unibz _ Master in Computational Data Science (LM-18)
Decision Making and Support Systems
Practice: strategic games

GENERAL MOTIVATION

Implement the tools to assist a decision maker to make a decision in a game-theoretical situation, in the presence of other self-interested players.

We concentrate on **2-players strategic matrix games**. There is one row player, and one column player. We concentrate on **pure strategies** (no randomization).

DEFINITIONS

An action a_1 **strictly dominates** an action a_2 if $a_1$yields at a *greater* utility as a_2 no matter what the other player does.

A **strictly dominated action** is an action for which there exists another action that strictly dominates it.

An action a_1 **strongly dominates** an action a_2 if $a_1$yields at at least the same utility as a_2 no matter what the other player does, and at least for some action of the other player it yields a strictly greater utility.

A **(strongly) dominant action** is an action that strongly dominates all other actions.

The **iterated elimination of strictly dominated strategies** consists in iteratively deleting *strictly* dominated actions of any player, and in any order, until no strictly dominated action remains.

A **profile** is a pair of action ids, one for the row player, one for the column player.

A profile is a **Nash equilibrium** if no player has an incentive to unilaterally change his action.

OBJECTIVE

Strategic game matrices will be represented simply as an array of arrays of pairs of numerical values.

Write the code necessary to get the expected results.

ITERATED ELIMINATION OF STRICTLY DOMINATED STRATEGIES

```
def main():
    m0 = StrategicGame([[(10, 10), (0, 12)], [(12, 0), (1, 1)]])
    m0.assign_row_actions_names(["high", "low"])
    m0.assign_col_actions_names(["high", "low"])
    m1 = StrategicGame([[(0, 0), (1, 1), (2, 3)],
                        [(1, -1), (-1, 0), (-1, -2)],
                        [(0, 1), (1, 1), (5, 2)]])
    m2 = StrategicGame([[(3, 0), (4, 1), (2, 3)],
                        [(1, -1), (3, 0), (-1, -2)],
                        [(2, 1), (2, 1), (1, 2)]])
    for m in [m0, m1, m2]:
        print("\n========================================\n")
        print(f"Before IESDS\n{m}")
        print(f"After IESDS\n{m.iesds(verbose=True)}")
```

Should yield something like:

```
========================================

Before IESDS
+------+---------+---------+
|      |   high  |   low   |
+------+---------+---------+
| high | (10, 10) | (0, 12) |
| low  | (12, 0)  |  (1, 1) |
+------+---------+---------+
Removing row high (strictly dominated by low).
Removing column high (strictly dominated by low).
After IESDS
```

```
+-----+--------+
|     |  low   |
+-----+--------+
| low | (1, 1) |
+-----+--------+
```

==========

Before IESDS

```
+---+---------+---------+----------+
|   |    0    |    1    |    2     |
+---+---------+---------+----------+
| 0 |  (0, 0) |  (1, 1) |  (2, 3)  |
| 1 | (1, -1) | (-1, 0) | (-1, -2) |
| 2 |  (0, 1) |  (1, 1) |  (5, 2)  |
+---+---------+---------+----------+
```

After IESDS

```
+---+---------+---------+----------+
|   |    0    |    1    |    2     |
+---+---------+---------+----------+
| 0 |  (0, 0) |  (1, 1) |  (2, 3)  |
| 1 | (1, -1) | (-1, 0) | (-1, -2) |
| 2 |  (0, 1) |  (1, 1) |  (5, 2)  |
+---+---------+---------+----------+
```

==========

Before IESDS

```
+---+---------+--------+----------+
|   |    0    |   1    |    2     |
+---+---------+--------+----------+
| 0 |  (3, 0) | (4, 1) |  (2, 3)  |
| 1 | (1, -1) | (3, 0) | (-1, -2) |
| 2 |  (2, 1) | (2, 1) |  (1, 2)  |
+---+---------+--------+----------+
Removing row 1 (strictly dominated by 0).
Removing row 2 (strictly dominated by 0).
Removing column 0 (strictly dominated by 1).
Removing column 1 (strictly dominated by 2).
```

After IESDS

```
+---+--------+
|   |   2    |
+---+--------+
```

```
| 0 | (2, 3) |
+---+--------+
```

Make sure that iesds does return the matrix that is the result of the iterated elimination of strictly dominated strategies. Make sure that it does so non-destructively.

Continuing the script above:

```
    for m in [m0, m1, m2]:
        print(f"My game is still unchanged \n{m}")
```

Should print something like:

```
My game is still unchanged
+-------+----------+----------+
|       |   high   |  low $   |
+-------+----------+----------+
|  high | (10, 10) | (0, 12)  |
| low $ | (12, 0)  | (1, 1) * |
+-------+----------+----------+
My game is still unchanged
+---+---------+---------+----------+
|   |    0    |    1    |    2     |
+---+---------+---------+----------+
| 0 |  (0, 0) |  (1, 1) |  (2, 3)  |
| 1 | (1, -1) | (-1, 0) | (-1, -2) |
| 2 |  (0, 1) |  (1, 1) | (5, 2) * |
+---+---------+---------+----------+
My game is still unchanged
+-----+---------+--------+----------+
|     |    0    |   1    |    2     |
+-----+---------+--------+----------+
| 0 $ |  (3, 0) | (4, 1) | (2, 3) * |
|  1  | (1, -1) | (3, 0) | (-1, -2) |
|  2  |  (2, 1) | (2, 1) |  (1, 2)  |
+-----+---------+--------+----------+
```

NASH EQUILIBRIA

```
    m3 = StrategicGame([[(0, 0), (-1, 1), (1, -1)],
                        [(1, -1), (0, 0), (-1, 1)],
```

```
                    [(-1, 1), (1, -1), (0, 0)]])
    m3.assign_row_actions_names(["rock", "paper", "scissors"])
    m3.assign_col_actions_names(["rock", "paper", "scissors"])
    m4 = StrategicGame([[(1, 4), (2, 4)],
                        [(1, 10), (4, 0)],
                        [(0, 10), (8, 11)]])
    for m in [m0, m1, m2, m3, m4]:
        print("Nash equilibria:", m.find_Nash_profiles())
```

Should yields something like:

```
Nash equilibria: [(1, 1)]
Nash equilibria: [(2, 2)]
Nash equilibria: [(0, 2)]
Nash equilibria: []
Nash equilibria: [(0, 0), (1, 0), (2, 1)]
```