Homework 4, Game Theory

Erik Norlin, 19970807-9299

December 3, 2022

13.1: a, b, c

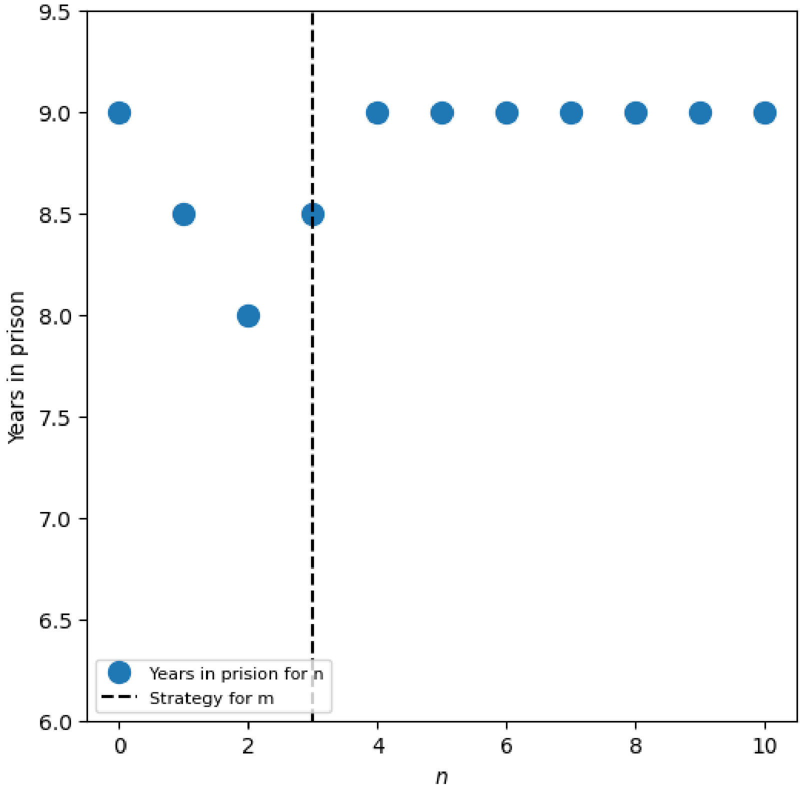

Figure 1: T = 0, R = 0.5, P = 1, S = 1.5.

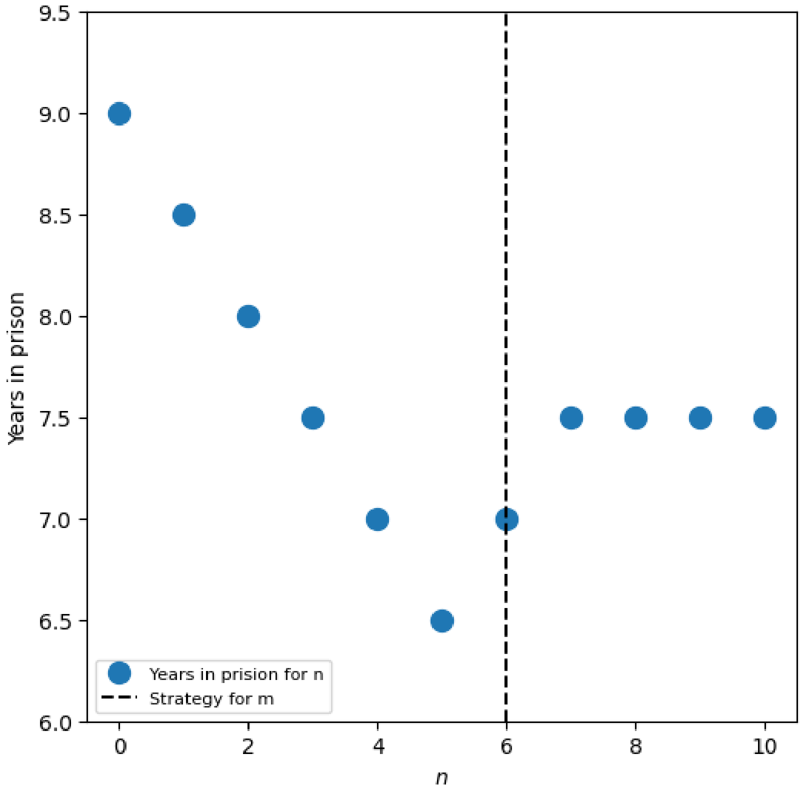

Figure 2: T = 0, R = 0.5, P = 1, S = 1.5.

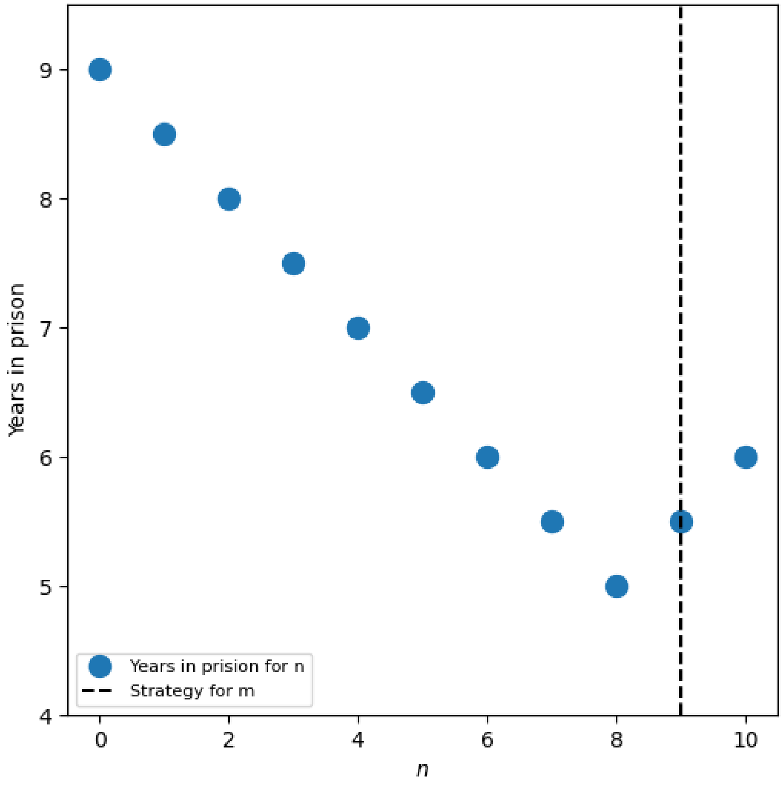

Figure 3: T = 0, R = 0.5, P = 1, S = 1.5.

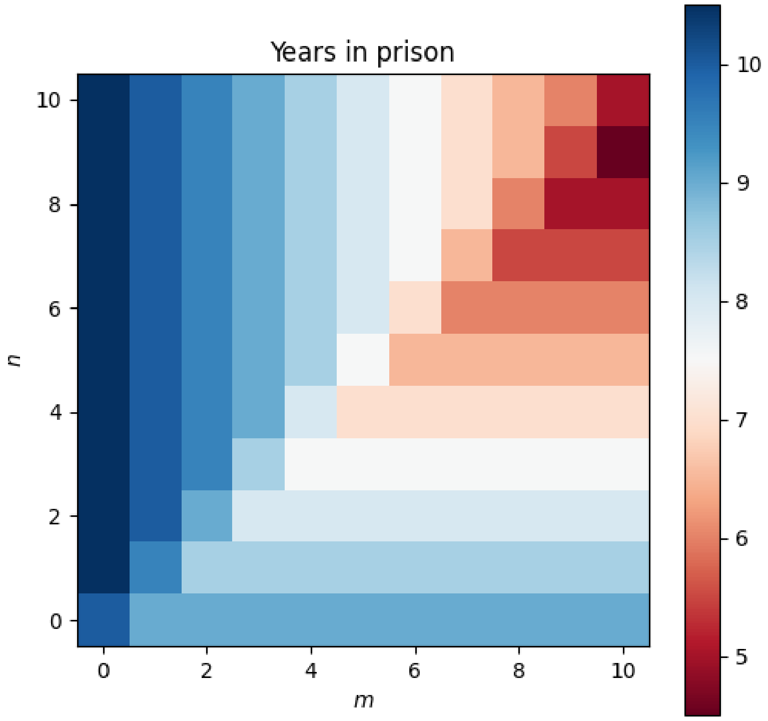

Figure 4: T = 0, R = 0.5, P = 1, S = 1.5.

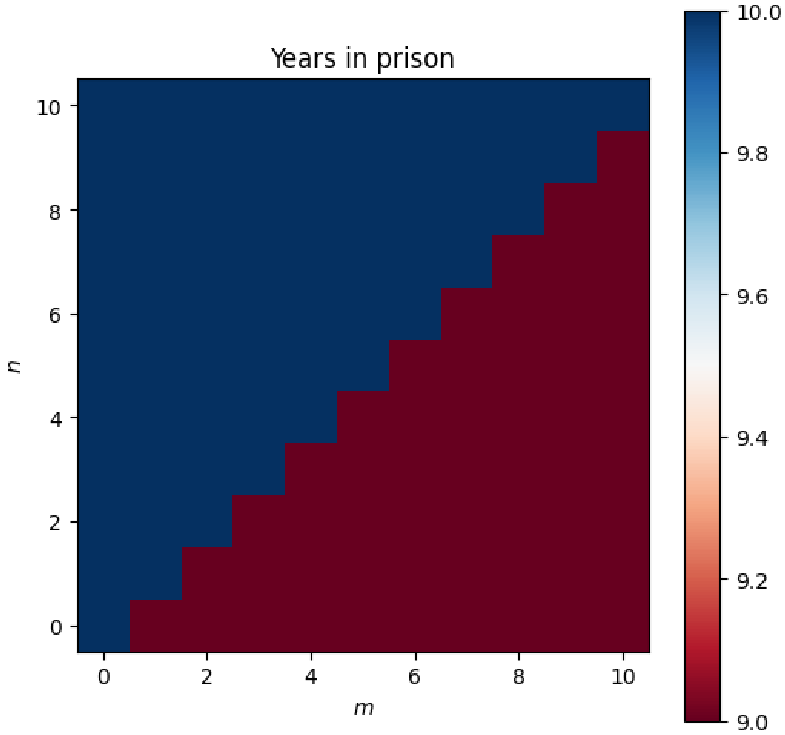

Figure 5: T = 0, R = 1, P = 1, S = 1.

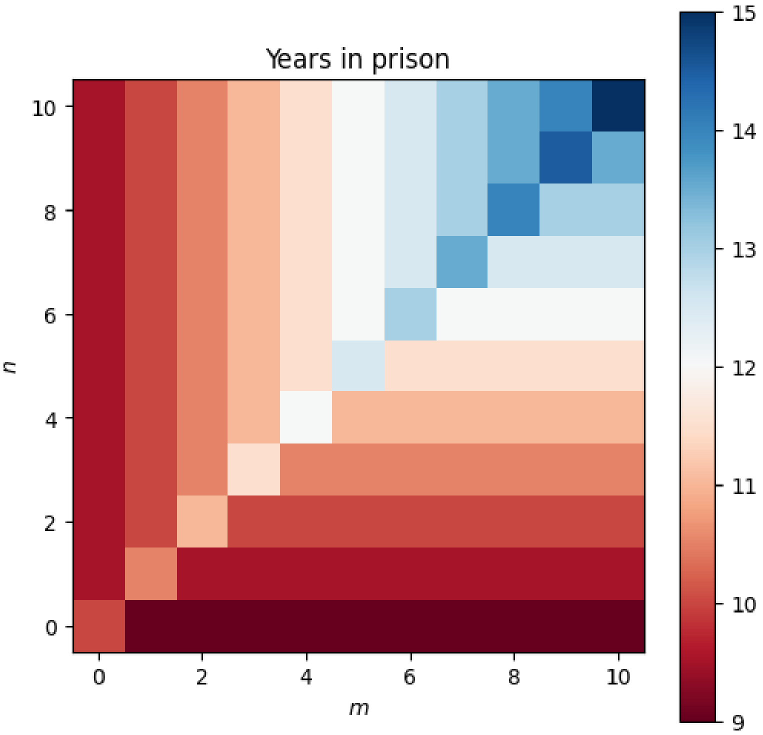

Figure 6: T = 0, R = 1.5, P = 1, S = 0.5

13.2: a, b, c

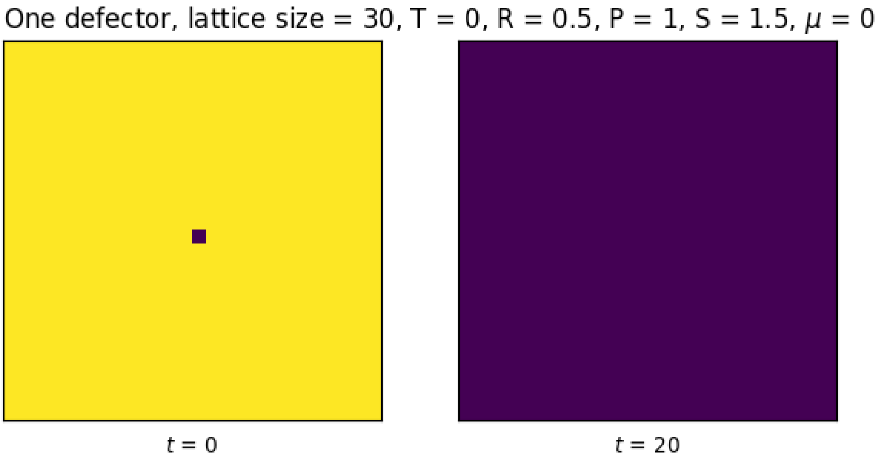

Figure 7: Left: yellow = 7, purple = 0. Right: purple = 7 (colors switched).

