M1 CRYPTIS
Game Theory Report

presented by
NGUYEN Thi Mai Phuong

Instructor: Mr. Oussama Habachi

April 25, 2021

Contents

1 Introduction

The game theory project consists of three parts, they are:

Part 1: Implement a game with two players, each player performs two actions.

Part 2: Implement a game with two players, each player performs many actions.

Part 3: Implement a game with many players, each player performs many actions.

For every part, we must execute the same following functions:

- Determine if the game is zero-sum or not.
- Determine pure Nash Equilibria.
- Determine the dominated strategies and weakly dominant strategies.

Look at general, if we can solve the game with `n` players and many actions for each of them in **Part 3**, the game with `n = 2` like **Part 1** and **Part 2** will also be solved. Therefore, after a couple of weeks used to implement **Part 1**, we decided to implement **Part 3** and apply it to solve others.

Additionally, with the game of two players, two actions for each player, we must execute two more functions:

- Finding mixed Nash Equilibria
- Check the optimization of the mixed strategy by simulating the game.

2 Implementation

2.1 Data Structure

Besides some basic parameters like the number of players or the number of actions for each player, the most important is the structure of the player's payoffs. To find out the optimal structure for it, we examine a normal-form game with 3 players which is shown in Table 1.

		Player 2					
		Action 1		Action 2		Action 3	
		Player 3		**Player 3**		**Player 3**	
		Action 1	Action 2	Action 1	Action 2	Action 1	Action 2
Player 1	Action 1	4,5,6	2,3,0	2,2,4	1,3,4	0,0,2	1,2,3
	Action 2	4,2,3	4,5,-6	5,-2,3	2,3,1	1,2,6	6,2,0

Table 1: Example 1

After a few trials, we decide to read the payoffs from left to right ($\rightarrow$) and line by line ($\downarrow$). According to this, we have payoffs list of example 1:

```
payoffs = [(4, 5, 6),
           (2, 3, 0),
           (2, 2, 4),
           (1, 3, 4),
           (0, 0, 2),
           (1, 2, 3),
           (4, 2, 3),
           (4, 5, -6),
           (5, -2, 3),
           (2, 3, 1),
           (1, 2, 6),
           (6, 2, 0)]
```

Listing 1: Payoffs list of Example 1

In the implementation process, we have some difficulties in getting the player's payoffs according to a specific action. Therefore, we defined a new parameter called `jump` which is the distance used to move from the current action to the next action in the playoffs list of a player.
For example, in the list `payoffs` of example 1, the first 6 items are the combination of action 1 of Player 1 with other players' actions while the last 6 items are the combination of his action 2 with others. Hence, the `jump` of Player 1 in example 1 is `6`. Do the same with Player 2 and Player 3 we have a list: `jumps = [6, 2, 1]`.

2.2 Zero-sum checking

A game is said to be zero-sum if the sum of the payoffs to all players is zero with any strategy combination. With the data structure we build up in subsection 2.1, it is pretty simple to execute zero-sum checking.

```
1 def check_zero_sum(self):
2         for payoff in self.payoffs:
3             if sum(payoff) != 0:
4                 return False
5         return True
```

Listing 2: Function check zero-sum

2.3 Pure Nash Equilibrium

A pure Nash equilibrium recommends a strategy for each player such that no player can improve his benefit by unilaterally changing his strategy. However, in a game which players simultaneously choose their strategies, does not always exist pure Nash equilibrium.

In the game with many players and many actions, the steps used to find Pure Nash Equilibrium as the following:

- **Step 1:** We find the list of Nash Equilibrium candidates for each player.
  A Nash Equilibrium candidate is the best response of the player to other players' strategies. For easier control, we save the index of Nash Equilibrium candidate in `payoffs` to the list of Nash Equilibrium candidates instead of detailed payoff.
- **Step 2:** Get Pure Nash Equilibrium by intersecting all the above lists.

