Iterated Prisioners Dilemma Tournament

Group Project for COMP 3710

By: Mio Tanaka, Suraiya Khanda, Samar Houssami, Ben Davidson

This project uses Jupyter Notebooks, Python 3.7, the Alexrod-Python and the Axelrod-Dojo library to run, analyze and visualise an Iterated Prisioners Dilemma Tournament and introduce machine learning strategies with finite state machines.

We made some minor modifications to the dojo library to improve the reporting and output. This is most reflected in the `training_output.csv` which now records detailed information about the players used in the simulation, mutation rate, bottleneck, size of state machine and the date/time to aid in reproducing the results.

We also made minor modifications to the main Axelrod library (`player.py`) to keep strategy name short so they would display correctly in charts and graphs

Import the axelrod library

```
import axelrod as axl
%matplotlib inline

from datetime import datetime
datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

```
'2019-04-10 21:00:07'
```

Prepare our Finite State Machine created with GA

```
0:C:0_C_0_C:0_D_2_D:1_C_2_C:1_D_2_C:2_C_2_D:2_D_0_D:3_C_5_D:3_D_6_C:
4_C_0_C:4_D_2_D:5_C_6_D:5_D_1_C:6_C_6_C:6_D_5_C:7_C_2_D:7_D_3_C
```

Was the best FSM that was could create with IPD Evolutionary Training. Using an Axelrod Finate State Machine Meta-Strategy we are able to import this into our tournament.

Appears in charts and graphs as **FSM Player**

```
from axelrod import Action
C, D = Action.C, Action.D
MMSB_GA_Strat_transitions = (
    (0, C, 0, C),
    (0, D, 2, D),
    (1, C, 2, C),
    (1, D, 2, C),
    (2, C, 2, D),
    (2, D, 0, D),
    (3, C, 5, D),
    (3, D, 6, C),
    (4, C, 0, C),
    (4, D, 2, D),
    (5, C, 6, D),
    (5, D, 1, C),
    (6, C, 6, C),
    (6, D, 5, C),
    (7, C, 2, D),
    (7, D, 3, C)
)

from axelrod.strategies.finite_state_machines import FSMPlayer

MMSB_GA_Strat = FSMPlayer(transitions=MMSB_GA_Strat_transitions,
    initial_state=0, initial_action=C)
```

Prepare the tournament

```
axl.seed(0)

# players = [s() for s in axl.demo_strategies]
players = [axl.Cooperator(), axl.Defector(),
           axl.TitForTat(), axl.Grudger(),
           axl.Random(), MMSB_GA_Strat]
tournament = axl.Tournament(players)
tournament.name = 'IPDP for COMP 3710'      # set the experiment name
tournament.turns = 200                      # set the number of turns
tournament.repetitions = 10                 # number of times to repeat
results = tournament.play()
```

```
Playing matches: 100%|██████████| 21/21 [00:00<00:00, 21.93it/s]
Analysing: 100%|██████████| 25/25 [00:00<00:00, 142.96it/s]
```

Tournament

{{tournament.name}}
Ran for {{tournament.turns}} turns and repeated {{tournament.repetitions}} times

```
print("The players were: ")
print(results.players)


print("\nTheir final ranking was: ")
print(results.ranked_names)
```

```
The players were:
['Cooperator', 'Defector', 'Tit For Tat', 'Grudger', 'Random', 'FSM Player']

Their final ranking was:
['Grudger', 'FSM Player', 'Tit For Tat', 'Defector', 'Cooperator', 'Random']
```

Detailed Results

For each round, how many times did each strategy win?

```
i = 0
for e in results.wins:
    print( "{0:>15}".format(results.players[i]) + " " + str(e) )
    i += 1
```

```
     Cooperator [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
       Defector [5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
    Tit For Tat [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        Grudger [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
         Random [1, 1, 2, 2, 2, 1, 1, 1, 1, 1]
     FSM Player [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

