

# N×N-GAMES-ON-NETWORKS.NLOGO IV

---

**Metodologie e tecniche di simulazione**

Francesca Caretti



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

# n×n-games-on-networks

## **Obiettivo:**

In questa simulazione viene implementata l'evoluzione della cooperazione su una rete complessa seguendo la regola di aggiornamento proposta da Santos e Pacheco. Come vedremo, gli agenti accumulano i payoff seguendo diverse regole di interazione e aggiornano le loro strategie con una probabilità che dipende dalla differenza normalizzata di payoff e dal grado massimo.

# Model games on networks



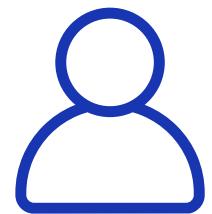
→ è un modello di gioco evolutivo su reti il cui **obiettivo** è capire come evolve la cooperazione tra agenti che interagiscono su reti complesse



Ogni agente interagisce con i propri  $n$  vicini nel grafo



La rete è costruita utilizzando il modello di **Barabási-Albert\***



Ogni agente ha una funzione di payoff che dipende dalla sua strategia e dalle strategie dei suoi vicini.



0 = defezione, 1 = cooperazione



Ogni agente aggiorna la propria strategia in base a una regola evolutiva che dipende dal payoff e dal numero di vicini.



## \*Modello Barabási-Albert

ogni nuovo nodo si collega a nodi esistenti con probabilità proporzionale al loro grado (preferential attachment)



la rete risultante è una scale-free network  
(pochi nodi molto connessi, molti nodi con poche connessioni)



```
self.graph = nx.barabasi_albert_graph(n_players, 2)  
self.grid = NetworkGrid(self.graph)
```

parametro min-degree  
→ numero minimo di connessioni che ogni nuovo nodo stabilisce entrando nella rete.



Modello come  
generalizzazione  
dei giochi evolutivi

**Gli agenti possono interagire  
secondo diversi modi**

Parametro play\_with

**Possono aggiornare le strategie  
secondo diverse regole**

Parametro decision\_rule

**Si può controllare l'intensità  
delle decisioni.**

Parametro m



# Parametro play\_with

Parametro che determina come i payoff degli agenti sono calcolati per ogni invio

## one-random-nbr

.....

Gli agenti giocano con uno dei loro vicini scelto random

## all-nbrs-AVG-payoff

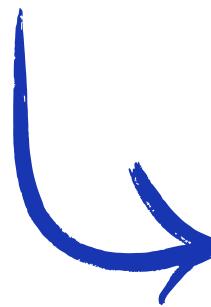
.....

Gli agenti giocano con tutti i loro  $n$  vicini e il payoff è la media di tutti

## all-nbrs-TOTAL-payoff

.....

Gli agenti giocano con tutti i loro vicini  $n$  e usano payoff accumulati (somma)



Funzione **calculate\_payoff** che, in base al valore di play\_with, permette di scegliere come ogni agente calcola il suo payoff.

```

10  class PlayerAgent(Agent):
11 >     def __init__(self, model, strategy): ...
12
13
14
15
16
17     def calculate_payoff(self):
18         # Trova gli agenti vicini
19         neighbors = list(self.model.graph.neighbors(self.pos))
20         agents = [self.model.grid.get_cell_list_contents([n])[0] for n in neighbors if self.model.grid.get_cell_list_contents([n])]
21
22         if not agents:
23             self.payoff = 0
24             return
25
26
27         # Se si gioca contro un solo vicino casuale
28         if self.model.play_with == "one-random-nbr":
29             partner = self.random.choice(agents)
30             self.payoff = self.model.payoff_matrix[self.strategy][partner.strategy]
31         else:
32             # Si gioca contro tutti i vicini
33             total_payoff = 0
34             for neighbor in agents:
35                 total_payoff += self.model.payoff_matrix[self.strategy][neighbor.strategy]
36
37             # Se è richiesto il totale, lo lasciamo; altrimenti, calcoliamo la media
38             if self.model.play_with.endswith("all-nbrs-TOTAL-payoff"):
39                 self.payoff = total_payoff
40             else:
41                 self.payoff = total_payoff / len(agents)
42
43     def decide_strategy(self):
44         neighbors = self.model.graph.neighbors(self.pos)

```



# Parametro decision\_rule

Parametro che determina la decisione che ogni agente prenderà per migliorare la propria strategia, sulla base dei payoff altrui.

- ◆ best-neighbor
- ◆ imitate-if-better
- ◆ imitative-pairwise-difference
- ◆ imitative-positive-proportional-m
- ◆ fermi-m
- ◆ **Santos-Pacheco**



L'agente  $i$  guarda ad uno dei suoi vicini  $j$  (casuale) e copia la sua strategia con una probabilità proporzionale alla differenza di payoff

# Santos-Pacheco

Gli agenti accumulano i payoff sommando le interazioni con tutti i vicini e aggiornano le loro strategie con una probabilità che dipende dalla differenza normalizzata di payoff e dal grado massimo.

→ scegli un vicino random → confronta i payoff → prendi il grado massimo tra i due → calcola la probabilità di imitazione → decidi se cambiare strategia

```

dr = self.model.decision_rule
if dr == "best-neighbor":
    best = max(agents + [self], key=lambda a: a.payoff)
    self.strategy_after_revision = best.strategy
elif dr == "imitate-if-better":
    n = self.random.choice(agents)
    self.strategy_after_revision = n.strategy if n.payoff > self.payoff else self.strategy

elif dr == "Santos-Pacheco":
    n = self.random.choice(agents) #scelta di un vicino casuale
    diff = n.payoff - self.payoff #calcolo della differenza di payoff
    # per capire quanto gli agenti sono connessi nella rete:
    ki = len(list(neighbors)) #numero dei vicini dell'agente attuale
    kj = len(list(self.model.grid.get_neighbors(n.pos, include_center=False))) #numero dei vicini dell'agente scelto
    denom = self.model.max_payoff_diff_matrix * max(ki, kj) #calcolo un denominatore che tiene conto sia della rete che dei payoff
    prob = max(0, diff / denom) #calcolo quant'è la probabilità di imitare
    self.strategy_after_revision = n.strategy if self.random.random() < prob else self.strategy #prendo la decisione finale
else:
    self.strategy_after_revision = self.strategy

```

differenza massima di  
payoff possibile

grado più alto



# Parametro m

Parametro che influenza l'intensità nelle decisioni di alcune regole (come fermi-m)  
→ modula quanto è sensibile l'agente alla differenza di payoff

```
class GameModel(Model):
    def __init__(self, n_players, strategies, payoff_matrix,
                 play_with="all-nbrs-TOTAL-payoff", decision_rule="best-neighbor",
                 m=0.1, noise=0.0, seed=None):
        super().__init__(seed=seed)
        self.strategies = strategies
        self.n_strategies = len(strategies)
        self.payoff_matrix = payoff_matrix
        self.play_with = play_with
        self.decision_rule = decision_rule
        self.noise = noise
        self.m = m
```



# Classe PlayerAgent

Ad ogni step:

**calcolo payoff → interazione con vicini → aggiornamento  
strategie → raccolta dati (monitoraggio)**

```
10 ##### Player Agent Class #####
11 class PlayerAgent(Agent):
12 >     def __init__(self, model, strategy): ...
17
18 >     def calculate_payoff(self): ...
41
42 >     def decide_strategy(self): ...
90
91 >     def update_strategy(self): ...
93
```



# Classe GameModel

**inizializzazione dei parametri → preparazione degli agenti → piazzamento degli agenti → calcolo differenze massime di payoff → calcolo della distribuzione delle strategie degli agenti → ciclo operativo della simulazione → costruzione modello**

```
95 ##### Game Model Class #####
96 class GameModel(Model):
97 >     def __init__(self, n_players, strategies, payoff_matrix, ...
134
135 >     def strategy_distribution(self): ...
141
142 >     def step(self): ...
147
```

# Simulazioni implementate:



- 1. Variazione della modalità di gioco (play\_with)**
- 2. Variazione del parametro di intensità (m)**
- 3. Introduzione del rumore (noise)**
- 4. Variazione del numero di agenti (n\_players)**
- 5. Analisi della sensibilità rispetto alla modalità di interazione ('play\_with')**

# 1. Variazione della modalità di gioco (play\_with)

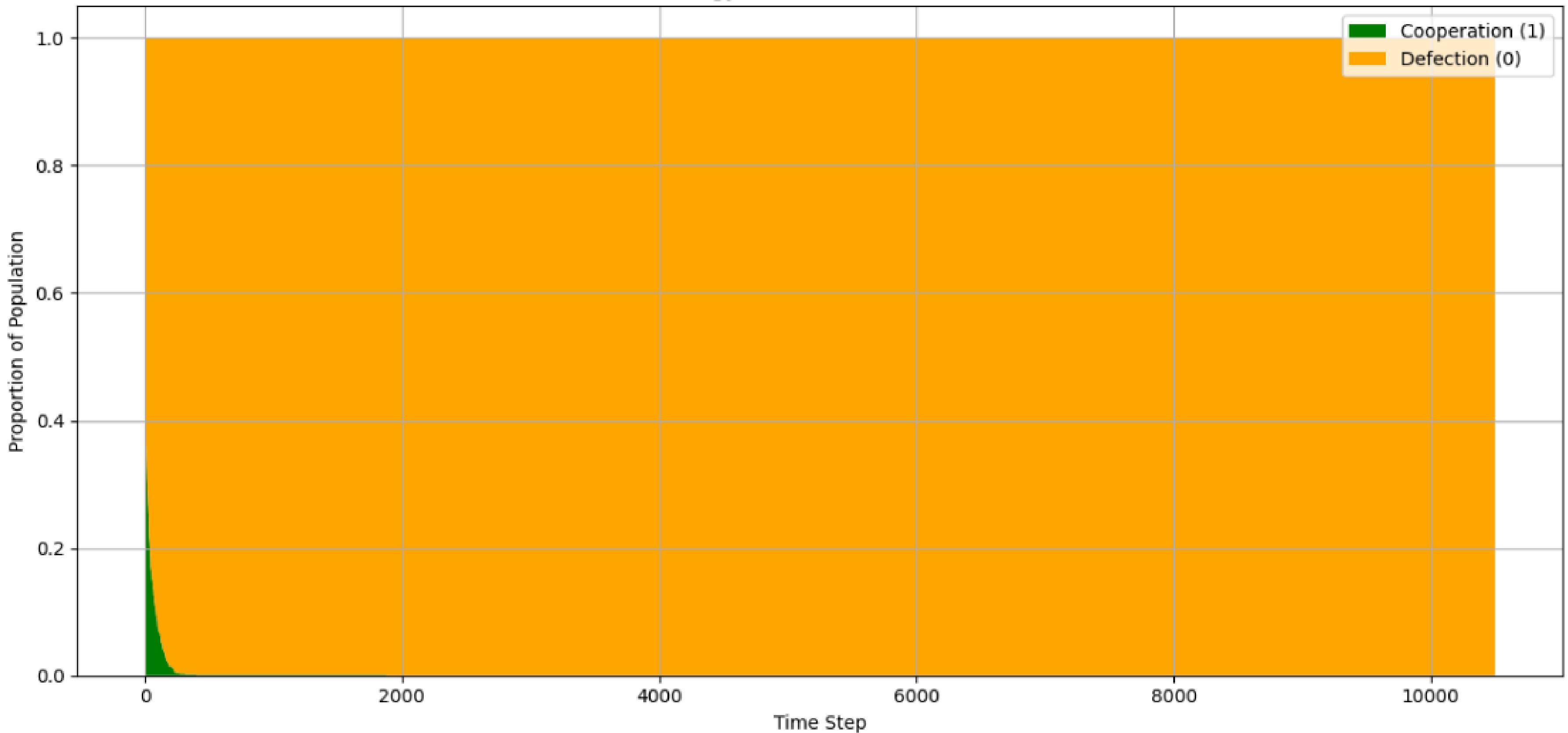


```
10 ##### Player Agent Class #####
11 > class PlayerAgent(Agent): ...
94
95
96 ##### Game Model Class #####
97 > class GameModel(Model): ...
148
149
150 ##### Main Function to Run the Model #####
151 if __name__ == "__main__":
152 >     import matplotlib.pyplot as plt...
156
157
158     model = GameModel(
159         n_players=1000,
160         strategies=[500, 500],
161         payoff_matrix=[[0, 1.1875], [-0.1875, 1]],
162         play_with="one-random-nbr",
163         decision_rule="Santos-Pacheco",
164         m=0.5,
165         noise=0.0,
166         seed=38
167     )
168
```

