2022-2023 第二十七届智能体大赛 - 游戏文档

Edition: 0318-RC3

一、游戏规则简介

本游戏属于**塔防游戏**。双方玩家建设防御塔、使用辅助技能来防守己方基地，还需要升级基地来增强己方攻势，最终破坏掉对方基地以取得最终胜利。本游戏的核心在于双方的“工蚁”机器人并不直接被玩家所操纵，而是通过**信息素**介入的寻路算法（参见“‘工蚁’寻路算法”一节）自主适应地图环境。面对自适应的“工蚁”，如何动态调整策略、平衡进攻与防御的资金投入是决定胜负的关键。

本游戏是**回合制游戏**，双方玩家按照顺序依次传达自己操作后，游戏逻辑会进行一回合的结算，并告知双方玩家回合结算的结果。回合数到达上限时，会根据双方玩家基地的剩余血量等标准判断胜负。

受自然界的蚂蚁启发，双方玩家所拥有的”工蚁“机器人将会根据信息素寻路算法自动尝试攻入地方基地。通过投入资金升级基地，我们可以**优化生产流水线**来提升“工蚁”机器人的组装速度，也可以**列装高级护甲**提升“工蚁”机器人的生命值上限。

为了防御敌方“工蚁”的进攻，我们可以建造多种多样的防御塔，如“一炮一个”的**重型加农炮**，专注攻速的**轻机枪**，还有经典的AOE**迫击炮**。注意，信息素算法加持的”工蚁“可能会很快适应你的防线，而且由于新建防御塔的价格是**指数级增长**的，因此需要审慎选择建造点位和升级路线。

你也可以使用各类**超级武器**为自己的进攻或防御创造有利条件，比如造成高额范围伤害的**闪电风暴**和瘫痪敌人防御塔的**EMP轰炸**。虽然效果非常强力，但是他们使用代价不菲、冷却时间很长，利用它们在关键时刻一击制胜吧！

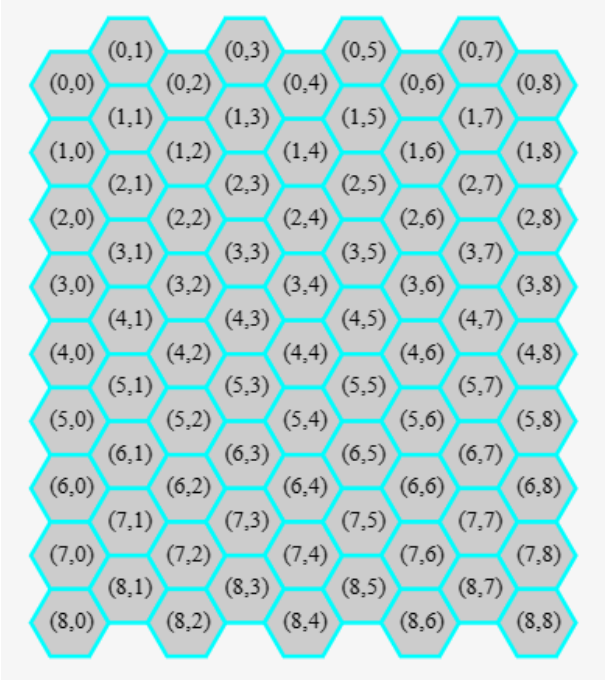
二、游戏规则详解

1. 基本数据

- 本游戏回合数从**0**开始，双方分别操作一次算作一回合。回合数到达512时结束游戏。
- “工蚁”具有年龄。年龄等于**当前回合数-生成回合数**。当年龄大于32时，蚂蚁死亡/消失。
- 规定AI每回合运行时间上限为**1秒**。
-

