

Homework 4, Game Theory

Erik Norlin, 19970807-9299

December 3, 2022

13.1: a, b, c

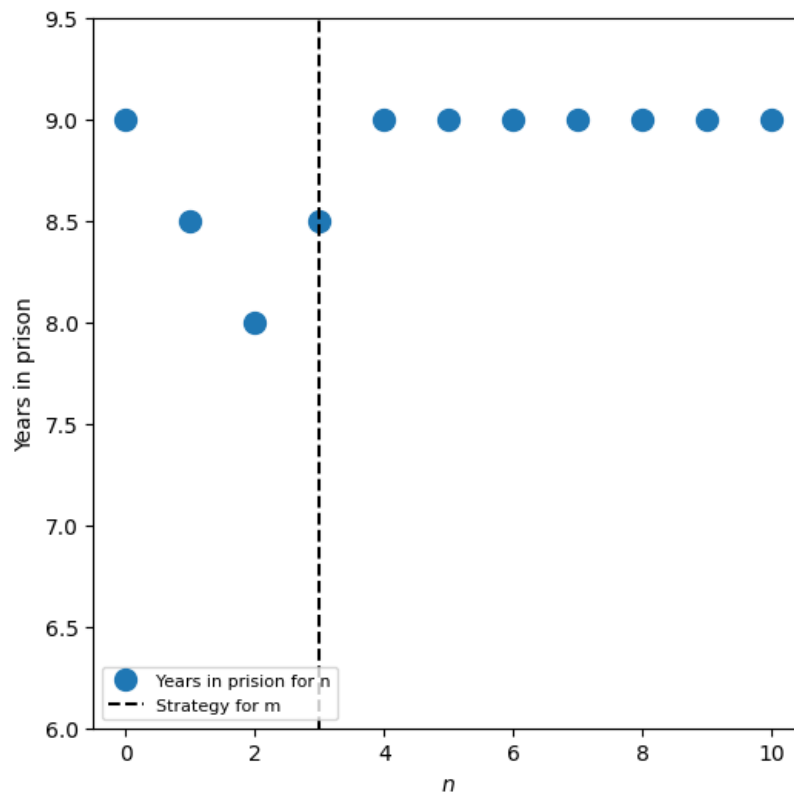


Figure 1: $T = 0$, $R = 0.5$, $P = 1$, $S = 1.5$.

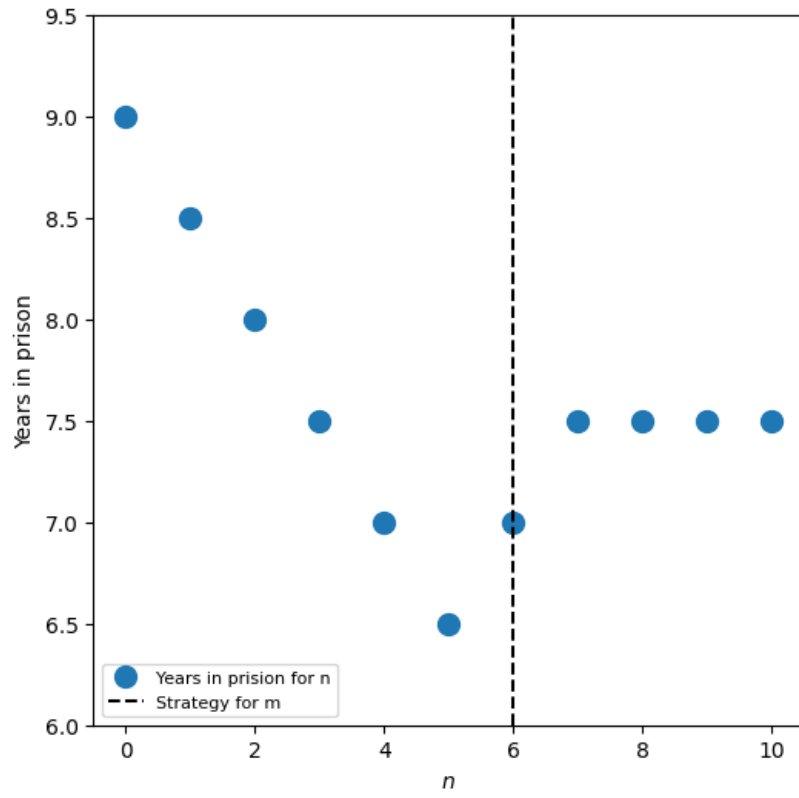


Figure 2: $T = 0$, $R = 0.5$, $P = 1$, $S = 1.5$.

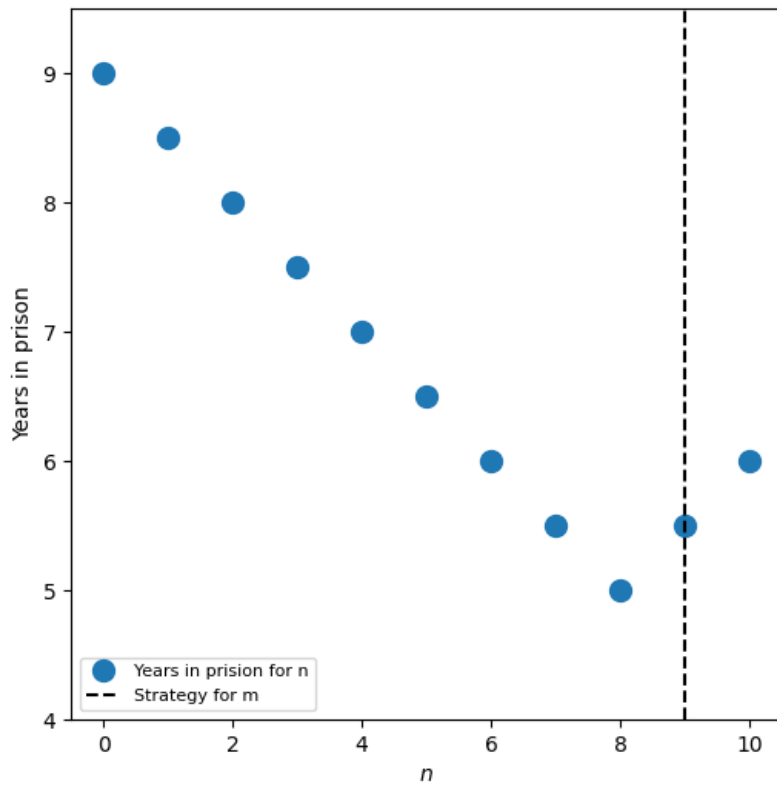


Figure 3: $T = 0$, $R = 0.5$, $P = 1$, $S = 1.5$.

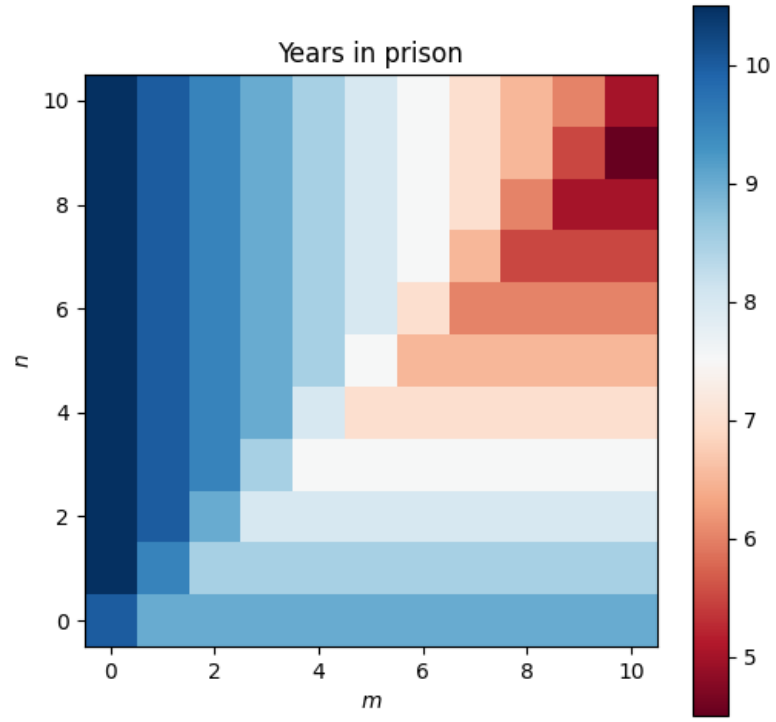


Figure 4: $T = 0$, $R = 0.5$, $P = 1$, $S = 1.5$.

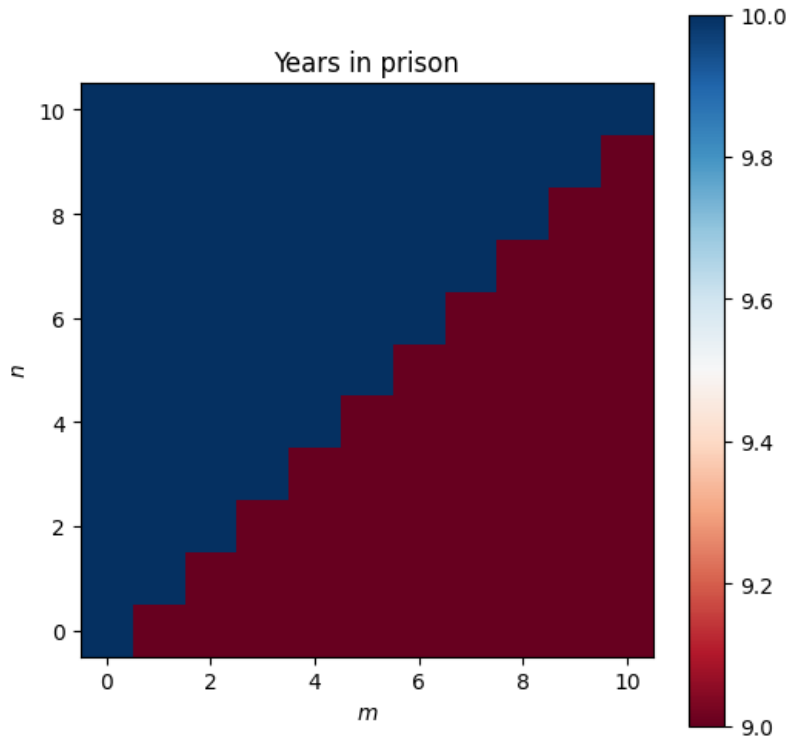


Figure 5: $T = 0$, $R = 1$, $P = 1$, $S = 1$.

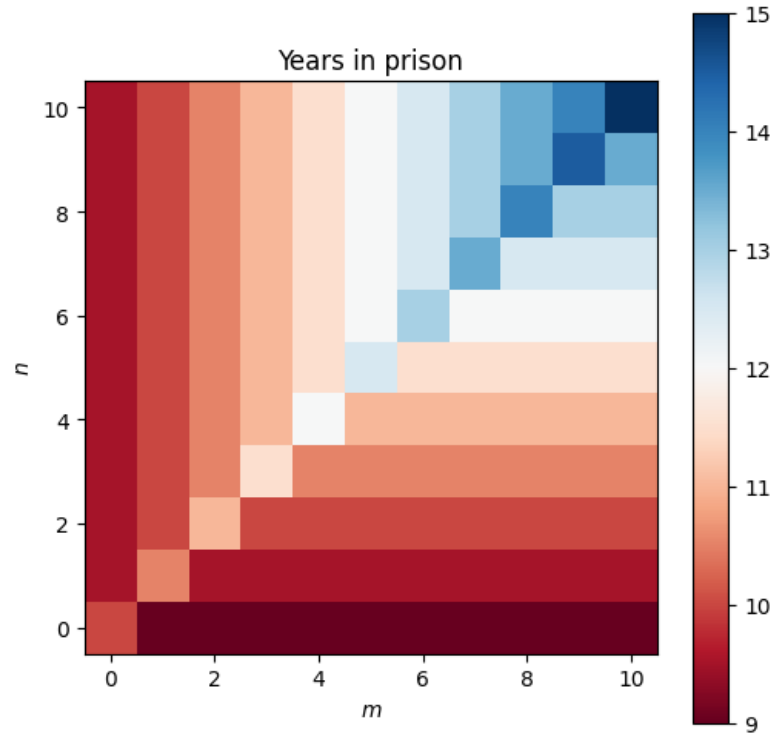


Figure 6: $T = 0$, $R = 1.5$, $P = 1$, $S = 0.5$

13.2: a, b, c

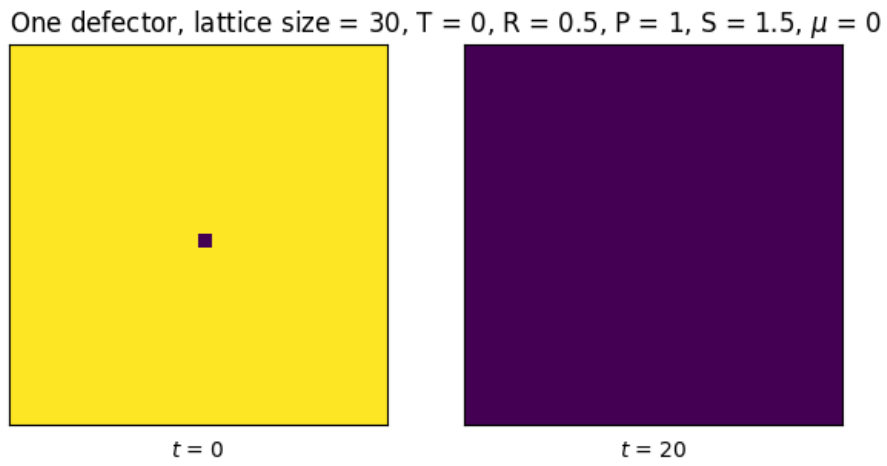


Figure 7: Left: yellow = 7, purple = 0. Right: purple = 7 (colors switched).