For each round, what were the scores?

```
i = 0
for s in results.scores:
    print("{0:>15} ".format(results.players[i]) + " " + str(s) )
    i += 1
```

```
     Cooperator  [2079, 2100, 2121, 2133, 2091, 2082, 2082, 2118, 2082, 2109]
       Defector  [2196, 2188, 2204, 2220, 2188, 2200, 2200, 2184, 2164, 2244]
    Tit For Tat  [2461, 2442, 2448, 2447, 2431, 2441, 2446, 2449, 2424, 2474]
        Grudger  [2590, 2622, 2558, 2556, 2584, 2630, 2594, 2552, 2642, 2606]
         Random  [1835, 1751, 1746, 1746, 1712, 1772, 1724, 1772, 1752, 1729]
     FSM Player  [2478, 2508, 2499, 2514, 2562, 2483, 2516, 2528, 2534, 2495]
```

Okay, pretty good, now let’s normalize those scores…

```
i = 0

for r in results.normalised_scores:
    e = []
    for c in r:
        e += [ float('{0:.2f}'.format(c ))]
    print("{0:>15} {1:} ".format(results.players[i], e ) )
    i += 1

# results.normalised_scores
```

```
     Cooperator [2.08, 2.1, 2.12, 2.13, 2.09, 2.08, 2.08, 2.12, 2.08, 2.11]
       Defector [2.2, 2.19, 2.2, 2.22, 2.19, 2.2, 2.2, 2.18, 2.16, 2.24]
    Tit For Tat [2.46, 2.44, 2.45, 2.45, 2.43, 2.44, 2.45, 2.45, 2.42, 2.47]
        Grudger [2.59, 2.62, 2.56, 2.56, 2.58, 2.63, 2.59, 2.55, 2.64, 2.61]
         Random [1.84, 1.75, 1.75, 1.75, 1.71, 1.77, 1.72, 1.77, 1.75, 1.73]
     FSM Player [2.48, 2.51, 2.5, 2.51, 2.56, 2.48, 2.52, 2.53, 2.53, 2.5]
```

This means that **{{results.players[0]}}** got, on average, a score of **{{results.normalised_scores[0][0]}}** per turn, per opponent

So who did well against who?

```python
player = 0
opponent = 0

for r in results.score_diffs:
    print("{:_<88}".format(results.players[player] + " vs "))
    opponent = 0
    for opp in r:
        e = []
        for c in opp:
            e += [ float('{0:2.2f}'.format(c ))]
        print("{0:>15} {1:} ".format(results.players[opponent], e ) )
        opponent += 1
    player += 1
    print()

# results.score_diffs
```

```
Cooperator vs ______________________________________________________________________
     Cooperator [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
       Defector [-5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0]
    Tit For Tat [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        Grudger [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
         Random [-2.67, -2.5, -2.33, -2.23, -2.58, -2.65, -2.65, -2.35, -2.65, -2.42]
     FSM Player [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Defector vs ________________________________________________________________________
     Cooperator [5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0]
       Defector [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    Tit For Tat [0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03]
        Grudger [0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03]
         Random [2.4, 2.35, 2.45, 2.55, 2.35, 2.42, 2.42, 2.33, 2.2, 2.7]
     FSM Player [0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03]

Tit For Tat vs _____________________________________________________________________
     Cooperator [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
       Defector [-0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03]
    Tit For Tat [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        Grudger [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
         Random [0.0, 0.0, -0.03, -0.03, -0.03, 0.0, 0.0, 0.0, 0.0, 0.0]
     FSM Player [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Grudger vs _________________________________________________________________________
     Cooperator [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
       Defector [-0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03]
    Tit For Tat [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        Grudger [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
         Random [2.4, 2.6, 2.23, 2.2, 2.38, 2.67, 2.45, 2.17, 2.7, 2.52]
     FSM Player [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Random vs __________________________________________________________________________
     Cooperator [2.67, 2.5, 2.33, 2.23, 2.58, 2.65, 2.65, 2.35, 2.65, 2.42]
       Defector [-2.4, -2.35, -2.45, -2.55, -2.35, -2.42, -2.42, -2.33, -2.2, -2.7]
    Tit For Tat [0.0, 0.0, 0.03, 0.03, 0.03, 0.0, 0.0, 0.0, 0.0, 0.0]
        Grudger [-2.4, -2.6, -2.23, -2.2, -2.38, -2.67, -2.45, -2.17, -2.7, -2.52]
         Random [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
     FSM Player [-0.68, -1.05, -1.05, -1.07, -1.55, -0.82, -1.3, -1.1, -1.18, -1.15]

FSM Player vs ______________________________________________________________________
     Cooperator [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
       Defector [-0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03, -0.03]
    Tit For Tat [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        Grudger [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
         Random [0.68, 1.05, 1.05, 1.07, 1.55, 0.82, 1.3, 1.1, 1.18, 1.15]
     FSM Player [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Take aways

- We can see that **Cooperator** is at a big disadvantage to **Defector**.

- **Defector** does very well against **Cooperator** and **Random** and is the only strategy will all positive scores.
- **FSM Player** (Our FSM) is very similar to **Tit-for-tat**. It performs better again random though.

When do different strategies cooperate?

