Lukas Fu Homework 4

Exercise 13.1

Across all variable alterations, ideal strategy n did not seem to change from 5, although changing parameters so that the punishment arrangement order is broken altered the results so that strategy $n \geq 7$ is also equally viable. The changes in parameters R and S only seemed to change the punishment quantity, but ideal strategy remained unchanged.

Subproblem (a) and (b)

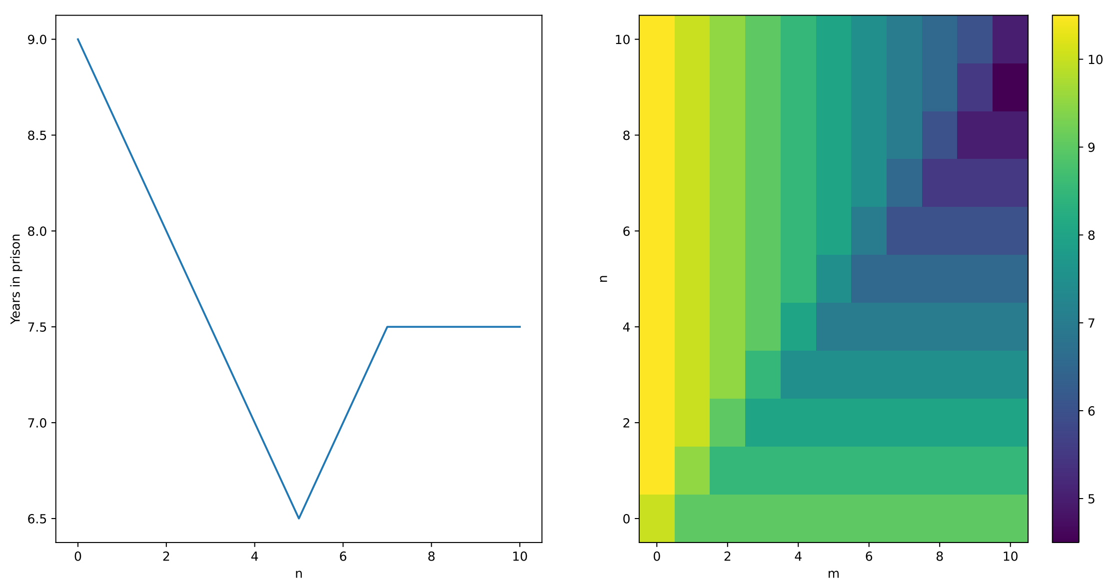


Figure 1: For parameter values $N = 10$, $T = 0$, $R = 0.5$, $P = 1$ and $S = 1.5$. Best strategy at $n = 5$, and the upswing happens immediately after when $n = 6 = m$.

Subproblem (c)

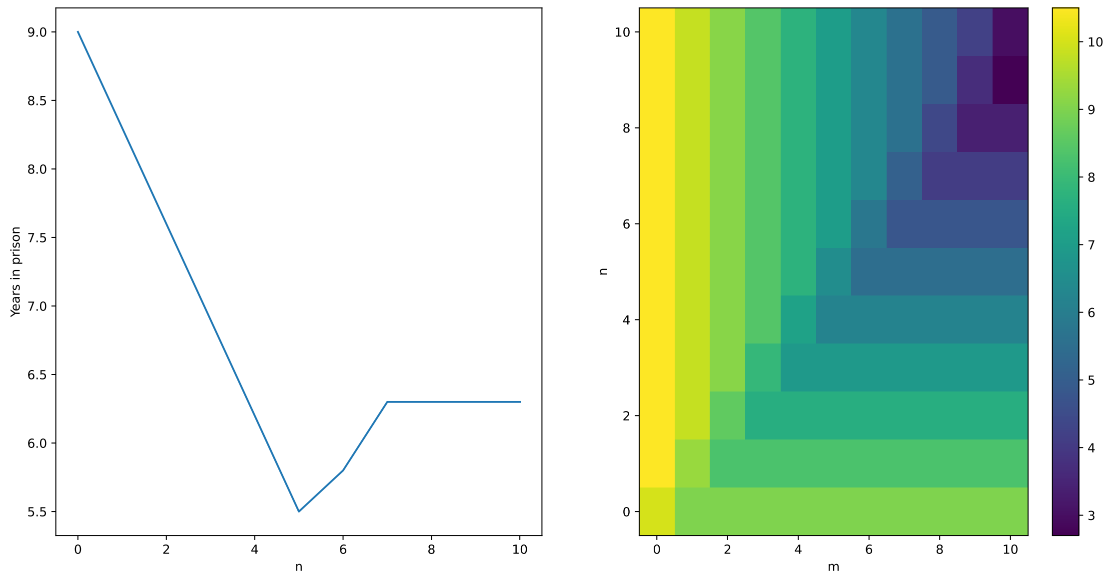


Figure 2: Changing only parameter R to $R = 0.3$.