One defector, lattice size = 30, $T = 0$, $R = 0.9$, $P = 1$, $S = 1.5$, $\mu = 0$

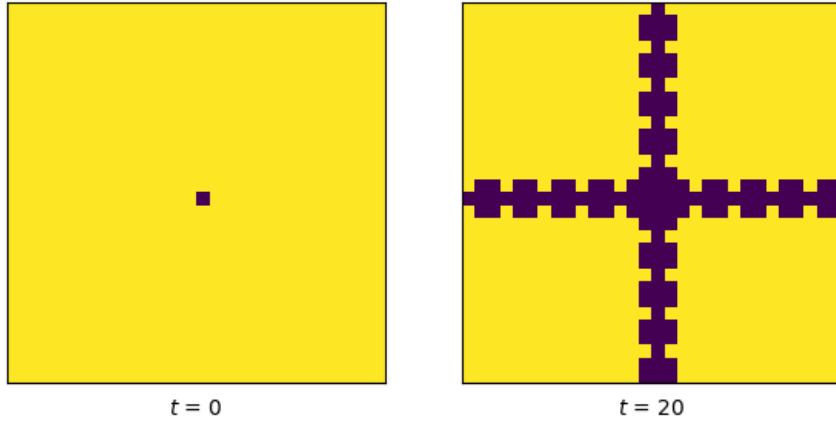


Figure 8: Yellow = 7, purple = 0.

One defector, lattice size = 31, $T = 0$, $R = 1$, $P = 1$, $S = 1.5$, $\mu = 0$

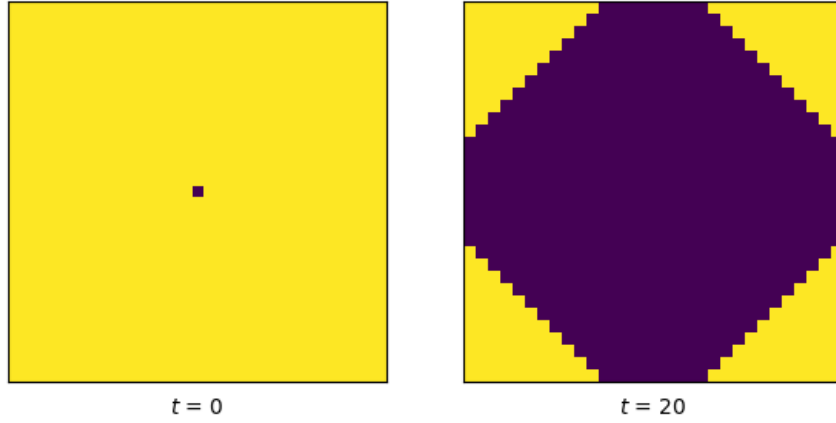


Figure 9: Yellow = 7, purple = 0.

Two defectors, lattice size = 30, $T = 0$, $R = 0.9$, $P = 1$, $S = 1.5$, $\mu = 0$

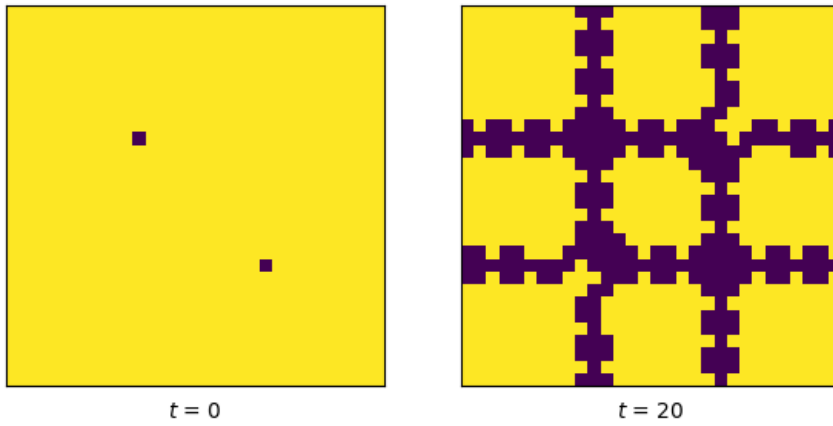


Figure 10: Yellow = 7, purple = 0.

Three defectors, lattice size = 30, $T = 0$, $R = 0.9$, $P = 1$, $S = 1.5$, $\mu = 0$

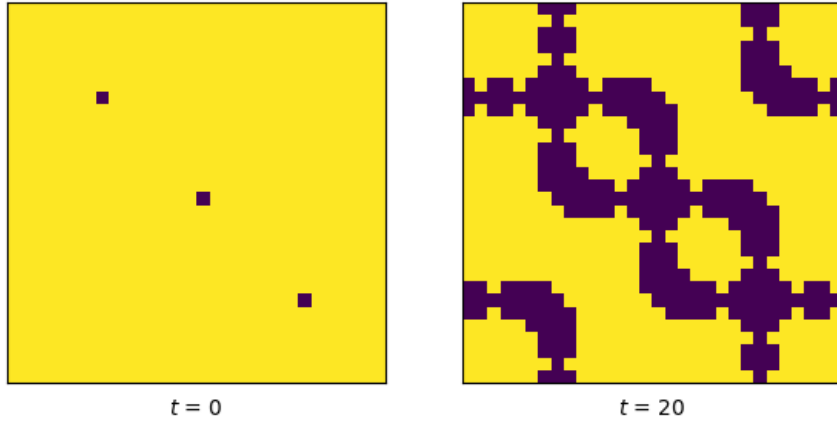


Figure 11: Yellow = 7, purple = 0.

Four defectors, lattice size = 30, $T = 0$, $R = 0.9$, $P = 1$, $S = 1.5$, $\mu = 0$

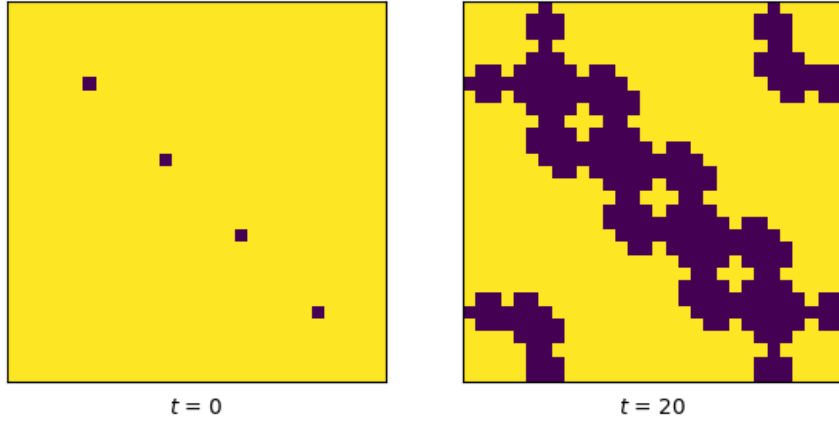


Figure 12: Yellow = 7, purple = 0.

One cooperator, lattice size = 31, $T = 0$, $R = 0.01$, $P = 1$, $S = 1.5$, $\mu = 0$

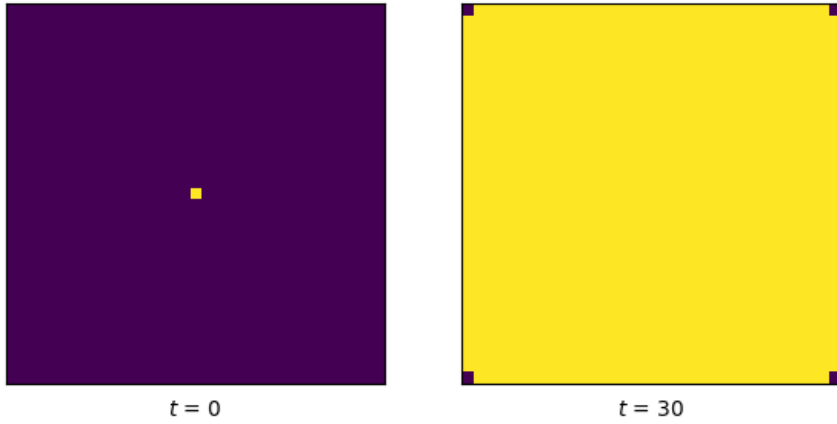


Figure 13: Yellow = 7, purple = 0.

One cooperator, lattice size = 31, $T = 0$, $R = 0.1$, $P = 1$, $S = 1.5$, $\mu = 0$

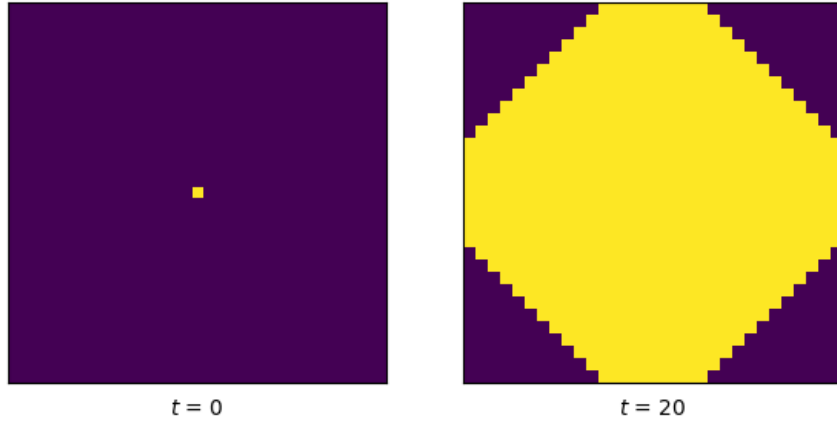


Figure 14: Yellow = 7, purple = 0.

One cooperator, lattice size = 31, $T = 0$, $R = 0.5$, $P = 1$, $S = 1.5$, $\mu = 0$

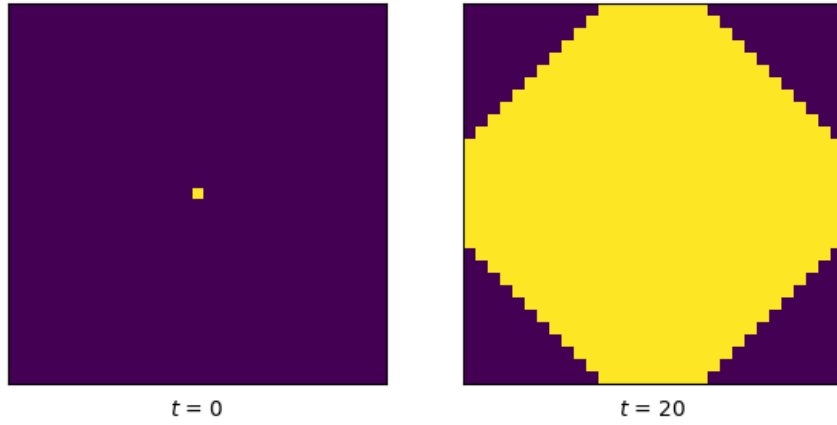


Figure 15: Yellow = 7, purple = 0.

One cooperator, lattice size = 31, $T = 0$, $R = 0.6$, $P = 1$, $S = 1.5$, $\mu = 0$

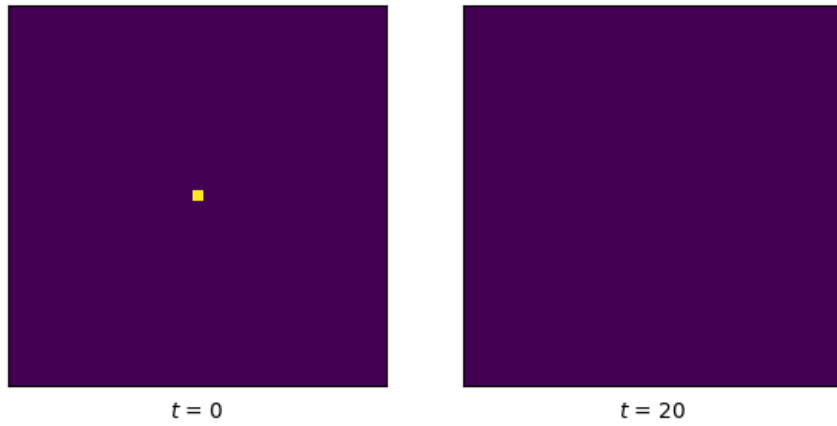


Figure 16: Yellow = 7, purple = 0.

13.3: a, b, c, d, e

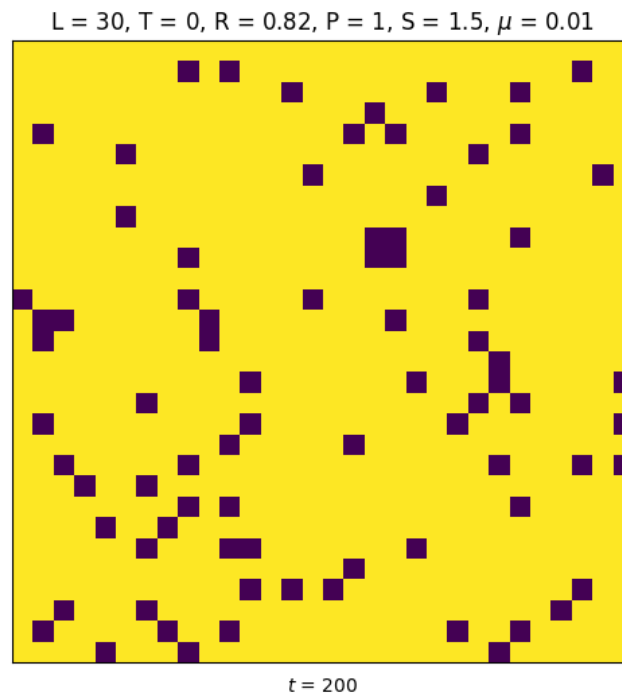


Figure 17: Yellow = 7, purple = 0.