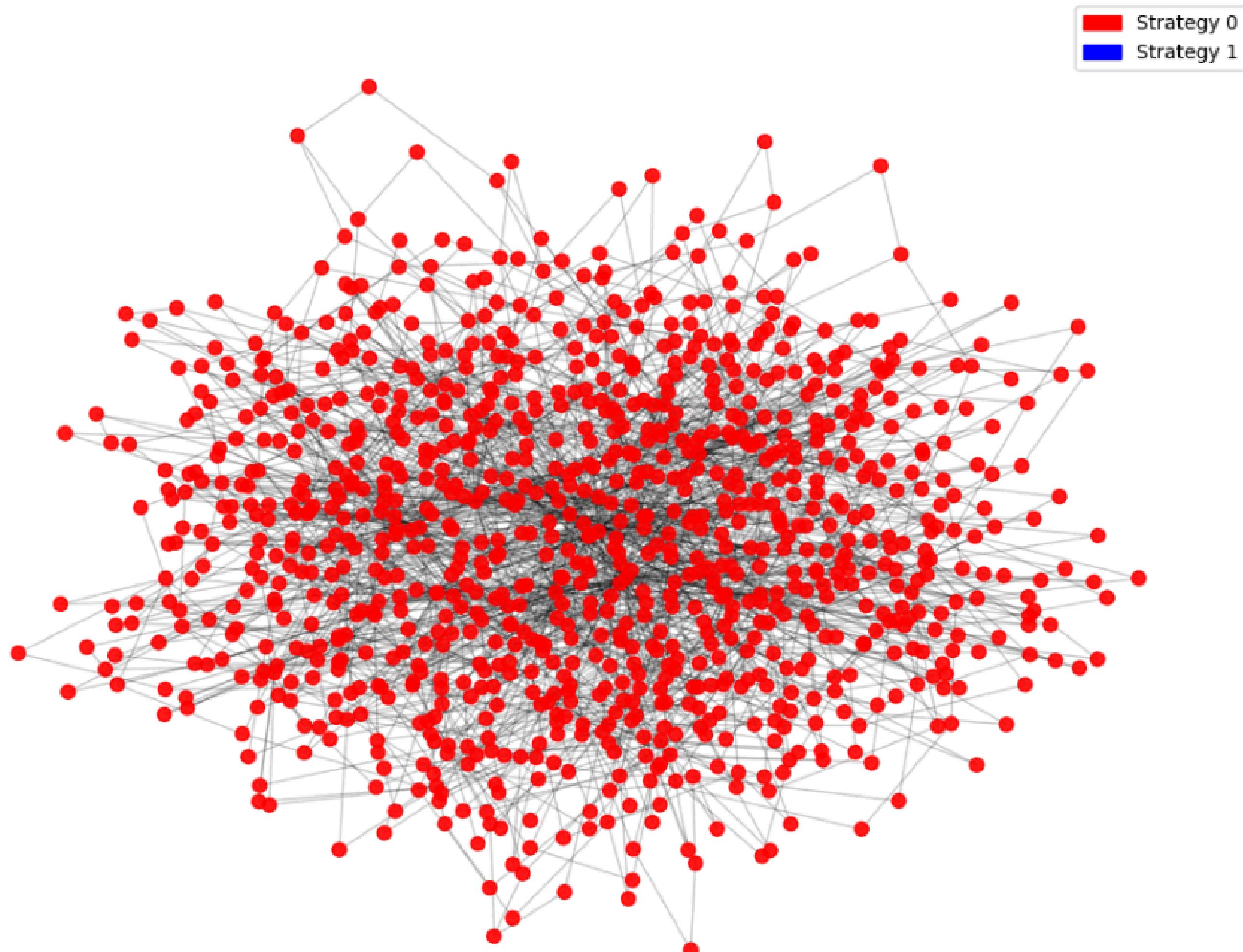
## → Esecuzione del modello

- popolazione di 1000 agenti
- gioco del prigioniero con payoffs [0 1.1875] [-0.1875 1]
- parametro “**one-random-nbr**”
- decision rule: Santos-Pacheco
- parametro m = 0.5
- assenza di rumore

### Strategy Distribution Over Time



Final Network: Node Color = Strategy, Size = Payoff



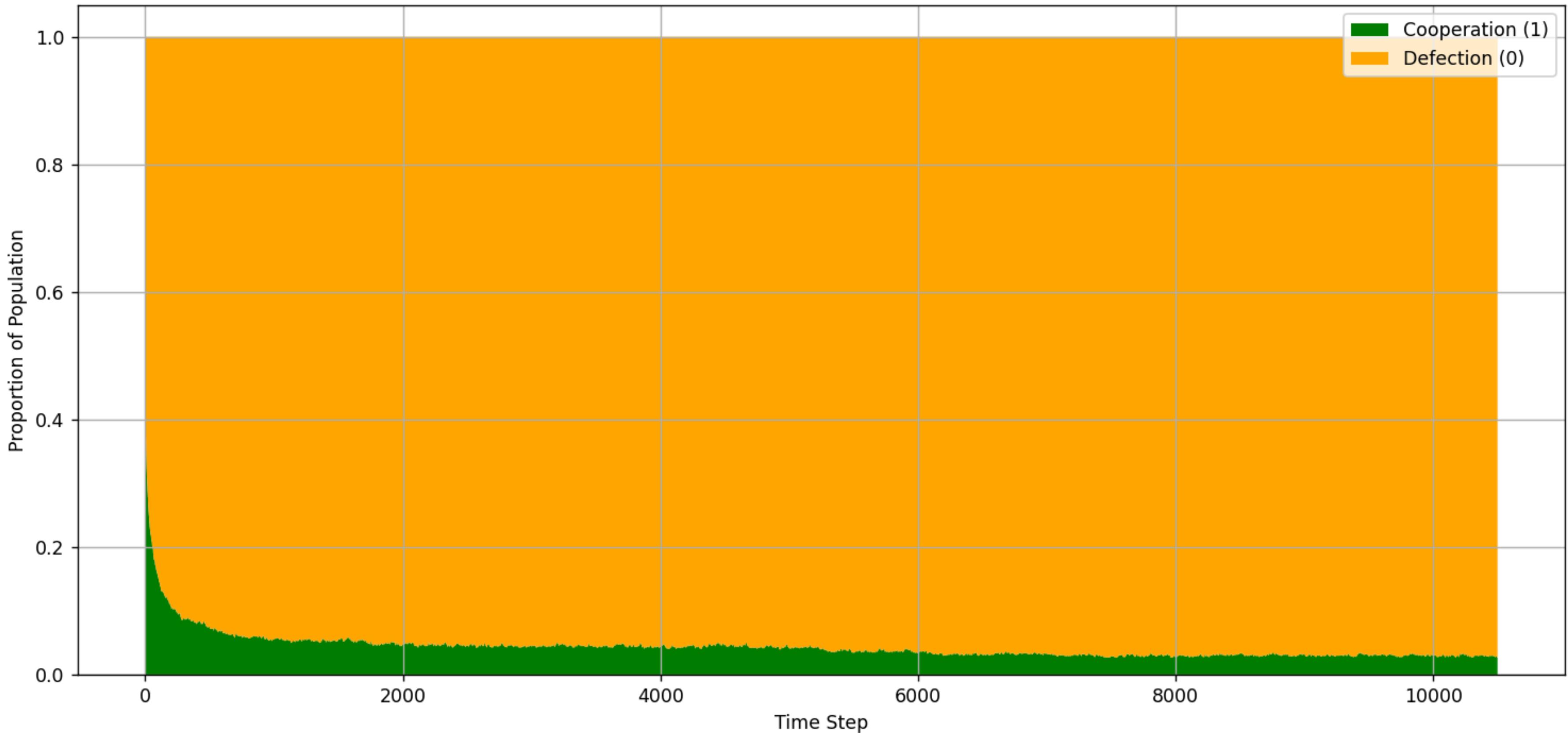


```
10 ##### Player Agent Class #####
11 > class PlayerAgent(Agent): ...
94
95
96 ##### Game Model Class #####
97 > class GameModel(Model): ...
148
149
150 ##### Main Function to Run the Model #####
151 if __name__ == "__main__":
152 >     import matplotlib.pyplot as plt...
156
157
158     model = GameModel(
159         n_players=1000,
160         strategies=[500, 500],
161         payoff_matrix=[[0, 1.1875], [-0.1875, 1]],
162         play_with="all-nbrs-AVG-payoff",
163         decision_rule="Santos-Pacheco",
164         m=0.5,
165         noise=0.0,
166         seed=38
167     )
168
169     for _ in range(10500):
170         model.step()
```