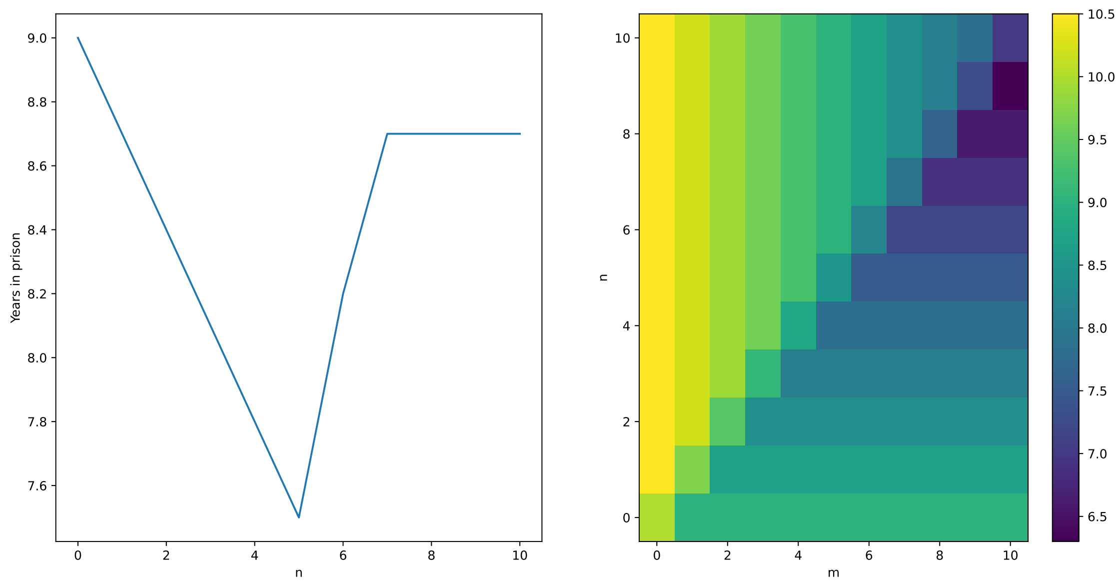


Figure 3: Changing only parameter R to $R = 0.7$.

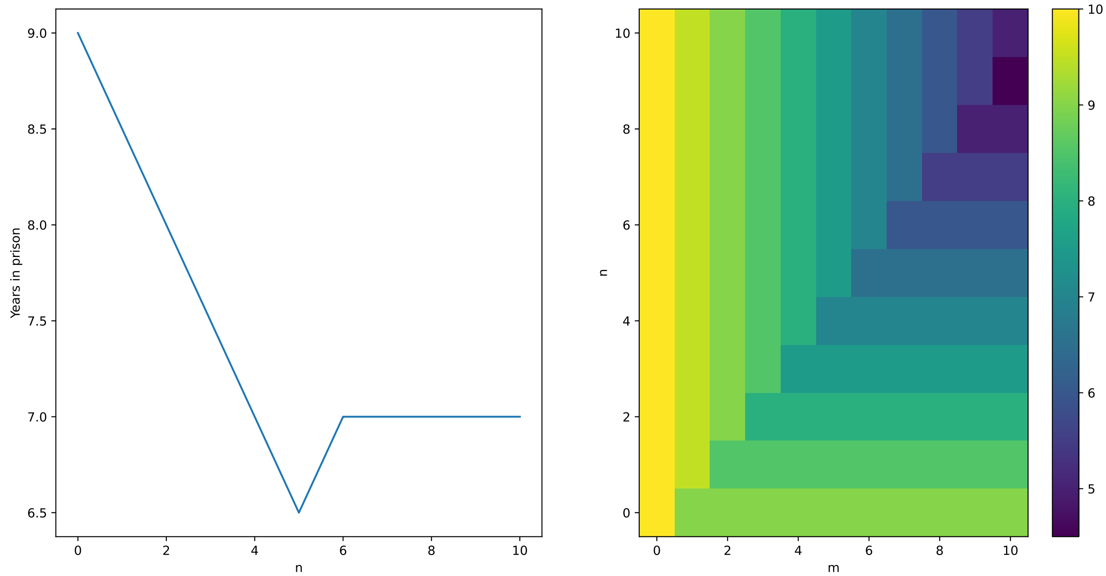


Figure 4: Changing only parameter S to $S = 1.0$.