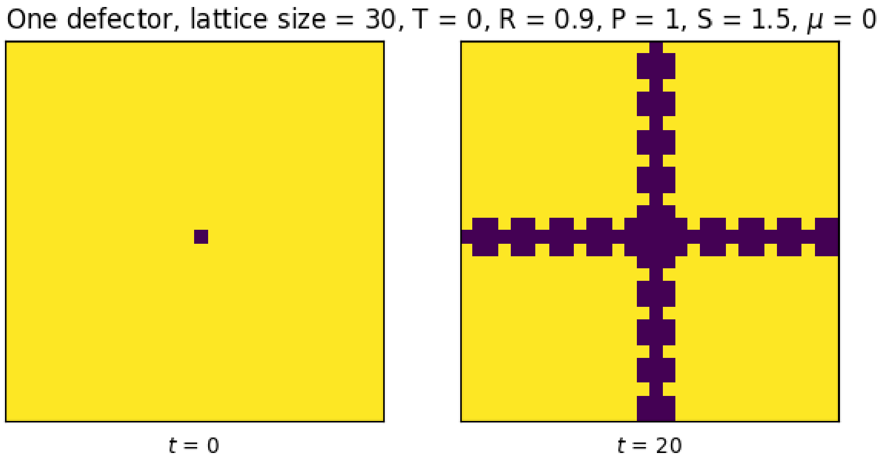

Figure 8: Yellow = 7, purple = 0.

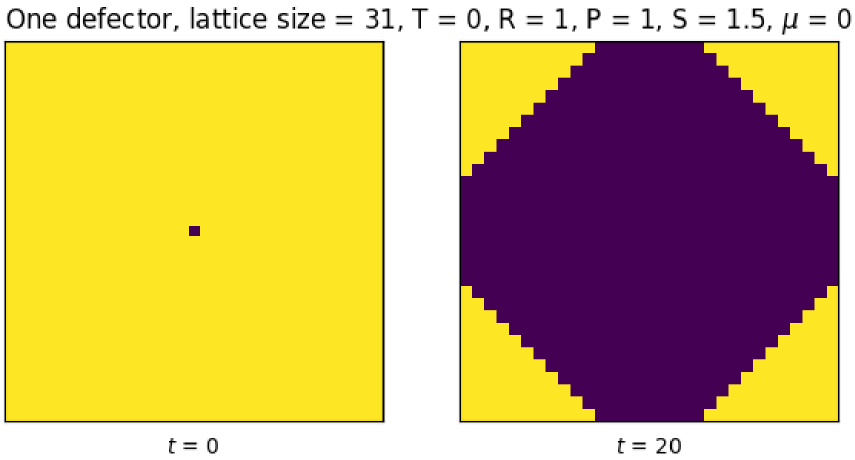

Figure 9: Yellow = 7, purple = 0.

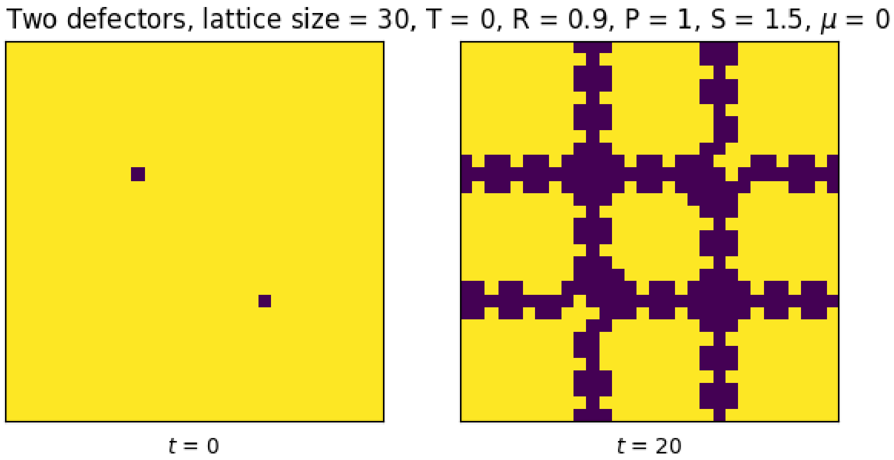

Figure 10: Yellow = 7, purple = 0.

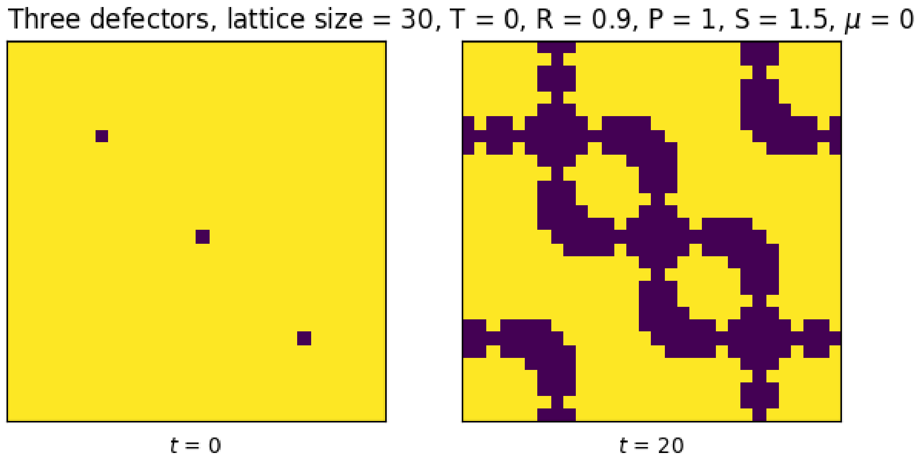

Figure 11: Yellow = 7, purple = 0.

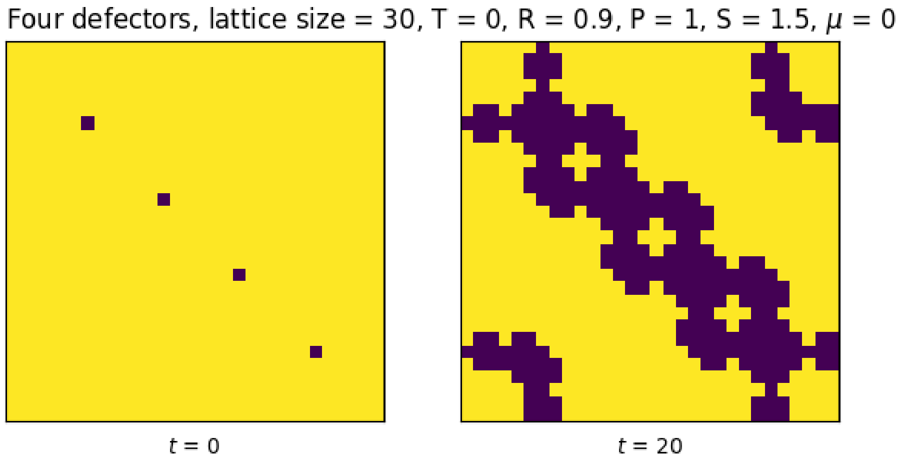

Figure 12: Yellow = 7, purple = 0.

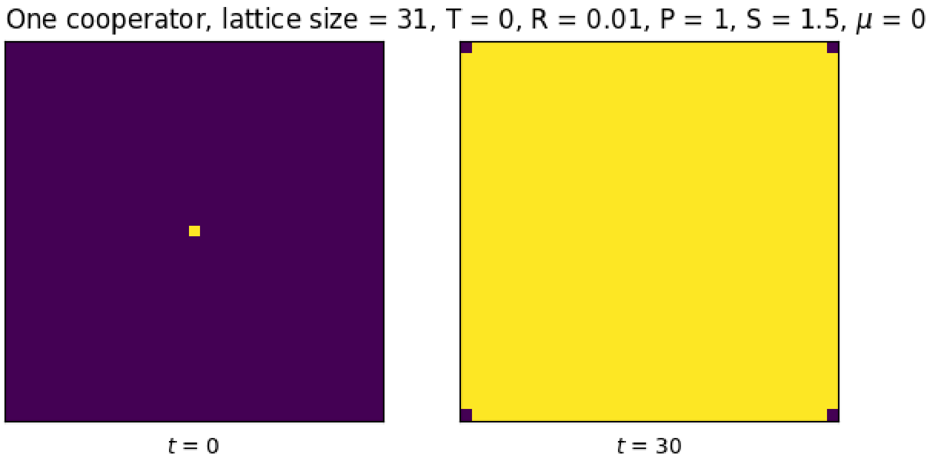

Figure 13: Yellow = 7, purple = 0.

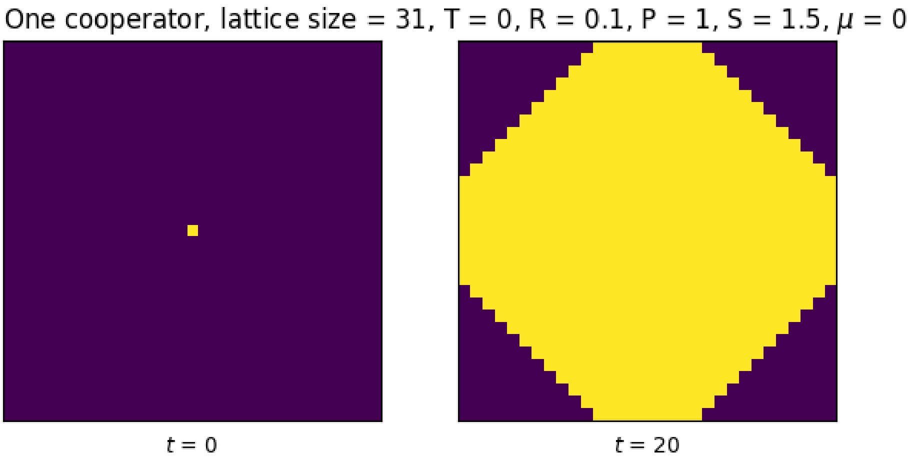

Figure 14: Yellow = 7, purple = 0.

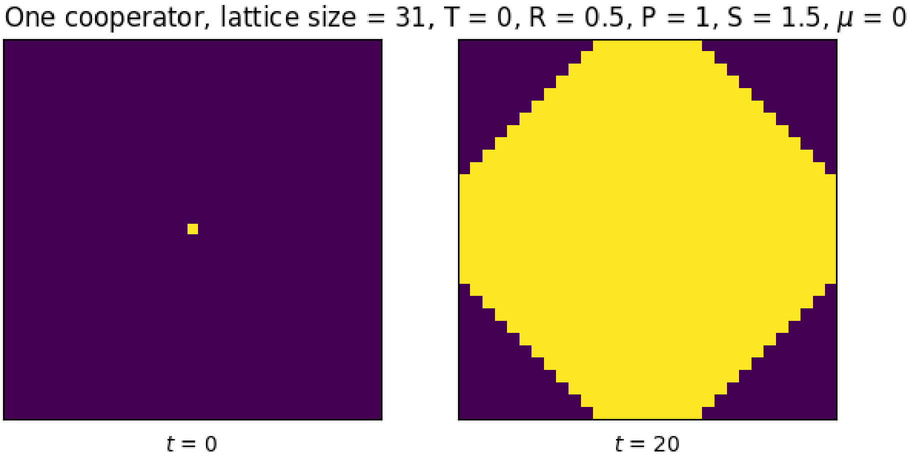

Figure 15: Yellow = 7, purple = 0.

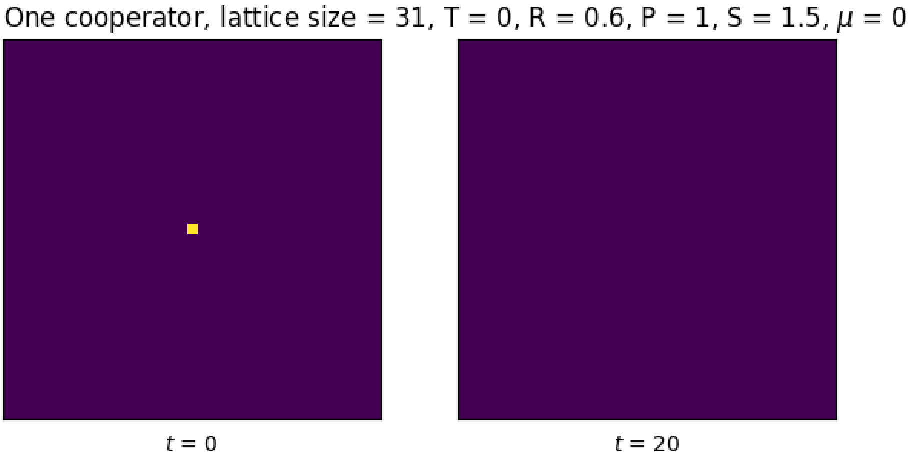

Figure 16: Yellow = 7, purple = 0.

13.3: a, b, c, d, e

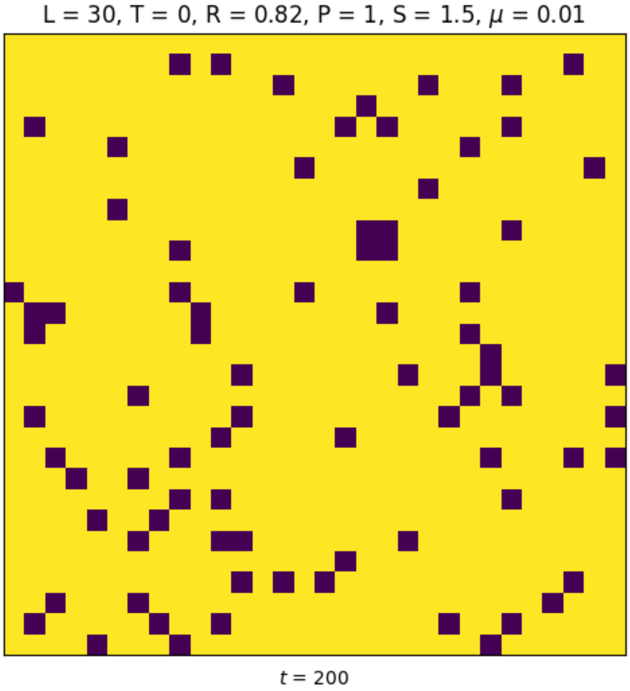

Figure 17: Yellow = 7, purple = 0.