2. 游戏地图与坐标

本游戏采用六边形格点地图，采用“even-q”坐标系，如右图所示：



每个格点都有六个方向，计算相邻坐标的算法如下所示。需要额外注意的就是本坐标系在奇数列和偶数列需要分别处理。

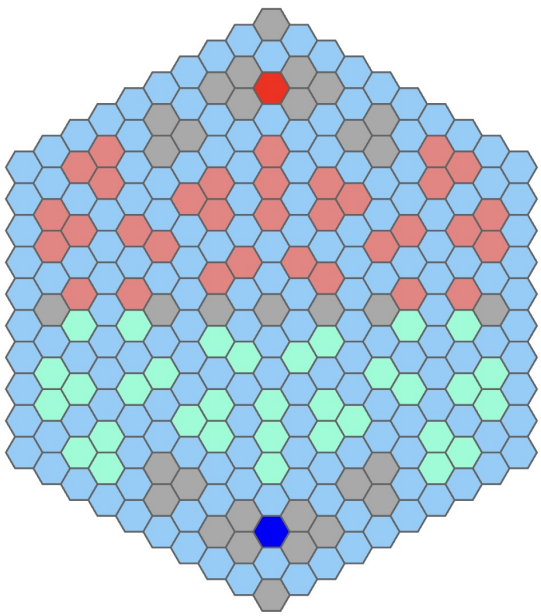
六个下标的顺序为右上、上、左上、左下、下、右下。

```
direction_difference = [[
    # even columns
    [0,  1], [-1, 0], [0, -1],
    [1, -1], [ 1, 0], [1,  1],
],[
    # odd columns
    [-1,  1], [-1,  0], [-1, -1],
    [ 0, -1], [ 1,  0], [ 0,  1],
]]

def calculate_neighbor(pos, direction):
    diff = direction_difference[place % 2][direction]
    return [pos[0] + diff[0], pos[1] + diff[1]]

```

本游戏的地图是一个边长为10的正六边形，也就是到中心点(9,9)的距离小于等于9的所有点的集合，如右图所示。其中：



- 红色网格(2,9)为玩家0的基地
- 深蓝色网格(16,9)为玩家1的基地
- 粉色区域是玩家0可以放置防御塔的区域
- 淡绿色区域是玩家1可以建造防御塔的区域
- 浅蓝色网格（以及两个基地）为蚂蚁可移动的区域，而其余颜色是蚂蚁不可移动的区域。

下面是不可移动区域的坐标列表：

```
{
    {6, 1}, {7, 1}, {9, 1}, {11, 1}, {12, 1}, {4, 2}, {6, 2}, {8, 2},
    {9, 2}, {11, 2}, {13, 2}, {4, 3}, {5, 3}, {13, 3}, {14, 3}, {6, 4},
    {8, 4}, {9, 4}, {11, 4}, {3, 5}, {4, 5}, {7, 5}, {9, 5}, {11, 5},
    {14, 5}, {15, 5}, {3, 6}, {5, 6}, {12, 6}, {14, 6}, {2, 7}, {5, 7},
    {6, 7}, {8, 7}, {9, 7}, {10, 7}, {12, 7}, {13, 7}, {16, 7}, {1, 8},
    {2, 8}, {7, 8}, {10, 8}, {15, 8}, {16, 8}, {0, 9}, {4, 9}, {5, 9},
    {6, 9}, {9, 9}, {12, 9}, {13, 9}, {14, 9}, {18, 9}, {1, 10}, {2, 10},
    {7, 10}, {10, 10}, {15, 10}, {16, 10}, {2, 11}, {5, 11}, {6, 11}, {8, 11},
    {9, 11}, {10, 11}, {12, 11}, {13, 11}, {16, 11}, {3, 12}, {5, 12}, {12, 12},
    {14, 12}, {3, 13}, {4, 13}, {7, 13}, {9, 13}, {11, 13}, {14, 13}, {15, 13},
    {6, 14}, {8, 14}, {9, 14}, {11, 14}, {4, 15}, {5, 15}, {13, 15}, {14, 15},
    {4, 16}, {6, 16}, {8, 16}, {9, 16}, {11, 16}, {13, 16}, {6, 17}, {7, 17},
    {9, 17}, {11, 17}, {12, 17}
}
```

下面是粉色区域的坐标列表：

```
{
    {6, 1}, {7, 1}, {4, 2}, {6, 2}, {8, 2}, {4, 3}, {5, 3}, {6, 4},
    {8, 4}, {7, 5}, {5, 6}, {5, 7}, {6, 7}, {8, 7}, {7, 8}, {4, 9},
    {5, 9}, {6, 9}, {7, 10}, {5, 11}, {6, 11}, {8, 11}, {5, 12}, {7, 13},
    {6, 14}, {8, 14}, {4, 15}, {5, 15}, {4, 16}, {6, 16}, {8, 16}, {6, 17}, {7,
}
```

下面是淡绿色区域的坐标列表：

```
{
    {11, 1}, {12, 1}, {9, 2}, {11, 2}, {13, 2}, {13, 3}, {14, 3}, {9, 4},
    {11, 4}, {11, 5}, {12, 6}, {10, 7}, {12, 7}, {13, 7}, {10, 8}, {12, 9},
    {13, 9}, {14, 9}, {10, 10}, {10, 11}, {12, 11}, {13, 11}, {12, 12}, {11, 13}
    {9, 14}, {11, 14}, {13, 15}, {14, 15}, {9, 16}, {11, 16}, {13, 16}, {11, 17}
}
```