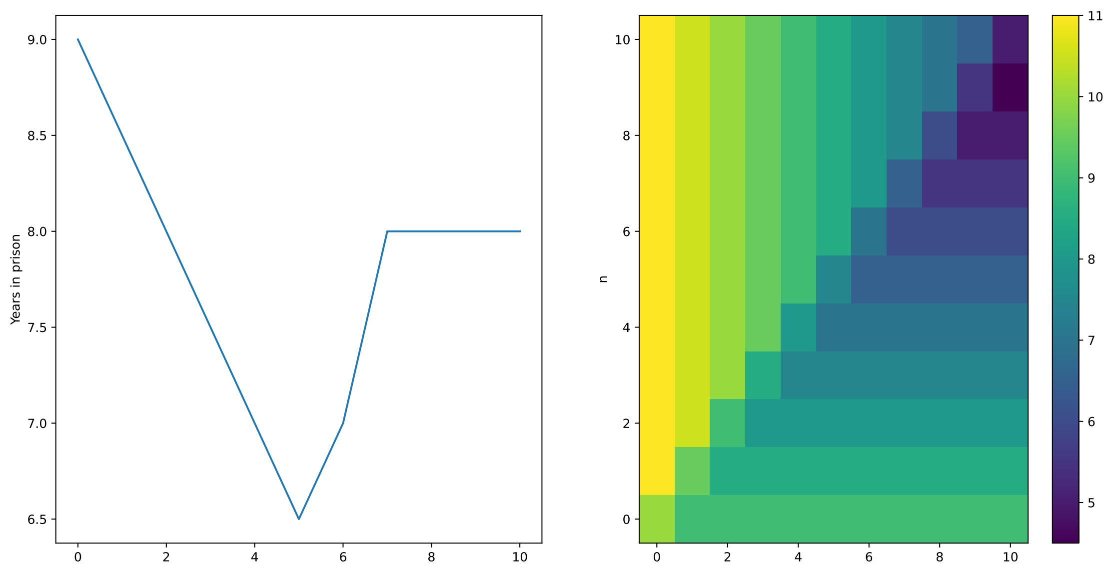


Figure 5: Changing only parameter S to $S = 2.0$.

