Aristotle University of Thessaloniki

Computational Mathematics II

Spatial Prisoner's Dilemma

Zachariadou Anastasia

September 2024

Abstract

The aim of this project was to explore the dynamics of evolutionary games and spatial chaos through simulating a repeated Prisoner's Dilemma game in a spatial grid. Using a deterministic model, our study demonstrated how chaotic patterns can emerge from simple rules and highlighted the critical dependency of the system's dynamic evolution to the payoff parameter b.To achieve this, a Python program was developed through which the findings presented in the Nature article "Evolutionary Games and Spatial Chaos", by M. A. Nowak and R. M. May [3], were replicated and further analyzed.

Contents

Chapter 1

Introduction

1.1 Evolutionary Game Theory

Game theory is defined as the study of mathematical models of conflict and cooperation between rational decision-makers [4]. Players make decisions based on maximizing their "gain". It is assumed that the game is only played once, with no future player interactions, so there is no need for consideration of the long-term consequences of one's selected strategy. **Evolutionary game theory** extends classical game theory by incorporating the concept of evolution. It involves **repeated interactions** and studies how strategies evolve and spread among populations based on their success.

In Evolutionary Game theory, the concept of payoff is often referred to as **"fitness"** [2]. The fitness of individuals is not constant, but **frequency dependent**, changing based on the relative abundance of the different **phenotypes** in the population (the distinct strategies of players): **successful strategies reproduce faster**.

Evolutionary game dynamics can be studied in a spatial context, a subcategory known as spatial games. Our aim is to simulate such a game and study the dynamic behavior of its players' population. More context on the general idea behind spatial games and its implementation on the present work will be given in following sections.

1.2 Key Concepts behind our Setup

1.2.1 The Prisoner's Dilemma

One of the most studied games in evolutionary game theory is the **Prisoner's Dilemma** , as it can serve as a metaphor for cooperation scenarios in real life. In this game, two players are given two choices, either **cooperate (C)** or **defect**

(D) [1]. Each must make the choice without knowledge of the other player's decision. No matter what the other does, defection yields higher payoff than cooperation. The dilemma arises from the fact that if both players decided to defect, both receive bigger penalties than if both had decided to cooperate. The outcomes are represented by the following matrix (known as a **payoff matrix**):

$$\begin{array}{c|cc} & \text{C} & \text{D} \\ \hline \text{C} & R, R & S, T \\ \text{D} & T, S & P, P \end{array}$$

The conclusion: individual rationality (choosing to defect) leads to worse outcome for both (higher penalty if both defect) than is possible. Everyone is better off with mutual cooperation.

The general idea of this game is the inspiration behind the setup of our implementation. Our study however is concerned with a spatial game involving repeated interactions.

- R is the reward for mutual cooperation,
- S is the sucker's payoff (when one cooperates while the other defects),
- T is the temptation to defect (when one defects while the other cooperates),
- P is the punishment for mutual defection.

In the context of evolutionary game theory, the focus is on how these payoffs influence the evolution of strategies in a population.

1.2.2 Spatial Games

In spatial evolutionary games, players are located on a grid or **lattice**, and they interact with their **neighbors** [2]. This setup allows for the exploration of spatial patterns and the effects of local interactions on the global evolution of strategies and consequently the population balance. The spatial version of the Prisoner's Dilemma, which is the focus of this study, can exhibit complex behaviors such as the emergence of cooperation or **spatial chaos**. In the context of evolutionary games, spatial chaos manifests as irregular patterns and fluctuations in the distribution of strategies over time.

Chapter 2

Implementation

2.1 Study Setup

In our study, we created a deterministic, **spatial version of the Prisoner's Dilemma** in an attempt to analyze the mechanism of spatial reciprocity and investigate the **dynamic behavior** of such a system. The process implemented was based on the 1992 work of Martin A. Nowak and Robert M. May: "**Evolutionary Games and Spatial Chaos**"[3] and the mathematical setup was strongly influenced by ideas presented in Nowak's book: "Evolutionary Games and Population Dynamics"[2]. Our setup considered only two kinds of players (agents):

1. **C: players who always cooperate**
2. **D: players who always defect**

The players were placed in an $\mathbf{n} \times \mathbf{n}$ fixed-boundary **lattice**, with each one occupying a lattice site (point in the grid). At every round, (generation) each player participated in Prisoner's Dilemma game with its **8 immediate neighbors** (self-interaction was included as well). In our implementation, the payoff values were initialized as:

- **R=1** (reward for mutual cooperation)
- **S=0** (sucker's payoff)
- **T=b** (temptation payoff)
- **P=0** (punishment for mutual defection)

Presented in **payoff matrix** form (1.2.1):

$$\textbf{Payoff} = \begin{bmatrix} 1 & 0 \\ b & 0 \end{bmatrix}$$

Each player's total score was calculated as the sum of the individual game payoffs of the current round with its neighbors. At every new generation's start, each site was occupied either by the original owner, or by one of the neighbors, depending on **who scored highest**.

The way the simulation was set, **b**, which represents the incentive to defect (advantage of defectors against cooperators), is the only parameter in the model. Thus, just by altering its value, it was possible to observe **different dynamical states** of the system. Section 2.3 contains a comprehensive explanation of the way this was achieved computationally.

2.2 Theoretical Basis

As mentioned in the previous section, the dynamical behavior of the system depends on b. It has been shown that **distinct classes of b** regions exist that cause **different dynamical states** to exist.

- **8-Neighbor + self-interaction** [3]:

  1. $b > 1.8$: large D clusters continue to grow
  2. $b < 1.8$: D clusters shrink
  3. $b < 2$: large C clusters continue to grow
  4. $b > 2$: C clusters do not grow
  5. $1.8 < b < 2$ (**interesting region**): Both C and D clusters can keep growing. The result is **dynamic chaos** when it comes to patterns. For the **case of defector invasion** specifically, it can be proven that the frequency of cooperators is almost constant, oscillating around the **limit of** $f_C = 0.318$ (dynamic equilibrium).

