pc] &= (~32)
                if players[i].enhance == 0:
                    players[i].strength += 10
                players[i].enhance += 10
        for i in range(4):
            if players[i].dead:
                continue
            if players[i].enhance > 0:
                players[i].enhance -= 1
                if players[i].enhance == 0:
                    players[i].strength -= 10
        for i in range(4):
            if players[i].dead or players[i].strength > 0:
                continue
            gameContent[players[i].x][players[i].y] &= (~(1 << i))
            players[i].dead = True
            players[i].strength = 0
            aliveNum -= 1
        turnId += 1
        if aliveNum <= 1:
            for i in range(4):
                if not(players[i].dead):
                    players[i].strength += smallScore
                    return False
                if turnId >= 100:
                    return False
                return True
    def __init__(self,data,id):
        my_id = id
        for i in range(height):
            for j in range(width):
                if gameContent[i][j] & 16:
                    smallScore += 1
                if gameContent[i][j] & 1:
                    players[0].x, players[0].y, players[0].idi = i, j, 0
                if gameContent[i][j] & 2:
                    players[1].x, players[1].y, players[1].idi = i, j, 1
                if gameContent[i][j] & 4:
                    players[2].x, players[2].y, players[2].idi = i, j, 2
                if gameContent[i][j] & 8:
                    players[3].x, players[3].y, players[3].idi = i, j, 3
                if gameStatic[i][j] & 16:
                    generators[generateTurnCount].x, generators[generateTurnCount].y = i, j

                    generateTurnCount += 1
    def getInMat(self,_turn, act):
        ind = 0
```

```
        feedlis = [0. for i in range(1223)]
        feedlis[ind] = _turn / 100
        ind += 1
        # my_id
        for i in range(4):
            if i == myId:
                feedlis[ind] = 1
            ind += 1
        # strength
        for i in range(4):
            if players[i].dead == False:
                feedlis[ind] = players[i].strength / 100
            else:
                feedlis[ind] = 0
            ind += 1
        # enhance
        for i in range(4):
            feedlis[ind] = players[i].enhance
            ind += 1
        # each grid
        for i in range(height):
            for j in range(width):
                # wall
                for k in range(4):
                    if gameStatic[i][j] & (1 << k):
                        feedlis[ind] = 1
                    ind = ind + 1
                # player
                for k in range(4):
                    if gameContent[i][j] & (1 << k):
                        feedlis[ind] = 1
                    ind = ind + 1
                # fruit
                if gameContent[i][j] & 16:
                    feedlis[ind] = 1
                ind += 1
                if gameContent[i][j] & 32:
                    feedlis[ind] = 1
                ind += 1
        feedlis.append(act)
        return feedlis
```

> 其他数据预处理部分因为代码完成性原因先不展示。

第一次 POJ 报告

09018316 黄开鸿 第一次报告

POJ 1064

- 题意概述：

  按一定长度切割若干根长短不一的网线，求出最长的切割长度。

- 题意分析

  如果直接暴力搜索，那么切割长度的可能值是0.00到10000.00，也就是达到了10的六次方，而每次验证该长度可不可行又需要遍历最多10000次，明显无法在1000MS内完成。

  经分析，每次验证切割长度的可行性必须遍历所有网线，无法优化，故只能优化尝试切割长度的次数。因为切割长度的选取是有明确的大小关系，**所以可以采用二分法，将原本 O(n) 复杂度的尝试过程缩短到 O(log(n))。**

- 实际代码

```
#include <string>
#include <iomanip>
#include <iostream>
#include <cmath>
```

```
using namespace std;

const int maxN = 10000;//n最大值
const int maxK = 10000;//k最大值

int n = 0;
  int k = 0;

  double arr[10000] ; //储存所有网线


bool isSatisfied(double num){
    int total =0;
    for(int i = 0; i <n;i++){
        total+=arr[i]/num;
    }
    return total>=k;
}

int main(){
    cin>>n>>k;
    int i = 0;
    double maxLength = 0;
    while(n>i){
        cin>>arr[i];
        maxLength = arr[i]>maxLength?arr[i]:maxLength;
        i++;

    }
    double l = 0;double r = maxLength;
    for(int cut = 0;cut<100;cut++){
        double m = (l+r)/2.0;
        if(isSatisfied(m)){
              l = m;
        }
        else{
            r= m;
        }
    if(r-l<0.001){
        break;
    }
    }
    cout<<fixed<<setprecision(2)<<floor(r*100)/100;
    return 0;
}
```

POJ 3714

- 题意概述

  在给定的两个点集中找到距离最近的一组点对，两点来自不同点集

- 题意分析

  直接暴力搜索的复杂度最大为 10 的十次方，在5000ms内明显无法执行完。

  利用分治思想将问题分解，并在小问题中加以优化。实现算法的关键在于利用子问题的返回值（即自身一部分中的最小距离点对），排除部分不可能的解，使得自身问题简化，防止部分无效比较。

  但是该算法的某一子问题解决时，仍使用暴力搜索比较所有可能的可行解（遍历所有与分界点 x 坐标之差不超过左右子问题返回的最小值 d 的点），此处其实也能优化，但是还未能完全实现，现阶段只是多排除了横坐标距离以超过 d 的点对。该题给出 5000ms 时限使得此步不需要完美优化也能在限时内完成，如果限时为标准按 1000ms 则还需优化。

- 实际代码