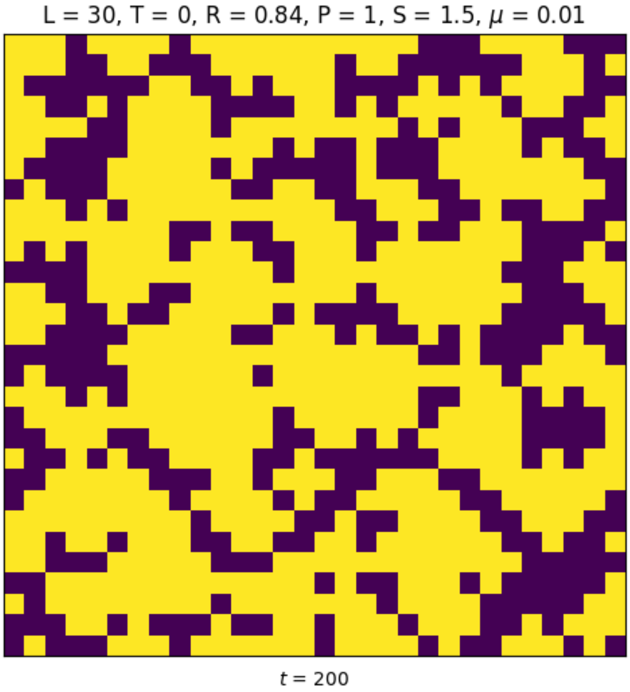

Figure 18: Yellow = 7, purple = 0.

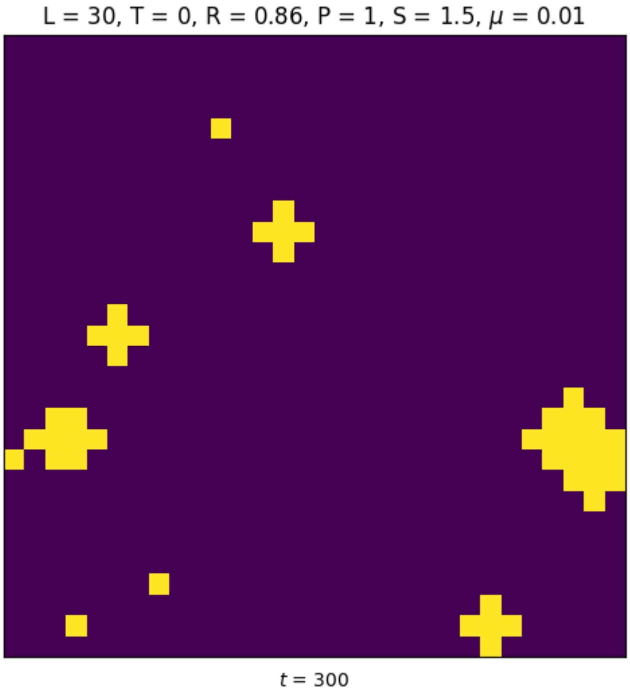

Figure 19: Yellow = 7, purple = 0.

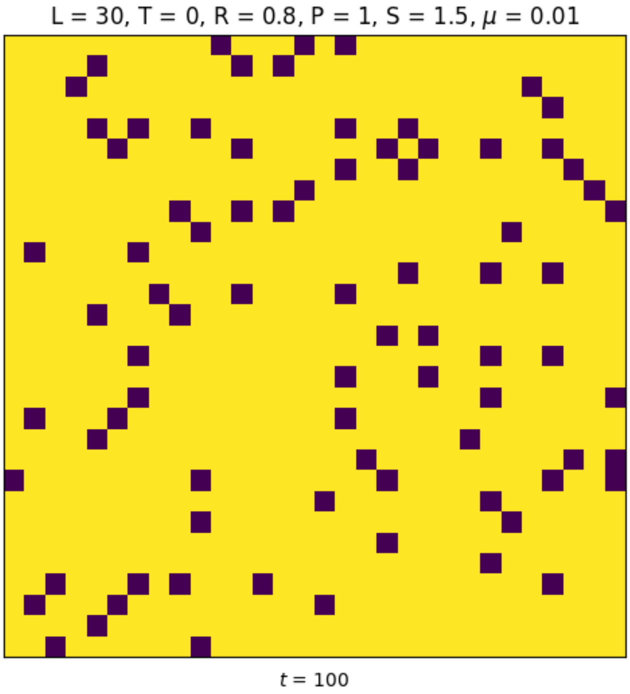

Figure 20: Yellow = 7, purple = 0.

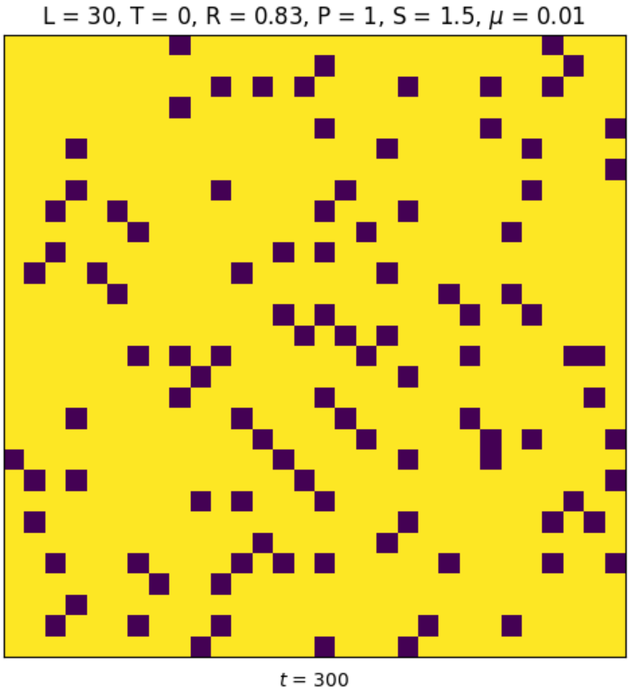

Figure 21: Yellow = 7, purple = 0.

Shift Between R = 0.83 and R = 0.835

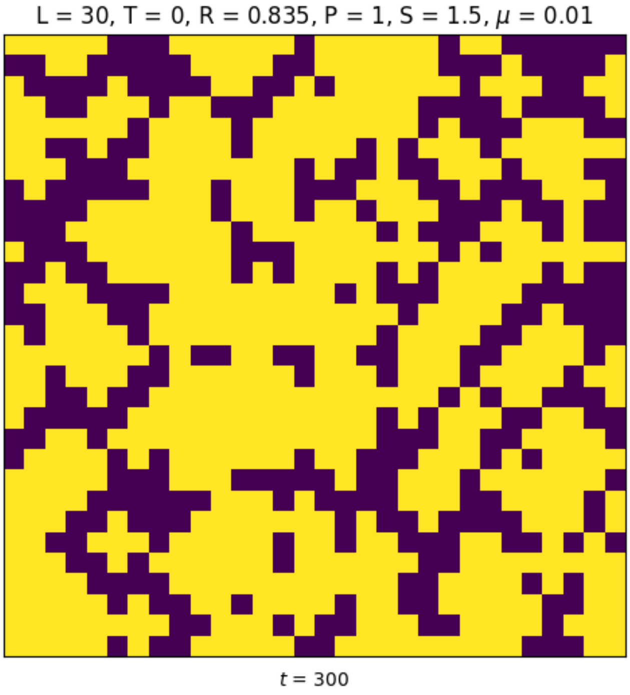

Figure 22: Yellow = 7, purple = 0.

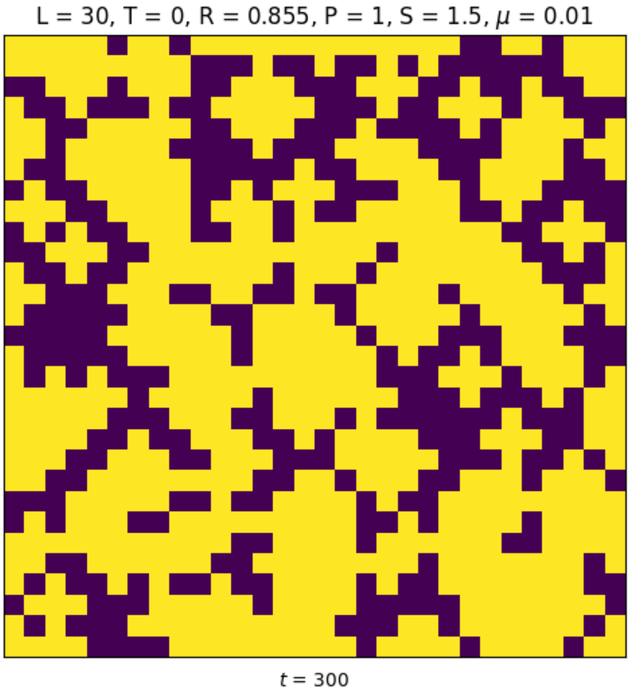

Figure 23: Yellow = 7, purple = 0.

Shift Between R = 0.855 and R = 0.8575

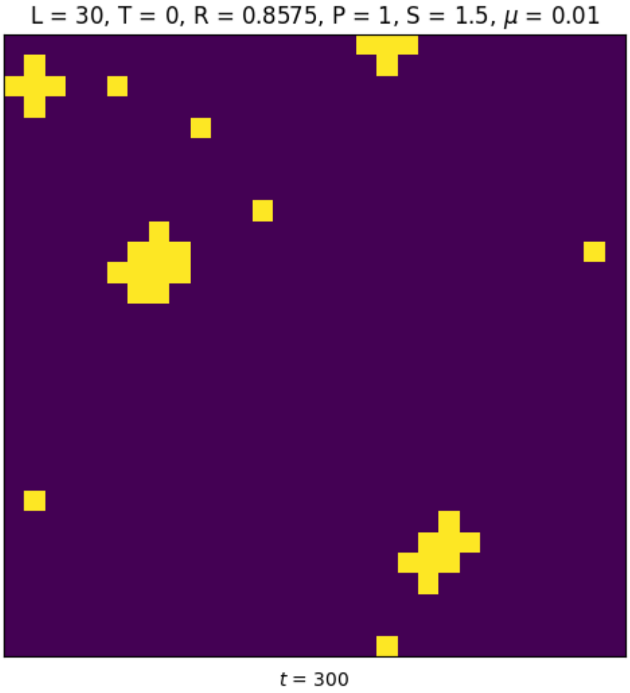

Figure 24: Yellow = 7, purple = 0.

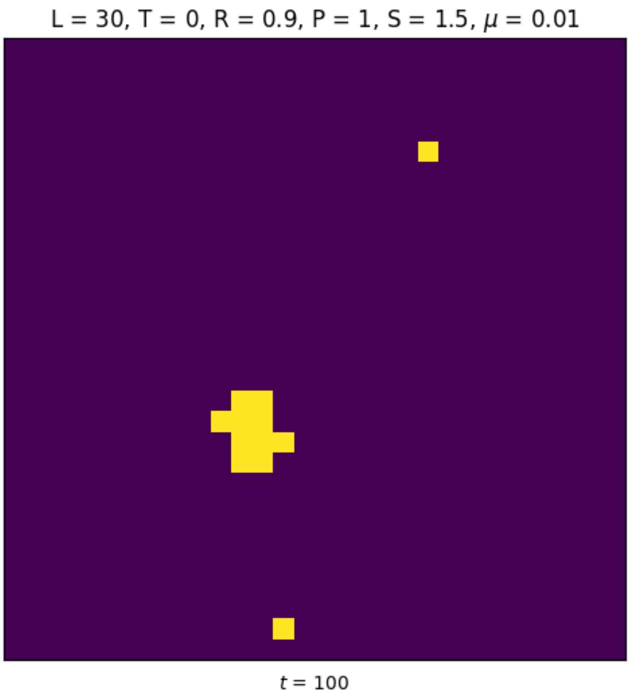

Figure 25: Yellow = 7, purple = 0.

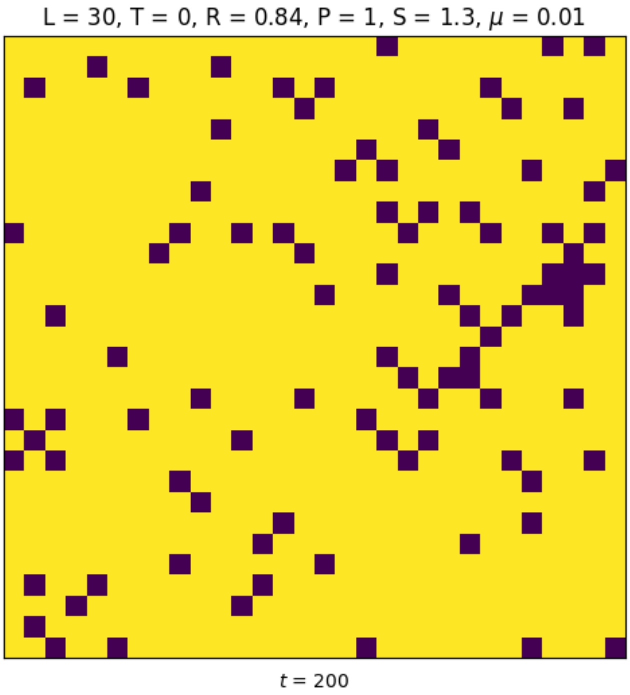

Figure 26: Yellow = 7, purple = 0.

Shift Between S = 1.3 and S = 1.4.

