

```
In [3]: import random
from scipy.stats import norm
import matplotlib.pyplot as plt
from scipy.stats import beta
import numpy as np
```

```
In [4]: def tit_for_tat(num_trials):

    player_a_moves = ["C"]
    player_b_moves = ["C"]

    for _ in range(num_trials - 1):

        if player_b_moves[-1] == "C":
            player_a_moves.append("C")
        else:
            player_a_moves.append("D")

        opponent_response = random.choice(["C", "D"])
        player_b_moves.append(opponent_response)

    return player_a_moves, player_b_moves

def tit_for_tat_with_threshold(num_rounds, threshold):
    actions_a = []
    actions_b = []

    for round_num in range(1, num_rounds+1):
        if round_num <= threshold:
            actions_a.append("C")
            actions_b.append(random.choice(["C", "D"]))
        else:
            if actions_b[-1] == "C":
                actions_a.append("C")
            else:
                actions_a.append("D")
            opponent_response = random.choice(["C", "D"])
            actions_b.append(opponent_response)

    return actions_a, actions_b

def one_cooperating_strategy(num_trials):
    player_d_moves=[]
    player_c_moves=[]
    for _ in range(num_trials):

        player_d_moves.append(random.choice(["C", "D"]))
        player_c_moves.append("C")
    return player_d_moves, player_c_moves

def both_cooperating_strategy(num_trials):
    player_e_moves=[]
    player_f_moves=[]
    for _ in range(num_trials):
```

```

        player_e_moves.append("C")
        player_f_moves.append("C")
    return player_e_moves, player_f_moves

"""payoff_matrix = {
    "CC": (11, 11), # Payoff for mutual cooperation
    "CD": (0, 20), # Payoff for A: Cooperation, B: Defection
    "DC": (20, 0), # Payoff for A: Defection, B: Cooperation
    "DD": (9, 9)   # Payoff for mutual defection
}"""
payoff_matrix = {'CC': (11, 11), 'CD': (0, 20), 'DC': (20, 0), 'DD': (9, 9)

# Simulate trials
num_trials = 300
player_a, player_b = tit_for_tat(num_trials)
player_a1, player_b1 = tit_for_tat_with_threshold(num_trials, threshold=2)
player_a2, player_b2 = one_cooperating_strategy(num_trials)
player_a3, player_b3 = both_cooperating_strategy(num_trials)

mu3 = 4
mu4 = 0
sigma = 1
pdf_values_AB=[]
pdf_values_BA=[]
pdf_values1=[]
pdf_values2=[]
pdf_values3=[]

#print(f"Value of A: {player_a}, Value of y: {player_b}")
print("Strategy: Tit-for-Tat")
for round_num, (move_a, move_b) in enumerate(zip(player_a, player_b), start=1):
    print(f"Round {round_num}: Player A moves {move_a}, Player B moves {move_b}")

print("\nStrategy: Tit-for-Tat with Threshold")
for round_num, (move_a, move_b) in enumerate(zip(player_a1, player_b1), start=1):
    print(f"Round {round_num}: Player A moves {move_a}, Player B moves {move_b}")

print("\nStrategy: One Cooperating Strategy")
for round_num, (move_a, move_b) in enumerate(zip(player_a2, player_b2), start=1):
    print(f"Round {round_num}: Player A moves {move_a}, Player B moves {move_b}")

print("\nStrategy: Both Cooperating Strategy")
for round_num, (move_a, move_b) in enumerate(zip(player_a3, player_b3), start=1):
    print(f"Round {round_num}: Player A moves {move_a}, Player B moves {move_b}")

```

Strategy: Tit-for-Tat

Round 1: Player A moves C, Player B moves C
Round 2: Player A moves C, Player B moves C
Round 3: Player A moves C, Player B moves D
Round 4: Player A moves D, Player B moves C
Round 5: Player A moves C, Player B moves C
Round 6: Player A moves C, Player B moves D
Round 7: Player A moves D, Player B moves D
Round 8: Player A moves D, Player B moves D
Round 9: Player A moves D, Player B moves D
Round 10: Player A moves D, Player B moves D
Round 11: Player A moves D, Player B moves C
Round 12: Player A moves C, Player B moves D
Round 13: Player A moves D, Player B moves C
Round 14: Player A moves C, Player B moves D
Round 15: Player A moves D, Player B moves D
Round 16: Player A moves D, Player B moves D
Round 17: Player A moves D, Player B moves D
Round 18: Player A moves D, Player B moves D
Round 19: Player A moves D, Player B moves C
Round 20: Player A moves C, Player B moves D
Round 21: Player A moves D, Player B moves C
Round 22: Player A moves C, Player B moves C
Round 23: Player A moves C, Player B moves D
Round 24: Player A moves D, Player B moves C
Round 25: Player A moves C, Player B moves D
Round 26: Player A moves D, Player B moves C
Round 27: Player A moves C, Player B moves C
Round 28: Player A moves C, Player B moves C
Round 29: Player A moves C, Player B moves D
Round 30: Player A moves D, Player B moves C
Round 31: Player A moves C, Player B moves C
Round 32: Player A moves C, Player B moves D
Round 33: Player A moves D, Player B moves D
Round 34: Player A moves D, Player B moves C
Round 35: Player A moves C, Player B moves D
Round 36: Player A moves D, Player B moves D
Round 37: Player A moves D, Player B moves D
Round 38: Player A moves D, Player B moves D
Round 39: Player A moves D, Player B moves C
Round 40: Player A moves C, Player B moves D
Round 41: Player A moves D, Player B moves C
Round 42: Player A moves C, Player B moves C
Round 43: Player A moves C, Player B moves C
Round 44: Player A moves C, Player B moves C
Round 45: Player A moves C, Player B moves D
Round 46: Player A moves D, Player B moves C
Round 47: Player A moves C, Player B moves D
Round 48: Player A moves D, Player B moves C
Round 49: Player A moves C, Player B moves C
Round 50: Player A moves C, Player B moves C
Round 51: Player A moves C, Player B moves C
Round 52: Player A moves C, Player B moves D
Round 53: Player A moves D, Player B moves D
Round 54: Player A moves D, Player B moves D
Round 55: Player A moves D, Player B moves C
Round 56: Player A moves C, Player B moves D
Round 57: Player A moves D, Player B moves C
Round 58: Player A moves C, Player B moves D
Round 59: Player A moves D, Player B moves C

[illegible]

[illegible]

[illegible]

[illegible]

Round 300: Player A moves D, Player B moves C

Strategy: Tit-for-Tat with Threshold

Round 1: Player A moves C, Player B moves C
Round 2: Player A moves C, Player B moves D
Round 3: Player A moves C, Player B moves D
Round 4: Player A moves C, Player B moves C
Round 5: Player A moves C, Player B moves C
Round 6: Player A moves C, Player B moves D
Round 7: Player A moves C, Player B moves C
Round 8: Player A moves C, Player B moves D
Round 9: Player A moves C, Player B moves C
Round 10: Player A moves C, Player B moves C
Round 11: Player A moves C, Player B moves C
Round 12: Player A moves C, Player B moves C
Round 13: Player A moves C, Player B moves D
Round 14: Player A moves C, Player B moves C
Round 15: Player A moves C, Player B moves C
Round 16: Player A moves C, Player B moves D
Round 17: Player A moves C, Player B moves D
Round 18: Player A moves C, Player B moves C
Round 19: Player A moves C, Player B moves D
Round 20: Player A moves C, Player B moves D
Round 21: Player A moves C, Player B moves C
Round 22: Player A moves D, Player B moves D
Round 23: Player A moves C, Player B moves D
Round 24: Player A moves D, Player B moves C
Round 25: Player A moves D, Player B moves C
Round 26: Player A moves D, Player B moves C
Round 27: Player A moves C, Player B moves D
Round 28: Player A moves C, Player B moves C
Round 29: Player A moves C, Player B moves C
Round 30: Player A moves C, Player B moves C
Round 31: Player A moves C, Player B moves C
Round 32: Player A moves C, Player B moves D
Round 33: Player A moves C, Player B moves C
Round 34: Player A moves C, Player B moves D
Round 35: Player A moves D, Player B moves C
Round 36: Player A moves D, Player B moves C
Round 37: Player A moves D, Player B moves C
Round 38: Player A moves C, Player B moves C
Round 39: Player A moves D, Player B moves D
Round 40: Player A moves D, Player B moves C
Round 41: Player A moves D, Player B moves D
Round 42: Player A moves C, Player B moves C
Round 43: Player A moves C, Player B moves D
Round 44: Player A moves D, Player B moves D
Round 45: Player A moves D, Player B moves D
Round 46: Player A moves D, Player B moves C
Round 47: Player A moves C, Player B moves C
Round 48: Player A moves D, Player B moves C
Round 49: Player A moves C, Player B moves C
Round 50: Player A moves C, Player B moves C
Round 51: Player A moves C, Player B moves C
Round 52: Player A moves C, Player B moves C
Round 53: Player A moves C, Player B moves C
Round 54: Player A moves C, Player B moves D
Round 55: Player A moves C, Player B moves D
Round 56: Player A moves C, Player B moves D
Round 57: Player A moves C, Player B moves C

