2021年地空数算"星际群落`Stellar`"大作业开发文档

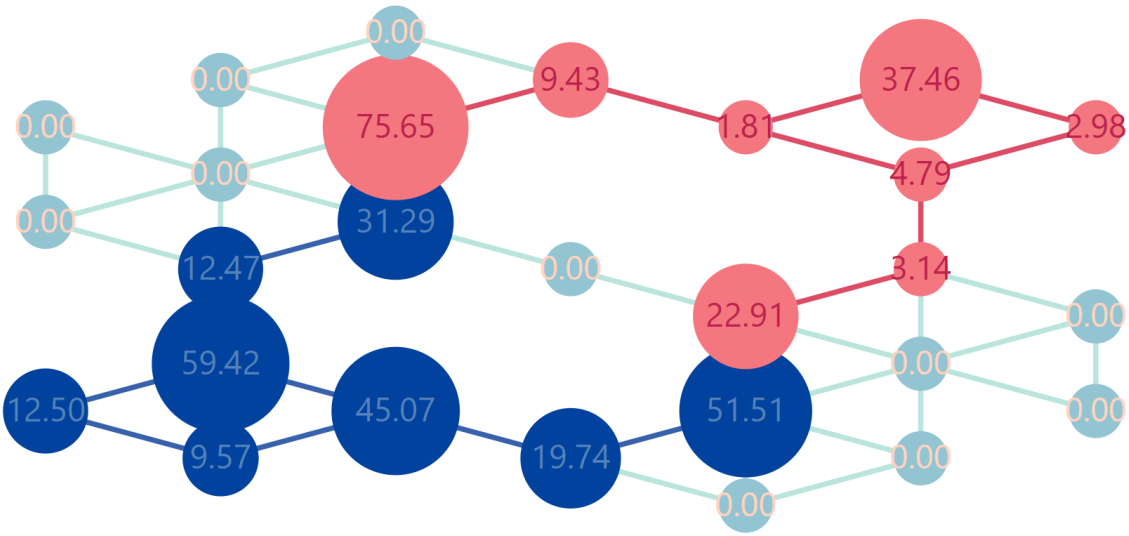
一 游戏背景及参赛要求

一年一度的地空数算创意大作业再次到来~这次的大作业主题是围绕星际群落的生命演化游戏展开的。在这个游戏中，双方玩家将操纵己方生命单位向不同节点移动，在生产、衰减、抢占资源、与敌方作战等多重策略博弈中享受生命演化过程的快乐。最终，率先占领对方大本营的玩家将获胜，赢得在这片星际中的绝对统治。本游戏中，玩家的操作内容十分简单，但涉及的策略却变化多端，充满挑战性。游戏创意与知名P社游戏《群星(`Stellaris`)》有关，因此本次大作业命名为"星际群落`Stellar`"。

本次大作业原则上每组不超过4人。自5月25日公布起，每个小组有3周的时间分析问题，设计算法，调试参数，撰写大作业实习报告。建议每个小组成员仔细阅读此开发文档，严格遵守文档上的代码编写等方面的要求，避免非算法因素对竞赛结果造成负面影响。

二 游戏规则与运行机制

本游戏的星际宇宙模型化为对称的无向图。整张地图由节点和节点之间相连的通道构成。节点包括两端的两个大本营节点和普通节点两种，每回合都有生产兵力的功能。



如图为某回合战斗过程的截图，此地图中左下角为蓝方大本营，右上角为红方大本营。

游戏以回合制进行。回合规定有上限，其获取方式为：

```
import config
print(config.MAX_TURN)
```

沿对称轴两端最远两端：一侧为初始玩家1的节点，另一侧为初始玩家2的节点。这两个节点分别为双方玩家的大本营。

游戏开始时，双方只控制各自的大本营节点，且大本营中有一定数量的相同兵力。其他节点都属于无主势力。

2.0 一般获胜条件

- 在回合上限内，率先占领对方大本营节点的玩家获胜。(注意，只需占领大本营，不需要占领全图)
- 特殊地，如果双方在同一个回合同时攻破对方大本营，则该回合结束之后，拥有兵力总数量更多的玩家获胜。
- 如果达到回合上限后无玩家攻破对方大本营，则该回合结束之后，拥有兵力总数量更多的玩家获胜。