- **8-Neighbor interaction (no self)** [2]:

  1. $b > 1.667$: Only D clusters can keep growing
  2. $b < 1.6$: Only C clusters can keep growing
  3. $1.6 < b < 1.667$ (**interesting region**): Both C and D clusters can keep growing. Only difference with the self-interaction case is that the **theoretical is** $f_C = 0.299$ **for the case of defector invasion.**

2.3 Code Implementation

As mentioned in 2.1, a Python program was developed to simulate the described repeated Prisoner's Dilemma. The **asymptotic behavior of the system** was studied for a **variety of b values** and **different initial proportions**

of Cooperators (C) and Defectors (D) placed at random on a square lattice. As the dynamics of the system are critically dependent on b, it was expected that there will be **a series of discrete transition values of b that lead from one regime to another**, due to the discrete nature of the possible payoffs [3]. In this section, the simulation process is explained step by step, accompanied with the relevant code snippets.

Simulation Function:

The function responsible for creating the game simulation takes 9 parameters as inputs:

1. **b**: Defection payoff
2. **p**: Defectors' proportion $0 <= p <= 1$
3. **self_interaction**: Controls whether cell interacts with itself or not
4. **gens**: Total number of generations (game rounds)
5. **n**: Lattice dimensions
6. **condition**: Controls initial lattice configuration
7. **evolution_plot**: Displays evolution plot
8. **fc_plot**: Displays frequency of cooperators plot
9. **theoretical_limit**: Sets theoretical limit for final frequency of cooperators and displays it on fc_plot

```
def spatial_chaos_8_nn(b, p, self_interaction, gens, n, condition,
    evolution_plot, fc_plot, theoretical_limit):
```

Step 1. Parameter Initialization

The code begins by initializing the **payoff parameters** with the selected setup values (2.1). Arrays to store payoffs, transitions and strategies are also initialized. Their relevancy and use will be discussed later.

```
# Initializing parameters
nn = n * n  # Total number of cells (n x n grid)
R, S, T, P = 1, 0, b, 0 # Payoff values
payoff_matrix = np.array([[R, S], [T, P]])
```

Step 2. Lattice Initialization

The **spatial grid** is initialized with each cell (point) representing an agent (player), either a cooperator ($C = 1$) or a defector ($D = 2$). **Two distinct initial conditions** are considered which define the **initial configuration of players** on the grid:

1. **condition=1**: The lattice is initialized with all agents as cooperators (C), except for a **single defector (D)** placed in the center.
2. **Otherwise**: The grid is **randomly populated** with defectors with probability (proportion) **p**. The proportion of cooperators will be $1 - p$

This corresponds to the following code:

```python
if condition == 1:
    lattice[n // 2, n // 2] = 2
else:
    lattice_random = np.random.rand(n, n)
    lattice[lattice_random < p] = 2
```

Step 3. Game

Each round, every agent (point in the grid) plays the game with itself and its 8 neighbors, scoring points from each interaction. The round's final payoffs determine the type of agent that will inhabit the current point in the next round.

Payoff Calculation for each player

The **round's payoff** for each agent is calculated based on points scored from interactions within the Moore neighborhood. The total payoff for an agent at position (i, j) is given by:

$$\text{payoff}[i,j] = \sum_{\Delta i=-1}^{1} \sum_{\Delta j=-1}^{1} \text{payoff\_matrix}\Big[\text{lattice}[i,j]-1, \text{lattice}[i+\Delta i, j+\Delta j]-1\Big] \tag{2.1}$$

where the sums run over the 8 neighbors of the agent. If it is specified, self-interaction can be excluded by skipping the case where $\Delta i = 0$ and $\Delta j = 0$.

```python
for row in range(n):
    for col in range(n):
        temp_payoff = 0

        # Check neighboring cells for payoff calculation
        for delta_row in range(-1, 2):
            for delta_col in range(-1, 2):
                # Skip self-interaction if specified
                if self_interaction== 0 and delta_row== 0 and delta_col== 0:
                    continue

                neighbor_row, neighbor_col = row + delta_row, col + delta_col

                # Ensure neighbor is within bounds
                if neighbor_row < 0 or neighbor_row >= n or neighbor_col < 0 or neighbor_col >= n:
                    continue

                # Accumulate the payoff from the neighbor
                temp_payoff += payoff_matrix[int(lattice[row, col]) - 1, int(lattice[neighbor_row, neighbor_col]) - 1]

        # Assign the calculated payoff
        payoff[row, col] = temp_payoff
```

Strategy Update

After calculating the payoffs, each player's **strategy is updated** its by adopting the strategy of the neighboring player with the **highest payoff**. If none of them a higher payoff, the current strategy is retained. Mathematically, the new strategy for an agent at position (i, j) is:

$$\text{strategy}[i, j] = \arg\max_{k,l} (\text{payoff}[i + k, j + l]) \tag{2.2}$$

This part is a represenation of the idea presented earlier in 1.1 (spread of successful strategies).

```
# Updating each cell's strategy based on the highest neighboring payoff
        for row in range(n):
            for col in range(n):
                max_payoff= payoff[row, col] # Start with cell's own payoff
                strategy= lattice[row, col] # Start with cell's own strategy
                for delta_row in range(-1, 2):
                    for delta_col in range(-1, 2):
                        # Case: no self-interaction
                        if self_interaction==0 and delta_row==0 and delta_col==0:
                            continue
                        # Calculating neighbor's coordinates
                        neighbor_row, neighbor_col = row + delta_row, col + delta_col
                        # Skipping if neighbor out of bounds
                        if neighbor_row < 0 or neighbor_row >= n or neighbor_col < 0 or neighbor_col >= n:
                            continue
                        # If neighbor's payoff is higher, adopt their strategy
                        if payoff[neighbor_row, neighbor_col] > max_payoff:
                            max_payoff = payoff[neighbor_row, neighbor_col]
                            strategy = lattice[neighbor_row, neighbor_col]
                new_lattice[row, col] = strategy  # Updating  strategy
```

Step 4. Post-Game

At the **end of each generation**, the **lattice is updated** appropriately for the start of the next. The current **proportion of cooperators** is also kept track of, in order to compare it with the theoretically calculated value at the end.

