|  |
| --- |
|  |
| ipd-sim: A Simulation Framework for Prisoner’s Dilemma Scenarios  CSC400 |
| |  |  |  | | --- | --- | --- | | Jessica Boe | 12/1/18 | Dr. Antonios | |

Contents

[Introduction 2](#_Toc518254999)

[Architecture 3](#_Toc518255000)

[Type of System 3](#_Toc518255001)

[Platform 3](#_Toc518255002)

[Inputs and Outputs 3](#_Toc518255003)

[User Interface 7](#_Toc518255004)

[Use Cases 8](#_Toc518255005)

[Sequence Diagrams 9](#_Toc518255006)

[Structural Design 10](#_Toc518255007)

[Class Diagrams 10](#_Toc518255008)

[Data 11](#_Toc518255009)

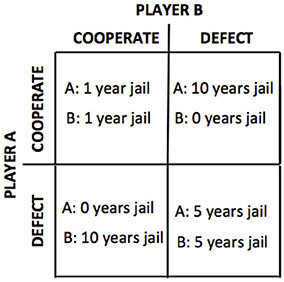
[User Interface Design 12](#_Toc518255010)

# Introduction

## The Prisoner’s Dilemma: Background

The ‘prisoner’s dilemma’ is widely regarded as one of the foundations in the area of game theory. With applications in diverse areas from environmental protectionism, to economics, to even foreign relations and defense, the prisoner’s dilemma is an interesting case study in human interactions across a host of disciplines.

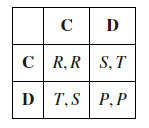
In a basic, single game prisoner’s dilemma, there are two participants with a simple, four square matrix that allows each player two options: to cooperate with the other player, or to defect. This was originally set up in the context of an actual ‘prisoner’s dilemma’, where the two participants are criminals who have had no prior interaction with one another, with the following payoff matrix:



From this example, we can see that the least risky move to take would be for each player to cooperate. However, if a player is selfish, they would probably be more likely to defect under the assumption that their opponent will cooperate, so that they get off with no time spent in jail while their opponent gets the maximum sentence.

Additionally, from this matrix, we can view what is referred to as the Nash Equilibrium, or the solution to a game in which each player has nothing to gain by changing their strategy. In this case, the Nash Equilibrium is for each player to betray one another, because even if mutual cooperation leads to a better outcome, if one player chooses to cooperate and the other doesn’t the outcome for the other player is worse.

From the classic prisoner’s dilemma game, we can derive the matrix into a simpler form to apply it to a broader spectrum of areas. The new matrix is:



Here, the parameters have taken variable forms, with R pertaining to the reward payoff, formerly -1, T pertaining to the temptation payoff, formerly 0, S pertaining to the sucker payoff, or -10 before, and P pertaining to the punishment payoff, formerly -5. These parameters can be tweaked to suit different models and purposes, and the parameters will have a drastic effect on the way that a player attempts the game. In order for a game to be classified as a prisoner’s dilemma, the parameters must satisfy the following chain of inequalities, although when using this software, these variables can be set to whatever the simulation end user desires to fit their unique purposes:

**T > R > S > P**

The final thing to keep in mind when experimenting with the prisoner’s dilemma is that players have unique strategies, which must be modeled when making simulations. While including what are regarded as ‘dumb’ strategies is important in regards to making sure that the data output from the simulation makes sense, it’s important to model in more ‘intelligent’ strategies in as well. These provide a more realistic feel for how a real human may play the game.

## The Iterated Prisoner’s Dilemma

While the single game prisoner’s dilemma is interesting, it’s easy to conclude that the safest way to play is to always cooperate, and the riskiest yet most potentially rewarding way to play is to defect. That gets boring pretty quickly, so what happens when players are pitted each other in matches that last several turns? What happens when players are pitted in multiple matches that last several turns each? What if multiple players with multiple strategies play several matches? This muddies up the waters quite a bit, but also provides an end user an easy way to set parameters for whatever purpose they need to and send in several types of players to see what the results are. This will be applied in three different ways within this simulation framework:

* Tournament
* Ecological
* Moran

The results of these simulations can be stored in CSV format, as well as viewable through various graphs and population maps.

## Applications for Ensuring Fair Play in Blockchains

Mining for cryptocurrencies such as bitcoin in a proof-of-work environment is deliberately consumptive of energy. Bitcoin and Bitcoin Cash currently account for use of about 0.31% of the world’s energy consumption. Due to these costs, the second-largest cryptocurrency by market cap, Ethereum, is switching to a hybrid of a proof of work environment with what is known as a proof-of-stake environment. Ethereum will rely on virtual miners, or validators, who stake Ether to the system. Validators are rewarded when they follow the rules and lose some or all of their deposit when they break them. Proof of stake is used as a checkpoint every 50 blocks to offer evidence that the blocks hold valid transactions.

ipd-sim can show how a crypto economy may look with different user types, population numbers, time periods, incentives, and disincentives for a proof-of-stake-based system, and whether the current incentive structure is playing out fairly or is being exploited. eth-sim will allow its user to set parameters and run a program via a console-based system that enables them to test whether a sample population of a crypto economy is adequately incentivized into validating transactions on the blockchain. The requirement for the amount of Ether to be staked can be tweaked, as a smaller population will have a large enough stake of Ether and thus be able to validate transactions. If the “rich” keep getting “richer”, what effects will that have on the overall economy if an actor behaves selfishly? If only a relatively small amount of Ether needs to be put at stake and the pool for potential validators grows larger, will the “bad” actors outweigh those acting fairly? This is just another application for ipd-sim, and perhaps in the future the base parameters can be adjusted to fit blockchain validation purposes.