```
#include <iostream>
#include <cstdio>
#include <iomanip>
```

```cpp
#include <cstring>
#include <algorithm>
#include <cmath>
using namespace std;


const long long INF = 1000000000000;
const int N = 100010;//比100000略大，保证稳定性

struct Node
{
    long long x, y;//储存坐标
    int id;//标记信息，用于判定点集
    Node(long long x = 0, long long y = 0, int id = 0) : x(x), y(y), id(id) {}
    const bool operator<(const Node A) const //重载比较符用于直接调用sort函数
    {
        return x == A.x ? y < A.y : x < A.x;
    }
} node[2 * N];//用以储存所有点

int n;
//计算两点距离
double dis(int a, int b)
{
    return sqrt((double)((node[a].x - node[b].x) * (node[a].x - node[b].x) + (node[a].y - node[b].y) *
(node[a].y - node[b].y)));
}
//递归调用函数
double solve(int l, int r)
{
    //如果递归到只有一个点，返回无穷大
    if (l == r)
        return INF;
    //位运算取中点
    int mid = (l + r) >> 1;
    double a = solve(l, mid);
    double b = solve(mid + 1, r);
    //求下层递归返回的最小距离
    double d = min(a, b);
    //归并两边的递归结果
    for (int i = mid; i >= l; --i)
    {
        //如果左边当前点已经和mid距离大于下层递归结果，break
        if (node[mid].x - node[i].x > d)
            break;
        for (int j = mid + 1; j <= r; ++j)
        {
             //如果右边当前点已经和左边当前点距离大于下层递归结果，break
            if (node[j].x - node[i].x > d)
                break;
            double tmp = dis(i, j);
            //比较两不同点集距离
            if (node[i].id != node[j].id && tmp < d)
                d = tmp;
        }
    }
    return d;
}

int main()
{
    int time;
    cin>>time;
    while (time--)
    {

        cin>>n;
        for (int i = 0; i < n; ++i)
        {
            cin>>node[i].x>>node[i].y;
```

```
            node[i].id = 1;
        }
        for (int i = 0; i < n; ++i)
        {
            cin>>node[i + n].x>>node[i + n].y;
            node[i + n].id = 2;
        }
        sort(node, node + 2 * n);
        double res = solve(0, 2 * n - 1);
        //不知道为啥一定要这样取才对，取注释段代码则 PE
        printf("%.3lf\n", res);
        //cout<<fixed<<setprecision(3)<<res;
    }
}
```

09018330 孙毅远 第一次报告

POJ 1064

- 题意

  有n根棍子可以截取，问要求最后给出K根等长的棍子，求每根棍子的最大长度。保留2位小数（去尾）

  $n \leq 10000, k \leq 10000$

- 题目分析

  由题及数据范围可知$O(n^2)$算法不可行，而保留两位小数，最大长度 $\leq 10^5$容易想到二分长度，对每个长度计算是否满足条件

  精度要保留两位小数，设置单位区间为10^-5，可以满足题目需要

  时间复杂度为$O(nlogt), t = 10^{10}$

- 代码

