International Journal of Innovative Research in Engineering
Volume 3, Issue 3 (May-June 2022), PP: 432-436
www.theijire.com
ISSN No: 2582-8746

Card Game Using PYGAMBIT

Dr.M.Marimuthu[1] ,Dr. V.Savithri [2], Roshini G[3], Kavya B[4], Harini C[5]

[1,2] Assistant Professor, Department of Data Science, Coimbatore Institute of Technology, Tamilnadu, India.
[3,4,5] Student, Department of Decision and Computing Sciences, Coimbatore Institute of Technology, Tamilnadu, India.

Howtocitethispaper:
Dr.M.Marimuthu[1] ,Dr. V.Savithri [2], Roshini G[3], Kavya B[4], Harini C[5], "Card Game Using Pygambit", IJIRE-V3I03-432-436.

Abstract: *Game theory problems are interesting to solve and often post many rules and challenges to the developer as well as the player. There can be n number of players with multiple criteria and they would need to decide on ways to win the game to maximize their profit and minimize their losses. Card game is another interesting multi criterion decision making problem, which has rules and strategies defined for each player. A payoff matrix is generated and a Nash equilibrium is established to find the maximum profit that can be gained from the payoff matrix for each player. Also we will test the importance score predicted by Nash game and see which level of importance brings in more profit.*

I.INTRODUCTION

Solve game theory problems using gambit, there can be n number of players with multiple criteria and they would need to decide on ways to win the game to maximize their profit and minimize their losses. Card game is another interesting multi criterion decision making problem, which has rules and strategies defined for each player. A payoff matrix is generated and a Nash equilibrium is established to find the maximum profit that can be gained from the payoff matrix for each player. Also we will test the importance score predicted by Nash game and see which level of importance brings in more profit.

II. RULES

We consider a game of cards where we state that there are 3 players, who will play with 3 face cards Jack, Queen and King. Once a player drops a card, the other gives out his card, one round consists of 3 cards, one card from each player. Scores assigned to Jack will be 10, Queen is 20 and King is 30. The highest card gets the score only if the face card is not present in one round. When the card is repeated twice or thrice then jack gets a score of 5 for all the players with the same card, queen gets a score of 10 and king gets a score of 15.

We tend to find the steady state and find the maximum outputs that the players can get from the game they play.

III. TOOLS USED

Tools and libraries used for solving a MCDM problem of cards include:

- Python
- Pygambit
- Numpy
- Nash
- Matplotlib

IV. STEADY STATE

A Steady state is a state or condition of a system or process that does not change in time.

Nash equilibrium:

Definitions:

Nash equilibrium is a concept in game theory in which every participant in a noncooperative game can optimize their outcome based on the other players' decisions. Nash equilibrium is achieved in a game when no player has any incentive for deviating from their own strategy, even if they know the other players' strategies. Nash equilibrium in game theory is a situation in which a player will continue with their chosen strategy, having no incentive to deviate from it, after taking into consideration the opponent's strategy. The Nash equilibrium is a decision-making theorem within game theory that states a player can achieve the desired outcome by not deviating from their initial strategy.

