Python SDK 手册

选手填写函数

需要选手填写的 SDK 函数为 `Pick`, `Assert` 和 `Act` 三个函数。

Pick

选鱼函数

功能

给定当前局面，从剩余的鱼中挑选 4 条鱼出战。值得注意的是拟态鱼的格式略有不同，将在下方说明。

这个函数的调用时机是游戏刚开始的时候，或是一轮对战结束，进入下一轮对战并开始选鱼的时候。

函数原型

```
def Pick(self: AIClient, game: Game) -> list[int]: ...
```

参数

传入的参数是一个表示当前局面的类 `Game`，其定义点击这里查看。

返回值

返回的值是一个 `list[int]`.

一个合法的返回值应该包含 4 个 `int` 类型变量，其含义为挑选的 4 条鱼的 ID. 鱼的编号从 1 至 12 分别代表：

1. 射水鱼
2. 喷火鱼
3. 电鳗
4. 翻车鱼
5. 梭子鱼
6. 蝠鲼
7. 海龟
8. 章鱼
9. 大白鲨
10. 锤头鲨
11. 小丑鱼
12. 拟态鱼

其中 12 不能作为合法的挑选类型存在，因为拟态鱼需要选择一条其他鱼进行拟态，所以拟态鱼的编号应该设置为 $12 + x$，其中 x 的含义为模仿的鱼的编号。所以一个合法的返回值，其形式可能为 `[1, 2, 4, 15]`，代表出战的鱼为射水鱼、喷火鱼、翻车鱼和拟态鱼（模仿电鳗）。

当然，使用 `int` 类型表示鱼显得很不直观。为此我们提供了预定义的枚举类型 `fish_type`，使用鱼的名字进行表示，并且可以用 `int(...)` 转型。关于该枚举变量，可以点击这里查看所有鱼的定义。

Assert

断言函数

功能

给定当前局面，选择是否需要**断言**一条敌方的鱼。关于断言是什么，可以查看游戏规则介绍文档。

这个函数的调用时机是一个回合开始时，此时选手需要断言。由于此时对方理应完成了一个回合的操作，所以 `Game` 中的一些信息会对应更新。

函数原型

```
def Assert(self: AIClient, game: Game) -> tuple[int, int]: ...
```

参数

传入的参数是一个表示当前局面的类 `Game`，其定义点击这里查看。

返回值

返回的值是一个 `tuple[int, int]`.

对于断言操作而言，我们只需要获取两个变量：敌方鱼的**位置**和“断言”敌方鱼的 **ID**。所以返回值的含义十分清楚：第一个 `int` 代表的是敌方鱼的**位置**，其取值范围为 $[0, 3]$；第二个 `int` 代表的是敌方鱼的 ID，其取值范围为 $[1, 12]$. 敌方鱼的 ID 也可以使用枚举类型 `fish_type` 转型后来表示。关于该枚举变量，可以点击这里查看所有鱼的定义。

当然，如果你不能确定敌方鱼是什么，可以令第一个 `int` 值为 -1，来表示这一回合放弃断言。

所以一个合法的返回值，其形式可能为 `(0, 3)`，其含义为断言敌方 0 号鱼是电鳗；还有可能是 `(-1, -1)`，其含义为我方该回合放弃断言。

提示

如果你没有把握自己的 AI 有一定的推断能力，那么建议 AI 不要轻易断言——因为断言失败会带来很大的负面效果！所以如果你对自己的 AI 没有信心，可以复制下面这段代码来跳过断言阶段：

```
return (-1, -1)
```

Act

行动函数

功能

给定当前局面，选择这一回合需要进行的行动。

这个函数的调用时机是断言阶段结束后，行动回合开始阶段。如果你在断言阶段进行了一次断言，那么 `Game` 里的内容会随之进行更新（主要是 `my_assert`）。

函数原型

```
def Act(self: AIClient, game: Game) -> Action: ...
```

参数

传入的参数是一个表示当前局面的类 `Game`，其定义点击这里查看。

返回值

返回值是一个类 `Action`，代表这一回合做出的行动。关于 `Action`，可以点击这里查看定义和使用方法。

Action 的简单使用

由于每条鱼的主动技能不一样，所以做出的操作会比较复杂。为了让这件事情变得更简单一些，我们定义了一个 `Action` 类，里面存有这一回合的操作。下面将介绍怎么使用 `Action` 类进行行动的操作。

