



Nxn-imitate-if- better- noise.nlogo

Metodologie e Tecniche di Simulazione - **GEPID**


A.a. 2024/25

Giada Sechi

Noise and initial conditions

Questo modello è un'estensione del precedente nxn-imitate-if-better e permette:

- Di **configurare esplicitamente** le condizioni di partenza
- Si può inserire la possibilità che un agente riveda la strategia in maniera **casuale** (**estremamente bassa**)



**Al modello sasso, carta forbici
viene aggiunto del rumore**

Che cosa si
intende con il
termine **rumore**?

```
#parametri  
WIDTH = 50  
HEIGHT = 50  
STEPS = 100  
PROB_REVISION = 0.5  
NOISE = 0.05
```

Il valore del rumore viene **scelto manualmente** e inserito nei parametri

```
class RPSAgent:  
    def __init__(self, unique_id, model, strategy):  
        self.unique_id = unique_id  
        self.model = model  
        self.strategy = strategy  
        self.strategy_after_revision = strategy  
        self.payoff = 0.0  
        self.pos = None  
  
    def step(self):  
        mate = random.choice([  
            agent for agent in self.model.schedule.agents if agent != self  
        ])  
        self.payoff = self.model.payoff_matrix[self.strategy, mate.strategy]  
  
    def advance(self):  
        if random.random() < PROB_REVISION:  
            if random.random() < NOISE:  
                self.strategy_after_revision = random.randint(0, N_STRATEGIES - 1)  
            else:  
                observed = random.choice([  
                    agent for agent in self.model.schedule.agents if agent != self  
                ])  
                if observed.payoff > self.payoff:  
                    self.strategy_after_revision = observed.strategy  
        self.strategy = self.strategy_after_revision
```

Nel codice il rumore entra in gioco nella **definizione dell'agente** (durante l'advance)

Problemi con il codice

Mesa.time non funziona

Necessità di creare uno scheduler manuale con CustomRandomActivation

```
#scheduler perchè mesa.time non funziona
class CustomRandomActivation:
    def __init__(self, model):
        self.model = model
        self.agents = []

    def add(self, agent):
        self.agents.append(agent)

    def step(self):
        for agent in random.sample(self.agents, len(self.agents)):
            agent.step()
        for agent in self.agents:
            agent.advance()
```

Mantiene una lista degli agenti

Quando il modello fa step richiama

- step() → sceglie a caso un avversario e dopo aver giocato calcola il payoff
- advance() → con la probabilità di prob_revision l'agente decide se cambiare strategia e poi si introduce il rumore

Problemi con il codice

Creazione di uno slider

Problemi nella creazione di uno slider su Visual Studio, si risolve inserendo nel codice un blocco dove è possibile modificare manualmente i parametri

```
def update(frame):  
    #rumore a 0.001  
    if frame == 30:  
        model.noise = 0.001  
  
    model.step()  
    dist = model.get_distribution()
```

Nella creazione del grafico, in corrispondenza di un certo frame, viene aggiunto del rumore (anche minimo)

Descrizione del modello

- Un numero n di players gioca ripetutamente in modo simmetrico 2vs2 con diverse strategie possibili scelte dagli agentuì
- Il payoff è definito dall'utente nella matrice iniziale
- Può essere inserito del rumore