## → Esecuzione del modello

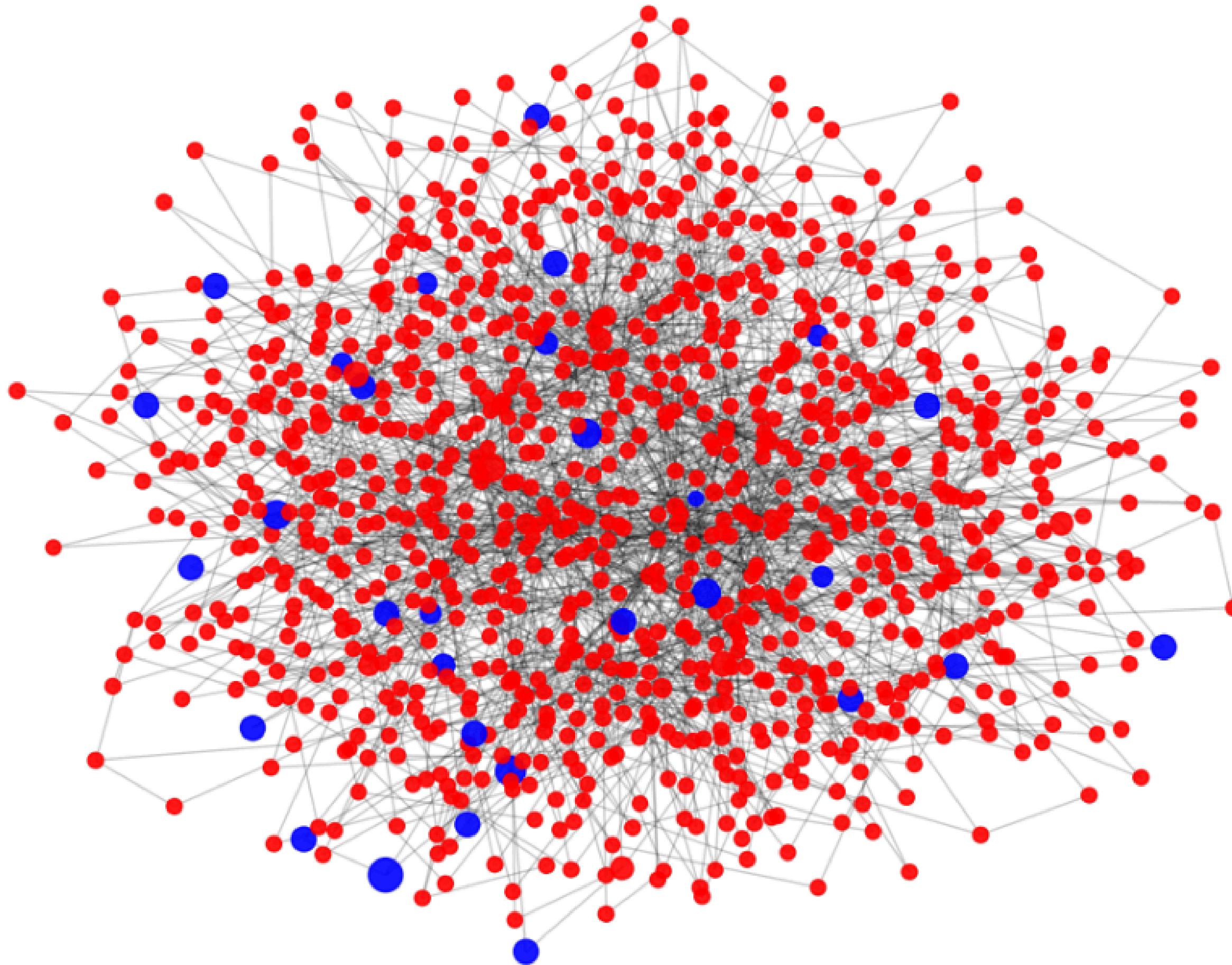
- popolazione di 1000 agenti
- gioco del prigioniero con payoffs  
[0 1.1875] [-0.1875 1]
- parametro “**all-nbrs-AVG-payoff**”
- decision rule: Santos-Pacheco
- parametro m = 0.5
- assenza di rumore

### Strategy Distribution Over Time



Final Network: Node Color = Strategy, Size = Payoff

Strategy 0  
Strategy 1



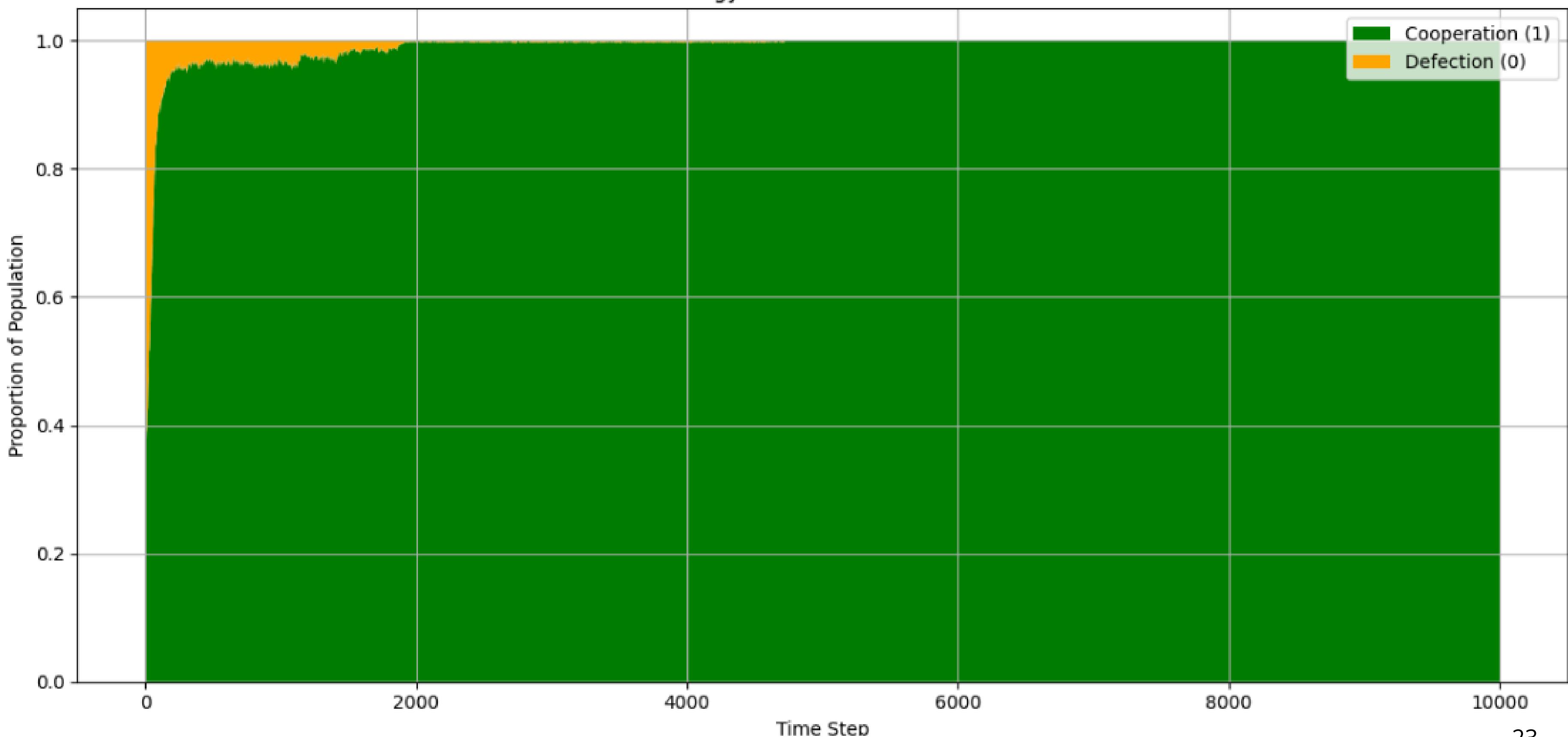


```
11 ##### Player Agent Class #####
12 > class PlayerAgent(Agent): ...
95
96
97 ##### Game Model Class #####
98 > class GameModel(Model): ...
149
150
151 ##### Main Function to Run the Model #####
152 if __name__ == "__main__":
153 > import matplotlib.pyplot as plt...
157
158
159 model = GameModel(
160     n_players=1000,
161     strategies=[500, 500],
162     payoff_matrix=[[0, 1.1875], [-0.1875, 1]],
163     play_with="all-nbrs-TOTAL-payoff",
164     decision_rule="Santos-Pacheco",
165     m=0.5,
166     noise=0.0,
167     seed=38
168 )
169
170     for _ in range(10500):
171         model.step()
```

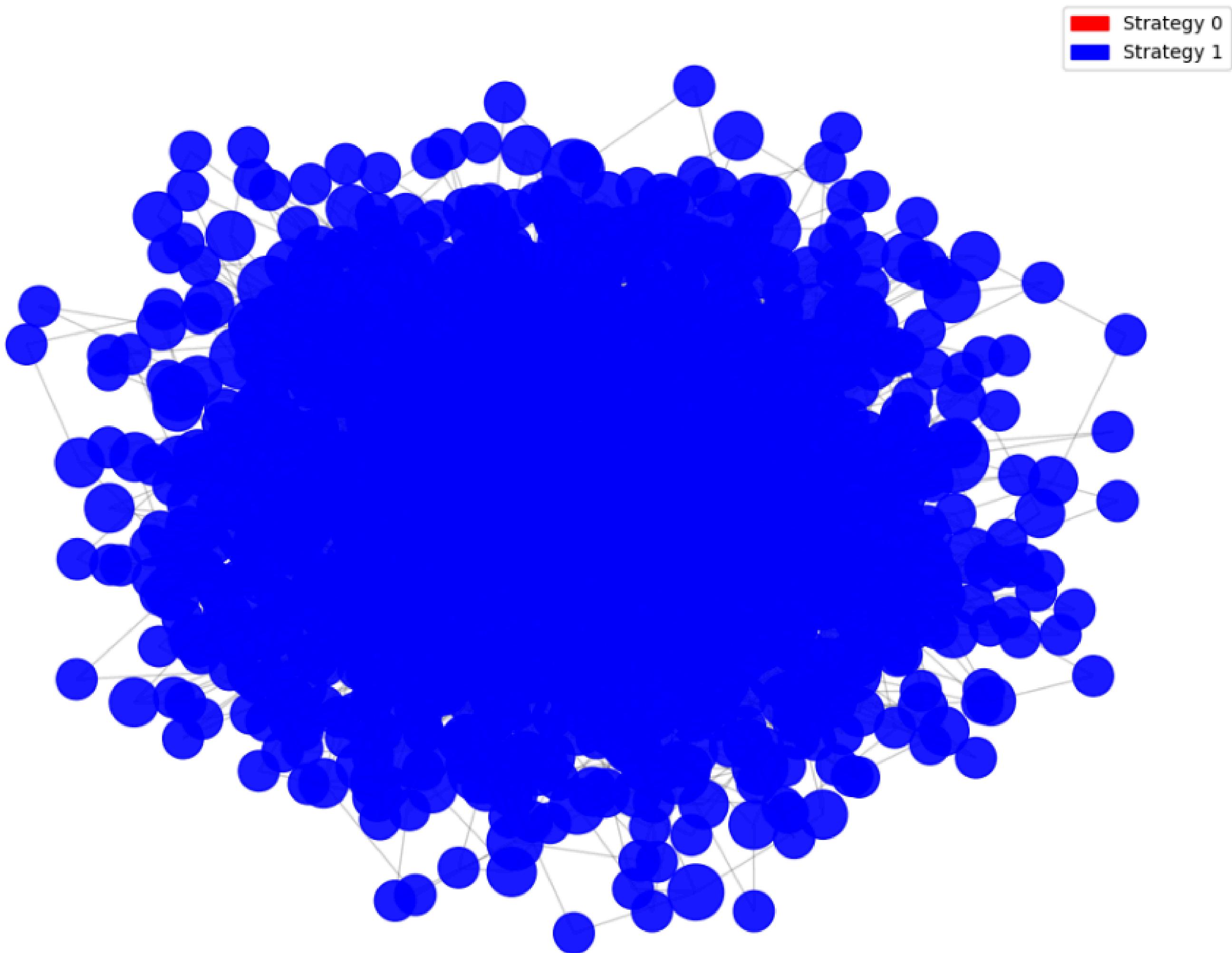
## → Esecuzione del modello

- popolazione di 1000 agenti
- gioco del prigioniero con payoffs [0 1.1875] [-0.1875 1]
- parametro “**all-nbrs-TOTAL-payoff**”
- decision rule: Santos-Pacheco
- parametro m = 0.5
- assenza di rumore

