清华大学人工智能挑战赛 A1 组赛题

一、游戏规则

基本设定

每张地图的一条对角线两端有双方玩家的主基地，地图上会随机生成若干条道路。玩家可以在非道路单元格上建造建筑，建筑类型有：生产型、防御型和资源型，三者分别自动进行单位生产、攻击敌方单位和资源获取操作。同时玩家可以花费一定资源升级时代，每升级一次时代都会解锁更强大的功能。通过合理地建造建筑，击毁敌方主基地或者在达到回合上限时拥有更多战斗力即可取得胜利。

地图设定

（1）基本设定：200*200 正方形地图，实体与数据两层；1 代表道路，0 代表空地，2 代表基地。玩家只能在空地上进行建造，且同一片空地上只能有一个建筑。唯一可能出现重叠的情况为两玩家于同一回合在同一位置建造建筑。

（2）主基地：主基地位于主对角线两个端点处，大小为 7*7，主基地内不能存在任何单位与建筑，单位对主基地的距离按照顶点计算。

（3）道路：随机产生若干条宽度为 1 的道路连接双方主基地，所有道路成中心对称，道路之间不存在交叉。地图正中央有设有 2*2 的一块方形路面以使前后道路连通。

回合逻辑顺序

（1）防御塔与单位攻击。

（2）死亡与被击毁判定。

（3）单位移动。

（4）建筑建造、修复、升级\出售。

（5）单位生产与资源产出。

（6）科技升级。

（7）胜负判定。

时代

一共分为 6 个时代：比特、电路、处理器、算法、网络、AI。

升级时代基础消耗：2000(每级+1500）

升级时代可以解锁以下功能：

（1）解锁新的建筑

（2）可建造已解锁的建筑的高级版

（3）可以对已建造在地图上的建筑进行建筑升级操作

（4）获得更高的建造力

建筑

（1）建筑占据 1 个单元格，不可位于道路或主基地内，玩家只能在“建造范围”内建造新建筑。

（2）建造范围：玩家每个已有建筑周围 8 格内以及主基地边缘 4 格宽度内（即玩家 0 横纵坐标<11，玩家 1 横纵坐标>188，且不在基地范围内）为建造范围，新建筑只能建于建造范围内。

（3）建筑的属性：建筑 ID、生命值、位置、类型、CD、维修状态、攻击类型、攻击力、资源消耗、射程、AOE（单体伤害为 0）、建造力需求。

（4）生产型建筑：可在生产范围内的任意一个“道路”单元格上生产兵种。

（5）防御型建筑：对攻击范围内的敌方兵种进行自动攻击。

（6）资源型建筑：每回合获取资源值。

（7）初始建筑上限为 40，每提升 1 级科技加 20。

单位

（1）单位分为实体单位、数据单位两大类。

（2）单位会自动行动，玩家不可操作，行动模式有：推塔、冲锋、抗线。

（3）每种单位均由对应的生产建筑自动生产，不需要消耗资源。

（4）单位属性：ID、位置、生命值、行动模式、速度、攻击力、移动类型、射程。

（5）推塔模式：攻击范围内存在敌方建筑时停止移动并进行攻击直至摧毁目标。

（6）冲锋模式：单位冲向敌方主基地，在到达其边缘时死亡并对主基地造成大量伤害（不会攻击沿途建筑）。

（7）抗线模式：一边移动，同时攻击路过的建筑。

（8）单位没有碰撞体积，一个单元格内理论上可以存在无数个单位。

胜利条件

（1）限定回合内，一方摧毁另一方主基地即获胜。

（2）规定回合数内，不满足（1）或者双方主基地同时被摧毁，则按以下顺序判定胜负：

- a.科技时代高的一方获胜
- b.主基地剩余血量多的一方获胜
- c.剩余建筑数量多的一方获胜
- d.剩余资源总量多的一方获胜
- e.以上均无法判定则平局

（3）一方 AI 崩溃，游戏将继续进行至结束

主基地

主基地初始血量为 10000，每升一级科技，最大血量提升 5000，当前血量保持百分比不变。

关于建筑升级、维修与出售

进行科技升级后，已造的建筑并不会升级，需要单独进行建筑升级操作。

建筑升级采用以下规则：

（1）补齐低级建筑与高级建筑之间的差价。

（2）将建筑恢复至满血并按照建筑已损失的血量百分比，等比例消耗建造建筑所需的资源。

（3）建筑升级并不会改变建筑种类。例如，在科技 2 级时，升级一个 0 级 bool，它会变成一个 1 级 bool，还可以继续升级成 2 级 bool，但不能超过 3 级。但是如果在科技 2 级时建造 bool 将会直接建造 2 级 bool，花费更多的资源，拥有更强的属性。

