THE PRISONER'S DILEMMA

WITH REACTIVE NOISE

BY JENNA JORDAN (MSLIS)

See the code at:

https://github.com/ jenna-jordan/ Prisoners-Dilemma





Payoffs

Lose

Win much

^

Lose much

When you betray while the other party cooperates

When you betray, and so does the other party

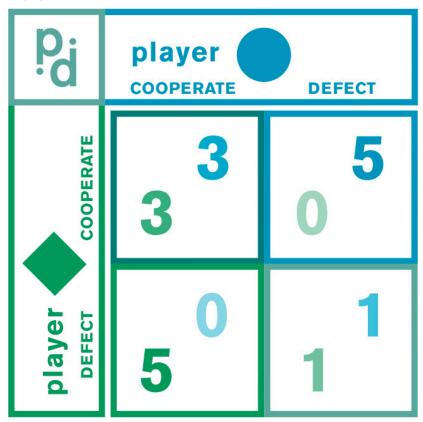
When you cooperate, and so does the other party

When you cooperate, but the other party betrays

WHAT IS THE

PRISONER'S DILEMMA?

- a classic cooperation game in Game Theory
- 2 players, each with choice to defect or cooperate
- Reward Matrix:



- Iterated Prisoner's Dilemma: repeated rounds, with each player knowing what the other played in the previous round. Players do not know how many rounds there will be per game
- Tournament style: each player plays against every other player - the player with the most points wins

PROGRAM STRUCTURE:

CLASSES

Player

- keeps track of tournament-level stats (wins, losses, ties, points) and game-level info (history)
- each Player has a Strategy

Strategy

• abstract base class Strategy, with one subclass for each specific strategy (e.g. Tit For Tat)

Game

- each game is between two players, with X rounds and randomized noise parameters
- keeps track of game-level stats for the overall game (e.g. history) and for each player (e.g. points)
- methods to change the starting noise level, how payoffs are determined, and the payoff values
- primary methods:
 - play_round() plays through 1 round
 - send_history() sends perceived round history to the player, which will be used by the strategy
 - play_game() plays through the entire game,
 executes play_round() and send_history() X times

INTRODUCING NOISE: THE RANDOM FACTOR

- Noisy Iterated Prisoner's Dilemma: IPD tournament with some level of environmental noise
- Noise = the chance that a player's move (to cooperate or defect) will get switched.

REACTIVE NOISE

- Normally, noise throughout the game is static.
- In my version, noise is reactive it increases as players defect, and decreases as players cooperate
- Payoffs can be determined by either:
 - player's actual moves (misperception) OR
 - player's perceived moves (misimplementation)
- I add randomness when implementing the noise
 - starting noise: random # between 0 and .5
 - noise increment: random # between 0 and .01
 - maximum noise: random # between 0 and .5

Let's see the code!

```
p1chance = random.random() # a value between 0 and 1
p2chance = random.random() # a different value between 0 and 1
if realMoves is ('D', 'D'):
    noiseIncrementor = random.uniform(0, self.noise_growthMax * 2) # more noise for more
elif 'D' in realMoves:
    noiseIncrementor = random.uniform(0, self.noise_growthMax) # if only one player defe
else:
    noiseIncrementor = (random.uniform(0, self.noise_growthMax)) * -1 # if both cooperat
self.noise += noiseIncrementor
if self.noise > self.noiseMax:
    self.noise = self.noiseMax
elif self.noise < 0:</pre>
    self.noise = 0
    self.noise = self.noise
if p1chance < self.noise:</pre>
    p1PerMove = self.flip(p1move)
    p1PerMove = p1move
if p2chance < self.noise:</pre>
    p2PerMove = self.flip(p2move)
    p2PerMove = p2move
noisyMoves = (p1PerMove, p2PerMove)
self.gameHistory.append(noisyMoves)
T, R, P, S = self.payoffs
if self.implementNoise:
   moves = noisyMoves
   moves = realMoves
if moves == ('C', 'C'):
   p1payoff = R
   p2payoff = R
elif moves == ('D', 'D'):
   p1payoff = P
   p2payoff = P
elif moves == ('C', 'D'):
   p1payoff = S
   p2payoff = T
elif moves == ('D', 'C'):
   p1payoff = T
   p2payoff = S
    raise Exception("Invalid move(s) made, choose 'C' or 'D'.")
self.p1Score += p1payoff
self.p2Score += p2payoff
```