```
#include<iostream>
#include<cmath>
#include<cstdio>
using namespace std;
int n,k;
double f[10005];
bool p(double t)//检验当前值是否可切出k根
{
    int sum=0;
    for(int i=1;i<=n;i++)
        sum+=(int)(f[i]/t);
    return sum>=k;
}
int main()
{
    cin>>n>>k;
    for(int i=1;i<=n;i++) scanf("%lf",&f[i]);
    double l=0,r=100000,mid=0;
    while(r-l>1e-5)//二分过程
    {
        mid=(l+r)/2;
        if(p(mid)) l=mid;
        else r=mid;
    }
    printf("%.2f\n",floor(r*100)/100);//截尾取整
    return 0;
}
```

POJ3714

- 题意

  有N个核电站和N个特工，均已知坐标，求特工与核电站之间的最短距离,$1 \leq N \leq 10^5$

- 题目分析

对核电站和特工进行标记，然后求平面上的最近点对，具体方法是先对所有点按先x后y排序，把区域划分为两块，分别求两部分内部的最短距离再对两部分之间的求最短距离，递归下去，其中对于两区域之间的距离，肯定大于两部分最短距离的部分无需考虑，时间复杂度为$O(nlogn)$

- 代码

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
const double MD=1e100;
const int MAXN=200005;
int T,n,ys[MAXN];
struct P{
    double x, y;
    bool flag;
}a[MAXN];
double dist(const P&a,const P&b) {
    if(a.flag!=b.flag) return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
    return MD;
}
bool cmp(const P &a,const P&b){
    return a.x<b.x;
}
bool y_cmp(const int&u,const int&v) {
    return a[u].y<a[v].y;
}
double calc(int l,int r){
    if(r-l==1)return dist(a[l], a[r]);
    else if(r-l==2) return min(min(dist(a[l], a[l+1]), dist(a[l], a[l+2])),dist(a[l+1], a[l+2]));
    int mid=(l+r)/2,yn=0;
    double ans=min(calc(l,mid),calc(mid+1, r));//分治
    if(ans==0) return 0;
    for(int i=mid;a[mid].x-a[i].x<ans&&i>=l;i--) ys[yn++]=i;
    int y_mid=yn;
    for(int i=mid+1;a[i].x-a[mid].x<ans&&i<=r;i++) ys[yn++]=i;
    for(int i=0;i<y_mid;i++)
        for(int j=y_mid;j<yn;++j)
            ans=min(ans,dist(a[ys[i]],a[ys[j]]));//对两区域之间的有可能小于当前最小值的点进行比较
    return ans;
}
int main()
{
    cin>>T;
    while(T--)
    {
        scanf("%d",&n);
        for(int i=0;i<n;i++) {
            scanf("%lf%lf",&a[i].x,&a[i].y);
            a[i].flag=false;
        }
        for(int i=n;i<2*n;i++) {
            scanf("%lf%lf",&a[i].x, &a[i].y);
            a[i].flag=true;
        }
        sort(a,a+2*n,cmp);
        printf("%.3f\n", calc(0,2*n-1));
    }
    return 0;
}
```

第二次 POJ 报告

09018316 黄开鸿 第二次报告

POJ 1328

- 题意概述：

> 二维坐标系中第一第二象限有岛屿，岛屿左边为整数，X 轴上能布置覆盖距离固定的雷达，求能覆盖所有岛屿的最小雷达数。

- 题意分析

> 每个岛屿在 X 轴上有个可覆盖范围，其实问题很容易被转化为**求若干线段中的最小公共点个数**。因为线段长度并非离散，并且起终点无上下界，因此以长度为遍历变量的算法必不存在，必然是以线段为研究对象。 运用贪心算法，先按每个岛屿的可选范围的左端点升序排序，然后每次都将待选线段并入已选线段集合中，计算相应的最小需要雷达数。

- 实际代码

```
#include <cmath>
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>

using namespace std;

#define MAXN 1010 //岛屿数的最大值

//岛屿
typedef struct island
{
    int x, y;
} island;

//岛屿可行域
typedef struct
{
    double l, r;
} range;

int n;                //岛屿数
int d;                //雷达范围
island islands[MAXN]; //岛屿数组
range ranges[MAXN];

int cmp(range a, range b)
{
    return a.l < b.l;
}

//计算岛屿可行域，如不可行返回false
bool calcRange(island i, range &r, int d)
{
    if (i.y > d || i.x < 0)
    {
        return false;
    }
    r.l = i.x - sqrt(d * d * 1.0 - i.y * i.y * 1.0);
    r.r = i.x + sqrt(d * d * 1.0 - i.y * i.y * 1.0);
    return true;
}

int main()
{
    int count = 0;
    while (cin >> n)
    {
        count++;
        cin >> d;
        if (n == 0 && d == 0)
            break;
        //不合法输入直接返回 -1
        if (d < 0)
```

```
        {
            cout << "Case " << count << ": " << -1 << endl;
            continue;
        }
        //每次清空岛屿数组
        memset(islands, 0, sizeof(islands));

        bool hasSolution = true;
        //输入岛屿并计算每个岛屿的可行域
        for (int i = 0; i < n; i++)
        {
            cin >> islands[i].x;
            cin >> islands[i].y;
            hasSolution = calcRange(islands[i], ranges[i], d);
        }
        //如果有不可达的岛屿，直接返回-1
        if (!hasSolution)
        {
            cout << "Case " << count << ": " << -1 << endl;
            continue;
        }
        sort(ranges, ranges + n, cmp);
        int res = 1;
        //从第二个岛屿开始遍历，只有一个岛屿必然是需要一个雷达
        range tmp = ranges[0];
        for (int i = 1; i < n; i++)
        {
            if (ranges[i].l > tmp.r)
            {
                res++;
                tmp = ranges[i];
            }
            else if (ranges[i].r < tmp.r)
            {
                tmp = ranges[i];
            }
        }
        cout << "Case " << count << ": " << res << endl;
    }
    return 0;
}
```