3. 经济系统

- 玩家初始具有50金币。
- 每回合**结束**时，给予双方玩家1金币。
- **在结算阶段**，本回合每使用防御塔或超级武器杀死一只对方蚂蚁，按照对方蚂蚁的等级，等级1的蚂蚁给予3金币，等级2的蚂蚁给予5金币，等级3的蚂蚁给予7金币。其余原因死亡的蚂蚁不对金币产生影响。

4. 防御塔

- **建造价格**：建造**新的**防御塔的价格为 15×2^i，其中i为当前存在的己方防御塔的数量。或者说，建造第一个防御塔的金额为15，建造第二座防御塔的价格为30，建造第三座防御塔的价格为60，以后每座防御塔的建造价格均翻倍。
- **升级价格**：防御塔一共分为三个等级，共13种类。等级1升级到等级2需要60金币，等级2升级到等级3需要200金币。
- **降级、拆除返还**：降级、拆除防御塔会返还 80% 的建造花销，即等级3降级为等级2返还80金，等级2降级为等级1返还48金，拆除等级1返还 12×2^i 金，其中i为**拆除之后**己方防御塔的数量。
- **防御塔攻击**：每种类防御塔都有固定的伤害、攻击间隔、攻击次数、攻击方式、攻击范围。*攻击间隔为K，攻击次数为M*意为防御塔内置CD为0时，可以进行M次攻击，若攻击到任何“工蚁“，则重置CD为K，否则CD不变。用代码表示为：

```
tower.cd = max(0, tower.cd - 1);
if (tower.cd == 0) {
  if (has_targets_in_range(tower)) {
      for (i of 0..M) tower.attack();
      tower.cd = K;
  }
}
```

还需指出：建造、升级、降级防御塔都会令内置CD为攻击间隔K。

- **索敌逻辑**：选取范围内血量大于0且非己方的敌人，按照与防御塔的距离为主键、“工蚁”的ID为附键升序排列。顺序越靠前攻击优先级越高。用代码表示为：

```
targets = ants_in_range(tower.x, tower.y, tower.range)
            .filter((ant) => ant.hp > 0 && ant.player != tower.player)
            .sort((a, b) => a_dist == b_dist ? a.id - b.id : a_dist - b_dist)
// a_dist = distance([tower.x, tower.y], [ant.x, ant.y])
```

- 若非特殊说明，默认的攻击方式就是对优先级最高的“工蚁”造成一定的伤害。即：

```
bool attack() {
  targets = find_targets();
  if (targets.empty()) return false;
  targets[0].hp -= this.atk;
  return true;
}
```

- **AOE攻击**：部分防御塔拥有“AOE”(Area of effect)攻击属性。若为“范围为R的AOE攻击”，即为对优先级最高的“工蚁”及到其所在格距离不大于R的格子中所有“工蚁”均造成伤害。**这有可能影响到本来不处于攻击范围内的敌方“工蚁”。**
- **防御塔数据列表**

	A	B	C	D	E	F
1	**类型ID**	**名称**	**伤害**	**间隔**	**范围**	**攻击方式**
2	0	Basic	5	2	2	Default
3						
4	1	Heavy	15	2	2	Default
5	11	Heavy+	35	2	2	Default
6	12	Ice	15	2	2	Default，但会“冻结”造成伤害的蚂蚁一回合
7	13	Cannon	50	4	3	Default
8						
9	2	Quick	6	1	3	Default
10	21	Quick+	8	0.5	3	Default
11	22	Double	10	1	4	最多可以攻击优先级前二的目标
12	23	Sniper	13	2	6	Default
13						
14	3	Mortar	16	4	3	范围为1的AOE
15	31	Mortar+	35	4	4	范围为1的AOE
16	32	Pulse	30	3	2	同时攻击范围内所有目标
17	33	Missile	45	6	5	范围为2的AOE