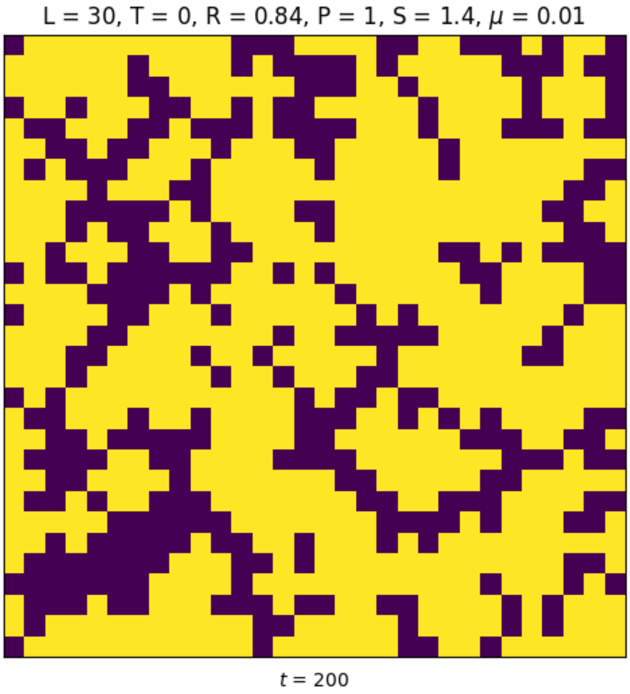

Figure 27: Yellow = 7, purple = 0.

Shifts slowly between S = 1.7 and S = 2.5. Major shift at S = 2.5.

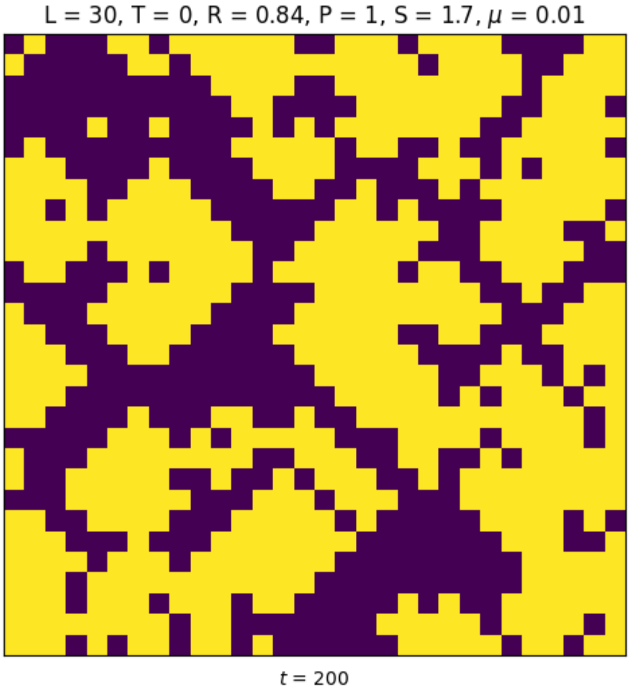

Figure 28: Yellow = 7, purple = 0.

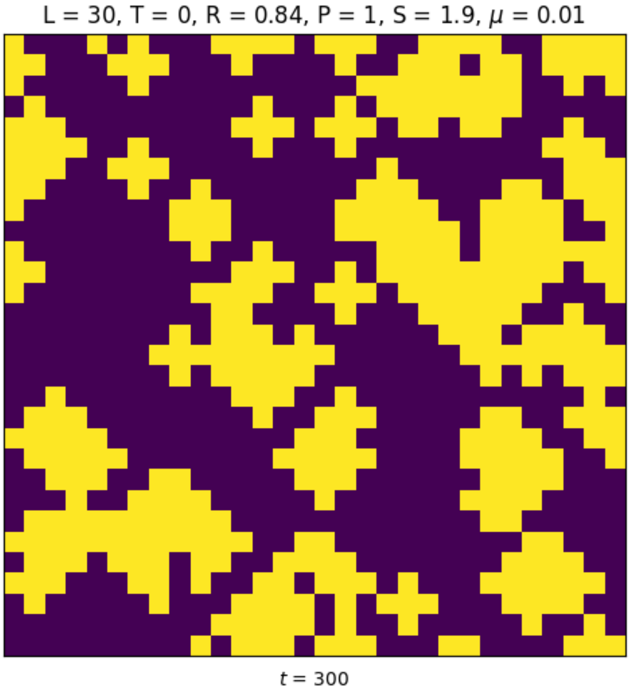

Figure 29: Yellow = 7, purple = 0.

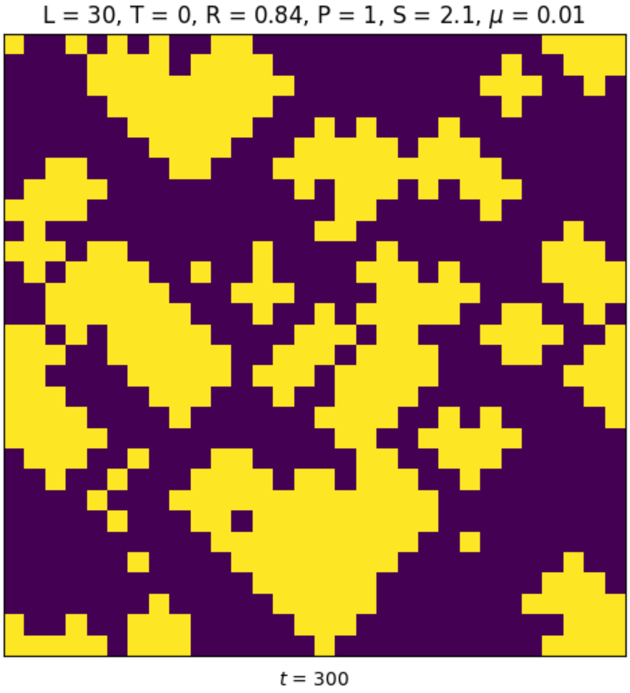

Figure 30: Yellow = 7, purple = 0.

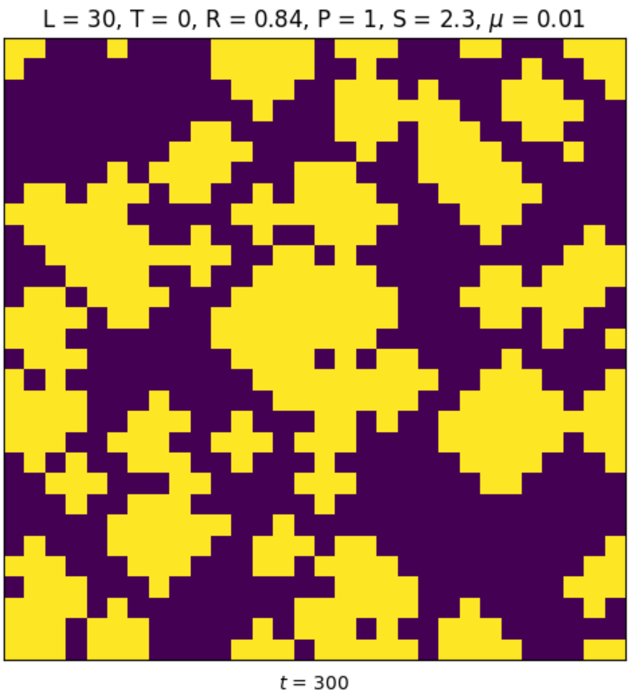

Figure 31: Yellow = 7, purple = 0.

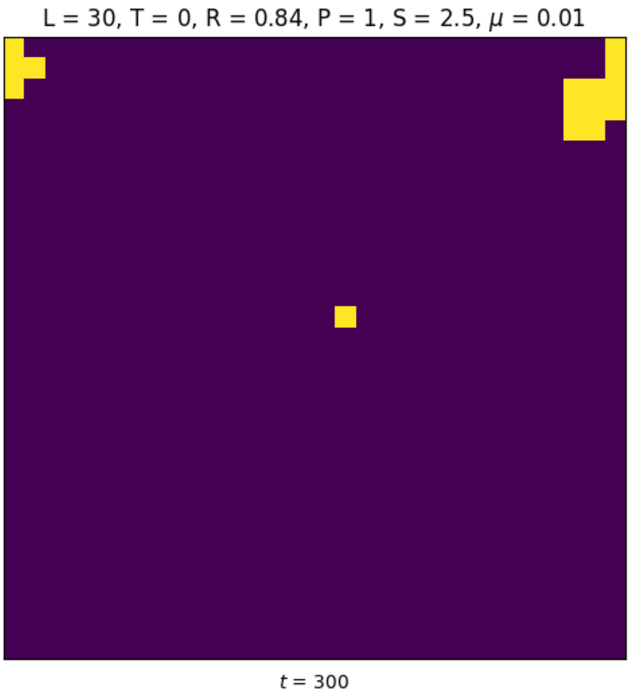

Figure 32: Yellow = 7, purple = 0.