游戏的每个回合，都要经历三个阶段：运输->战斗->生产。其中玩家仅允许在运输阶段发出运输的指令，其余流程完全由游戏的对战后台控制。三个阶段的详细说明如下：

2.1 运输

每回合第一阶段为运输阶段。玩家可以对每个已控制的节点发出命令，向不同节点送出不同数量的兵力。特别地，同一个节点可以同时向若干个邻接节点分摊运输兵力，且运输有兵力损耗。考虑到运输的损耗，假设从A节点运输数量为`send`的兵力，那么B节点就只能增加`gain`个兵力。具体计算`gain`的算法如下：

$$gain = \max(send - \sqrt{send}, 0)$$

平台获取到双方的运输指令之后，会同时执行完所有的运输指令，然后再进入第二阶段——战斗。

2.2 战斗

每回合第二阶段为战斗阶段，由对战平台根据第一阶段的运输结果自动执行。

若某节点上同时出现双方兵力，会触发战斗机制。战斗运算规则如下：假定双方在此节点的兵力分别为a和b，战斗后双方剩余兵力为a'和b'，其中：

$$\begin{aligned} if \quad a \geq b: \quad & a' = \sqrt{a^2 - b^2}; \quad b' = 0 \\ else: \quad & a' = 0; \quad b' = \sqrt{b^2 - a^2} \end{aligned}$$

即兵力较少者被全歼，兵力较多者取胜，其残余兵力占领此节点。

开局不属于双方玩家的无主节点中没有任何兵力，一旦有玩家的兵力进入就视为无阻碍占领该节点。特殊地，如果同一个无主节点上同时进入了双方玩家的兵力，那么先触发上述的战斗机制，然后胜者的残余兵力占据此无主节点。

2.3 生产

每回合第三阶段为生产阶段，由对战平台根据第二阶段的战斗结果自动执行。

若当前兵力小于某一阈值，则采用Logistic增长函数的差分表达进行兵力增长。若当前兵力高于某一阈值，高于阈值的部分会损失四分之一。兵力生产的具体计算方式为：

$$New = \begin{cases} x + (1 - x/powerLimit) * x * spawnRate, & \text{if } x \leq powerLimit \\ powerLimit + despawnRate * (x - powerLimit), & \text{if } x > powerLimit \end{cases}$$

其中x是当前兵力，New表示更新后的兵力；其中`powerLimit`和`spawnRate`、`despawnRate`为常量，其获取方式为：

```
import config
print(config.POWER_LIMIT) #100
print(config.SPAWN_RATE) #1
print(config.DESPAWN_RATE) #0.75
```

特别地：无主节点内永远不会生产兵力。

三 游戏数据结构及玩家须知属性

本游戏的实现过程中，使用了`GameMap`类来管理整个地图，其中包含了若干个节点Node类组成的列表nodes.特别注意，规定玩家1的大本营节点编号为1号，而玩家2的大本营节点编号为N号，其中`N=len(nodes)-1`为总节点个数。`nodes[i]`存储的是编号为i的节点，特别地，`nodes[0]`无定义，请不要引起混淆。

节点Node类是整个游戏的核心类。其存储的有效信息有：

- 节点的编号。`self.number=number`，为int型，规则和`GameMap`里规定的一致。
- 节点的归属势力。`self.belong=belong`，为int型，-1代表无主；0代表玩家1；1代表玩家2；
- 节点的兵力数量。`self.power=(power1,power2),`为元组tuple，内部存储两个float类型浮点数，`power1`代表玩家1的兵力，`power2`代表玩家2的兵力。示例：无主节点`self.power=(0,0)`；若玩家1对某节点有1.2的兵力，则`self.power=(1.2,0)`。
- 邻接节点的信息。`self.nextinfo=[node_id1,node_id2,...]`,为列表, 内部存储邻接节点的编号；可通过调用`self.get_next()`访问。

四 可调用接口及玩家提交文件要求

4.1 可调用接口概述

本游戏提供给每个小组的数据是一个`GameMap`类的实例。

玩家可以调用`GameMap`类中涉及的属性和方法：

- ”游戏数据结构“部分中涉及的所有Node的属性。再次强调，`GameMap.nodes`是存储所有节点的列表，每个编号的Node可以从`GameMap.nodes[i]`直接访问，注意之前的编号的约定。
- 如果只想要获得某节点相连的其他节点的编号，必须直接调用`node.get_next()`，返回的是存储int型邻接节点编号的列表。访问`node.nextinfo`是非法行为。

4.2 关于玩家提交的文件

玩家需要编写并提交一个python文件，其中应包含`player_class`类。`player_class`类中至少要实现成员方法`player_func(self, map_info)`与初始化方法`__init__(self, player_id)`。

其中，`map_info`为提供的地图`GameMap`类实例，内部存储了当前回合开始前地图的状态；玩家编号`player_id`，编号为int型。编号为0指玩家1，初始拥有1号节点大本营；编号为1指玩家2，拥有N号节点大本营。