POJ 2392

- 题意概述

> 有若干种石头，每种石头有若干个，且每种石头有自己的高度值与可以达到的高度值，求这些石头垒起来能到达的最高高度

- 题意分析

> 对比起普通的背包问题，这次限制最优解的限制条件其实不是一个全局的“重量”限制。从极限情况来看，如果每种石头可以达到的最大高度都无穷大，易证得最大高度便是所有石头高度值之和。因此本题的基本思想一定是把可达高度低的石头先垒。但是如果单纯地按可达高度的顺序放石头，在某些情况下可能导致得不偿失（即有些石头放上去导致更多石头不能放）。

> 因为高度是离散的，于是运用贪心思想，对每一钟石头的可达高度区间按最大可达高度的升序进行遍历。因为目标值为高度，所以只要当前高度有解，必定是当前高度的最优解之一，遍历时只需要考虑当前高度是否已有解（已有解即表示可达高度低的其他石头已经能达到这一高度了，那么我们只需要在之前的基础上继续增加就行，对于当前高度就已是最优解）

- 实际代码

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
#define MAXV 410    //石头种类的最大值
#define MAXM 40100 //代表能到达的最大高度（也是DP数组的大小）
```

```
typedef struct
{
    int height, //单个石头的高度
    maxH, //最高高度
    count;//总共有几个
} Blocks;

Blocks v[MAXV];

int cmp(Blocks x, Blocks y)
{
    return x.maxH < y.maxH;
}


int dp[MAXM]; //用作DP储存数组
int user[MAXM];//标记此高度某种石头已经用的数量

int main()
{
    int i, typeNum, j, Max;
    while (cin>>typeNum)
    {
        for (i = 1; i <= typeNum; i++)
        {
            cin>>v[i].height;
            cin>>v[i].maxH;
            cin>>v[i].count;
        }
        //最石头种类按可达最大高度升序排序，可达高度小的必须先进
        sort(v + 1, v + typeNum + 1, cmp);
        memset(dp, 0, sizeof(dp));
        dp[0] = 1;
        Max = 0; //赋值为0，高度可能为零
        //从可达高度最小的开始遍历
        for (i = 1; i <= typeNum; i++)
        {
            //每次重新初始化已使用数记录数组（因为石头换了一种）
            memset(user, 0, sizeof(user));
            for (j = v[i].height; j <= v[i].maxH; j++)
            {
                if (!dp[j] //如果此高度还未达到最优解
                && dp[j - v[i].height] //且此石头能放上去（即减去此石头高度已经有石头垫着了）
                && user[j - v[i].height] + 1 <= v[i].count)//且石头还没用完
                {
                    dp[j] = 1;//标记此高度已经确定最优解
                    user[j] = user[j - v[i].height] + 1;//标记到这个高度用了多少这种石头
                    if (j > Max)
                        Max = j;
                }
            }
        }
        cout<<Max<<endl;
    }
    return 0;
}
```

POJ 1163

- 题意概述

> 给定一个由数字构成的三角形，第一行有 1 个数字，第二行有2个，以此类推。现有一种路径，从第一行开始，能向左下或右下取值，求路径上数字和最大的路径。

- 题意分析

> 不同元素的最大路径之间有明确的迭代关系，转化为动规问题。从三角形底部开始向上迭代，求取每个位置的最大路径和，即比较当前位置的左下最优解和右下最优解的大小，取最大值再加上当前值即是当前最优解，如此迭代直到得到第一行唯一数字的最优解。 对于数字三角形的存储可以模仿二叉树的线性存储，用一维数组储存二维三角形后通过简单运算二维坐标转换为一维坐标进行访问。

- 实际代码

```
#include <iostream>

using namespace std;
#define max(a, b) (((a) > (b)) ? (a) : (b))
int row = 0;        //行数
int num[50000] = {}; //储存数组
int res[50000] = {}; //DP结果数组

int main()
{
    cin >> row;
    int max = row * (row + 1) / 2 - 1; //最大下标;
    for (int i = 0; i <= max; i++)
    {
        cin >> num[i];
    }
    for (int i = row; i > 0; i--)
    {
        for (int index = 0; index < i; index++)
        {
            int loc = i * (i - 1) / 2 + index ; //当前下标
            if (loc + row > max)                    //如果没有下一行，最优解为当前值
            {
                res[loc] = num[loc];
            }
            else
            {
                //递推公式，当前最优解 = 下层两可选最优解最大值 + 当前值
                res[loc] = max(res[loc + i], res[loc + i + 1]) + num[loc];
            }
        }
    }
    cout << res[0];
    return 0;
}
```

POJ 2533

- 题意概述

> 求一给定数组的最大升序子序列，可以不连续

- 题意分析

> 动规问题。但是与之前的动规问题不同的是，这个问题必须比较所有已知最优解，即当前值之前的所有最长升序子序列长度，才能得出当前最优解。这使得复杂度实际上还是n^2，让人有些怀疑此题使用动规的意义。但是除此之外没有得出多项式时间内可解的算法，在 POJ题解中找到一种单调队列方法，但是实在无法掌握。

- 实际代码