[illegible]

[illegible]

[illegible]

[illegible]

Round 298: Player A moves D, Player B moves C
Round 299: Player A moves C, Player B moves C
Round 300: Player A moves C, Player B moves C

Strategy: One Cooperating Strategy

Round 1: Player A moves C, Player B moves C
Round 2: Player A moves D, Player B moves C
Round 3: Player A moves D, Player B moves C
Round 4: Player A moves C, Player B moves C
Round 5: Player A moves C, Player B moves C
Round 6: Player A moves D, Player B moves C
Round 7: Player A moves D, Player B moves C
Round 8: Player A moves C, Player B moves C
Round 9: Player A moves D, Player B moves C
Round 10: Player A moves C, Player B moves C
Round 11: Player A moves D, Player B moves C
Round 12: Player A moves C, Player B moves C
Round 13: Player A moves C, Player B moves C
Round 14: Player A moves D, Player B moves C
Round 15: Player A moves D, Player B moves C
Round 16: Player A moves D, Player B moves C
Round 17: Player A moves D, Player B moves C
Round 18: Player A moves C, Player B moves C
Round 19: Player A moves D, Player B moves C
Round 20: Player A moves C, Player B moves C
Round 21: Player A moves C, Player B moves C
Round 22: Player A moves C, Player B moves C
Round 23: Player A moves D, Player B moves C
Round 24: Player A moves C, Player B moves C
Round 25: Player A moves D, Player B moves C
Round 26: Player A moves D, Player B moves C
Round 27: Player A moves C, Player B moves C
Round 28: Player A moves C, Player B moves C
Round 29: Player A moves D, Player B moves C
Round 30: Player A moves D, Player B moves C
Round 31: Player A moves D, Player B moves C
Round 32: Player A moves C, Player B moves C
Round 33: Player A moves D, Player B moves C
Round 34: Player A moves C, Player B moves C
Round 35: Player A moves D, Player B moves C
Round 36: Player A moves D, Player B moves C
Round 37: Player A moves C, Player B moves C
Round 38: Player A moves C, Player B moves C
Round 39: Player A moves D, Player B moves C
Round 40: Player A moves C, Player B moves C
Round 41: Player A moves D, Player B moves C
Round 42: Player A moves C, Player B moves C
Round 43: Player A moves C, Player B moves C
Round 44: Player A moves D, Player B moves C
Round 45: Player A moves D, Player B moves C
Round 46: Player A moves C, Player B moves C
Round 47: Player A moves C, Player B moves C
Round 48: Player A moves D, Player B moves C
Round 49: Player A moves C, Player B moves C
Round 50: Player A moves D, Player B moves C
Round 51: Player A moves D, Player B moves C
Round 52: Player A moves C, Player B moves C
Round 53: Player A moves C, Player B moves C
Round 54: Player A moves D, Player B moves C
Round 55: Player A moves C, Player B moves C