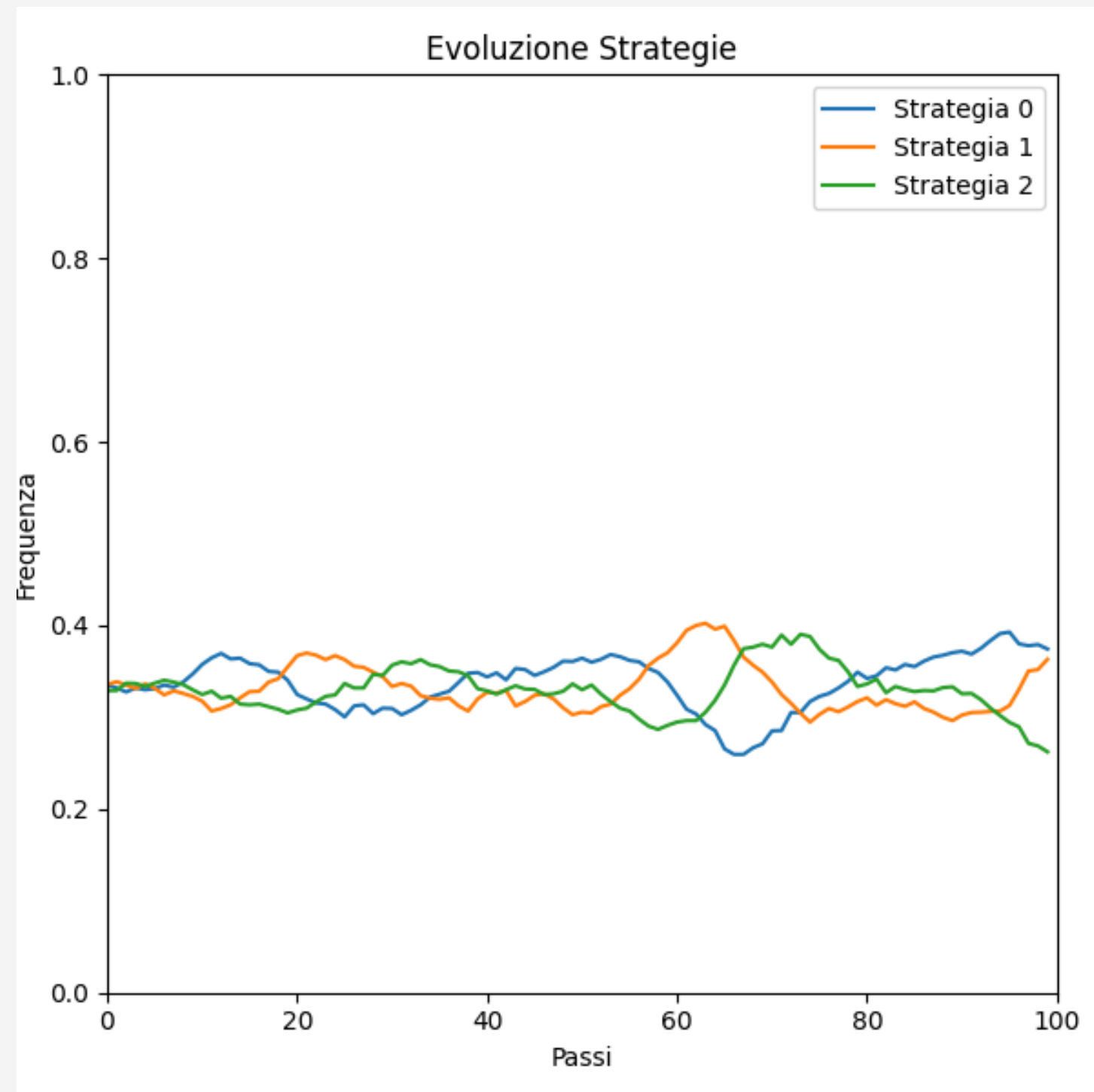
In questo modello:

- **Numero di agenti:** 2500
- **Distribuzione iniziale di strategie:** equa
- **Matrice di payoff:**
$$\begin{Bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{Bmatrix}$$
- **prob_revision:** 0.5
- **Rumore:** 0.05