### Strategy Distribution Over Time



Final Network: Node Color = Strategy, Size = Payoff

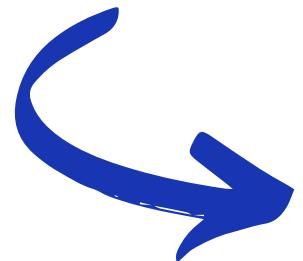




# Evidenze:

Posto:

- il parametro decision-rule = Santos Pacheco;
- la matrice dei payoff =[[0, 1.1875], [-0.1875, 1]];
- $m=0.5$ ;
- noise=0.0.



Quando ogni agente gioca contro un solo vicino scelto casualmente, la cooperazione scende rapidamente fino a quasi azzerarsi. Se invece il payoff è calcolato come media delle interazioni con tutti i vicini, la cooperazione si mantiene bassa ma stabile nel tempo. Infine, se il payoff è la somma delle interazioni con tutti i vicini, la cooperazione cresce e si mantiene su livelli molto alti.

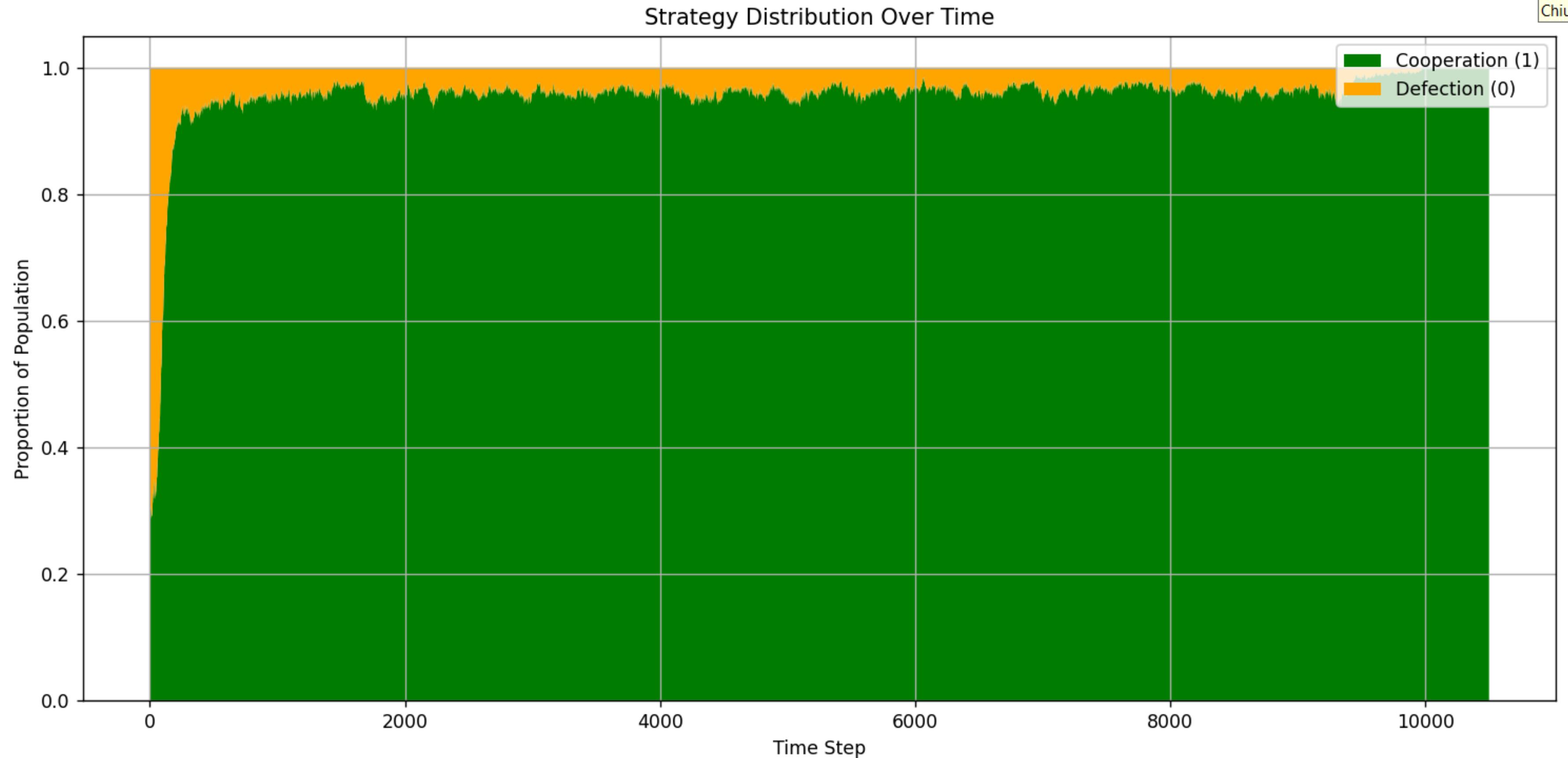
## **2. Variazione del parametro di intensità (m)**



```
10 ##### Player Agent Class #####
11 > class PlayerAgent(Agent): ...
94
95
96 ##### Game Model Class #####
97 > class GameModel(Model): ...
148
149
150 ##### Main Function to Run the Model #####
151 if __name__ == "__main__":
152 >     import matplotlib.pyplot as plt...
156
157
158     model = GameModel(
159         n_players=1000,
160         strategies=[500, 500],
161         payoff_matrix=[[0, 1.1875], [-0.1875, 1]],
162         play_with="all-nbrs-TOTAL-payoff",
163         decision_rule="Santos-Pacheco",
164         m=0.1,
165         noise=0.0,
166         seed=38
167     )
```

## → Esecuzione del modello

- popolazione di 1000 agenti
- gioco del prigioniero con payoffs [0 1.1875] [-0.1875 1]
- parametro “all-nbrs-TOTAL-payoff”
- decision rule: Santos-Pacheco
- **parametro m = 0.1**
- assenza di rumore



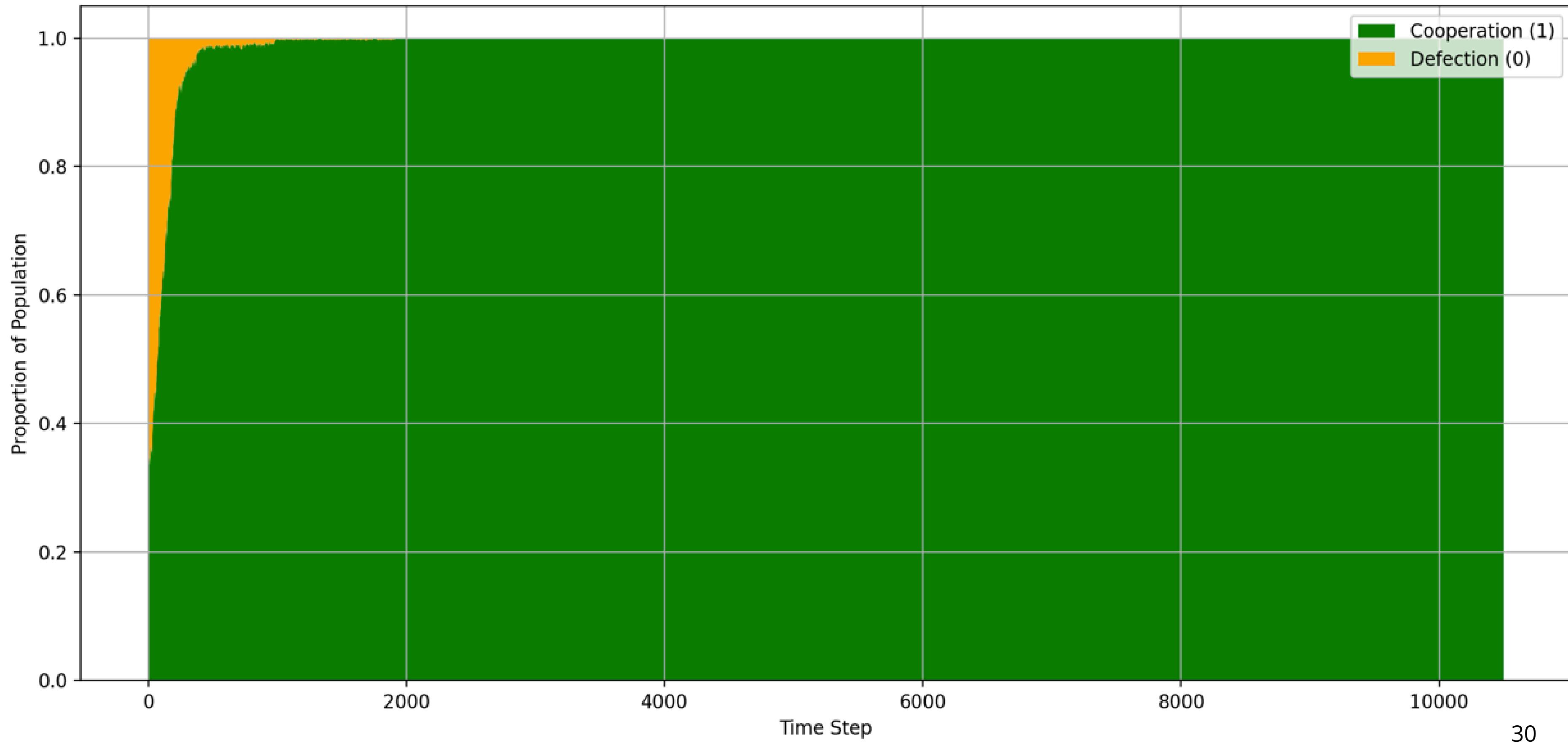


```
94 ##### Game Model Class #####
95 class GameModel(Model):
96     def __init__(self, n_players, strategies, payoff_matrix, ...
133
134     def strategy_distribution(self): ...
140
141     def step(self): ...
146
147
148 ##### Main Function to Run the Model #####
149 if __name__ == "__main__":
150     import matplotlib.pyplot as plt ...
154
155
156     model = GameModel(
157         n_players=1000,
158         strategies=[500, 500],
159         payoff_matrix=[[0, 1.1875], [-0.1875, 1]],
160         play_with="all-nbrs-TOTAL-payoff",
161         decision_rule="Santos-Pacheco",
162         m=0.5,
163         noise=0.0,
164         seed=38
165     )
```