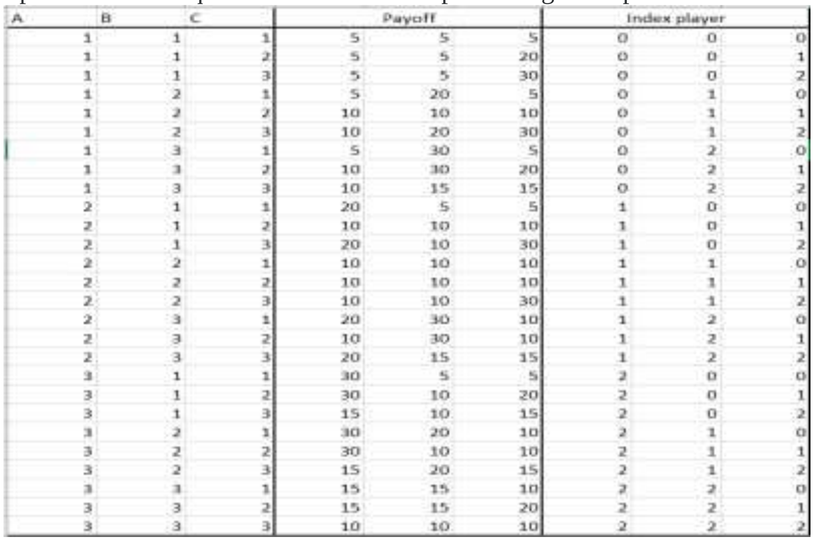
In the Nash equilibrium, each player's strategy is optimal when considering the decisions of other players. Every player wins because everyone gets the outcome they desire.The Nash equilibrium does not always mean that the most optimal strategy is chosen. Nash equilibrium is important because it helps a player determine the best payoff in a situation based not only on their decisions but also on the decisions of other parties involved. Nash equilibrium can be utilized in many facets of life.

Limitations of Nash Equilibrium:

The primary limitation of the Nash equilibrium is that it requires an individual to know their opponent's strategy. A Nash equilibrium can only occur if a player chooses to remain with their current strategy if they know their opponent's strategy.Furthermore, in multiple games played with the same opponents, the Nash equilibrium does not take into consideration past behavior, which often predicts future behavior.

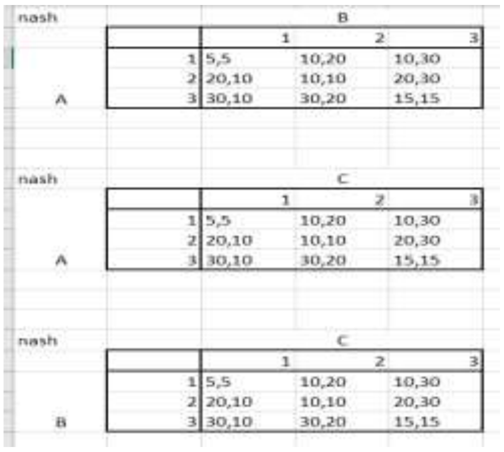
V. PAYOFF MATRIX

States the reward that each player gets for choosing an action in correspondence to an action by another player.
The below pay off is for all the chances that can occur in a game of cards for three players. We will then divide the payoff into two person game to implement Nash Equilibrium and obtain optimistic gain or profit.



A	B	C	Payoff			Index player		
1	1	1	5	5	5	0	0	0
1	1	2	5	5	20	0	0	1
1	1	3	5	5	30	0	0	2
1	2	1	5	20	5	0	1	0
1	2	2	10	10	10	0	1	1
1	2	3	10	20	30	0	1	2
1	3	1	5	30	5	0	2	0
1	3	2	10	30	20	0	2	1
1	3	3	10	15	15	0	2	2
2	1	1	20	5	5	1	0	0
2	1	2	10	10	10	1	0	1
2	1	3	20	10	30	1	0	2
2	2	1	10	10	10	1	1	0
2	2	2	10	10	10	1	1	1
2	2	3	10	10	30	1	1	2
2	3	1	20	30	10	1	2	0
2	3	2	10	30	10	1	2	1
2	3	3	20	15	15	1	2	2
3	1	1	30	5	5	2	0	0
3	1	2	30	10	20	2	0	1
3	1	3	15	10	15	2	0	2
3	2	1	30	20	10	2	1	0
3	2	2	30	10	10	2	1	1
3	2	3	15	20	15	2	1	2
3	3	1	15	15	10	2	2	0
3	3	2	15	15	20	2	2	1
3	3	3	10	10	10	2	2	2

Individual Payoff matrix for two players:



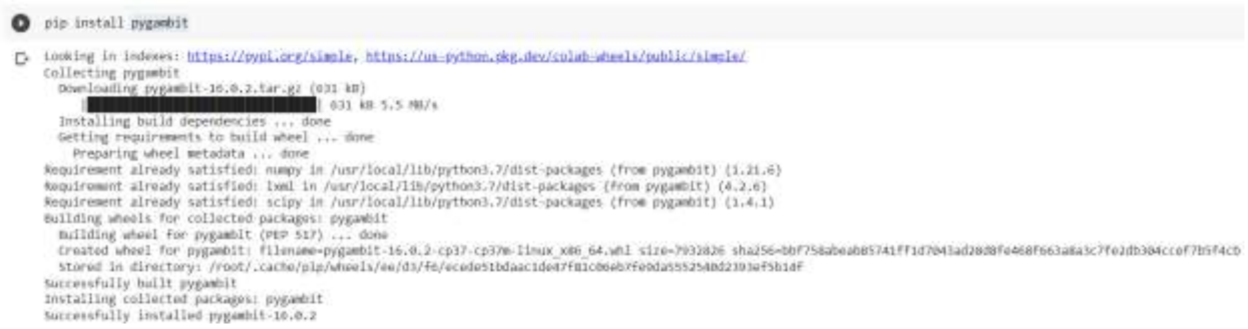
nash		B		
		1	2	3
	1	5,5	10,20	10,30
	2	20,10	10,10	20,30
A	3	30,10	30,20	15,15

nash		C		
		1	2	3
	1	5,5	10,20	10,30
	2	20,10	10,10	20,30
A	3	30,10	30,20	15,15

nash		C		
		1	2	3
	1	5,5	10,20	10,30
	2	20,10	10,10	20,30
B	3	30,10	30,20	15,15

VI. CODE

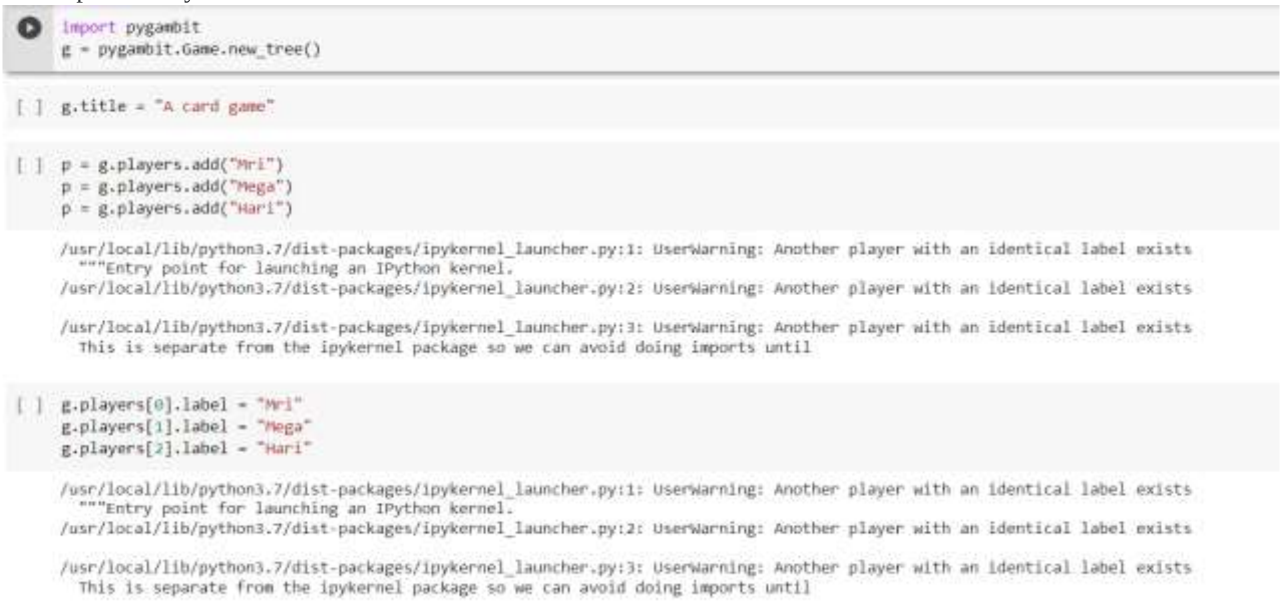
We will find out the best strategy for Player A Mri and Player B Mega.
NOTE: to install pygambit and nash



```
pip install pygambit
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pygambit
  Downloading pygambit-16.0.2.tar.gz (631 kB)
     |████████████████████████████████| 631 kB 5.5 MB/s
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from pygambit) (1.21.6)
Requirement already satisfied: lxml in /usr/local/lib/python3.7/dist-packages (from pygambit) (4.2.6)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from pygambit) (1.4.1)
Building wheels for collected packages: pygambit
  Building wheel for pygambit (PEP 517) ... done
  Created wheel for pygambit: filename=pygambit-16.0.2-cp37-cp37m-linux_x86_64.whl size=7932826 sha256=bbf758abeab85741ff1d7043ad2828fe468f663a8a3c7fe2db384ccof7b5f4cb
  Stored in directory: /root/.cache/pip/wheels/ee/d3/f6/ecede51bdaac1de47f81c06eb7fe0da55525402393ef5b1dF
Successfully built pygambit
Installing collected packages: pygambit
Successfully installed pygambit-16.0.2
```