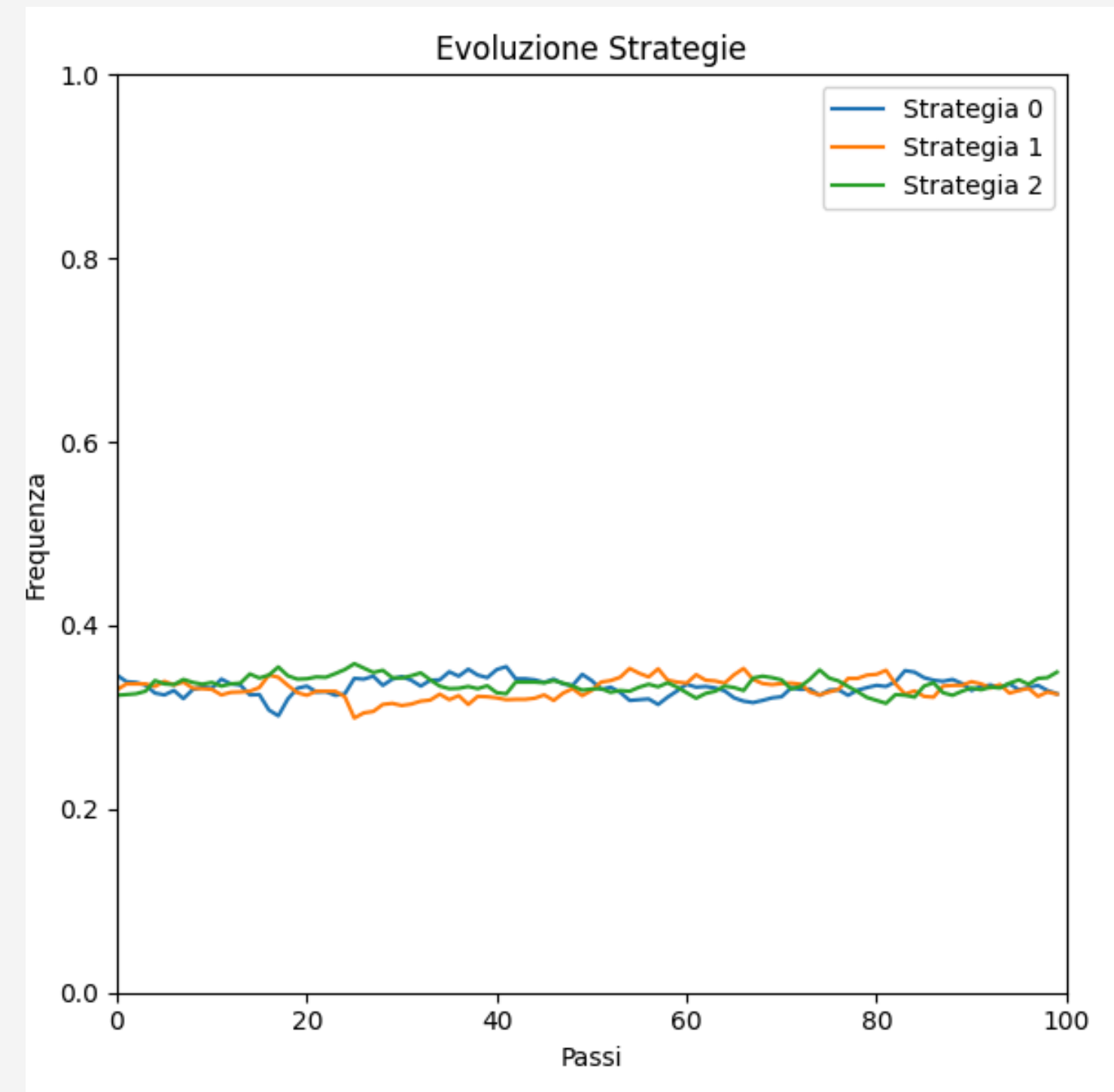
Distribuzione strategie

Nxn-imitate-if-better-noise.nlogo

Rumore = 0.01



Rumore = 0.2



Esercizio 1

Considerando il **Dilemma del Prigioniero** con un payoff $[[2\ 4][1\ 3]]$ dove la strategia 0 è “Defect” e la strategia 1 è “Cooperate”. Settare *prob-revision* a 0.1 e il rumore a 0. Settare il numero iniziale di giocatori che usano la ogni strategia a $[0\ 200]$, (tutti giocano con “Cooperate”). Mentre il modello è in esecuzione introdurre una quantità minima di rumore. Spiegare cosa succede.

S0 → Defezione → 0 agenti

S1 → Collaborazione → 200 agent

Prob-revision → 0.1

Rumore iniziale → 0

Rumore → 0.02 (inserito al frame 10)

Payoff:

$\begin{Bmatrix} 2 & 4 \\ 1 & 3 \end{Bmatrix}$

Che cosa succede?

Con rumore = 0

Gli agenti sono tutti cooperanti (strategia = 1) e siccome non c'è rumore non cambiano strategia casualmente