13.4: a, b, c

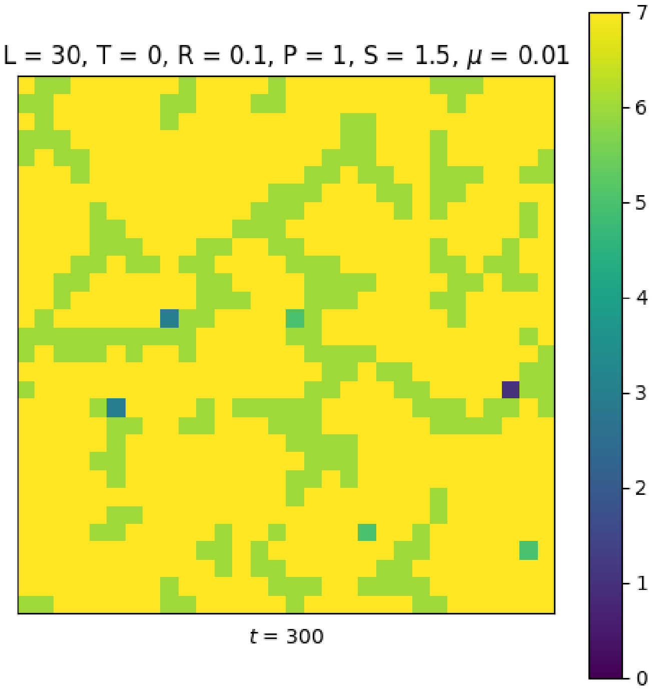

Figure 33

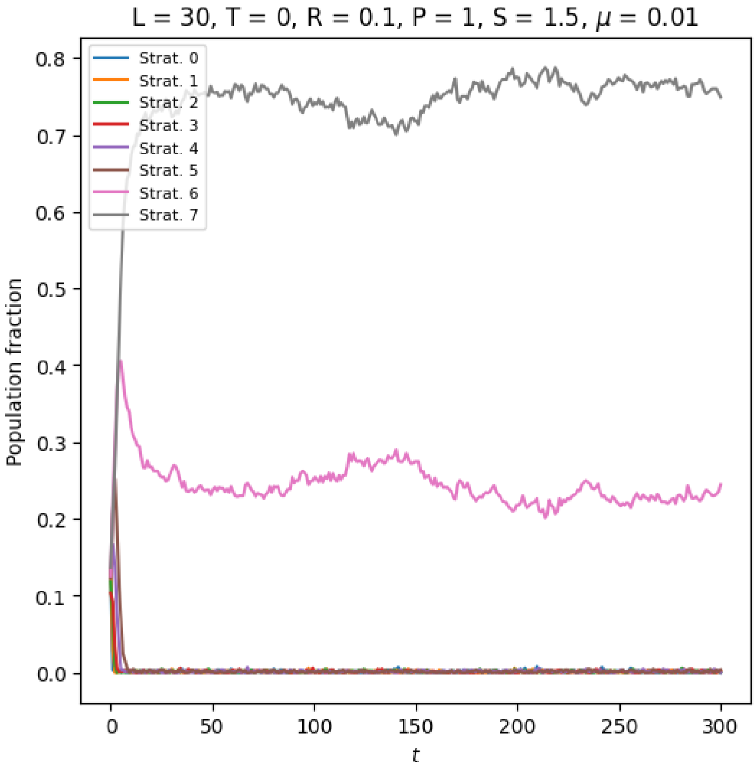

Figure 34

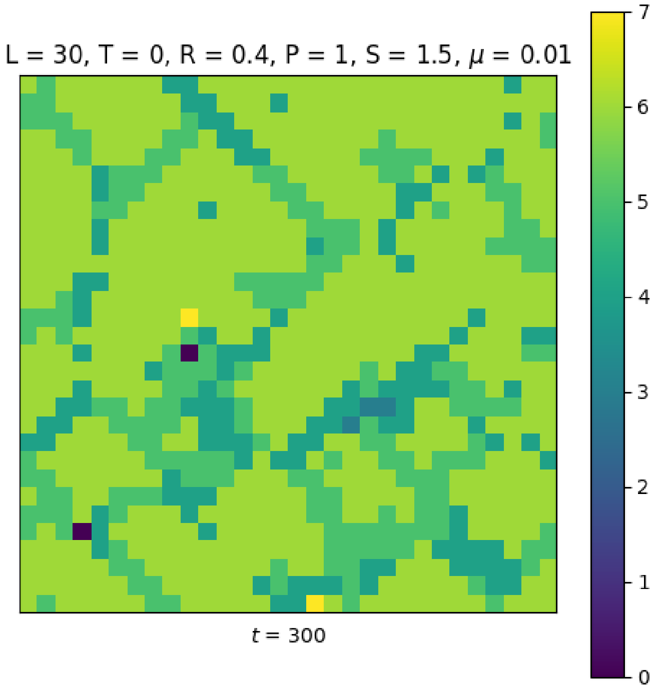

Figure 35

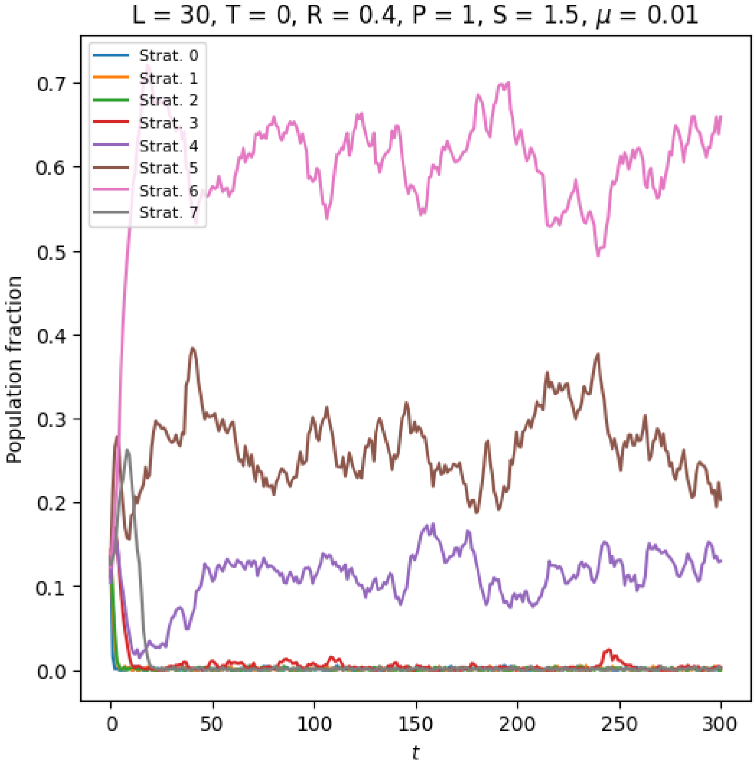

Figure 36

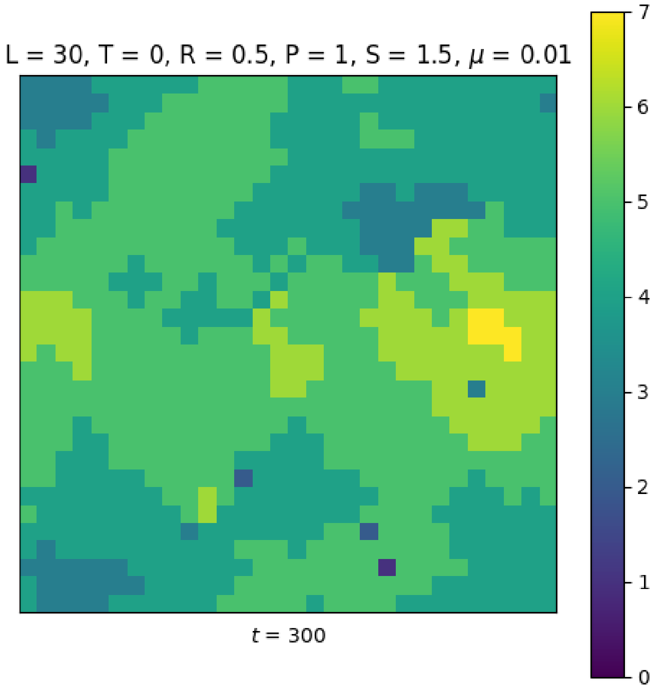

Figure 37

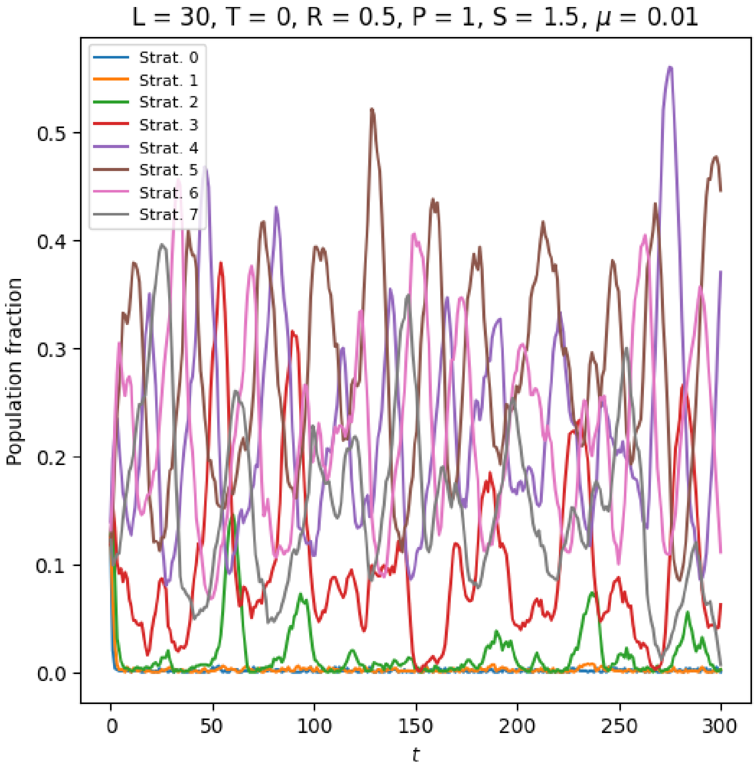

Figure 38

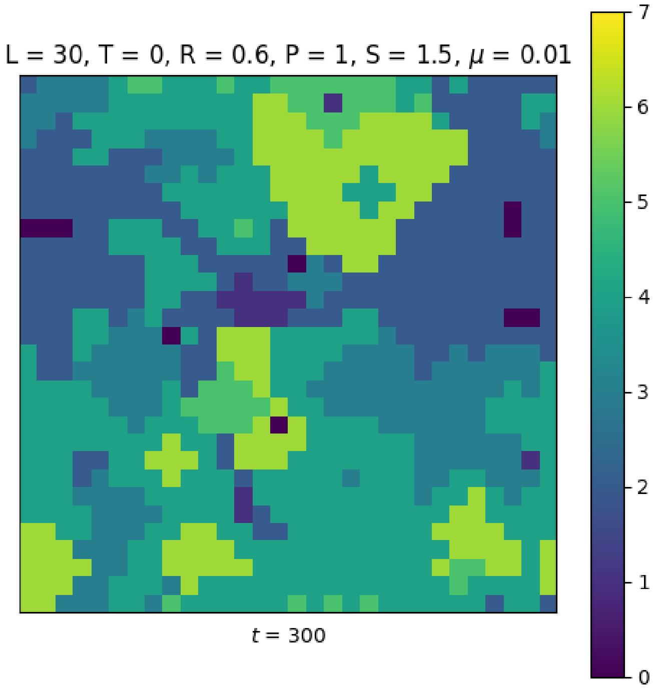

Figure 39

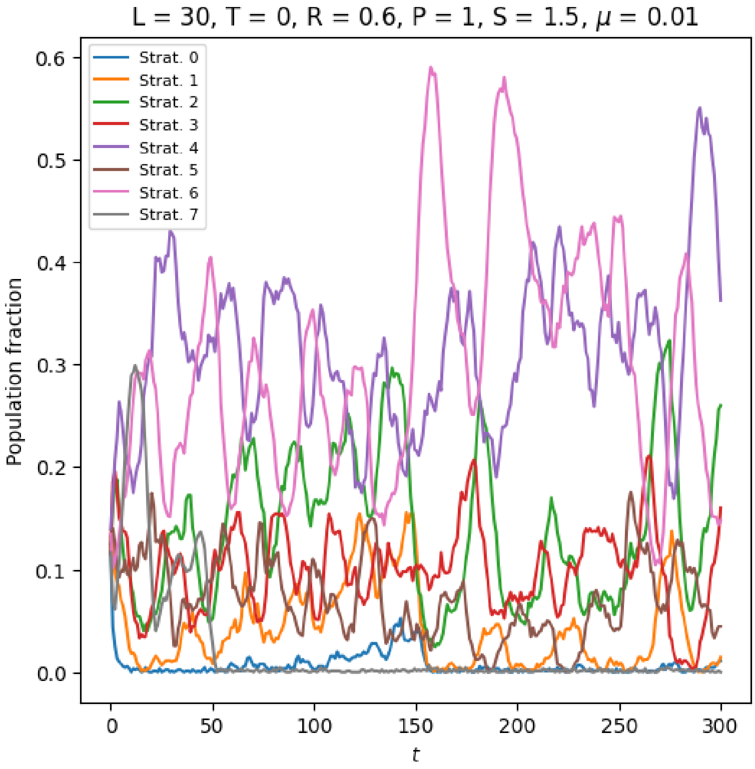

Figure 40

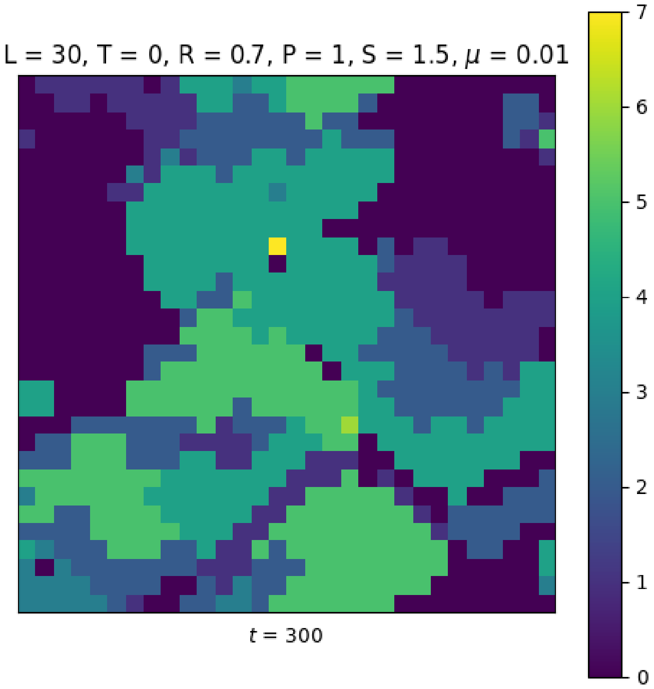

Figure 41

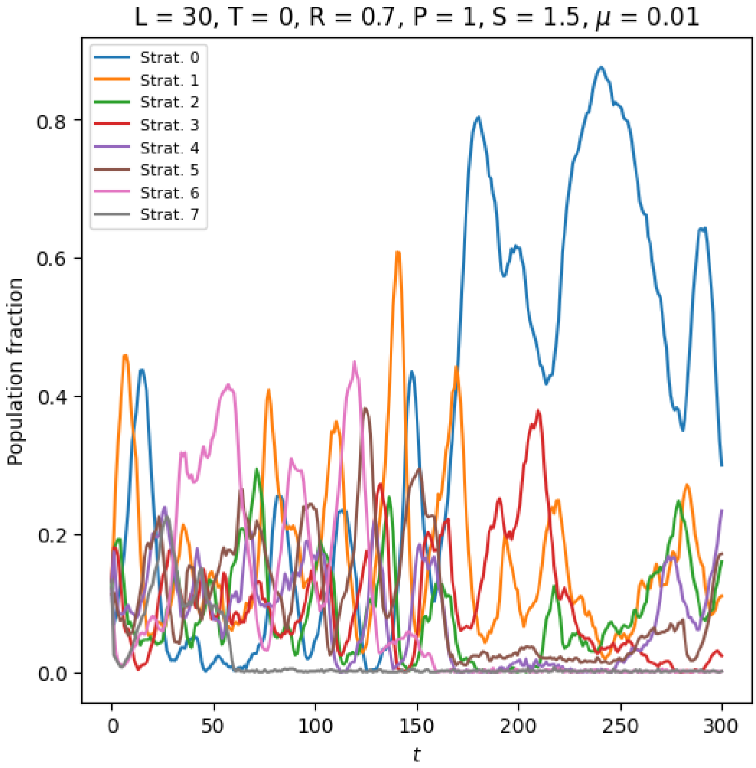

Figure 42

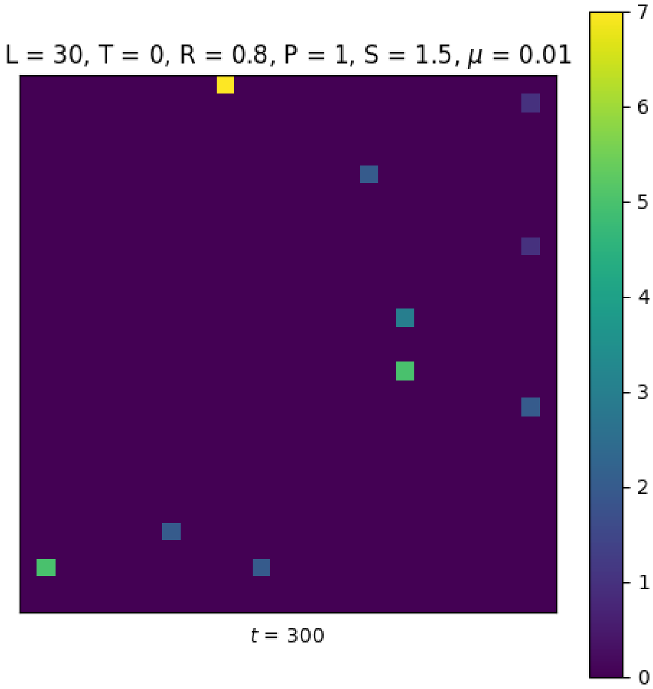

Figure 43

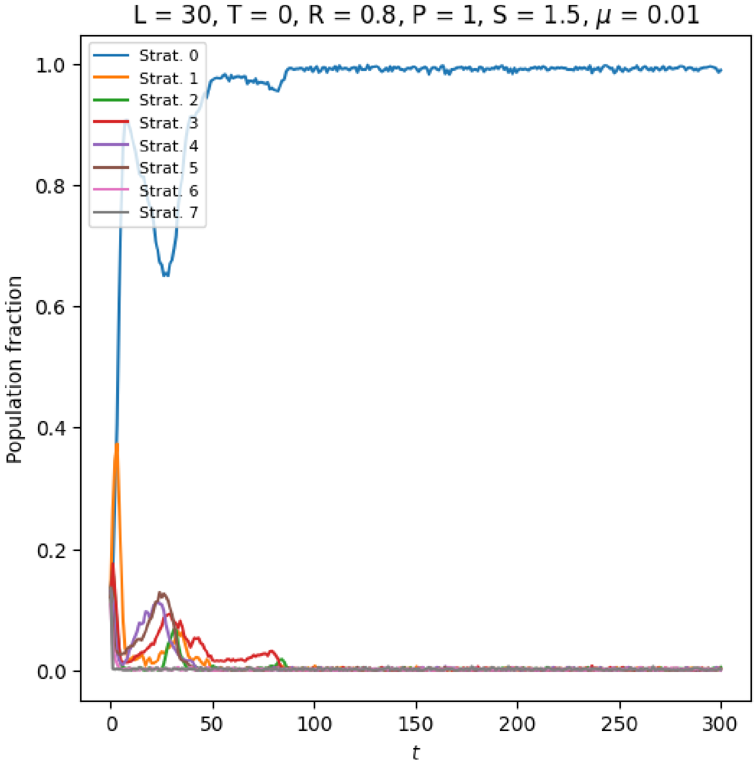

Figure 44

The larger R is (punishment for cooperation) the more the population defects. The smaller R is, the more the population cooperates. Stable strategies are: defect every time, cooperate almost all the time to all the time. Which strategy that will become stable depends on R. Cooperation and defection fluctuates past each other continuously if R is in between 0.5 and 0.75.

