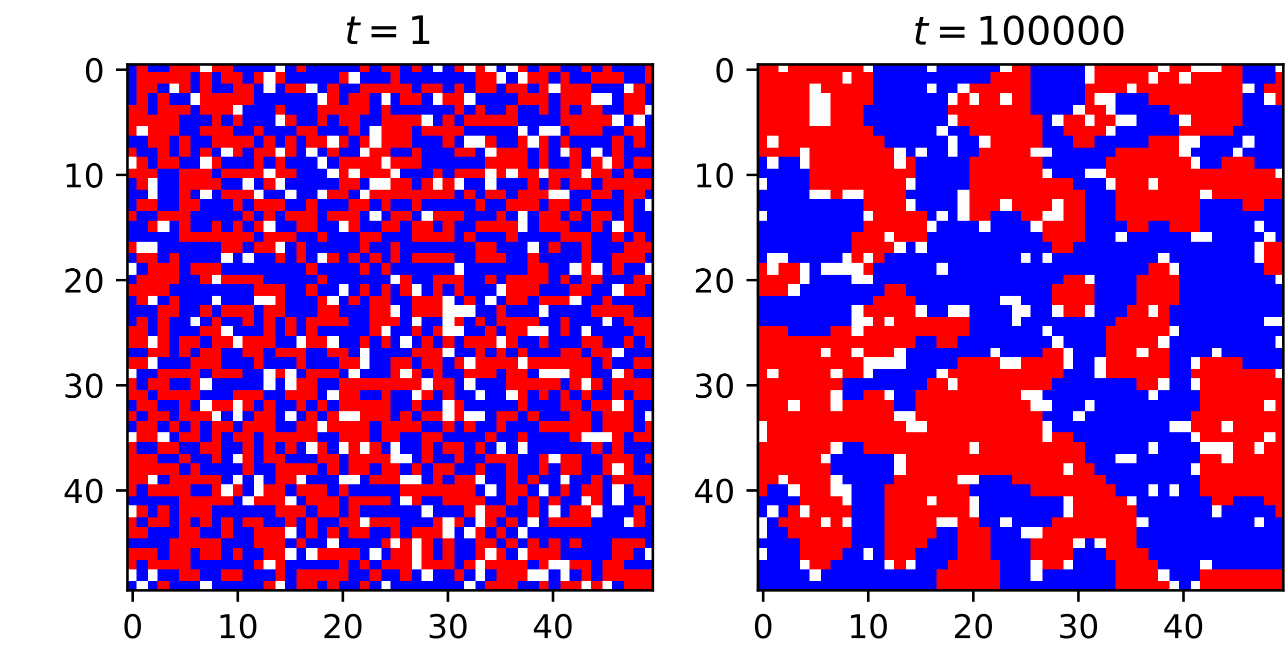
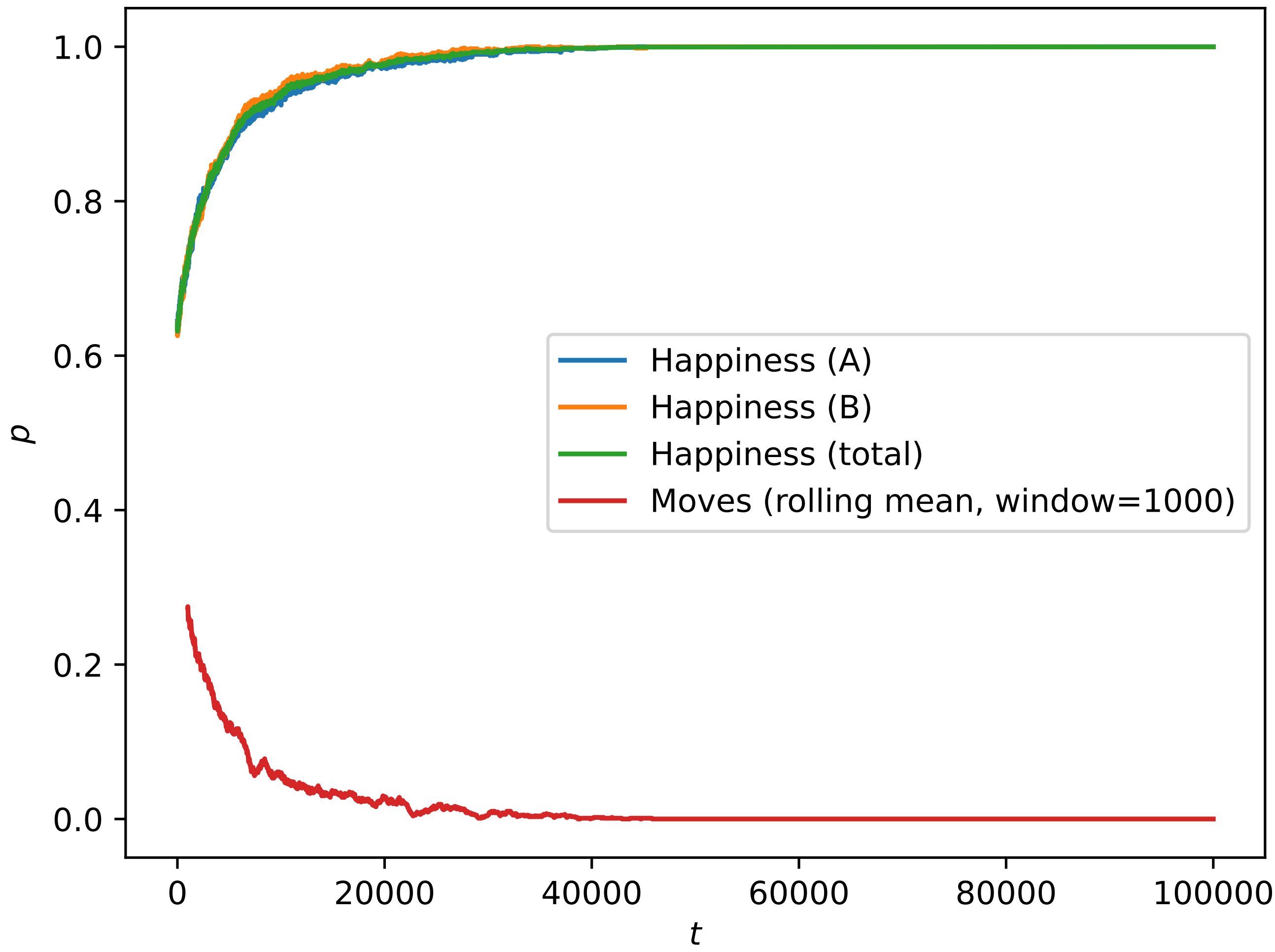
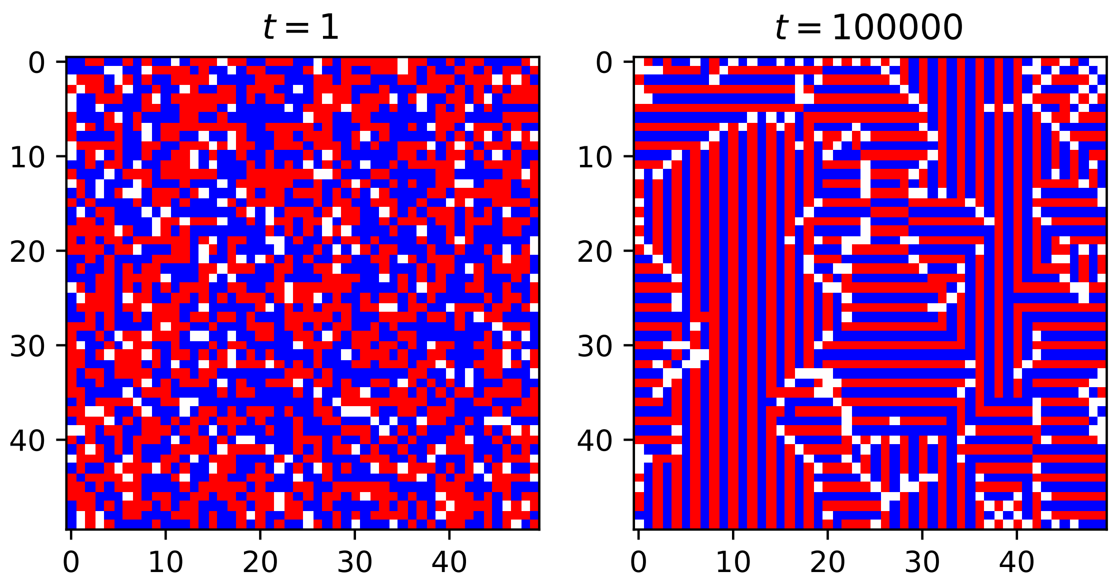
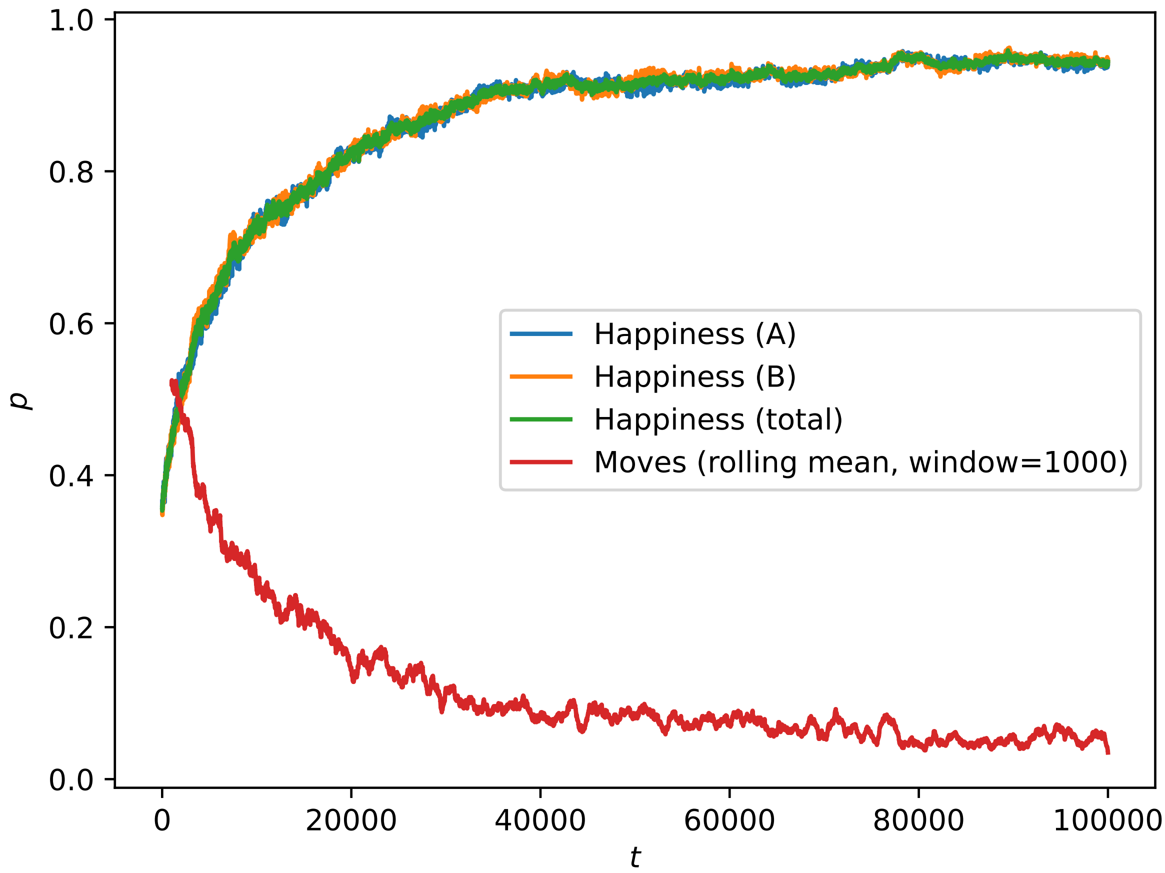

$t = 1$
$t = 100000$
0
10
20
30
40

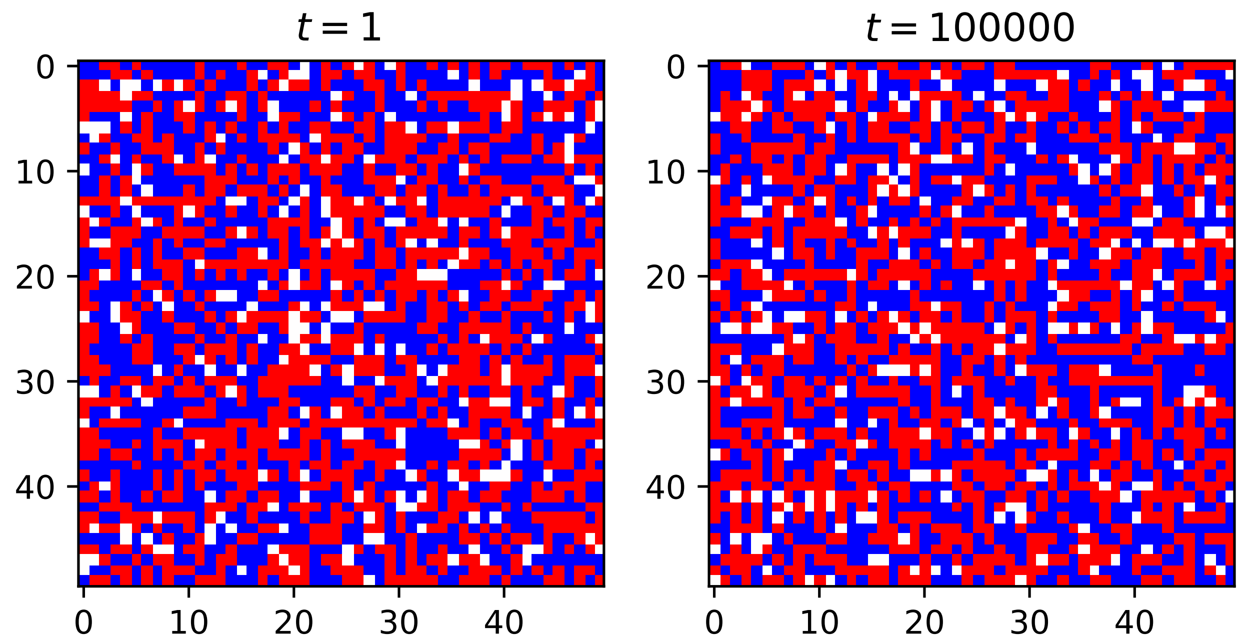

Happiness (A)
Happiness (B)
Happiness (total)
Moves (rolling mean, window=1000)
p
t
0.0
0.2
0.4
0.6
0.8
1.0
0
20000
40000
60000
80000
100000



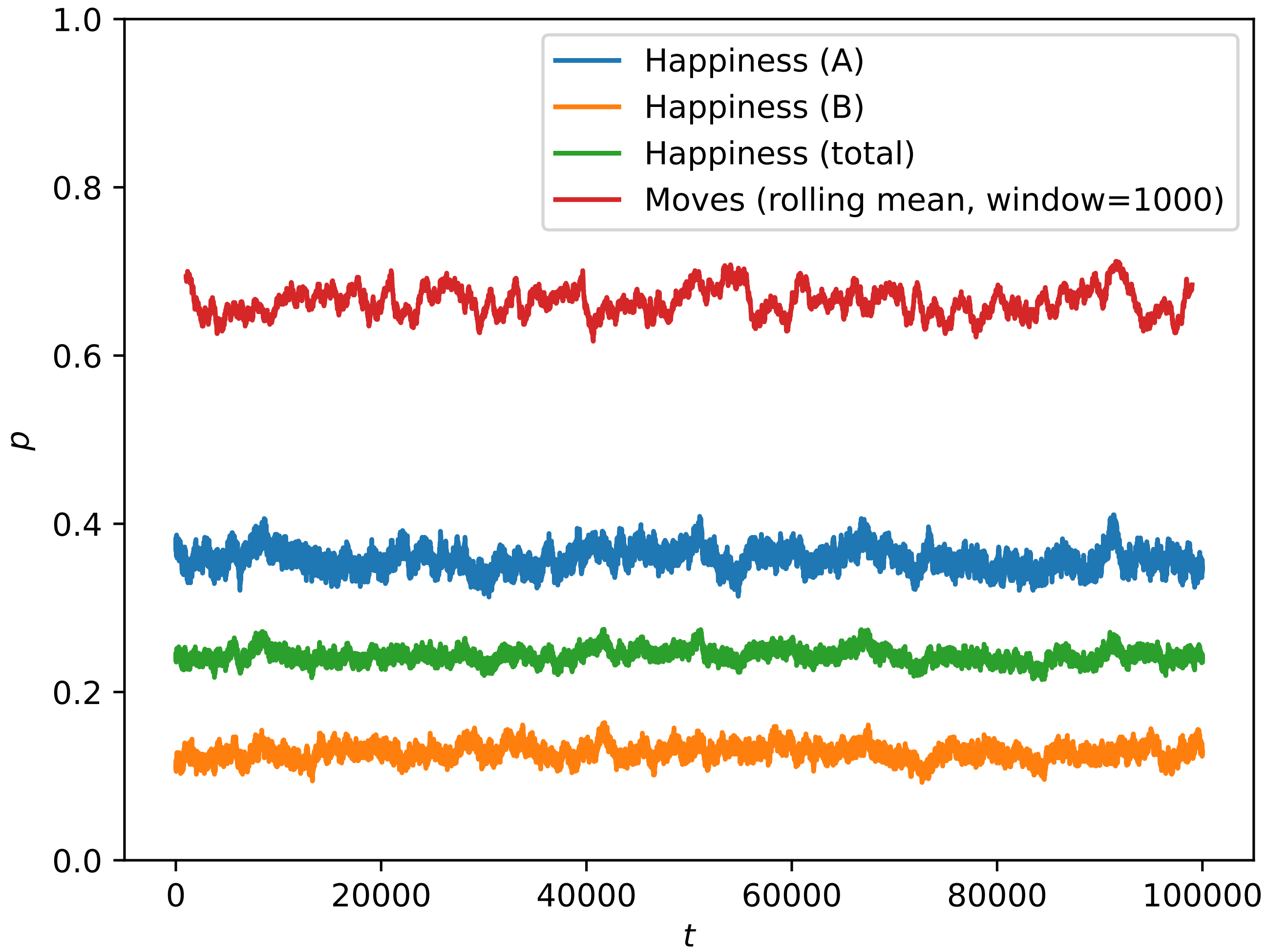
$t = 1$
$t = 100000$
0
10
20
30
40

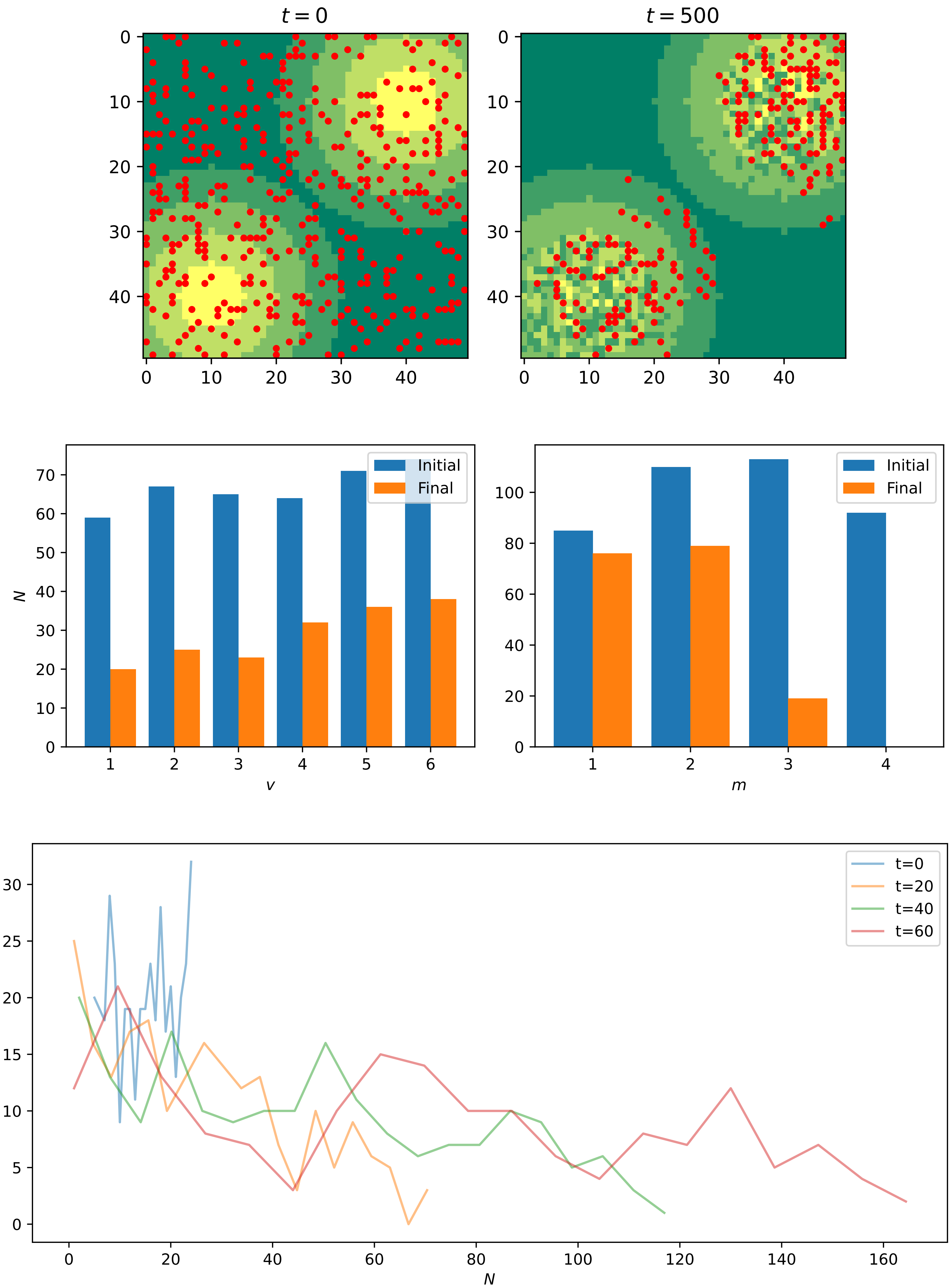
Happiness (A)
Happiness (B)
Happiness (total)
Moves (rolling mean, window=1000)
$p$
$t$
0.0
0.2
0.4
0.6
0.8
1.0