We use these steps to find pure Nash Equilibrium in the example of Table 1. The lists of Nash Equilibrium candidates of each player:

- Player 1: `[0, 6, 7, 8, 9, 10, 11]`
- Player 2: `[0, 1, 3, 6, 10, 7]`
- Player 3: `[0, 2, 3, 5, 6, 8, 10]`

Next, we intersecting three lists and get the final one: `[0, 6, 10]`. Finally, we get payoff value at this indexes: `[(4, 5, 6),(1, 2, 6),(4, 2, 3)]`.

```
    def pure_Nash_equilibria(self):
        nash_candiates = []
        for i in range(self.n_players):
            flag = [0 for i in range(len(self.payoffs))]
            player_best_select = []
            for j in range(len(self.payoffs)):
                if flag[j] == 0:
                    idx = j
                    tmp_max = self.payoffs[idx][i]
                    tmp_best_idx = []
                    for tmp_action in range(self.n_actions_of_player[i]):
                        gain = self.payoffs[idx][i]
                        if gain == tmp_max:
                            tmp_best_idx.append(idx)
                        elif gain > tmp_max:
                            tmp_best_idx.clear()
                            tmp_best_idx.append(idx)
                            tmp_max = gain
                        flag[idx] = 1
                        idx = idx + self.jumps[i]
                    player_best_select = player_best_select + tmp_best_idx
            if len(nash_candiates) == 0:
                nash_candiates = player_best_select
            else:
                nash_candiates =
                ↪ list(set(nash_candiates).intersection(player_best_select))
        return nash_candiates
```

Listing 3: Function find pure Nash Equilibrium

2.4 Mixed Nash Equilibrium

The mixed Nash Equilibrium involves at least one player randomly select one of his pure strategies with definite probabilities and same as pure Nash Equilibrium that no player can gain more benefit by unilateral changes.

In the scope of this project, we only apply function find mixed Nash Equilibrium for the game of two players and two actions for each player. Therefore, we investigate an example in Table 2 which is the game of two players and two actions.

		Player 2		
		Action 1	Action 2	
Player 1	Action 1	3,2	1,1	p
	Action 2	0,0	2,3	(1-p)
		q	(1-q)	

Table 2: Example 2

To compute the mixed Nash Equilibrium, assign Player 1 the probability `p` of playing Action 1 and `(1-p)` of playing Action 2. Then, assign Player 2 the probability `q` of playing Action 1 and `(1-q)` of playing Action 2, Now we have:

- Payoff when Player 1 playing Action 1: `3q + (1-q) = 2q + 1`
- Payoff when Player 1 playing Action 2: `0q + 2(1-q) = 2 - 2q`

The only case a player supposed randomize between his strategies is when all strategies give him the same payoff. According to it we have: `2q + 1 = 2 - 2q` or `q = 1/4`. Do exactly the same thing with Player 2, we have `p = 3/4`.
Therefore, the mixed Nash Equilibrium of example in Table 2 is:

- Player 1: Action 1 with probability 3/4 and Action 2 with probability 1/4.
- Player 2: Action 2 with probability 1/4 and Action 2 with probability 3/4.

According to the above example, we have the generalized form of the game of two players and two actions like Table 3.

		Player 2		
		Action 1	Action 2	
Player 1	Action 1	g_{00}, g_{01}	g_{10}, g_{11}	p
	Action 2	g_{20}, g_{21}	g_{30}, g_{31}	(1-p)
		q	(1-q)	

Table 3: Generalized form of the game of 2 players - 2 actions

With the same steps to find mixed Nash Equilibrium for Example 2, we have equations to calculate `p` and `q`:

$$\texttt{p} = \frac{g_{31} - g_{21}}{g_{01} - g_{11} - g_{21} + g_{31}} \tag{1}$$

$$\texttt{q} = \frac{g_{30} - g_{10}}{g_{00} - g_{10} - g_{20} + g_{30}} \tag{2}$$

2.5 Dominated and Dominant Strategies

2.5.1 Dominated strategies

The dominated strategy is the one that always gives the player lower payoffs than at least one of the other strategies.

To find dominated strategies in the game with many players and many actions, we sequentially examine each player follow the below steps:

- **Step 1:** Get the lists of payoffs of this player corresponding to each action.
- **Step 2:** For each pair of strategies, we compare their payoffs list.
  Note: If there is at least one item in list payoffs of strategy A is greater than strategy B while all other items is the same, then strategy B is **weakly dominated** by strategy A.

We apply these steps to find out dominated strategies of Player 2 in Table 1. Firstly, we get the lists of payoff corresponding to actions.