# Architecture

## Type of System

ipd-sim is a simulation software that interacts with its users via a simple GUI system. The user is prompted to input parameters that grant access to various simulations, with the end goal showing how prisoner’s dilemma players may react to incentivization structures, with Moran Process and Ecological tournament-styled generation mechanics to model which type of actor responds best through multiple generations of birthing and eliminating bots. The results are then split up in visual form,

## Platform

The application is standalone for PC, potentially for Mac if the build allows it. It is built in Python 3 with the help of numerous plugins to implement important features without reinventing the wheel. Such examples include Axelrod to help model the prisoner’s dilemma process and multiple actor types, simulation helpers, GUI incorporation with PyQt, and more elegant file creation for displaying results.

## Inputs and Outputs

Actors are pitted against each other in a generic Prisoner’s Dilemma setup that is multiplied against positive and negative incentives and disincentives. In every generation, participants are pitted against a surviving opponent with four possible outcomes:

* Mutual cooperation: (C, C)
* Defection: (C, D) or (D, C)
* Mutual defection: (D, D)

These correspond to payouts (both positive and negative) as shown:

|  |  |  |
| --- | --- | --- |
|  | **Cooperate** | **Defect** |
| **Cooperate** | (R, R) | (S, T) |
| **Defect** | (T, S) | (P, P) |

For this to constitute a Prisoner’s Dilemma, these parameters must be true:  
 **T > R > P > S**

* **R:** Reward payoff
* **P:** Punishment payoff
* **S:** Sucker payoff
* **T:** Temptation payoff

The parameters that ipd-sim’s user can tweak are:

1. Reward – The top left block in the matrix, this is the reward both players get for mutually cooperating
2. Punishment – The bottom right block in the matrix, this is the punishment both players are subject to for mutually defecting
3. Sucker – If one player cooperates and one defects, the cooperating player is subject to this parameter
4. Temptation – If one player cooperates and one defects, the defecting player enjoys this reward
5. Simulation Length – How many generations the simulation will consist of between all participants. This parameter is only applicable to ecological tournaments, as in the Moran and Tournament games, a winner is found without regard for simulation length
6. Games per Generation – How many matches a player must compete against another player to determine a winner
7. Types of simulation participants – These are the various types of strategies that actors could use in their attempt to win. These are the basic types of bots that a user can experiment with. These parameters are also subject to noise, a changeable parameter that can make an actor’s decision subject to mistakes (i.e., a bad actor accidentally cooperates, and a good actor unintentionally acts badly):
   1. Cooperator – Actor will always cooperate
   2. Defector – Actor will always defect
   3. Tit for Tat – A player starts by cooperating and then mimics the previous action of the opponent
   4. Adaptive Tit for Tat – If an opponent cooperated last cycle, then a ‘world’ variable is set to allow them to be more likely to cooperate based on the opponent’s overall cooperation rate, or else they will defect. This is a ‘smart’ adaption of Tit for Tat
   5. Grudger – Player starts by cooperating, then will defect if at ANY point the opponent defects
   6. Copycat – Actor will cooperate with probability *p* if the opponent’s cooperation ratio is p, starting with a random decision
   7. Revised Downing – Attempts to estimate the next move of the opponent by estimating the probability of cooperating given that they defected or cooperated on the previous round, (p(C|D)) or (p(C|C)). These probabilities are continuously updated during play and the strategy attempts to maximize the long-term play. Initial values: p(C|C)=p(C|D)= 0.5. This strategy came 10th in Axelrod’s original tournament but it would have won if it had been implemented correctly.
   8. Pavlov – Attempts to classify its opponent as one of four strategies: Cooperative, Always Defect, Tit for Tat, or Random. It then responds in a manner intended to achieve mutual cooperation or to defect against uncooperative opponents
   9. Random – Actor will always make completely random decisions

Nydegger – Arguably the most complex algorithm included in this set, the program begins with Tit for Tat for the first three moves, except that if it was the only one to cooperate on the first move and the only one to defect on the second move, it defects on the third move. After the third move, its choice is determined from the 3 preceding outcomes in the following manner.

A=16a1+4a2+a3

Where ai is dependent on the outcome of the previous i th round. If both strategies defect, ai=3, if the opponent only defects: ai=2 and finally if it is only this strategy that defects then ai=1.

Finally this strategy defects if and only if:

A∈{1,6,7,17,22,23,26,29,30,31,33,38,39,45,49,54,55,58,61}

Thus if all three preceding moves are mutual defection, A = 63 and the rule cooperates. This rule was designed for use in laboratory experiments as a stooge which had a memory and appeared to be trustworthy, potentially cooperative, but not gullible. This strategy came 3rd in Axelrod’s original tournament.