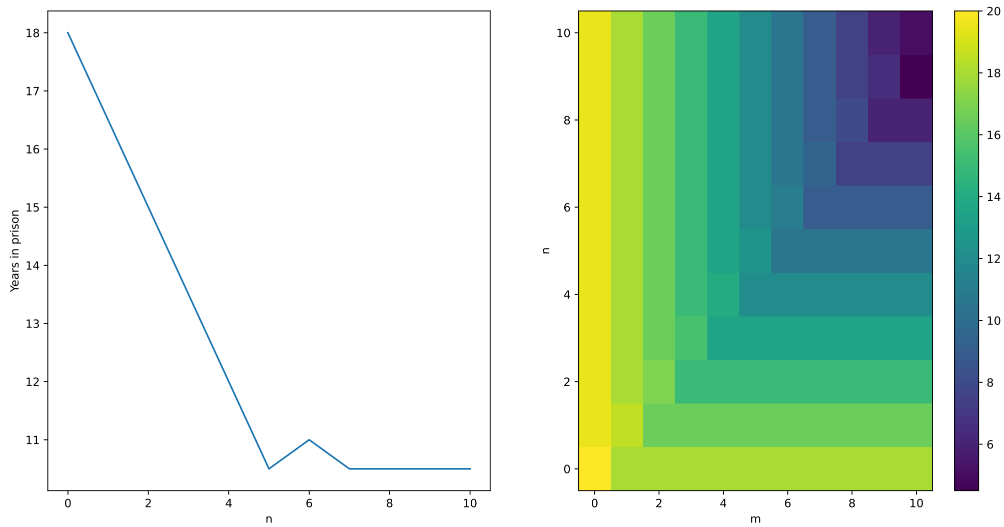
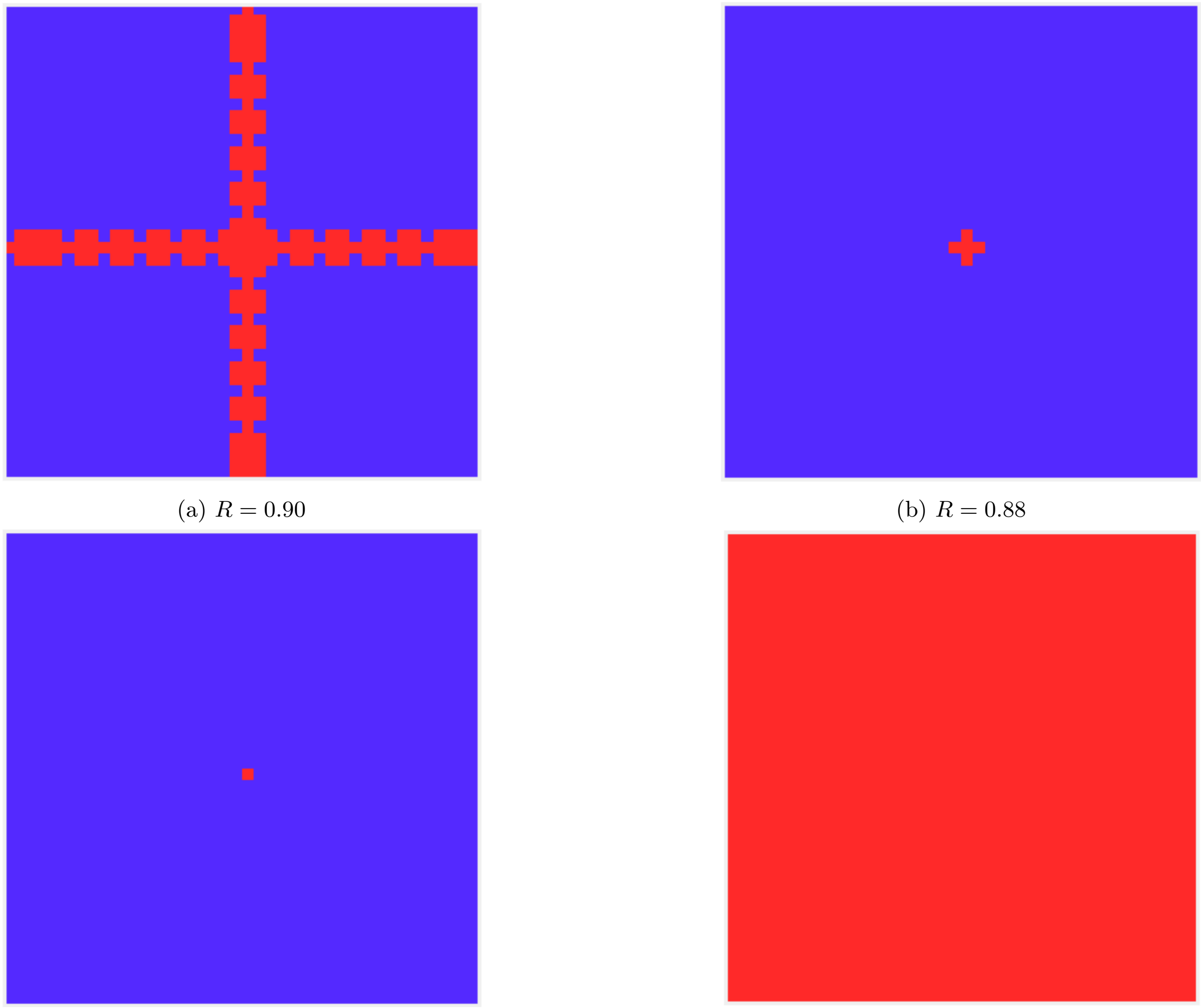


Figure 6: Changing only parameter T to $T = 2.0$.

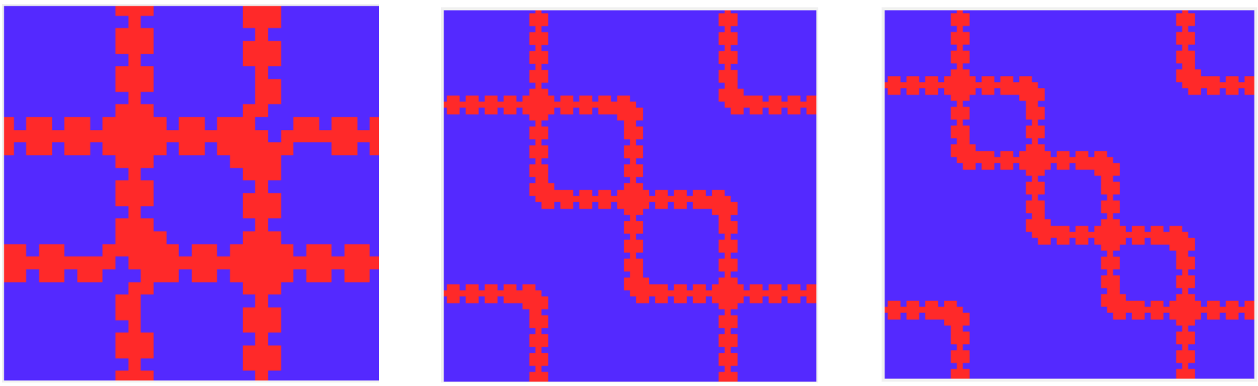
Exercise 13.2



(a) $R = 0.90$ (b) $R = 0.88$ (c) $R = 0.83$ (d) $R = 0.93$

Figure 7: Various stable patterns are shown when $0.83 \leq R \leq 0.92$, overrun by defectors for $R \geq 0.93$ and by cooperators for $R \leq 0.82$.