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Round 294: Player A moves C, Player B moves C
Round 295: Player A moves C, Player B moves C
Round 296: Player A moves C, Player B moves C
Round 297: Player A moves C, Player B moves C
Round 298: Player A moves C, Player B moves C
Round 299: Player A moves C, Player B moves C
Round 300: Player A moves C, Player B moves C

```
In [6]: import math
import numpy as np
import matplotlib.pyplot as plt

def memory_strength(initial_strength, decay_constant, time):
    """
    Calculate memory strength at a given time using an exponential decay

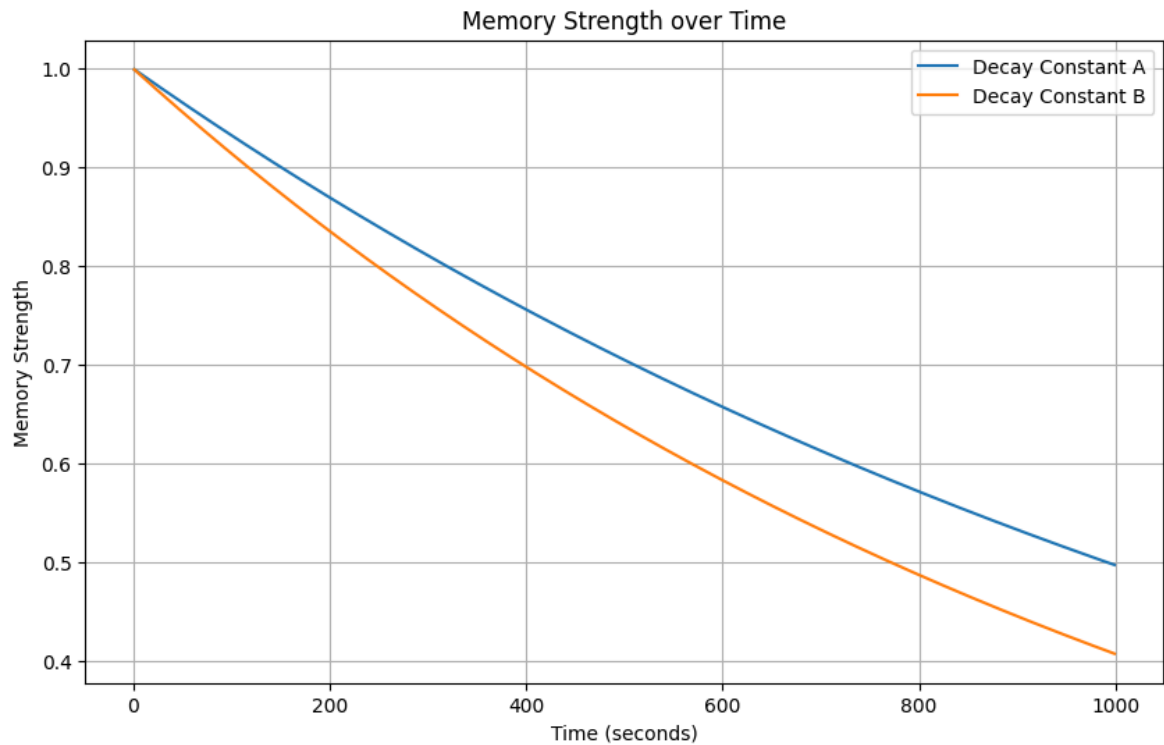
    Parameters:
        initial_strength (float): Initial strength of memory.
        decay_constant (float): Decay constant determining the rate of de
        time (float): Time elapsed since memory formation.

    Returns:
        float: Memory strength at the given time.
    """
    return initial_strength * math.exp(-decay_constant * time)

# Define parameters
initial_strength = 1.0 # Initial strength of memory
decay_constant_A = 0.0007 # Decay constant
decay_constant_B = 0.0009
# Time range from 0 to 100 seconds with 0.1 second intervals
time_range = np.arange(1, 1000, 1)

# Calculate memory strength at each time point
memory_strength_values_A = [memory_strength(initial_strength, decay_const
# print(memory_strength_values)
# Calculate memory strength at each time point
memory_strength_values_B = [memory_strength(initial_strength, decay_const

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time_range, memory_strength_values_A, label='Decay Constant A')
plt.plot(time_range, memory_strength_values_B, label='Decay Constant B')
plt.title('Memory Strength over Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Memory Strength')
plt.legend()
plt.grid(True)
plt.show()
```



```
In [7]: # For the tit-for-tat
payoffs_a = []
w_payoffs_a = []
payoffs_b = []
w_payoffs_b = []
sum_rep_AB=0
sum_rep_BA=0
#A has for B=AB
trust_values_tit_tat_AB = []
trust_values_tit_tat_BA = []
rep_values_AB=[]
rep_values_BA=[]
for i in range(len(player_a)):
    action_a = player_a[i]
    action_b = player_b[i]
    key = action_a + action_b
    payoffs_a.append(payload_matrix[key][0])
    payoffs_b.append(payload_matrix[key][1])
    for j in range(len(payoffs_a)):
        w_payoffs_a.append( memory_strength_values_A[j]*payoffs_a[len(pay
        w_payoffs_b.append( memory_strength_values_B[j]*payoffs_b[len(pay
    if (sum(w_payoffs_a)+sum(w_payoffs_b)) == 0:
        rep_values_AB.append(0.4) # Set trust value to a default value
        rep_values_BA.append(0.4)
    else:
        rep_values_AB.append(sum(w_payoffs_a) / (sum(w_payoffs_a)+sum(w_p
        rep_values_BA.append(sum(w_payoffs_b) / (sum(w_payoffs_a)+sum(w_p
    w_payoffs_a.clear()
    w_payoffs_b.clear()

    for k in range(len(rep_values_AB)):
        sum_rep_AB+=rep_values_AB[k]
        sum_rep_BA+=rep_values_BA[k]
    trust_values_tit_tat_AB.append(sum_rep_AB/(k+1))
    trust_values_tit_tat_BA.append(sum_rep_BA/(k+1))
```



```

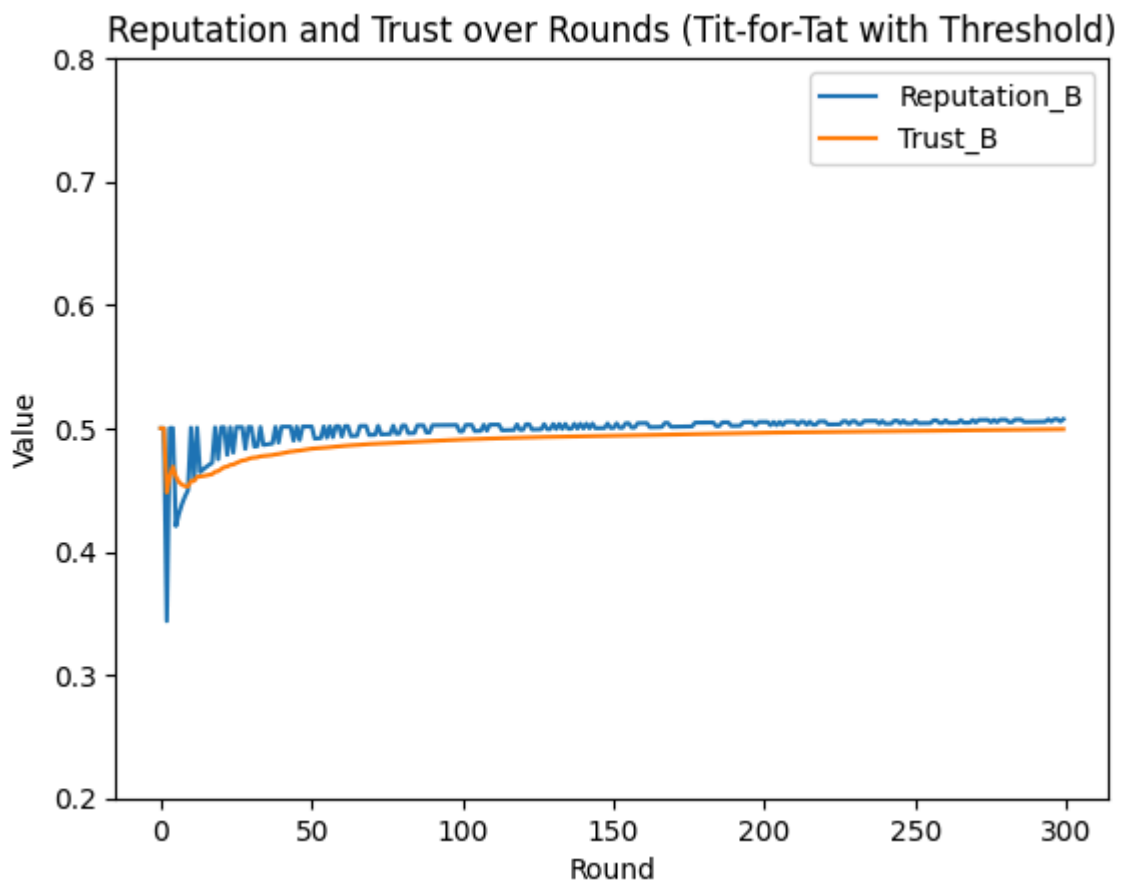
sum_rep_AB=0
sum_rep_BA=0

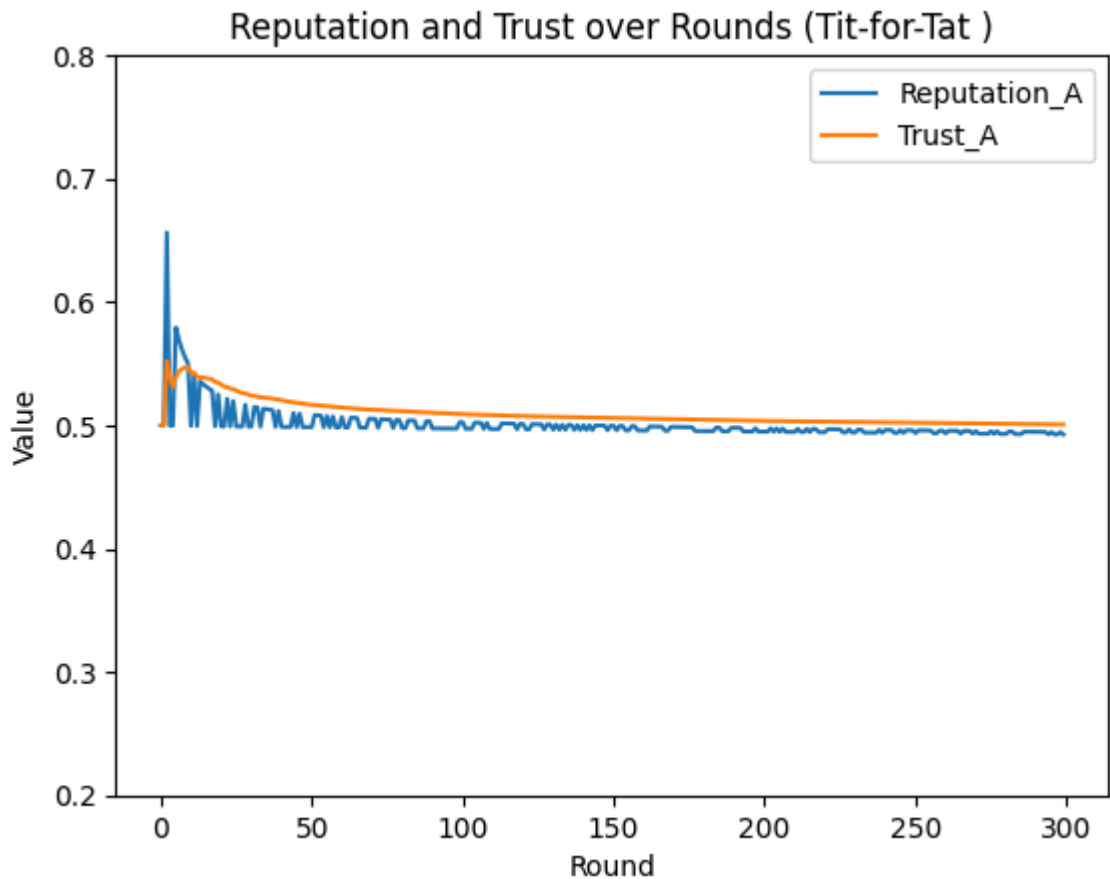
#print(trust_values_tit_tat_BA)
#print(trust_values_tit_tat_AB)

# Plot Rep values and Trust values for tit-for-tat FOR AB
plt.plot(range(len(rep_values_AB)), rep_values_AB, label='Reputation_B')
plt.plot(range(len(trust_values_tit_tat_AB)), trust_values_tit_tat_AB, la
plt.xlabel('Round')
plt.ylabel('Value')
plt.title('Reputation and Trust over Rounds (Tit-for-Tat with Threshold)')
plt.legend()
plt.ylim(0.2, 0.8) # Set y-axis limits
plt.show()

# Plot Rep values and Trust values for tit-for-tat FOR BA
plt.plot(range(len(rep_values_BA)), rep_values_BA, label='Reputation_A')
plt.plot(range(len(trust_values_tit_tat_BA)), trust_values_tit_tat_BA, la
plt.xlabel('Round')
plt.ylabel('Value')
plt.title('Reputation and Trust over Rounds (Tit-for-Tat )')
plt.legend()
plt.ylim(0.2, 0.8) # Set y-axis limits
plt.show()