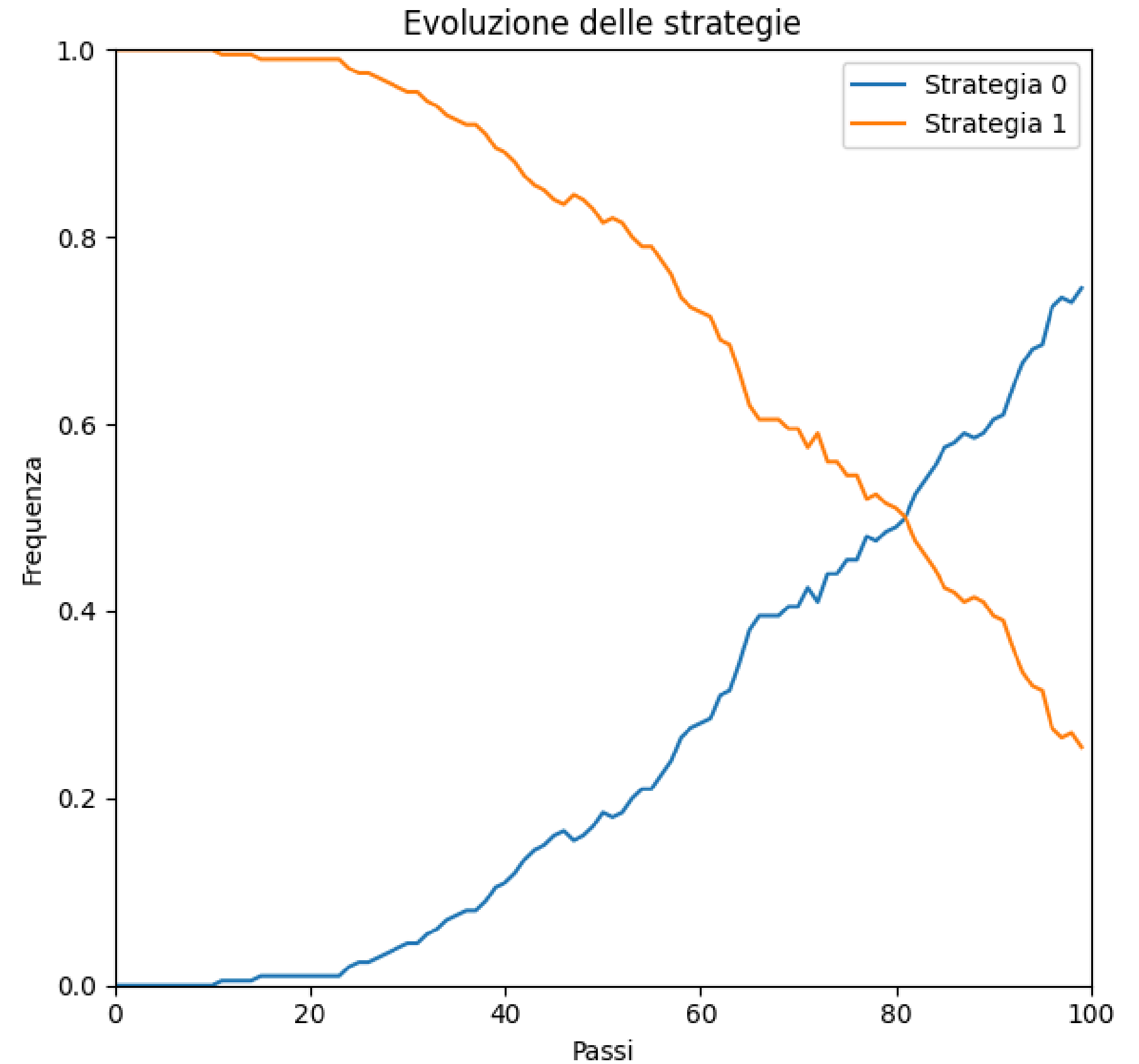
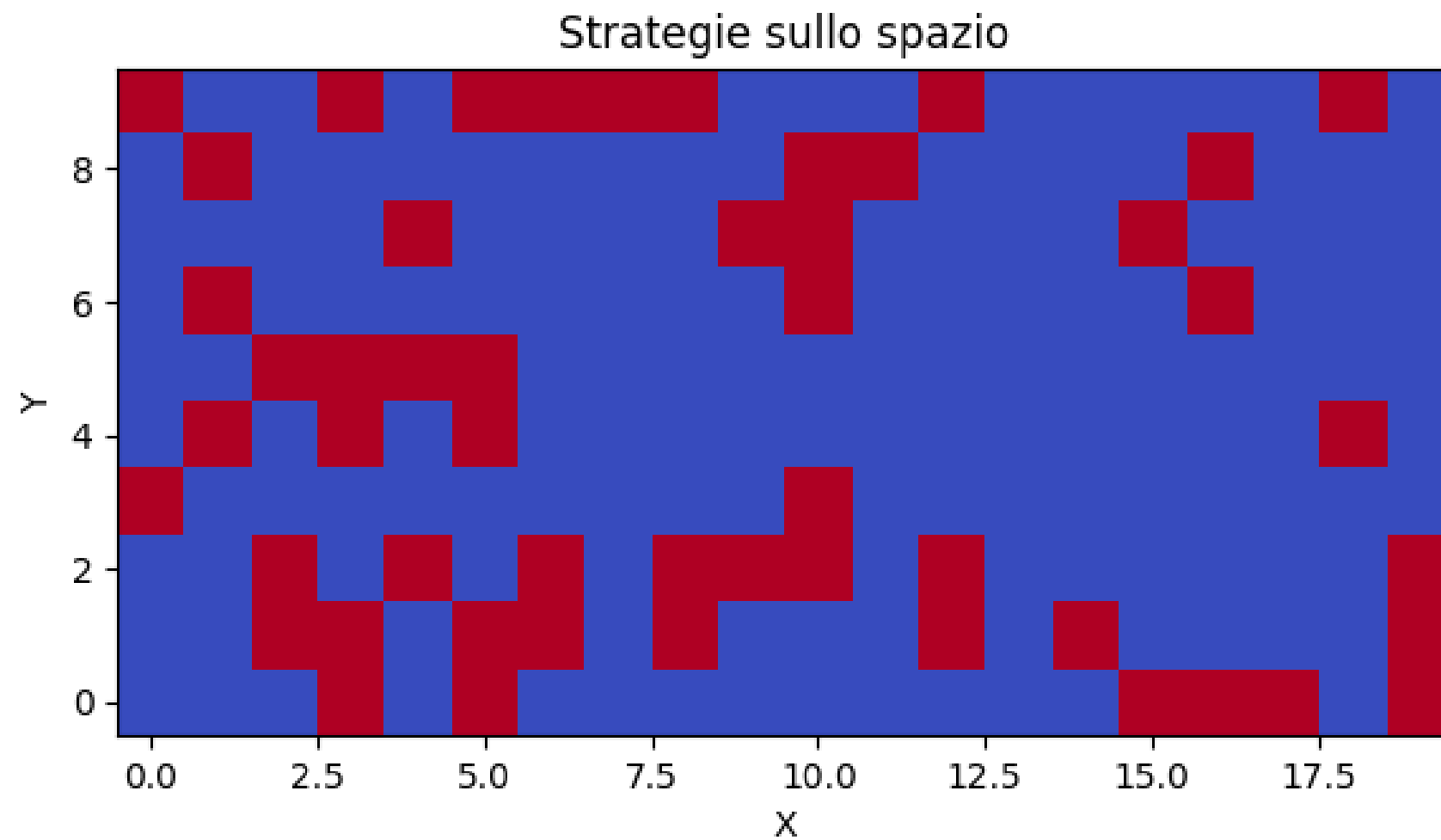
Gli agenti hanno tutti lo stesso payoff e non sono incentivati a cambiare strategia:
100% cooperazione