## → Esecuzione del modello

- popolazione di 1000 agenti
- gioco del prigioniero con payoffs [0 1.1875] [-0.1875 1]
- parametro “all-nbrs-TOTAL-payoff”
- decision rule: Santos-Pacheco
- **parametro m = 0.5**
- assenza di rumore

### Strategy Distribution Over Time



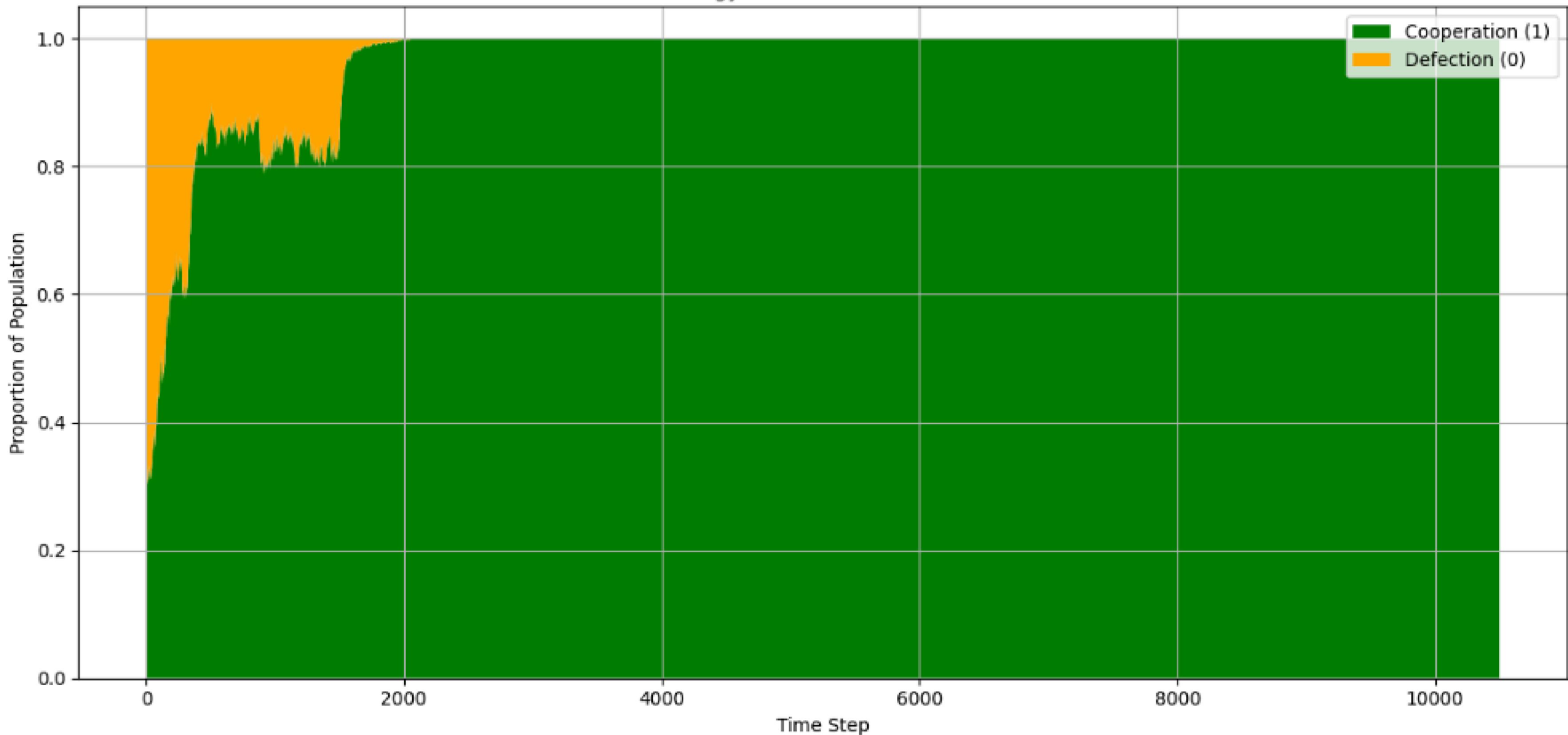


```
10 ##### Player Agent Class #####
11 > class PlayerAgent(Agent): ...
94
95
96 ##### Game Model Class #####
97 > class GameModel(Model): ...
148
149
150 ##### Main Function to Run the Model #####
151 if __name__ == "__main__":
152 >     import matplotlib.pyplot as plt...
156
157
158     model = GameModel(
159         n_players=1000,
160         strategies=[500, 500],
161         payoff_matrix=[[0, 1.1875], [-0.1875, 1]],
162         play_with="all-nbrs-TOTAL-
payoff",
163         decision_rule="Santos-Pacheco",
164         m=1,
165         noise=0.0,
166         seed=38
167     )
168
169     for _ in range(10500):
170         model.step()
```

## → Esecuzione del modello

- popolazione di 1000 agenti
- gioco del prigioniero con payoffs [0 1.1875] [-0.1875 1]
- parametro “all-nbrs-TOTAL-payoff”
- decision rule: Santos-Pacheco
- **parametro m = 1**
- assenza di rumore

### Strategy Distribution Over Time





# Evidenze:

Posto:

- il parametro decision-rule = Santos Pacheco;
- il parametro play-with = all-nbrs-TOTAL-payoff
- la matrice dei payoff =[[0, 1.1875], [-0.1875, 1]];
- noise=0.0.



La cooperazione più alta si ha quando  $m$  è un valore medio. Questo perchè, quando  $m$  è grande, gli agenti imitano molto fedelmente chi ha il payoff maggiore favorendo la selezione rapida delle strategie dominanti, che in giochi sociali spesso sono strategie non cooperative (in quanto nel breve periodo pagano di più)

### **3. Introduzione del rumore (noise = 0.01)**

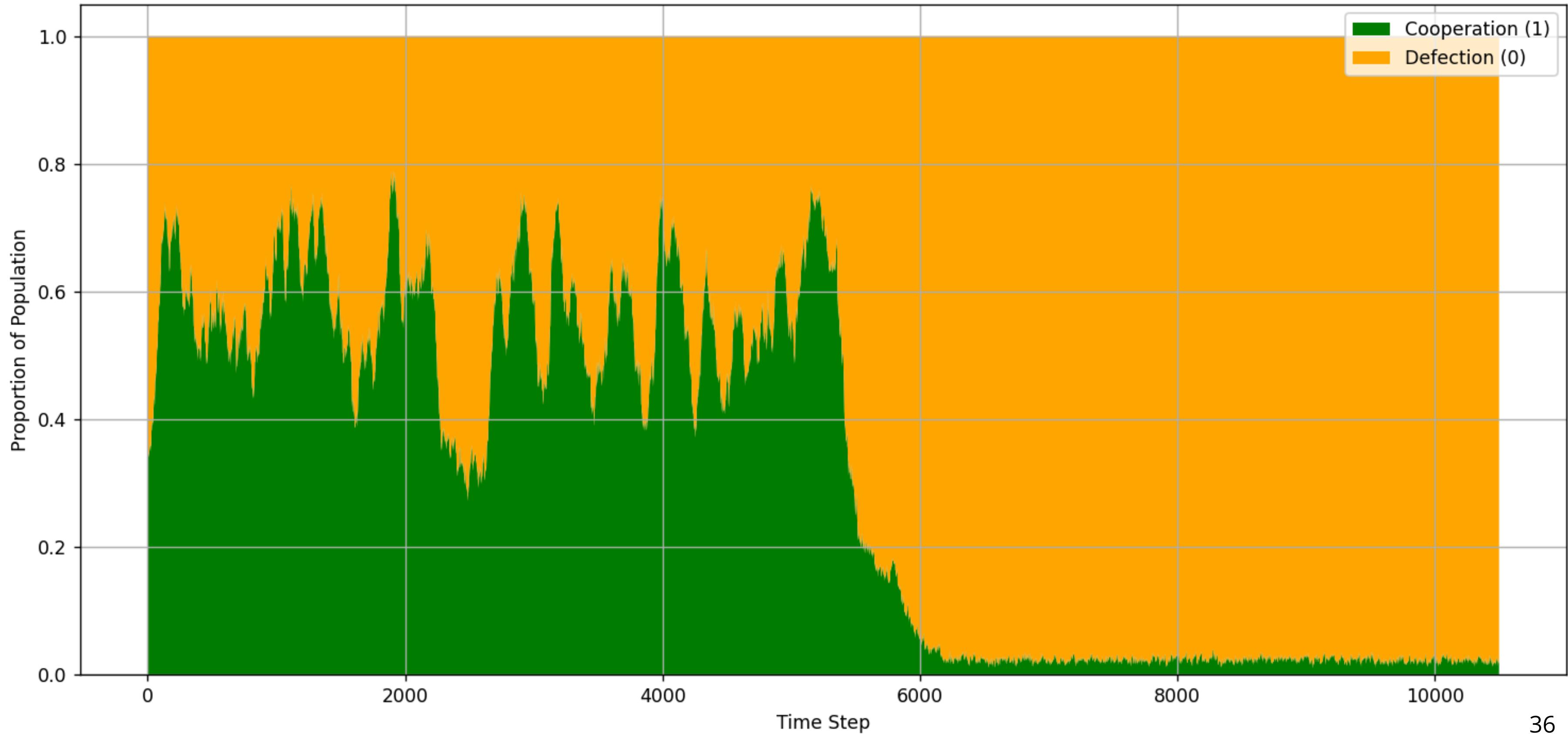


```
11 ##### Player Agent Class #####
12 > class PlayerAgent(Agent): ...
96
97
98 ##### Game Model Class #####
99 > class GameModel(Model): ...
150
151
152 ##### Main Function to Run the Model #####
153 if __name__ == "__main__":
154 >     import matplotlib.pyplot as plt...
158
159     model = GameModel(
160         n_players=1000,
161         strategies=[500, 500],
162         payoff_matrix=[[0, 1.1875], [-0.1875, 1]],
163         play_with="all-nbrs-TOTAL-payoff",
164         decision_rule="Santos-Pacheco",
165         m=0.5,
166         noise=0.01,
167         seed=38
168     )
170
171     for _ in range(10500):
172         model.step()
```