5. 超级武器

- 选手可以使用一定的金币，在指定的位置发射超级武器。超级武器具有一定的冷却时间。
- 超级武器列表：

	A	B	C	D	E
1	**ID**	**名称**	**花费**	**冷却**	**效果**
2	1	闪电风暴 Lightning Storm	150	100	令指定位置**范围为3**的区域内进入“闪电风暴”状态，**持续20回合**。围内的**所有**敌方“工蚁”造成100伤害。
3	2	EMP轰炸 EMP Blaster	150	100	令指定位置**范围为3**的区域内陷入**“电磁脉冲干扰”**状态，**持续20回**法在有“电磁脉冲干扰”的区域内建造、升级、降级防御塔（超级）。在“电磁脉冲干扰”区域内的敌方防御塔无法攻击，内置CD不
4	3	引力护盾 Deflectors	100	50	令指定位置**范围为3**的区域内进入**“引力护盾”**状态，**持续10回合**。盾”区域内的己方“工蚁”免疫单次小于自身最大生命值50%的伤等于自身最大生命值50%的伤害不影响。
5	4	紧急回避 Emergency Evasion	100	50	**立刻**给予指定位置**范围为3**的区域内的所有我方“工蚁”**2层“紧急**“紧急回避”可以抵消一次防御塔造成的伤害（优先于“引力护盾”结算）。不会过期。

6. 基地

- 每方基地在游戏开始时有50血量，每当一只敌方“工蚁”移动到基地，则会减少1血量。当有一方基地血量不大于0时游戏立即结束。
- 基地可以进行如下升级：
- **优化生产流水线**：当前回合被 K 整除时，都会在基地处建造一只“工蚁”。

	1级（初始）	2级	3级
K	4	2	1

- **列装高级护甲**：产生“工蚁”机器人的最大生命值。注意这一属性升级时，已产生的“工蚁”最大生命值**不会改变**。

	1级（初始）	2级	3级
最大生命	10	25	50

- **花费**：1级升级为2级：200。2级升级为3级：250。无法降级。
- 一回合只能对基地进行一种升级。

7. “工蚁”寻路算法

- **可移动区域**：如“游戏地图与坐标”一节中所示的浅蓝色区域都是可移动区域，另外，“工蚁”不会走**回头路**（即本回合不会以上回合所在的位置为移动目标）。
- **信息素**：信息素是寻路算法的核心。
  - 每个网格都存在双方玩家独立的信息素数值
  - **初始值**：游戏开始时每一格的信息素为 $\tau_0 + \epsilon$, $\tau_0 = 10$, $\epsilon \sim U(-2, 2)$（即为[-2,2]区间的均匀分布，由随机数种子生成），每格独立，双方独立
  - **更新**：
    - 当有”工蚁“**攻入敌方基地**，它所走过的路径上的所有点的信息素都会 $\Delta\tau_1 = +10$
    - 当有”工蚁“因为**生命值耗尽而死亡**，它所走过的路径上的所有点的信息素都会 $\Delta\tau_2 = -5$
    - 当有”工蚁“因为**年龄过大而死亡**，它所走过的路径上的所有点的信息素都会 $\Delta\tau_3 = -3$
    - 注意，上述更新对于“工蚁”重复经过的点都只更新一次。
    - **每回合**更新信息素时，按照以下公式进行：$\tau'_P = \lambda\tau_P + (1-\lambda)\tau_0 + \sum_k \Delta\tau_P^{(k)}$。其中 $\lambda = 0.97$ 为信息素衰减比例。$\Delta\tau_P^{(k)}$ 为第 k 只“工蚁”对点 P 产生的信息素贡献/变化。
  - **目标吸引度**：“工蚁”的目标是攻入敌方基地
    - 设“工蚁”现在的位置为P，相邻的位置共有六个 $P_d, d = 0, 1, \cdots, 5$。那么对应方向移动的**吸引度**为
    - $\eta_d = \eta(\text{dist}(P_d) - \text{dist}(P)) = \eta(\Delta D) = \begin{cases} 1.25, \Delta D = -1 \\ 1.00, \Delta D = 0 \\ 0.75, \Delta D = 1 \end{cases}$
    - 其中 $\text{dist}(P)$ 为点P到敌方基地的距离。即向靠近敌方基地的方向移动要更具有**吸引力**。
- **寻路算法**：
  a. 设六个方向的相邻点的坐标分别为 $P_d, d = 0, 1, \cdots, 5$
  b. 有效性向量：$\vec{v} = (v(P_0), \cdots, v(P_5)), v(P) = \begin{cases} 1, P\text{可移动} \\ 0, P\text{不可移动} \end{cases}$
  c. 信息素向量：$\vec{\tau} = (\tau_{P_0}, \cdots, \tau_{P_5})$
  d. 吸引度向量：$\vec{\eta} = (\eta_0, \cdots, \eta_5)$
  e. 移动向量：$\vec{P} = \vec{v} \cdot \vec{\tau} \cdot \vec{\eta}$
  f. 最终移动方向：$d = \text{maxidx}(\vec{P})$
  g. 若存在 $\vec{P}$ 内元素相同的情况，优先取信息素更高的方向。若还相同，取较小的方向值。