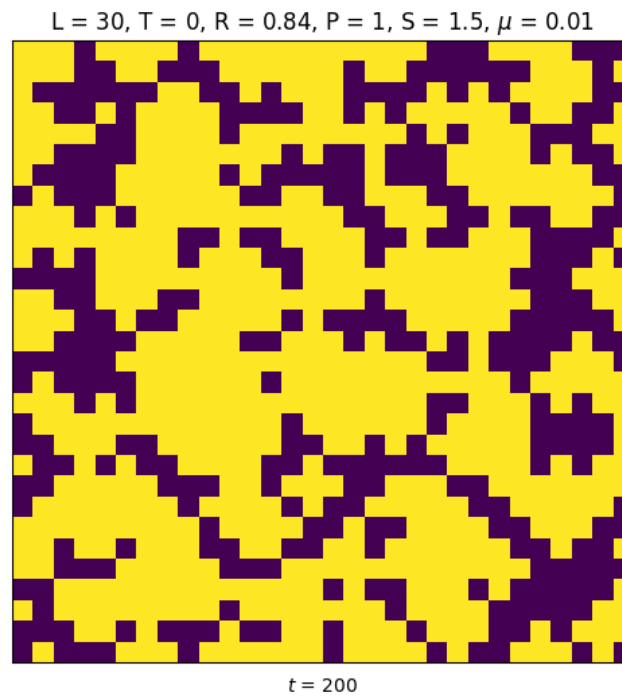


Figure 18: Yellow = 7, purple = 0.

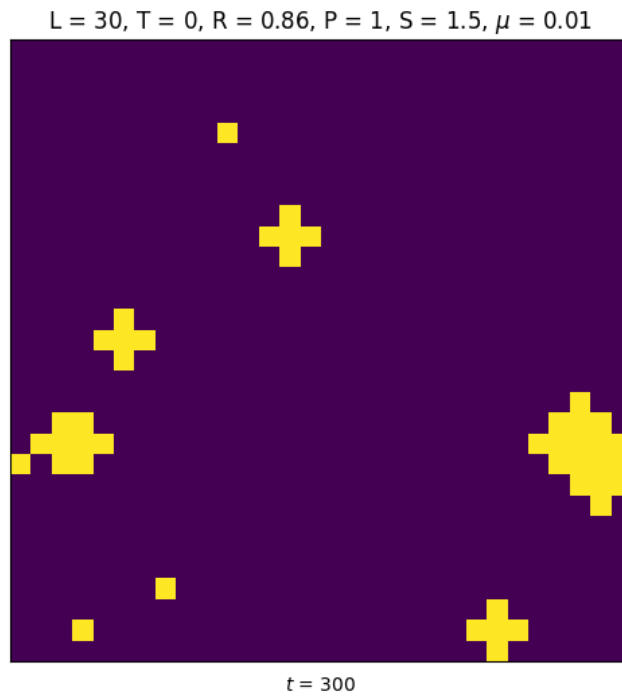


Figure 19: Yellow = 7, purple = 0.

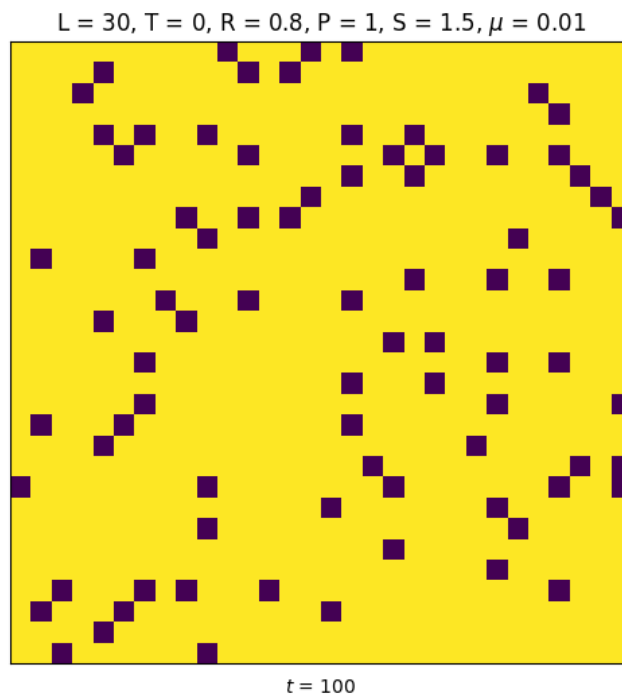


Figure 20: Yellow = 7, purple = 0.

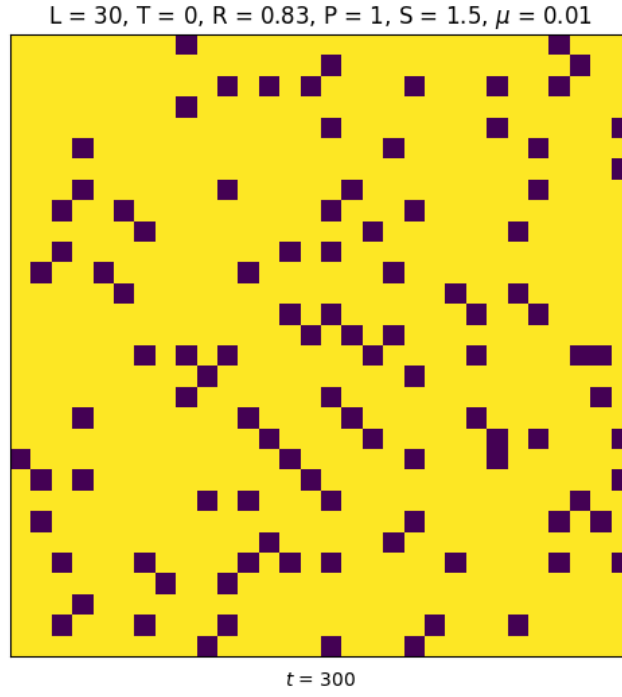


Figure 21: Yellow = 7, purple = 0.

Shift Between $R = 0.83$ and $R = 0.835$

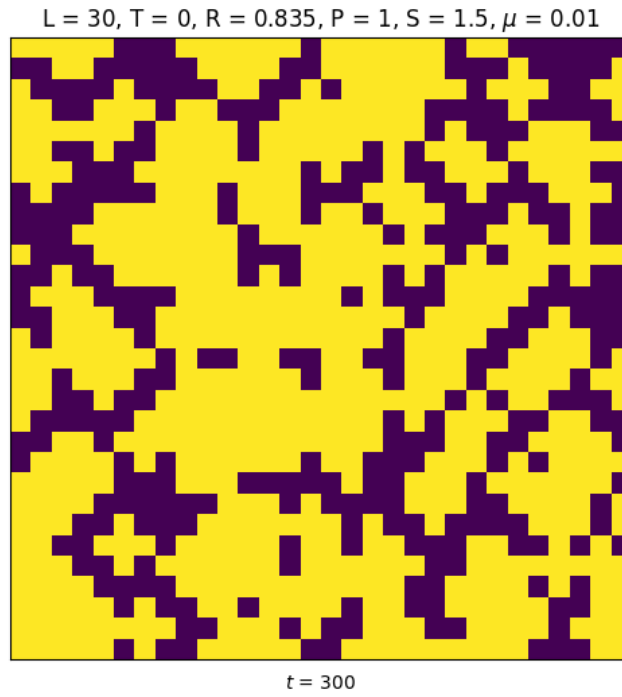


Figure 22: Yellow = 7, purple = 0.

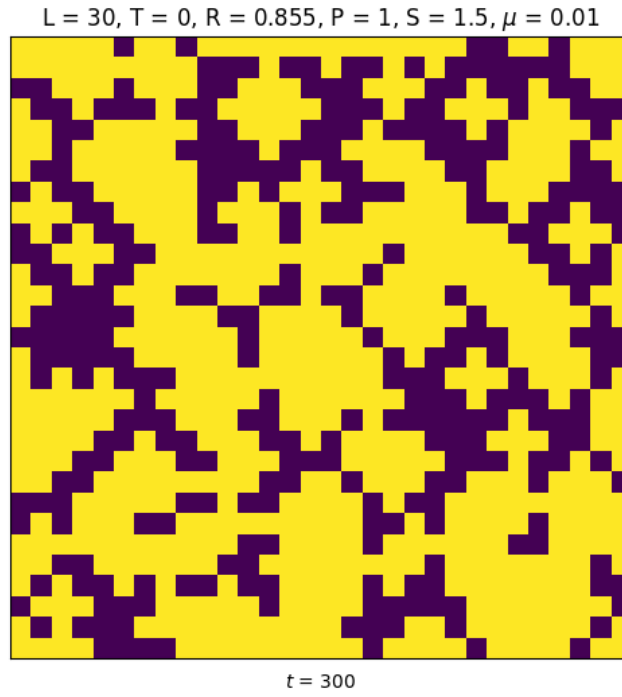


Figure 23: Yellow = 7, purple = 0.

Shift Between $R = 0.855$ and $R = 0.8575$

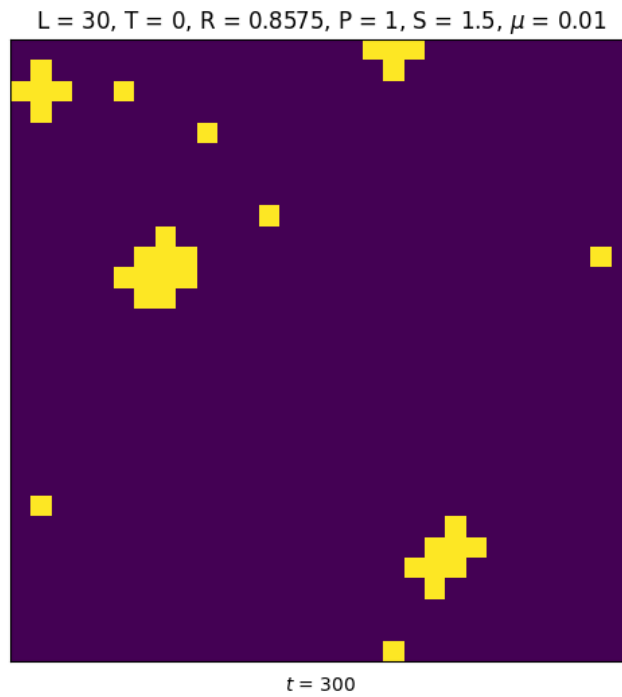


Figure 24: Yellow = 7, purple = 0.

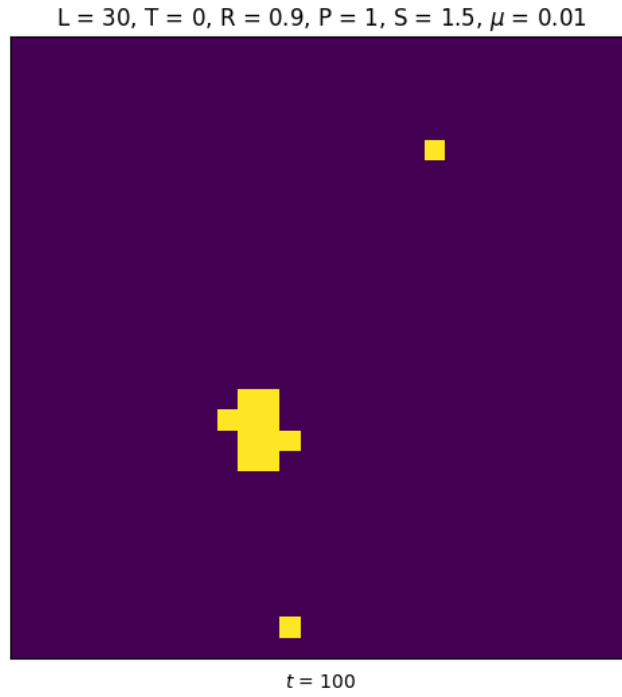


Figure 25: Yellow = 7, purple = 0.

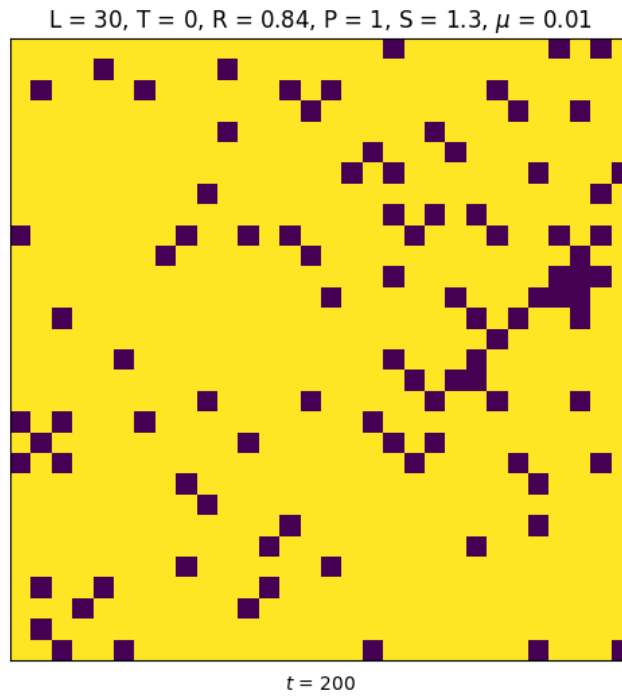


Figure 26: Yellow = 7, purple = 0.

Shift Between $S = 1.3$ and $S = 1.4$.