1. Number of each actor – For each actor type in the game, the user can specify the number represented in the genesis generation. This can be approximated to a ratio to minimize complexity; for example, if you assume that most people are cooperative, you might include mostly cooperators and perhaps one or two of other strategies.
2. Mistake probability – The probability that an actor will act against the decision that they implemented
3. Mode – The user can checkbox whether they’d like to run a tournament, an eco tournament, a Moran process, a combination of some of those, or all three
4. Output – The user can checkbox whether they’d like to see a CSV file, a population map, various charts, a combination of some of those, or all three

This information is generated and the computer may have to run a potentially computationally-intensive process to crunch the numbers. Data for each generation is summed to find statistics of interest, such as how much Ether the richest and poorest population members contain and which actor classes fared the best and worst as far as longevity through multiple generations, through charts and CSV files. Additionally, this will be saved into a file that keeps track of summary data for previous scenarios, which the user can view to think about how they would build out their incentivization structures.

## User Interface

The UI looks like this:

# 

# Implementation

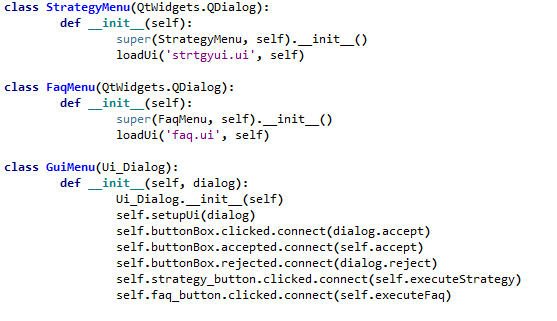
This program relies heavily on the use of two key plugins, Axelrod and PyQT. Axelrod has a sophisticated library of vetted strategies that can be pushed into Python projects. It is well documented and has a small but active community behind it, who were quick to help when issues were encountered during this project. Axelrod effectively operates much of what is “under the hood” in this program, while PyQT provides a mean to a convenient user interface which can be easily manipulated to emulate a plethora of scenarios.

Figure 1: Initializing GUI

This two first classes, strategyMenu and faqMenu, load separate UI files that have been prebuilt with PyQT Designer, which allows for creation of custom interfaces. Ultimately, a UI file looks like this example, part of the primary view. While tedious to look at, it provides a somewhat efficient way to link Python and a GUI.

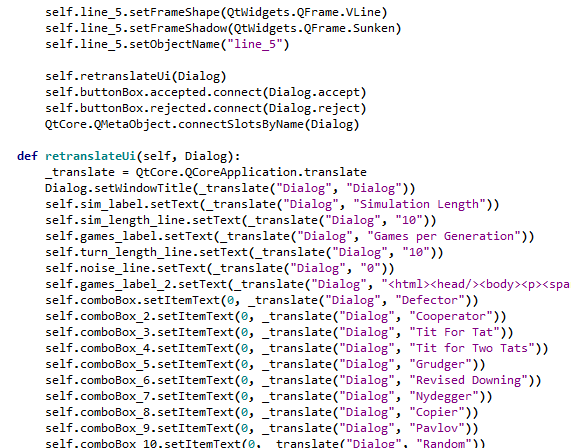


Figure 2: Part of Main UI

After the GUI is built, a parent window that lets the user input their desired settings appears. These settings are then fed into the back end of the program for simulating experiments accordingly. Tournament and Moran results are accessible via CSV and graphs, while Ecological style mode can be viewed visually via graph. The reason for this is the large amount of computational intensity required with these modes to produce results (especially Moran), and sifting through the data is extremely hard. In later versions of the program, this might be changed when more time can be devoted to creating an easy way to analyze this output.



Figure 3: Parameters to run sim

As can be seen from the figure above, the runSim function takes 15 parameters. This is more or less the “operating system” of this program, as it directs which simulations get run and how.

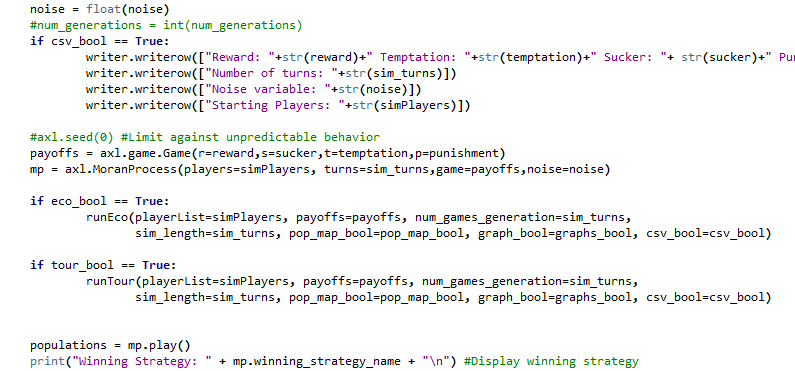


Figure 4: Part of main program

In Figure 4, we set the payoffs the game with the user’s input and also initializes the record keeping for the CSV file should it be selected. Of note, we can see the commented out function axl.seed(0). Feeding a seed into the simulation is important if you are dealing with strategies who may react randomly or are so close in score to one another that a clear winner is only a few fractions of a point in front of the second-place winner. While this is unnecessary for the “dumber” strategies, as we get into more complex ways to play the game like Nydegger or Revised Downing, we see more and more randomness.