```
i = 0

# Headers for the columns
print("{:17}".format(''), end ='')
for player in results.players:
    print("{:.3s} {:1}".format(player, ''), end='')
print()

# Rows
for r in results.normalised_cooperation:
    e = []
    for c in r:
        e += [ float('{0:.2f}'.format(c ))]
    print("{0:>15} {1:} ".format(results.players[i], e ) )
    i += 1

# results.normalised_cooperation
```

```
                 Coo  Def  Tit  Gru  Ran  FSM
     Cooperator [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
       Defector [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    Tit For Tat [1.0, 0.01, 1.0, 1.0, 0.5, 1.0]
        Grudger [1.0, 0.01, 1.0, 1.0, 0.01, 1.0]
         Random [0.5, 0.48, 0.5, 0.5, 0.5, 0.5]
     FSM Player [1.0, 0.01, 1.0, 1.0, 0.28, 1.0]
```

- As expected, **Cooperator** *always* cooperates.
- **Random: 0.5** cooperates… **50%** of the time
- **Tit for Tat** cooperates when the other player does.
- **FSM Player** (our FSM) cooperates like Tit-for-Tat except in learns it's lesson against Random and doesn't cooperate as oftern.

Ok, taken care of, let's send the main data to a .csv file

```
results.write_summary('results/summary.csv')
# import csv
# with open('summary.csv', 'r') as outfile:
#     csvreader = csv.reader(outfile)
#     for row in csvreader:
#         print(row)
```

Now let's visualize the results

```
# some extra graph options
# import seaborn as sns
import matplotlib.pyplot as plt
```

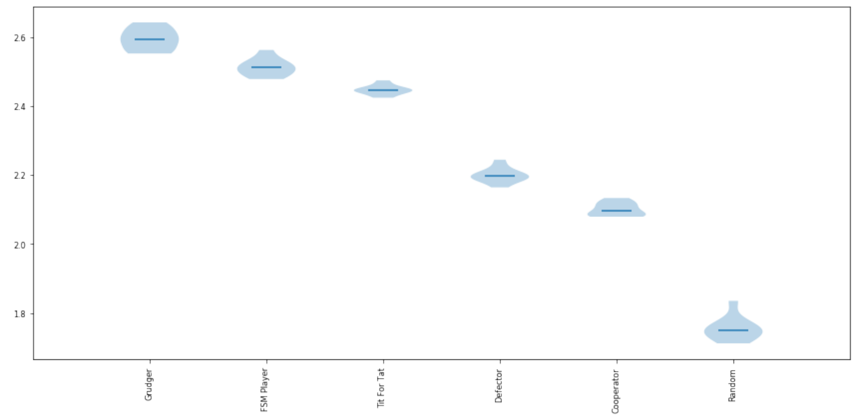
Violin plot Graph

The violin plot graph visualizes average scores in a way that helps us see stochastic effects. If all plyaers behave predictabilly horizontal lines will be shown. If a player behave stochastically, the random effects will be able to be visualized. The addition of a single stochastic strategy introduces stochastic effects into all results.

Violin plots are similar to boxplots but they are able to show the depth of data. The curve of the shape is a kernel density estimation, that shows the distribution of of the results.

Source: Violin Plots in Seaborn, Violin Plots 101: Visualizing Distribution and Probability Density