- Action 1: $list_{a1}$ = `[5, 3, 2, 5]`
- Action 2: $list_{a2}$ = `[2, 3, -2, 3]`
- Action 3: $list_{a3}$ = `[0, 2, 2, 2]`

Then, we compare each pair of strategies.

- Action 1 - Action 2: $list_{a1}[i] \geq list_{a2}[i]$ with any values `i` $\in$ `[0,3]`.
- Action 1 - Action 3: $list_{a1}[i] \geq list_{a3}[i]$ with any values `i` $\in$ `[0,3]`.
- Action 2 - Action 3: $list_{a2}[0] > list_{a3}[0]$. However, $list_{a2}[2] < list_{a3}[2]$.

According to this comparison, we find out two dominated strategies of Player 2: Action 2 (weakly dominated by Action 1) and Action 3 (weakly dominated by Action 1).

2.5.2 Dominant strategies

In contrast with dominated strategies, the dominant strategy always provides higher payoffs to the player than other strategies, no matter what the other player's strategy is. Besides, a player can not have more than one dominant strategy.

We can determine strategy A is the dominant strategy by checking if all of the other strategies are dominated by A or not. It is pretty simple because we can use the result from subsubsection 2.5.1 to check.
For example, Player 2 in Table 1 have two dominated strategies: Action 2 (weakly dominated by Action 1) and Action 3 (weakly dominated by Action 1). This player has three actions so we can conclude that Action 1 is a weakly dominant strategy.

2.6 Check mixed Nash Equilibrium by simulating the game

As the requirement, we implement a function that allows us to enter a mixed strategy for a player. Then we simulating the game over a hundred times with that mixed strategy and with the mixed Nash equilibrium. This part aims to check if the mixed strategy obtained by the Nash equilibrium is optimal or not. Similar to the subsection 2.4, this function always appears in the game of two players and two actions.

We simulate the game follow below steps:

- **Step 1:** Get entered p, q.
- **Step 2:** With each round of the simulation, we randomly select the action for each player base on entered p, q and calculated p, q. Then, we get the player's payoff in each case.
  Note: We use function `numpy.random.choice(...)` to randomly select the action.
- **Step 3:** Calculate the player's average payoff in each case

If the user does not enter p or q, we will use the calculated p, q from mixed Nash Equilibrium instead.

3 Graphic User Interface

In this project, we use the library `tkinter` to build the "user-friendly" GUI. The required inputs will be different according to each type of game. Such as the game has many players and many actions, the number of players and the number of actions for each player is the required inputs.

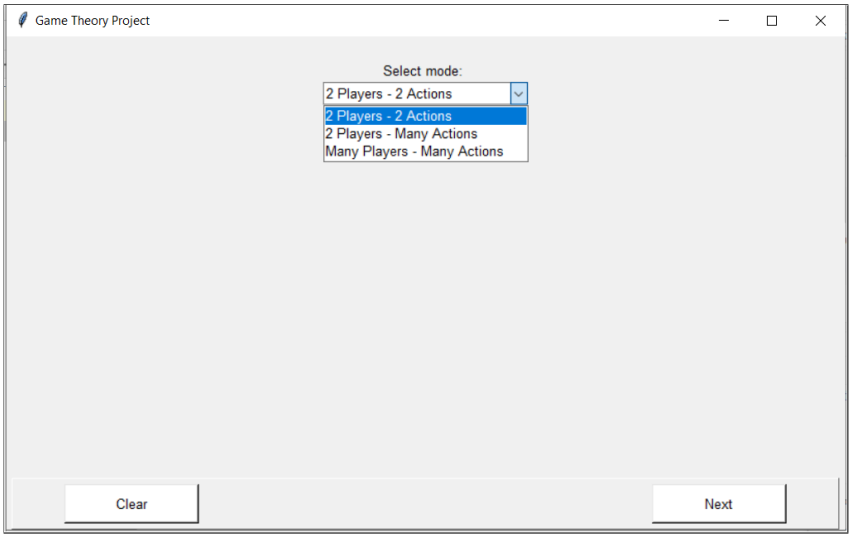

Figure 1: GUI - Begin Phase

More details of GUI interface will be shown in the section 4.