8. 胜负判定

- 大本营剩余血量多者，胜。如果剩余血量相同：

- 击败对方蚂蚁数多者，胜。如果击败蚂蚁数相同：
- 使用超级武器少者，胜。如果使用超级武器次数相同：
- AI用时少者，胜。如果用时相同：
- 先手胜。

9. 结算流程

1. 玩家操作
    a. 等待玩家0的操作，若玩家0程序崩溃、运行超时、返回了不符合协议的数据、执行了非法的操作，那么立刻判负。
    b. 执行玩家0的操作，如建造、升级防御塔，升级基地、使用超级武器等。
    c. 然后对玩家1执行同样的步骤
2. 回合结算
    a. 结算闪电风暴
    b. 按照建造顺序结算防御塔攻击。
    c. 若本回合造成敌方”工蚁“死亡（生命值变为非正数），则按照死亡的”工蚁“的等级获得金币。
    d. 如果蚂蚁的年龄大于最大年龄，即**当前回合数-蚂蚁生成回合数大于32**，标记为老死。
    e. 按照生成顺序结算蚂蚁移动。
        i. 已死亡的蚂蚁不可以移动。
        ii. 被冻结的蚂蚁这回合无法移动，并解除冻结状态。
        iii. 如果蚂蚁移动到敌方基地内，那么立刻减少敌方基地血量，并判断是否降为0，若降为0，游戏结束。
    f. 结算信息素并移除已死亡、已到达的蚂蚁。
    g. 按照基地等级尝试生成蚂蚁。
    h. 双方金币+1
    i. 回合数+1
    j. 若回合数等于512，执行胜负判断，结束游戏

三、选手AI编写指南

10. 输入输出

选手AI可以从标准输入流**直接读取**来自评测系统的信息。

但是，选手AI通过标准输入流向评测系统返回信息时，需要在发送的信息之前添加一个**四字节大端序**整数，代表信息的长度。如选手想要输出以下信息：

```
1 2
2 1 2 3 4 5
```

这条信息的长度为11，其十六进制数据表示为（其中第二行的 `_` 为空格）：

```
1 HEX : 32 0A 31 20 32 20 33 20 34 20 35
2 TEXT: 1 \n  1  _  2  _  3  _  4  _  5
```

因此需要在数据包前面加上11的四字节大端序表示 `00 00 00 0B`，最终结果为：

```
1 HEX : 00 00 00 0B 32 0A 31 20 32 20 33 20 34 20 35
2 TEXT:  0  0  0 11  1 \n  1  _  2  _  3  _  4  _  5
```

选手AI可以通过向标准错误流输出信息进行调试，在Saiblo平台上评测完成后，会提供标准错误流产生的信息。请注意最终提交时请尽量减少调试信息的输出，因为这可能会占用大量运行时间。

11. 评测流程

- 总体评测流程如下
  a. 平台启动双方玩家AI程序，并向双方玩家发送初始化信息
  b. 每一回合均按照以下流程顺序、循环执行
    i. 等待先手玩家操作
    ii. 平台接收到先手玩家操作后，进行验证和执行，如果成功完成，将操作转发给后手玩家
    iii. 等待后手玩家操作
    iv. 平台接收到后手玩家操作后，进行验证和执行，如果成功完成，将操作转发给先手玩家
    v. 游戏逻辑进行一回合的结算，如果回合正常结束，则将局面信息通过平台分别发送给两位玩家。如果游戏结束，那么会向平台汇报游戏结果，终止评测流程。
- 对于**先手玩家**的AI，执行流程如下：
  a. 接受初始化信息，判断自己是先手玩家
  b. 每一回合中：
    i. 进行决策，发送自己的操作