13.5: a, b

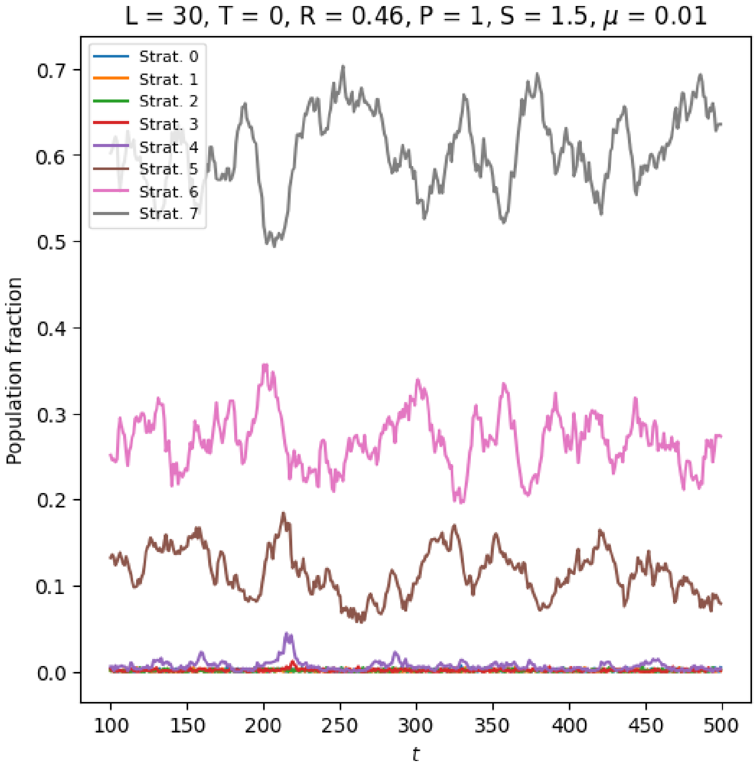

Figure 45: No competition.

Table 1: Variances for R = 0.46, S = 1.5.

n	σ_n^2
0	1.24
1	1.21
2	1.11
3	1.76
4	27.53
5	596.1
6	825.52
7	1660.53
Sum	3115

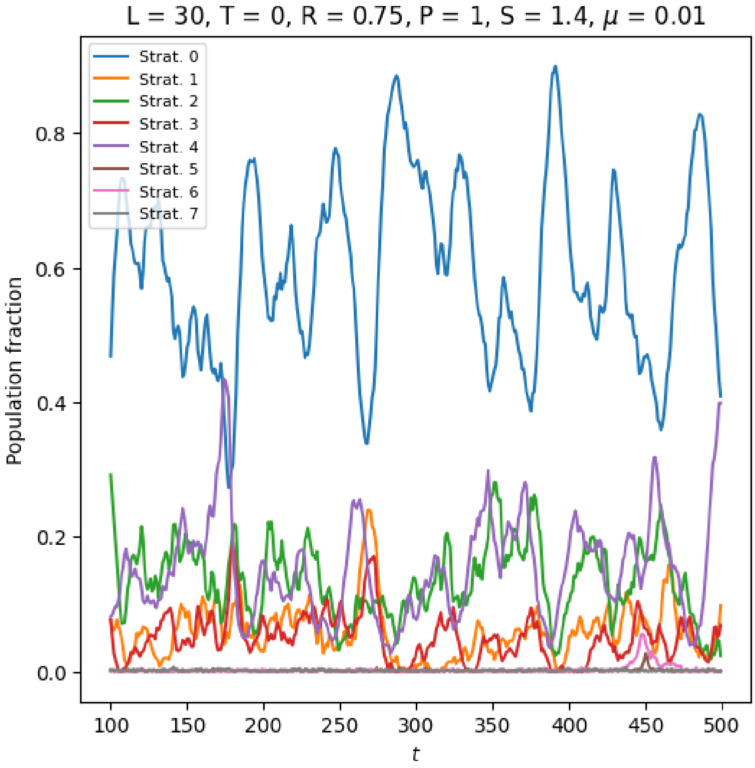

Figure 46: No competition.

Table 2: Variances for R = 0.75, S = 1.4.

n	σ_n^2
0	14344.6
1	1349.99
2	2460.17
3	993.29
4	4655.47
5	3.83
6	37.44
7	1.15
Sum	23845.94

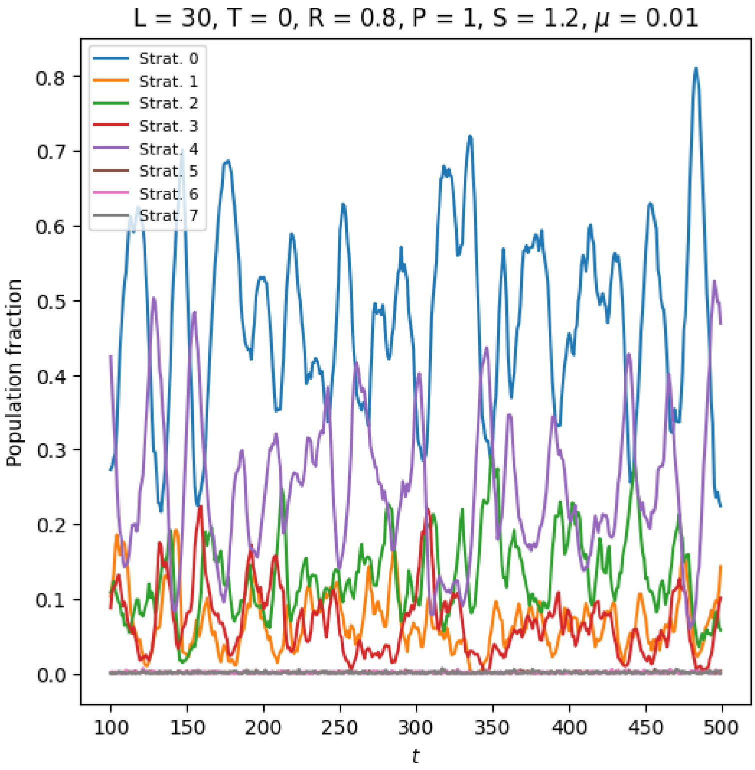

Figure 47: Competition between strategies 0 and 4 starting to become visible.

Table 3: Variances for R = 0.8, S = 1.2.

n	σ_n^2
0	12301.95
1	1174.64
2	2342.39
3	1630.79
4	8088.33
5	1.1
6	1.2
7	1.23
Sum	25541.63

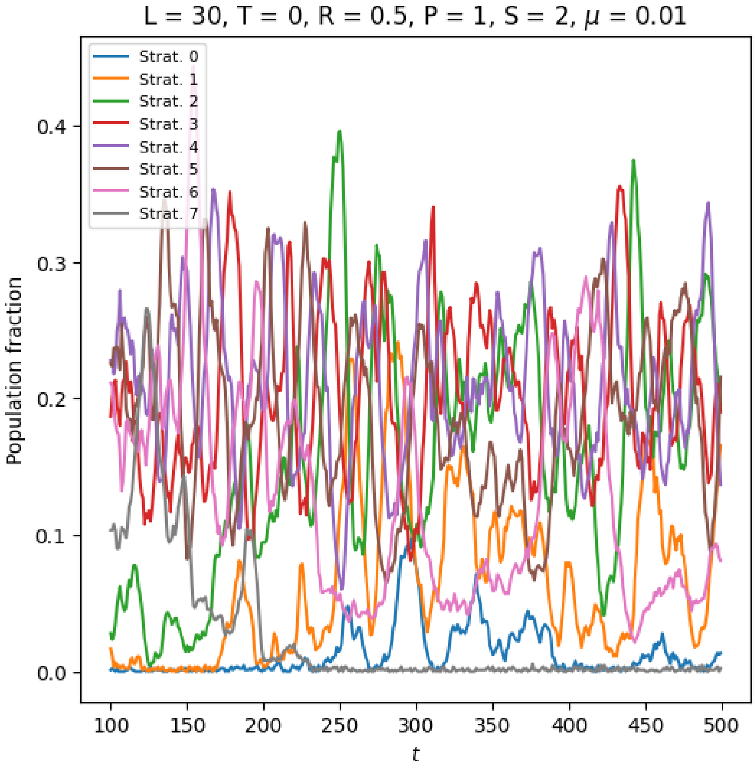

Figure 48: Competition between most strategies are visible.

Table 4: Variances for R = 0.5, S = 2.

n	σ_n^2
0	292.8
1	2889.92
2	6212.41
3	2850.66
4	2638.49
5	3235.15
6	5608.2
7	2307.93
Sum	26035.56

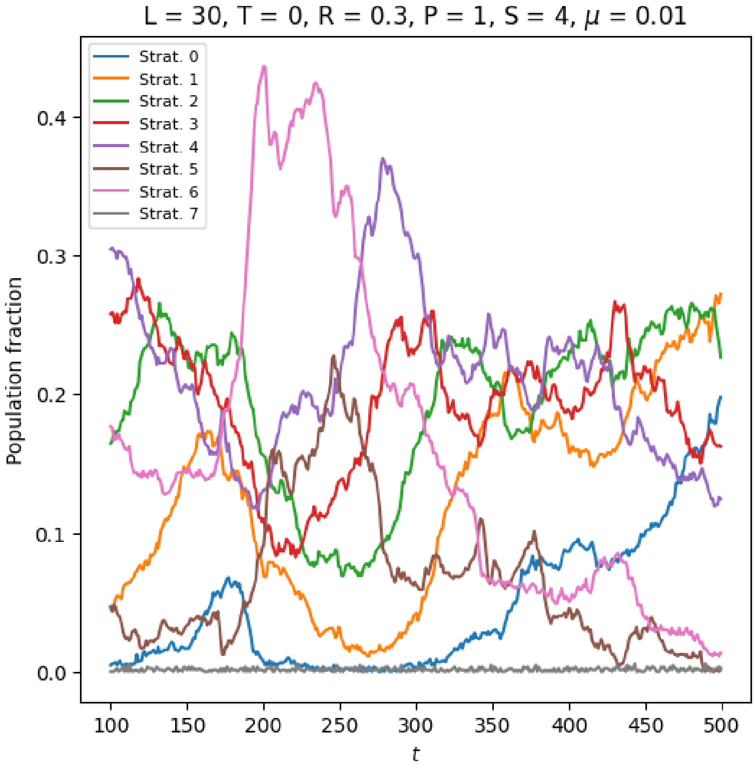

Figure 49: Competition between most strategies are visible here too.