Subproblem (b)



(a) Two initial defectors in a lattice of cooperators.

(b) Three initial defectors in a lattice of cooperators.

(c) Four initial defectors in a lattice of cooperators.

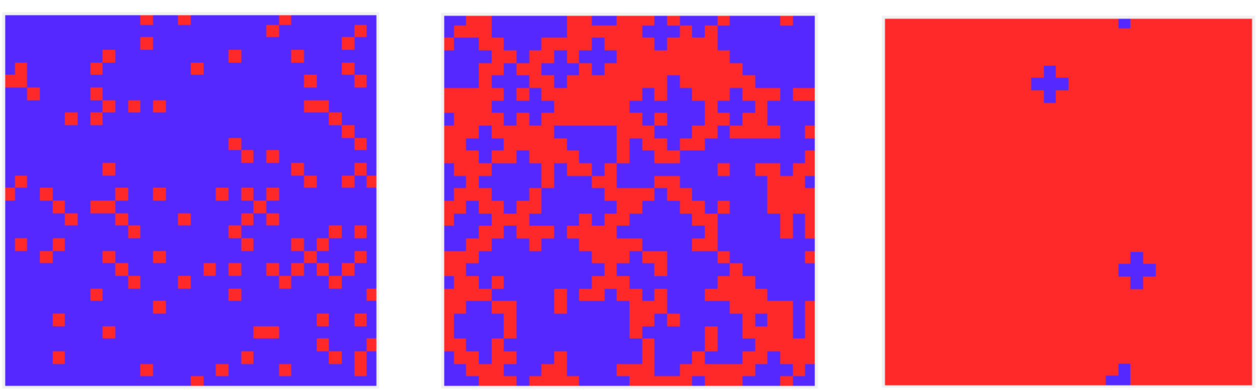
Figure 8: Pattern formations caused by placing defectors evenly at a diagonal througout the lattice. Lattice size was increased.

Subproblem (c) and (d)

For a single cooperator in a lattice of defectors, the single cooperator immediately turns to defector and nothing happens. For a 3 by 3 cluster of cooperators in a lattice of defectors, it is stable for $R \in [0.86, 0.93]$, overrun by defectors for $R \leq 0.85$ or taken over by cooperators for $R \geq 0.94$.

Exercise 13.3

Subproblem (a), (b) and (c)



(a) The pattern shown for $0.79 \leq R \leq 0.83$.

(b) The pattern shown for $0.84 \leq R \leq 0.85$.

(c) The pattern shown when cooperation is fading away for $R \geq 0.86$.

Figure 9: Patterns of cooperators and defectors when mutation rates are added. The patterns for each regime is the same when varying S instead of R.

Subproblem (d) and (e)

$0.79 \leq R \leq 0.83$ displays a semi-dominant pattern in favour of cooperation, while $R \leq 0.78$ is heavily dominant to cooperation. While $0.84 \leq R \leq 0.85$, the lattice is evenly split between cooperation and defection. When $R \geq 0.86$, the lattice will gradually turn to defectors, and the rate of conversion increases while approaching $R = 1.0$.

When varying S instead and fixing $R = 0.85$, the semi-dominant pattern for cooperation is shown while $1.00 \leq S \leq 1.15$, the even split pattern while $1.16 \leq S \leq 1.55$ and dominant for defection for $R \geq 1.56$. The patterns for the regimes when varying S are the same as when varying R.