建筑维修采用建筑升级第（2）条。

建筑出售按剩余生命值比例返还最高 50%的建造消耗。

关于建造力

（1）建造力是一个只与科技水平相关的值，初始为 80，每提升一级科技增加 60。

（2）建造建筑需要花费建造力，建造力不足时无法进行建造。

（3）建造力每回合自动补充至满值，不会溢出，也不可囤积。

所有建筑与单位的详细设定

```
1.科技因数 = 1+建筑等级*0.5
2.所有给出基础值的实际数值为基础值*科技因数，即 bit 级别的数值为基础值，此后每级增加 50%;
  例如，BIT 科技时建造的 0 级布尔属性刚好为下表所示基础属性，而 AI 科技建造的建筑，一经建造就是 5 级，属性为下表属性*3
3.关于 AOE：AOE 为 2 表示可以攻击某一格和其相邻格内所有单位，即与该格距离小于 2，而非小于等于 2，以此类推
4.数据类单位只会被数据攻击打到，实体单位同理
```

资源建筑	基础生命值	基础资源采集值	基础资源	基础建造力
码农	100	10	100	10

生产建筑	基础生命值	单位	生产范围	生产CD	解锁时代	基础资源	基础建造力
香农	150	比特流	10	2	比特	150	15
戴维南	200	电压源	5	6	电路	160	16
诺顿	180	电流源	5	6	电路	160	16

生产建筑	基础生命值	单位	生产范围	生产CD	解锁时代	基础资源	基础建造力
冯诺依曼	200	ENIAC	15	15	处理器	200	20
伯纳斯李	150	数据包	20	3	网络	250	25
高锟	160	光纤	25	8	网络	300	30
图灵	250	图灵机	15	20	AI	600	60
托尼斯塔克	220	奥创	10	20	AI	600	60

单位	类型	行动模式	基础生命值	基础攻击力	攻击范围	移动速度
比特流	数据	推塔	16	10	16	12
电压源	实体	推塔	60	18	24	8
电流源	实体	冲锋	80	160	6	15
ENIAC	实体	抗线	550	12	10	4
数据包	数据	冲锋	50	300	6	16
光纤	实体	推塔	40	15	40	12
图灵机	数据	抗线	850	8	12	3
奥创	实体	推塔	100	500	20	8

防御建筑	基础生命值	基础攻击力	攻击CD	射程	攻击目标	AOE	解锁时代	基础资源	基础建造力	特效
布尔	250	16	1	36	数据	0	比特	150	15	布尔代数：有一半的几率攻击无效
欧姆	320	20	3	30	实体	6	电路	200	20	欧姆定律：同时击中电压源和电流源时对所有击中单位造成 4 倍伤害
摩尔	250	4	1	60	实体	0	处理器	225	22	摩尔定律：优先攻击上回合目标且对一个目标连续攻击时每次伤害*2
蒙	350	25	2	40	实	0	算	200	20	蒙特卡罗方法：对

防御建筑	基础生命值	基础攻击力	攻击CD	射程	攻击目标	AOE	解锁时代	基础资源	基础建造力	特效
特卡罗					体		法			单位造成的伤害会乘以一个 0~2 之间的随机数
拉里罗伯茨	220	6	1	50	全部	8	网络	250	25	阿帕网：优先攻击数据包且对数据包造成的伤害*3
罗伯特卡恩	520	5	1	36	数据	0	网络	450	45	TCP\IP 协议：攻击任何目标均附加其最大生命值百分之 10（*科技因数）的伤害
马斯克	1000	0	1	24	全部	0	AI	500	50	AI 末日论：使其攻击范围内的 AI 兵种无法移动与攻击
霍金	480	∞	2	20	全部	2	AI	500	50	黑洞：你也看到这伤害了，消灭 1 格和邻格内所有单位