```
plot = axl.Plot(results)
b = plot.boxplot()
```

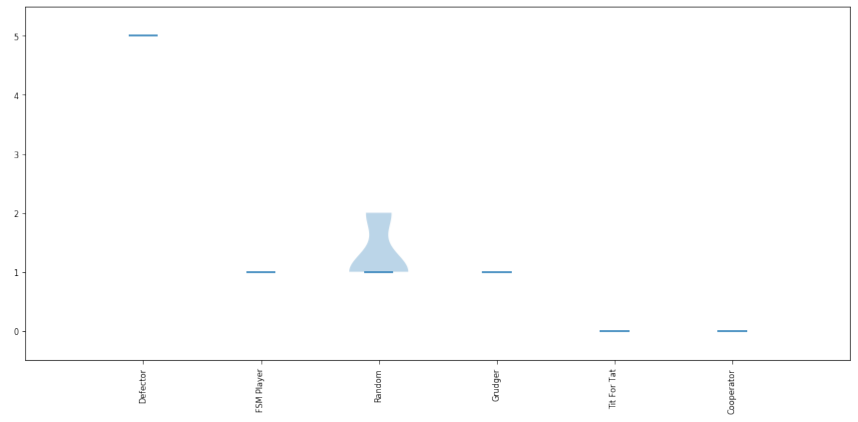


png

Win plot Graph

The win plot graph is…

```
p = plot.winplot()
```

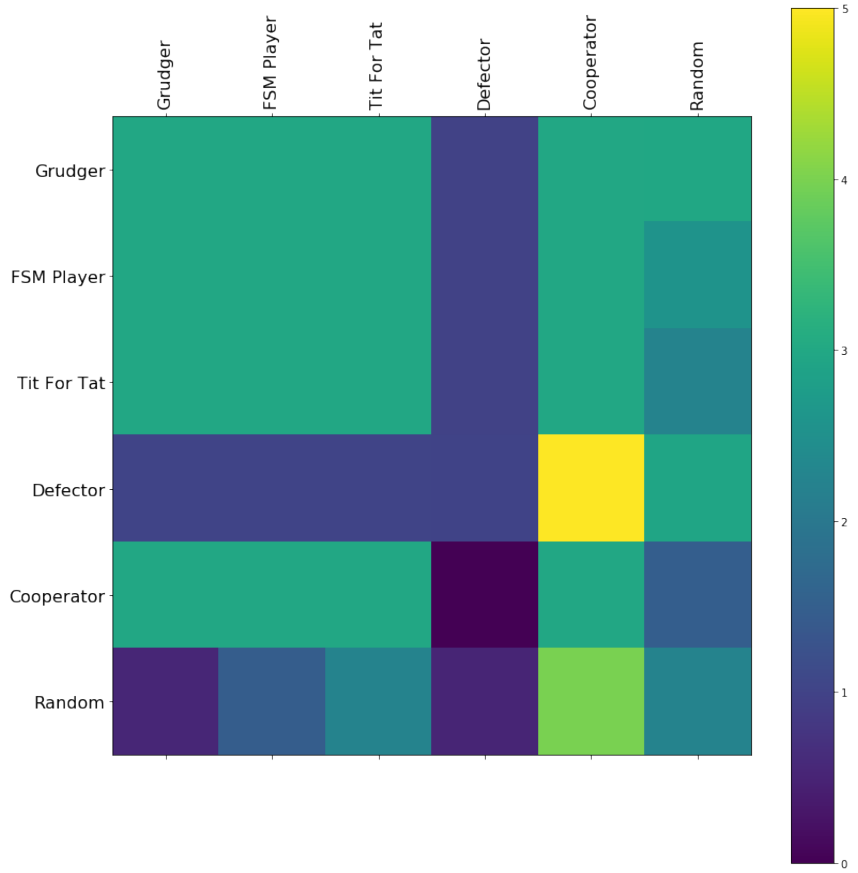


png

Payoff Graph

The payoff graph visualizes performance against other competitors. Brighter colours represent more success. For example, bright yellow indicates that **Defector** does very well against **Cooperator**

```
pay = plot.payoff()
```



png

Finally, let’s save a copy of our graphs

```
axl.Plot.save_all_plots(plot)
```

```
Obtaining plots: 100%|██████████| 6/6 [00:00<00:00,  6.87it/s]
```

output_32_0.png (png) [798x821]