```





```
In [8]: # For the tit-for-tat with threshold strategy
payoffs_a1 = []
w_payoffs_a1 = []
payoffs_b1 = []
w_payoffs_b1 = []
sum_rep_AB1=0
sum_rep_BA1=0
#A has for B=AB
trust_values_tit_tat_AB1 = []
trust_values_tit_tat_BA1= []
rep_values_AB1=[]
rep_values_BA1=[]
for i in range(len(player_a)):
    action_a1 = player_a1[i]
    action_b1 = player_b1[i]
    key = action_a1 + action_b1
    payoffs_a1.append(payload_matrix[key][0])
    payoffs_b1.append(payload_matrix[key][1])
    for j in range(len(payoffs_a1)):
        w_payoffs_a1.append( memory_strength_values_A[j]*payoffs_a1[len(p
        w_payoffs_b1.append( memory_strength_values_B[j]*payoffs_b1[len(p
    if (sum(w_payoffs_a1)+sum(w_payoffs_b1)) == 0:
        rep_values_AB1.append(0.4) # Set trust value to a default value
        rep_values_BA1.append(0.4)
    else:
        rep_values_AB1.append(sum(w_payoffs_a1) / (sum(w_payoffs_a1)+sum(
        rep_values_BA1.append(sum(w_payoffs_b1) / (sum(w_payoffs_a1)+sum(
    w_payoffs_a1.clear()
    w_payoffs_b1.clear()
    for k in range(len(rep_values_AB1)):
        sum_rep_AB1+=rep_values_AB1[k]
        sum_rep_BA1+=rep_values_BA1[k]
```

```

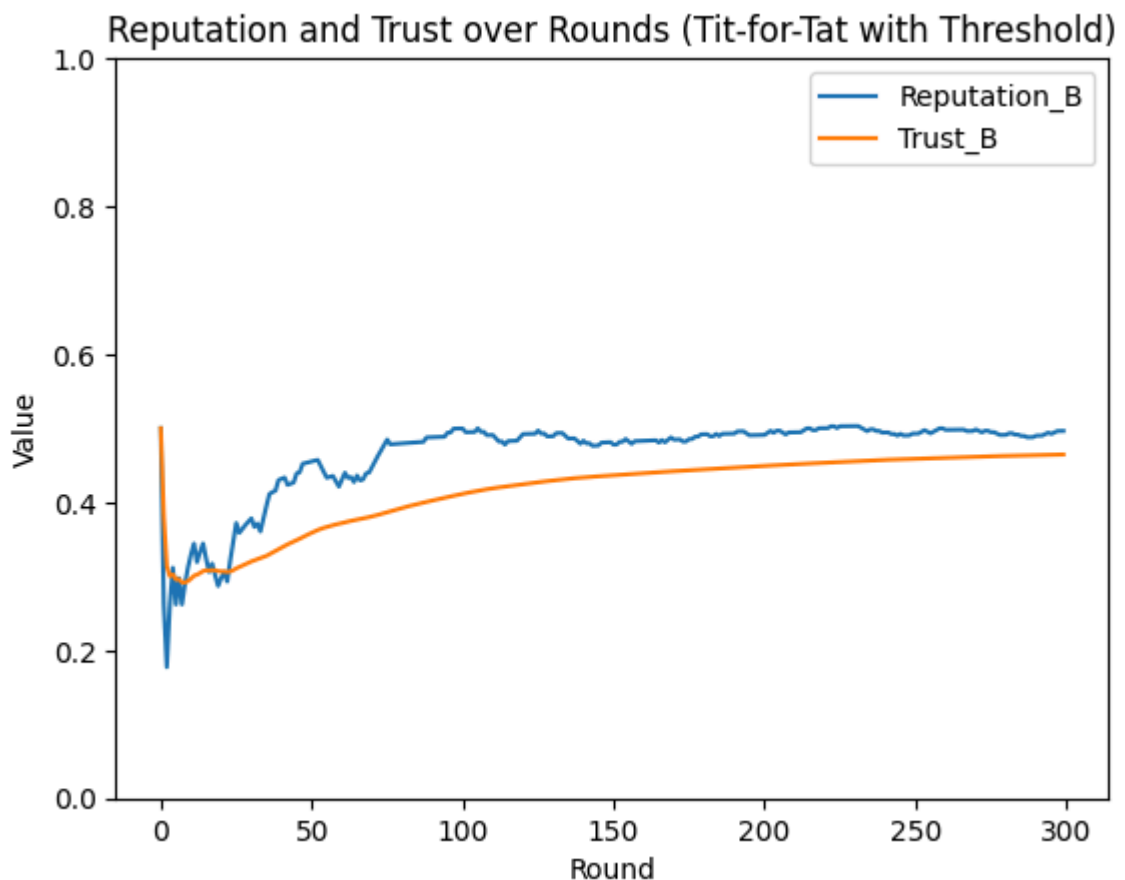
trust_values_tit_tat_AB1.append(sum_rep_AB1/(k+1))
trust_values_tit_tat_BA1.append(sum_rep_BA1/(k+1))
sum_rep_AB1=0
sum_rep_BA1=0

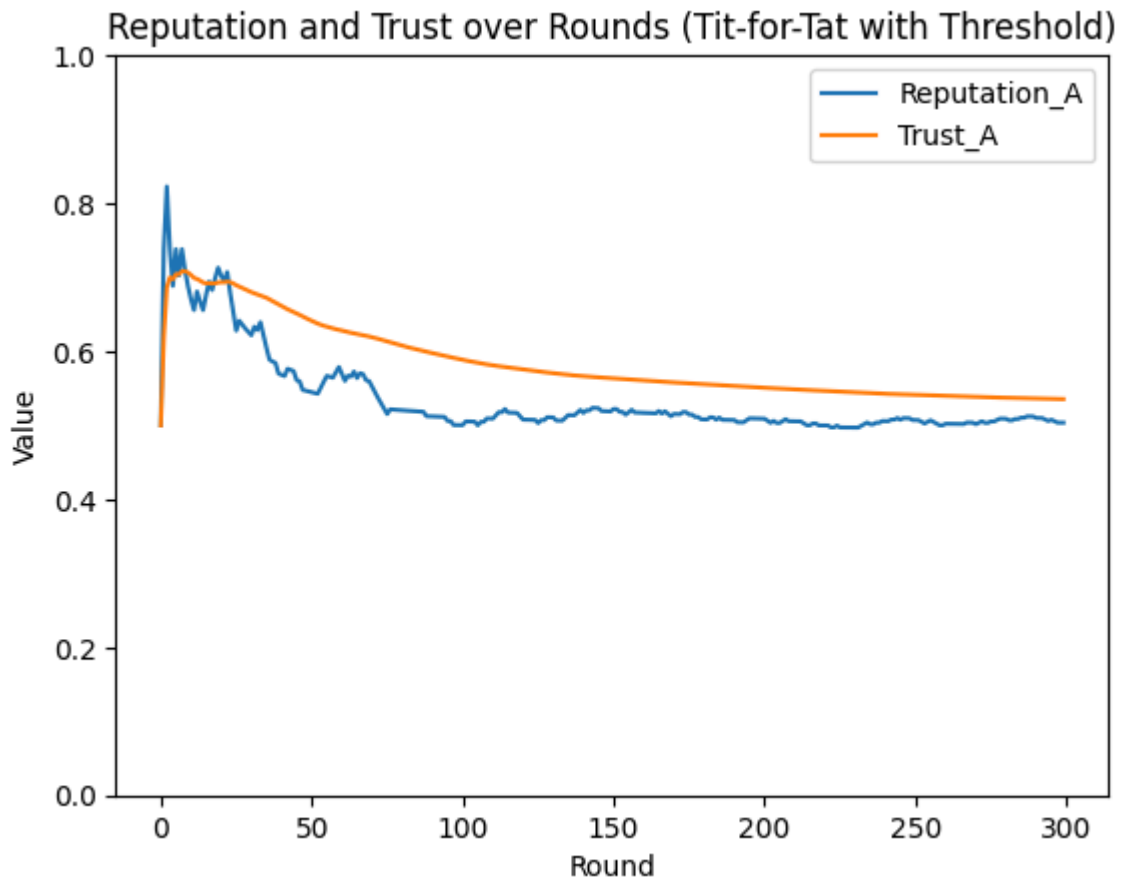
#print(trust_values_tit_tat_BA1)
#print(trust_values_tit_tat_AB1)

# Plot Rep values and Trust values for tit-for-tat FOR AB
plt.plot(range(len(rep_values_AB1)), rep_values_AB1, label='Reputation_B')
plt.plot(range(len(trust_values_tit_tat_AB1)), trust_values_tit_tat_AB1,
plt.xlabel('Round')
plt.ylabel('Value')
plt.title('Reputation and Trust over Rounds (Tit-for-Tat with Threshold)')
plt.legend()
plt.ylim(0, 1) # Set y-axis limits
plt.show()

# Plot Rep values and Trust values for tit-for-tat FOR BA
plt.plot(range(len(rep_values_BA1)), rep_values_BA1, label='Reputation_A')
plt.plot(range(len(trust_values_tit_tat_BA1)), trust_values_tit_tat_BA1,
plt.xlabel('Round')
plt.ylabel('Value')
plt.title('Reputation and Trust over Rounds (Tit-for-Tat with Threshold)')
plt.legend()
plt.ylim(0, 1) # Set y-axis limits
plt.show()