Code

13.1

```python
import numpy as np
import matplotlib.pyplot as plt

"""
N = 10      # Rounds
T = 0       # Betrayer
R = 0.5     # Both cooperate
P = 1       # Both defect
S = 1.5     # Betrayed
"""

class Main:

    def __init__(self, defect):
        self.defect = defect
        self.init_defect = defect
        self.years = 0.
        self.choices = []

    def reset(self):
        self.years = 0
        self.choices = []
        self.defect = self.init_defect


def play(rules, player1, player2):
    player1.reset()
    player2.reset()

    for i in range(rules.get("Rounds")):
        if i < player1.defect:
            player1.choices.append("coop")
        else:
            player1.choices.append("defect")
        if i < player2.defect:
            player2.choices.append("coop")
        else:
            player2.choices.append("defect")

        if player1.choices[i] == "defect":
            player2.defect = 0
        elif player2.choices[i] == "defect":
            player1.defect = 0

    count_years(player1, player2, rules)


def count_years(p1, p2, rules):  # add years based on choices and rules
    for i in range(rules.get("Rounds")):
        if p1.choices[i] == p2.choices[i] and p1.choices[i] == "coop":
```

```
            p1.years += rules.get("R")
            p2.years += rules.get("R")
        if p1.choices[i] == p2.choices[i] and p1.choices[i] == "defect":
            p1.years += rules.get("P")
            p2.years += rules.get("P")
        if p1.choices[i] == "coop" and p2.choices[i] == "defect":
            p1.years += rules.get("S")
            p2.years += rules.get("T")
        if p1.choices[i] == "defect" and p2.choices[i] == "coop":
            p1.years += rules.get("T")
            p2.years += rules.get("S")


if __name__ == "__main__":
    # initialize arrays and constants
    dilemma_rules = {"Rounds": 10, "T": 0, "R": 0.5, "P": 1, "S": 1.5}
    final_years_p1 = []
    years_p1 = []
    years_p2 = []
    values = 11
    for i in range(values):
        prisoner1 = Main(defect=i)  # initialize a prisoner with defect of 0-10
        prisoner2 = Main(defect=6)  # initialize a prisoner with defect of 6
        play(dilemma_rules, prisoner1, prisoner2)  # play the game with the two prisoners
        final_years_p1.append(prisoner1.years)  # add the results from the game to the array

    for j in range(values):
        player1 = Main(defect=j)
        for k in range(values):
            player2 = Main(defect=k)
            play(dilemma_rules, player1, player2)
            years_p1.append(player1.years)
            years_p2.append(player2.years)

    #print("p1 years:", len(years_p1))
    #print("p2 years:", len(years_p2))
    #print(np.array(years_p1).reshape(-1, 11))
    #print(np.array(years_p2).reshape(-1, 11))

    fig, axes = plt.subplots(nrows=1, ncols=2)
    x = np.linspace(0, 10, values, endpoint=True)  # m
    y = np.linspace(0, 10, values, endpoint=True)  # n
    axes[0].plot(x, final_years_p1)
    axes[0].set_xlabel('n')
    axes[0].set_ylabel('Years in prison')
    c = axes[1].pcolor(x, y, np.array(years_p1).reshape(-1, 11))
    axes[1].set_xlabel('m')
    axes[1].set_ylabel('n')
    fig.colorbar(c, ax=axes[1])
    plt.show()
```