0
20000
40000
60000
80000
100000



$t = 1$
$t = 100000$
0
10
20
30
40

Happiness (A)
Happiness (B)
Happiness (total)
Moves (rolling mean, window=1000)
p
t
0.0
0.2
0.4
0.6
0.8
1.0
0
20000
40000
60000
80000
100000



$t = 0$
$t = 500$
Initial
Final
$N$
$v$
Initial
Final
$m$
t=0
t=20
t=40
t=60
$N$

/Schelling-Model/main.py

```
import numpy as np

from matplotlib import pyplot as plt
from scipy.signal import fftconvolve
from tqdm import trange

IMAGE = 0
HAPPINESS = 1

def get_majorities(town: np.ndarray) -> int:
    kernel = np.ones((3, 3))
    conv = fftconvolve(town, kernel, mode="same")
    # Has the same value as the majority (or 0) of the neighbourghood
    major = (conv > 0.0) * 1 - (conv < 0.0) * 1
    return major

if __name__ == "__main__":
    N = 50
    town = np.concatenate((np.zeros(int(N*N*0.1)), np.ones(int(N*N*0.45)), np.ones(int(N*N
        *0.45)) * (-1)))
    np.random.shuffle(town)
    town = town.reshape((N,N))

    town0 = np.copy(town)


    T = 100000
    a_happy = np.zeros(T)
    b_happy = np.zeros(T)
    moving  = np.zeros(T)

    plotting = IMAGE

    for t in trange(T):
        majorities = get_majorities(town)
        happy = majorities * town