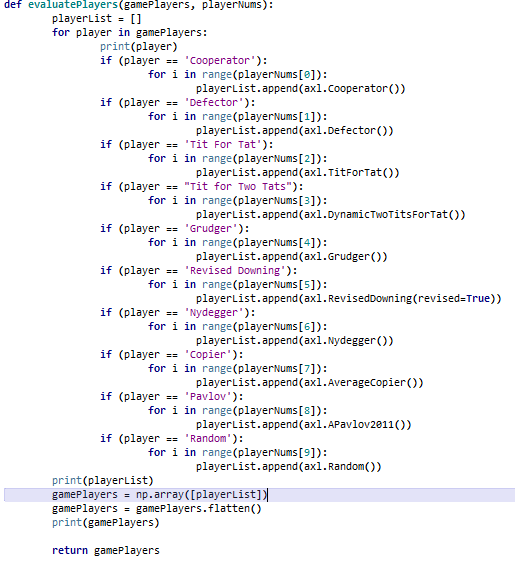


Figure 5: Adding players to the game

In Figure 5, we find a relatively simple function that looks for the 10 different player types that were specified in the UI, searching for string matches. The probability of an input error here is eliminated through use of a prefilled input. evaluatePlayers takes two lists, gamePlayers and playerNums. The first list, gamePlayers, consists of the 10 types of built in strategies while the second list, playerNums, takes the number of each type of player respectively. A new array, gamePlayers, has the number of each type of player added to it while the for loop works. After the loop is done, this multidimensional array is flattened to 1-D with the help of numPy, and the final list of players is returned.

This project uses Axelrod for strategies. Even though these aren’t part of the main program and are instead imported, studying these helps to understand what is going on at the end of the day. The cooperator strategy is this simple:



Figure 6: Cooperator Strategy

In comparison, by taking a look at Nydegger’s decision-making process, things are far more complicated with a dictionary and separate function to keep track of the score history in order to determine how to play each round:

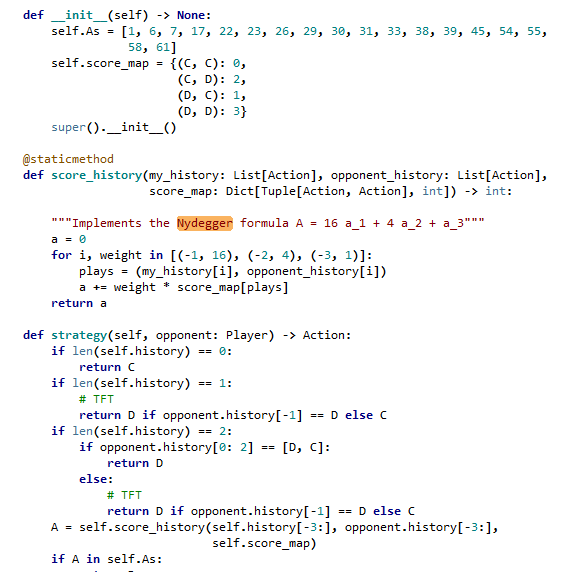


Figure 7: Nydegger strategy

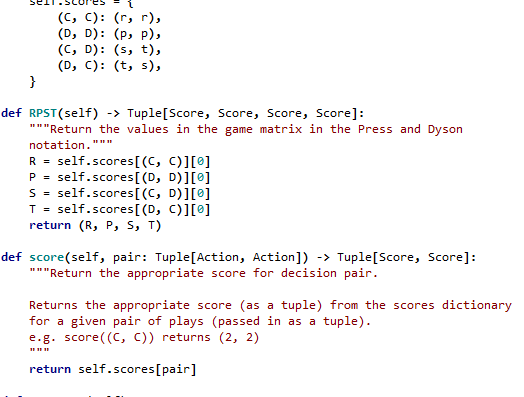


Figure 8: How a game is played between 2 participants

Understanding the process for how the three gameplay modes work is also important. Pictured below are the main mechanics of the tournament process, ecological process, and Moran process, respectively.