13.2

```
import numpy as np
from numpy import arctan2 as atan2, sin, cos
import matplotlib.pyplot as plt
from IPython import display
from scipy.constants import Boltzmann as kB
from tkinter import *
```

```python
from tkinter import ttk
from PIL import ImageGrab
from PIL import Image
from PIL import ImageTk as itk
import time

res = 500  # Resolution of the animation
tk = Tk()
tk.geometry(str(int(res * 1.1)) + 'x' + str(int(res * 1.3)))
tk.configure(background='white')

canvas = Canvas(tk, bd=2)  # Generate animation window
tk.attributes('-topmost', 0)
canvas.place(x=res / 20, y=res / 20, height=res, width=res)

ccolor = ['#5429FF', '#29A6FF', '#29F7FF', '#29FFB0', '#7BFF29', '#D9FF29', '#FFD429', '#FF2929
ccolor = ccolor[::-1]

N = 7  # Number of rounds
L = 30  # Lattice size
strategies = np.ceil(np.random.rand(L, L) * (N + 1)) - 1

dynamic_mu = Scale(tk, from_=0, to=0.1, resolution=0.001, orient=HORIZONTAL, label='Mutation pr
dynamic_mu.place(relx=.5, rely=.84, relheight=0.12, relwidth=0.2)
dynamic_mu.set(0.01)

dynamic_S = Scale(tk, from_=1, to=2, resolution=0.01, orient=HORIZONTAL, label='S')
dynamic_S.place(relx=.27, rely=.84, relheight=0.12, relwidth=0.2)
dynamic_S.set(1.5)

dynamic_R = Scale(tk, from_=0, to=1, resolution=0.01, orient=HORIZONTAL, label='R')
dynamic_R.place(relx=.05, rely=.84, relheight=0.12, relwidth=0.2)
dynamic_R.set(0.72)


def pd(R, S, N, n1, n2):
    # Prisoner's dilemma with two agents with strategies n1 and n2
    r = min(n1, n2)
    if n1 < n2:
        p1 = r * R + (N - 1 - r)
        p2 = r * R + S + (N - 1 - r)
    elif n1 == n2:
        p1 = r * R + (N - r)
        p2 = p1
    else:
        p1 = r * R + S + (N - 1 - r)
        p2 = r * R + (N - 1 - r)
    return p1, p2


def mditer(strategies, R, S, N, L, mu):
    # Iterate through the lattice and update the strategies
    P = np.zeros((L, L))
    pind = np.roll(np.arange(L), -1)  # Index of the next element in the lattice
    mind = np.roll(np.arange(L), 1)  # Index of the previous element in the lattice

    # play the game with the next neighbours and register points
    for i in range(L):
        for j in range(L):
```

```python
            p1, p2 = pd(R, S, N, strategies[i, j], strategies[pind[i], j])
            P[i, j] += p1
            P[pind[i], j] += p2

            p1, p2 = pd(R, S, N, strategies[i, j], strategies[i, pind[j]])
            P[i, j] += p1
            P[i, pind[j]] += p2

    newstrategies = np.zeros((L, L))

    for i in range(L):
        for j in range(L):
            if np.random.rand() < mu:  # mutate the strategy with probability mu
                newstrategies[i, j] = np.random.randint(N + 1)
            else:
                # copy the strategy of the neighbor with the lowest P, if P is equal, choose r
                pp = [P[i, j], P[mind[i], j], P[i, mind[j]], P[pind[i], j], P[i, pind[j]]]
                ss = [strategies[i, j], strategies[mind[i], j], strategies[i, mind[j]], strateg
                      strategies[i, pind[j]]]
                newstrategies[i, j] = np.random.choice([ss[i] for i in range(5) if pp[i] == mi

    return newstrategies


def parameters_binary():
    global dynamic_mu, dynamic_S, dynamic_R, strategies, N, L, mu
    dynamic_mu.set(0)
    dynamic_S.set(1.5)
    dynamic_R.set(0.86)
    strategies = np.ones((L, L)) * 0  # always cooperate -> n=N
    #strategies[int(L / 5), int(L / 5)] = 0  # always defect -> n=0
    #strategies[int(L * 2 / 5), int(L * 2 / 5)] = 0
    #strategies[int(L * 3 / 5), int(L * 3 / 5)] = 0
    #strategies[int(L * 4 / 5), int(L * 4 / 5)] = 0
    # for all defect and placing cooperator
    #strategies = np.zeros((L, L))  # everyone always defects
    strategies[int(L / 2), int(L / 2)] = N  # cooperating point
    strategies[int(L / 2-1), int(L / 2-1)] = N
    strategies[int(L / 2-1), int(L / 2)] = N
    strategies[int(L / 2), int(L / 2-1)] = N
    strategies[int(L / 2+1), int(L / 2+1)] = N
    strategies[int(L / 2+1), int(L / 2)] = N
    strategies[int(L / 2), int(L / 2+1)] = N
    strategies[int(L / 2+2), int(L / 2)] = N
    strategies[int(L / 2), int(L / 2+2)] = N

rest = Button(tk, text='Binary', command=parameters_binary)
rest.place(relx=0.75, rely=.85, relheight=0.12, relwidth=0.15)
evolutionary_games = np.uint8(np.zeros((L, L, 3)))

while True:
    strategies = mditer(strategies, dynamic_R.get(), dynamic_S.get(), N, L, dynamic_mu.get())

    for i in range(L):
        for j in range(L):
            rgb = []
            for t in (1, 3, 5):
                rgb.append(int(ccolor[int(strategies[i, j])][t:t + 2], 16))

```

```
            evolutionary_games[i, j, :] = rgb

    img = itk.PhotoImage(Image.fromarray(np.uint8(evolutionary_games), 'RGB').resize((res, res
    canvas.create_image(0, 0, anchor=NW, image=img)
    tk.title('time' + str(t))
    time.sleep(0.01)
    tk.update()
```

