Problem 1 – Report

For project repository, please check: <https://github.com/DariMe20/Game-Theory-Applications>

**Problem Description**

The task is to create an algorithm to determine the **pure Nash Equilibria** (pure NE) of a normal-form game with 2 players. The algorithm should answer the following questions:

* **Existence** (or not) of pure NE in the game.
* **Find all pure NE in the game**.

The algorithm should be tested with finite normal-form games defined in terms of 2 players, strategies, and payoffs. There should also be an option to check the algorithm on games with payoffs randomly generated by the computer.

**Inputs:**

* Number of strategies for **Player 1**.
* Number of strategies for **Player 2**.
* A choice to input payoffs manually or randomly by the computer (YES/NO).
* If manually, payoffs for both players.

**Outputs:**

* All pure Nash Equilibria of the defined game.

**Pseudocode for finding pure NE**

BEGIN find\_pure\_nash\_equilibrium:

1. Initialize variable **pure\_ne = [] (empty list)** to store the pure Nash Equilibrium.

2. Initialize two lists to store the best responses:

**best\_response\_player1:** List of best responses for Player 1 to each of Player 2's strategies.

**best\_response\_player2:** List of best responses for Player 2 to each of Player 1's strategies.

3. FOR each Strategy j of Player 2 (loop through all strategies of Player 2):

Initialize **max\_payoff** = maximum payoff for Player 1 when Player 2 plays strategy j.

**best\_response\_player1[j] = i** if payoff\_player1(strategy i, strategy j)== max\_payoff.

4. FOR each strategy i of Player 1 (loop through all strategies of Player 1):

a. Initialize **max\_payoff** = maximum payoff for Player 2 when Player 1 plays strategy i.

b. **best\_response\_player2[i] = k** if payoff\_player2(strategy k, strategy i)== max\_payoff.

5. FOR each strategy i of Player 1:

FOR each strategy j of Player 2:

Check if strategy i is in the list of best responses for Player 1 against Player 2's strategy j. **AND**

Check if strategy j is in the list of best responses for Player 2 against Player 1's strategy i.

**if i in best\_response\_player1[j] and j in best\_response\_player2[i]**

IF both conditions are true:

- This means (i, j) is a Pure Nash Equilibrium.

- Add (Player 1's strategy i, Player 2's strategy j) along with their respective payoffs to pure\_ne:

**pure\_ne.append((player1\_strategies[i],player2\_strategies[j],payoffs\_player1[i][j],payoffs\_player2[j][i]))**

6. RETURN **pure\_ne** (list of all pure Nash Equilibrium found).

END find\_pure\_nash\_equilibrium

**Time Complexity Evaluation**

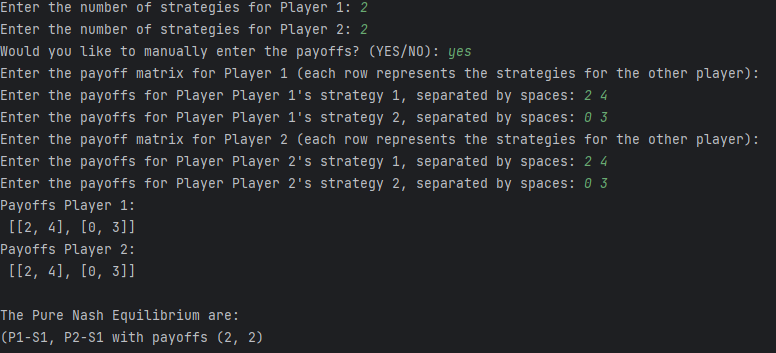
The time complexity of proposed **find\_pure\_nash\_equilibrium** function is **O(n \* m \* (n + m)),** where **n** is the number of strategies for Player 1 and **m** is the number of strategies for Player 2. The function can be broken down as follows:

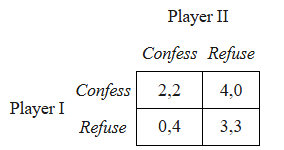
1. **Finding best responses for Player 1**: For each strategy of Player 2 **(O(m)),** the function iterates over all strategies of Player 1 **(O(n))** to calculate the best response. This results in a time complexity of **O(n \* m).**
2. **Finding best responses for Player 2:** Similarly, for each strategy of Player 1 **(O(n)),** the function iterates over all strategies of Player 2 **(O(m))** to calculate the best response, also resulting in a time complexity of **O(n \* m).**
3. **Checking each strategy pair for Nash Equilibrium**: The function checks all possible pairs of strategies (i, j) from both players **(O(n \* m))** to see if both are best responses to each other, resulting in a complexity of **O(n \* m).**