首先，选手需要做的事情是创建一个 `Action` 类的实例：

```
action = Action(game)
```

这一行代码不需要改动。这样我们就拥有了一个 `Action` 类实例。接下来只需要向 `action` 中添加行动，并返回即可。

其次，选手需要挑选一条该回合能行动的鱼，并通过 `set_action_fish` 函数向 `action` 中添加行动的鱼的信息。举个例子，假设我方 2 号位的鱼是海龟，我想让海龟这一回合进行行动，那么我需要写以下代码：

```
action.set_action_fish(2)
```

参数 `2` 代表行动的鱼是 2 号位的鱼。

然后，选手需要指定选中的鱼这一回合是需要进行普通攻击还是使用技能。假设我让海龟使用技能，那么我需要写以下代码：

```
action.set_action_type(1)
```

参数 `1` 代表使用主动技能，`0` 代表使用普通攻击。

接下来，由于海龟的主动技能需要选中一个友方随从和一个敌方随从，所以在返回 `action` 之前还需要再指定两个目标。对于指定目标为友方鱼的情况，我需要使用 `set_friend_target` 来选择友方随从。假设我选择友方 1 号鱼为友方目标，那么我需要写以下代码:

```
action.set_friend_target(1)
```

参数 `1` 代表选中的友方目标是位置为 1 的友方鱼。

最后我们考虑选择敌方目标。我需要使用 `set_enemy_target` 函数来选择敌方随从。假设我选择敌方 3 号鱼为目标，那么我需要写以下代码：

```
action.set_enemy_target(3)
```

参数 `3` 代表选中的敌方目标是位置为 3 的敌方鱼。

这时候，我已经设置完了全部的行动参数了！所以我需要提交我的行动，将 `action` 返回：

```
return action
```

这样子，我就完成了这个函数的任务了。

注意：以上函数的调用次序最好不要更改！否则可能会导致设置行动失败！

选手可调用函数

debug_msg

该函数的作用是将一个给定字符串**追加**输出到一个名为 **ai_debug_info.txt** 的文件里，可以输出调试信息。

注意：该函数只能在本地运行的时候使用，不能在 saiblo 上使用，否则可能导致 AI 错误！

函数原型

```
def debug_msg(self: AIClient, str: str) -> None: ...
```

使用示范

```
self.debug_msg("This is a debug info")
```

等到本地运行结束后，在 **judger.py** 文件的同级目录下会出现一个名为 ai_debug_info.txt 的文件（若已存在，则在末尾追加新内容），里面写有“This is a debug info”。

关于本地运行，可以参考文档：**judger 使用说明。**

get_my_remain_fishes

该函数的作用是将当前剩余可选的鱼以 `list[int]` 的形式返回给选手，一般在选鱼阶段使用。

函数原型

```
def get_my_remain_fishes(self: AIClient) -> list[int]: ...
```

使用示范

```
remain_fishes = self.get_my_remain_fishes()
```

get_enemy_id

该函数的作用是获取对方指定位置的鱼的 ID（或者称之为**种类**）。如果这一条鱼已经被断言成功，则返回这条鱼的 ID；如果这一条鱼还没有被断言成功，则返回 -1.

可以直接访问 `Game` 中的变量来替代本函数（`pos` 为敌方鱼的位置）：

```
enemy_id = game.enemy_id[pos].id
```

函数原型

```
def get_enemy_id(self: AIClient, pos: int) -> int: ...
```

使用示范

```
enemy_id = self.get_enemy_id(pos)
```

get_enemy_hp

该函数的作用是获取对方指定位置的鱼的 HP.

可以直接访问 `Game` 中的变量来替代本函数（`pos` 为敌方鱼的位置）：

```
enemy_hp = game.enemy_id[pos].hp
```

函数原型

```
def get_enemy_hp(self: AIClient, pos: int) -> int: ...
```

使用示范

```
enemy_hp = self.get_enemy_hp(pos)
```

get_my_id

该函数的作用是获取我方指定位置的鱼的 ID. 如果这一条鱼已经被断言成功，则返回这条鱼的 ID 的**相反数**；如果这一条鱼还没有被断言成功，则返回这条鱼的 ID.