ii. 等待接受后手玩家的操作

iii. 等待接受局面信息

- 对于**后手玩家**的AI，执行流程如下：

a. 接受初始化信息，判断自己是后手玩家

b. 每一回合中：

i. 等待接受先手玩家的操作

ii. 进行决策，发送自己的操作

iii. 等待接受局面信息

12. 游戏初始化信息

一行两个整数 `K M` ， `K == 0` 代表自己为先手玩家P0， `K == 1` 代表自己为后手玩家P1； `M` 为随机数种子，用于计算初始局面的信息素，计算算法如下：

```
unsigned long long lcg_seed;
unsigned long long lcg(){
    lcg_seed = (25214903917 * lcg_seed) & ((1ll << 48) - 1);
    return lcg_seed;
}

void init_pheromon(unsigned long long M){
    lcg_seed = M;
    for(int i = 0; i < 2; i++)
        for(int j = 0; j < MAP_SIZE; j++)
            for(int k = 0; k < MAP_SIZE; k++)
                pheromone[i][j][k] = lcg() * pow(2, -46) + 8;
}
```

```
lcg_seed = 0
def lcg():
    global lcg_seed
    lcg_seed = (25214903917 * lcg_seed) & ((1 << 48) - 1)
    return lcg_seed

def init_pheromon(M):
    global lcg_seed
    lcg_seed = M
    for i in [0,1]:
        for j in range(0, MAP_SIZE):
            for k in range(0, MAP_SIZE):
```

```
13                  pheromone[i][j][k] = lcg() * pow(2, -46) + 8
```

13. 玩家操作信息

第一行一个整数 N，代表操作数。接下来 N 行，每行表示一个操作，第一个整数 T 代表操作类型，后面需要根据操作类型提供一些参数。操作类型和参数如下表：

操作名称	操作类型	操作参数与描述	例子
建造防御塔	`11`	`x y` 建造位置	`11 11 1` 在 `(11, 1)` 处建
升级防御塔	`12`	`towerId towerTypeId` 将ID为 `towerId` 的防御塔升级为 `towerTypeId` 所代表的防御塔类型	`12 1 31` 将ID为 `1` 的防御塔也即 `Mortar+`
降级防御塔	`13`	`towerId` 降级ID为 `towerId` 的防御塔。如果已经是 `Basic` 防御塔，那么会将其**拆除**。	`13 1` 降级ID为 `1` 的防御塔。`Basic` 防御塔，那么会将其
超级武器	`21/22/23/24`	`x y` 施放超级武器的位置。21/22/23/24分别代表使用ID为1/2/3/4的超级武器。	`21 9 9` 在 `(9, 9)` 的位置
基地升级	`31/32`	无参数。31/32分别代表升级“生产流水线（出兵速度）”和升级“高级装甲（最大生命）”	`31` 升级“生产流水线（出兵

可能的非法操作/限制包括：

- 操作类型不合法
- 建造、升级防御塔、使用超级武器、基地升级所需的金币不足
- 建造防御塔、使用超级武器的位置非法
- 升级、降级防御塔所指定的防御塔ID不存在
- 升级防御塔所指定的防御塔类型不存在
- 升级防御塔只能升级到下一等级，如不可以直接从 `Basic` 直接升级到 `Heavy+`，也不允许升级到另一条线路上，如 `Quick` 升级为 `Heavy+`
- 一回合之内，只能对一个ID的防御塔进行一次操作，也只能对基地进行一次操作。即不允许连续升级、不允许建造后直接升级、不允许升级后降级等。
- 使用的超级武器还在冷却
- 指定的基地升级已经到最高等级

如下是一个完整的操作信息的示例：

```
```

```
5
11 11 1
12 1 31
13 2
21 9 9
31
```

14. 局面信息

第一行一个整数 `R` ，表示回合数。

第二行一个整数 `N_1` ，代表场上总防御塔数量。接下来 `N_1` 行，每行6个整数，分别为 `id player x y type cd` ，即每个防御塔的ID、归属、坐标、类型、攻击CD。

接下来一个整数 `N_2` ，代表场上总“工蚁”数量。接下来 `N_2` 行，每行8个整数，分别为 `id player x y hp lv age state` ，即每只“工蚁”的ID、阵营、坐标、当前生命、等级、 当前寿命、当前状态。

- 蚂蚁等级分别为：0/1/2
- 当前状态：蚂蚁一共有5种状态，对应 0-4

```
enum State {
  Alive,   // Still alive
  Success, // Reach the other camp
  Fail,    // No HP
  TooOld   // Too old
  Frozen   // Frozen
};
```

接下来两个整数 `G0 G1` ，分别代表先后手玩家的剩余金币。

接下来两个整数 `HP0 HP1` ，分别代表先后手玩家的剩余基地血量。

四、AI SDK使用指南

为了方便玩家编写AI，我们分别提供了C++和Python的SDK库。我们的SDK提供各种层面的支持

1. `rawio`

提供AI编写指南中所提到的输入输出协议支持。

- `write_to_judger` ：将信息通过4+N协议转化后再输出。

- `debug`：实际上是向标准错误流输出信息。这些信息被作为调试信息呈现在saiblo评测结果页面中。

```python
import rawio

def write_to_judger(msg: str) -> None
def debug(msg: str) -> None
```

```cpp
#include "rawio.h"

void write_to_judger(const std::string& msg);
void debug(const std::string& msg);
```

