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

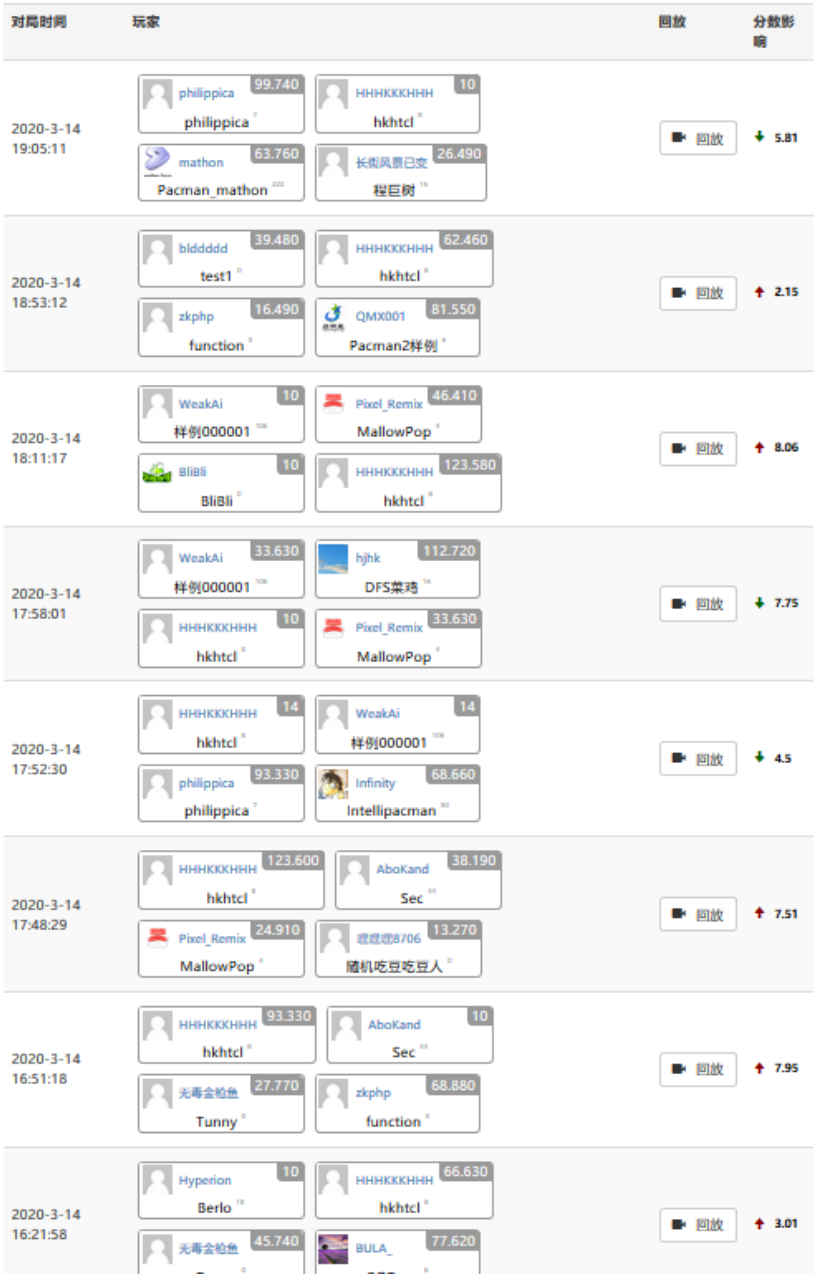
botzone 网站上 Pacman2 游戏的开发 BOT。旨在开发一个基于非机器学习思路的高适应性，高内聚低耦合的 Pacman2 游戏 Bot，将众多基础算法与游戏规则进行有机结合，并在开发中保证 Bot 代码的可扩展性和可读性，保证快速通过对局分析，迭代 Bot 的决策逻辑。

以下表格为当前在 botzone 网站上的天梯数据。

截止 2020-3-14 天梯数据

Bot 名	天梯分数	天梯排名
hkhtcl	1010.6	60
normal1（程序版本未更新）	998.2	79

截止 2020-3-14 被动天梯对战



Version 0.1.0

更新时间：2020-3-12

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();
    }
}
```

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>
```

```cpp
#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

```cpp
#include <iostream>
#include <cstdio>
```

```cpp
#include <iomanip>
#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;
}
```