TÉCNICO LISBOA

AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS

2ND SEMESTER - 2017/2018

PROGRAMMING EXERCISES (v2)

1. GAME THEORY

A competitive game between two agents can be represented as a bi-matrix where the payoffs of each agent are presented. The best strategy depends on the strategy of the other so it is not easy to find how to behave. We will develop a code to compute the following:

1.1 NASH EQUILIBRIUM FOR PURE-STRATEGIES (2.0)

Develop a function that receives a matrix game and returns the Nash equilibria, if it exits. The returned variables are: **ne**, a list of the actions; and **nev**, a list of the corresponding values.

def findNE(M):

(…)

return ne,nev

```
ne,nev = findNE(bM)
print(bM)
[[[ -1.  -1.]
  [-10.   0.]]

 [[  0. -10.]
  [ -5.  -5.]]]
print(ne)
[[1, 1]]
print(nev)
[[-5., -5.]]
```

1.2 MIXED STRATEGIES (1.0)

In some cases there is no Nash equilibria for pure strategies but a better result can be obtained if the players do a mixed strategy. Develop a new function that finds the mixed strategy equilibrium that improves the average gain of the players. Assume the bi-matrix has two actions per player.

def findmixedNE(M):

(…)

return ne,nev

In this case the **ne** is a matrix with the probabilities, for each agent, of selecting each action.

2. NEGOTIATION

If agents can negotiate with each other then they can get a better result for each one individually than if no negotiation is possible.

2.1 NEGOTIATION IN BI-MATRIX GAMES (1.0)

Check again the bi-matrix games of question 1 and create a function that finds the best strategies that the agents can obtain. The negotiation that agents can make is to promise to pay part of their pay to the other agent so that the other picks a better option. For instance, in the following game

$$\begin{bmatrix} (2,2) & (2,6) \\ (3,1) & (3,2) \end{bmatrix}$$

a nash equilibria is (3,2). If the agent 2 ask the agent 1 to play action 1 then the result will be (2,6) but for this agent 2 has to offer to pay at least 1, resulting in gains of (3,5) that is equal for agent 1 and better for agent 2. For the function to develop consider that the extra gains are divided equally. That is the agent 2 would pay 2 to agent one and the results would be (4,4) again better for both.

```
def payPromiseE(M):
        (…)
    return ne,nev,sp
```

In this case **ne** is the same list of actions, **nev** is the gain of each one considering the side payment, and **sp** is the side payment done.

2.2 ZEUTHERN STRATEGY (2.0)

Consider that each agent attributes an individual value for each item in a set. Each agent will surely have different values for each item. Develop a function that computes the zeuthern strategy, showing each step of the negotiation. Assume that there are always at least 2 objects. At least one object must be offered.

The following function computes de zeuthern strategy. And must return the sequence of offers and which agent made that offer.

```
def zeuthern (ItemsUtility1, ItemsUtility2)
        (…)
```

The following function, taking into account the last offers, computes the new offer. It must return which agents makes the offer and the new offer.

```
def zeuthernstep (ItemsUtility1, ItemsUtility2, lastoffer1, lastoffer2)
        (…)
```

Example:

Set of items [a,b,c,d,e]

Utility of each item for player 1 U1 = [1, 2, 2, 3, 4]

Utility of each item for player 2 U2 = [4, 3, 2, 2, 2]

The first offer of player 1 is

of = [[b,c,d,e],[a]] meaning that player 1 keeps b,c,d,e and gives a to player 2. Value for player 1 = 2+2+3+4; value for player 2 = 4

The first offer of player 2 is

of = [[e],[a,b,c,d]] meaning that player 2 keeps a,b,c,d and gives e to player 1. Value for player 1 = 4; value for player 2 = 4+3+2+2

After the agents will follow the zeuthern strategy to finally arrive at a decision.