

选手接口文档

1.选手接口

`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(非造兵建筑空缺这个参数)

`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.游戏静态信息

选手通过 `bool` 变量 `flag` 获取自己是 0/1 号玩家

选手通过变量 `int** 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。

3.获取游戏的全局动态信息

`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 and 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，其中 `[0]` 是 `player0` 的信息，`[1]` 是 `player1` 的信息

其中 `building[i] solider[i]` 储存的是装有结构体 `Building`, `Soldier` 实例的向量（STL 的 `vector`）。

4.对应资源信息结构体的实例

`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
    };

```

注：1.选手是指在 **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。

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. 我们单回合指令上限是 50 条，请不要在一回合发出过多指令，如果超出一些，但并不太多，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，如我们实验中测试发送 4w 条指令，通信速度会下降，每次会落后 4 回合左右。
2. 在使用我们的状态的时候，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。