可以直接访问 `Game` 中的变量来替代本函数（`pos` 为我方鱼的位置）：

```
my_id = game.my_id[pos].id
```

函数原型

```
def get_my_id(self: AIClient, pos: int) -> int: ...
```

使用示范

```
my_id = self.get_my_id(pos)
```

get_my_hp

该函数的作用是获取我方指定位置的鱼的 HP.

可以直接访问 `Game` 中的变量来替代本函数（`pos` 为我方鱼的位置）：

```
my_hp = game.my_id[pos].hp
```

函数原型

```
def get_my_hp(self: AIClient, pos: int) -> int: ...
```

使用示范

```
my_hp = self.get_my_hp(pos)
```

get_my_atk

该函数的作用是获取我方指定位置的鱼的攻击力。

可以直接访问 `Game` 中的变量来替代本函数（`pos` 为我方鱼的位置）：

```
my_atk = game.my_id[pos].atk
```

函数原型

```
def get_my_atk(self: AIClient, pos: int) -> int: ...
```

使用示范

```
my_atk = self.get_my_atd(pos)
```

get_lowest_health_enemy

该函数的作用是获取敌方血量最低的一条鱼。

返回值为一个 `int`，代表敌方血量最少的鱼的位置。如果有**多条**敌方鱼是敌方血量最低的鱼，则返回**位置编号最小**的鱼。

函数原型

```
def get_lowest_health_enemy(self: AIClient) -> int: ...
```

使用示范

```
# lowest_health_enemy 是一个 0~3 的 int，代表敌方鱼的位置。
lowest_health_enemy = self.get_lowest_health_enemy()
```

get_lowest_health_enemies

该函数的作用是获取敌方血量最低的**所有**鱼。

返回值为一个 `list[int]`，里面储存了**所有**敌方血量最少的鱼的位置。

函数原型

```
def get_lowest_health_enemies(self: AIClient) -> list[int]: ...
```

使用示范

```
# lowest_health_enemies 是一个 list[int]，里面存储敌方所有血量最少的鱼。
lowest_health_enemies = self.get_lowest_health_enemies()
```

get_enemy_living_fishes

该函数的作用是获取所有敌方当前存活的鱼。

返回值为一个 `list[int]`，里面储存了所有敌方活着的鱼的位置。

```
def get_enemy_living_fishes(self: AIClient) -> list[int]: ...
```

使用示范

```
# enemy_living_fishes 是一个 list[int]，里面储存敌方所有活着的鱼。
enemy_living_fishes = self.get_enemy_living_fishes()
```

get_my_living_fishes

该函数的作用是获取所有我方当前存活的鱼。

返回值为一个 `list[int]`，里面储存了所有我方活着的鱼的位置。

函数原型

```
def get_my_living_fishes(self: AIClient) -> list[int]: ...
```

使用示范

```
# my_living_fishes 是一个 list[int]，里面储存我方所有活着的鱼。
my_living_fishes = self.get_my_living_fishes()
```

get_my_living_ally

该函数的作用是获取我方一条鱼的其余存活队友。

返回值为一个 `list[int]`， 里面储存了传入的鱼之外的我方活着的鱼的位置。

与 `get_my_living_fishes` 相比，这个函数需要**额外**指定一条友方鱼，获取其余的友方活着的鱼的位置。

函数原型

```
def get_my_living_ally(self: AIClient, pos: int) -> list[int]: ...
```

使用示范

```
# 假设当前我需要发动位置为 3 的鱼的主动。
pos = 3
# 当前这条鱼的主动需要选取一个存活的队友，所以我用 my_living_ally 来表示其余的存活队友。
my_living_ally = self.get_my_living_ally(pos)
```

get_avatar_id

该函数的作用是获取己方拟态鱼本轮模仿的鱼的编号（如果存在拟态鱼）。

函数原型

```
def get_avatar_id(self: AIClient) -> int: ...
```

使用示范

```
# avatar_id 是我方拟态鱼模仿的鱼的编号。
avatar_id = self.get_avatar_id()
```

get_first_mover

该函数的作用是获取己方这一轮是否是先手，如果是先手会得到 `True`.