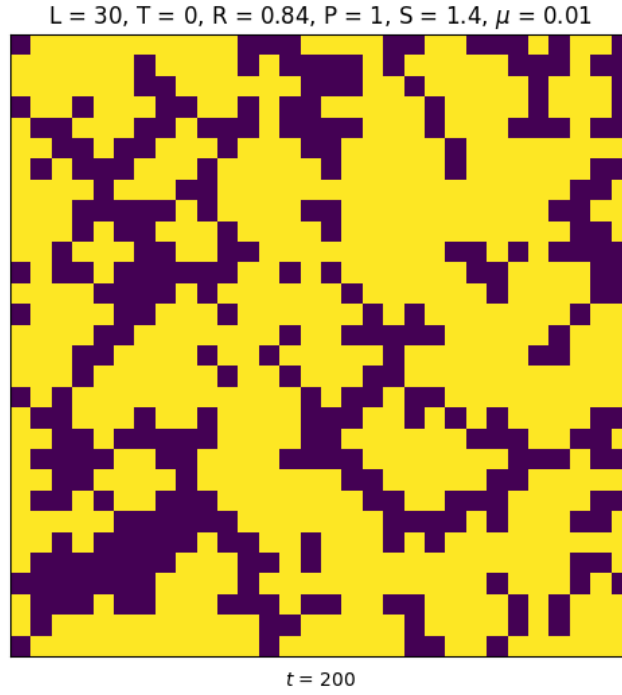


Figure 27: Yellow = 7, purple = 0.

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

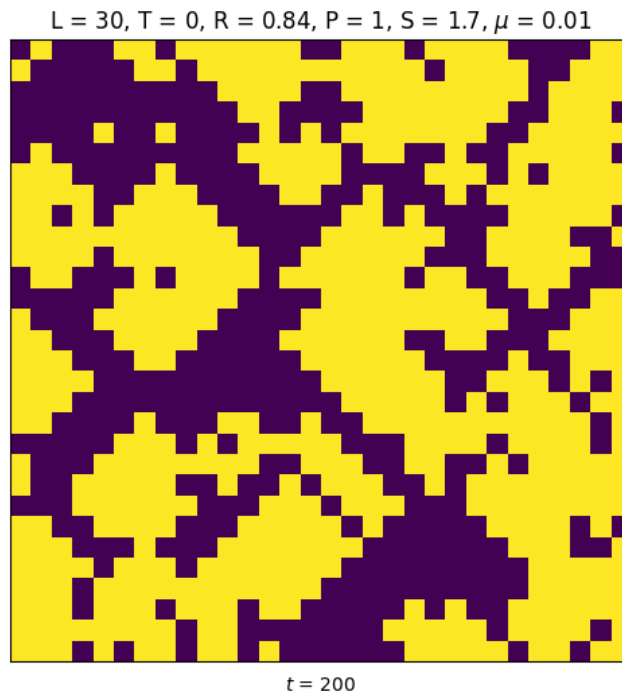


Figure 28: Yellow = 7, purple = 0.

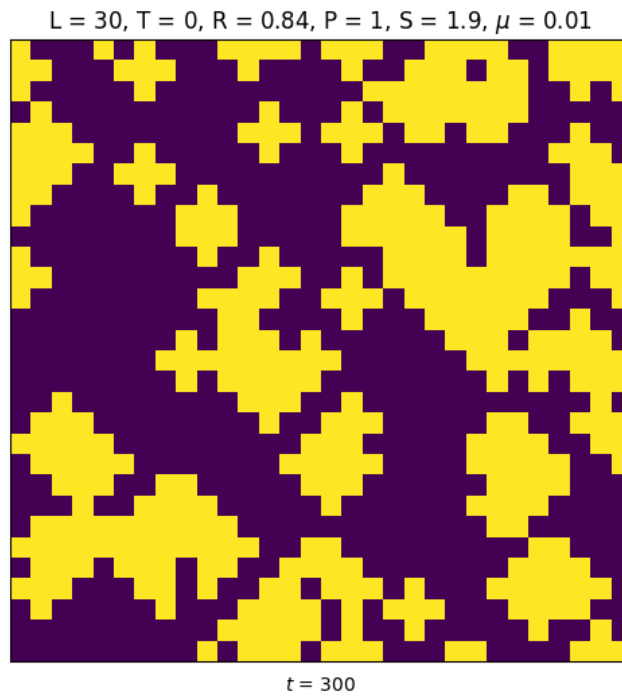


Figure 29: Yellow = 7, purple = 0.

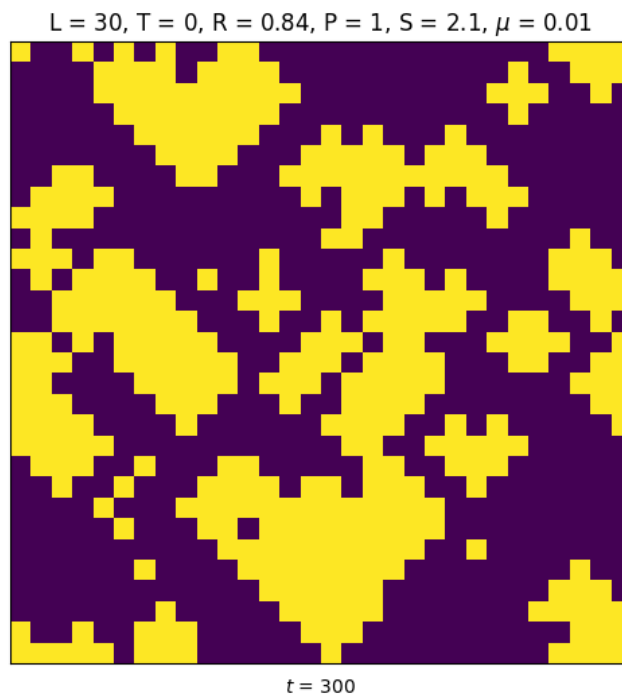


Figure 30: Yellow = 7, purple = 0.

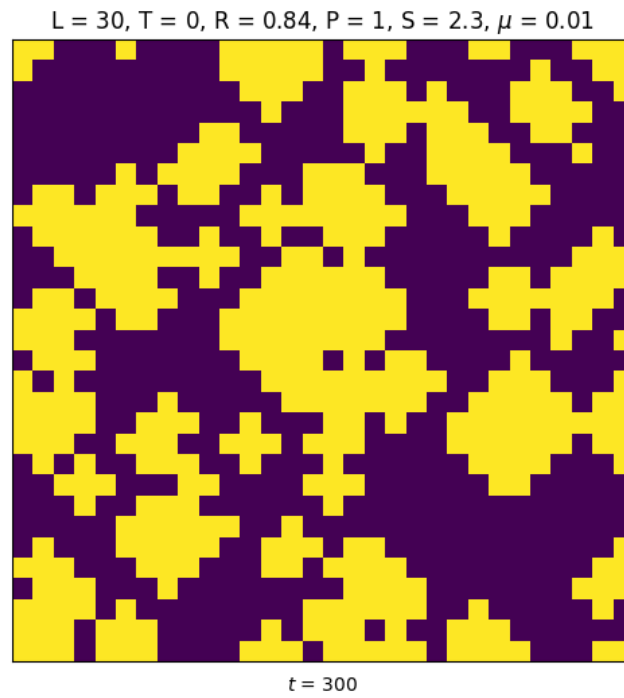


Figure 31: Yellow = 7, purple = 0.

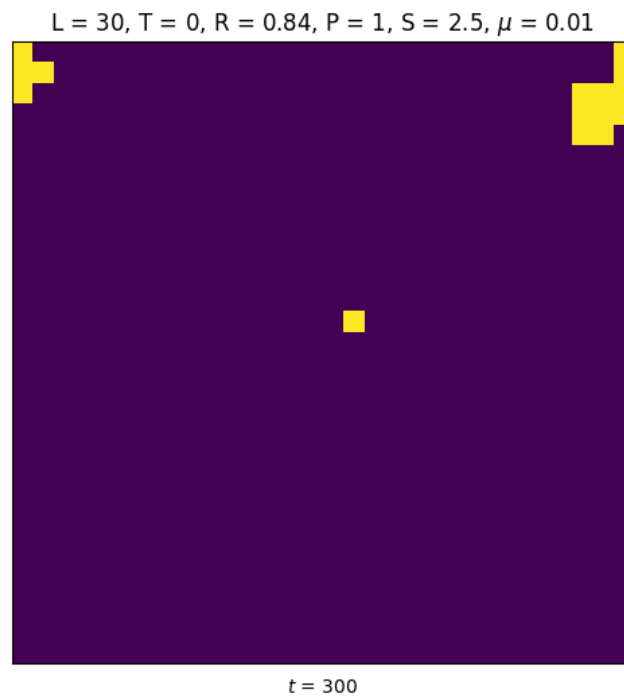


Figure 32: Yellow = 7, purple = 0.