```





```
In [9]: # For one ever co-operating
payoffs_a2 = []
w_payoffs_a2 = []
payoffs_b2 = []
w_payoffs_b2 = []
sum_rep_AB2=0
sum_rep_BA2=0
#A has for B=AB
trust_values_tit_tat_AB2 = []
trust_values_tit_tat_BA2= []
rep_values_AB2=[]
rep_values_BA2=[]
for i in range(len(player_a2)):
    action_a2 = player_a2[i]
    action_b2 = player_b2[i]
    key = action_a2 + action_b2
    payoffs_a2.append(payload_matrix[key][0])
    payoffs_b2.append(payload_matrix[key][1])
    for j in range(len(payoffs_a2)):
        w_payoffs_a2.append( memory_strength_values_A[j]*payoffs_a2[len(p
        w_payoffs_b2.append( memory_strength_values_B[j]*payoffs_b2[len(p
    if (sum(w_payoffs_a2)+sum(w_payoffs_b2)) == 0:
        rep_values_AB2.append(0.4) # Set trust value to a default value
        rep_values_BA2.append(0.4)
    else:
        rep_values_AB2.append(sum(w_payoffs_a2) / (sum(w_payoffs_a2)+sum(
        rep_values_BA2.append(sum(w_payoffs_b2) / (sum(w_payoffs_a2)+sum(
    w_payoffs_a2.clear()
    w_payoffs_b2.clear()
    for k in range(len(rep_values_AB2)):
        sum_rep_AB2+=rep_values_AB2[k]
        sum_rep_BA2+=rep_values_BA2[k]
```

```

trust_values_tit_tat_AB2.append(sum_rep_AB2/(k+1))
trust_values_tit_tat_BA2.append(sum_rep_BA2/(k+1))
sum_rep_AB2=0
sum_rep_BA2=0

#print(trust_values_tit_tat_BA2)
print(trust_values_tit_tat_AB2)

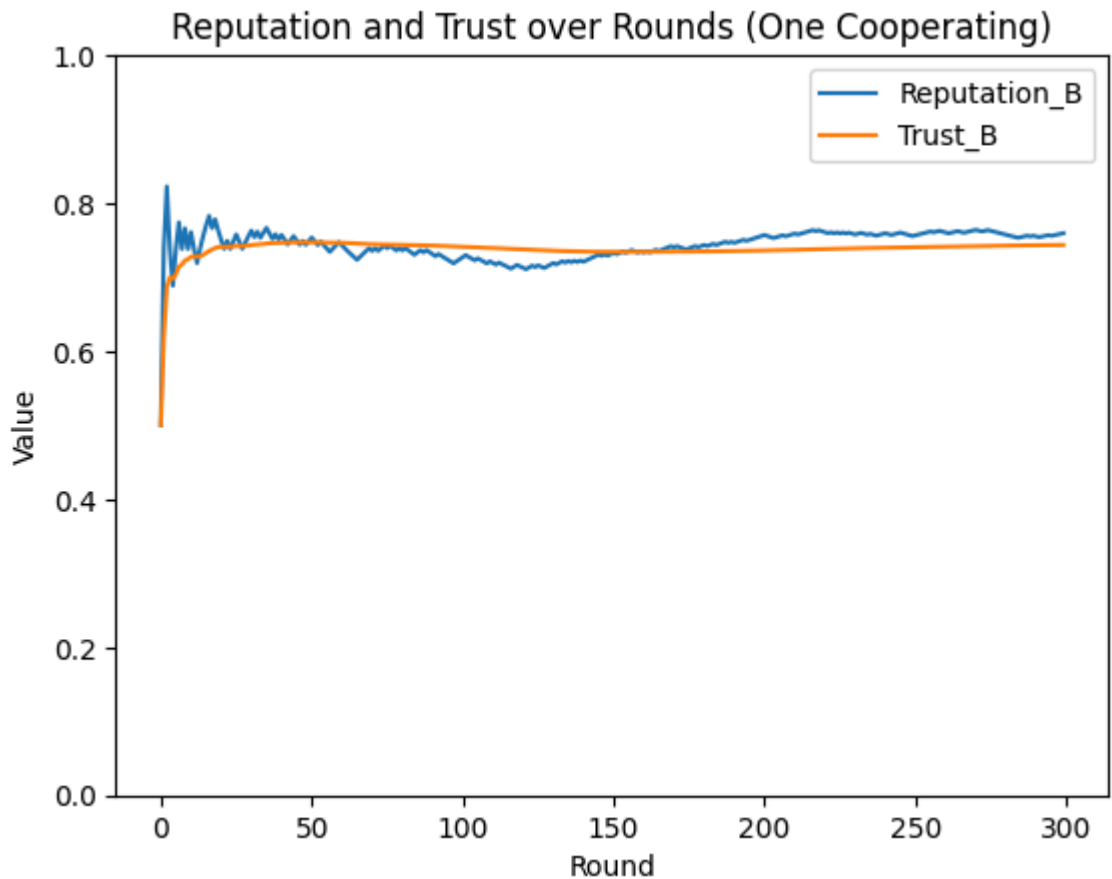
# Plot Rep values and Trust values for tit-for-tat FOR AB
plt.plot(range(len(rep_values_AB2)), rep_values_AB2, label='Reputation_B')
plt.plot(range(len(trust_values_tit_tat_AB2)), trust_values_tit_tat_AB2,
plt.xlabel('Round')
plt.ylabel('Value')
plt.title('Reputation and Trust over Rounds (One Cooperating)')
plt.legend()
plt.ylim(0, 1) # Set y-axis limits
plt.show()

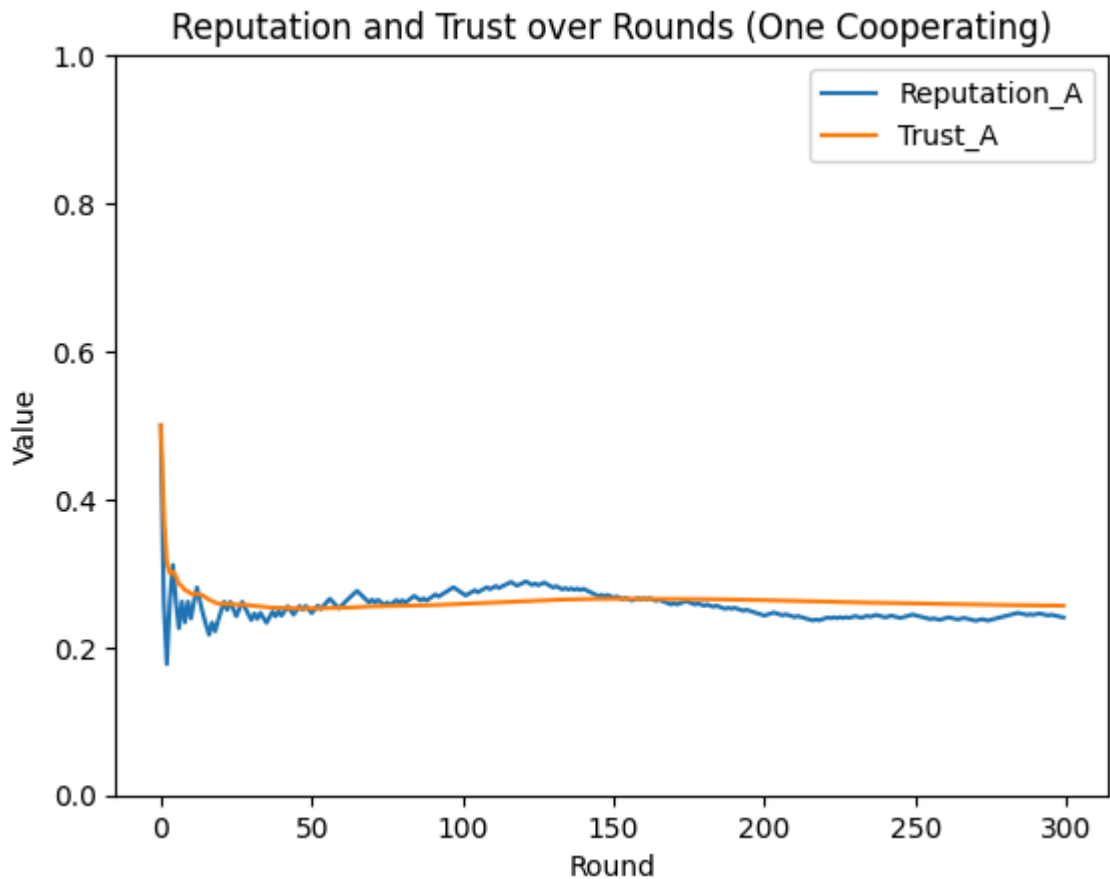
# Plot Rep values and Trust values for tit-for-tat FOR BA
plt.plot(range(len(rep_values_BA2)), rep_values_BA2, label='Reputation_A')
plt.plot(range(len(trust_values_tit_tat_BA2)), trust_values_tit_tat_BA2,
plt.xlabel('Round')
plt.ylabel('Value')
plt.title('Reputation and Trust over Rounds (One Cooperating)')
plt.legend()
plt.ylim(0, 1) # Set y-axis limits
plt.show()