4 Verify with Test cases

You can check the detail demo at `https://youtu.be/n6Mgz4t0yZc`.

4.1 Two players - Two actions

We consider example 2 which is shown up in Table 2.

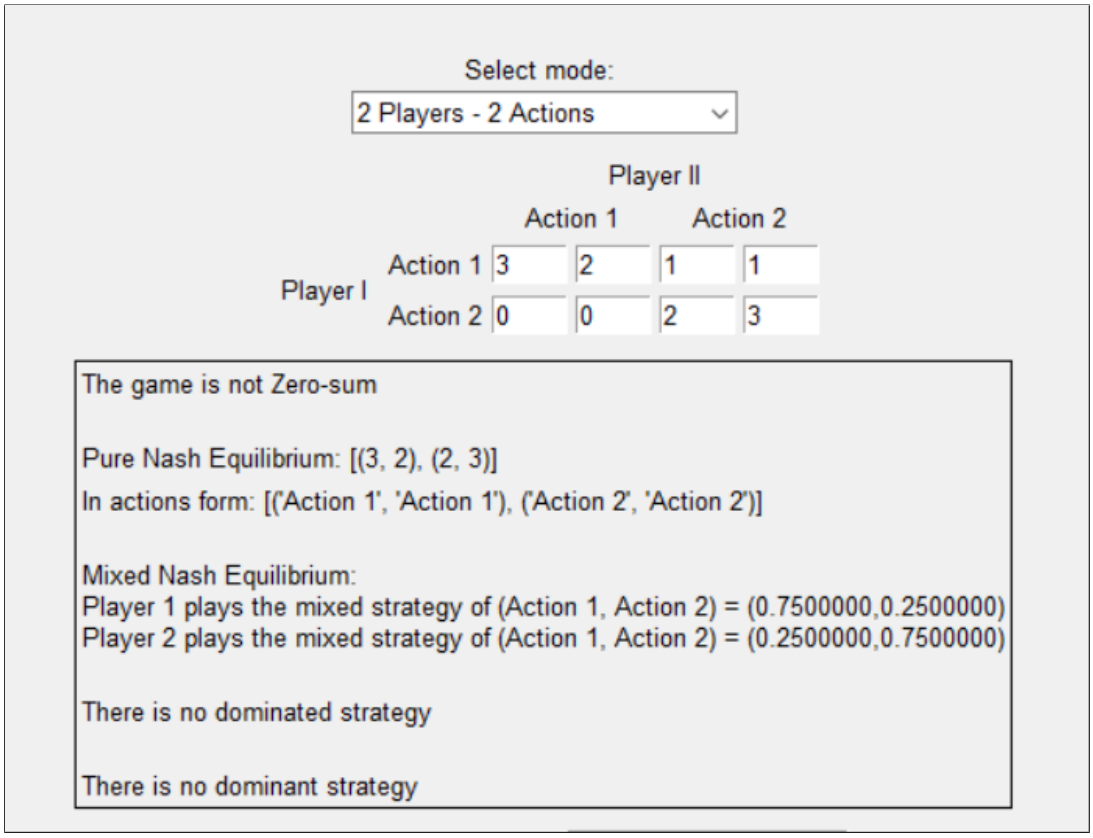

Figure 2: Result of Example 1

To check calculated mixed Nash Equilibrium is optiaml or not, we run the simulating function with `p = 0.5` and `q = 0.3`.

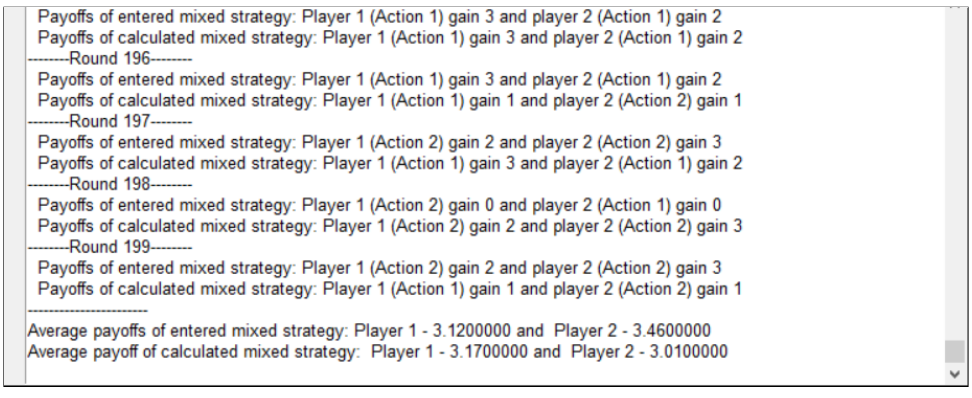

Figure 3: Result of simulating function

4.2 Two players - Many actions

We create an example to test this game. The below is an example of the game with two players and each player have three actions.