13.4: a, b, c

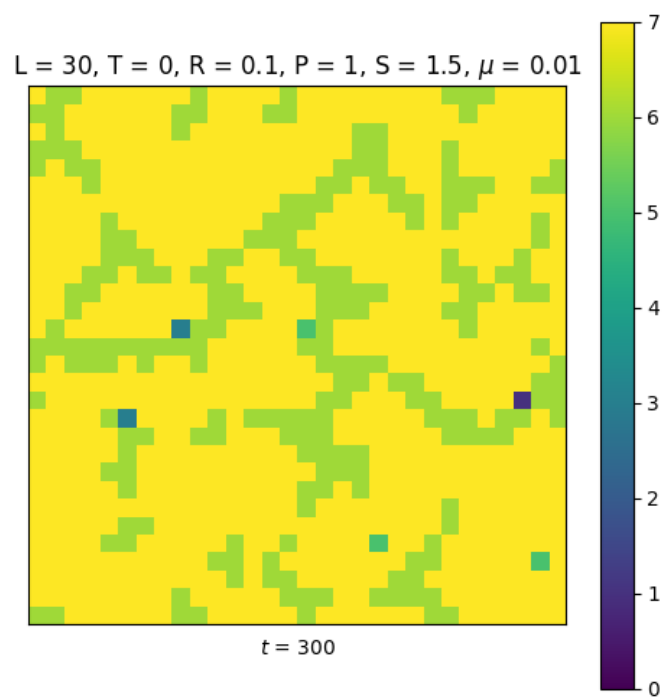


Figure 33

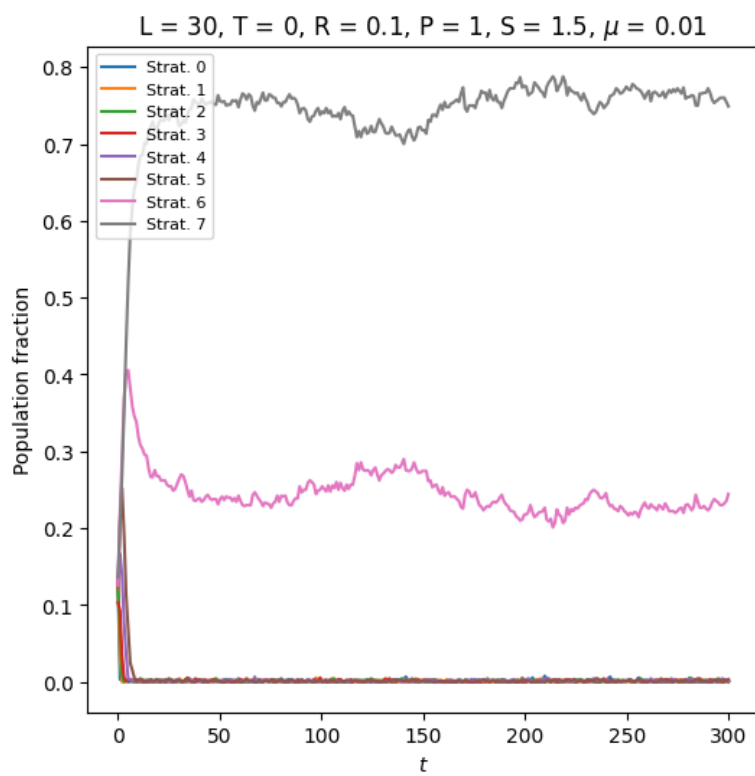


Figure 34

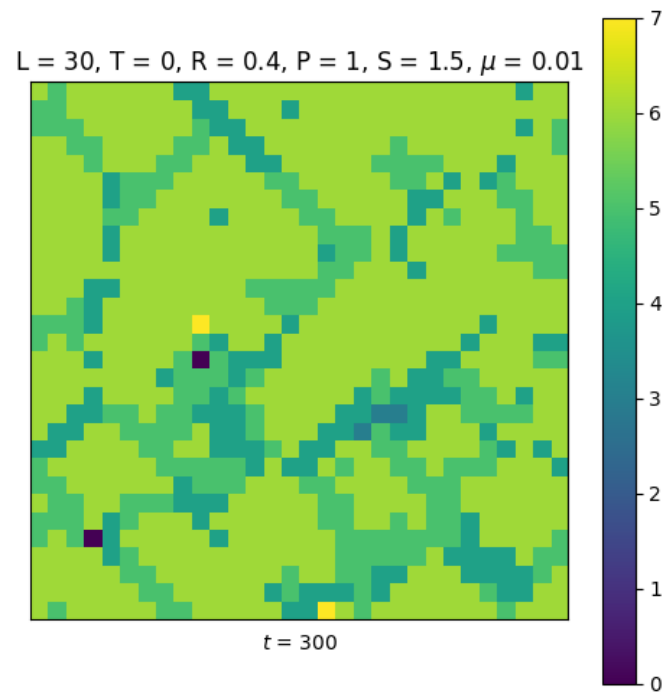


Figure 35

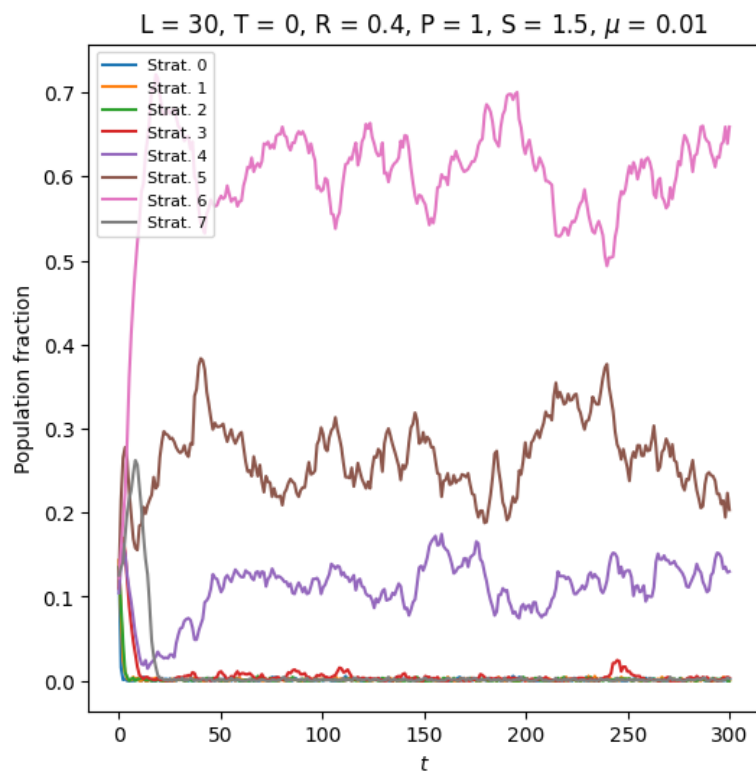


Figure 36

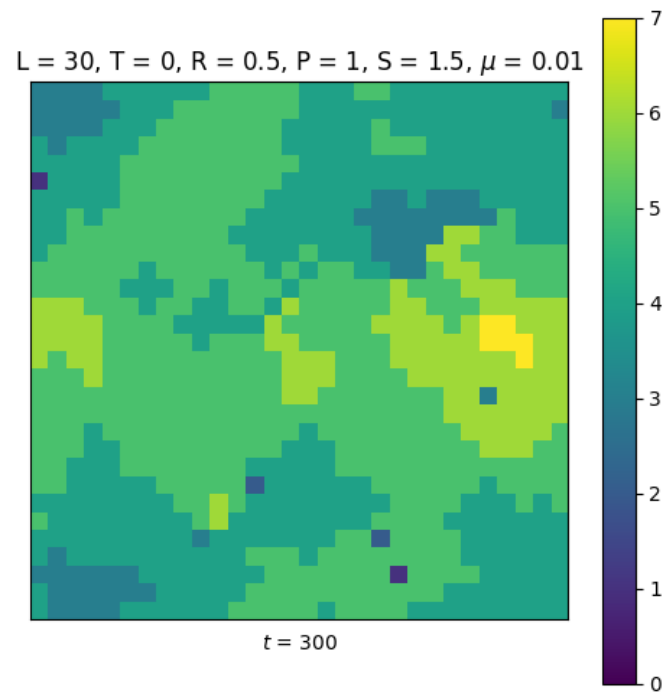


Figure 37

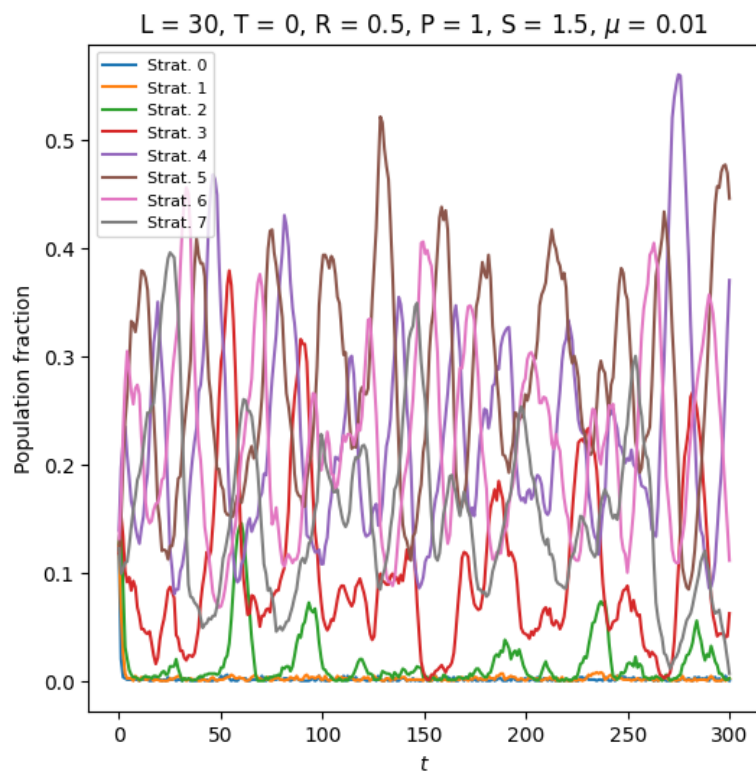


Figure 38

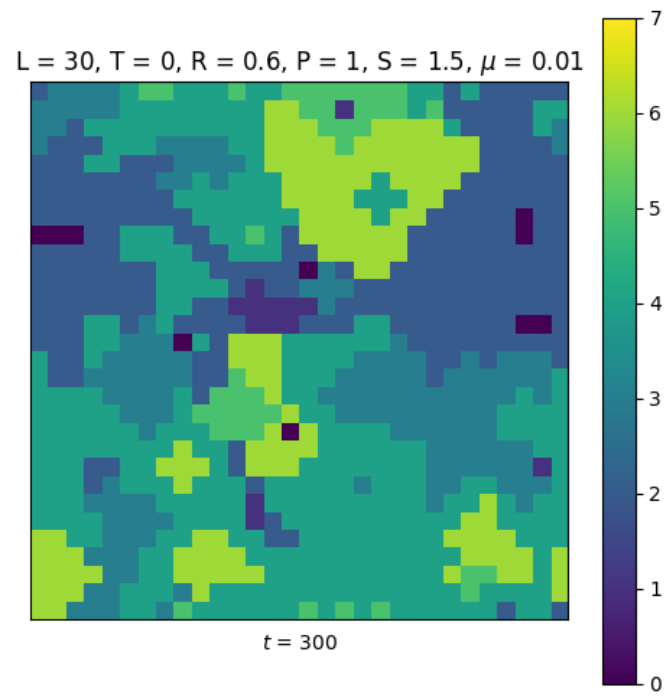


Figure 39

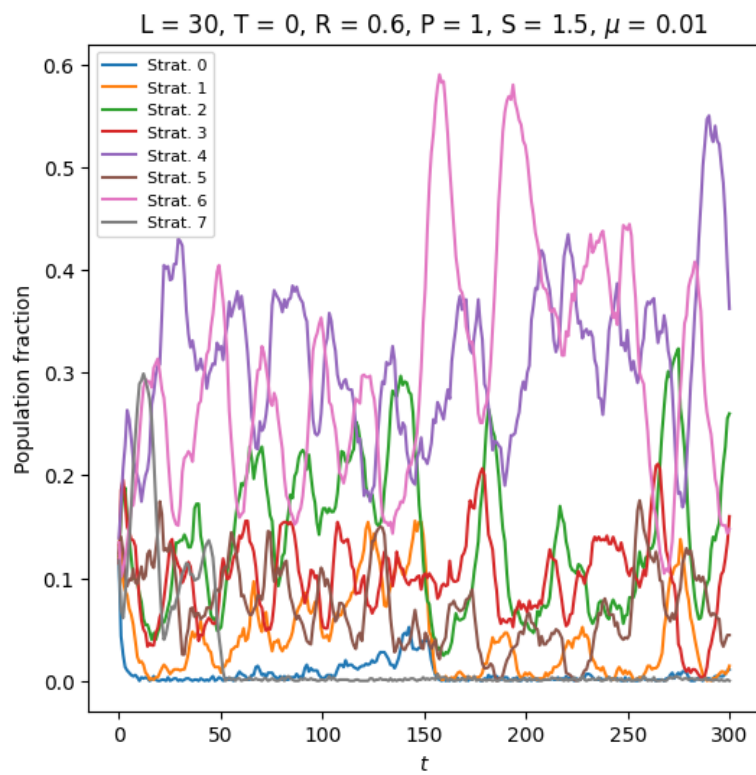


Figure 40

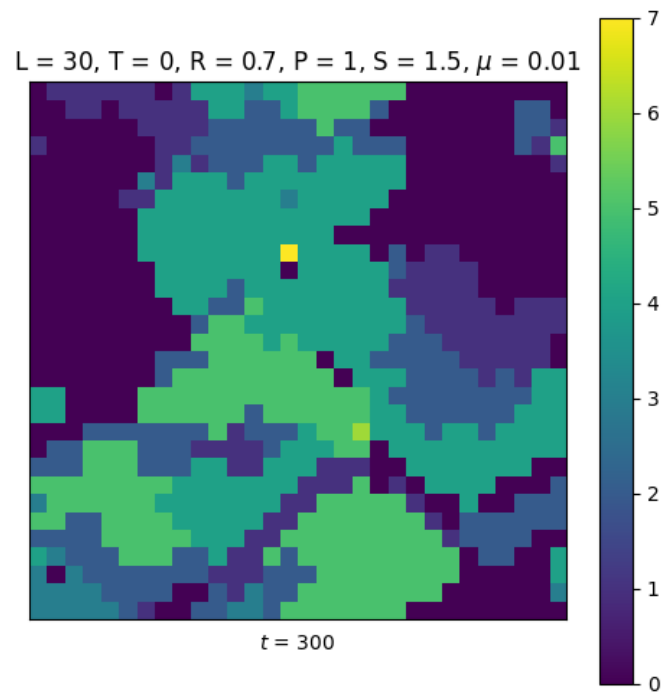


Figure 41

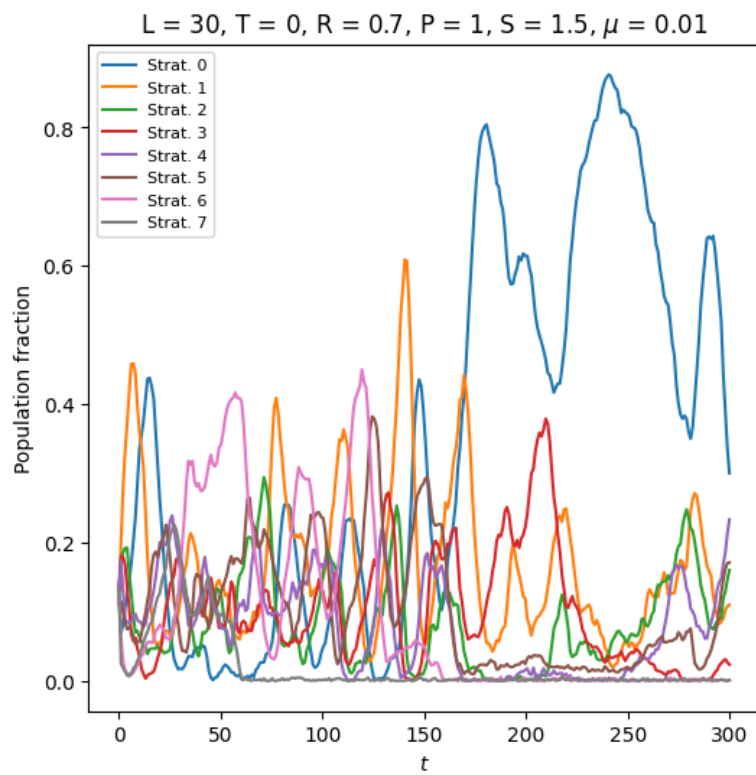


Figure 42

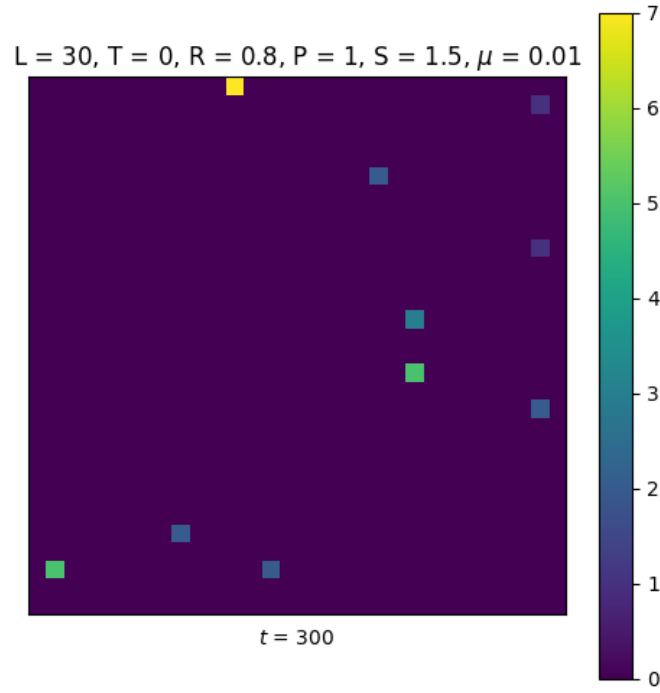


Figure 43

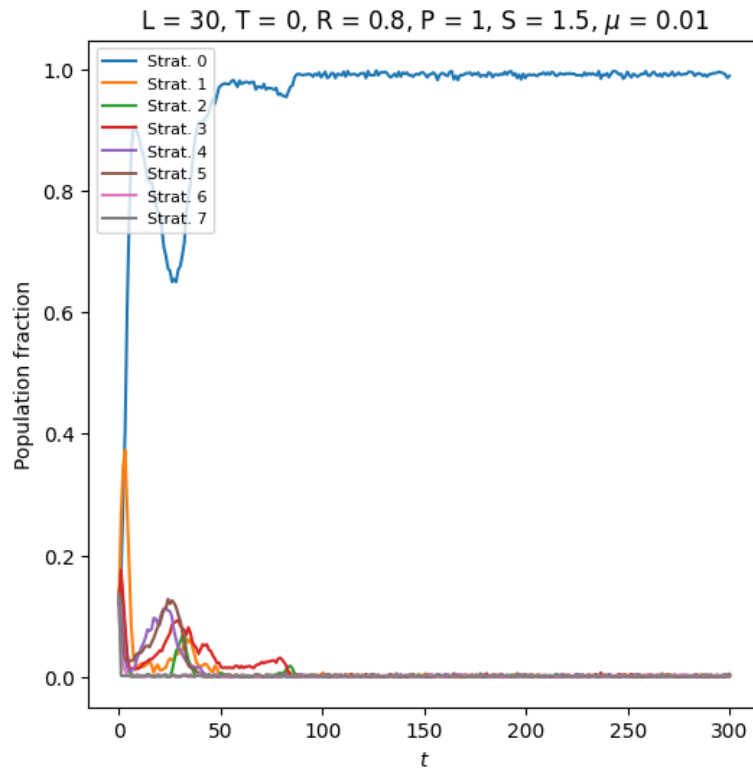


Figure 44

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

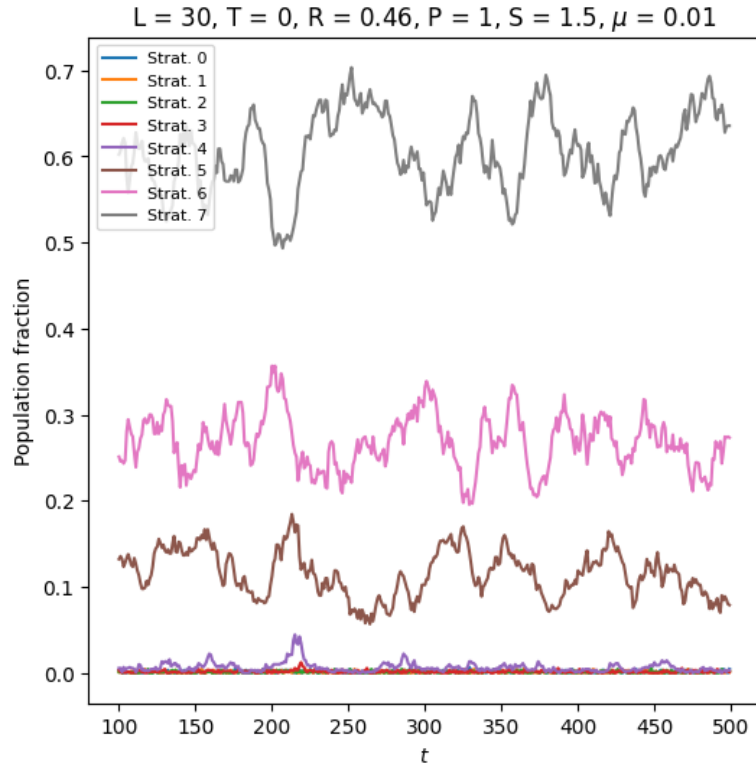


Figure 45: No competition.

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

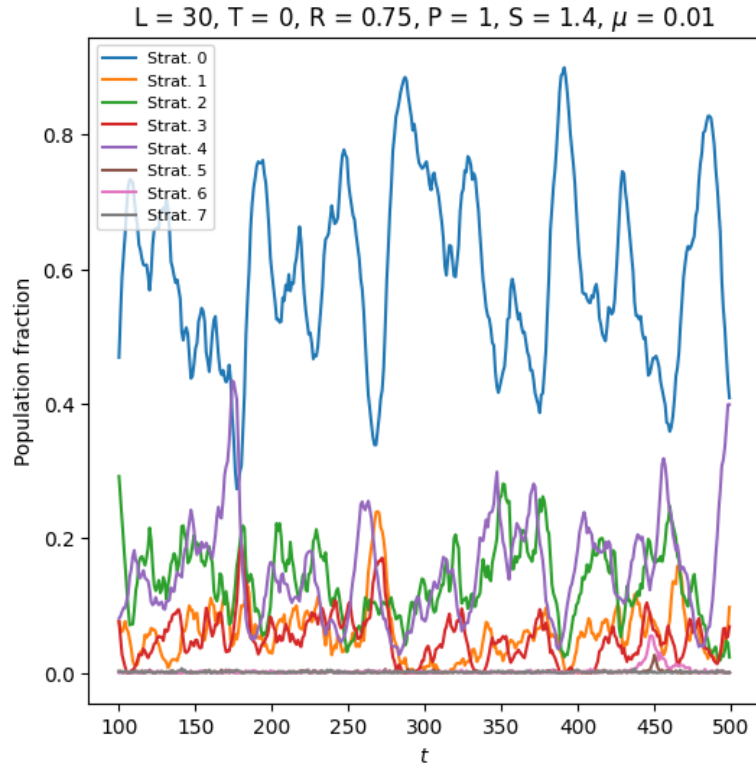


Figure 46: No competition.

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

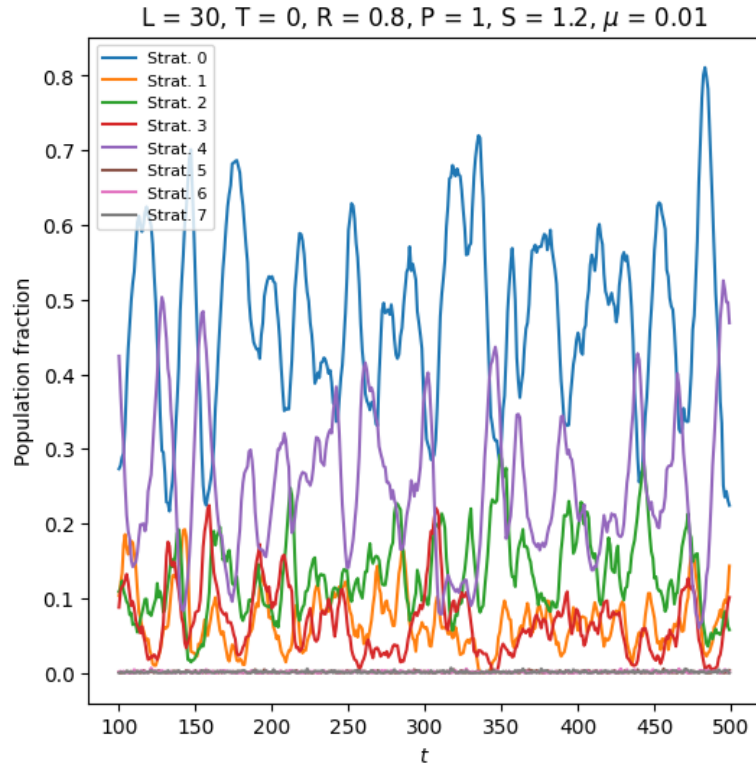


Figure 47: Competition between strategies 0 and 4 starting to become visible.

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

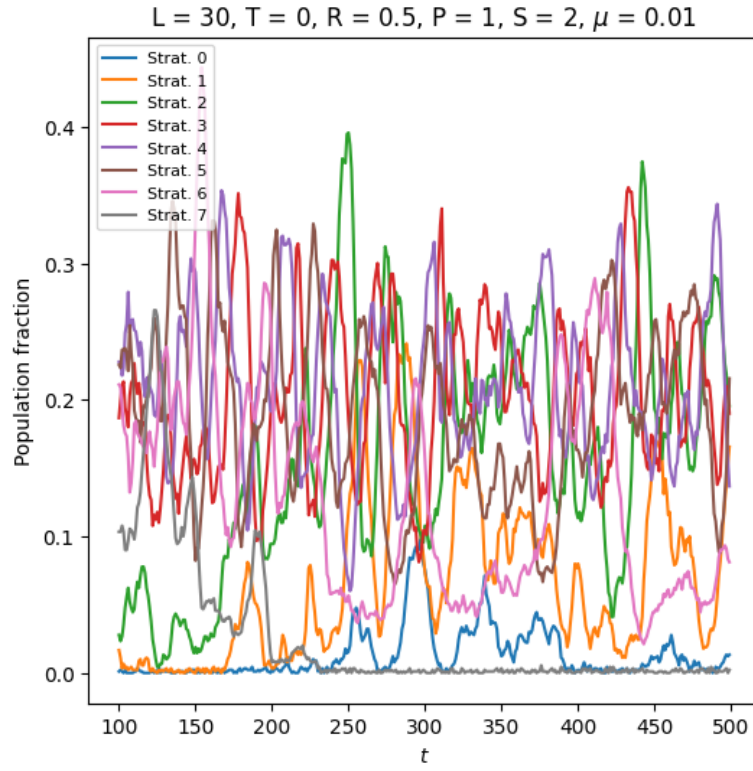


Figure 48: Competition between most strategies are visible.

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

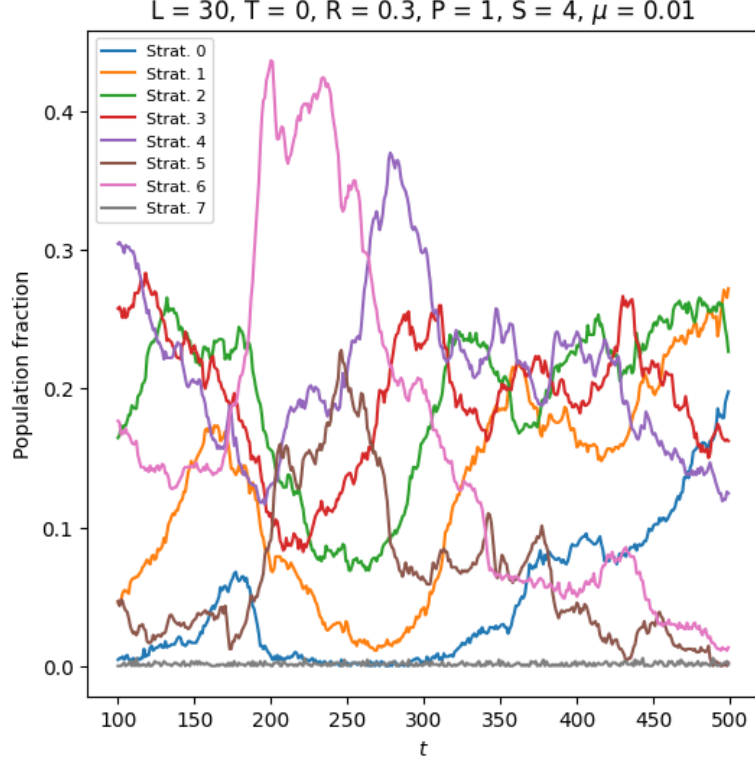


Figure 49: Competition between most strategies are visible here too.

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

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

```
1 # Exercise 13.1a
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import sys
5
6 N = 10
7 T = 0
8 R = 0.5
9 P = 1
10 S = 1.5
11 m_strat = 9
12 n_strats = np.linspace(0,N,N+1)
13 no_years_array = []
14 coop = True
15 defect = False
16
17 for n_strat in n_strats:
18
19     m_previous = coop
20     n_previous = coop
21     no_years = 0
22
23     for round in range(1,N+1):
24
25         if round <= n_strat and m_previous == coop:
26             n = coop
27         else:
28             n = defect
29
30         if round <= m_strat and n_previous == coop:
31             m = coop
32         else:
33             m = defect
34
35         if n == coop and m == coop:
36             no_years += R
37         elif n == coop and m == defect:
38             no_years += S
