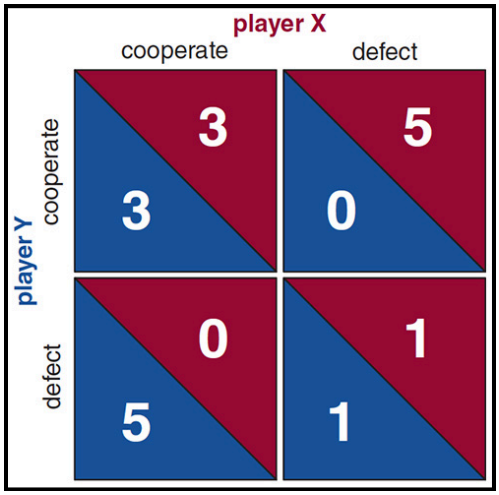
Tournament requirements

The tournament is based on the modified version of the Prisoner's Dilemma, Iterated Prisoners Dilemma. The Iterated Prisoner's Dilemma is an extension of the general form, except the game is repeatedly played by the same participants. The main principle is that the participants, in our case, your algorithms, should get as many points as possible.



You should implement an algorithm that chooses one step at a time either to cooperate or to defect while playing together with another algorithm. So in a way, it's a PvP (Player versus Player) with algorithms created by your other colleagues. You will play in rounds, and every round the program must make a choice: to deflect (0) or to cooperate (1). These choices will determine how many points you will receive. As shown in the image, the points will be given in the following way:

- cooperate together: receive both 3 points
- deflect together: receive both 1 point
- one deflects, one cooperates: who deflects gets 5 points, the one who tried to cooperate will get nothing

How the algorithm will work is up to you: is it friendly and tries to cooperate, or is it retaliatory and gets mad really quickly if the opponent defects first. It may be a simple algorithm that alternates between the 0 and 1, or it can be a complex one that tries predicting how the opponent will react next. It might be nice to others and work toward cooperation, or it can be a nasty one and deflect each time. Again, it's up to you how the algorithm will come out.

Based on these algorithms will be made a tournament, in order to find out the best performing algorithm(s). The requirements for this algorithm will be analyzed later in the current document.

The tournament is taking place in 2 parts: first part has already taken place (bravo, smartie!); second one will take after Easter vacation.

The deadline to submit your algorithm for the **second part** of the tournament is **27th of April, 23:59**. The submission link is the following: https://forms.gle/t7xLX16yosridYzu9

Those who participated in the first round: you do not need to complete the submission form. Just add the algorithm in your GitHub. Remember, same GitHub that you submitted initially. Same GitHub, 2 files, named differently! We will consider the last time of the commit as your submission time. If you upload later – there will be consequences (the mark will be much lower than you might expect).

Those who joined for the second round: Complete the submission form. The last date of the commit will be considered as the submission day.

You will work alone, and will submit your GitHub with a README file that will explain a bit of the algorithm logic. In the end, your GitHub will contain 2 different files (or just the last one) with your algorithm, and 1 README. Our Algorithm Experts will analyze your algorithm and offer you feedback if needed. That's why you will have to complete the form to get in the tournament, if not done so already.

Requirements for the first part of the tournament

Congratulations! You passed the first part. Now you should focus on the requirements of the second part🎉🎉🎉

Requirements for the second part of the tournament

The second part will happen similar to what happened in 1 stage, but with modified rules.
The algorithm stays the same, but now your algorithm will have the possibility to choose the algorithms it will work with.

What does this mean? *Let me explain using an example:* Let's imagine you have 500 lives, named also rounds, and 50 algorithms. In the previous part, you could play a fixed number of rounds with each algorithm, but just once. Now, you have the possibility to play a fixed number of rounds, with any algorithm you want! The only constraint is the maximum number of rounds you can play with an algorithm, let's consider 100 rounds max. That means you can play with 5 players, 100 rounds each. Or you could play with all 50 of them, 10 rounds each. Or you can select opponents with prime number ID and play with them. You can select the most collaborative algorithms and work to the maximum with them. Or you could find some friendly algorithm and ~~gaslight them~~ defect as much as possible. Or you might want to detect dynamically if the algorithm is mean and choose another players and never play with traitors again. Your algorithm decides who to play with. You might decide that a social status for each player might be more fitting for your play style. That will depend on you. The only requirement:

do not get over the max limit, use the same algorithm as in the last part, and get the most points out of this.

Submission file format

Each participant submits a single **file with *.py* extension**. It should contain **only their algorithm function, called *strategy_round_2***. No additional scripts, libraries, or external dependencies. No exception even for random module. The file should be also contained in the GitHub (the one you registered with in the form).
The file should follow the naming format :

lastname_firstname_algorithm_name_round_2.py

First goes your last name and first name, then algorithm name, all with only with small letters and underscores.