2. `coord`

提供坐标相关的帮助函数。

- `Coord`：描述坐标的简单结构体
- `is_in_map`：是否在地图中。
- `is_ant_can_go`：是否为蚂蚁可以移动的位置。等价于 `is_in_map(c) && !is_highland()`
- `is_highland`：是否为高台，也即蚂蚁无法行动的区域。
- `is_player_highland`：是否为对应玩家控制的/可以建造防御塔的高台。
- `distance`：计算两坐标之间的距离。即两点之间最短路径的长度。相同点之间的距离定义为0。
- `neighbour`：计算坐标对应方向的相邻点的坐标。

```python
import coord

class Coord:
    x: int
    y: int

def is_in_map(coord: Coord) -> bool
def is_ant_can_go(coord: Coord) -> bool
def is_highland(coord: Coord) -> bool
def is_player_highland(coord: Coord, player: int) -> bool

def distance(c0: Coord, c1: Coord) -> int
def neighbour(coord: Coord, direction: int) -> Coord
```

3. `gamedata`

提供对蚂蚁、防御塔、超级武器的相关封装。

- `Ant`：蚂蚁
  - `id`：编号，从0开始。
  - `player`：所属玩家。
  - `hp` `maxhp` `level`：蚂蚁的当前生命值，最大生命值，等级。
  - `age`：蚂蚁的年龄。
  - `evasion_count`：蚂蚁剩余的“紧急回避”次数。
  - `state`：蚂蚁的状态
  - `path`：蚂蚁经过的路径点的列表。
- `Tower`：防御塔
  - `id`：编号，从0开始。
  - `player`：所属玩家。
  - `type`：`TowerType` 防御塔的类型
  - `cd`：防御塔冷却。若为0，说明冷却完成，当回合可以攻击。
- `SuperWeapon`：超级武器
  - `player`：所属玩家。
  - `type`：`SuperWeaponType` 超级武器类型。
  - `duration`：超级武器持续时间，大于0为还在生效。
  - `coord`：部署坐标。

```python
import gamedata

class AntState(IntEnum):
    Alive = 0
    Success = 1
    Fail = 2
    TooOld = 3
    Frozen = 4

class Ant:
    id: int
    player: int
```

```
    hp: int
    maxhp: int
    level: int
    age: int
    evasion_count: int
    state: AntState
    path: list[Coord]

class TowerType(IntEnum):
    Basic = 0
    Heavy = 1
    HeavyPlus = 11
    Ice = 12
    Cannon = 13
    Quick = 2
    QuickPlus = 21
    Double = 22
    Sniper = 23
    Mortor = 3
    MortorPlus = 31
    Pulse = 32
    Missile = 33

class Tower:
    id: int
    player: int
    type: TowerType
    cd: int

class SuperWeaponType(IntEnum):
    LightningStorm = 1
    EMPBlaster = 2
    Deflectors = 3
    EmergencyEvasion = 4

class SuperWeapon:
    player: int
    type: SuperWeaponType
    duration: int
    coord: Coord

```

```
#include "gamedata.h"

struct Ant {
```

```cpp
    int id, player, hp, maxhp, level;
    int evasion_count;
    std::vector<Coord> path;
}
```

4. `protocol`

```python
import protocol

class InitInfo():
    self_player: int
    seed: int

def read_init_info() -> InitInfo

class OperationType(IntEnum):
    BuildTower = 11
    UpgradeTower = 12
    DowngradeTower = 13
    DeployLightningStorm = 21
    DeployEMPBlaster = 22
    DeployDeflector = 23
    DeployEmergencyEvasion = 24
    UpgradeGenerateSpeed = 31
    UpgradeAntMaxHP = 32

class Operation:
    type: OperationType
    arg0: int
    arg1: int

def build_tower_op(coord: Coord) -> Operation
def upgrade_tower_op(id: int, type: TowerType) -> Operation
def downgrade_tower_op(id: int) -> Operation
def deploy_super_weapon(type: SuperWeaponType, coord: Coord) -> Operation
def upgrade_generate_speed_op() -> Operation
def upgrade_ant_maxhp_op() -> Operation

def read_enemy_operations() -> list[Operation]
def write_our_operation(ops: list[Operation]) -> None

class RoundInfo:
    ants: list[Ant]
    towers: list[Tower]
    coin: [int, int]
```

```
    hp: [int, int]

def read_round_info() -> RoundInfo
```