		Player 2		
		Action 1	Action 2	Action 3
Player 1	Action 1	-1,1	2,4	4,2
	Action 2	2,0	1,2	3,1
	Action 3	1,1	0,5	3,1

Table 4: Example 3

The result of this game in Table 4 as follows.

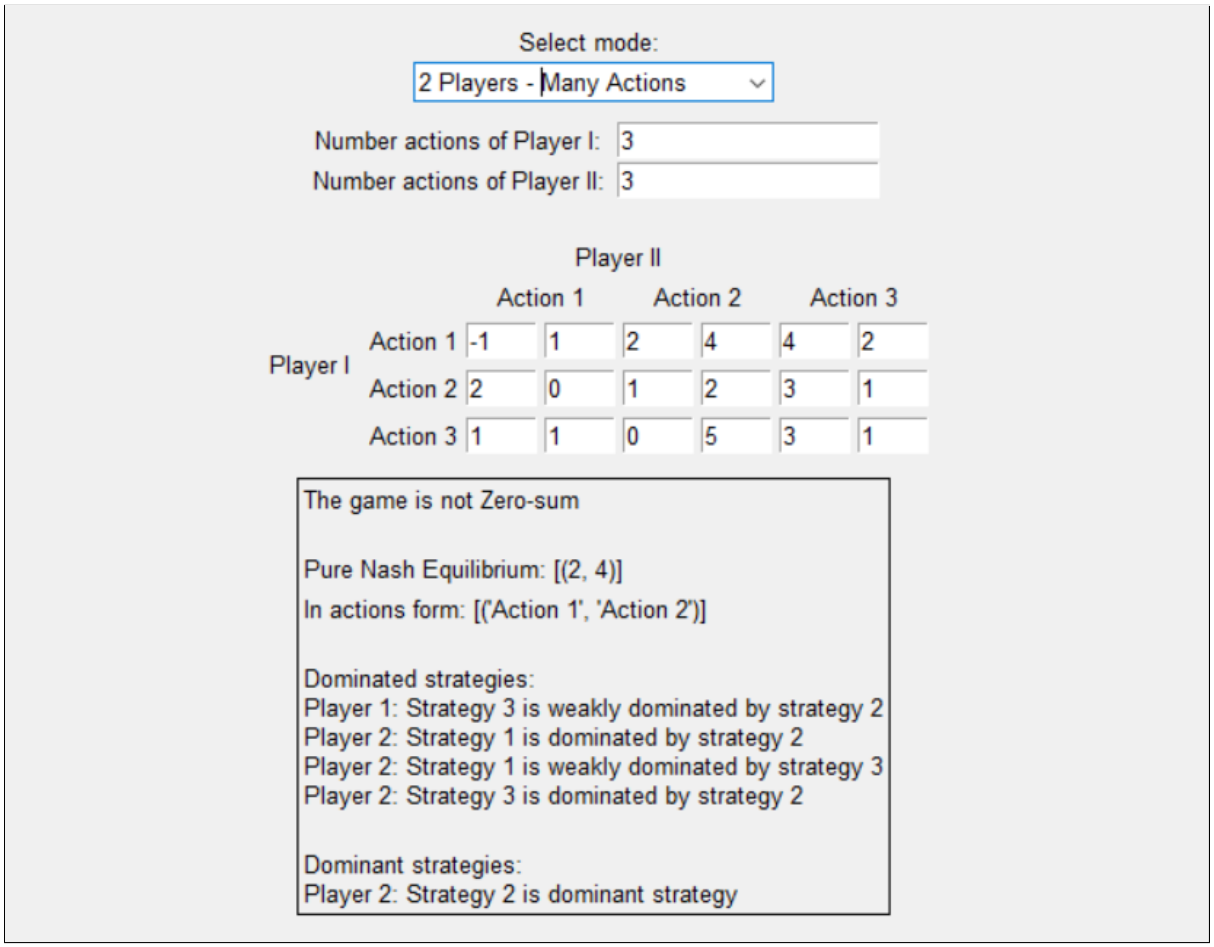

Figure 4: Result of Example 3

4.3 Many players - Many actions

We verify the example in Table 1. That is the game with 3 players: Player 1 have 2 actions, Player 2 have 3 actions and player 3 have 2 actions.