Examples: *evangelista_jessica_tit_for_tat_round_2.py*;
istrati_stefan_strategy_name_round_2.py, etc.

P.S. Please, try to write the file name with English letters. If you want to write the name in any other language, transliterate it.

Participants should ensure that their code does not include:

- unnecessary comments
- debugging prints
- other things that might create difficulties during run time.

Before submission, they should thoroughly test their algorithm to ensure that it behaves as expected and does not crash under any conditions. We will provide at the end of the document a link to GitHub that contains a program that tests your algorithms.

Function signature

In the file there should be only the function responsible for their algorithm. The function also must stick to a predifined format, which will ensure that it is compatible to play with other algorithms. Same thing as in previous part.

The function should be named ***strategy_round_2.*** It will take 3 arguments: *opponent_id, my_history*, *opponents_history.*

Thus, the function must be defined as:

def strategy_round_2(opponent_id: int, my_history: dict[int, list[int]], opponents_history: dict[int, list[int]]) -> tuple[int, int]:

This style will ensure fair evaluation for each algorithm.
Do not name your algorithm function any other than *strategy_round_2*.

Parameter *opponent_id*:

An integer representing the ID of the opponent the player is currently facing.

Parameter *opponents_history*:

A dictionary where keys are all player IDs (int), and values are lists of 0s and 1s representing the opponents' past moves against the player.

Parameter *my_history*:

A dictionary where keys are all player IDs (int), and values are lists of 0s and 1s representing the player's past interactions with those specific opponents.

The return value:

The function must return a tuple (*move*, *next_opponent*), where:

- *move* (int): 0 for defect or 1 for cooperate in the current round.
- *next_opponent* (int): The ID of the player the function wants to challenge next.

Example of return value:

```
(1, 2)
```

This means 1 (for cooperating) and choose player 2 as the next opponent. Next, you can pick a player you have not played yet or the same opponent.

Example of inputs:

```
my_history = { 1: [1, 0, 1, 1],
               2: [0, 0, 1],
               3: [1, 1, 0, 1, 0]
```

```
                    4: [] }
```

These represent how you played against other players (strategies). Played 4 rounds against player 1, played 3 rounds against player 2, played 5 rounds against player 3, etc.

```
opponents_history = { 1: [0, 1, 1, 1],
                      2: [1, 0, 0],
                      3: [1, 0, 1, 0, 1],
                      4: [] }
```

These represent how other algorithms played against you: Player 1's actions against you, player 2's actions against you, player 3's actions against you, etc.

```
opponent_id = 3
```

This represents who you play right now, with this value you get from the dictionary of your and opponent's history.

Game rules and code constraits

Rules:

1. The function must not exceed **200 rounds total** with any individual player (i.e., the length of any list in *my_history* or *opponents_history* will not exceed 200). Before selecting the next opponent, the algorithm must exclude from potential opponents any algorithm with which it has already played the maximum allowed rounds (200). In other words, do not allow the algorithm play more than it should with any algorithm.
2. The function should **only use the provided arguments** (*opponent_id*, *my_history*, *opponents_history*).
3. No external data sources(additional libraries), file access, or network usage.
4. Do not try to cheat using threading or other ingineous moves to get more max rounds to play with one algorithm. This will be considered as a serious break of the rules.

Code constraints:

1. Execution time per call should not exceed *200ms* to prevent infinite loops or overly complex computations.
2. Memory usage should stay within *50MB*, ensuring that no strategy monopolizes system resources.

! *Please, consider the rules and constraints we provided to you, in order to have fair competition between the algorithms! Otherwise, we might not allow the participation to the tournament.*

Please take serious the deadlines. People that will fail the deadline not only might have lower mark for this tournament, but also might become chocolate and sweets investors for our second round fund.
Next page you will find some useful information during your development that might ease a bit the process.

Useful resources and links

We know that you might be confused how you should test such algorithms, thus we created a short program that might help you understand and test your program: GitHub - IsStephy/AA_competition_test
You can take this code, insert your algorithms (be aware where you put the functions!) and test how everything works. Hope it will help you. Random function is used for opponent example algorithm, so do not use it for yourself or delete it. Do not submit your code together with the test code! In your file should be contained only your strategy function!

Since some of you might be not understanding what is happening here, or you want to refresh your mind, we decided to provide some useful links that might help you during the development.

- https://www.youtube.com/watch?v=mScpHTIi-kM&t=27s
- https://www.youtube.com/watch?v=BOvAbjfJ0x0
- https://faculty.sites.iastate.edu/tesfatsi/archive/econ308/tesfatsion/axeltmts.pdf

---

In case if you need some help understanding the requirements provided, or you have some useful feedback that will make the tournament even better, you can contact the following people and they will help you:

- Mihalevschi Alexandra, FAF-232, alexandra.mihalevschi@isa.utm.md
- Untu Mihaela, FAF-232, mihaela.untu@isa.utm.md