Transition Tracking and Lattice Update

Strategy changes are recorded during the generations using a transition matrix initialized at the beginning of the code. **Transitions** are encoded with numbers 1-4 in the following way:

- 1: Cooperation to Cooperation
- 2: Defection to Defection
- 3: Defection to Cooperation
- 4: Cooperation to Defection

```
# Transition matrix to track changes in strategies (1: C to C, 2: D to D, 3: D to C, 4: C to D)
    transitions = np.array([[1, 3], [2, 4]])
```

Comparing the agent's initial and new strategies, the transition matrix is updated accordingly and finally the current point is prepared for the next generation by obtaining a **new owner**.

```
# Updating the lattice and recording the transitions
        for row in range(n):
            for col in range(n):
                # Recording the transition type (C to C, D to D, D to C or C to D)
                transition_matrix[row, col, generation] = transitions[int(new_lattice[row, col]) - 1, int(lattice[row, col]) - 1]
                lattice[row, col] = new_lattice[row, col]  # Updating the lattice for the next generation
```

Cooperator Frequency Calculation

The last step in the loop is calculating the fraction of the lattice cells occupied by Cooperators at the end of the current generation. This is a key value for observing the dynamics of cooperation within the population. Mathematically, the **frequency of cooperators** is determined by:

$$f_C(g) = \frac{\sum_i \sum_j \delta\left(\text{transition\_matrix}[i,j,g],1\right) + \delta\left(\text{transition\_matrix}[i,j,g],3\right)}{n^2} \tag{2.3}$$

where $\delta(x,y)$ is the Kronecker delta, which equals 1 if $x = y$ and 0 otherwise. This way, only transitions leading to cooperation (1 or 3) are taken into account within the sum. This translates to code logic as:

```
# Calculating the cooperators' frequency (fraction of cells occupied by C)
        fc[gen] = np.sum((transition_matrix[:, :, gen]== 1) | (
    transition_matrix[:, :, gen]== 3)) /nn
```

Step 5. Visualization

The final section of the code is dedicated to the visualization of the **evolution of the system**.

Evolution Plot

The **dynamic evolution** of the lattice throughout generations is visualized with an animation of the transition matrix, where each color represents a different type of **strategy transition**. The transitions are color-coded as following:

Transition Type	Color
C → C	**Blue**
D → D	**Red**
D → C	**Green**
C → D	**Yellow**

```
# Lattice evolution animation plot
    if evolution_plot == 1:
        fig, ax = plt.subplots(figsize=(5, 5))
        # Initializing the plot with the first frame
        mat = ax.matshow(transition_matrix[:, :, 0], cmap=color_matrix)
        plt.subplots_adjust(top=0.85)

        # Title
        ax.set_title(f'Spatial Prisoner\'s Dilemma\nb={b}, p={p}, self
    interaction={self_interaction}, theor. lim.={theoretical_limit}', pad=20)
```

```python

        # Function to update the animation for each frame
        def update(frame):

            # Updating the lattice display for the current frame
            mat.set_data(transition_matrix[:, :, frame])
            # Updating the title to reflect the current round and parameters
            ax.set_title(f'Generation: {frame + 1}\n'
                         f'b={b}, p={p}, self interaction={self_interaction}, theor. lim.={theoretical_limit}', pad=20)

            return [mat]

        # Creating the animation (update every 500 ms)
        ani = animation.FuncAnimation(fig, update, frames=gens, interval=500, blit=True)
        ani.save('evolution_animation.gif', writer='pillow', fps=10)

        # Displaying the GIF
        display(Image(filename='evolution_animation.gif'))
```

An **animation plot** is displayed in the style of a gif, showing how cells transition from one strategy to another and subsequently how the **cooperator and defector populations change with time**.

Cooperator Frequency Plot

As a natural next step, the code **plots the frequency of cooperators over time**, comparing it to the theoretical value when provided.

```python
# Frequency of cooperators plot over time
    if fc_plot == 1:
        plt.figure(figsize=(6, 4))
        plt.plot(range(1, gens + 1), fc, '-p', color='purple', markersize=5, markeredgewidth=0.5)
        plt.xlabel('Generation')
        plt.ylabel('Frequency of cooperators (fraction of sites occupied by C)')
        # Title
        plt.title('SPATIAL PRISONER\'S DILEMMA\n'
                  f'b={b}, p={p}, self interaction={self_interaction}, theor. lim.={theoretical_limit}', pad=20)
        plt.grid(True)

        # Case: If the theoretical limit is specified, draw a horizontal line at that value
        if theoretical_limit > 0:
            plt.axhline(y=theoretical_limit, color='k', linestyle='--')
            plt.text(gens, theoretical_limit, str(theoretical_limit))
        plt.show()
```

Chapter 3

Results and Discussion

In this chapter, the results of our code implementation will be showcased. Each section contains the figures resulting from **different simulations**, i.e. for different input parameters in the function we created. More specifically, the **dependency of the system's dynamics** and spread of strategies to the **value of the temptation parameter b** was studied with the help of the plots produced by our program. We chose to focus primarily on the regions of most interest as characterized in 2.2 for two general cases:

1. **Random initial lattice Configuration**: Cooperators and Defectors of specified initial proportions are randomly placed in the grid
2. **Defector Invasion**: A single Defector is placed in the center of a Cooperator-populated grid.

Some extra cases were presented as well, for the sake of exploring different dynamical regimes.
For a better understanding of the dynamic behavior of the systems, it is advised to look for the relevant sections either on the notebook of the code, or the gifs corresponding to each case. All provided here.

3.1 Figures and Results

3.1.1 Random initial configuration: p=0.1

In this part, two simulations are presented, both on a 200×200 square lattice for **200 generations** of the game, but for **different b** parameters. As showcased, as b passes from one value region to the other, there is a complete change in the dynamical regime. The initial proportion of phenotypes is the same in both examples, **10% defectors and 90% cooperators, randomly placed in the grid**.

1.75 < b <1.8, p=0.1

In 3.1 the lattice evolution plot for b values in the region of $1.75 < b < 1.8$ is presented. It is clear that **cooperators dominate**, as the lattice is **mostly blue**. An **irregular and relatively static network of 'D lines'** is observed, as initial D clusters quickly shrink (strategies that don't do well are quickly minimized [2]). This is in agreement with Nowak's remarks about this specific region [3]. Another interesting observation is the lack of green (C to D) and yellow (D to C) points in the final lattice, but also throughout the generations; the number of strategy changes was minimal.