```

[0.5000499999998334, 0.619154920699519, 0.6870327213700618, 0.6998224788387991, 0.6976029619751172, 0.7043801309411205, 0.7143596675077245, 0.7173483248304273, 0.7227799328332586, 0.7243305502210631, 0.7276792362791532, 0.7285651657679634, 0.7278394187397955, 0.7285915548114884, 0.7303627023827424, 0.7328361934724354, 0.7357898961133346, 0.7374914258566596, 0.7396407829477902, 0.7408393305027078, 0.7412931600224159, 0.7411613029957269, 0.7415184030057003, 0.741389887347775, 0.7416760965192901, 0.7423006111062507, 0.7425053430473567, 0.7423623165878224, 0.7425301979156141, 0.7429593819103775, 0.7436083210725368, 0.7439447617109713, 0.7444802598039736, 0.744744643812101, 0.7451898791509817, 0.745790773477911, 0.7461524205543555, 0.7463048909726153, 0.7466086048905763, 0.7467261517645513, 0.7469823808666309, 0.7470718239871401, 0.7470135236622831, 0.747086692492763, 0.7472772263364378, 0.7473313045809477, 0.747263269316933, 0.7473067023516086, 0.7472382703348478, 0.747272859429898, 0.7474007342166604, 0.7474240126806706, 0.7473525119399123, 0.7473699111004668, 0.7472995573267942, 0.7471493767145965, 0.7469265642144047, 0.7467888945592984, 0.7467295838755049, 0.7467424153494572, 0.746684137240237, 0.7465605600315923, 0.7463770087576015, 0.7461383684453138, 0.7458491247662029, 0.7455134004605025, 0.7452482411929238, 0.7450488699633329, 0.744910854316079, 0.744830078114753, 0.7447011215384498, 0.744626147627686, 0.7445054668753046, 0.7444357846988289, 0.7444138675213778, 0.7443477474668695, 0.744326935615409, 0.7442641335501403, 0.744162060590868, 0.7441034520610648, 0.7440073970485214, 0.7439526483179983, 0.743862152628574, 0.7437381095816187, 0.7435825814918183, 0.7434669040296147, 0.743388921857221, 0.7432801169099844, 0.7432072485823809, 0.7431048071006044, 0.7429744854085915, 0.7428178785481678, 0.7426959464730476, 0.7425486740174717, 0.742377477442528, 0.7421836940297998, 0.7419685870061824, 0.7417333501220406, 0.7415315048746833, 0.7413615131564416, 0.7412219126822471, 0.7411113127254711, 0.7409797131845155, 0.7408281870340152, 0.7406577525119433, 0.7405152659834815, 0.7403544130470476, 0.740176112726455, 0.7399812386396013, 0.7398132329421205, 0.7396290723505897, 0.7394295492848028, 0.7392558166315805, 0.7390670845228602, 0.7388640739763268, 0.738647472298262, 0.738417934837965, 0.7382131253435327, 0.7380320927077731, 0.7378380178246162, 0.7376315031442984, 0.737413124242823, 0.7372175725204995, 0.7370440047737847, 0.7368584700480133, 0.7366941060788343, 0.7365180090519075, 0.7363306895668613, 0.7361637632748184, 0.7360165076545484, 0.7358882276378887, 0.7357484181473044, 0.7356269331669337, 0.7355231354830396, 0.7354078277921979, 0.73530962294834, 0.7352001562718231, 0.7351072369282005, 0.7350032970943008, 0.7349153761392497, 0.7348166693607283, 0.734733478340328, 0.7346652921881788, 0.7346116178214688, 0.7345719792569947, 0.7345459169357831, 0.7345086898055273, 0.7344846525379002, 0.7344497110954712, 0.7344275897845381, 0.7344178800303972, 0.7343974012289387, 0.7343889985021522, 0.7343922923181694, 0.7343849632917727, 0.7343890259077742, 0.7344041270155184, 0.7344087601814162, 0.7344032602841405, 0.7344085797938708, 0.7344039974490846, 0.734409963878691, 0.7344062517908377, 0.734412828136165, 0.7344293889441047, 0.7344364068819468, 0.734453171244021, 0.7344793977397117, 0.7345148106444754, 0.7345591425073005, 0.7345939426899444, 0.7346374654733746, 0.7346716739533944, 0.7346968367230587, 0.7347132142940768, 0.7347382336440905, 0.7347716542741795, 0.7348132425779578, 0.734846121020658, 0.7348869954384666, 0.7349356461393752, 0.7349757265911242, 0.7350234252393468, 0.735078535226711, 0.7351252157014085, 0.7351791618003047, 0.7352401785224657, 0.7353080761926335, 0.7353676476986231, 0.7354339723839259, 0.7354921510723842, 0.7355569587136839, 0.7356282214203629, 0.7357057699207109, 0.7357752821420224, 0.7358509710386082, 0.7359326764405771, 0.7360202423564017, 0.7361135168501013, 0.73621235192253, 0.7363166033966132, 0.7364128718744463, 0.736501348892195, 0.7365822210890841, 0.7366685376730766, 0.73676016206604, 0.7368569611108682, 0.7369462746643627, 0.7370406863671128, 0.7371400696817165, 0.7372443011889478, 0.7373411705525729, 0.7374428207999175, 0.7375491344959018, 0.7376599970532676, 0.7377752966550849, 0.7378949241796507, 0.7380072949557079, 0.7381239409254914, 0.7382334803181585, 0.7383360686073236, 0.7384318576216177, 0.738531967390816, 0.7386254135