## → Esecuzione del modello

- popolazione di 1000 agenti
- gioco del prigioniero con payoffs [0 1.1875] [-0.1875 1]
- parametro “all-nbrs-TOTAL-payoff”
- decision rule: Santos-Pacheco
- parametro m = 0.5
- **noise = 0.01**
- **10500 step**

Strategy Distribution Over Time - noise



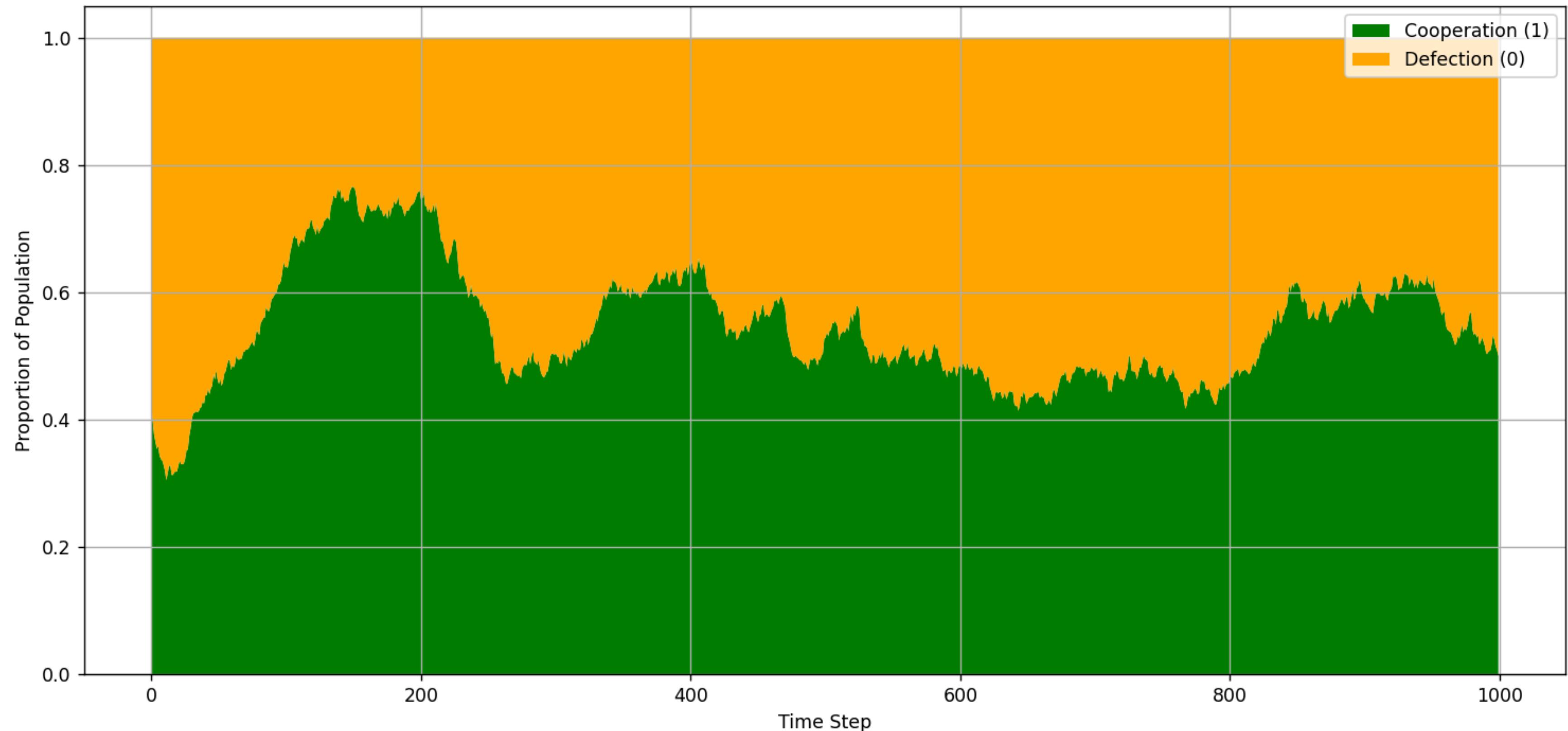


```
11 ##### Player Agent Class #####
12 > class PlayerAgent(Agent): ...
96
97
98 ##### Game Model Class #####
99 > class GameModel(Model): ...
150
151
152 ##### Main Function to Run the Model #####
153 if __name__ == "__main__":
154 >     import matplotlib.pyplot as plt...
158
159     model = GameModel(
160         n_players=1000,
161         strategies=[500, 500],
162         payoff_matrix=[[0, 1.1875], [-0.1875, 1]],
163         play_with="all-nbrs-TOTAL-payoff",
164         decision_rule="Santos-Pacheco",
165         m=0.5,
166         noise=0.01,
167         seed=38
168     )
169     for _ in range(1000):
170         model.step()
172
```

## → Esecuzione del modello

- popolazione di 1000 agenti
- gioco del prigioniero con payoffs [0 1.1875] [-0.1875 1]
- parametro “all-nbrs-TOTAL-payoff”
- decision rule: Santos-Pacheco
- parametro m = 0.5
- **noise = 0.01**
- **1000 step**

Strategy Distribution Over Time - noise



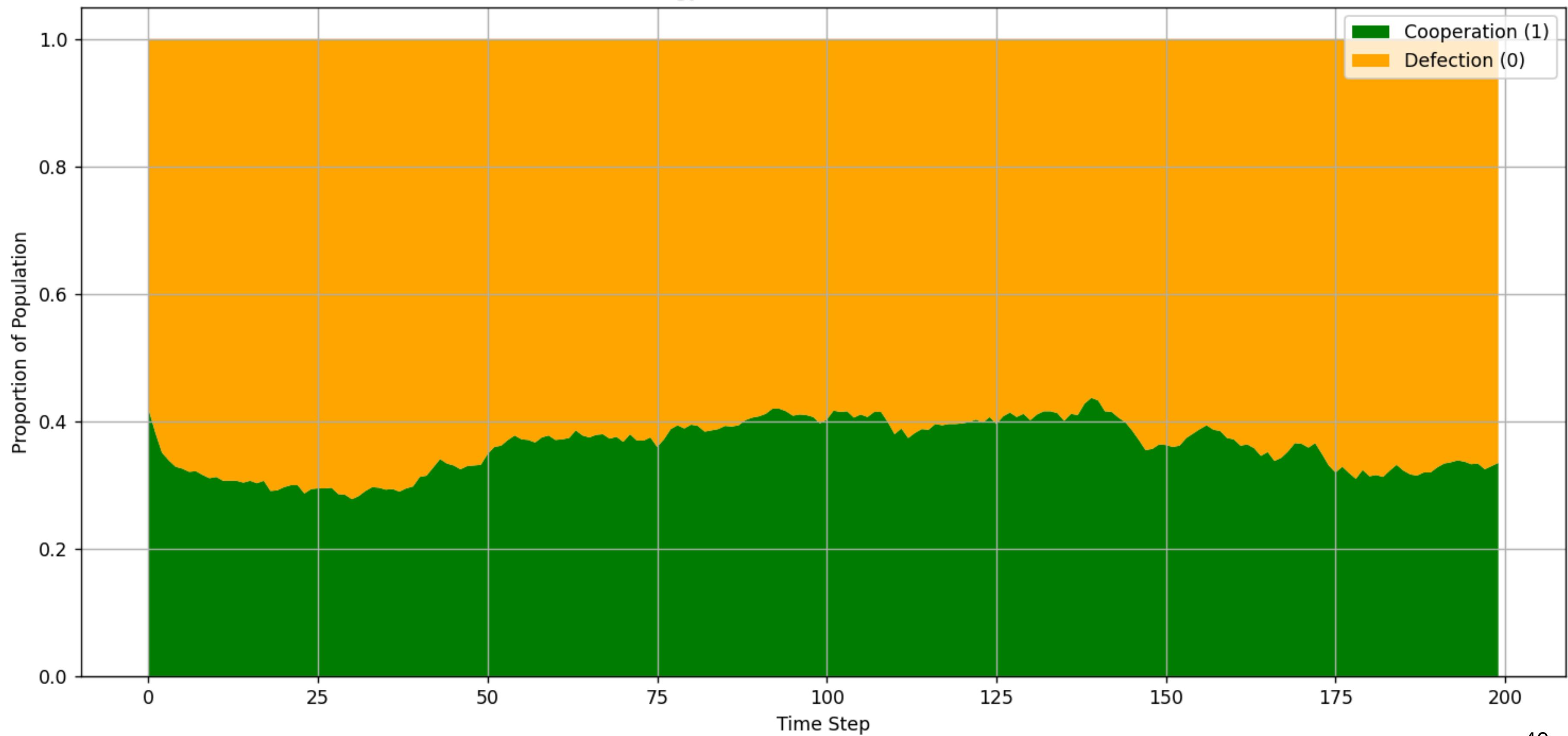


```
12 > class PlayerAgent(Agent): ...
96
97
98 ##### Game Model Class #####
99 > class GameModel(Model):
150
151
152 ##### Main Function to Run the Model #####
153 if __name__ == "__main__":
154 >     import matplotlib.pyplot as plt...
158
159
160     model = GameModel(
161         n_players=1000,
162         strategies=[500, 500],
163         payoff_matrix=[[0, 1.1875], [-0.1875, 1]],
164         play_with="all-nbrs-TOTAL-payoff",
165         decision_rule="Santos-Pacheco",
166         m=0.5,
167         noise=0.01,
168         seed=38
169     )
170
171     for _ in range(200):
172         model.step()
173
174     df = model.datacollector.get_model_vars_dataframe()
```

## → Esecuzione del modello

- popolazione di 1000 agenti
- gioco del prigioniero con payoffs [0 1.1875] [-0.1875 1]
- parametro “all-nbrs-TOTAL-payoff”
- decisione rule: Santos-Pacheco
- parametro m = 0.5
- **noise = 0.01**
- **200 step**

### Strategy Distribution Over Time - noise

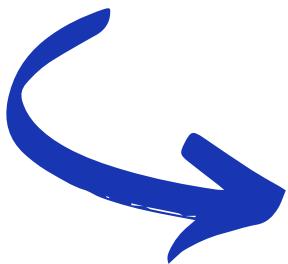




# Evidenze:

Posto:

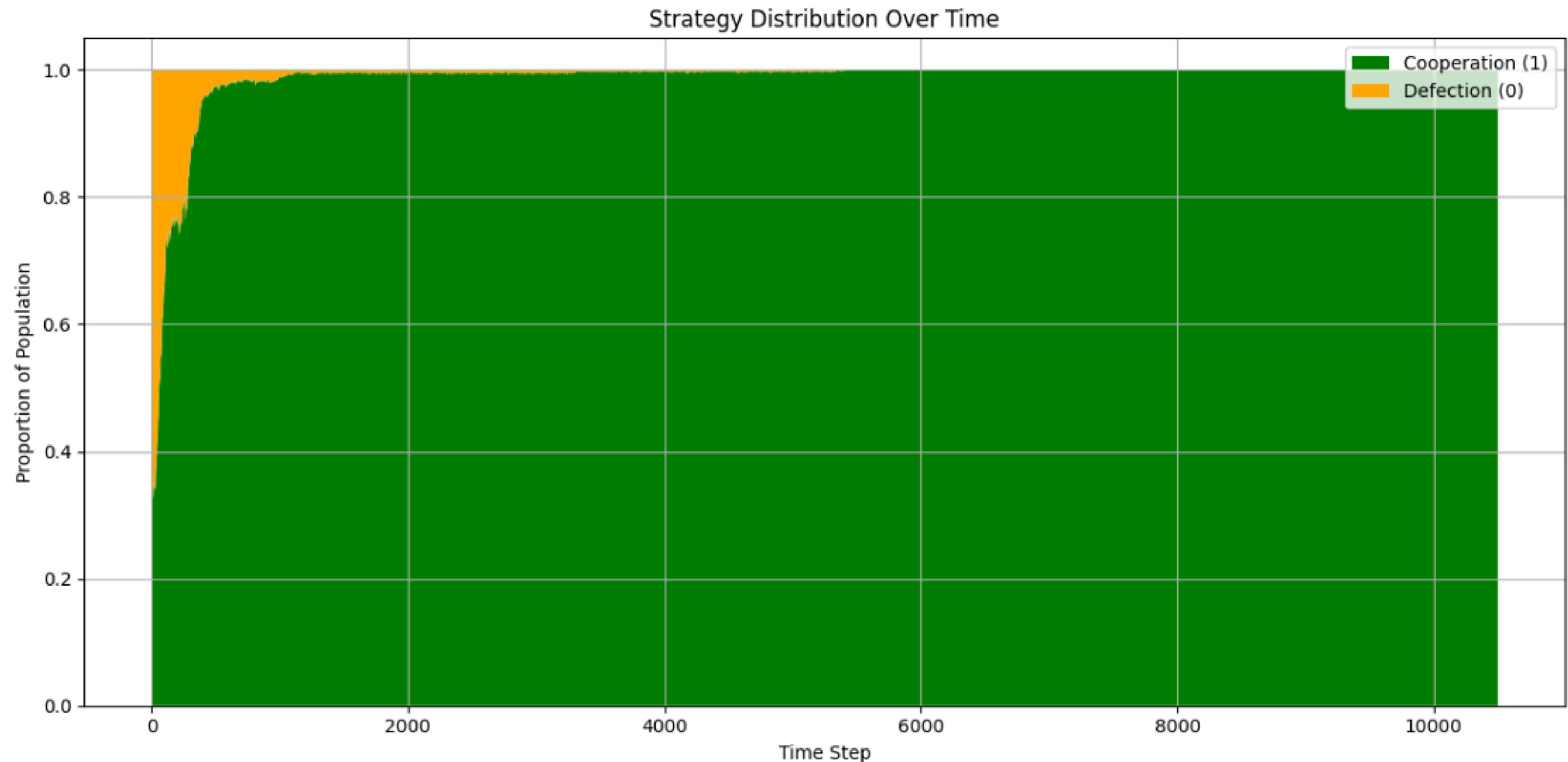
- il parametro decision-rule = Santos Pacheco;
- il parametro play-with = all-nbrs-TOTAL-payoff
- la matrice dei payoff =[[0, 1.1875], [-0.1875, 1]];
- m=0.5;



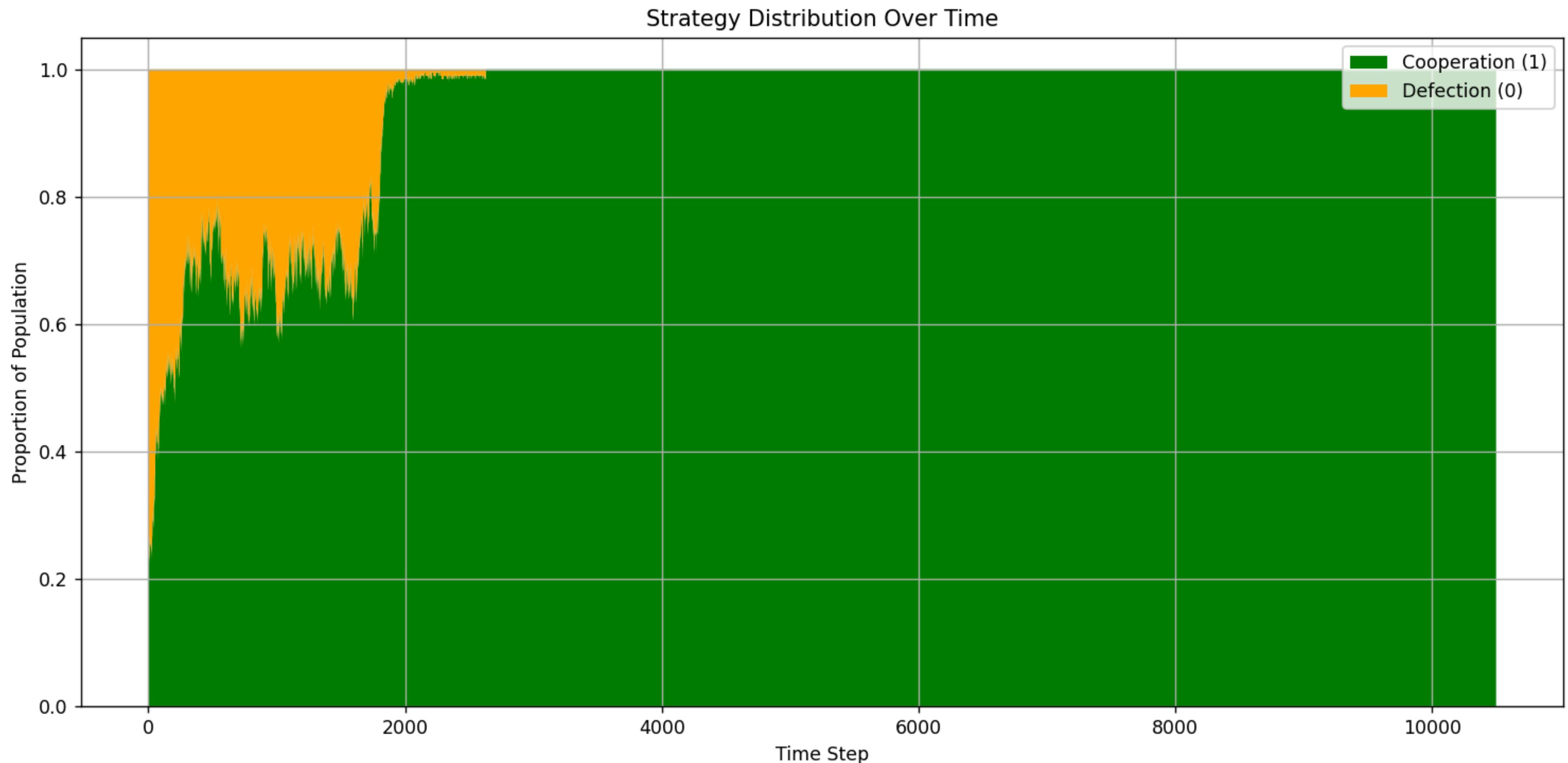
Nonostante un'introduzione molto piccola di quantità di rumore (0.01), le scelte cooperative degli agenti sono cambiate notevolmente.

## 4. Variazione del numero di agenti (n\_players)

`n_players = 1000`



`n_players = 200`

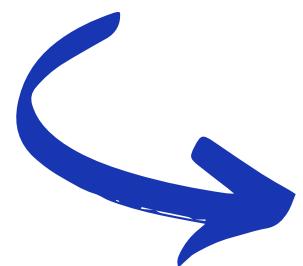




# Evidenze:

Posto:

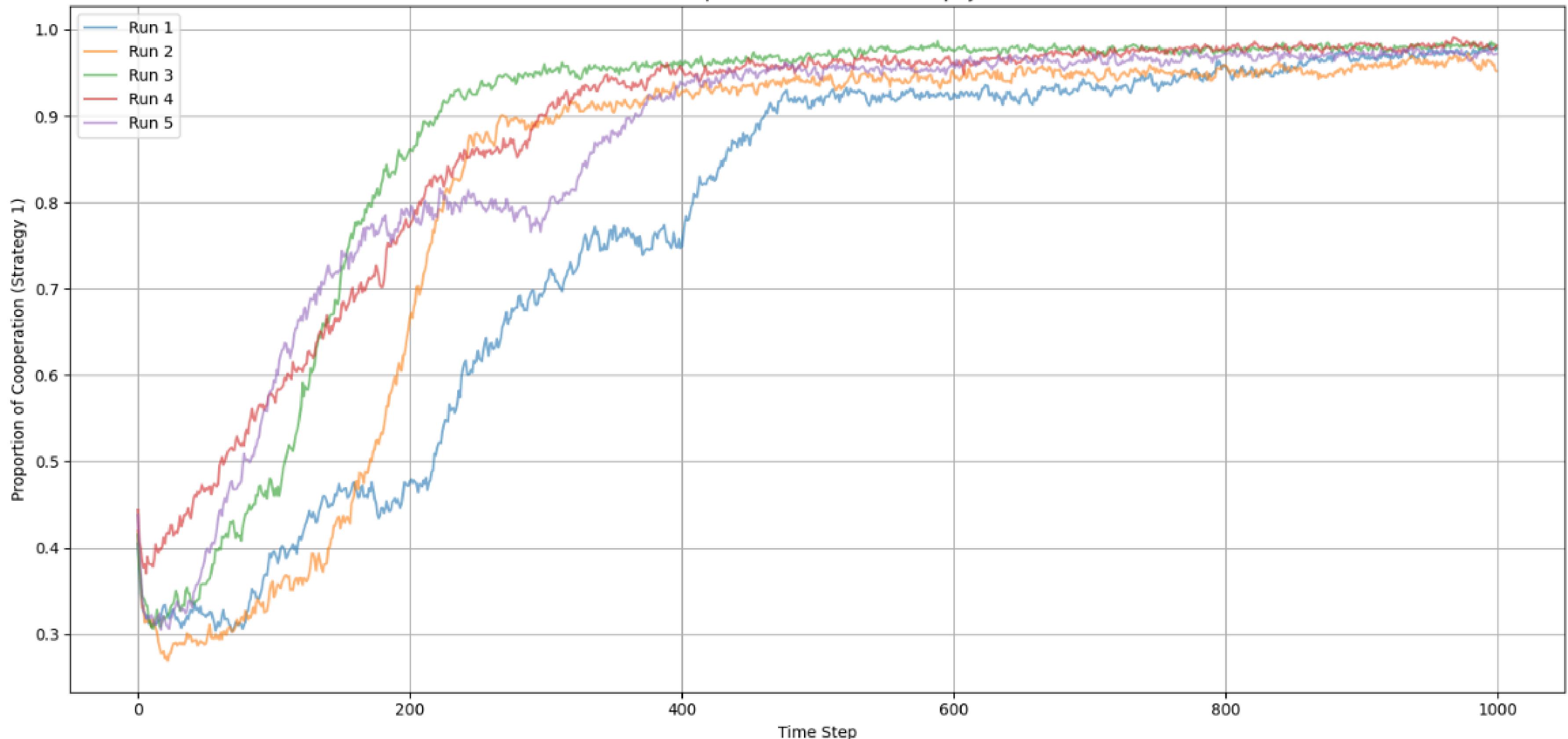
- il parametro decision-rule = Santos Pacheco;
- il parametro play-with = all-nbrs-TOTAL-payoff;
- la matrice dei payoff =[[0, 1.1875], [-0.1875, 1]];
- $m=0.5$ ;
- noise=0.0.