Import library and initialize the matrix.



```
import pygambit
g = pygambit.Game.new_tree()
```

```
[ ] g.title = "A card game"
```

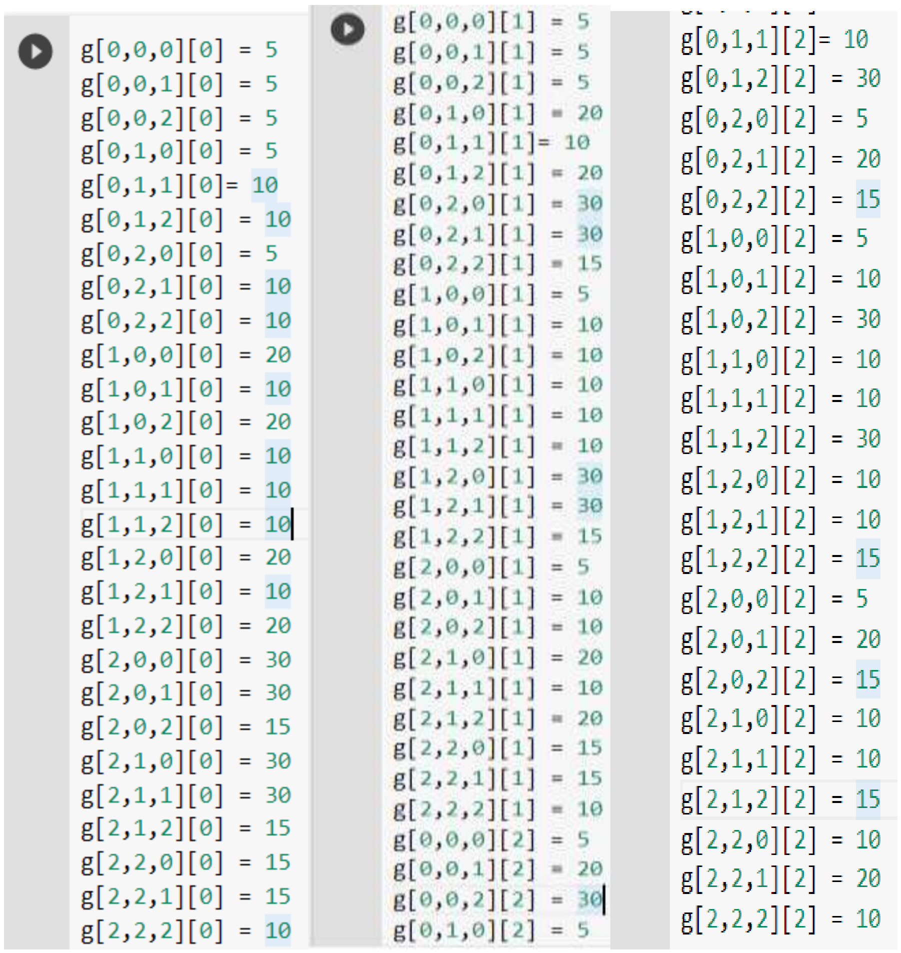
```
[ ] p = g.players.add("Mri")
    p = g.players.add("Mega")
    p = g.players.add("Hari")

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: Another player with an identical label exists
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: UserWarning: Another player with an identical label exists

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: Another player with an identical label exists
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
[ ] g.players[0].label = "Mri"
    g.players[1].label = "Mega"
    g.players[2].label = "Hari"

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: Another player with an identical label exists
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: UserWarning: Another player with an identical label exists

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarning: Another player with an identical label exists
  This is separate from the ipykernel package so we can avoid doing imports until
```