13.3

```
import numpy as np
from numpy import arctan2 as atan2, sin, cos
import matplotlib.pyplot as plt
from IPython import display
from scipy.constants import Boltzmann as kB
from tkinter import *
from tkinter import ttk
from PIL import ImageGrab
from PIL import Image
from PIL import ImageTk as itk
import time

res = 500  # Resolution of the animation
tk = Tk()
tk.geometry(str(int(res * 1.1)) + 'x' + str(int(res * 1.3)))
tk.configure(background='white')

canvas = Canvas(tk, bd=2)  # Generate animation window
tk.attributes('-topmost', 0)
canvas.place(x=res / 20, y=res / 20, height=res, width=res)

ccolor = ['#5429FF', '#29A6FF', '#29F7FF', '#29FFB0', '#7BFF29', '#D9FF29', '#FFD429', '#FF292
ccolor = ccolor[::-1]

N = 7  # Number of rounds
L = 30  # Lattice size
strategies = np.ceil(np.random.rand(L, L) * (N + 1)) - 1

dynamic_mu = Scale(tk, from_=0, to=0.1, resolution=0.001, orient=HORIZONTAL, label='Mutation p
dynamic_mu.place(relx=.5, rely=.84, relheight=0.12, relwidth=0.2)
dynamic_mu.set(0.01)

dynamic_S = Scale(tk, from_=1, to=2, resolution=0.01, orient=HORIZONTAL, label='S')
dynamic_S.place(relx=.27, rely=.84, relheight=0.12, relwidth=0.2)
dynamic_S.set(1.5)

dynamic_R = Scale(tk, from_=0, to=1, resolution=0.01, orient=HORIZONTAL, label='R')
dynamic_R.place(relx=.05, rely=.84, relheight=0.12, relwidth=0.2)
dynamic_R.set(0.72)


def pd(R, S, N, n1, n2):
    # Prisoner's dilemma with two agents with strategies n1 and n2
    r = min(n1, n2)
    if n1 < n2:
        p1 = r * R + (N - 1 - r)
        p2 = r * R + S + (N - 1 - r)
    elif n1 == n2:
        p1 = r * R + (N - r)
```

```
        p2 = p1
    else:
        p1 = r * R + S + (N - 1 - r)
        p2 = r * R + (N - 1 - r)
    return p1, p2


def mditer(strategies, R, S, N, L, mu):
    # Iterate through the lattice and update the strategies
    P = np.zeros((L, L))
    pind = np.roll(np.arange(L), -1)  # Index of the next element in the lattice
    mind = np.roll(np.arange(L), 1)  # Index of the previous element in the lattice

    # play the game with the next neighbours and register points
    for i in range(L):
        for j in range(L):
            p1, p2 = pd(R, S, N, strategies[i, j], strategies[pind[i], j])
            P[i, j] += p1
            P[pind[i], j] += p2

            p1, p2 = pd(R, S, N, strategies[i, j], strategies[i, pind[j]])
            P[i, j] += p1
            P[i, pind[j]] += p2

    newstrategies = np.zeros((L, L))

    for i in range(L):
        for j in range(L):
            if np.random.rand() < mu:  # mutate the strategy with probability mu
                if newstrategies[i, j] == 0:  # change for the binary case and mutation
                    newstrategies[i,j] = N
                newstrategies[i,j] = 0
            else:
                # copy the strategy of the neighbor with the lowest P, if P is equal, choose r
                pp = [P[i, j], P[mind[i], j], P[i, mind[j]], P[pind[i], j], P[i, pind[j]]]
                ss = [strategies[i, j], strategies[mind[i], j], strategies[i, mind[j]], strateg
                      strategies[i, pind[j]]]
                newstrategies[i, j] = np.random.choice([ss[i] for i in range(5) if pp[i] == mi

    return newstrategies


def parameters_binary():
    global dynamic_mu, dynamic_S, dynamic_R, strategies, N, L, mu
    dynamic_mu.set(0.01)
    dynamic_S.set(1.55)
    dynamic_R.set(0.85)
    strategies = np.ones((L, L), dtype=int) * N  # always cooperate -> n=N
    strategies[int(L / 2), int(L / 2)] = 0  # always defect -> n=0


rest = Button(tk, text='Binary', command=parameters_binary)
rest.place(relx=0.75, rely=.85, relheight=0.12, relwidth=0.15)
evolutionary_games = np.uint8(np.zeros((L, L, 3)))

while True:
    strategies = mditer(strategies, dynamic_R.get(), dynamic_S.get(), N, L, dynamic_mu.get())

    for i in range(L):
```

```
        for j in range(L):
            rgb = []
            for t in (1, 3, 5):
                rgb.append(int(ccolor[int(strategies[i, j])][t:t + 2], 16))

            evolutionary_games[i, j, :] = rgb

    img = itk.PhotoImage(Image.fromarray(np.uint8(evolutionary_games), 'RGB').resize((res, res
    canvas.create_image(0, 0, anchor=NW, image=img)
    tk.title('time' + str(t))
    time.sleep(0.01)
    tk.update()
```