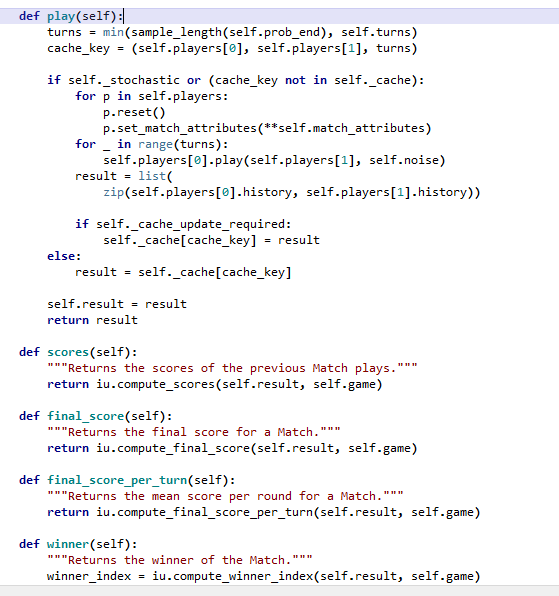


Figure 9: Tournament match

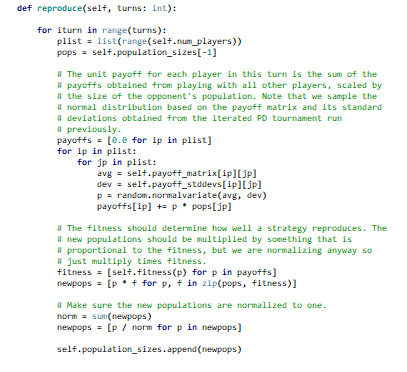


Figure 10: Ecological tournament

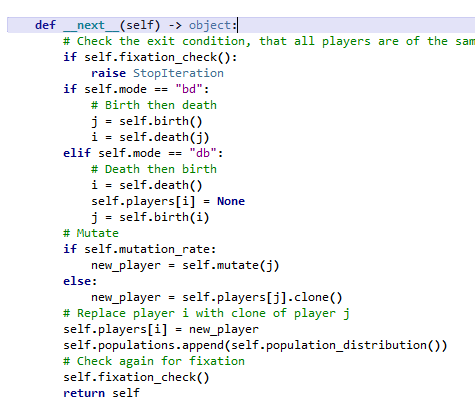


Figure 11: Moran birthing process

Moran is very computationally intensive. A Moran process stops not after a specified amount of generations like in Ecological, but only after all other types of participants are eliminated. Meanwhile, Ecological shows the sustainability of how the strategies “mesh” together over time, and depending on the simulation setup, may be very homogenous or very diverse. Tournament mode is a classic round robin tournament, with two sets of competitors playing a specified number of turns per game, and matches of multiple games, to find a winner. Moran is most susceptible to randomness, as it doesn’t stop until it finds a winner, so sometimes it must reach to get there by choosing one with .000001 points more than the next strategy. For this reason, results must be taken with a grain of salt in close matches, but as a tool for analyzing competitional proficiency, Moran is very helpful.

# Experiment Results

In order to test the integrity of results from ipd-sim, we must run several experiments to see if the simulator is emulating real-world results or not. We try this with different setups numerous times, and take the average of the results of each experiment type, in order to qualitatively analyze the normative data generated by ipd-sim in various scenarios.

## Cooperator vs Defector

The most basic of experiments, this experiment pits the two classic “dumb” strategies, cooperator versus defector, with various settings.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Game #1** | 1 C vs 1 D (seed) | **Game #2** | 1 C vs 1 D (no seed) | **Game #3** | 2 C vs 2 D |
| Seed set? | Yes (1) | Seed set? | No | Seed set? | No |
| **Setup** |  | **Setup** |  | **Setup** |  |
| Sim Length | 10 | Sim Length | 10 | Sim Length | 10 |
| Games per Generation | 10 | Games per Generation | 10 | Games per Generation | 10 |
| Noise | 0 | Noise | 0 | Noise | 0 |
| **Payoffs** |  | **Payoffs** |  | **Payoffs** |  |
| Temptation | 5 | Temptation | 5 | Temptation | 5 |
| Reward | 3 | Reward | 3 | Reward | 3 |
| Punishment | 0 | Punishment | 0 | Punishment | 0 |
| Sucker | -5 | Sucker | -5 | Sucker | -5 |
| **Participants** | | **Participants** | | **Participants** | |
| Cooperators | 1 | Cooperators | 1 | Cooperators | 2 |
| Defectors | 1 | Defectors | 1 | Defectors | 2 |
| **Total # individuals** | 2 | **Total # individuals** | 2 | **Total # individuals** | 4 |
| **Game Stats** | | **Game Stats** | | **Game Stats** | |
| Tournament Winner | Defector | Tournament Winner | Defector | Tournament Winner | Defector |
| Moran Winner | Defector | Moran Winner | Defector | Moran Winner | Defector |
| Eco Winner | Defector | Eco Winner | Defector | Eco Winner | Defector |
| Moran Winner by Iteration | 2 | Moran Winner by Iteration | 2 | Moran Winner by Iteration | 9 |
| Cooperator Median Score | -5 | Cooperator Median Score | -5 | Cooperator Median Score | -2.3 |
| Defector Median Score | 5 | Defector Median Score | 5 | Defector Median Score | 3.3 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Game #4** | 5 C vs 5 D | **Game #5** | 10 C vs 10 D |
| Seed set? | No | Seed set? | No |
| **Setup** |  | **Setup** |  |
| Sim Length | 10 | Sim Length | 10 |
| Games per Generation | 10 | Games per Generation | 10 |
| Noise | 0 | Noise | 0 |
| **Payoffs** |  | **Payoffs** |  |
| Temptation | 5 | Temptation | 5 |
| Reward | 3 | Reward | 3 |
| Punishment | 0 | Punishment | 0 |
| Sucker | -5 | Sucker | -5 |
| **Participants** | | **Participants** | |
| Cooperators | 5 | Cooperators | 10 |
| Defectors | 5 | Defectors | 10 |
| **Total # individuals** | 10 | **Total # individuals** | 20 |
| **Game Stats** | | **Game Stats** | |
| Tournament Winner | Defector | Tournament Winner | Defector |
| Moran Winner | Defector | Moran Winner | Defector |
| Eco Winner | Defector | Eco Winner | Defector |
| Moran Winner by Iteration | 20 | Moran Winner by Iteration | 30 |
| Cooperator Median Score | -1.44 | Cooperator Median Score | -5 |
| Defector Median Score | 2.77 | Defector Median Score | 5 |
|  |  |  |  |

in 3 scenarios, the defector class won. This is because cooperators are programmed strictly to cooperate, just like defectors strictly defect. The ever-trusting cooperator will lose miserably every time to the defector, who gains the temptation payoff, leaving the cooperator with the sucker payoff.