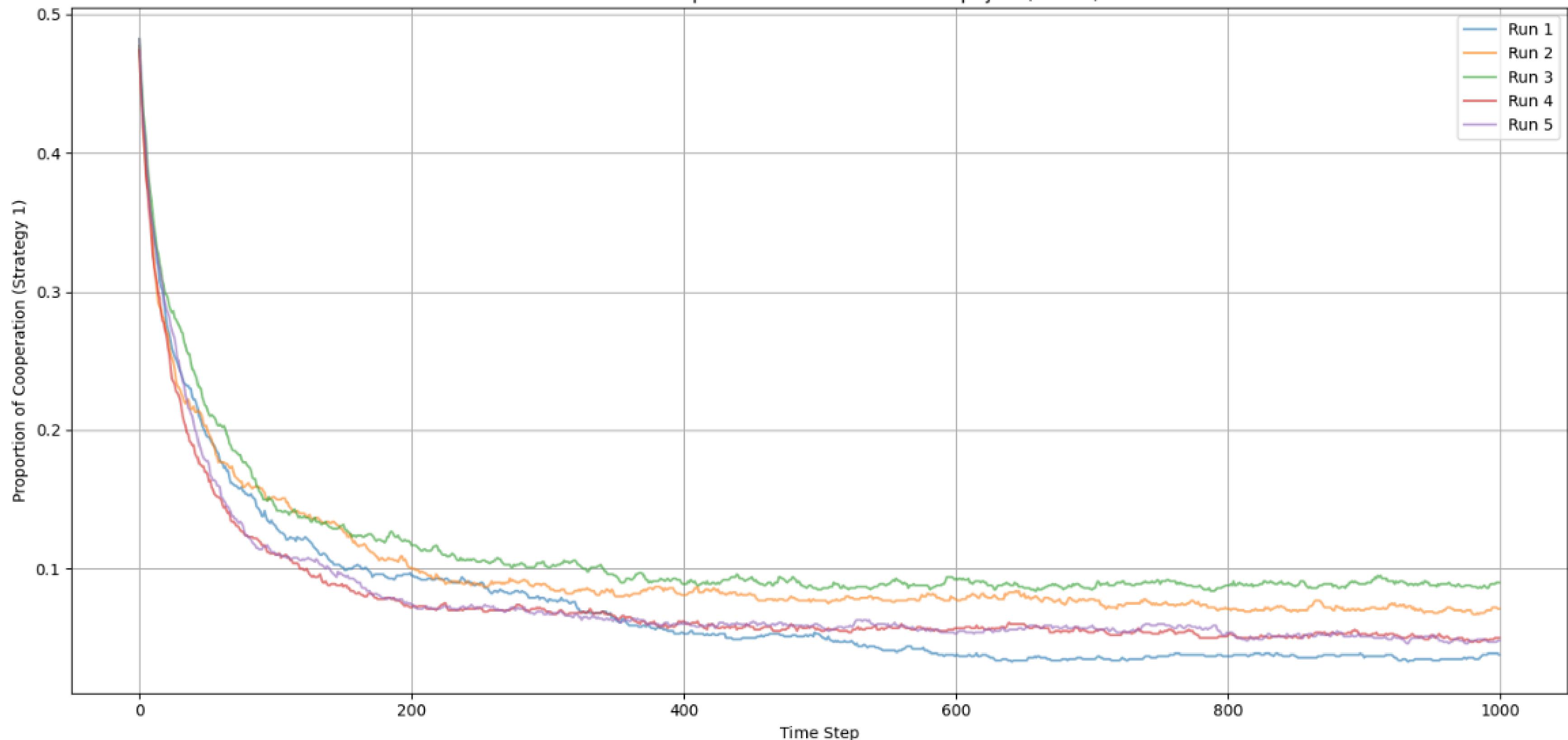
Al diminuire del numero degli agenti, la cooperazione tende a diminuire (anche se di poco).

## **5 . Analisi della sensibilità rispetto alla modalità di interazione ('play\_with')**

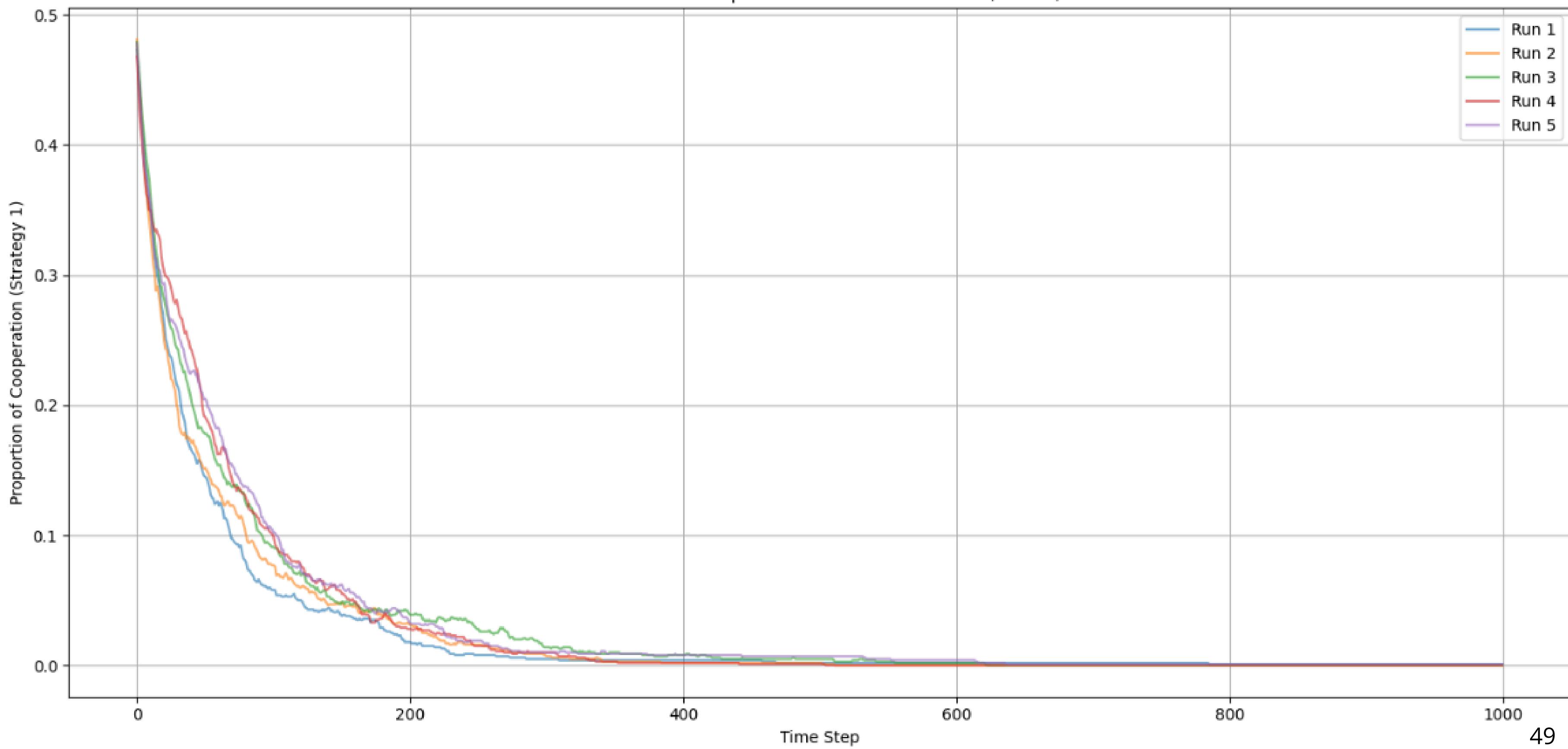
Evolution of Cooperation - all-nbrs-TOTAL-payoff (5 runs)



Evolution of Cooperation - all-nbrs-AVERAGE-payoff (5 runs)



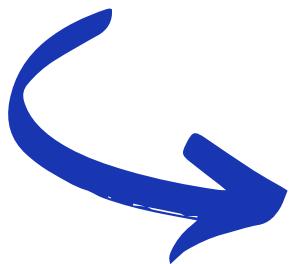
### Evolution of Cooperation - one-random-nbr (5 runs)





# Evidenze: (modifica)

Volendo studiare la sensibilità rispetto alla modalità di interazione tra agenti (`play_with`) che seguono la regola di aggiornamento Santos Pacheco → dopo aver ripetuto 5 simulazioni per ciascuna configurazione, si può osservare che:



Nel tempo, la modalità di interazione “all-nbrs-TOTAL-payoff” risulta essere quella con la strategia collaborativa più elevata. Di contro, la modalità di gioco “one-random nbr” risulta essere quella meno collaborativa.



# Considerazioni finali

Per la regola di aggiornamento proposta da Santos e Pacheco:

- ◆ Se il payoff è la somma delle interazioni con tutti i vicini (play with = all-nbrs-TOTAL-payoff) la cooperazione tende a crescere e a mantenersi su livelli molto alti.
- ◆ Aumentare di troppo il valore del parametro di intensità  $m$  non porta a maggiore cooperazione: la cooperazione più alta, infatti, si ha quando  $m$  è un valore medio (0.5).
- ◆ Aggiungere una piccola quantità di rumore (es.  $noise = 0.01$ ) riduce significativamente la cooperazione. Anche un rumore basso introduce incertezza nelle decisioni degli agenti, impedendo la formazione e il consolidamento stabile dei cluster cooperativi. Pertanto, se aggiungiamo una bassa quantità di rumore il modello più cooperativo sarà quello con il minor numero di step.
- ◆ Paragonando le tre diverse modalità di interazione, e ripetendo 5 simulazioni per ciascuna configurazione, la modalità di interazione “all-nbrs-TOTAL-payoff” risulta essere quella con la strategia collaborativa più elevata



Grazie per  
l'attenzione!