```
#include <iostream>
#define max(a, b) (((a) > (b)) ? (a) : (b))
using namespace std;
int a[1010];
int dp[1010] = {};
int main()
{

    int N = 0; // 数据数
    cin >> N;
    int res = 0; //结果数据
    for (int i = 0; i < N; i++)
    {
```

```
        cin >> a[i];
    }
    for (int i = 0; i < N; i++)
    {
        //初始化当前长度为1
        dp[i] = 1;
        //遍历当前数的前面所有数，每次判断是否可以将当前值插入其头部
        for (int j = 0; j < i; j++)
        {
            if (a[i] > a[j])//如果可以插入
            {
                //当前值权衡是否要插入，插入的话便是 j 项中子序列最后再加一个 i 项
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
    }
    for (int i = 0; i < N; i++)
    {
        if (dp[i] > res)
        {
            res = dp[i];
        }
    }
    cout << res << endl;

    return 0;
}
```

09018330 孙毅远 第二次报告

POJ 2393

- 题意

  一个工厂要供应n周的酸奶，已知每升酸奶保存一周花费s，每周每升酸奶的造价为c，需求量为y，问满足这n周需求量的最小花费

  $1 <= n <= 10000, 1 <= s <= 100, 1 <= ci <= 5000, 1 <= yi <= 10000$

- 题目分析

  贪心，第i周选取前i周中最小花费，开一个变量m记录$(cj + s * (i - j))$的最小值即可，其中$j = 1, 2, \ldots, i$，值得注意的是答案要用$long long$

  时间复杂度为$O(n)$

- 代码

```
#include<iostream>
#include<cstdio>
using namespace std;
int n, s, c, y, m;
long long ans;
int min(int x, int y) { return x < y ? x : y; }
int main()
{
    cin >> n >> s;
    ans = 0, m = 0x3f3f3f3f;
    while (n--)
    {
        cin >> c >> y;
        m = min(m + s, c);//更新每周当前的最小值
        ans += 1ll * m * y;
    }
    cout << ans << endl;
    return 0;
}
```

POJ 1328

- 题意

  平面直角坐标系上有n个点，求在x轴上找尽量少的点，以这些点为圆心画一个半径为d的圆，使得给定的点都在画出来的圆里。如果不能输出-1。

- 题目分析

  如果想要在x轴上选点画圆覆盖(a,b),那么这个点的横坐标就一定在$[a-\sqrt{d^2+b^2},a+\sqrt{d^2-b^2}]$区间内。

  因此，这道题就变成了在数轴上选最少的点，覆盖给定线段。这就类似于整数区间了。

  因此按右端点牌序，切割即可

- 代码

```
#include<cstdio>
#include<cmath>
#include<algorithm>
#include<iostream>
using namespace std;
struct P
{
    double l, r;
    bool operator < (const P& x) const//右端点排序
    {
        if (r != x.r) return r < x.r;
        return l < x.l;
    }
}p[2005];
int n, x, y, d, sp;
double D;
int main()
{
    while (1)
    {
        cin >> n >> d;
        if (!n && !d) break;
        bool f = 1;
        for (int i = 1; i <= n; i++)
        {
            cin >> x >> y;
            D = 1.0 * d * d - 1.0 * y * y;
            if (d >= y)
                p[i].l = x - sqrt(D),
                p[i].r = x + sqrt(D);
            else f = 0;
        }
        if (!f)
        {
            printf("Case %d: -1\n", ++sp);
            continue;
        }
        sort(p + 1, p + 1 + n);
        double now = p[1].r;
        int ans = 1;
        for (int i = 2; i <= n; i++)
            if (p[i].l > now)
                ans++, now = p[i].r;
        printf("Case %d: %d\n", ++sp, ans);
    }
}
```

POJ 3262（自选贪心）

- 题意

  有n个牛在FJ的花园乱吃。所以FJ要赶他们回牛棚。 每个牛在被赶走之前每秒吃Di个花朵。赶它回去FJ来回要花的总时间是Ti×2。在被赶走的过程中，被赶走的牛就不能乱吃

- 题目分析

贪心。按D/T将牛进行排序，然后计算即可

- 代码