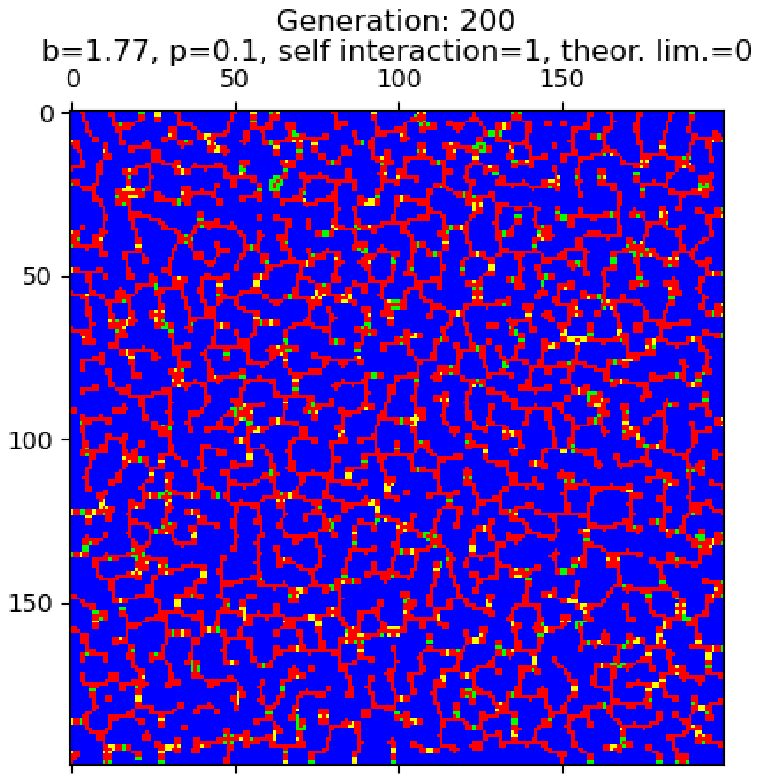

Figure 3.1: Lattice Evolution plot. $1.75 < b < 1.8$, 10% D, 90% C randomly populated. Click here for Evolution gif.

In 3.2, the fast domination of cooperators in the grid is also evident. The frequency of C agents quickly reaches equilibrium with a proportion close to **75%** (even before 30 rounds are completed).

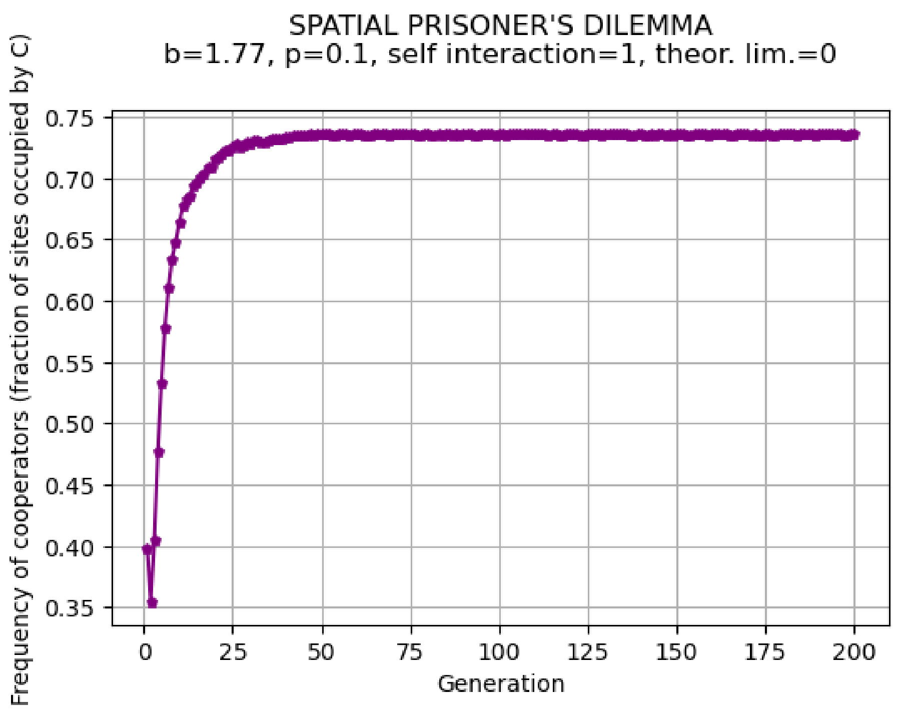

Figure 3.2: Cooperators frequency (f_C) plot. $1.75 < b < 1.8$, 10% D, 90% C randomly populated.

1.8 < b < 2, p=0.1

The region of $1.8 < b < 2$ is characterized by **spatial chaos**, as visualized in 3.3. The proportion of green and yellow points is considerably larger than the previous case, indicating an abundance of strategy shifts. Large C clusters (blue) are observed "invading" D regions (red) and conversely. **C and D coexist in the lattice, with their balance shifting in a chaotic manner** throughout the generations. This is better visualized here.

An interesting observation, is that although populations fluctuate, the **proportion of cooperators remains relatively stable** over generations, fluctuating around the value of 0.318, as shown in 3.4.

3.1.2 Invasion: Single Defector

An interesting case to investigate is what happens when a new strategy agent invades a grid of same-strategy-players. This is the nature equivalent of a new mutation appearing; *Under what conditions does natural selection favor the spread of the new mutant?* [2]. We focused on the case of Defector Invasion and explored the regions of b that produced the most intricate results 2.2. Two subcases are presented, one with self-interaction and one without.