39         elif n == defect and m == coop:
40             no_years += T
41         elif n == defect and m == defect:
42             no_years += P
43
44         n_previous = n
45         m_previous = m
46
47     no_years_array.append(no_years)
48
49 fig,ax = plt.subplots(figsize=(6,6))
50 ax.plot(n_strats, no_years_array, 'o', markersize=10, label='Years in prison for n')
51 ax.plot([m_strat,m_strat], [4,10], '--', color='black', label='Strategy for m')
52 ax.set_xlabel('$n$')
53 ax.set_ylabel('Years in prison')
54 ax.set_ylim(4,9.5)
55 ax.set_box_aspect(1)
56
57 plt.legend(loc="lower left",fontsize=8)
58 plt.savefig('exercise_13.1a_m=9.png', bbox_inches='tight')
```

```
59 plt.show()
```

```

1 # Exercise 13.1bc
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import sys
5
6 N = 10
7 T = 0
8 R = 0.5
9 P = 1
10 S = 1.5
11 m_strats = np.linspace(0,N,N+1)
12 n_strats = np.linspace(0,N,N+1)
13 no_years_array = np.zeros((N+1, N+1))
14 coop = True
15 defect = False
16
17 for m_strat in m_strats:
18     for n_strat in n_strats:
19
20         m_previous = coop
21         n_previous = coop
22         no_years = 0
23
24         for round in range(1,N+1):
25
26             if round <= n_strat and m_previous == coop:
27                 n = coop
28             else:
29                 n = defect
30
31             if round <= m_strat and n_previous == coop:
32                 m = coop
33             else:
34                 m = defect
35
36             if n == coop and m == coop:
37                 no_years += R
38             elif n == coop and m == defect:
39                 no_years += S
40             elif n == defect and m == coop:
41                 no_years += T
42             elif n == defect and m == defect:
43                 no_years += P
44
45             n_previous = n
46             m_previous = m
47
48         no_years_array[int(n_strat), int(m_strat)] = no_years
49
50 fig,ax = plt.subplots(figsize=(6,6))
51 x,y = np.meshgrid(m_strats, n_strats)
52 years_min, years_max = no_years_array.min(), no_years_array.max()
53 map = ax.pcolormesh(x, y, no_years_array, cmap='RdBu', vmin=years_min,
54                    vmax=years_max)
55 fig.colorbar(map, ax=ax)
56
57 # ax.plot(n_strats, no_years_array, 'o', markersize=10)
58 # ax.plot([m_strat,m_strat], [4,10], '--', color='black', label='Strategy for m')