252058, 0.7387231249183454, 0.7388143042052239, 0.738909694171442, 0.7389986796126795, 0.7390918220586488, 0.73917868376983, 0.7392593974410427, 0.7393442557059356, 0.739433165800842, 0.7395160314597045, 0.7395929784178955, 0.7396739642186879, 0.7397491408425041, 0.7398186280435349, 0.7398921377598371, 0.7399695833370407, 0.7400508799851403, 0.7401265646650902, 0.7401967506210694, 0.740270773900181, 0.7403485533539227, 0.7404300095453576, 0.7405060416078004, 0.7405767567363901, 0.7406422598735984, 0.7407026537632689, 0.7407667739668337, 0.7408345444625446, 0.7409058907836644, 0.740980739982627, 0.7410590205961376, 0.7411406626111846, 0.7412172280048196, 0.741297117186298, 0.7413802629329126, 0.7414584105032306, 0.7415316535755414, 0.7416000839578658, 0.7416717931150157, 0.7417467164658542, 0.741824790695094, 0.741898114510974, 0.7419667751441317, 0.7420385798397614, 0.7421134677184559, 0.7421913790726178, 0.7422722553410926, 0.7423485210915435, 0.7424202589639579, 0.7424949578484862, 0.7425725613630458, 0.7426457028700683, 0.7427144613123938, 0.7427789141460138, 0.7428391373720571, 0.7428952055679889, 0.7429471919180463, 0.7429951682429287, 0.743039205028767, 0.7430793714553883, 0.7431157354238973, 0.743155170082625, 0.7431976213624955, 0.7432430361601488, 0.7432846820862005, 0.7433292581531477, 0.7433701199075272, 0.7434073309145819, 0.7434474569347468, 0.7434904477729863, 0.7435362541099969, 0.7435784417908262, 0.7436234143158351, 0.7436711240996026, 0.7437215243773991, 0.7437745691892053]





```
In [10]: # For both ever co-operating
payoffs_a3= []
w_payoffs_a3 = []
payoffs_b3 = []
w_payoffs_b3 = []
sum_rep_AB3=0
sum_rep_BA3=0
#A has for B=AB
trust_values_tit_tat_AB3 = []
trust_values_tit_tat_BA3= []
rep_values_AB3=[]
rep_values_BA3=[]
for i in range(len(player_a3)):
    action_a3 = player_a3[i]
    action_b3 = player_b3[i]
    key = action_a3 + action_b3
    payoffs_a3.append(payload_matrix[key][0])
    payoffs_b3.append(payload_matrix[key][1])
    for j in range(len(payoffs_a3)):
        w_payoffs_a3.append( memory_strength_values_A[j]*payoffs_a3[len(p
        w_payoffs_b3.append( memory_strength_values_B[j]*payoffs_b3[len(p
    if (sum(w_payoffs_a3)+sum(w_payoffs_b3)) == 0:
        rep_values_AB3.append(0.4) # Set trust value to a default value
        rep_values_BA3.append(0.4)
    else:
        rep_values_AB3.append(sum(w_payoffs_a3) / (sum(w_payoffs_a3)+sum(
        rep_values_BA3.append(sum(w_payoffs_b3) / (sum(w_payoffs_a3)+sum(
    w_payoffs_a3.clear()
    w_payoffs_b3.clear()
    for k in range(len(rep_values_AB3)):
        sum_rep_AB3+=rep_values_AB3[k]
        sum_rep_BA3+=rep_values_BA3[k]
```



```

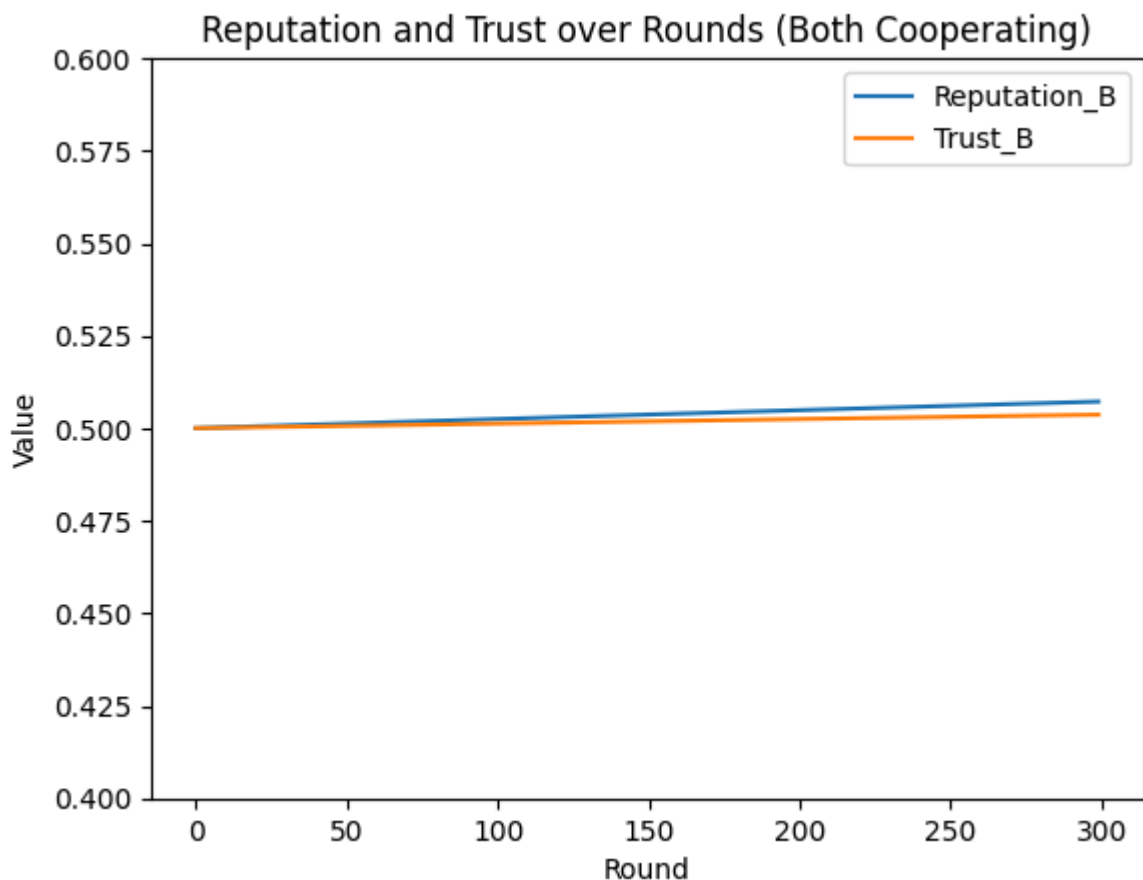
trust_values_tit_tat_AB3.append(sum_rep_AB3/(k+1))
trust_values_tit_tat_BA3.append(sum_rep_BA3/(k+1))
sum_rep_AB3=0
sum_rep_BA3=0

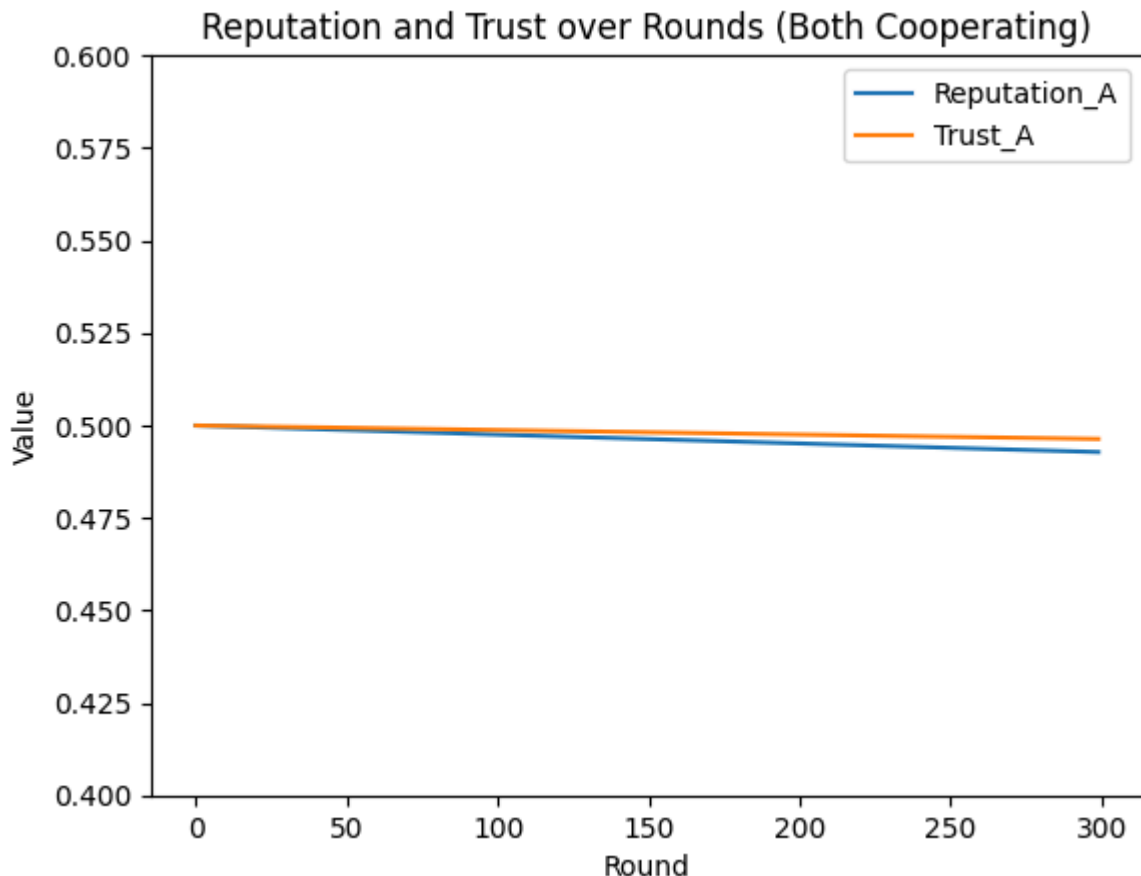
#print(rep_values_AB3)
#print(trust_values_tit_tat_AB3)

# Plot Rep values and Trust values for tit-for-tat FOR AB
plt.plot(range(len(rep_values_AB3)), rep_values_AB3, label='Reputation_B')
plt.plot(range(len(trust_values_tit_tat_AB3)), trust_values_tit_tat_AB3,
plt.xlabel('Round')
plt.ylabel('Value')
plt.title('Reputation and Trust over Rounds (Both Cooperating)')
plt.legend()
plt.ylim(0.4, 0.6) # Set y-axis limits
plt.show()

# Plot Rep values and Trust values for tit-for-tat FOR BA
plt.plot(range(len(rep_values_BA3)), rep_values_BA3, label='Reputation_A')
plt.plot(range(len(trust_values_tit_tat_BA3)), trust_values_tit_tat_BA3,
plt.xlabel('Round')
plt.ylabel('Value')
plt.title('Reputation and Trust over Rounds (Both Cooperating)')
plt.legend()
plt.ylim(0.4, 0.6) # Set y-axis limits
plt.show()