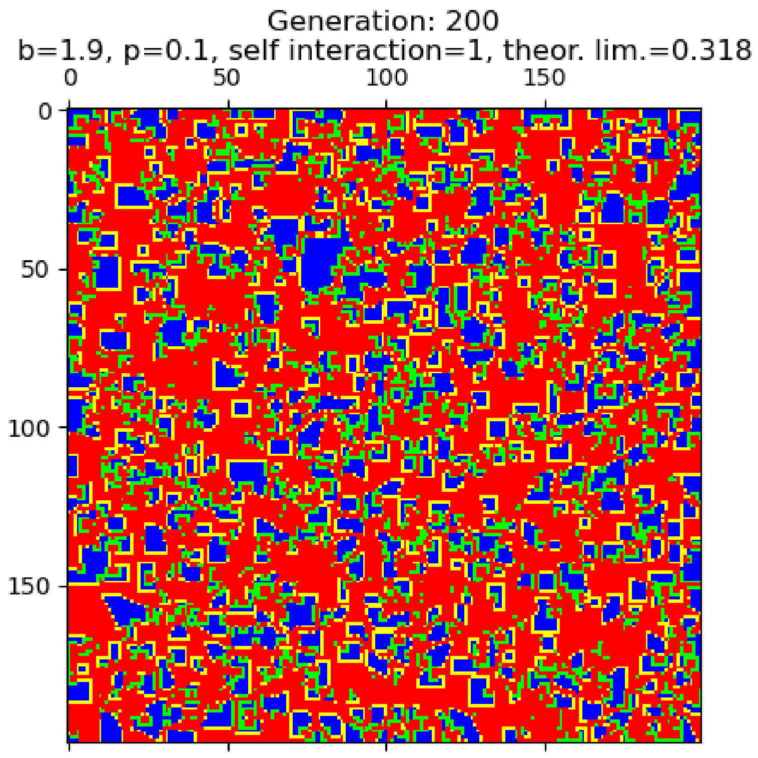

Figure 3.3: Lattice Evolution plot. $1.8 < b < 2$, 10% D, 90% C randomly populated. Click here for Evolution gif.

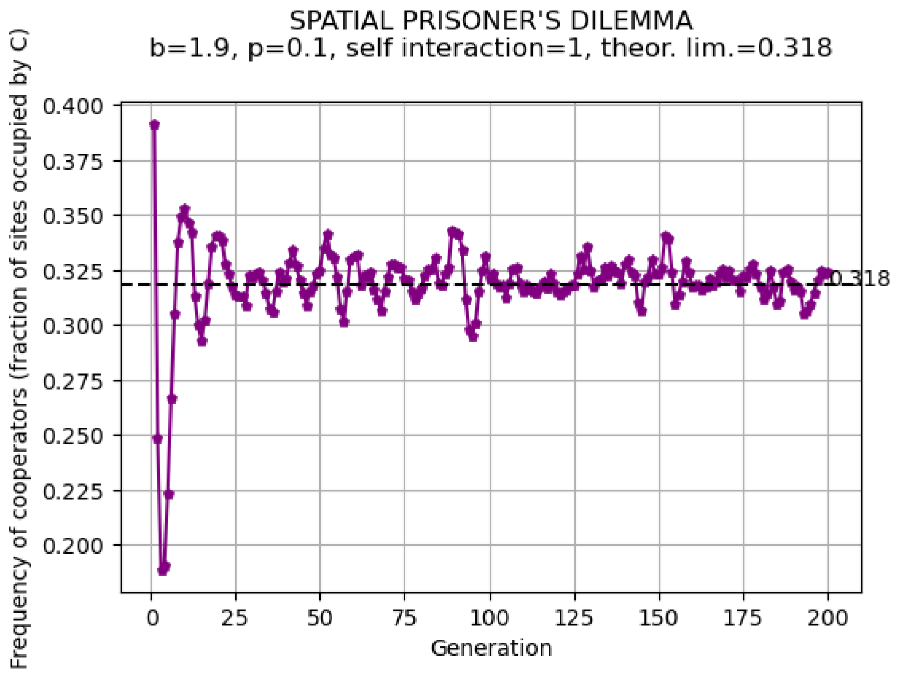

Figure 3.4: Cooperators frequency (f_C plot. $1.8 < b < 2$, 10% D, 90% C randomly populated

Self-Interaction: 1.8 < b < 2

For the self-interaction case of a single defector placed in the middle of cooperator-filled lattice, the region of most interest is $1.8 < b < 2$. In 3.5, 4 snapshots captured during the lattice evolution are presented. We witness an **"evolutionary kaleidoscope"**. A sequence of **dynamic fractals** is produced, ever-changing from generation to generation, but always maintaining **symmetry**.