## Adding Tit for Tat

Here, we analyze how Tit for Tat behaves when we introduce it into matches with cooperators and defectors. Tit for Tat behaves predictably with both cooperators and defectors, cooperating on the very first round to test its opponent, and then defecting if and only if the last round, the opponent defected.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Game #1** | 1 C vs 1 TfT | | **Game #2** | | 1 D vs 1 TfT (no seed) | | **Game #3** | | Replay of Game #2 |
| Seed set? | No | | Seed set? | | No | | Seed set? | | No |
| **Setup** |  | | **Setup** | |  | | **Setup** | |  |
| Sim Length | 10 | | Sim Length | | 10 | | Sim Length | | 10 |
| Games per Generation | 10 | | Games per Generation | | 10 | | Games per Generation | | 10 |
| Noise | 0 | | Noise | | 0 | | Noise | | 0 |
| **Payoffs** |  | | **Payoffs** | |  | | **Payoffs** | |  |
| Temptation | 5 | | Temptation | | 5 | | Temptation | | 5 |
| Reward | 3 | | Reward | | 3 | | Reward | | 3 |
| Punishment | 0 | | Punishment | | 0 | | Punishment | | 0 |
| Sucker | -5 | | Sucker | | -5 | | Sucker | | -5 |
| **Participants** | | | **Participants** | | | | **Participants** | | |
| Cooperators | 1 | | Cooperators | | 0 | | Cooperators | | 0 |
| Defectors | 0 | | Defectors | | 1 | | Defectors | | 1 |
| Tit for Tat | 1 | | Tit for Tat | | 1 | | Tit for Tat | | 1 |
| **Total # individuals** | 2 | | **Total # individuals** | | 2 | | **Total # individuals** | | 2 |
| **Game Stats** | | | **Game Stats** | | | | **Game Stats** | | |
| Tournament Winner | Tit for Tat | | Tournament Winner | | Defector | | Tournament Winner | | Defector |
| Moran Winner | Tie | | Moran Winner | | Defector | | Moran Winner | | Defector |
| Eco Winner | Tie | | Eco Winner | | Defector | | Eco Winner | | Defector |
| Moran Winner by Iteration | 4 | | Moran Winner by Iteration | | 1 | | Moran Winner by Iteration | | 1 |
| Cooperator Median Score | 3 | | Tit for Tat | | -0.4 | | Tit for Tat | | -0.4 |
| Tit for Tat | 3 | | Defector Median Score | | 0.4 | | Defector Median Score | | 0.4 |
|  |  | |  | |  | |  | |  |
| **Game #4** | | C V D V TfT | | **Game #5** | | 10 C vs 10 D vs 10 TfT | |
|  | |  | |  | |  | |
| **Setup** | |  | | **Setup** | |  | |
| Sim Length | | 10 | | Sim Length | | 10 | |
| Games per Generation | | 10 | | Games per Generation | | 10 | |
| Noise | | 0 | | Noise | | 0 | |
| **Payoffs** | |  | | **Payoffs** | |  | |
| Temptation | | 5 | | Temptation | | 5 | |
| Reward | | 3 | | Reward | | 3 | |
| Punishment | | 0 | | Punishment | | 0 | |
| Sucker | | -5 | | Sucker | | -5 | |
| **Participants** | | | | **Participants** | | | |
| Cooperators | | 1 | | Cooperators | | 10 | |
| Defectors | | 1 | | Defectors | | 10 | |
| Tit for Tat | | 1 | | Tit for Tat | | 10 | |
| **Total # individuals** | | 3 | | **Total # individuals** | | 30 | |
| **Game Stats** | | | | **Game Stats** | | | |
| Tournament Winner | | Defector | | Tournament Winner | | Defector | |
| Moran Winner | | Defector | | Moran Winner | | Defector | |
| Eco Winner | | Tit for Tat | | Eco Winner | | - | |
| Moran Winner by Iteration | | 4 | | Moran Winner by Iteration | | 80 | |
| Cooperator Median Score | | 1 | | Cooperator Median Score | | 0.2413 | |
| Defector Median Score | | 2.75 | | Defector Median Score | | 1.8965 | |
| Tit for Tat | | 1.25 | | Tit for Tat | | 1.793 | |

## Result Analysis

In all 5 games, the Defector strategy wins the tournament mode, because Defector gets an extra move in on Tit for Tat before it has a chance to adapt its strategy.