**The overall time complexity is thus O(n \* m \* (n + m)), combining these steps.**

**Results on different examples**

**Example 1: Prisoner’s Dilema – manual input**

*****Payoff Matrix:***

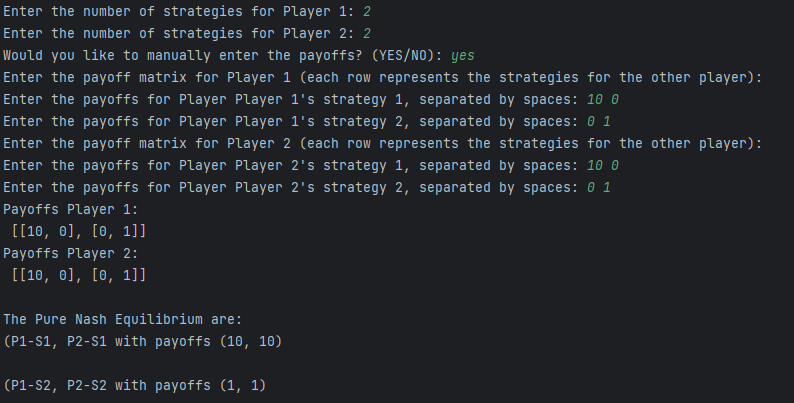
****

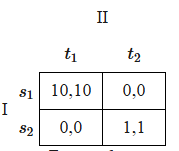
***Algorithm Results:***

The Pure Nash Equilibrium are:

P1-Strategy1, P2-Strategy1 with payoffs (2, 2)

**Example 2:**

*****Payoff Matrix:***

****

***Algorithm Results:***

The Pure Nash Equilibrium are:

(P1-S1, P2-S1 with payoffs (10, 10)

(P1-S2, P2-S2 with payoffs (1, 1)

**Example 3 – random generated payoffs on small number of strategies**

Enter the number of strategies for Player 1: 3 **The Pure Nash Equilibrium are:**

Enter the number of strategies for Player 2: 3 **P1-S3, P2-S2 with payoffs (6, 15)**

Would you like to manually enter the payoffs? (YES/NO): no **P1-S3, P2-S3 with payoffs (13, 15)**

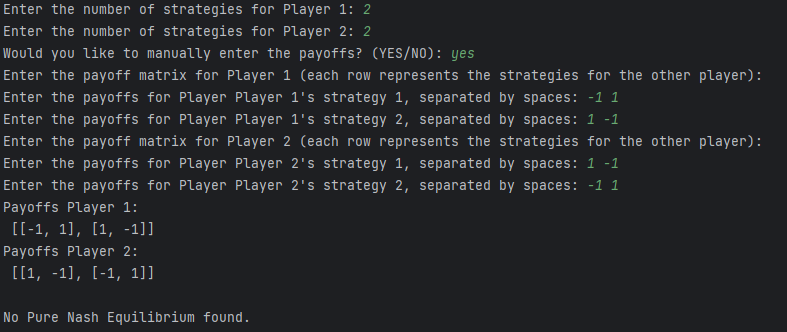
Payoffs Player 1:

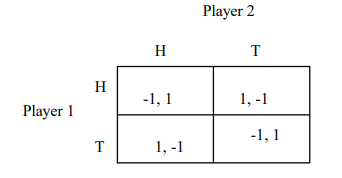
[[6, 2, 4], [7, 2, 7], [2, 6, 13]]

Payoffs Player 2:

[[1, 5, 6], [12, 13, 15], [4, 1, 15]]

**Example 4 – Matching Pennies**

****Payoff Matrix:**

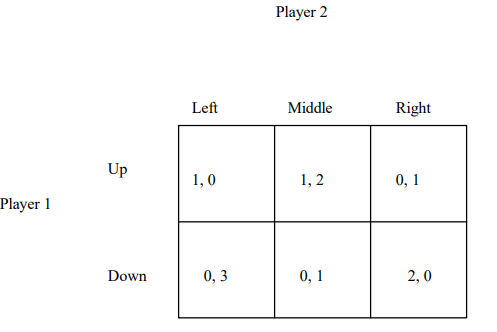


**Algorithm Results:**

*No Pure Nash Equilibrium found.*

**Example 5 – directions game – different number of strategies for the 2 players**

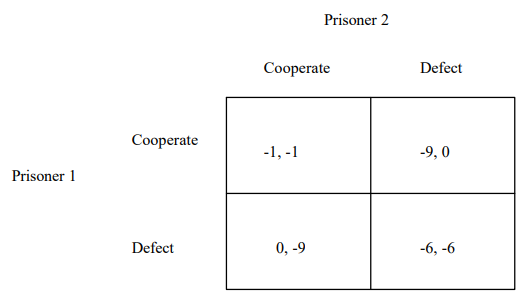
**Payoff Matrix Algorithm Results:**

* The Pure Nash Equilibrium are:*

*P1-S1, P2-S2 with payoffs (1,2)*

**Example 6 – Prisoner’s Dilema with other payoffs**

**Payoff Matrix Algorithm Result**



*The Pure Nash Equilibrium are:*

*P1-S2, P2-S2 with payoffs (-6, -6)*