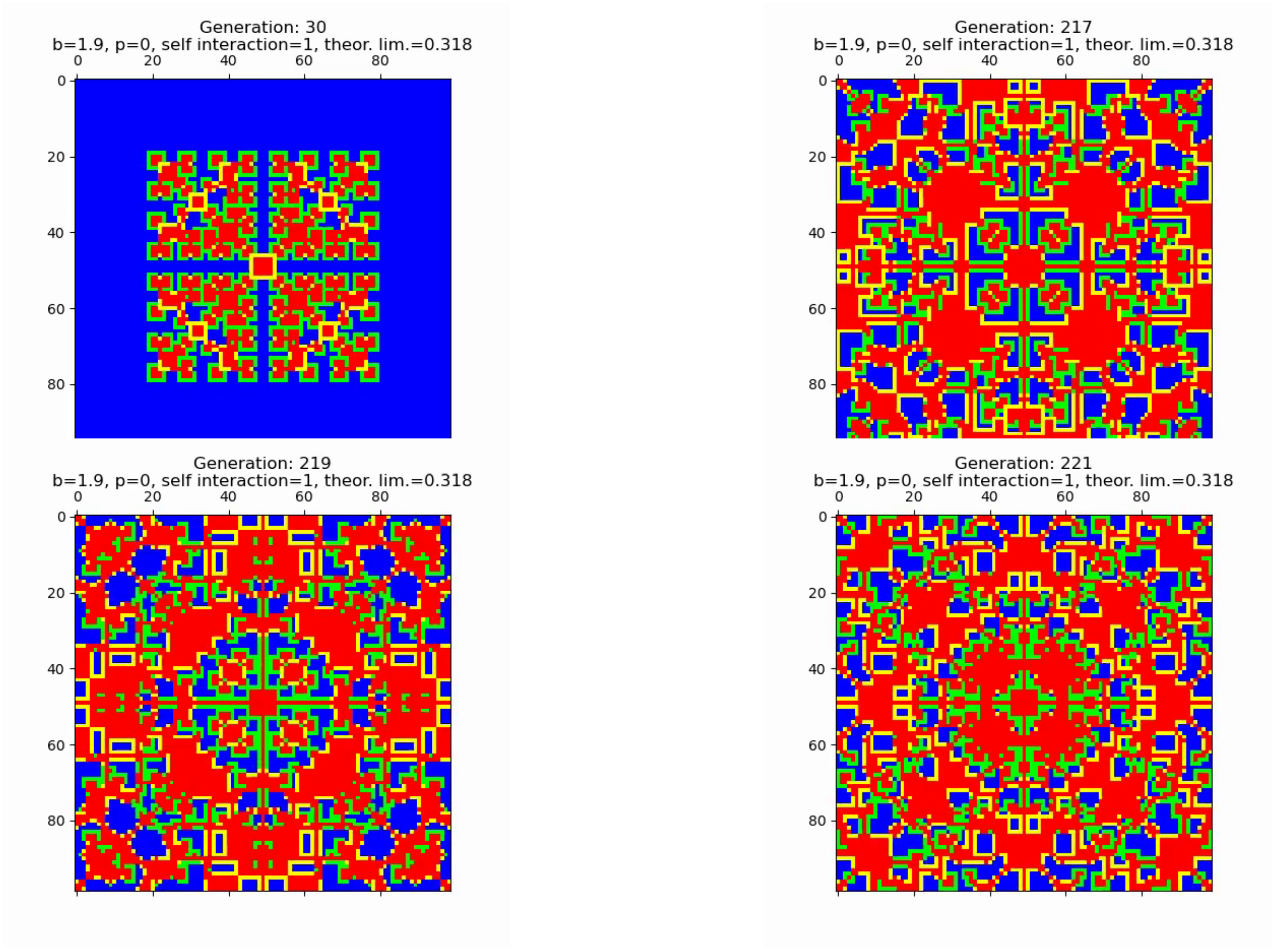

Figure 3.5: Lattice Evolution for single defector in the center of the grid with $1.8 < b < 2$ and self-interaction included. Snapshots for generations: 30 (top left), 217 (top right), 219 (bottom left) and 221 (bottom right)

This phenomenon can be appreciated to its fullest in this animation, which offers a visualization of the game for 1000 generations. The last frame is given in 3.6

In the frequency plot (3.7), **the proportion of C, though chaotically, oscillates around the theoretical limit of 0.318**, represented by the dashed line.

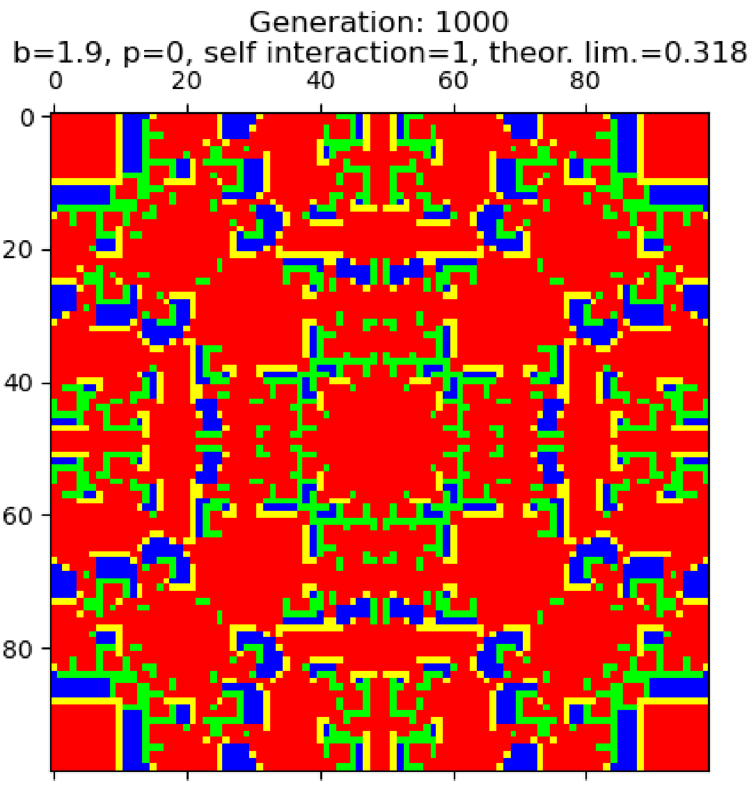

Figure 3.6: Lattice Evolution plot with $1.8 < b < 2$ and self-interaction included. Click here for Evolution gif.

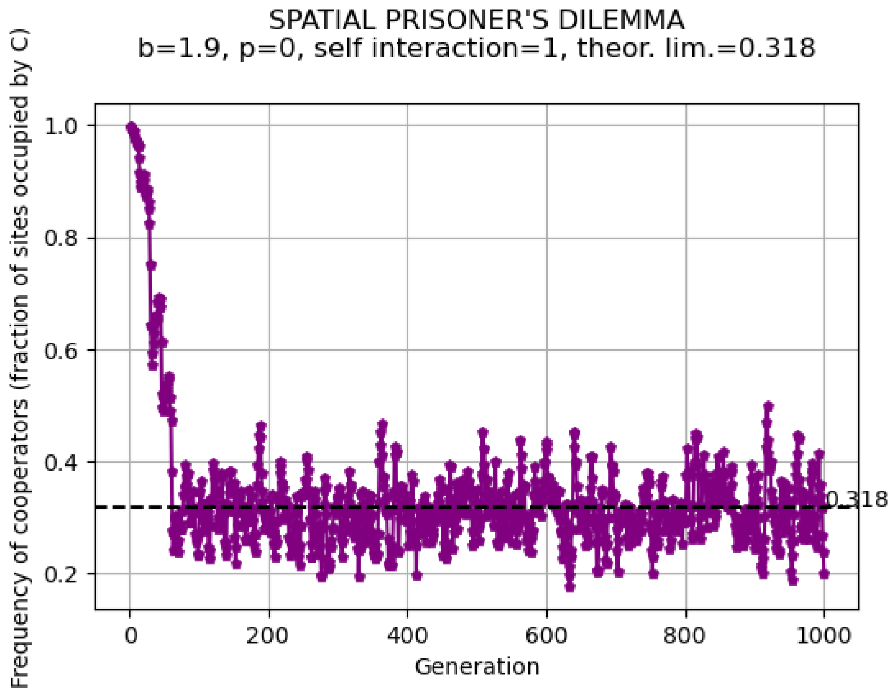

Figure 3.7: Cooperators frequency (f_C) plot for single defector in the center of the grid with $1.8 < b < 2$ and self-interaction included.