## Intermediate Strategies

For these games, we introduce strategies that are still relatively simple, yet introduce interesting dynamics to the simulations. These include Adaptive Tit for Tat, Grudger, Random, and Average Copier. Adaptive Tit for Tat keeps a world variable that starts exactly at 0.5, between 0.0 and 1.0. As an opponent defects, the world variable becomes smaller. When it is below 0.5, Adaptive Tit for Tat defects. Otherwise, it cooperates. The Random strategy is exactly how it sounds, with the player randomly choosing between cooperating and defecting. The Grudger strategy starts by cooperating, and if an opponent defects just once, it will defect forever. The Average Copier strategy starts with a random cooperation/defection decision and then cooperates with the same probability as the opponent’s cooperation rate.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Game #1** | 1 TfT vs 1 TTfT | **Game #2** | C,D,TfT,TTfT | **Game #3** | C,D,TfT, TTfT, G |
| Seed set? | No | Seed set? | No | Seed set? | No |
| **Setup** |  | **Setup** |  | **Setup** |  |
| Sim Length | 10 | Sim Length | 10 | Sim Length | 10 |
| Games per Generation | 10 | Games per Generation | 10 | Games per Generation | 10 |
| Noise | 0 | Noise | 0 | Noise | 0 |
| **Payoffs** |  | **Payoffs** |  | **Payoffs** |  |
| Temptation | 5 | Temptation | 5 | Temptation | 5 |
| Reward | 3 | Reward | 3 | Reward | 3 |
| Punishment | 0 | Punishment | 1 | Punishment | 1 |
| Sucker | -5 | Sucker | 0 | Sucker | 0 |
| **Participants** | | **Participants** | | **Participants** | |
| Cooperators | 0 | Cooperators | 1 | Cooperators | 1 |
| Defectors | 0 | Defectors | 1 | Defectors | 1 |
| Tit for Tat | 1 | Tit for Tat | 1 | Tit for Tat | 1 |
| Tit for Two Tats | 1 | Tit for Two Tats | 1 | Tit for Two Tats | 1 |
| Grudger | 0 | Grudger | 0 | Grudger | 1 |
| **Total # individuals** | **2** | **Total # individuals** | **4** | **Total # individuals** | **5** |
| **Game Stats** | | **Game Stats** | | **Game Stats** | |
| Tournament Winner | Tie | Tournament Winner | Defector | Tournament Winner | TfT, 2TfT, Grudger |
| Moran Winner | Tit for Tat | Moran Winner | Tit for Tat | Moran Winner | Tit for Tat |
| Eco Winner | Tie | Eco Winner | - | Eco Winner | - |
| Moran Winner by Iteration | 1 | Moran Winner by Iteration | 8 | Moran Winner by Iteration | 60 |
| Two Tits for Tat | 3 | Cooperator Median Score | 0.2413 | Cooperator Median Score | 2.25 |
| Tit for Tat | 3 | Defector Median Score | 1.8965 | Defector Median Score | 2.3 |
|  |  | Tit for Tat | 1.793 | Tit for Tat | 2.475 |
|  |  | 2 Tit for Tat | 1.793 | 2 Tit for Tat | 2.475 |
|  |  |  |  | Grudger | 2.475 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Setup** |  | **Setup** |  |
| Sim Length | 10 | Sim Length | 10 |
| Games per Generation | 10 | Games per Generation | 10 |
| Noise | 0 | Noise | 0 |
| **Payoffs** |  | **Payoffs** |  |
| Temptation | 5 | Temptation | 5 |
| Reward | 3 | Reward | 3 |
| Punishment | 1 | Punishment | 1 |
| Sucker | 0 | Sucker | 0 |
| **Participants** | | **Participants** | |
| Cooperators | 0 | Cooperators | 10 |
| Defectors | 1 | Defectors | 10 |
| Tit for Tat | 0 | Tit for Tat | 10 |
| Tit for Two Tats | 0 | Tit for Two Tats | 10 |
| Grudger | 1 | Grudger | 10 |
| **Total # individuals** | **2** | **Total # individuals** | **50** |
| **Game Stats** | | **Game Stats** | |
| Tournament Winner | Defector | Tournament Winner | Defector |
| Moran Winner | Defector | Moran Winner | Tit For Tat |
| Eco Winner | Defector | Eco Winner | Grudger |
| Moran Winner by Iteration | 1 | Moran Winner by Iteration | 80 |
| Cooperator Median Score | | Cooperator Median Score | 0.2413 |
| Defector Median Score | 1.4 | Defector Median Score | 1.8965 |
| Tit for Tat |  | Tit for Tat | 1.793 |
| 2 Tit for Tat | | 2 Tit for Tat | |
| Grudger | 0.9 | Grudger |  |