Initialize strategies:

```
#3 strategies for each player
g = pygambit.Game.new_table([3,3,3])
```

Payoff matrix:

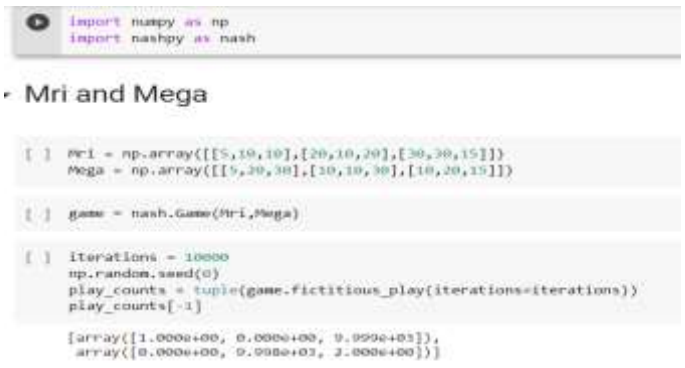


```
g[0,0,0][0] = 5
g[0,0,1][0] = 5
g[0,0,2][0] = 5
g[0,1,0][0] = 5
g[0,1,1][0]= 10
g[0,1,2][0] = 10
g[0,2,0][0] = 5
g[0,2,1][0] = 10
g[0,2,2][0] = 10
g[1,0,0][0] = 20
g[1,0,1][0] = 10
g[1,0,2][0] = 20
g[1,1,0][0] = 10
g[1,1,1][0] = 10
g[1,1,2][0] = 10
g[1,2,0][0] = 20
g[1,2,1][0] = 10
g[1,2,2][0] = 20
g[2,0,0][0] = 30
g[2,0,1][0] = 30
g[2,0,2][0] = 15
g[2,1,0][0] = 30
g[2,1,1][0] = 30
g[2,1,2][0] = 15
g[2,2,0][0] = 15
g[2,2,1][0] = 15
g[2,2,2][0] = 10
```

```
g[0,0,0][1] = 5
g[0,0,1][1] = 5
g[0,0,2][1] = 5
g[0,1,0][1] = 20
g[0,1,1][1]= 10
g[0,1,2][1] = 20
g[0,2,0][1] = 30
g[0,2,1][1] = 30
g[0,2,2][1] = 15
g[1,0,0][1] = 5
g[1,0,1][1] = 10
g[1,0,2][1] = 10
g[1,1,0][1] = 10
g[1,1,1][1] = 10
g[1,1,2][1] = 10
g[1,2,0][1] = 30
g[1,2,1][1] = 30
g[1,2,2][1] = 15
g[2,0,0][1] = 5
g[2,0,1][1] = 10
g[2,0,2][1] = 10
g[2,1,0][1] = 20
g[2,1,1][1] = 10
g[2,1,2][1] = 20
g[2,2,0][1] = 15
g[2,2,1][1] = 15
g[2,2,2][1] = 10
g[0,0,0][2] = 5
g[0,0,1][2] = 20
g[0,0,2][2] = 30
g[0,1,0][2] = 5
```

```
g[0,1,1][2]= 10
g[0,1,2][2] = 30
g[0,2,0][2] = 5
g[0,2,1][2] = 20
g[0,2,2][2] = 15
g[1,0,0][2] = 5
g[1,0,1][2] = 10
g[1,0,2][2] = 30
g[1,1,0][2] = 10
g[1,1,1][2] = 10
g[1,1,2][2] = 30
g[1,2,0][2] = 10
g[1,2,1][2] = 10
g[1,2,2][2] = 15
g[2,0,0][2] = 5
g[2,0,1][2] = 20
g[2,0,2][2] = 15
g[2,1,0][2] = 10
g[2,1,1][2] = 10
g[2,1,2][2] = 15
g[2,2,0][2] = 10
g[2,2,1][2] = 20
g[2,2,2][2] = 10
```