No Self-Interaction: 1.6< b < 1.667

As before, a single defector is placed in a grid of cooperators. For the no self-interaction case however, the interesting region of b values is $1.6 < b < 1.667$.

The dynamic behavior of the system throughout generations appears similar to the case where self-interaction was included. As seen in 3.8, where 4 different stages of the population evolution are showcased, similar **kaleidoscopic symmetric patterns** appear, rapidly changing with the passage of the rounds. The last generation frame is that of 3.9 and the lattice evolution in its entirety can be studied here.

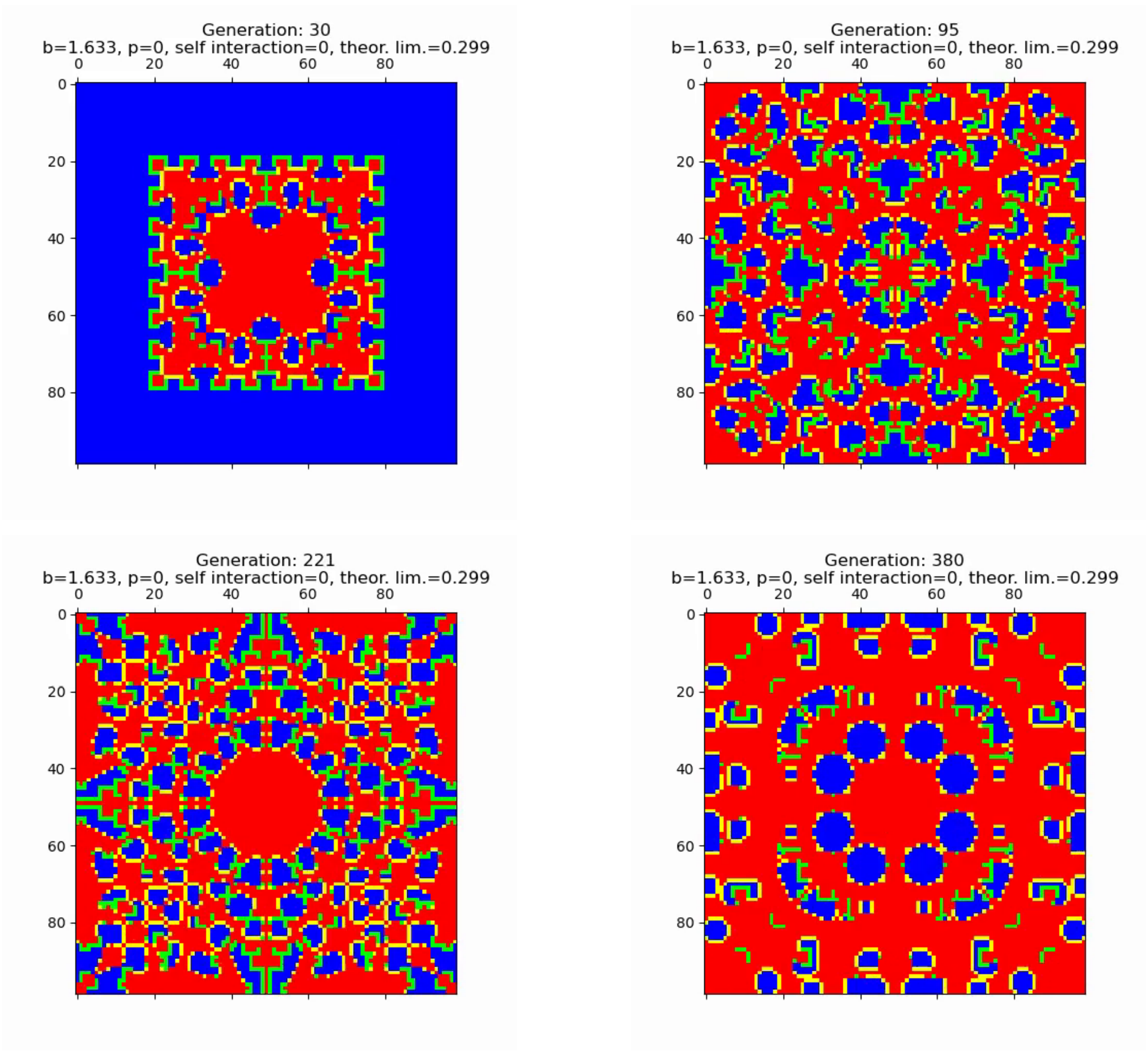

Figure 3.8: Lattice Evolution for single defector in the center of the grid with $1.6 < b < 1.667$ and no self-interaction. Snapshots for generations: 30 (top left), 95 (top right), 221 (bottom left) and 380 (bottom right).

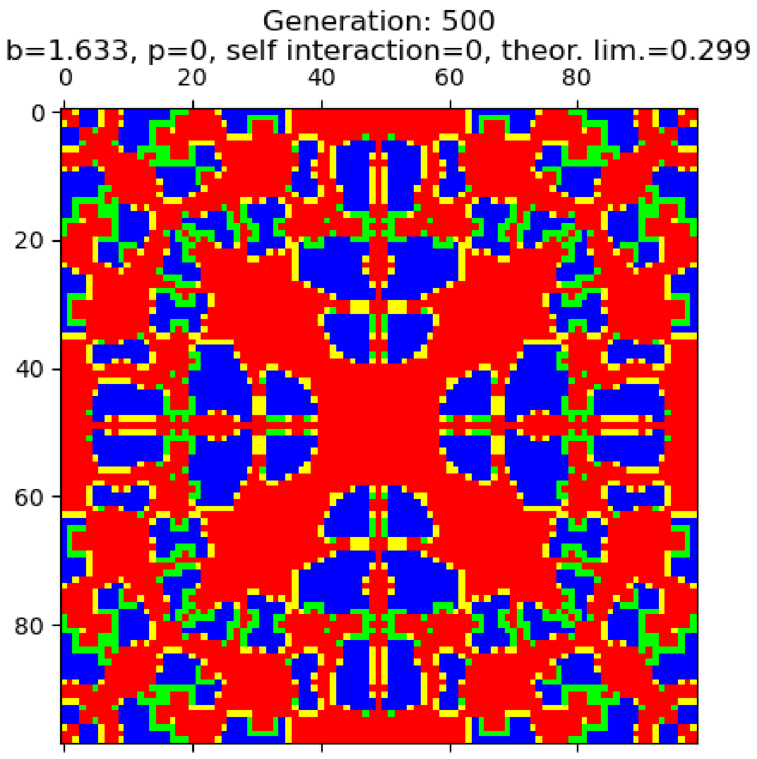

Figure 3.9: Lattice Evolution for single defector in the center of the grid with $1.6 < b < 1.667$ and no self-interaction. Click here for Evolution gif.