`player_func`方法需要返回的，是一个包含若干元组的列表list。每一个元组代表一条运输指令。具体描述如下：

元组构成为`(from_node,to_node,num)`。分别代表兵力出发的节点编号(int型)，兵力输送目的地节点编号(int)，输送兵力数量(float型)。

若干这样的元组组成的列表作为返回值，代表该玩家在运输阶段的决策。

示例如下：

```
from random import randint as rd

def player_func(map_info,player_id): #辅助函数
```

```
    ACTIONS = []
    tmp_left = [i.power[player_id] for i in map_info.nodes]

    def isValid(action):
        a, b, c = action
        if map_info.nodes[a].belong != player_id:
            return False
        if b not in map_info.nodes[a].get_next():
            return False
        if tmp_left[a] <= c + 0.01:
            return False
        tmp_left[a] -= c
        return True

    for i in range(1000):
        tmp_action = (rd(1, len(map_info.nodes) - 1), rd(1, len(map_info.nodes)
- 1), rd(1, 1000) / 10)
        if isValid(tmp_action):
            ACTIONS.append(tmp_action)

    # 随机出兵
    return ACTIONS

class player_class: #格式要求
    def __init__(self, player_id:int):
        self.player_id = player_id

    def player_func(self, map_info):
        return player_func(map_info, self.player_id)
```

#以上代码仅为格式示例

五 异常处理

当玩家提交代码后，实际作战过程中，系统会在后台代码竞技场运行双方的代码进行游戏。当任何一方的代码出现异常行为时，就会触发异常处理机制，忽略该玩家本回合的所有运输行为。常见的异常行为包括但不限于：

- 玩家返回的列表中，从同一个节点输送出去的兵力总和超过了该节点存储的兵力(透支)；
- 玩家返回的列表中，存在并不直接相连的两个节点之间的运输操作；
- 玩家返回的列表中，出现非法或不合理的数据类型(比如运输兵力为负或为字符串等等...)在此提醒大家请勿投机取巧，技术组对于异常返回类型的考虑非常周全。
- 对于单回合，某玩家的算法在时间限制`config.MAX_TIME`范围内没有返回结果。

六 关于测试与调试

6.1 关于统一公布的AI

技术组提供了若干个公开的、较为基础的AI文件，供各小组强化自己的算法。其中包括随机移动AI，无脑冲锋AI，以及攻守兼备、强度偏高的AI。各小组可以利用这些AI，根据自身情况逐步调试、开发自己的算法。目前在线上代码竞技场中稳定运行的AI有：

- **测试工程师们心满意足地离开了酒吧。（随机**；是随机样例，里面的代码和4.2的示例一样。

- **测试代码随机**：另一个可以正常运行的随机AI
- **测试工程师**：一个攻守兼备，强度较高的AI

6.2 本地调试工具

- **方法一** 在命令行窗口中输入以下命令：

```
python3 debuggerCmd.py player_1.py player_2.py
```

其中`player_1.py,player2.py`为同文件夹下对战双方提交的python文件名。

- **方法二** 各小组可以直接运行`debugger/debuggerCmd.py`，选取两个python文件进行调试。

成功运行之后，会生成复盘数据存储于`debugger/output.json`文件中。

6.3 复盘数据可视化工具

- **方法一** 首先在命令行窗口输入`pip install flask`，再运行`debugger/debugVisualizer.py`，按指示进行复盘数据可视化。
- **方法二** 访问https://mi.js.org/dsa21vis/battleground.html。将`output.json`文件拖入网页界面，就可以按网页指示进行复盘数据可视化。

对于本地调试及复盘数据可视化的步骤细节，可参见`debugger/README.md`文件。

七 模拟赛和正式赛

7.1 模拟赛

模拟赛从即日起开始到6月11日结束。所有同学都可以注册个人用户，上传自己的文件至代码竞技场自由对战，厮杀。代码竞技场的网页链接：http://gis4g.pku.edu.cn/ai_arena/game/5/。模拟赛天梯分不计入大作业成绩。

7.2 正式赛

暂定于6月10日`18:00`发放小组账号，各小组以小组账号登录，上传正式参赛代码文件（限一份）。上传时间期限为6月11日`18:00`。注意提交后小组的参赛代码文件**不可修改**。

6月12日-6月14日为天梯赛，代码竞技场选取天梯前八名出线，进入淘汰赛决赛。

最后6月15日在课堂进行淘汰赛决赛，决出冠亚季军，发放奖品、纪念品等。保留人机作战为彩蛋。

祝各小组大作业顺利~