```

```
59 ax.set_title('Years in prison')
60 ax.set_xlabel('$m$')
61 ax.set_ylabel('$n$')
62 ax.set_box_aspect(1)
63
64 # plt.legend(loc="lower left",fontsize=8)
65 plt.savefig('exercise_13.1c_R' + str(R) + '_S' + str(S) + '.png',
66           bbox_inches='tight')
67 plt.show()
```

```
1 # Exercise 13.2abcd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.animation as animation
5 import sys
6
7 def play_game(self_strat, neighbor_strat):
8
9     coop = True
10    defect = False
11    self_previous = coop
12    neighbor_previous = coop
13    self_punishment = 0
14
15    for round in range(1,N+1):
16
17        if round <= self_strat and neighbor_previous == coop:
18            self_action = coop
19        else:
20            self_action = defect
21
22        if round <= neighbor_strat and self_previous == coop:
23            neighbor_action = coop
24        else:
25            neighbor_action = defect
26
27        if self_action == coop and neighbor_action == coop:
28            self_punishment += R
29        elif self_action == coop and neighbor_action == defect:
30            self_punishment += S
31        elif self_action == defect and neighbor_action == coop:
32            self_punishment += T
33        elif self_action == defect and neighbor_action == defect:
34            self_punishment += P
35
36        self_previous = self_action
37        neighbor_previous = neighbor_action
38
39    return self_punishment
40
41 N = 7
42 T = 0
43 R = 0.99
44 P = 1
45 S = 1.5
46 mu = 0
47 timesteps = 20
48
49 L = 30
50 strat_array = np.ones((L,L))*0
51 strat_array[int(L/2),int(L/2)] = N
52 # strat_array[int(2*L/5),int(2*L/5)] = 0
53 # strat_array[int(-L/5),int(-L/5)] = 0
54 # strat_array[int(-2*L/5),int(-2*L/5)] = 0
55 # strat_array[int(L/5),int(L/5)] = 0
56 new_strat_array = strat_array.copy()
57 strat_array_t0 = strat_array.copy()
58
59 # Animation
```

```

60 fig1, ax = plt.subplots()
61 ims = []
62 im = ax.imshow(strat_array_t0)
63 ims.append([im])
64
65 for t in range(1,timesteps):
66
67     P_array = np.zeros_like(strat_array)
68     next_neighbor = np.roll(np.arange(L),-1)
69     previous_neighbor = np.roll(np.arange(L),1)
70
71     # Accumulate punishment for every agent i,j
72     for i in range(L):
73         for j in range(L):
74
75             # Punishment for Von Neumann neighbors and self
76             pSelf = play_game(strat_array[i,j], strat_array[i,j])
77             pUp = play_game(strat_array[i,j], strat_array[previous_neighbor[i],j])
78             pLeft = play_game(strat_array[i,j], strat_array[i,previous_neighbor[j]])
79             pDown = play_game(strat_array[i,j], strat_array[next_neighbor[i],j])
80             pRight = play_game(strat_array[i,j], strat_array[i,next_neighbor[j]])
81             P_array[i,j] = pUp + pLeft + pDown + pRight
82
83     # Compute new strategies for every agent
84     for i in range(L):
85         for j in range(L):
86
87             r = np.random.uniform()
88             if r < mu:
89                 new_strat_array[i,j] = np.random.choice([0,N])
90             else:
91                 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]]]
92                 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]]]
93                 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)])
94
95                 # 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]]])
96                 # if pMin == 0:
97                 #     new_strat_array[i,j] = strat_array[i,j]
98                 # if pMin == 1:
99                 #     new_strat_array[i,j] = strat_array[next_neighbor[i],j]
100                 # elif pMin == 2:
101                 #     new_strat_array[i,j] = strat_array[previous_neighbor[i],j]
102                 # elif pMin == 3:
103                 #     new_strat_array[i,j] = strat_array[i,next_neighbor[j]]
104                 # elif pMin == 4:
105                 #     new_strat_array[i,j] = strat_array[i,previous_neighbor[j]]
106
107     strat_array = new_strat_array.copy()
108     # Images for animation
109     im = ax.imshow(new_strat_array.copy(), animated=True)
110     ims.append([im])
111
112 fig, axs = plt.subplots(1,2,figsize=(7,7))

```



```
113 title = 'One defector, lattice size = {}, T = {}, R = {}, P = {}, S = {},  $\mu$  =  
    {}'.format(L,T,R,P,S,mu)  
114  
115 axs[0].set_title(title, loc='left')  
116 axs[0].imshow(strat_array_t0)  
117 axs[0].set_yticks(())  
118 axs[0].set_xticks(())  
119 axs[0].set_xlabel('$t$ = 0')  
120  
121 axs[1].imshow(strat_array)  
122 axs[1].set_yticks(())  
123 axs[1].set_xticks(())  
124 axs[1].set_xlabel('$t$ = {}'.format(timesteps))  
125  
126 ani = animation.ArtistAnimation(fig1, ims, interval=5, blit=True)  
127 writergif = animation.PillowWriter(fps=30)  
128 # ani.save('exercise_13.2_1def_R{}.gif'.format(R), writer=writergif)  
129  
130 # plt.savefig('exercise_13.2_1def_R{}.png'.format(R), bbox_inches='tight')  
131 plt.show()  
132  
133  
134  
135  
136  
137
```

```
1 # Exercise 13.3abcde
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.animation as animation
5 import sys
6
7 def play_game(self_strat, neighbor_strat):
8
9     coop = True
10    defect = False
11    self_previous = coop
12    neighbor_previous = coop
13    self_punishment = 0
14
15    for round in range(1,N+1):
16
17        if round <= self_strat and neighbor_previous == coop:
18            self_action = coop
19        else:
20            self_action = defect
21
22        if round <= neighbor_strat and self_previous == coop:
23            neighbor_action = coop
24        else:
25            neighbor_action = defect
26
27        if self_action == coop and neighbor_action == coop:
28            self_punishment += R
29        elif self_action == coop and neighbor_action == defect:
30            self_punishment += S
31        elif self_action == defect and neighbor_action == coop:
32            self_punishment += T
33        elif self_action == defect and neighbor_action == defect:
34            self_punishment += P
35
36        self_previous = self_action
37        neighbor_previous = neighbor_action
38
39    return self_punishment
40
41 N = 7
42 T = 0
43 R = 0.84
44 P = 1
45 S = 2.3
46 mu = 0.01
47 timesteps = 300
48
49 L = 30
50 strat_array = np.ones((L,L))*N
51 # strat_array[int(L/2),int(L/2)] = 0
52 # strat_array[int(L/5),int(L/5)] = 0
53 # strat_array[int(-L/5),int(-L/5)] = 0
54 # strat_array[int(-2*L/5),int(-2*L/5)] = 0
55 # strat_array[int(L/2),int(L/2)] = 7
56 new_strat_array = strat_array.copy()
57 strat_array_t0 = strat_array.copy()
58
59 # Animation
```

```

60 fig1, ax1 = plt.subplots()
61 ims = []
62 im = ax1.imshow(strat_array_t0)
63 ims.append([im])
64
65 for t in range(1,timesteps):
66
67     P_array = np.zeros_like(strat_array)
68     next_neighbor = np.roll(np.arange(L),-1)
69     previous_neighbor = np.roll(np.arange(L),1)
70
71     # Accumulate punishment for every agent i,j
72     for i in range(L):
73         for j in range(L):
74
75             # Punishment for Von Neumann neighbors and self
76             pSelf = play_game(strat_array[i,j], strat_array[i,j])
77             pUp = play_game(strat_array[i,j], strat_array[previous_neighbor[i],j])
78             pLeft = play_game(strat_array[i,j], strat_array[i,previous_neighbor[j]])
79             pDown = play_game(strat_array[i,j], strat_array[next_neighbor[i],j])
80             pRight = play_game(strat_array[i,j], strat_array[i,next_neighbor[j]])
81             P_array[i,j] = pUp + pLeft + pDown + pRight
82
83     # Compute new strategies for every agent
84     for i in range(L):
85         for j in range(L):
86
87             r = np.random.uniform()
88             if r < mu:
89                 new_strat_array[i,j] = np.random.choice([0,N])
90             else:
91                 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]]]
92                 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]]]
93                 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)])
94
95                 # 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]]])
96                 # if pMin == 0:
97                 #     new_strat_array[i,j] = strat_array[i,j]
98                 # if pMin == 1:
99                 #     new_strat_array[i,j] = strat_array[next_neighbor[i],j]
100                 # elif pMin == 2:
101                 #     new_strat_array[i,j] = strat_array[previous_neighbor[i],j]
102                 # elif pMin == 3:
103                 #     new_strat_array[i,j] = strat_array[i,next_neighbor[j]]
104                 # elif pMin == 4:
105                 #     new_strat_array[i,j] = strat_array[i,previous_neighbor[j]]
106
107     strat_array = new_strat_array.copy()
108     # Images for animation
109     im = ax1.imshow(new_strat_array.copy(), animated=True)
110     ims.append([im])
111
112 fig2, ax2 = plt.subplots(figsize=(6,6))

```

```
113 title = 'L = {}, T = {}, R = {}, P = {}, S = {},  $\mu$  = {}'.format(L,T,R,P,S,mu)
114
115 ax2.set_title(title)
116 ax2.imshow(strat_array_t0)
117 ax2.set_yticks(())
118 ax2.set_xticks(())
119 ax2.set_xlabel('$t$ = 0')
120
121 ax2.imshow(strat_array)
122 ax2.set_yticks(())
123 ax2.set_xticks(())
124 ax2.set_xlabel('$t$ = {}'.format(timesteps))
125
126 ani = animation.ArtistAnimation(fig1, ims, interval=5, blit=True)
127 writergif = animation.PillowWriter(fps=30)
128 ani.save('exercise_13.3_S{}.gif'.format(S), writer=writergif)
129
130 plt.savefig('exercise_13.3_S{}.png'.format(S), bbox_inches='tight')
131 plt.show()
132
133
134
135
136
137
```

```
1 # Exercise 13.4abcd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.animation as animation
5 import sys
6
7 def play_game(self_strat, neighbor_strat):
8
9     coop = True
10    defect = False
11    self_previous = coop
12    neighbor_previous = coop
13    self_punishment = 0
14
15    for round in range(1,N+1):
16
17        if round <= self_strat and neighbor_previous == coop:
18            self_action = coop
19        else:
20            self_action = defect
21
22        if round <= neighbor_strat and self_previous == coop:
23            neighbor_action = coop
24        else:
25            neighbor_action = defect
26
27        if self_action == coop and neighbor_action == coop:
28            self_punishment += R
29        elif self_action == coop and neighbor_action == defect:
30            self_punishment += S
31        elif self_action == defect and neighbor_action == coop:
32            self_punishment += T
33        elif self_action == defect and neighbor_action == defect:
34            self_punishment += P
35
36        self_previous = self_action
37        neighbor_previous = neighbor_action
38
39    return self_punishment
40
41 N = 7
42 T = 0
43 R = 0.65
44 P = 1
45 S = 1.5
46 mu = 0.01
47 timesteps = 300
48
49 L = 30
50 strat_array = np.random.randint(0,N+1,size=(L,L))
51 new_strat_array = strat_array.copy()
52 strat_array_t0 = strat_array.copy()
53
54 # Distribution fraction
55 no_0 = np.zeros(timesteps)
56 no_1 = no_0.copy()
57 no_2 = no_0.copy()
58 no_3 = no_0.copy()
59 no_4 = no_0.copy()
```

```

60 no_5 = no_0.copy()
61 no_6 = no_0.copy()
62 no_7 = no_0.copy()
63
64 no_0[0] = np.count_nonzero(strat_array == 0)
65 no_1[0] = np.count_nonzero(strat_array == 1)
66 no_2[0] = np.count_nonzero(strat_array == 2)
67 no_3[0] = np.count_nonzero(strat_array == 3)
68 no_4[0] = np.count_nonzero(strat_array == 4)
69 no_5[0] = np.count_nonzero(strat_array == 5)
70 no_6[0] = np.count_nonzero(strat_array == 6)
71 no_7[0] = np.count_nonzero(strat_array == 7)
72
73
74 # Animation
75 fig1, ax1 = plt.subplots()
76 ims = []
77 im = ax1.imshow(strat_array_t0, vmin=0, vmax=N, animated=True)
78 ims.append([im])
79
80 for t in range(1,timesteps):
81
82     P_array = np.zeros_like(strat_array)
83     next_neighbor = np.roll(np.arange(L),-1)
84     previous_neighbor = np.roll(np.arange(L),1)
85
86     # Accumulate punishment for every agent i,j
87     for i in range(L):
88         for j in range(L):
89
90             # Punishment for Von Neumann neighbors and self
91             pSelf = play_game(strat_array[i,j], strat_array[i,j])
92             pUp = play_game(strat_array[i,j], strat_array[previous_neighbor[i],j])
93             pLeft = play_game(strat_array[i,j], strat_array[i,previous_neighbor[j]])
94             pDown = play_game(strat_array[i,j], strat_array[next_neighbor[i],j])
95             pRight = play_game(strat_array[i,j], strat_array[i,next_neighbor[j]])
96             P_array[i,j] = pUp + pLeft + pDown + pRight
97
98     # Compute new strategies for every agent
99     for i in range(L):
100         for j in range(L):
101
102             r = np.random.uniform()
103             if r < mu:
104                 new_strat_array[i,j] = np.random.randint(0,N+1)
105             else:
106                 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]]]
107                 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]]]
108                 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)])
109
110                 # 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]]])
111                 # if pMin == 0:
112                 #     new_strat_array[i,j] = strat_array[i,j]