A notable difference from the self-interaction case is that the asymptotic limit now takes a value of **about 0.3** (3.10). The chaotic oscillating behavior persists.

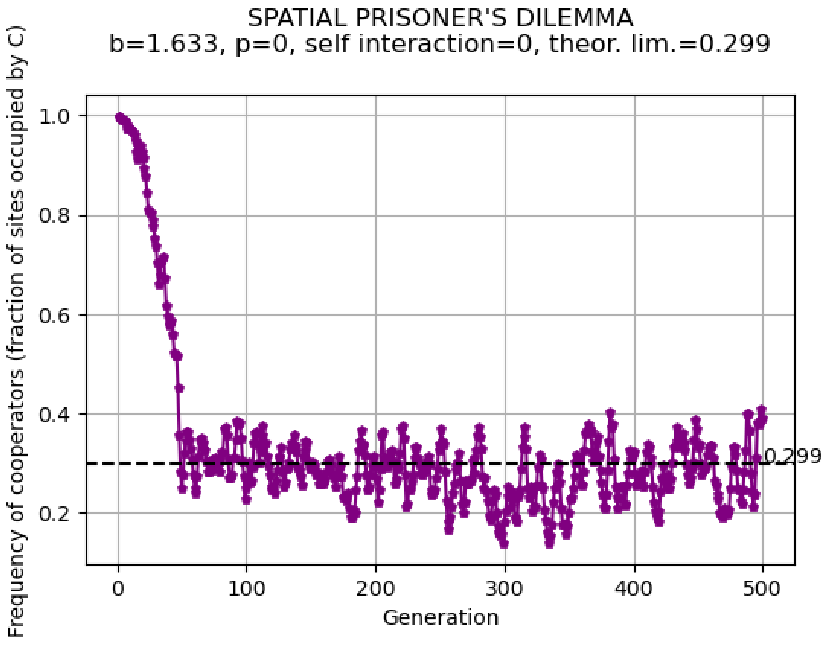

Figure 3.10: Cooperators frequency (f_C) plot for single defector in the center of the grid with $1.6 < b < 1.667$ and no self-interaction.

Chapter 4

Concluding Remarks

4.1 Summary of Findings

In this project, we investigated the dynamics of cooperation and defection in evolutionary games using spatial simulations on a square lattice. By varying the value of the temptation parameter b, we explored different dynamical regimes, focusing on random initial configurations of cooperators and defectors as well as defector invasion cases. Our findings matched perfectly those of the reference article [3]. Specifically:

1. **Random initial configuration scenario**:

   - For values of $1.75 < b < 1.8$, **cooperators consistently dominated**, forming stable clusters, while **defectors quickly diminished**, forming "D-lines".
   - For $1.8 < b < 2$, we observed **spatial chaos**, with clusters of cooperators and defectors continuously shifting in size and position. This resulted in dynamic coexistence, with both strategies present in the system over long periods. An interesting remark can be made regarding the **asymptotic limit of f_C**. It seems to match with the theoretical limit expected for cases of defector invasion ($f_C = 0.318$), as stated in 2.2. This agrees with Nowak's findings, but the reason behind it is not clear.

2. **Defector Invasion scenario**:

   - In the defector invasion scenarios (both for self-interaction included and excluded), we saw symmetric, **fractal-like patterns** emerge, indefinitely changing across generations, indicating continuous **strategy shifts**. The kaleidoscopic look matched the one described in Nowak's work.

- The f_c **asymptotic limit** resulting from the simulations agrees with the values expected from theory, both for the self-interaction (**0.318**) and the no self-interaction case (**0.299**).

These results reinforce the idea that the system's behavior is highly sensitive to the temptation parameter b and initial conditions, revealing the intricate relationship between individual strategies and collective dynamics.

4.2 Further Research and Applications

While our study provided significant insights, there are several directions for future research in the field evolutionary game dynamics:

- **Different Neighbor Interactions:** In our simulations, we used Moore Neighborhood interactions. Future work could study a 4-neighbor interaction regime or introduce more complex interaction rules, such as next-nearest neighbor interactions or longer-range connections, to see how these affect the system's dynamics.
- **Different Lattice Topologies:** Periodic boundary conditions could be added and different structures (like a hexagonal grid or random graphs) could be explored.
- **Strategy variations and updates:** It would be interesting to introduce different rules for updating strategies, or even taking a probabilistic approach instead of the deterministic employed in this study.
- **Machine Learning:** ML can be employed to predict long-term evolutionary outcomes. A reinforcement learning agent could adapt strategies over time to identify patterns in strategy evolution and discover optimal conditions for cooperation.

Though our model is deterministic and thus has limitations, it can be used to understand cooperative behavior in stable ecosystems, like symbiotic relationships in species, or evolutionary biology experiments. It could also find application in systems where interactions follow structured, predictable rules, like the spread of behaviors in a workplace.

Chapter 5

Appendix

5.1 Python Code

Spatial Prisoner's Dilemma Notebook

Bibliography

[1] R. Axelrod and R.M. Axelrod. *The Evolution of Cooperation*. Basic books. Basic Books, 1984. ISBN: 9780465021215. URL: https://books.google.gr/books?id=NJZBCGbNs98C.

[2] Martin A Nowak. *Evolutionary dynamics: exploring the equations of life*. Harvard university press, 2006.

[3] Martin A Nowak and Robert M May. "Evolutionary games and spatial chaos". In: *nature* 359.6398 (1992), pp. 826–829.

[4] B Myerson Roger et al. "Game theory: analysis of conflict". In: *The President and Fellows of Harvard College, USA* 66 (1991).