```
#include<iostream>
#include<stdio.h>
#include<algorithm>
#include<string.h>
using namespace std;
const int MAXN=100005;
typedef long long ll;
const ll INF=100000000000;
int used[MAXN];
struct Cow
{
    int t,d;
    double div;
}cows[MAXN];
int n;
bool Cmp(const Cow a,const Cow b)
{
    return a.div>b.div;
}
ll Solve()
{
    ll res=0,sum=0;
    for(int i=0;i<n;i++) sum+=cows[i].d;
    for(int i=0;i<n;i++)
    {
        sum-=cows[i].d;
        res+=sum*2*cows[i].t;
    }
    return res;
}
int main()
{
    while(scanf("%d",&n)==1)
    {
        for(int i=0;i<n;i++){
            scanf("%d%d",&cows[i].t,&cows[i].d);
            cows[i].div=double(cows[i].d)/double(cows[i].t);
        }
        sort(cows,cows+n,Cmp);
        cout<<Solve()<<endl;
    }
    return 0;
}
```

POJ 3233（自选分治）

- 题意

给出$n\times n$矩阵A和正整数，求出$s=A+A^2+A^3+...+A^k$,取值范围，$n\leq 30,k\leq 10^9,m<10^4$

- 题目分析

注意到k值很大，直接做矩阵乘法会超时，类似于二分快速幂算法

$$k\text{为偶数：}sum(k)=(1+A^{(k/2)})*(A+A^2+.....+A^{(k/2)})=(1+A^{(k/2)})*sum(k/2)$$

$$k\text{为奇数：}sum(k)=(1+A^{((k-1)/2)})*sum(k/2)+A^k$$

- 代码

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
int n,m;
struct matrix
```

```
{
    int m[55][55];
}a;
matrix multiply(matrix x, matrix y)
{
    matrix ans;
    memset(ans.m, 0, sizeof(ans.m));
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            if(x.m[i][j])
                for(int k = 1; k <= n; k++)
                    ans.m[i][k] = (ans.m[i][k] + x.m[i][j] * y.m[j][k]) % m;
    return ans;
}
matrix add(matrix x, matrix y)
{
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            x.m[i][j] = (x.m[i][j] + y.m[i][j]) % m;
    return x;
}
matrix quickmod(matrix a, int p)
{
    matrix ans;
    memset(ans.m, 0, sizeof(ans.m));
    for(int i = 1; i <= n; i++)
        ans.m[i][i] = 1;
    while(p)
    {
        if(p & 1)
            ans = multiply(ans, a);
        p >>= 1;
        a = multiply(a, a);
    }
    return ans;
}
matrix solve(matrix a, int k)
{
    if(k == 1)
        return a;
    matrix ans;
    memset(ans.m, 0, sizeof(ans.m));
    for(int i = 1; i <= n; i++)
        ans.m[i][i] = 1;
    ans = add(ans, quickmod(a, k >> 1));
    ans = multiply(ans, solve(a, k >> 1));
    if(k & 1) //奇数
        ans = add(ans, quickmod(a, k));
    return ans;
}
int main()
{
    int k;
    scanf("%d %d %d", &n, &k, &m);
    matrix ans, a;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            scanf("%d", &a.m[i][j]);
    ans = solve(a, k);
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j < n; j++)
            printf("%d ", ans.m[i][j]);
        printf("%d\n",ans.m[i][n]);
    }
    return 0;
}
```

第三次POJ报告

09018316 黄开鸿 第三次报告

POJ 1050

- 题意概述

  > 给定一个n阶矩阵（方阵），矩阵元素为int，求出其中子矩阵元素的总和的最大值

- 题意分析

  > 不妨从简化问题开始分析。假设本题是求一位数组中所有元素的连续最大和，那结果易得
  >
  > 直接使用简单的遍历，将非负数加入结果，遇到负数则重新计算，取遍历中出现过的最大值即可。
  >
  > 但回到本题，要求求出子矩阵元素最大和，也就意味着选择的粒度从每个元素降到了一行一列。于是可以将一维数组中的做法扩展到二维，DP数组扩展到3维，以行为主迭代，对每一列，以其为子矩阵的起始列，分别算出不同结束列的最大值，递推公式为：
  >
  > $$DP[i][j][k]=max(DP[i-1][j][k]+sum,sum)$$
  >
  > i是指行，j是左起始列，k是右结束列，sum为当前新增的第i行从j列到k列的元素和
  >
  > （因为是矩阵，要选这一行就得全选）

- 实际代码

```
#include <cstdio>
#include <string.h>
#include <string>
#include <iostream>
using namespace std;

#define maxn 100
#define inf 0x3f3f3f3f

int array[maxn][maxn];
int DP[maxn][maxn][maxn];

int main()
{
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> array[i][j];
        }
    }
    memset(DP, 0, sizeof(DP));
    int res = -inf; //负无穷初始化，保证可能值都比其=大
    //i是指行，j是左起始列，k是右结束列，当前值为在ijk范围内的元素和最大值
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            int sum = 0;
            for (int k = j; k < n; k++)
            {
                sum += array[i][k];
                //比较是接着上一行进行扩展还是舍弃
                if (i > 0)
                {
                    DP[i][j][k] = max(DP[i - 1][j][k] + sum, sum);
                }
                else
                {
                    DP[i][j][k] = sum;
                }
                //取出现过的最大值
```

```
                res = max(res, DP[i][j][k]);
            }
        }
    }
    cout << res << endl;
    return 0;
}
```

