Auction


10 grants!
15!
i'll give 20!


I haven't revelaed the fucking thing yet

Basic

- There are two files: **basic-train.txt, basic-test.txt**
- There are **2000** paintings in train data, **1000** paintings in test data
- There are **500** buyers, and we need to choose top-10 buyers
- In the two data, there are **500** lines, which means **500** buyers; each line has **2000 / 1000** values, which means the number of paintings.
- evaluation: rate of choosing the top-k buyers

Advanced

- There are four files:
  advanced-train.txt, advanced-test.txt, advanced-train-category.txt, advanced-test-category.txt
- There are **2000** paintings in train data, **1000** paintings in test data
- There are **500** buyers, and we need to choose top-10 buyers
- In **advanced-train.txt** and **advanced-test.txt**, there are **500** lines, which means **500** buyers; each line has **2000 / 1000** values, which means the number of paintings.
- In **advanced-train-category.txt** and **advanced-test-category.txt**, there are **2000 / 1000** lines and with 1 value per line, which means the category of paintings. (001 / 002 / 003)
- evaluation: rate of choosing the top-k buyers

Perfect or Nothing

whole source code is on https://github.com/Kikokushijo/Perfect-Or-Nothing-Environment

Basic

- you'll need to implement the **decide()** functions in class **Basic_2_Agent()** and **Basic_3_Agent()**
- evaluation: average rate of picking up one of the top-k value (basic-2) / best value (basic-3)

```
def decide(self, value):

    '''
    parameters:
    ----------------------------------------------------------------------
        value: the new value you get; you will only get a value when the decide
        function is called once.
    return:
    ----------------------------------------------------------------------
        a boolean; True if you want to pick the value or False otherwise.
    '''

    # The same decide function as sample code in the basic-1 problem.
    return super(Basic_2_Agent, self).decide(value)
```

Advanced

- you'll need to implement **decide()** functions in class **Advanced_1u_Agent()**, **Advanced_1n_Agent()**, and **Advanced_2_Agent()**
- evaluation: average rate of picking up the best value

```
def decide(self, value):

    '''
    parameters:
    ----------------------------------------------------------------------
        value: the new value you get; you will only get a value when the decide
        function is called once.
    return:
    ----------------------------------------------------------------------
        a boolean; True if you want to pick the value or False otherwise.
    '''

    # The same decide function as sample code in the basic-1 problem.
    return super(Advanced_1u_Agent, self).decide(value)
```

Deadline

- 23:59 on 6/19
- via ceiba
- language: python
- requirements:
  - the decide() functions
  - libraries needed
  - readme (just in case I cannot run your code)

How to shoot

Basic

- you'll only need to implement the following function
- evaluation: average winning rate of M-th shooter

```
def shoot_next_1(acc, alive, you):
    '''
    parameters:
    -----------------------------------------------------------------------
        acc: shooting accuracy of each player, [0(ignore), p_1, p_2,..., p_N]
        alive: whether the player is still alive, [0(ignore), a_1, a_2,..., a_N]
        you: your player index
        note that it's always your turn when this function is called
    return:
    -----------------------------------------------------------------------
        the index of player to shoot; return 0 if not to shoot
    '''
    for i in range(1, len(alive)):
        if alive[i] and i != you:
            return i
```

Advanced

- you'll also only need to implement the following function

```
def shoot_next_2(acc, alive, turn):
    '''
    parameters:
    ----------------------------------------------------------------------
        acc: shooting accuracy of each player, [0(ignore), p_1, p_2,..., p_N]
        alive: whether the player is still alive, [0(ignore), a_1, a_2,..., a_N]
        turn: who's turn to shoot
    return:
    ----------------------------------------------------------------------
        the index of player to shoot; return 0 if not to shoot
    '''
    for i in range(1, len(alive)):
        if alive[i] and i != turn:
            return i
```

Deadline

- 23:59 on 6/19
- via ceiba
- language: python
- requirements:
    - the two functions
    - libraries needed
    - readme (just in case I cannot run your code)

DEMO