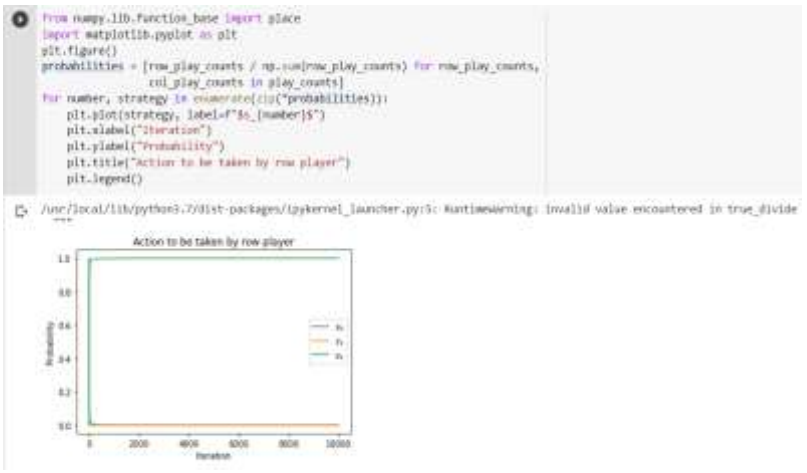
Nash Equilibrium:



```
import numpy as np
import nashpy as nash
```

Mri and Mega

```
[ ] Mri = np.array([[5,10,10],[20,10,20],[30,30,15]])
    Mega = np.array([[5,20,30],[10,10,30],[10,20,15]])

[ ] game = nash.Game(Mri,Mega)

[ ] iterations = 10000
    np.random.seed(0)
    play_counts = tuple(game.fictitious_play(iterations=iterations))
    play_counts[-1]

    [array([1.000e+00, 0.000e+00, 9.999e+03]),
     array([0.000e+00, 9.998e+03, 2.000e+00])]
```



```
from numpy.lib.function_base import place
import matplotlib.pyplot as plt
plt.figure()
probabilities = [row_play_counts / np.sum(row_play_counts) for row_play_counts,
                 col_play_counts in play_counts]
for number, strategy in enumerate(zip(*probabilities)):
    plt.plot(strategy, label=f"$s_{number}$")
    plt.xlabel("Iteration")
    plt.ylabel("Probability")
    plt.title("Action to be taken by row player")
    plt.legend()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: RuntimeWarning: invalid value encountered in true_divide
  """
```

```
[ ] equali = game.support_enumeration()
    for eq in equali:
        print(eq)

    (array([0., 1., 0.]), array([0., 0., 1.]))
    (array([0., 0., 1.]), array([0., 1., 0.]))
    (array([-0. ,  0.2,  0.8]), array([-0. ,  0.2,  0.8]))
```

Line 1: "(array([0., 1., 0.]), array([0., 0., 1.]))" Interpretation: This is the first Nash equilibrium. Player A chooses strategy 2 - queen and Player B chooses strategy 3 – king.