```





```
In [8]: payoff_matrix = {
        "CC": (1, 1), # Payoff for mutual cooperation
        "CD": (-10, 10), # Payoff for A: Cooperation, B: Defection
        "DC": (10, -10), # Payoff for A: Defection, B: Cooperation
        "DD": (-1, -1) # Payoff for mutual defection
    }

    # Find the minimum payoff value
    min_payoff = min(min(payoff) for payoff in payoff_matrix.values())

    # Subtract the minimum payoff value from all payoffs to normalize
    normalized_payoff_matrix = {
        key: tuple(payoff - min_payoff for payoff in value)
        for key, value in payoff_matrix.items()
    }

    print(normalized_payoff_matrix)

{'CC': (11, 11), 'CD': (0, 20), 'DC': (20, 0), 'DD': (9, 9)}
```

```
In [9]: import numpy as np

        payoff_matrix = {
            "CC": (1, 1), # Payoff for mutual cooperation
            "CD": (-10, 10), # Payoff for A: Cooperation, B: Defection
            "DC": (10, -10), # Payoff for A: Defection, B: Cooperation
            "DD": (-1, -1) # Payoff for mutual defection
        }

        # Extract the payoffs
        all_payoffs = [payoff for payoffs in payoff_matrix.values() for payoff in
```

```

# Calculate mean and standard deviation
mean_payoff = np.mean(all_payoffs)
std_dev_payoff = np.std(all_payoffs)

# Standardize the payoffs
standardized_payoff_matrix = {
    key: tuple((payoff - mean_payoff) / std_dev_payoff for payoff in value)
    for key, value in payoff_matrix.items()
}

print(standardized_payoff_matrix)

```

```

{'CC': (0.14071950894605836, 0.14071950894605836), 'CD': (-1.4071950894605838, 1.4071950894605838), 'DC': (1.4071950894605838, -1.4071950894605838), 'DD': (-0.14071950894605836, -0.14071950894605836)}

```