```

```

113         # if pMin == 1:
114         #     new_strat_array[i,j] = strat_array[next_neighbor[i],j]
115         # elif pMin == 2:
116         #     new_strat_array[i,j] = strat_array[previous_neighbor[i],j]
117         # elif pMin == 3:
118         #     new_strat_array[i,j] = strat_array[i,next_neighbor[j]]
119         # elif pMin == 4:
120         #     new_strat_array[i,j] = strat_array[i,previous_neighbor[j]]
121
122     strat_array = new_strat_array.copy()
123
124     # Images for animation
125     im = ax1.imshow(new_strat_array.copy(), vmin=0, vmax=N, animated=True)
126     ims.append([im])
127
128     no_0[t] = np.count_nonzero(strat_array == 0)
129     no_1[t] = np.count_nonzero(strat_array == 1)
130     no_2[t] = np.count_nonzero(strat_array == 2)
131     no_3[t] = np.count_nonzero(strat_array == 3)
132     no_4[t] = np.count_nonzero(strat_array == 4)
133     no_5[t] = np.count_nonzero(strat_array == 5)
134     no_6[t] = np.count_nonzero(strat_array == 6)
135     no_7[t] = np.count_nonzero(strat_array == 7)
136
137     fig1.colorbar(im, ax=ax1)
138     ani = animation.ArtistAnimation(fig1, ims, interval=50, blit=True)
139     writergif = animation.PillowWriter(fps=30)
140     ani.save('exercise_13.4_R{}.gif'.format(R), writer=writergif)
141
142     fig2, ax2 = plt.subplots(figsize=(6,6))
143     title = 'L = {}, T = {}, R = {}, P = {}, S = {},  $\mu$  = {}'.format(L,T,R,P,S,mu)
144     ax2.set_title(title)
145     ax2.imshow(strat_array, vmin=0, vmax=N)
146     ax2.set_yticks(())
147     ax2.set_xticks(())
148     ax2.set_xlabel('$t$ = {}'.format(timesteps))
149     fig2.colorbar(im, ax=ax2)
150     plt.savefig('exercise_13.4_R{}.png'.format(R), bbox_inches='tight')
151
152     fig3, ax3 = plt.subplots(figsize=(6,6))
153     t_linspace = np.linspace(0,timesteps,timesteps)
154     ax3.plot(t_linspace, no_0/(L*L), label='Strat. 0')
155     ax3.plot(t_linspace, no_1/(L*L), label='Strat. 1')
156     ax3.plot(t_linspace, no_2/(L*L), label='Strat. 2')
157     ax3.plot(t_linspace, no_3/(L*L), label='Strat. 3')
158     ax3.plot(t_linspace, no_4/(L*L), label='Strat. 4')
159     ax3.plot(t_linspace, no_5/(L*L), label='Strat. 5')
160     ax3.plot(t_linspace, no_6/(L*L), label='Strat. 6')
161     ax3.plot(t_linspace, no_7/(L*L), label='Strat. 7')
162     ax3.set_title(title)
163     ax3.set_xlabel('$t$')
164     ax3.set_ylabel('Population fraction')
165     plt.legend(loc="upper left", fontsize=8)
166     plt.savefig('exercise_13.4_PF_R{}.png'.format(R), bbox_inches='tight')
167
168     plt.show()
169
170     # The larger R is (punishment for cooperation) the more the population defects.
171     # The smaller R is, the more the population cooperates.

```

```
172 # Stable strategies are: defect every time, cooperate almost all the time to all the
    time.
173 # Which strategy that will become stable depends on R.
174 # Cooperation and defection fluctuates past each other continuously if R is "right
    in between".
```



```
1 # Exercise 13.5ab
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.animation as animation
5 import sys
6
7 def play_game(self_strat, neighbor_strat):
8
9     coop = True
10    defect = False
11    self_previous = coop
12    neighbor_previous = coop
13    self_punishment = 0
14
15    for round in range(1,N+1):
16
17        if round <= self_strat and neighbor_previous == coop:
18            self_action = coop
19        else:
20            self_action = defect
21
22        if round <= neighbor_strat and self_previous == coop:
23            neighbor_action = coop
24        else:
25            neighbor_action = defect
26
27        if self_action == coop and neighbor_action == coop:
28            self_punishment += R
29        elif self_action == coop and neighbor_action == defect:
30            self_punishment += S
31        elif self_action == defect and neighbor_action == coop:
32            self_punishment += T
33        elif self_action == defect and neighbor_action == defect:
34            self_punishment += P
35
36        self_previous = self_action
37        neighbor_previous = neighbor_action
38
39    return self_punishment
40
41 N = 7
42 T = 0
43 R = 0.46
44 P = 1
45 S = 1.5
46 mu = 0.01
47 timesteps = 500
48
49 L = 30
50 strat_array = np.random.randint(0,N+1,size=(L,L))
51 new_strat_array = strat_array.copy()
52 strat_array_t0 = strat_array.copy()
53
54 # Distribution fraction
55 t_variance = 0
56 omit_timesteps = 100
57 no_0 = np.zeros(timesteps-omit_timesteps)
58 no_1 = no_0.copy()
59 no_2 = no_0.copy()
```

```

60 no_3 = no_0.copy()
61 no_4 = no_0.copy()
62 no_5 = no_0.copy()
63 no_6 = no_0.copy()
64 no_7 = no_0.copy()
65
66 no_0[0] = np.count_nonzero(strat_array == 0)
67 no_1[0] = np.count_nonzero(strat_array == 1)
68 no_2[0] = np.count_nonzero(strat_array == 2)
69 no_3[0] = np.count_nonzero(strat_array == 3)
70 no_4[0] = np.count_nonzero(strat_array == 4)
71 no_5[0] = np.count_nonzero(strat_array == 5)
72 no_6[0] = np.count_nonzero(strat_array == 6)
73 no_7[0] = np.count_nonzero(strat_array == 7)
74
75
76 # Animation
77 fig1, ax1 = plt.subplots()
78 ims = []
79 im = ax1.imshow(strat_array_t0, vmin=0, vmax=N, animated=True)
80 ims.append([im])
81
82 for t in range(1,timesteps):
83
84     P_array = np.zeros_like(strat_array)
85     next_neighbor = np.roll(np.arange(L),-1)
86     previous_neighbor = np.roll(np.arange(L),1)
87
88     # Accumulate punishment for every agent i,j
89     for i in range(L):
90         for j in range(L):
91
92             # Punishment for Von Neumann neighbors and self
93             pSelf = play_game(strat_array[i,j], strat_array[i,j])
94             pUp = play_game(strat_array[i,j], strat_array[previous_neighbor[i],j])
95             pLeft = play_game(strat_array[i,j], strat_array[i,previous_neighbor[j]])
96             pDown = play_game(strat_array[i,j], strat_array[next_neighbor[i],j])
97             pRight = play_game(strat_array[i,j], strat_array[i,next_neighbor[j]])
98             P_array[i,j] = pUp + pLeft + pDown + pRight
99
100     # Compute new strategies for every agent
101     for i in range(L):
102         for j in range(L):
103
104             r = np.random.uniform()
105             if r < mu:
106                 new_strat_array[i,j] = np.random.randint(0,N+1)
107             else:
108                 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]]]
109                 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]]]
110                 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)])
111
112                 # 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]]])

```

```

113         # if pMin == 0:
114         #     new_strat_array[i,j] = strat_array[i,j]
115         # if pMin == 1:
116         #     new_strat_array[i,j] = strat_array[next_neighbor[i],j]
117         # elif pMin == 2:
118         #     new_strat_array[i,j] = strat_array[previous_neighbor[i],j]
119         # elif pMin == 3:
120         #     new_strat_array[i,j] = strat_array[i,next_neighbor[j]]
121         # elif pMin == 4:
122         #     new_strat_array[i,j] = strat_array[i,previous_neighbor[j]]
123
124     strat_array = new_strat_array.copy()
125
126     # Images for animation
127     im = ax1.imshow(new_strat_array.copy(), vmin=0, vmax=N, animated=True)
128     ims.append([im])
129
130     if t >= omit_timesteps:
131         no_0[t_variance] = np.count_nonzero(strat_array == 0)
132         no_1[t_variance] = np.count_nonzero(strat_array == 1)
133         no_2[t_variance] = np.count_nonzero(strat_array == 2)
134         no_3[t_variance] = np.count_nonzero(strat_array == 3)
135         no_4[t_variance] = np.count_nonzero(strat_array == 4)
136         no_5[t_variance] = np.count_nonzero(strat_array == 5)
137         no_6[t_variance] = np.count_nonzero(strat_array == 6)
138         no_7[t_variance] = np.count_nonzero(strat_array == 7)
139         t_variance += 1
140     print('R = {}, S = {}'.format(R,S))
141
142     # Mean and variance of the population
143     no_0_mean = np.sum(no_0) / len(no_0)
144     no_1_mean = np.sum(no_1) / len(no_1)
145     no_2_mean = np.sum(no_2) / len(no_2)
146     no_3_mean = np.sum(no_3) / len(no_3)
147     no_4_mean = np.sum(no_4) / len(no_4)
148     no_5_mean = np.sum(no_5) / len(no_5)
149     no_6_mean = np.sum(no_6) / len(no_6)
150     no_7_mean = np.sum(no_7) / len(no_7)
151
152     mean_array = np.round(np.array([no_0_mean, no_1_mean, no_2_mean, no_3_mean,
153                                     no_4_mean, no_5_mean, no_6_mean, no_7_mean]),2)
154     [print(f'Mean strat. {i}: ', mean_array[i]) for i in range(len(mean_array))]
155     print('\n')
156
157     no_0_variance = np.sum((no_0-no_0_mean)**2) / len(no_0)
158     no_1_variance = np.sum((no_1-no_1_mean)**2) / len(no_1)
159     no_2_variance = np.sum((no_2-no_2_mean)**2) / len(no_2)
160     no_3_variance = np.sum((no_3-no_3_mean)**2) / len(no_3)
161     no_4_variance = np.sum((no_4-no_4_mean)**2) / len(no_4)
162     no_5_variance = np.sum((no_5-no_5_mean)**2) / len(no_5)
163     no_6_variance = np.sum((no_6-no_6_mean)**2) / len(no_6)
164     no_7_variance = np.sum((no_7-no_7_mean)**2) / len(no_7)
165
166     variance_array = np.round(np.array([no_0_variance, no_1_variance, no_2_variance,
167                                         no_3_variance, no_4_variance, no_5_variance, no_6_variance, no_7_variance]), 2)
168     [print(f'Variance strat. {i}: ', variance_array[i]) for i in
169         range(len(variance_array))]
170     variance_sum = np.sum(variance_array)
171     print('Variance sum: {}'.format(variance_sum))
172     print('\n')

```

```
170
171 deviation_array = np.round(variance_array**0.5, 2)
172 [print(f'Standard deviation strat. {i}: ', deviation_array[i]) for i in
    range(len(deviation_array))]
173
174 fig1.colorbar(im, ax=ax1)
175 ani = animation.ArtistAnimation(fig1, ims, interval=50, blit=True)
176 writergif = animation.PillowWriter(fps=30)
177 ani.save('exercise_13.5_R{}_S{}.gif'.format(R,S), writer=writergif)
178
179 fig2, ax2 = plt.subplots(figsize=(6,6))
180 title = 'L = {}, T = {}, R = {}, P = {}, S = {},  $\mu$  = {}'.format(L,T,R,P,S,mu)
181 ax2.set_title(title)
182 ax2.imshow(strat_array, vmin=0, vmax=N)
183 ax2.set_yticks(())
184 ax2.set_xticks(())
185 ax2.set_xlabel('$t$ = {}'.format(timesteps))
186 fig2.colorbar(im, ax=ax2)
187 plt.savefig('exercise_13.5_R{}_S{}.png'.format(R,S), bbox_inches='tight')
188
189 fig3, ax3 = plt.subplots(figsize=(6,6))
190 t_arange = np.arange(omit_timesteps,timesteps,1)
191 ax3.plot(t_arange, no_0/(L*L), label='Strat. 0')
192 ax3.plot(t_arange, no_1/(L*L), label='Strat. 1')
193 ax3.plot(t_arange, no_2/(L*L), label='Strat. 2')
194 ax3.plot(t_arange, no_3/(L*L), label='Strat. 3')
195 ax3.plot(t_arange, no_4/(L*L), label='Strat. 4')
196 ax3.plot(t_arange, no_5/(L*L), label='Strat. 5')
197 ax3.plot(t_arange, no_6/(L*L), label='Strat. 6')
198 ax3.plot(t_arange, no_7/(L*L), label='Strat. 7')
199 ax3.set_title(title)
200 ax3.set_xlabel('$t$')
201 ax3.set_ylabel('Population fraction')
202 plt.legend(loc="upper left", fontsize=8)
203 plt.savefig('exercise_13.5_PF_R{}_S{}.png'.format(R,S), bbox_inches='tight')
204
205 plt.show()
```