5. pheromone

```
import pheromone

class Pheromone:
    value: list[list[float]]

    def init(seed: int) -> None

    def decay_rate() -> float
    def decay() -> None

    def pheromone_of_neighbours(coord: Coord) -> list[float]
    def multiplier_of_neighbours(coord: Coord, target: Coord) -> list[float]
    def next_move_direction(ant: Ant) -> int

    def modify_path(path: list[Coord], delta: float) -> None
    def modify_by_success_ant(ant: Ant) -> None
    def modify_by_failed_ant(ant: Ant) -> None
    def modify_by_too_old_ant(ant: Ant) -> None
```

6. gamestate

```
class GameState:
    ants: list[Ant]
    towers: list[Tower]
    coin: [int, int]
    hp: [int, int]
    active_super_weapon: list[SuperWeapon]
    phero: [Pheromone, Pheromone]
    gen_speed_lv: [int, int]
    ant_maxhp_lv: [int, int]

    next_ant_id: int
    next_tower_id: int

    def ant_idx_of_id(id: int) -> int
    def ant_of_id(id: int) -> Optional[Ant]
```

```
    def ant_at(coord: Coord) -> list[Ant]

    def tower_idx_of_id(id: int) -> int
    def tower_of_id(id: int) -> Optional[Tower]
    def tower_at(coord: Coord) -> Optional[Tower]

    def build_tower(player: int, coord: Coord) -> Optional[Tower]
    def upgrade_tower(id: int, type: TowerType) -> Optional[Tower]
    def downgrade_tower(id: int) -> Optional[Tower]

    def build_tower_cost(player: int) -> int
    def upgrade_tower_cost(id: int, type: TowerType) -> int
    def downgrade_tower_income(id: int) -> int

    def upgrade_generate_speed(player: int) -> bool
    def upgrade_ant_maxhp(player: int) -> bool

    def upgrade_generate_speed_cost(player: int) -> int
    def upgrade_ant_maxhp_cost(player: int) -> int

    def set_coin(player: int, new_coin: int) -> None
    def update_coin(player: int, delta: int) -> None

    def set_hp(player: int, new_hp: int) -> None
    def update_hp(player: int, delta: int) -> None

    def pheromone_decay() -> None

    def is_operation_valid(player: int, op: Operation) -> bool
    def apply_operation(player: int, op: Operation) -> bool
```

7. `controller`

```
import controller

class GameController:
    round: int
    self_player: int
    game_state: GameState

    def init() -> None

    def read_enemy_ops() -> list[Operation]
    def apply_enemy_ops(ops: list[Operation]) -> bool
    def read_and_apply_enemy_ops() -> bool
```

```
13
14      def try_apply_our_op(op: Operation) -> bool
15      def try_apply_our_ops(op: list[Operation]) -> bool
16      def finish_and_send_our_ops() -> None
17
18      def read_and_apply_round_info() -> RoundInfo
19
20      def simulate_next_round() -> bool
21
22  def run_antwar_ai(ai_func: Callable[[GameState], list[Operations]]) -> None
23  def run_antwar_ai(
24      ai0_func: Callable[[GameState], list[Operations]],
25      ai1_func: Callable[[GameState], list[Operations]]) -> None
```

Without `run_antwar_ai`

```
1  import * from controller
2
3  def make_decision_for_player_0(game: GameController) -> list[Operation]:
4      # User code
5
6  def make_decision_for_player_0(game: GameController) -> list[Operation]:
7      # User code
8
9  game = GameController()
10 game.init()
11
12 while(True):
13     if game.self_player == 0:
14         ops = make_decision_for_player0(game)
15         game.try_apply_our_operations(ops)
16         game.finish_and_send_our_operations()
17
18         game.read_and_apply_enemy_operations()
19
20         game.finish_and_send_our_ops()
21     else:
22         game.read_and_apply_enemy_operations()
23
24         ops = make_decision_for_player1(game)
25         game.try_apply_our_operations(ops)
26         game.finish_and_send_our_operations()
27
28         game.finish_and_send_our_ops()
29
```

With `run_antwar_ai`

```python
import * from controller

def make_decision_for_player_0(game: GameState) -> list[Operation]:
    # User code

def make_decision_for_player_1(game: GameState) -> list[Operation]:
    # User code

run_antwar_ai(
    make_decision_for_player_0,
    make_decision_for_player_1
)
```