

THE PRISONER'S DILEMMA

WITH REACTIVE NOISE

BY JENNA JORDAN
(MSLIS)

See the code at:

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



Payoffs

Win much	Win	Lose	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:

p.d		player 	
		COOPERATE	DEFECT
 player	COOPERATE	3 3	0 5
	DEFECT	5 0	1 1

- 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!

Game Class: play_round() method

```
# generate random value for each player to be compared to the noise
p1chance = random.random() # a value between 0 and 1
p2chance = random.random() # a different value between 0 and 1

# increase noise if defection occurred
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

# increment noise, make sure noise stays between 0 and max
self.noise += noiseIncrementor
if self.noise > self.noiseMax:
    self.noise = self.noiseMax
elif self.noise < 0:
    self.noise = 0
else:
    self.noise = self.noise

# use noise to possibly flip moves
if p1chance < self.noise:
    p1PerMove = self.flip(p1move)
else:
    p1PerMove = p1move

if p2chance < self.noise:
    p2PerMove = self.flip(p2move)
else:
    p2PerMove = p2move

# moves (potentially altered by noise) that will determine payoffs
noisyMoves = (p1PerMove, p2PerMove)
self.gameHistory.append(noisyMoves)

# get payoff values, either default or those specified when set_payoffs() is called
T, R, P, S = self.payoffs

# use game mode to determine if real moves or noisy moves are used to award payoffs
if self.implementNoise:
    moves = noisyMoves
else:
    moves = realMoves

# determine payoff values, to be added to the player's real scores
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
else:
    raise Exception("Invalid move(s) made, choose 'C' or 'D'.")

self.p1Score += p1payoff
self.p2Score += p2payoff
```