In [1]:
```
import nashpy as nash
import numpy as np
from texttable import Texttable
import random


#Pringint Payoff Matrix slighty more legible

def printPayoffMatrix(game):
    row_matrix = game.payoff_matrices[0]
    column_matrix = game.payoff_matrices[1]


    table = Texttable()
    col_str = ['']
    col_str.extend('Column {}'.format(c) for c in range (row_matrix.shape[1]))
    table.add_row(col_str)

    for row in range (row_matrix.shape[0]):
        row_str = ['Row '+str(row)]
        for column in range (row_matrix.shape[1]):
            row_str.append(str(row_matrix[row,column]) + ',' + str(column_matrix[row, column]))
        table.add_row(row_str)

    print(table.draw())

#flip coin fuction with while implications

def flipcoin(p):
    return True if random.random() < p else False


#game 1

A = np.array([[1, 2], [2, 0]])
B = np.array([[0, 1], [0, -1]])


game1 = nash.Game(A,B)

printPayoffMatrix(game1)
```

```
+-------+----------+----------+
|       | Column 0 | Column 1 |
+-------+----------+----------+
| Row 0 | 1,0      | 2,1      |
+-------+----------+----------+
| Row 1 | 2,0      | 0,-1     |
+-------+----------+----------+
```

In [2]:
```
# find the Nash Equilibria of Game 1

equilibria1 = game1.support_enumeration()
for eq in equilibria1:
    print(eq)
```

```
(array([1., 0.]), array([0., 1.]))
(array([0., 1.]), array([1., 0.]))
(array([0.5, 0.5]), array([0.66666667, 0.33333333]))
```

In [3]:
```
#finding utility of Game 1

cumu_util_r = 0
cumu_util_c = 0
flips = True
rounds = 0

while flips == True:
    flips = flipcoin(.999)

    r_action = random.choices([[1,0], [0,1]], k = 1, weights = [1,0])[0]
    c_action = random.choices([[1,0], [0,1]], k = 1, weights = [0.66666667,0.33333333])[0]

    sigma_r = np.array(r_action)
    sigma_c = np.array(c_action)

    util = game1[sigma_r, sigma_c]

    cumu_util_r += util[0]
    cumu_util_c += util[1]
    rounds +=1

else:
    avg_util_r = cumu_util_r / rounds
    avg_util_c = cumu_util_c / rounds

    print('Average Utility R = ', avg_util_r, ', Average Utility C = ', avg_util_c)
    print('Total Number of rounds is ' + str(rounds))
```

```
Average Utility R =  1.3414634146341464 , Average Utility C =  0.34146341463414637
Total Number of rounds is 574
```

In [4]:
```
#game 2

A = np.array([[3, 0], [5, 1]])
B = np.array([[3, 5], [0, 1]])

game2 = nash.Game(A,B)

printPayoffMatrix(game2)
```

```
+-------+----------+----------+
|       | Column 0 | Column 1 |
+-------+----------+----------+
| Row 0 | 3,3      | 0,5      |
+-------+----------+----------+
| Row 1 | 5,0      | 1,1      |
+-------+----------+----------+
```

In [5]:
```
# find the Nash Equilibria of Game 2

equilibria2 = game2.support_enumeration()
for eq in equilibria2:
    print(eq)
```

```
(array([0., 1.]), array([0., 1.]))
```

In [6]:
```
#finding utility of Game 2

cumu_util_r = 0
cumu_util_c = 0
flips = True
rounds = 0

while flips == True:
    flips = flipcoin(.999)

    r_action = random.choices([[1,0], [0,1]], k = 1, weights = [1,0])[0]
    c_action = random.choices([[1,0], [0,1]], k = 1, weights = [0,1])[0]

    sigma_r = np.array(r_action)
    sigma_c = np.array(c_action)

    util = game2[sigma_r, sigma_c]

    cumu_util_r += util[0]
    cumu_util_c += util[1]
    rounds +=1

else:
    avg_util_r = cumu_util_r / rounds
    avg_util_c = cumu_util_c / rounds

    print('Average Utility R = ', avg_util_r, ', Average Utility C = ', avg_util_c)
    print('Total Number of rounds is ' + str(rounds))
```

```
Average Utility R =  0.0 , Average Utility C =  5.0
Total Number of rounds is 529
```

In [7]:
```
#game 3

A = np.array([[4, 0, 2, -2], [0, 1, -2, -1]])
B = np.array([[1, 0, 1, 0], [0, 4, 0, 4]])

game3 = nash.Game(A,B)

printPayoffMatrix(game3)
```

```
+-------+----------+----------+----------+----------+
|       | Column 0 | Column 1 | Column 2 | Column 3 |
+-------+----------+----------+----------+----------+
| Row 0 | 4,1      | 0,0      | 2,1      | -2,0     |
+-------+----------+----------+----------+----------+
| Row 1 | 0,0      | 1,4      | -2,0     | -1,4     |
+-------+----------+----------+----------+----------+
```

