unibz _ Master in Computational Data Science (LM-18)
Decision Making and Support Systems
Assignment 3: strategic games

Deadline: Monday 10th, January 2022 (23:59, Rome)

You can discuss your work together. You must submit on OLE an individual assignment that reflects your personal and original work. No submission will be possible after the system is closed.

Complete the following tasks:

1. Complete the practice exercise on strategic games. The main tasks are the implementation of the class `StrategicGame`, containing among others, the method `iesds` and the function `find_Nash_profiles`.
2. A profile is a *dominant profile* if every player's action is a (strongly) dominant strategy. Implement the `StrategicGame` class function `find_dominant_strategy_profiles` that returns the list of dominant profiles, as a list of row action / column action ID pairs.
```
def find_dominant_strategy_profiles(self):
    ...
```
3. Add the `StrategicGame` class method `ienbr` that non-destructively implements the iterated elimination of never best response strategies.
```
def ienbr(self, verbose=False)
    …
```
4. Improve the visualization of 2-player matrix games to present more information to the user, showing for instance, the pure-strategy dominant strategies and the pure-strategy Nash equilibria.
5. Implement a `LocationGame` extension of `StrategicGame`.
    - For every number *n*, `LocationGame(n)` builds a two-player discrete location game along "Main Street" with *n* locations.

```
lg = LocationGame(5)
print(lg)
lg = LocationGame(3)
print(lg)
```

would give something like that (exact utilities are not important, but they must be permissible variants on an ordinal scale):

```
+---+------------+------------+-------------+------------+------------+
|   |     0      |     1      |      2      |     3      |     4      |
+---+------------+------------+-------------+------------+------------+
| 0 | (2.5, 2.5) |   (1, 4)   |  (1.5, 3.5)  |   (2, 3)   | (2.5, 2.5) |
| 1 |   (4, 1)   | (2.5, 2.5) |    (2, 3)    | (2.5, 2.5) |   (3, 2)   |
| 2 | (3.5, 1.5) |   (3, 2)   | (2.5, 2.5) * |   (3, 2)   | (3.5, 1.5) |
| 3 |   (3, 2)   | (2.5, 2.5) |    (2, 3)    | (2.5, 2.5) |   (4, 1)   |
| 4 | (2.5, 2.5) |   (2, 3)   |  (1.5, 3.5)  |   (1, 4)   | (2.5, 2.5) |
+---+------------+------------+-------------+------------+------------+
+-----+------------+--------------+------------+
|     |     0      |     1 $      |     2      |
+-----+------------+--------------+------------+
|  0  | (1.5, 1.5) |    (1, 2)    | (1.5, 1.5) |
| 1 $ |   (2, 1)   | (1.5, 1.5) * |   (2, 1)   |
|  2  | (1.5, 1.5) |    (1, 2)    | (1.5, 1.5) |
+-----+------------+--------------+------------+
```

- Use your implementation to run the iterated elimination of strictly dominated strategies and never best response strategies and to determine what are the Nash equilibria in location games with an arbitrary number of locations:

```
lg = LocationGame(28)
print(f"IESDS with 28 locations\n{lg.iesds(verbose=True)}")
lg = LocationGame(187)
print(f"With 187 locations, NE: {lg.find_Nash_profiles()}")
```

would give something like that:

```
Removing row 0 (strictly dominated by 1).
Removing row 27 (strictly dominated by 9).
[...]
Removing column 16 (strictly dominated by 13).
Removing row 15 (strictly dominated by 13).
IESDS with 28 locations
+----+----------------+----------------+
|    |       13       |       14       |
+----+----------------+----------------+
| 13 | (14.0, 14.0) * |   (14, 14) *   |
| 14 |   (14, 14) *   | (14.0, 14.0) * |
+----+----------------+----------------+
With 187 locations, NE: [(93, 93)]
```

6. Implement a `CournotDuopolyGame` extension of `StrategicGame` where `CournotDuopolyGame(prod_poss1, prod_poss2, c1, c2, price_fun)` is a Cournot duopoly where:
    - `c1` and `c2` are unit costs for each firm, e.g., `c1` = `c2` = 30.
    - `prod_fun` is the product demand price function with two parameters that stand for the quantities produced by each firm. E.g, `lambda x, y: 150 - x - y`.

- The profit of one firm producing *q* and with unit cost *c* is *q(P - c)*.
- `prod_poss1` and `prod_poss2` are possible production amounts, e.g, `range(0, 100)`, or `[i / 25 for i in range(38 * 25, 42 * 25)]`.
- Use your implementation to run the iterated elimination of strictly dominated strategies and never best response strategies and to determine what are the Nash equilibria in various Cournot duopoly games.
- An example with (very) asymmetric firms in terms of production capacity and costs, and with a non-linear price function:

```
cdg = CournotDuopolyGame(range(0, 60), range(0, 100), 10, 87,
                         lambda x, y: 100 - (x + y) ** (1/2))
print("Nash equilibria and unit price:",
      [((q1, q2), cdg.price_fun(q1, q2))
       for (q1, q2) in cdg.find_Nash_profiles()])
print(f"After IESDS\n{cdg.iesds()}")
```

would give something like:

```
Nash equilibria and unit price: [((59, 52), 89.46434624714726)]
After IESDS
+------+-------------------------------------------+
|      |                  52 #                     |
+------+-------------------------------------------+
| 59 # | (4688.396428581688, 128.1460048516576) * |
+------+-------------------------------------------+
```

7. Write a 3-page report.

Criteria and relative weights: general understanding (2), completeness and correctness of solution (5), quality of writing (2), quality of code (1).

Bonus: in a section of the report (which will not count towards the 3-page limit), clearly write a bid from 0 to 10 (any real number amount). If the lowest submitted bid is b, and k bidders have submitted it, then each of these k lowest bidders will receive a bonus of b/k on their grade for this assignment. You can explain your thought process.