函数原型

```
def get_first_mover(self: AIClient) -> bool: ...
```

使用示范

```
if self.get_first_mover():
    ...  # 展开我的先手策略！
else:
    ...  # 可恶，启用备用方案！
```

get_skill_used

该函数的作用是获取自己的一条鱼的主动技能使用次数。

传入参数是己方鱼的位置，返回值是这条鱼的主动技能使用次数。

函数原型

```
def get_skill_used(self: AIClient, pos: int) -> int: ...
```

使用示范

```
# skill_used_times 表示我方第 3 条鱼使用主动技能的次数
skill_used_times = self.get_skill_used(3)
```

auto_valid_action

入门神器！新手福音！

如果你不知道行动回合该进行什么操作的话，请使用这个函数来进行自动行动。

该函数的作用是自动为一个位置的鱼选取合法的行动目标，优先高概率使用主动技能，如果不可能合法则使用普通攻击。但是在这之前请先在 `Action` 中设置你的选鱼，否则会出现错误。

注意：长时间使用主动技能可能会让对手更容易地猜出你的鱼的配置，从而进行断言获得优势。

传入参数是一个 `int`（代表鱼的位置）和一个 `Action`（代表鱼的行动）。

返回一个 `Action`（自动处理后的合法行动）。

函数原型

```
def auto_valid_action(self: AIClient, pos: int, action: Action) -> Action:
...
```

使用示范

以一个最简单的行动作示范：

```
def Act(self: AI, game: Game) -> Action:
    action = Action(game)
    # 获取我这一回合行动的鱼。
    my_pos: int = self.get_my_living_fishes()[0]
    # 设置这一回合行动的鱼。
    action.set_action_fish(my_pos)
    # 自动行动。
    return self.auto_valid_action(my_pos, action)
```

预定义枚举变量

Python AI SDK 是通过 pybind11 让 Python 调用 C++ 动态链接库实现的，所以这里的“枚举”只是借鉴 C++ 中的 `enum`，通过特殊的类进行模仿。

对于一个 C++ 枚举类型：

```
enum Foo { B = 1, A = 10, R = -1 };
```

pybind11 会翻译成如下形式的 Python 代码：

```
class Foo:
    def __eq__(self: object, other: object) -> bool: ...
    def __getstate__(self: object) -> int: ...
    def __hash__(self: object) -> int: ...
    def __index__(self: Foo) -> int: ...
    def __init__(self: Foo, value: int) -> None: ...
    def __int__(self: Foo) -> int: ...
    def __ne__(self: object, other: object) -> bool: ...
    def __repr__(self: object) -> str: ...
    def __setstate__(self: Foo, state: int) -> None: ...
    def __str__(self: Foo) -> str: ...

    @property
    def name(self: Foo) -> str: ...

    B: Foo  # = 1
    A: Foo  # = 10
    R: Foo  # = -1
```

```
19
20
21 B = Foo.B
22 A = Foo.A
23 R = Foo.R
```

其中：

- `Foo` 的构造函数只能接收 C++ 中 `Foo` 取值范围的 `int`.
  - `Foo` 有静态成员变量 `B`、`A` 和 `R`，并且为了模拟 C++ 中无作用域枚举的性质，还会在类定义之外定义与其**同名**的**引用**，例如：

    ```
    1 assert B is Foo.B
    ```

- `Foo` 的实例可以转型，例如：

  ```
  1 assert int(Foo(1)) == int(Foo.B) == 1
  2 assert str(Foo(-1)) == str(Foo.R) == "Foo.R"
  ```

- `Foo` 的实例有只读属性 `name`，例如：

  ```
  1 assert Foo.A.name == "A"
  ```

如果您想了解更多细节，可以查阅 pybind11 官方文档。

fish_type

`fish_type` 是为了让鱼的 ID 更加直观而定义的一个枚举变量，其对应关系如下：

鱼的名称	**ID**	`fish_type`
射水鱼	1	`spray`
喷火鱼	2	`flame`
电鳗	3	`eel`
翻车鱼	4	`sunfish`
梭子鱼	5	`barracuda`
蝠鲼	6	`mobula`
海龟	7	`turtle`
章鱼	8	`octopus`
大白鲨	9	`whiteshark`
锤头鲨	10	`hammerhead`
小丑鱼	11	`clownfish`
拟态鱼	12	`imitator`

使用 `fish_type` 可以替换对应的 ID，比如下面的代码：

```
chosen_fish = []
chosen_fish.append(spray)
chosen_fish.append(sunfish)
chosen_fish.append(barracuda)
chosen_fish.append(int(imitator) + int(turtle))  # fish_type 无法直接相加
chosen_fish = list(map(lambda fish: int(fish), chosen_fish))
return chosen_fish
```