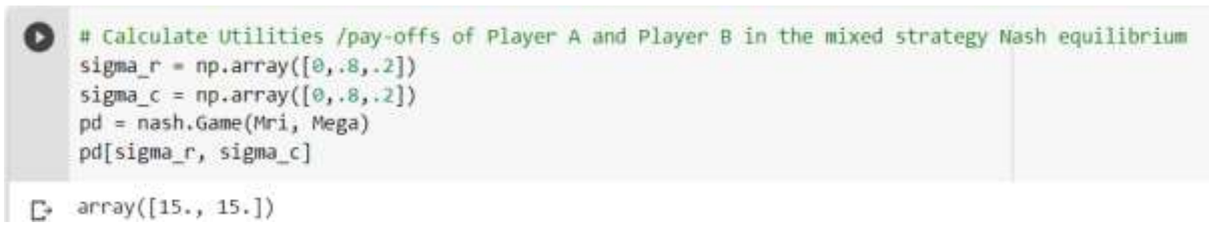
Line 2: "(array([0., 0., 1.]), array([0., 1., 0.]))" Interpretation: This is the first Nash equilibrium. Player A chooses strategy 3 - King and Player B chooses strategy 2 – Queen.

Instead what the players can do is randomize their strategies. Probabilities can be assigned to each pure strategy and the players can choose the strategies as per the probability assigned to it.

Line 3: (array([-0. , 0.2, 0.8]), array([-0. , 0.2, 0.8])) Player A assigns 20% importance to strategy 2 and 80% imporance to strategy 3. Similarly, Player B assigns the same.

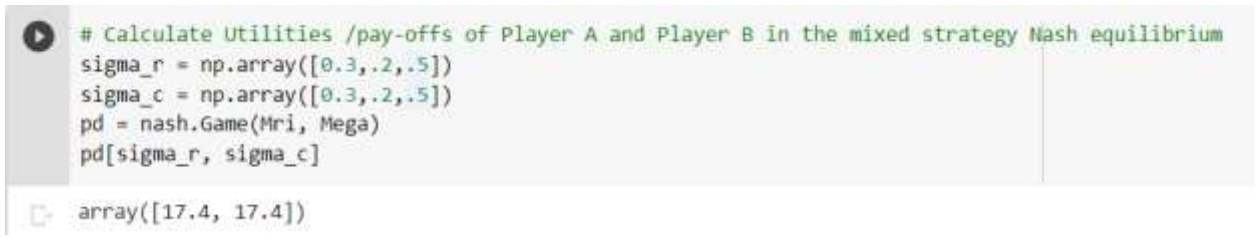
```
[ ] # Calculate Utilities /pay-offs of Player A and Player B in the mixed strategy Nash equilibrium
    sigma_r = np.array([0,.2,.8])
    sigma_c = np.array([0,.2,.8])
    pd = nash.Game(Mri, Mega)
    pd[sigma_r, sigma_c]

    array([18., 18.])
```

Since the payoff matrix is the same for all we retain all the players to select the results from line 1, 2 or 3. We compare the results from 20, 80 to 80, 20 % and compare. There is a reduction in pay-offs.



```
# Calculate Utilities /pay-offs of Player A and Player B in the mixed strategy Nash equilibrium
sigma_r = np.array([0,.8,.2])
sigma_c = np.array([0,.8,.2])
pd = nash.Game(Mri, Mega)
pd[sigma_r, sigma_c]

array([15., 15.])
```

Comparing strategy 1 with 2 and 3, there is a reduction in payoffs when importance is changed to strategy 1 and 3.



```
# Calculate Utilities /pay-offs of Player A and Player B in the mixed strategy Nash equilibrium
sigma_r = np.array([0.3,.2,.5])
sigma_c = np.array([0.3,.2,.5])
pd = nash.Game(Mri, Mega)
pd[sigma_r, sigma_c]

array([17.4, 17.4])
```

VII. CONCLUSION

Based on the payoff generated and rules we suggest the player to choose strategy 2 or 3, having a queen or king with maximum priority to queen card. This might change when the rule or the payoff matrix changes.

References

1. *https://gambitproject.readthedocs.io/en/latest/pyapi.html#:~:text=Gambit%20provides%20a%20Python%20interface,and%20install%20the%20Python%20extension.*
2. *https://gambitproject.readthedocs.io/en/latest/pyapi.html#:~:text=Gambit%20provides%20a%20Python%20interface,and%20install%20the%20Python%20extension.*