二、选手接口与程序使用教程

1. 选手操作接口

void updateAge()　　升级时代

void construct(BuildingType building_type, Position pos, Position soldier_pos) 建造 命令 参数一次是建造的建筑类型，建造的位置和出兵位置(非造兵建筑缺省参数)

void upgrade(int unit_id) 升级单位命令，参数为目标单位 id void

sell(int unit_id)　　出售命令，参数为单位 id

void toggleMaintain(int unit_id)　　修理命令，参数为目标单位id

2. 获取游戏静态信息

选手通过 bool 变量 ts19_flag 获取自己是 0/1 号玩家
选手通过变量 int** ts19_map 来获取地图信息，其中 map 是一个 200*200 的 int 数组。其中 0 代表无路，1 代表有路，2 代表主建筑。

```
const int UPDATE_COST         = 2000;          基础升级科技资源消耗
const int UPDATE_COST_PLUS    = 1500;          每升一级升级科技消耗增量
const int MAX_BD_NUM          = 40;            基础建筑上限
const int MAX_BD_NUM_PLUS     = 20;            每升一级建筑上限增量
const int MAP_SIZE            = 200;           地图大小
const int BASE_SIZE           = 7;             基地范围
const int BD_RANGE            = 8;             建筑周围建造范围
const int BD_RANGE_FROM_BASE  = 4;             基地周围建造范围
const float AGE_INCREASE      = 0.5;           其他所有基础属性每升一级提高比例
```

3. 获取全局动态信息

```
struct State
{
    int turn;
    int winner;
    _resource resource[2];
    Age age[2];
    vector<Building> building[2];
    vector<Solider> soldier[2];
};
```

游戏信息储存在这样一个结构体中，玩家可以通过指针 state 获取这些信息。

turn 是当前游戏回合数目，winner 为当前胜利者，其他为选手的资源和建筑，兵 种。
其中 resource age building soldier 这几个数组分别储存了两个选手的信息。
[0]是 player0 的信息，[1]是 player1 的信息。
building[i] solider[i]储存的是装有结构体 Building，Soldier 实例的向量（STL 的 vector）。

4. 各类信息结构体

```
struct _resource
{
    int building_point;
    int resource;
};


struct Position { int x;
    int y;
};
struct Soldier {
    SoldierName soldier_name;      //soldier 的类型
    int heal;       //soldier 的血量
    Position pos;   //soldier 的位置
    int flag; //soldier 的阵营
```

```
    int unit_id;  //soldier 的 ID
};
struct Building {
    BuildingType building_type;   //building 的类型
    int heal;   //building 的血量
    Position pos;   //building 的位置
    int flag; //building 的阵营
    int unit_id;  //building 的 id
    int level;  //building 的等级
};
```

5.使用教程与提示

1.选手在 player.cpp 文件的 f_player()函数中编写自己的代码，请不要在其他文件中写代码，最终只提交一个文件。
2.对战方式：
windows
首先打开 server.exe 程序
运行编译链接好的 C++文件生成的.exe 文件 运行一次代表连接一个 AI，需要且仅需要运行两次。
Macos
同 windows
注意 g++编译命令 g++ main.cpp api_player.cpp player.cpp communication.cpp - pthread -o filename

3.程序崩溃惩罚： 选手应该首先保证自己的程序没有非法调用，一旦程序崩溃，无论在游戏中局势怎样，崩溃方直接判定为负，如果两人在同一局崩溃，就判定为平局。

4.超时惩罚 首先，无论选手超时情况如何，程序能够保证选手拿到的游戏局势一定是最新的。其次，如果选手程序超时，服务器不会等待你的命令，因此，超时会导致你落后别人回合。而你在之前第 m 回合的命令，超时了 x 轮，在服务器的 m+x 轮把你 第 m 轮的命令发送到了，那么服务器会接受其中的合法命令。因此，超时会带来落后回合的惩罚，请尽量保证不要超时。
提醒：

1. 单回合指令上限是 50 条，尽量不要在一回合发出过多指令，如果超出，我们会随机截取 50 条，再检查合法性。如果一回合发出过多指令，实测中若发送 4w 条指令，通信速度会下降，每次会落后4 回合左右。
2. 在使用状态的时候，请注意state[flag].building.size()是一个随最新游戏状态变化的值，因此如果你的程序跨越回合了，请注意你的代码是否会导致越界。