这段代码表示的是选中射水鱼、翻车鱼、蝠鲼和拟态鱼（模仿海龟）出战。

err_type

`err_type` 定义了 `Action` 类中一些常见的错误操作，同时也是 `Action` 中 `set_xxx` 类型函数的返回值。

错误类型	Value	`err_type`	备注
目标选取错误	0	`err_target`	当鱼行动时选取了非法目标（比如选取己方鱼作为攻击对象）
目标位置错误	1	`err_target_out_of_range`	通过位置指定鱼的时候，`pos` 不属于范围 $[0, 3]$ 内
目标行动类型错误	2	`err_action_type`	`set_action_type` 的时候参数值不为 0 或 1
选取的鱼 ID 错误	3	`err_action_fish`	正常情况下不会出现该错误
未选定行动类型	4	`err_action_not_choose`	未指定行动类型的时候调用 `set_enemy_target` 或 `set_friend_target`
未选取鱼	5	`err_fish_not_choose`	未选取鱼的时候调用 `set_enemy_target` 或 `set_friend_target`
成功	6	`success`	操作成功

passive_type

被动技能的类型，比较粗糙的归类，避免透露过多信息。

passive_type	描述
counter	膨胀反伤
deflect	承伤
reduce	减伤
heal	回血
explode	爆炸

skill_type

主动行动的类型，比较粗糙的归类，避免透露过多信息。

skill_type	描述
aoe	群体伤害
infight	伤害队友
crit	暴击
subtle	无作为
normalattack	普通攻击

预定义类

Python AI SDK 是通过 pybind11 让 Python 调用 C++ 动态链接库实现的，所以类的私有成员在 Python AI SDK 中均没有定义，关于这些类的私有成员，您可以查阅 C++ AI SDK 文档中的“预定义类”小节。如果您想要修改 AI SDK，可以查阅 pybind11 官方文档并修改 AI_SDK/Python/sdk/py_ai_sdk.cpp，而后重新编译动态链接库。

Action

`Action` 类用于定义鱼在一回合内的操作，是 `Act` 函数的返回值。

`Action` 类默认实现了一些简单的操作正确性检查。如果出现了非法操作，则会以返回值的形式告诉选手错误类型，方便选手判断和修改。错误类型是 `err_type`，点击此处了解该枚举类型的详细信息。

成员函数

set_action_type

- ```
  def set_action_type(self: Action, action_type: int) -> err_type: ...
  ```
- 设置操作类型 `actionType`（私有）。如果参数值为 0 或 1，将其赋值给 `acitionType` 并返回 `success`，否则返回 `err_action_type`.
  ```

set_action_fish

- ```
  def set_action_fish(self: Action, fish_id: int) -> err_type: ...
  ```

- 设置行动鱼 `actionFish`（私有）。如果参数值在区间 $[0,3]$ 内，将其赋值给 `actionFish` 并返回 `success`，否则返回 `err_target_out_of_range`.

set_enemy_target

- ```
  def set_enemy_target(self: Action, enemy_id: int) -> err_type: ...
  ```

- 设置敌方目标 `enemyTarget`（私有）。如果选取的行动为普通攻击，或者选取的行动为发动技能且需要指定敌方鱼为目标，则调用此函数。

  如果传入参数在区间 $[0,3]$ 内，则检查当前鱼和行动类型是否支持当前操作；若否，则根据错误类型返回对应参数。如果选取了鱼且发动主动且鱼的主动是对敌方的 `aoe`，则传入任意参数都将会把 `enemyTarget`（私有）置为 -2.

set_friend_target

- ```
  def set_friend_target(self: Action, friend_id: int) -> err_type: ...
  ```

- 同上。但是由于没有对友方鱼的群体技能，所以传入参数区间限定在 $[0,3]$.

ActionInfo

用于描述某一方在一个回合内进行的操作。

列表中的数据顺序会按照技能结算顺序给出。关于结算顺序请参考游戏规则的结算说明部分。

成员变量

action_fish

- ```
  self.action_fish: int = ...
  ```

- 行动的鱼的位置（pos）。初始值为 -1.

enemy_excepted_injury

- 出于兼容性考虑而保留的错误拼写版本，见 enemy_expected_injury.

enemy_expected_injury