POJ 1458

因为是上课过程中原题，没有过多赘述：

```
#include <iostream>
#include <cstring>

using namespace std;

char string1[5005];
char string2[5005];
int maxLen[5005][5005];
int main()
{
    while (cin >> string1 >> string2)
    {
        int length1 = strlen(string1);
        int length2 = strlen(string2);
        int nTmp;
        int i, j;
        for (i = 0; i <= length1; i++)
            maxLen[i][0] = 0;
        for (j = 0; j <= length2; j++)
            maxLen[0][j] = 0;
        for (i = 1; i <= length1; i++)
        {
            for (j = 1; j <= length2; j++)
            {
                if (string1[i - 1] == string2[j - 1])
                    maxLen[i][j] = maxLen[i - 1][j - 1] + 1;
                else
                    maxLen[i][j] = max(maxLen[i][j - 1], maxLen[i - 1][j]);
            }
        }
        cout <<相关论文 maxLen[length1][length2] << endl;
    }
    return 0;
}
```

自出题

> Pacman的DQN相关论文以及复刻裁判程序阅读pytorch文档实在花费了大量时间，自己试着写了一道DP 也没办法验证到底DP做不做得出来，实在力不从心。只能依照之前做题的心得写了一写题，但是实在没有去编数据验证的时间了，也可能是一个不可解的问题，希望老师见谅。

- 题目概述

  给定一个正整数数列，在其中找出一段连续的序列使其平均值最大。

- 题意分析

  乍看起来和求最大子序列和右异曲同工之妙，但是最大的区别是对于迭代中的某一个状态，需要储存每一段的区间长度。于是可以这么素：一个固定左端起点长度为a的区间向右遍历，直至平均值对大。循环n次来更换不同起点。

- 实际代码

```
#include<iostream>
#include<cstdio>
#include<string.h>
using namespace std;
const int MAXN=100000+5;
```

```
int N,F;
double num[MAXN];
double sum[MAXN];
double rmaxsum[MAXN];
double l=9999999,r;
inline bool qualify(double tans){
    for(int i=N;i>=1;i--)
        rmaxsum[i]=max(num[i]-tans,rmaxsum[i+1]+num[i]-tans);
    for(int i=1;i<=N-F+1;i++){
    //  cout<<sum[i+F-1]-sum[i-1]<<" "<<F*tans<<" "<<tans<<endl;
        if(sum[i+F-1]-sum[i-1]>=F*tans)
            return true;
    //  cout<<"sec:"<<sum[i+F-1]-sum[i-1]-F*tans+rmaxsum[i+F]<<endl;
        if(sum[i+F-1]-sum[i-1]-F*tans+rmaxsum[i+F]>=0)
            return true;
    }
    return false;
}
int main(){
    cin>>N>>F;
    for(int i=1;i<=N;i++){
        scanf("%lf",&num[i]);
        sum[i]=sum[i-1]+num[i];
        r=max(r,num[i]);
        l=min(l,num[i]);
    }
    while(l<r-0.0001){
        double mid=(r+l)/2;
        if(qualify(mid))
            l=mid;
        else
            r=mid;
    }
    cout<<int(r*1000)<<endl;
    return 0;
}
```

09018330孙毅远 第三次报告 POJ

POJ 1458

- 题意

  求两个字符串的最长公共子串的长度

- 题目分析

  经典动态规划，状态转移方程为
  $if(a[i-1]==b[j-1])dp[i][j]=dp[i-1][j-1]+1$
  $else\ dp[i][j]=max(dp[i-1][j],dp[i][j-1])$

  时间复杂度为$O(n)$

- 代码

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
const int inf = 0x3f3f3f;
int dp[505][505];
char a[505],b[505];
int main()
{
    while(~scanf(" %s %s",a,b))
    {
```

```
        int a1 = strlen(a);
        int b1 = strlen(b);
        for(int i = 1; i <= a1; i++)
        {
            for(int j = 1; j <= b1; j++)
            {
                if(a[i-1]==b[j-1])
                dp[i][j] = dp[i-1][j-1]+1;
                else
                dp[i][j] = max(dp[i-1][j],dp[i][j-1]);
            }
        }
        cout<<dp[a1][b1]<<endl;
    }
    return 0;
}
```

POJ1050

- 题意

  给定矩阵求最大子矩阵和，$N \leq 100$

- 题目分析

  考虑前缀和做法，$sum[i][j]$表示 $\sum_{k=1}^{j} a[i][k]$

  预处理之后在做dp时枚举起点i与终点j最内层，枚举行号，那么可以一行一行的累加，最后更新答案即可

- 代码