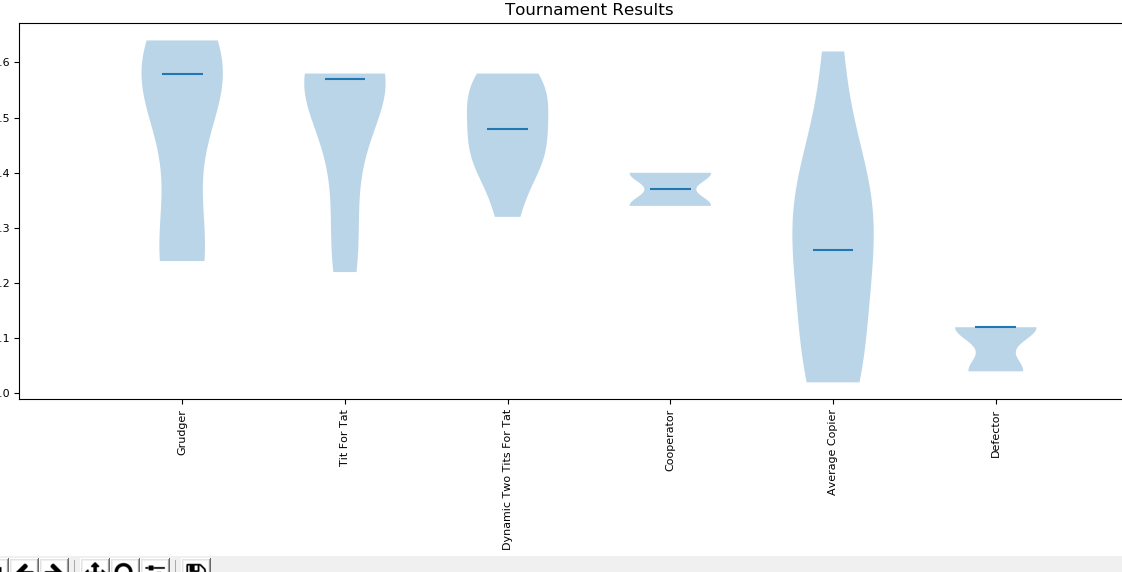


Figure 12: Tournament results by class median score. The blue highlight represents the maximum and minimum scores attained over the course of the Tournament

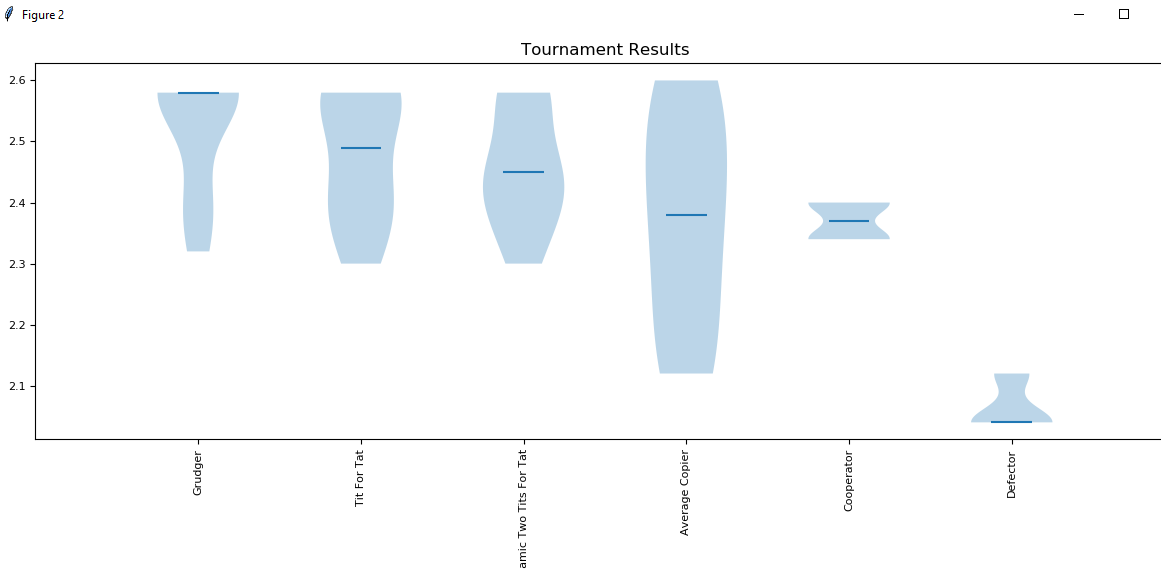


Figure 13: Same tournament played as in Figure 12. Noticeably, the randomness in the Copier's starting decision cause others to use different strategies, resulting in slightly different results.

## Result Analysis

These new strategies, no matter how simplistic, add interesting dynamics to the results. Grudger comes out on top because it cooperates when played nicely with but is unrelenting against aggressive strategies. The decision of the Average Copier to defect or cooperate against Grudger is important, as Grudger will unrelentingly defect as well, resulting in infinite defection between the two classes.

Of note, we can see a trend that the Defector starts to do worse when surrounded by larger opponent sets. These cooperative opponents behave with each other, so the importance of the Defector and its matches resulting in negative scores in kinder types is minimized. This is akin to real life, in a way. It’s generally accepted that most human beings are good, or “cooperative”. That’s not to say that they are naïve and would behave as the Cooperator class does, but they are not out to steal and lie at all times, like the Defector does. Conversely, all bad actors don’t necessarily act like a Defector class does and steal or lie at all times, or else how would they get anyone to trust you? This is a good segue into the next experiment set, where we introduce the last set of strategies included in ipd-sim. These are more advanced strategies, designed to more accurately emulate human thinking.

# Data

The information generated is more or less entirely dependent on the input parameters, so there is no need for a database. However, the output data is saved in a CSV file that lets the users see the results of previous experiments for which they can use for their own various purposes.https://axelrod.readthedocs.io/en/stable/