- ```
  self.enemy_expected_injury: list[int] = ...
  ```

- 该行动玩家的敌方预期受到的伤害（与实际受到的伤害相对）。列表中数据顺序与 `enemy_targets` 一致。即假设：

  ```
  assert enemy_expected_injury[0] == 50
  assert enemy_targets[0] == 1
  ```

  则代表行动玩家的敌方 1 号位置的鱼预期受到 50 点伤害。
  ```

enemy_injury

- ```
  self.enemy_injury: list[int] = ...
  ```
- 该行动玩家的敌方实际受到伤害。列表中数据顺序与 `enemy_injury_id` 一致。

enemy_injury_id

- ```
  self.enemy_injury_id: list[int] = ...
  ```
- 该行动玩家的敌方实际受到伤害的鱼的位置（pos）。列表中数据顺序与 `enemy_injury` 一致。

enemy_injury_timestamp

- ```
  self.enemy_injury_timestamp: list[int] = ...
  ```
- 该次**伤害**的时间戳，即结算过程中，该次伤害结算的次序（第几个进行结算的），从 1 开始标号，这个标号从小到大记录了敌我方被动技能和敌我方受到伤害事件的结算顺序。列表中数据顺序与 `enemy_injury` 和 `enemy_injury_id` 一致。

enemy_passives_id

- ```
  self.enemy_passives_id: list[int] = ...
  ```
- 该行动玩家的敌方触发的被动的鱼的位置（pos）。列表中数据顺序与 `enemy_passive_value` 和 `enemy_types` 一致。

enemy_passive_timestamp

- ```
  self.enemy_passive_timestamp: list[int] = ...
  ```
- 该次**被动**的时间戳，即结算过程中，该次被动结算的次序（第几个进行结算的），从 1 开始标号，这个标号从小到大记录了敌我方被动技能和敌我方受到伤害事件的结算顺序。列表中数据顺序与 `enemy_passives_id`，`enemy_passive_value` 和 `enemy_types` 一致。

enemy_injury_traceable

- ```
  self.enemy_injury_traceable: list[bool] = ...
  ```
- 该次伤害是否可追踪，即该次伤害来源是否是由攻击和技能**直接**造成的伤害；如果是，则对应值为 `True`，如果不是则为 `False`. 列表中数据顺序与 `enemy_injury` 和 `enemy_injury_id` 一致。

enemy_passive_value

- ```
  self.enemy_passive_value: list[float] = ...
  ```
- 如果该行动玩家的敌方触发的被动类型为减少伤害（`reduce`），则其值为减伤比；如果该行动玩家的敌方触发的被动类型为治疗（`heal`），则其值为**实际治疗量**（考虑了血量上限），否则其值无意义。列表中数据顺序与 `enemy_passives_id` 和 `enemy_types` 一致。
  ```

enemy_targets

- ```
  self.enemy_targets: list[int] = ...
  ```
- 该行动玩家选取的敌方鱼的位置（pos）。列表中数据顺序与 `enemy_expected_injury` 一致。

enemy_types

- ```
  self.enemy_types: list[passive_type] = ...
  ```
- 该行动玩家的敌方触发的被动类型（`passive_type` 点击这里查看定义）。列表中数据顺序与 `enemy_passives_id` 和 `enemy_passive_value` 一致。

friend_excepted_injury

- 出于兼容性考虑而保留的错误拼写版本，见 friend_expected_injury.

friend_expected_injury

- ```
  self.friend_expected_injury: list[int] = ...
  ```
- 该行动玩家的友方预期受到的伤害（与实际受到的伤害相对）。列表中数据顺序与 `friend_targets` 一致。即假设：

  ```
  assert friend_expected_injury[0] == 50
  assert friend_targets[0] == 1
  ```

  则代表行动玩家的友方 1 号位置的鱼预期受到 50 点伤害。

friend_injury

- ```
  self.friend_injury: list[int] = ...
  ```
- 该行动玩家的友方实际受到伤害。列表中数据顺序与 `friend_injury_id` 一致。

friend_injury_id

- ```
  self.friend_injury_id: list[int] = ...
  ```
- 该行动玩家的友方实际受到伤害的鱼的位置（pos）。列表中数据顺序与 `friend_injury` 一致。

friend_injury_timestamp