In [8]:
```
#finding Nash Equilibrium for game 3

equilibria3 = game3.support_enumeration()
for eq in equilibria3:
    print(eq)
```

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

```
/Users/izzy/opt/anaconda3/lib/python3.7/site-packages/nashpy/algorithms/support_enumeration.py:196: RuntimeWarning:
An even number of (6) equilibria was returned. This
indicates that the game is degenerate. Consider using another algorithm
to investigate.

  warnings.warn(warning, RuntimeWarning)
```

In [9]:
```
#finding utility of Game 3

cumu_util_r = 0
cumu_util_c = 0
flips = True
rounds = 0

while flips == True:
    flips = flipcoin(.999)

    r_action = random.choices([ [1,0], [0,1] ], k = 1, weights = [1,0])[0]
    c_action = random.choices([ [1,0,0,0], [0,1,0,0], [0,0,1,0],[0,0,0,1] ],  k= 1, weights = [0,0.8,0.2,0])[0]

    sigma_r = np.array(r_action)
    sigma_c = np.array(c_action)

    util = game3[sigma_r, sigma_c]

    cumu_util_r += util[0]
    cumu_util_c += util[1]
    rounds +=1

else:
    avg_util_r = cumu_util_r / rounds
    avg_util_c = cumu_util_c / rounds

    print('Average Utility R = ', avg_util_r, ', Average Utility C = ', avg_util_c)
    print('Total Number of rounds is ' + str(rounds))
```

```
Average Utility R =  0.4079051383399209 , Average Utility C =  0.20395256916996046
Total Number of rounds is 1265
```

In [ ]:
```

```

Understanding Game Theory

Israel Nolazco
israelnolazco@lewisu.edu
CPSC-57200-02, Fall Two
Artificial Intelligence 2
Lewis University

I. Introduction

Game Theory is a mathematical/theoretical calculation in decision making. As the name implies the calculation is done by analyzing the payoff score of strategic interactions for each player. From there, we can calculate the utility and estimate the likely hood of each player taking an action. In short, Game theory's goal is to calculate the decision a player is going to take given their situation. It is also, however, it is important to consider the following assumption: each player is going to choose an action with the highest yield of payoff. Albeit Game Theory's full potential is providing a numerical value for all potential choices, which is crucial to find the best possible option for either player can take given their situation and still have a preferred outcome. This is considered the Nash Equilibrium.[1] In some cases, it is described as the "no regrets" choice. This is because regardless of which player benefits from the results of the game, it is the choice that has the highest payout for all players. In this document, we are going to explore the capabilities of Game Theory and going to further analyze its shortcomings.

II. Methods

In this document, we are going to use the programming language of Python and its package Nashpy[2]. This package is primarily built with the concepts and methods to calculate Game Theory. Furthermore, it is important to make a note that because we are going to analyzing each player's choices and each player; Matrices are a major key component for this analysis, and the package Numpy is going to be utilized to represent our data as matrices. Finally, in this document, we are going to analyze the following three games:

```
# Game 1
A = np.array([[1, 2], [2, 0]])
B = np.array([[0, 1], [0, -1]])

# Game 2
A = np.array([[3, 0], [5, 1]])
B = np.array([[3, 5], [0, 1]])

# Game 3
A = np.array([[4, 0, 2, -2], [0, 1, -2, -1]])
B = np.array([[1, 0, 1, 0], [0, 4, 0, 4]])
```

III. Analysis

As we can see in our methods section. There are two players, a row player, and a column player. In order to measure our analysis, we had the following assumptions for all of the games:

1. The row player plays a specified strategy and the column player always plays one of the randomly selected Nash equilibrium strategies (derived using support enumeration algorithm).
2. Strategy for the column player is selected at the beginning of the game and never changes after that
3. In each round of the game, each player makes a choice of a move with probabilities specified by their mixed strategy.

4. Once the move for each player is determined, the payoffs are calculated and accumulated as a total utility for each player
5. Each round repeats with a probability of r, which is a parameter of the system.
6. After the game ends (no longer repeats), we compute the average total utility for both players and output them to the screen.

```
+-------+----------+----------+
|       | Column 0 | Column 1 |
+-------+----------+----------+
| Row 0 | 1,0      | 2,1      |
+-------+----------+----------+
| Row 1 | 2,0      | 0,-1     |
+-------+----------+----------+
```

```
# find the Nash Equilibria of Game 1

equilibria1 = game1.support_enumeration()
for eq in equilibria1:
    print(eq)
```

```
(array([1., 0.]), array([0., 1.]))
(array([0., 1.]), array([1., 0.]))
(array([0.5, 0.5]), array([0.66666667, 0.33333333]))
```

Figure1. Game 1 Pay off Matrix and its Nash Equilibrium.