Con rumore > 0

Con l'introduzione del rumore alcuni agenti cambiano casualmente strategia (strategia = 0)

Gli agenti con $s=0$ hanno un payoff maggiore quando incontrano $s=1$ e gli altri agenti cominciano ad imitarli: c'è un declino di cooperazione

Grafici esercizio 1



Esercizio 2

Considerando uno scenario **Sasso Carta Forbici** con un payoff $\begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$. Settare prob-revision a 0.1 e il rumore a 0. Settare il numero iniziale di giocatori che usano ciascuna strategia (n-of-players-for-each-strategy) a [100 100 100]. Mentre il modello è in azione modificare il rumore a 0.001. Che cosa succede?

S0 → Sasso → 100 agenti

S1 → Carta → 100 agenti

S2 → Forbici → 100 agenti

Prob-revision → 0.1

Rumore iniziale → 0

Rumore → 0.01

Payoff:

$\begin{Bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{Bmatrix}$

Che cosa succede?

Con rumore = 0

Le tre strategie sono inizialmente equilibrate e gli agenti non cambiano strategia casualmente

Il gioco si evolve in modo equilibrato

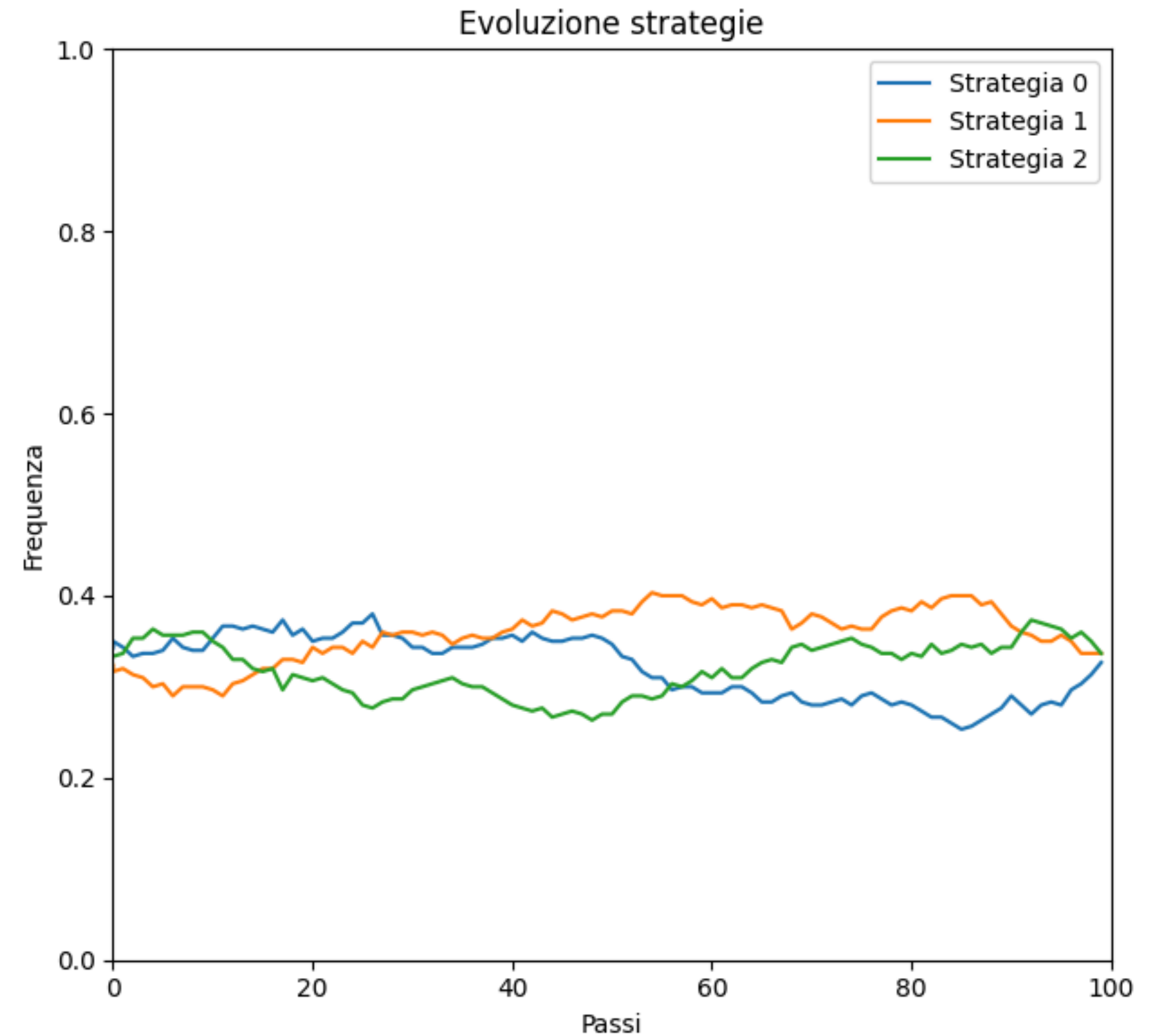
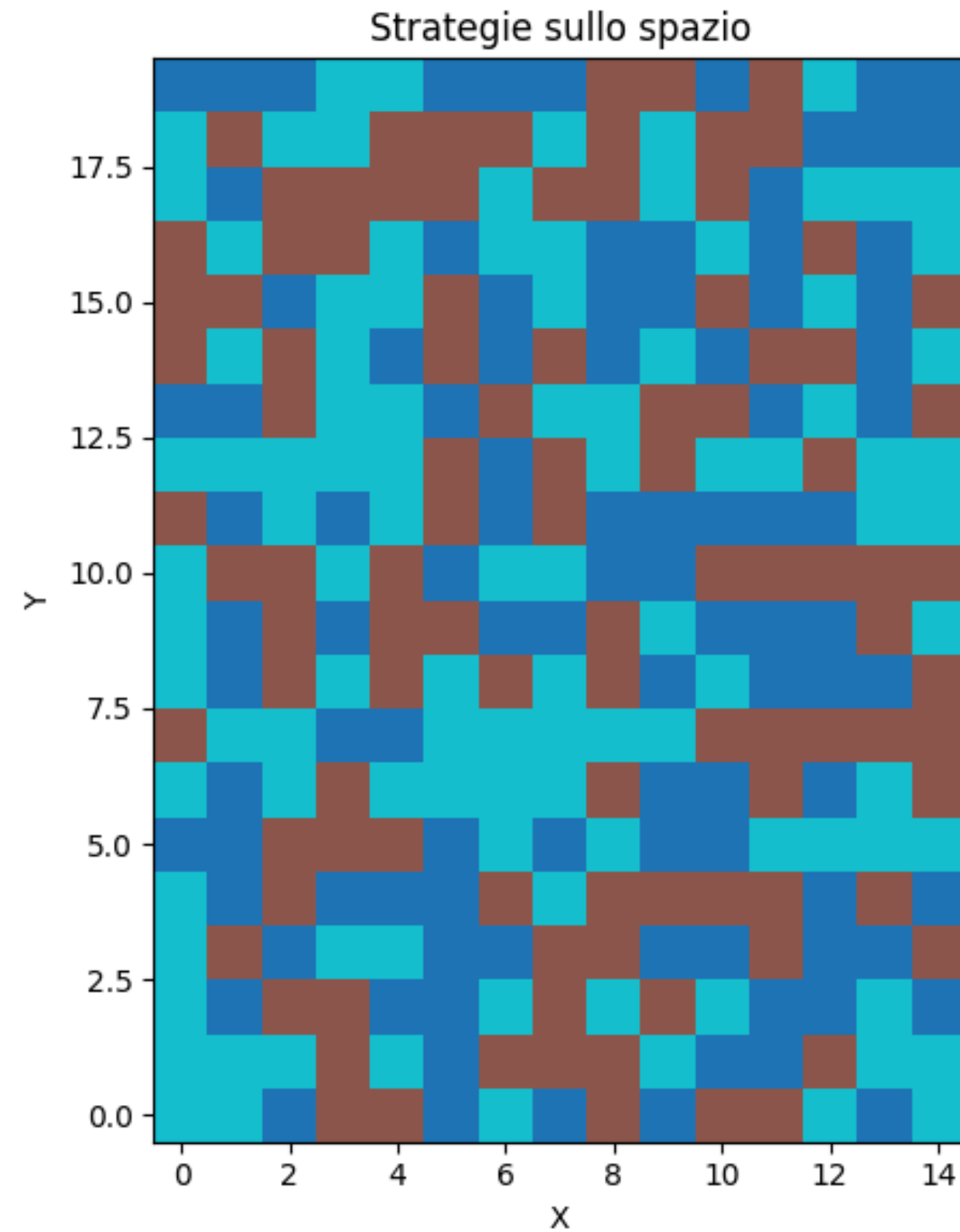
Con rumore = 0.001