- ```
  self.friend_injury_timestamp: list[int] = ...
  ```
- 该次**伤害**的时间戳，即结算过程中，该次伤害结算的次序（第几个进行结算的），从 1 开始标号，这个标号从小到大记录了敌我方被动技能和敌我方受到伤害事件的结算顺序。列表中数据顺序与 `friend_injury` 和 `friend_injury_id` 一致。

friend_injury_traceable

- ```
  self.friend_injury_traceable: list[bool] = ...
  ```

- 该次伤害是否可追踪，即该次伤害来源是否是由攻击和技能**直接**造成的伤害；如果是，则对应值为 `True`，如果不是则为 `False`. 列表中数据顺序与 `friend_injury` 和 `friend_injury_id` 一致。

friend_passives_id

- ```
  self.friend_passives_id: list[int] = ...
  ```

- 该行动玩家的友方触发的被动的鱼的位置（pos）。列表中数据顺序与 `friend_passive_value` 和 `friend_types` 一致。

friend_passive_timestamp

- ```
  self.friend_passive_timestamp: list[int] = ...
  ```

- 该次**被动**的时间戳，即结算过程中，该次被动结算的次序（第几个进行结算的），从 1 开始标号，这个标号从小到大记录了敌我方被动技能和敌我方受到伤害事件的结算顺序。列表中数据顺序与 `friend_passives_id`, `friend_passive_value` 和 `friend_types` 一致。

friend_passive_value

- ```
  self.friend_passive_value: list[float] = ...
  ```

- 如果该行动玩家的友方触发的被动类型为减少伤害（`reduce`），则其值为减伤比；如果该行动玩家的友方触发的被动类型为治疗（`heal`），则其值为**实际治疗量**（考虑了血量上限），否则其值无意义。列表中数据顺序与 `friend_passives_id` 和 `friend_types` 一致。

friend_targets

- ```
  self.friend_targets: list[int] = ...
  ```

- 该行动玩家选取的友方鱼的位置（pos）。列表中数据顺序与 `friend_expected_injury` 一致。

friend_types

- ```
  self.friend_types: list[passive_type] = ...
  ```

- 该行动玩家的友方触发的被动类型（`passive_type` 点击这里查看定义）。列表中数据顺序与 `friend_passives_id` 和 `friend_passive_value` 一致。

is_skill

- ```
  self.is_skill: bool = ...
  ```

- 该次行动是否使用了技能。`True` 代表使用了技能，`False` 代表使用了普通攻击。初始值为 `False`.
  ```

num_enemy_injury

- ```
  self.num_enemy_injury: int = ...
  ```
- 该行动玩家的敌方受伤鱼的数量。需要注意的是，该值为列表 `enemy_injury` 和 `enemy_injury_id` 的长度。

num_enemy_targets

- ```
  self.num_enemy_targets: int = ...
  ```
- 该行动玩家选取的敌方目标个数。需要注意的是，该值为列表 `enemy_targets` 和 `enemy_expected_injury` 的长度。初始值为 0.

num_enemy_passives

- ```
  self.num_enemy_passives: int = ...
  ```
- 该行动玩家的敌方鱼触发被动的个数。需要注意的是，该值为列表 `enemy_passives_id`, `enemy_types` 和 `enemy_passive_value` 的长度。初始值为 0.

num_friend_injury

- ```
  self.num_friend_injury: int = ...
  ```
- 该行动玩家的友方受伤鱼的数量。需要注意的是，该值为列表 `friend_injury` 和 `friend_injury_id` 的长度。初始值为 0.

num_friend_targets

- ```
  self.num_friend_targets: int = ...
  ```
- 该行动玩家选取的友方目标个数。需要注意的是，该值为列表 `friend_targets` 和 `friend_expected_injury` 的长度。初始值为 0.

num_friend_passives

- ```
  self.num_friend_passives: int = ...
  ```
- 该行动玩家的友方鱼触发被动的个数。需要注意的是，该值为列表 `friend_passives_id`, `friend_types` 和 `friend_passive_value` 的长度。

type

- ```
  self.type: skill_type = ...
  ```
- 如果行动类型为主动，则 `type` 表示该玩家发动的主动的类型。关于 `skill_type`，点击这里查看具体定义。
  ```

成员函数

is_empty

- ```
  def is_empty(self: ActionInfo) -> bool: ...
  ```

- 判定 `ActionInfo` 是否为空，即上一回合玩家是否有操作。

AssertInfo

用于描述某一方的断言操作和断言结果。

成员变量

assertContent