Given the above, let use analyze game 1 as it is shown in Figure 1. In this instance, the payoff matrix is shown along the Nash Equilibrium. As we can see, there are three potential equilibrium. Two of them are considered a pure strategy. This is because we can guarantee an equilibrium by having one hundred percent certainty of choosing this strategy will yield a "no regrets" situation. However, the third implies a probability measurement for each player to take a strategy and being "happy" with their option. In this case strategy with the row player array([0.5, 0.5]) indicates the row player will be happy 50/50 split if they decide to pick the first or second strategy. As for the column player, it looks like his strategy yield better by choosing Column 0 vs Column 1.

Now in order to further analyze the strengths in Game Theory, as explained earlier we went ahead calculated the utility of this game for each player based on the assumption that the row player will choose only one fix strategy while the column player will use a strategy from the Nash Equilibrium. The calculation was then run several times and then the average was then calculated over the number of iterations. In this document, it was then decided to investigate the utility based on a random feed to ensure the most accurate calculation. The following are the results:

Average Utility R = 1.3414634146341464 , Average Utility C = 0.34146341463414637

Total Number of rounds is 574

In essence, we are shown the row player has a higher utility than the column player, and therefore, the row player has a high probability of being satisfied and will more likely benefit from the results of the game based on the utility score.

```
+-------+----------+----------+
|       | Column 0 | Column 1 |
+-------+----------+----------+
| Row 0 | 3,3      | 0,5      |
+-------+----------+----------+
| Row 1 | 5,0      | 1,1      |
+-------+----------+----------+
```

```
# find the Nash Equilibria of Game 2

equilibria2 = game2.support_enumeration()
for eq in equilibria2:
    print(eq)
```

```
(array([0., 1.]), array([0., 1.]))
```

Figure 2. Game 2 Pay off Matrix and Nash Equilibrium

Although, it is not always the case where we see multiple Nash Equilibrium. In figure 2 for instance, we see only one Nash Equilibrium. Therefore, there is only one option for the players to be truly happy with our results. As for our average utility it was calculated as the following:

Average Utility R = 0.0 , Average Utility C = 5.0

Total Number of rounds is 529

```
+-------+----------+----------+----------+----------+
|       | Column 0 | Column 1 | Column 2 | Column 3 |
+-------+----------+----------+----------+----------+
| Row 0 | 4,1      | 0,0      | 2,1      | -2,0     |
+-------+----------+----------+----------+----------+
| Row 1 | 0,0      | 1,4      | -2,0     | -1,4     |
+-------+----------+----------+----------+----------+
```

```
#finding Nash Equilibrium for game 3

equilibria3 = game3.support_enumeration()
for eq in equilibria3:
    print(eq)
```

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

Figure 3. Game 3 Pay off Matrix and Nash Equilibrium

Finally, we consider our final game that has only two players, but four categories for options. As seen in Figure 3, our pay off matrix is a 2 x4 matrix. Also, our Nash equilibrium consists of six nash equilibrium. Immediately, Python warns us that this could be indicative of a degenerate game. What does this mean? Well, a degenerate game means that the game could potentially run an infinite number of times and the utility for either player shows no preference for either side.[3] In essence, everyone wins and everyone loses. There is no end in sight. This can be seen in our results:

Average Utility R = 0.4079051383399209 , Average Utility C = 0.20395256916996046

Total Number of rounds is 1265

The utility of the row player and column player is low and approaching the 50/50 split.

IV. Conclusion

Game Theory provides a great way to calculate the utility of each action for each player given their "payoff score," granted Game Theory works with several assumptions and lacks a general application. For instance, working on the assumption that each player is seeking its maximum payoff eliminates the possibility of players acting irrational or to accommodate the possibility of willfully ignoring the rules of the game. As the infamous line delivered by Alfred in the movie The Dark Knight, "some men just want to watch the world burn. [4]" Therefore, its application of Game Theory should be applied with caution and with a guarantee each player is aware of their actions and results for each step of the game. Lastly, each player should understand that Game Theory will not guarantee a winner or a loser, but rather a road map as to the likely utility of each action and a stalemate is more than likely.

References

[1] A. Hayes, "How Game Theory Works", Investopedia, 2020. [Online]. Available: https://www.investopedia.com/terms/g/gametheory.asp. [Accessed: 07- Dec- 2020].

[2] "Tutorial: building and finding the equilibrium for a simple game — Nashpy 0.0.19 documentation", Nashpy.readthedocs.io, 2020. [Online]. Available: https://nashpy.readthedocs.io/en/latest/tutorial/index.html#introduction-to-game-theory. [Accessed: 07- Dec- 2020].

[3] N. Nisan, T. Roughgarden, E. Tardos and V. Vazirani, 2007. [Online]. Available: http://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf. [Accessed: 07- Dec- 2020]

[4] V. De La Cruz, ""Some men just want to watch the world burn."", My Geek Wisdom, 2013. [Online]. Available: https://mygeekwisdom.com/2013/06/01/some-men-just-want-to-watch-the-world-burn/#:~:text="Some%20men%20just%20want%20to%20watch%20the%20world,understanding%20why%20he%20does%20the%20things%20he%20does. [Accessed: 07- Dec- 2020].