Con l'introduzione del rumore alcuni agenti cambiano casualmente strategia casualmente

C'è più fluttuazione nelle strategie, in questo caso la strategia 1 sembra emergere con più successo

Grafici esercizio 2

Nxn-imitate-if-better-noise.nlogo



Esercizio 3

Considerando un gioco con un payoff $\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$. Settare la prob-revision a 0.1, e il rumore a 0.05, e il numero iniziale di giocatori che adottano ciascuna strategia a $[500 \ 0 \ 500]$. Mentre il modello è in azione cambia i parametri iniziali. Spiega cosa succede.

S0 → 500 agenti

S1 → 0 agenti

S2 → 500 agenti

Prob-revision → 0.1

Rumore → 0.05

Payoff:

$\begin{Bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{Bmatrix}$

Che cosa succede?

Prima del reset

La strategia 1 inizialmente non esiste, ma grazie al rumore viene adottata casualmente da alcuni agenti che cominciano ad ottenere payoff migliori

Sempre più agenti adottano la strategia 1 perchè imitano quelli con il payoff più alto



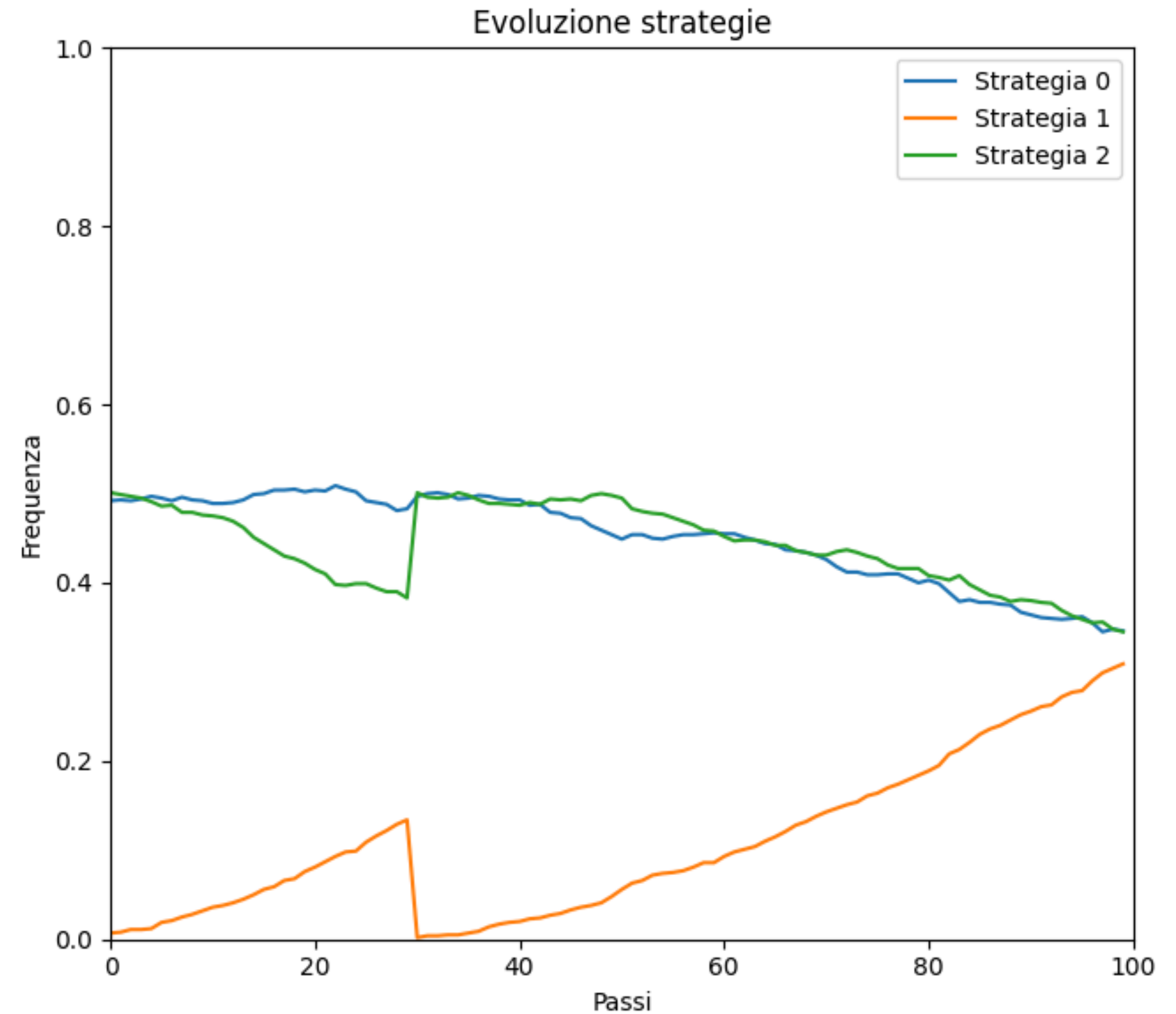
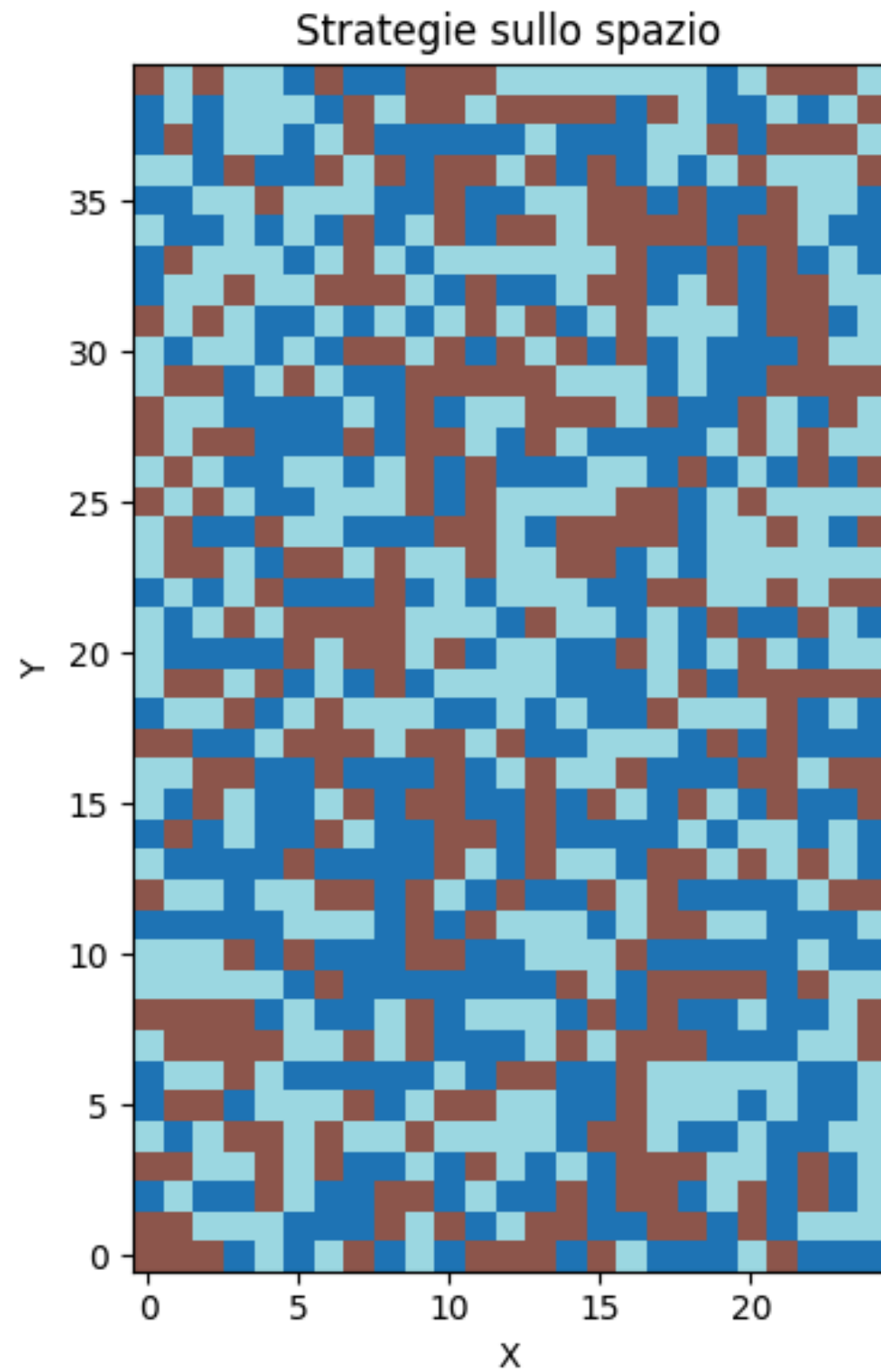
Dopo il reset

Il modello si riattiva con nuove situazioni iniziali ma mantenendo le stesse percentuali di distribuzione degli agenti

Il processo si verifica come prima, con la strategia 1 che viene adottata sempre da più agenti



Grafici esercizio 3



Conclusioni



Anche in una situazione di perfetta collaborazione il rumore agisce come disturbatore in grado di innescare processi competitivi



Il rumore è quindi in grado di mobilitare una situazione statica, dando il via ai processi di imitazione degli agenti

Grazie per l'attenzione!

Metodologie e Tecniche di Simulazione - **GEPID**

A.a. 2024/25

Giada Sechi