三、示例 AI

以下代码进行了简单的信息获取与游戏操作，均采用最简单的暴力写法，仅供参考

<player.cpp>

```cpp
#include"communication.h"
#include<vector>
#include<iostream>

using std::cout;
using std::endl;

extern State* state;
extern std::vector<State* > all_state;
extern int** ts19_map;
extern bool ts19_flag;


void player0();
void player1();



int dist(Position p1, Position p2) {
    return abs(p1.x - p2.x) + abs(p1.y - p2.y);
}
bool near_a_road(Position p) {
    int x = p.x;
    int y = p.y;
    if (ts19_map[x][y] == 0) {
        if (ts19_map[x - 1][y] == 1)
            return 1;
        if (ts19_map[x + 1][y] == 1)
            return 1;
        if (ts19_map[x][y - 1] == 1)
            return 1;
        if (ts19_map[x][y + 1] == 1)
            return 1;
    }
    return 0;
}
Position find_road(Position p) {
    int x = p.x;
    int y = p.y;
    if (ts19_map[x][y] == 0) {
        if (ts19_map[x - 1][y] == 1)
            return Position(x - 1, y);
        if (ts19_map[x + 1][y] == 1)
            return Position(x + 1, y);
        if (ts19_map[x][y - 1] == 1)
            return Position(x, y - 1);
        if (ts19_map[x][y + 1] == 1)
            return Position(x, y + 1);
    }
    return Position(-1, -1);
}
bool near_my_base(Position p) {
    if (ts19_flag == 0) {
        if (p.x < 11 && p.y < 11)
            return 1;
        else
            return 0;
    }
    else {
```

```
        if (p.x > 188 && p.y > 18)
            return 1;
        else
            return 0;
    }
}
bool near_my_building(Position p) {
    bool yes = 0;
    for (int i = 0; i < state->building[ts19_flag].size(); i++) {
        if (dist(p, state->building[ts19_flag][i].pos) <= 8)
            yes = 1;
        if (p.x == state->building[ts19_flag][i].pos.x && p.y == state->building[ts19_flag][i].pos.y)
            yes = 0;
    }
    return yes;
}
bool can_cons(Position p) {
    if (ts19_map[p.x][p.y] == 0 && (near_my_base(p) || near_my_building(p)))
        return 1;
    else
        return 0;
}
void print_my_bd() {
    for (int i = 0; i < state->building[ts19_flag].size(); i++) {
        Building bd = state->building[ts19_flag][i];
        cout << "id:" << bd.unit_id << " type:" << bd.building_type << " HP:" << bd.heal << " pos:" << bd.pos.x
<< ' ' << bd.pos.y << endl;
    }
}
void print_my_re() {
    cout << "resourse:" << state->resource[ts19_flag].resource << endl;
}
void print_my_base() {
    for (int i = 0; i < state->building[ts19_flag].size(); i++) {
        if (state->building[ts19_flag][i].building_type == 0) {
            cout << "baseHP:" << state->building[ts19_flag][i].heal << endl;
        }
    }
}
void print_info() {
    cout << "turn:" << state->turn << endl;
    print_my_re();
    print_my_base();
    print_my_bd();
}
void f_player()
{
    /*
    if (state->turn == 1) {
    for (int i = 0; i < 200; i++) {
    for (int j = 0; j < 200; j++) {
    cout << ts19_map[i][j];
    }
    cout << endl;
    }
    }
    */
    if (ts19_flag == 0) {
        Sleep(1000);
        player0();
    }
```

```
    else
        player1();

    print_info();


};
void player0() {
    int ins_num = 0;
    if (state->turn < 100) {
        for (int i = 6; i < 20; i++) {
            for (int j = 6; j < 20; j++) {
                if (can_cons(Position(i, j))) {
                    if (near_a_road(Position(i, j)) && state->resource[ts19_flag].resource >= 1000)
                        construct(Shannon, Position(i, j), find_road(Position(i, j)));
                    else
                        construct(Programmer, Position(i, j), Position(0, 0));
                    ins_num++;
                    if (ins_num > 50)
                        return;
                }
            }
        }
    }
}
void player1() {
    int ins_num = 0;
    if (state->turn < 100) {
        for (int i = 193; i > 180; i--) {
            for (int j = 193; j > 180; j--) {
                if (can_cons(Position(i, j))) {
                    if (near_a_road(Position(i, j)) && state->resource[ts19_flag].resource >= 1000)
                        construct(Shannon, Position(i, j), find_road(Position(i, j)));
                    else
                        construct(Programmer, Position(i, j), Position(0, 0));
                    ins_num++;
                    if (ins_num > 50)
                        return;
                }
            }
        }
    }
}
```