- ```
  self.assertContent: int = ...
  ```

- 断言鱼的 ID. 初始值为 0.
- **注意**：当对手上一回合没有进行断言的时候，该变量值为 0.

assertPos

- ```
  self.assertPos: int = ...
  ```

- 被断言鱼的位置（pos）。初始值为 0.
- **注意**：当对手上一回合没有进行断言的时候，该变量值为 0. 所以不要用该变量来判断对手上一回合是否进行了断言。

assertResult

- ```
  self.assertResult: bool = ...
  ```

- 断言结果。`True` 代表断言成功，`False` 代表断言失败。初始值为 `False`.
- **注意**：当对手上一回合没有进行断言的时候，该变量值为 `False`.

成员函数

is_empty

- ```
  def is_empty(self: AssertInfo) -> bool: ...
  ```

- 判定 `AssertInfo` 是否为空，即上一回合玩家是否有操作。

Fish

用于描述一条鱼的信息。

成员变量
  ```

id

- ```
  self.id: int = ...
  ```
- 鱼的 ID. 如果是己方鱼，则 ID 为其真实 ID；如果是敌方鱼，则如果这条鱼已经被断言成功，它的 ID 是其真实 ID，否则是 -1.

hp

- ```
  self.hp: int = ...
  ```
- 鱼的血量。

atk

- ```
  self.atk: int = ...
  ```
- 鱼的攻击力。如果是己方鱼，则该值为其实际攻击力；如果是敌方鱼，则这个值无意义。

Game

描述当前游戏局面的类。

只读属性

enemy_action

- ```
  @property
  def enemy_action(self: Game) -> ActionInfo: ...
  ```
- 描述了敌方的行动及其结果。关于 `ActionInfo`，点击这里查看其定义。

enemy_assert

- ```
  @property
  def enemy_assert(self: Game) -> AssertInfo: ...
  ```
- 描述了敌方的断言及其结果。关于 `AssertInfo`，点击这里查看其定义。

enemy_fish

- ```
  @property
  def enemy_fish(self: Game) -> list[Fish]: ...
  ```
- 描述了敌方的四条鱼的状态。关于 `Fish`，点击这里查看其定义。

my_action

- ```
  @property
  def my_action(self: Game) -> ActionInfo: ...
  ```
- 描述了我方上一回合的行动及其结果。关于 `ActionInfo`，点击这里查看其定义。
  ```

my_assert

- ```python
  @property
  def my_assert(self: Game) -> AssertInfo: ...
  ```
- 描述了我方上一回合的断言及其结果。关于 `AssertInfo`，点击这里查看其定义。

my_fish

- ```python
  @property
  def my_fish(self: Game) -> list[Fish]: ...
  ```
- 描述了我方的四条鱼的现在的状态。关于 `Fish`，点击这里查看其定义。

round

- ```python
  @property
  def round(self: Game) -> int: ...
  ```
- 当前回合数，取值范围为 $[1, 3]$.

avatar_id

- ```python
  @property
  def avatar_id(self: Game) -> int: ...
  ```
- 当前回合己方拟态鱼模仿的鱼的 ID，如果没有的话是 -1.

first_mover

- ```python
  @property
  def first_mover(self: Game) -> bool: ...
  ```
- 当前回合我方是不是先手，如果是的话是 `True`.

round1_win

- ```python
  @property
  def round1_win(self: Game) -> bool: ...
  ```
- 描述第一局结果。如果第一局我方获胜，则该值为 `True`，否则为 `False`.

round2_win

- ```python
  @property
  def round2_win(self: Game) -> bool: ...
  ```
- 描述第二局结果，同上。

round3_win

- ```python
  @property
  def round3_win(self: Game) -> bool: ...
  ```
- 描述第三局结果，同上。~~虽然完全没有用，但为了满足强迫症还是加上了。~~

last_round_finish_reason

- ```
  @property
  def last_round_finish_reason(self: Game) -> int: ...
  ```

- 描述上一轮结束原因和时刻。取值说明如下：

  0. 第一轮，没有上一轮的值。
  1. 我方断言后上一轮结束。
  2. 我方行动回合结束后上一轮结束。
  3. 敌方断言后上一轮结束。
  4. 敌方行动回合结束后上一轮结束。
  ```