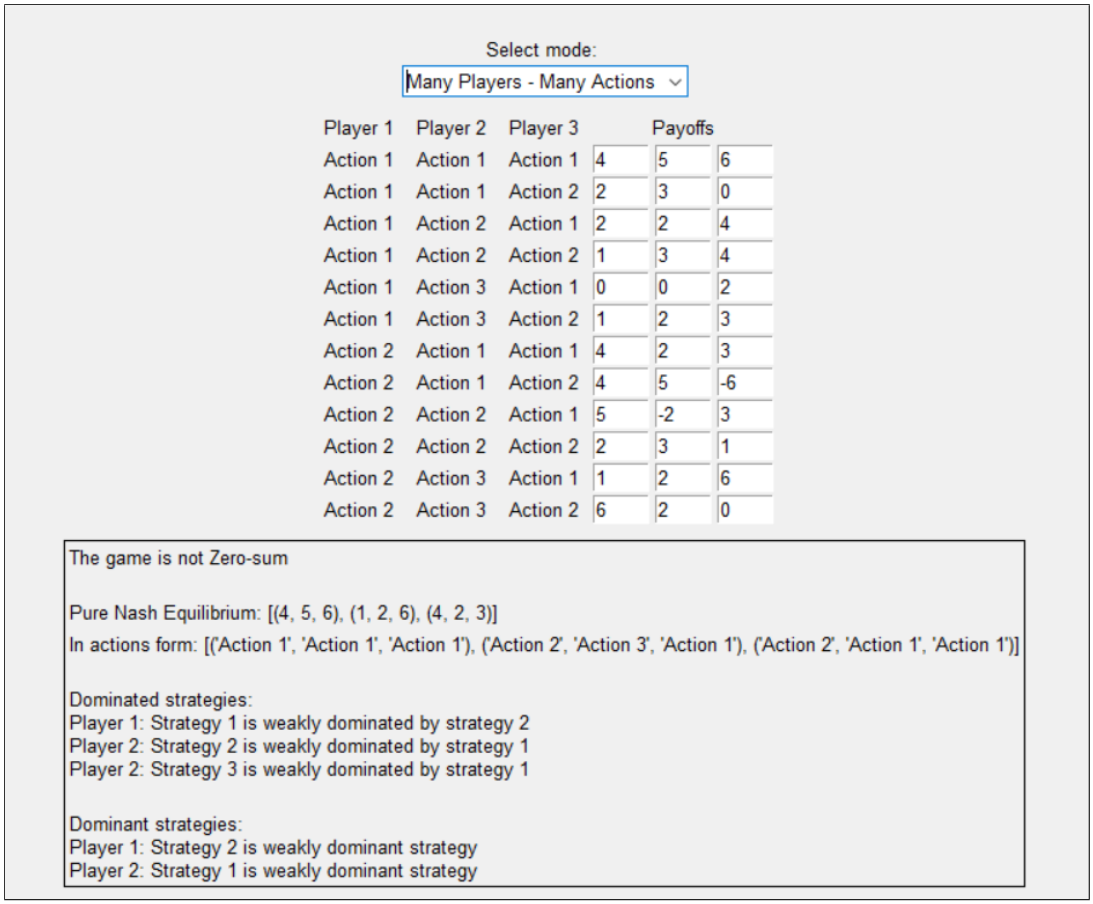

Figure 5: Result of Example 3

5 Conclusion

In this project, we successfully implement a program to solve the games with the different number of users and also the number of actions. I have completed this project through an understanding of the basic concept in Game Theory. Maybe there are some small deficiencies in the project but they have not affected the project, all test cases we tried are passed.