Table 5: Variances for R = 0.3, S = 4.

n	σ_n^2
0	1896.55
1	4212.72
2	2837.26
3	1844.99
4	2647.15
5	2382.78
6	11940.04
7	1.21
Sum	27762.7

Based on these observations we can conclude that there's active competition between populations if

$$\sum_{n=0}^{N} \sigma_n^2 >\approx 26000$$

.

```python
# Exercise 13.1a
import numpy as np
import matplotlib.pyplot as plt
import sys

N = 10
T = 0
R = 0.5
P = 1
S = 1.5
m_strat = 9
n_strats = np.linspace(0,N,N+1)
no_years_array = []
coop = True
defect = False

for n_strat in n_strats:

    m_previous = coop
    n_previous = coop
    no_years = 0

    for round in range(1,N+1):

        if round <= n_strat and m_previous == coop:
            n = coop
        else:
            n = defect

        if round <= m_strat and n_previous == coop:
            m = coop
        else:
            m = defect

        if n == coop and m == coop:
            no_years += R
        elif n == coop and m == defect:
            no_years += S
        elif n == defect and m == coop:
            no_years += T
        elif n == defect and m == defect:
            no_years += P

        n_previous = n
        m_previous = m

    no_years_array.append(no_years)

fig,ax = plt.subplots(figsize=(6,6))
ax.plot(n_strats, no_years_array, 'o', markersize=10, label='Years in prision for n')
ax.plot([m_strat,m_strat], [4,10], '--', color='black', label='Strategy for m')
ax.set_xlabel('$n$')
ax.set_ylabel('Years in prison')
ax.set_ylim(4,9.5)
ax.set_box_aspect(1)

plt.legend(loc="lower left",fontsize=8)
plt.savefig('exercise_13.1a_m=9.png', bbox_inches='tight')
plt.show()
```

```python
# Exercise 13.1bc
import numpy as np
import matplotlib.pyplot as plt
import sys

N = 10
T = 0
R = 0.5
P = 1
S = 1.5
m_strats = np.linspace(0,N,N+1)
n_strats = np.linspace(0,N,N+1)
no_years_array = np.zeros((N+1, N+1))
coop = True
defect = False

for m_strat in m_strats:
    for n_strat in n_strats:

        m_previous = coop
        n_previous = coop
        no_years = 0

        for round in range(1,N+1):

            if round <= n_strat and m_previous == coop:
                n = coop
            else:
                n = defect

            if round <= m_strat and n_previous == coop:
                m = coop
            else:
                m = defect

            if n == coop and m == coop:
                no_years += R
            elif n == coop and m == defect:
                no_years += S
            elif n == defect and m == coop:
                no_years += T
            elif n == defect and m == defect:
                no_years += P

            n_previous = n
            m_previous = m

        no_years_array[int(n_strat), int(m_strat)] = no_years

fig,ax = plt.subplots(figsize=(6,6))
x,y = np.meshgrid(m_strats, n_strats)
years_min, years_max = no_years_array.min(), no_years_array.max()
map = ax.pcolormesh(x, y, no_years_array, cmap='RdBu', vmin=years_min, vmax=years_max)
fig.colorbar(map, ax=ax)


# ax.plot(n_strats, no_years_array, 'o', markersize=10)
# ax.plot([m_strat,m_strat], [4,10], '--', color='black', label='Strategy for m')
```

```python
ax.set_title('Years in prison')
ax.set_xlabel('$m$')
ax.set_ylabel('$n$')
ax.set_box_aspect(1)

# plt.legend(loc="lower left",fontsize=8)
plt.savefig('exercise_13.1c_R' + str(R) + '_S' + str(S) + '.png', bbox_inches='tight')
plt.show()
```

```python
# Exercise 13.2abcd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import sys

def play_game(self_strat, neighbor_strat):

    coop = True
    defect = False
    self_previous = coop
    neighbor_previous = coop
    self_punishment = 0

    for round in range(1,N+1):

        if round <= self_strat and neighbor_previous == coop:
            self_action = coop
        else:
            self_action = defect

        if round <= neighbor_strat and self_previous == coop:
            neighbor_action = coop
        else:
            neighbor_action = defect

        if self_action == coop and neighbor_action == coop:
            self_punishment += R
        elif self_action == coop and neighbor_action == defect:
            self_punishment += S
        elif self_action == defect and neighbor_action == coop:
            self_punishment += T
        elif self_action == defect and neighbor_action == defect:
            self_punishment += P

        self_previous = self_action
        neighbor_previous = neighbor_action

    return self_punishment

N = 7
T = 0
R = 0.99
P = 1
S = 1.5
mu = 0
timesteps = 20

L = 30
strat_array = np.ones((L,L))*0
strat_array[int(L/2),int(L/2)] = N
# strat_array[int(2*L/5),int(2*L/5)] = 0
# strat_array[int(-L/5),int(-L/5)] = 0
# strat_array[int(-2*L/5),int(-2*L/5)] = 0
# strat_array[int(L/5),int(L/5)] = 0
new_strat_array = strat_array.copy()
strat_array_t0 = strat_array.copy()

# Animation
```

```python
fig1, ax = plt.subplots()
ims = []
im = ax.imshow(strat_array_t0)
ims.append([im])

for t in range(1,timesteps):

    P_array = np.zeros_like(strat_array)
    next_neighbor = np.roll(np.arange(L),-1)
    previous_neighbor = np.roll(np.arange(L),1)

    # Accumulate punishment for every agent i,j
    for i in range(L):
        for j in range(L):

            # Punishment for Von Neumann neighbors and self
            pSelf = play_game(strat_array[i,j], strat_array[i,j])
            pUp = play_game(strat_array[i,j], strat_array[previous_neighbor[i],j])
            pLeft = play_game(strat_array[i,j], strat_array[i,previous_neighbor[j]])
            pDown = play_game(strat_array[i,j], strat_array[next_neighbor[i],j])
            pRight = play_game(strat_array[i,j], strat_array[i,next_neighbor[j]])
            P_array[i,j] = pUp + pLeft + pDown + pRight

    # Compute new strategies for every agent
    for i in range(L):
        for j in range(L):

            r = np.random.uniform()
            if r < mu:
                new_strat_array[i,j] = np.random.choice([0,N])
            else:
                agent_p = [P_array[i,j],P_array[next_neighbor[i],j],P_array[previous_neighbor[i],j],P_array[i,next_neighbor[j]],P_array[i,previous_neighbor[j]]]
                agent_strat = [strat_array[i,j],strat_array[next_neighbor[i],j],strat_array[previous_neighbor[i],j],strat_array[i,next_neighbor[j]],strat_array[i,previous_neighbor[j]]]
                new_strat_array[i,j] = np.random.choice([agent_strat[k] for k in range(len(agent_p)) if agent_p[k] == np.min(agent_p)])

            # pMin = np.argmin([P_array[i,j],P_array[next_neighbor[i],j],P_array[previous_neighbor[i],j],P_array[i,next_neighbor[j]],P_array[i,previous_neighbor[j]]])
            # if pMin == 0:
            #     new_strat_array[i,j] = strat_array[i,j]
            # if pMin == 1:
            #     new_strat_array[i,j] = strat_array[next_neighbor[i],j]
            # elif pMin == 2:
            #     new_strat_array[i,j] = strat_array[previous_neighbor[i],j]
            # elif pMin == 3:
            #     new_strat_array[i,j] = strat_array[i,next_neighbor[j]]
            # elif pMin == 4:
            #     new_strat_array[i,j] = strat_array[i,previous_neighbor[j]]

    strat_array = new_strat_array.copy()
    # Images for animation
    im = ax.imshow(new_strat_array.copy(), animated=True)
    ims.append([im])

fig, axs = plt.subplots(1,2,figsize=(7,7))
```

```python
title = 'One defector, lattice size = {}, T = {}, R = {}, P = {}, S = {}, $\mu$ = {}'.format(L,T,R,P,S,mu)

axs[0].set_title(title, loc='left')
axs[0].imshow(strat_array_t0)
axs[0].set_yticks(())
axs[0].set_xticks(())
axs[0].set_xlabel('$t$ = 0')

axs[1].imshow(strat_array)
axs[1].set_yticks(())
axs[1].set_xticks(())
axs[1].set_xlabel('$t$ = {}'.format(timesteps))

ani = animation.ArtistAnimation(fig1, ims, interval=5, blit=True)
writergif = animation.PillowWriter(fps=30)
# ani.save('exercise_13.2_1def_R{}.gif'.format(R), writer=writergif)

# plt.savefig('exercise_13.2_1def_R{}.png'.format(R), bbox_inches='tight')
plt.show()
```

```python
# Exercise 13.3abcde
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import sys

def play_game(self_strat, neighbor_strat):

    coop = True
    defect = False
    self_previous = coop
    neighbor_previous = coop
    self_punishment = 0

    for round in range(1,N+1):

        if round <= self_strat and neighbor_previous == coop:
            self_action = coop
        else:
            self_action = defect

        if round <= neighbor_strat and self_previous == coop:
            neighbor_action = coop
        else:
            neighbor_action = defect

        if self_action == coop and neighbor_action == coop:
            self_punishment += R
        elif self_action == coop and neighbor_action == defect:
            self_punishment += S
        elif self_action == defect and neighbor_action == coop:
            self_punishment += T
        elif self_action == defect and neighbor_action == defect:
            self_punishment += P

        self_previous = self_action
        neighbor_previous = neighbor_action

    return self_punishment

N = 7
T = 0
R = 0.84
P = 1
S = 2.3
mu = 0.01
timesteps = 300

L = 30
strat_array = np.ones((L,L))*N
# strat_array[int(L/2),int(L/2)] = 0
# strat_array[int(L/5),int(L/5)] = 0
# strat_array[int(-L/5),int(-L/5)] = 0
# strat_array[int(-2*L/5),int(-2*L/5)] = 0
# strat_array[int(L/2),int(L/2)] = 7
new_strat_array = strat_array.copy()
strat_array_t0 = strat_array.copy()

# Animation
```

```python
fig1, ax1 = plt.subplots()
ims = []
im = ax1.imshow(strat_array_t0)
ims.append([im])

for t in range(1,timesteps):

    P_array = np.zeros_like(strat_array)
    next_neighbor = np.roll(np.arange(L),-1)
    previous_neighbor = np.roll(np.arange(L),1)

    # Accumulate punishment for every agent i,j
    for i in range(L):
        for j in range(L):

            # Punishment for Von Neumann neighbors and self
            pSelf = play_game(strat_array[i,j], strat_array[i,j])
            pUp = play_game(strat_array[i,j], strat_array[previous_neighbor[i],j])
            pLeft = play_game(strat_array[i,j], strat_array[i,previous_neighbor[j]])
            pDown = play_game(strat_array[i,j], strat_array[next_neighbor[i],j])
            pRight = play_game(strat_array[i,j], strat_array[i,next_neighbor[j]])
            P_array[i,j] = pUp + pLeft + pDown + pRight

    # Compute new strategies for every agent
    for i in range(L):
        for j in range(L):

            r = np.random.uniform()
            if r < mu:
                new_strat_array[i,j] = np.random.choice([0,N])
            else:
                agent_p = [P_array[i,j],P_array[next_neighbor[i],j],P_array[previous_neighbor[i],j],P_array[i,next_neighbor[j]],P_array[i,previous_neighbor[j]]]
                agent_strat = [strat_array[i,j],strat_array[next_neighbor[i],j],strat_array[previous_neighbor[i],j],strat_array[i,next_neighbor[j]],strat_array[i,previous_neighbor[j]]]
                new_strat_array[i,j] = np.random.choice([agent_strat[k] for k in range(len(agent_p)) if agent_p[k] == np.min(agent_p)])

            # pMin = np.argmin([P_array[i,j],P_array[next_neighbor[i],j],P_array[previous_neighbor[i],j],P_array[i,next_neighbor[j]],P_array[i,previous_neighbor[j]]])
            # if pMin == 0:
            #     new_strat_array[i,j] = strat_array[i,j]
            # if pMin == 1:
            #     new_strat_array[i,j] = strat_array[next_neighbor[i],j]
            # elif pMin == 2:
            #     new_strat_array[i,j] = strat_array[previous_neighbor[i],j]
            # elif pMin == 3:
            #     new_strat_array[i,j] = strat_array[i,next_neighbor[j]]
            # elif pMin == 4:
            #     new_strat_array[i,j] = strat_array[i,previous_neighbor[j]]

    strat_array = new_strat_array.copy()
    # Images for animation
    im = ax1.imshow(new_strat_array.copy(), animated=True)
    ims.append([im])

fig2, ax2 = plt.subplots(figsize=(6,6))
```

```python
title = 'L = {}, T = {}, R = {}, P = {}, S = {}, $\mu$ = {}'.format(L,T,R,P,S,mu)

ax2.set_title(title)
ax2.imshow(strat_array_t0)
ax2.set_yticks(())
ax2.set_xticks(())
ax2.set_xlabel('$t$ = 0')

ax2.imshow(strat_array)
ax2.set_yticks(())
ax2.set_xticks(())
ax2.set_xlabel('$t$ = {}'.format(timesteps))

ani = animation.ArtistAnimation(fig1, ims, interval=5, blit=True)
writergif = animation.PillowWriter(fps=30)
ani.save('exercise_13.3_S{}.gif'.format(S), writer=writergif)

plt.savefig('exercise_13.3_S{}.png'.format(S), bbox_inches='tight')
plt.show()
```

```python
# Exercise 13.4abcd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import sys

def play_game(self_strat, neighbor_strat):

    coop = True
    defect = False
    self_previous = coop
    neighbor_previous = coop
    self_punishment = 0

    for round in range(1,N+1):

        if round <= self_strat and neighbor_previous == coop:
            self_action = coop
        else:
            self_action = defect

        if round <= neighbor_strat and self_previous == coop:
            neighbor_action = coop
        else:
            neighbor_action = defect

        if self_action == coop and neighbor_action == coop:
            self_punishment += R
        elif self_action == coop and neighbor_action == defect:
            self_punishment += S
        elif self_action == defect and neighbor_action == coop:
            self_punishment += T
        elif self_action == defect and neighbor_action == defect:
            self_punishment += P

        self_previous = self_action
        neighbor_previous = neighbor_action

    return self_punishment

N = 7
T = 0
R = 0.65
P = 1
S = 1.5
mu = 0.01
timesteps = 300

L = 30
strat_array = np.random.randint(0,N+1,size=(L,L))
new_strat_array = strat_array.copy()
strat_array_t0 = strat_array.copy()

# Distribution fraction
no_0 = np.zeros(timesteps)
no_1 = no_0.copy()
no_2 = no_0.copy()
no_3 = no_0.copy()
no_4 = no_0.copy()
```

```python
no_5 = no_0.copy()
no_6 = no_0.copy()
no_7 = no_0.copy()

no_0[0] = np.count_nonzero(strat_array == 0)
no_1[0] = np.count_nonzero(strat_array == 1)
no_2[0] = np.count_nonzero(strat_array == 2)
no_3[0] = np.count_nonzero(strat_array == 3)
no_4[0] = np.count_nonzero(strat_array == 4)
no_5[0] = np.count_nonzero(strat_array == 5)
no_6[0] = np.count_nonzero(strat_array == 6)
no_7[0] = np.count_nonzero(strat_array == 7)


# Animation
fig1, ax1 = plt.subplots()
ims = []
im = ax1.imshow(strat_array_t0, vmin=0, vmax=N, animated=True)
ims.append([im])

for t in range(1,timesteps):

    P_array = np.zeros_like(strat_array)
    next_neighbor = np.roll(np.arange(L),-1)
    previous_neighbor = np.roll(np.arange(L),1)

    # Accumulate punishment for every agent i,j
    for i in range(L):
        for j in range(L):

            # Punishment for Von Neumann neighbors and self
            pSelf = play_game(strat_array[i,j], strat_array[i,j])
            pUp = play_game(strat_array[i,j], strat_array[previous_neighbor[i],j])
            pLeft = play_game(strat_array[i,j], strat_array[i,previous_neighbor[j]])
            pDown = play_game(strat_array[i,j], strat_array[next_neighbor[i],j])
            pRight = play_game(strat_array[i,j], strat_array[i,next_neighbor[j]])
            P_array[i,j] = pUp + pLeft + pDown + pRight

    # Compute new strategies for every agent
    for i in range(L):
        for j in range(L):

            r = np.random.uniform()
            if r < mu:
                new_strat_array[i,j] = np.random.randint(0,N+1)
            else:
                agent_p = [P_array[i,j],P_array[next_neighbor[i],j],P_array[previous_neighbor[i],j],P_array[i,next_neighbor[j]],P_array[i,previous_neighbor[j]]]
                agent_strat = [strat_array[i,j],strat_array[next_neighbor[i],j],strat_array[previous_neighbor[i],j],strat_array[i,next_neighbor[j]],strat_array[i,previous_neighbor[j]]]
                new_strat_array[i,j] = np.random.choice([agent_strat[k] for k in range(len(agent_p)) if agent_p[k] == np.min(agent_p)])

            # pMin = np.argmin([P_array[i,j],P_array[next_neighbor[i],j],P_array[previous_neighbor[i],j],P_array[i,next_neighbor[j]],P_array[i,previous_neighbor[j]]])
            # if pMin == 0:
            #     new_strat_array[i,j] = strat_array[i,j]
            # if pMin == 1:
            #     new_strat_array[i,j] = strat_array[next_neighbor[i],j]
            # elif pMin == 2:
            #     new_strat_array[i,j] = strat_array[previous_neighbor[i],j]
            # elif pMin == 3:
            #     new_strat_array[i,j] = strat_array[i,next_neighbor[j]]
            # elif pMin == 4:
            #     new_strat_array[i,j] = strat_array[i,previous_neighbor[j]]

    strat_array = new_strat_array.copy()

    # Images for animation
    im = ax1.imshow(new_strat_array.copy(), vmin=0, vmax=N, animated=True)
    ims.append([im])

    no_0[t] = np.count_nonzero(strat_array == 0)
    no_1[t] = np.count_nonzero(strat_array == 1)
    no_2[t] = np.count_nonzero(strat_array == 2)
    no_3[t] = np.count_nonzero(strat_array == 3)
    no_4[t] = np.count_nonzero(strat_array == 4)
    no_5[t] = np.count_nonzero(strat_array == 5)
    no_6[t] = np.count_nonzero(strat_array == 6)
    no_7[t] = np.count_nonzero(strat_array == 7)

fig1.colorbar(im, ax=ax1)
ani = animation.ArtistAnimation(fig1, ims, interval=50, blit=True)
writergif = animation.PillowWriter(fps=30)
ani.save('exercise_13.4_R{}.gif'.format(R), writer=writergif)

fig2, ax2 = plt.subplots(figsize=(6,6))
title = 'L = {}, T = {}, R = {}, P = {}, S = {}, $\mu$ = {}'.format(L,T,R,P,S,mu)
ax2.set_title(title)
ax2.imshow(strat_array, vmin=0, vmax=N)
ax2.set_yticks(())
ax2.set_xticks(())
ax2.set_xlabel('$t$ = {}'.format(timesteps))
fig2.colorbar(im, ax=ax2)
plt.savefig('exercise_13.4_R{}.png'.format(R), bbox_inches='tight')

fig3, ax3 = plt.subplots(figsize=(6,6))
t_linspace = np.linspace(0,timesteps,timesteps)
ax3.plot(t_linspace, no_0/(L*L), label='Strat. 0')
ax3.plot(t_linspace, no_1/(L*L), label='Strat. 1')
ax3.plot(t_linspace, no_2/(L*L), label='Strat. 2')
ax3.plot(t_linspace, no_3/(L*L), label='Strat. 3')
ax3.plot(t_linspace, no_4/(L*L), label='Strat. 4')
ax3.plot(t_linspace, no_5/(L*L), label='Strat. 5')
ax3.plot(t_linspace, no_6/(L*L), label='Strat. 6')
ax3.plot(t_linspace, no_7/(L*L), label='Strat. 7')
ax3.set_title(title)
ax3.set_xlabel('$t$')
ax3.set_ylabel('Population fraction')
plt.legend(loc="upper left",fontsize=8)
plt.savefig('exercise_13.4_PF_R{}.png'.format(R), bbox_inches='tight')

plt.show()

# The larger R is (punishment for cooperation) the more the population defects.
# The smaller R is, the more the population cooperates.
```

```python
# Stable strategies are: defect every time, cooperate almost all the time to all the time.
# Which strategy that will become stable depends on R.
# Cooperation and defection fluctuates past each other continiously if R is "right in between".
# Exercise 13.5ab
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import sys

def play_game(self_strat, neighbor_strat):

    coop = True
    defect = False
    self_previous = coop
    neighbor_previous = coop
    self_punishment = 0

    for round in range(1,N+1):

        if round <= self_strat and neighbor_previous == coop:
            self_action = coop
        else:
            self_action = defect

        if round <= neighbor_strat and self_previous == coop:
            neighbor_action = coop
        else:
            neighbor_action = defect

        if self_action == coop and neighbor_action == coop:
            self_punishment += R
        elif self_action == coop and neighbor_action == defect:
            self_punishment += S
        elif self_action == defect and neighbor_action == coop:
            self_punishment += T
        elif self_action == defect and neighbor_action == defect:
            self_punishment += P

        self_previous = self_action
        neighbor_previous = neighbor_action

    return self_punishment

N = 7
T = 0
R = 0.46
P = 1
S = 1.5
mu = 0.01
timesteps = 500

L = 30
strat_array = np.random.randint(0,N+1,size=(L,L))
new_strat_array = strat_array.copy()
strat_array_t0 = strat_array.copy()

# Distribution fraction
t_variance = 0
omit_timesteps = 100
no_0 = np.zeros(timesteps-omit_timesteps)
no_1 = no_0.copy()
no_2 = no_0.copy()
```

```python
no_3 = no_0.copy()
no_4 = no_0.copy()
no_5 = no_0.copy()
no_6 = no_0.copy()
no_7 = no_0.copy()

no_0[0] = np.count_nonzero(strat_array == 0)
no_1[0] = np.count_nonzero(strat_array == 1)
no_2[0] = np.count_nonzero(strat_array == 2)
no_3[0] = np.count_nonzero(strat_array == 3)
no_4[0] = np.count_nonzero(strat_array == 4)
no_5[0] = np.count_nonzero(strat_array == 5)
no_6[0] = np.count_nonzero(strat_array == 6)
no_7[0] = np.count_nonzero(strat_array == 7)


# Animation
fig1, ax1 = plt.subplots()
ims = []
im = ax1.imshow(strat_array_t0, vmin=0, vmax=N, animated=True)
ims.append([im])

for t in range(1,timesteps):

    P_array = np.zeros_like(strat_array)
    next_neighbor = np.roll(np.arange(L),-1)
    previous_neighbor = np.roll(np.arange(L),1)

    # Accumulate punishment for every agent i,j
    for i in range(L):
        for j in range(L):

            # Punishment for Von Neumann neighbors and self
            pSelf = play_game(strat_array[i,j], strat_array[i,j])
            pUp = play_game(strat_array[i,j], strat_array[previous_neighbor[i],j])
            pLeft = play_game(strat_array[i,j], strat_array[i,previous_neighbor[j]])
            pDown = play_game(strat_array[i,j], strat_array[next_neighbor[i],j])
            pRight = play_game(strat_array[i,j], strat_array[i,next_neighbor[j]])
            P_array[i,j] = pUp + pLeft + pDown + pRight

    # Compute new strategies for every agent
    for i in range(L):
        for j in range(L):

            r = np.random.uniform()
            if r < mu:
                new_strat_array[i,j] = np.random.randint(0,N+1)
            else:
                agent_p = [P_array[i,j],P_array[next_neighbor[i],j],P_array[previous_neighbor[i],j],P_array[i,next_neighbor[j]],P_array[i,previous_neighbor[j]]]
                agent_strat = [strat_array[i,j],strat_array[next_neighbor[i],j],strat_array[previous_neighbor[i],j],strat_array[i,next_neighbor[j]],strat_array[i,previous_neighbor[j]]]
                new_strat_array[i,j] = np.random.choice([agent_strat[k] for k in range(len(agent_p)) if agent_p[k] == np.min(agent_p)])

            # pMin = np.argmin([P_array[i,j],P_array[next_neighbor[i],j],P_array[previous_neighbor[i],j],P_array[i,next_neighbor[j]],P_array[i,previous_neighbor[j]]])
            # if pMin == 0:
            #     new_strat_array[i,j] = strat_array[i,j]
            # if pMin == 1:
            #     new_strat_array[i,j] = strat_array[next_neighbor[i],j]
            # elif pMin == 2:
            #     new_strat_array[i,j] = strat_array[previous_neighbor[i],j]
            # elif pMin == 3:
            #     new_strat_array[i,j] = strat_array[i,next_neighbor[j]]
            # elif pMin == 4:
            #     new_strat_array[i,j] = strat_array[i,previous_neighbor[j]]

    strat_array = new_strat_array.copy()

    # Images for animation
    im = ax1.imshow(new_strat_array.copy(), vmin=0, vmax=N, animated=True)
    ims.append([im])

    if t >= omit_timesteps:
        no_0[t_variance] = np.count_nonzero(strat_array == 0)
        no_1[t_variance] = np.count_nonzero(strat_array == 1)
        no_2[t_variance] = np.count_nonzero(strat_array == 2)
        no_3[t_variance] = np.count_nonzero(strat_array == 3)
        no_4[t_variance] = np.count_nonzero(strat_array == 4)
        no_5[t_variance] = np.count_nonzero(strat_array == 5)
        no_6[t_variance] = np.count_nonzero(strat_array == 6)
        no_7[t_variance] = np.count_nonzero(strat_array == 7)
        t_variance += 1
print('R = {}, S = {}\n'.format(R,S))

# Mean and variance of the population
no_0_mean = np.sum(no_0) / len(no_0)
no_1_mean = np.sum(no_1) / len(no_1)
no_2_mean = np.sum(no_2) / len(no_2)
no_3_mean = np.sum(no_3) / len(no_3)
no_4_mean = np.sum(no_4) / len(no_4)
no_5_mean = np.sum(no_5) / len(no_5)
no_6_mean = np.sum(no_6) / len(no_6)
no_7_mean = np.sum(no_7) / len(no_7)

mean_array = np.round(np.array([no_0_mean, no_1_mean, no_2_mean, no_3_mean, no_4_mean, no_5_mean, no_6_mean, no_7_mean]),2)
[print(f'Mean strat. {i}: ', mean_array[i]) for i in range(len(mean_array))]
print('\n')

no_0_variance = np.sum((no_0-no_0_mean)**2) / len(no_0)
no_1_variance = np.sum((no_1-no_1_mean)**2) / len(no_1)
no_2_variance = np.sum((no_2-no_2_mean)**2) / len(no_2)
no_3_variance = np.sum((no_3-no_3_mean)**2) / len(no_3)
no_4_variance = np.sum((no_4-no_4_mean)**2) / len(no_4)
no_5_variance = np.sum((no_5-no_5_mean)**2) / len(no_5)
no_6_variance = np.sum((no_6-no_6_mean)**2) / len(no_6)
no_7_variance = np.sum((no_7-no_7_mean)**2) / len(no_7)

variance_array = np.round(np.array([no_0_variance, no_1_variance, no_2_variance, no_3_variance, no_4_variance, no_5_variance, no_6_variance, no_7_variance]), 2)
[print(f'Variance strat. {i}: ', variance_array[i]) for i in range(len(variance_array))]
variance_sum =np.sum(variance_array)
print('Variance sum: {}'.format(variance_sum))
print('\n')
```

```python

deviation_array = np.round(variance_array**0.5, 2)
[print(f'Standard devation strat. {i}: ', deviation_array[i]) for i in range(len(deviation_array))]

fig1.colorbar(im, ax=ax1)
ani = animation.ArtistAnimation(fig1, ims, interval=50, blit=True)
writergif = animation.PillowWriter(fps=30)
ani.save('exercise_13.5_R{}_S{}.gif'.format(R,S), writer=writergif)

fig2, ax2 = plt.subplots(figsize=(6,6))
title = 'L = {}, T = {}, R = {}, P = {}, S = {}, $\mu$ = {}'.format(L,T,R,P,S,mu)
ax2.set_title(title)
ax2.imshow(strat_array, vmin=0, vmax=N)
ax2.set_yticks(())
ax2.set_xticks(())
ax2.set_xlabel('$t$ = {}'.format(timesteps))
fig2.colorbar(im, ax=ax2)
plt.savefig('exercise_13.5_R{}_S{}.png'.format(R,S), bbox_inches='tight')

fig3, ax3 = plt.subplots(figsize=(6,6))
t_arange = np.arange(omit_timesteps,timesteps,1)
ax3.plot(t_arange, no_0/(L*L), label='Strat. 0')
ax3.plot(t_arange, no_1/(L*L), label='Strat. 1')
ax3.plot(t_arange, no_2/(L*L), label='Strat. 2')
ax3.plot(t_arange, no_3/(L*L), label='Strat. 3')
ax3.plot(t_arange, no_4/(L*L), label='Strat. 4')
ax3.plot(t_arange, no_5/(L*L), label='Strat. 5')
ax3.plot(t_arange, no_6/(L*L), label='Strat. 6')
ax3.plot(t_arange, no_7/(L*L), label='Strat. 7')
ax3.set_title(title)
ax3.set_xlabel('$t$')
ax3.set_ylabel('Population fraction')
plt.legend(loc="upper left",fontsize=8)
plt.savefig('exercise_13.5_PF_R{}_S{}.png'.format(R,S), bbox_inches='tight')

plt.show()
```