```
#include<cstdio>
#include<algorithm>
using namespace std;
int sum[105][105],n,a[105][105],ans=-(1<<29);
int main(){
    scanf("%d",&n);
    for (int i=1;i<=n;i++){
        for (int j=1;j<=n;j++){
            scanf("%d",&a[i][j]);
            sum[i][j]=sum[i][j-1]+a[i][j];
        }
    }
    for (int i=1;i<=n;i++){
        for (int j=i;j<=n;j++){
            int num=0;
            for (int k=1;k<=n;k++){
                if(num<0) num=0;
                num+=sum[k][j]-sum[k][i-1];
                ans=max(ans,num);
            }
        }
    }
    printf("%d\n",ans);
    return 0;
}
```

砍树（自出贪心）

- 题意

无限长的数轴上有n个点，每个点的坐标为$x[i]$，种有高度为$h[i]$的树，现在要把一些树砍到，被砍倒的树要么倒向左边，要么倒向右边，会分别把$[x_i - h_i, x_i]$和$[x_i, x_i + h_i]$占用，如果某棵树不被砍倒，那么它就只占用$x[i]$这一个点的位置，现在给定你n个点的$x[i], h[i]$，问最多能砍倒几棵树？（$n <= 1e5, x[i], h[i] <= 1e9$）

- 题目分析

  在于对于某个区间[xi,x(i+1)]来说，这段区间要么被xi利用，要么被x(i+1)利用，要么被它们两个都利用，要么都不用，只有这四种可能。所以n个点把数轴划成了n+1个区间，我们从左往右依次枚举这n+1个区间即可，尽可能地让这些区间占用更多的树即可

- 代码

```
#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
const int inf=2e9;
const int maxn=100050;
int n;
bool used[maxn];
struct node{
    int x,h;
    bool operator<(const node& e)const{
        return x<e.x;
    }
}tree[maxn];

int main(){
    scanf("%d",&n);
    for(int i=0;i<n;++i){
        scanf("%d%d",&tree[i].x,&tree[i].h);
    }
    sort(tree,tree+n);
    int ans=0;
    for(int i=0;i<=n;++i){//枚举n+1段路
        if(0==i){//最左边的路放tree[0]
            used[0]=1;
            ++ans;
            continue;
        }
        if(n==i && !used[n-1]){//如果最右边的树没着落，那就放到最右边的路上
            used[n-1]=1;
            ++ans;
            continue;
        }
        int le= tree[i-1].x;
        int ri= tree[i].x;
        if(used[i-1]){//左边的树已经搞定了，只考虑当前的树就可以，能放就放
            int len=ri-le;
            if(len>=tree[i].h+1) { used[i]=1; ++ans; }
        }
        else{//左边的树还没着落，能都放下就都放下，放不下优先考虑放左边的
            int len=ri-le+1;
            if(len>=tree[i-1].h+1 + tree[i].h+1){
                used[i-1]=used[i]=1;
                ans+=2;
          }
            else if(ri-le>=tree[i-1].h+1){
                used[i-1]=1;
                ++ans;
            }
            else if(ri-le>=tree[i].h+1){
                used[i]=1;
                ++ans;
            }
        }
    }
    printf("%d\n",ans);
    return 0;
}
```

得分（自出DP）

- 题意

  给定一个n个数的序列，从中选取任意一个元素x，得到x分，同时消除序列中所有等于$x-1$和等于$x+1$的元素。问能得到的最高分
  $1 \leq n \leq 10^5, a_i \leq 10^5$

- 题目分析

  数组$c[i]$存储序列中i出现的次数。从左向右开始，若选取了$i-1$，那么i被消除，$dp[i]=dp[i-1]$
  若选取了i，那么$i-1$被消除，$dp[i]=dp[i-2]+c[i]*i$。取最大的那一种。

- 代码

```
#include<iostream>
#include<cstdio>
#include<stack>
#include<cmath>
#include<cstring>
#include<queue>
#include<set>
#include<algorithm>
#include<iterator>
#define INF 0x3f3f3f3f
typedef long long ll;
typedef unsigned long long ull;
using namespace std;

const int maxn=100005;

ll dp[maxn],c[maxn],temp,MIN=INF,MAX=0,t=0;
int n;

int main()
{
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        scanf("%lld",&temp);
        c[temp]++;
        MIN=min(MIN,temp);
        MAX=max(MAX,temp);
    }
    for(int i=MIN;i<=MAX;i++)
    {
        if(i==1)
            dp[i]=max(dp[i-1],c[i]);
        else
            dp[i]=max(dp[i-1],dp[i-2]+c[i]*i);
    }
    for(int i=MIN;i<=MAX;i++) t=max(t,dp[i]);
    printf("%lld\n",t);
    return 0;
}
```