        a_happy[t] = np.count_nonzero(np.logical_and(happy > 0, town > 0)) / int(N*N*0.45)
        b_happy[t] = np.count_nonzero(np.logical_and(happy > 0, town < 0)) / int(N*N*0.45)
        # Find non-empty house
        while True:
            i = np.random.randint(0, N)
            j = np.random.randint(0, N)
            if town[i,j] != 0:
                break

        if happy[i, j] == -1:
            # Find empty house to move to
            moving[t] = 1
            while True:
                i_move = np.random.randint(0, N)
                j_move = np.random.randint(0, N)
                if town[i_move,j_move] == 0 and i_move != i and j_move != j:
                    break

            town[i_move,j_move] = town[i,j]
            town[i,j] = 0

    if plotting == IMAGE:
        plt.subplot(1, 2, 1)
        plt.imshow(town0, cmap="bwr")
        plt.title("$t=1$")

        plt.subplot(1, 2, 2)
        plt.imshow(town, cmap="bwr")
```

```
        plt.title(f"$t={T}$")
    elif plotting == HAPPINESS:
        plt.plot(np.arange(T), a_happy, label="Happiness (A)")
        plt.plot(np.arange(T), b_happy, label="Happiness (B)")
        plt.plot(np.arange(T), (a_happy + b_happy) / 2, label="Happiness (total)")
        plt.plot(np.arange(T)[1000:], (fftconvolve(moving, np.ones(1000), mode="same")/1000)
            [1000:], label="Moves (rolling mean, window=1000)")
        plt.legend()
        plt.xlabel("$t$")
        plt.ylabel("$p$")

    plt.show()
```

/Anti-Gregarious/main.py

```
import numpy as np

from matplotlib import pyplot as plt
from scipy.signal import fftconvolve
from tqdm import trange

IMAGE = 0
HAPPINESS = 1

def get_majorities(town: np.ndarray) -> int:
    kernel = np.ones((3, 3))
    conv = fftconvolve(town, kernel, mode="same")
    # Has the same value as the majority (or 0) of the neighbourghood
    major = (conv > 0.0) * 1 - (conv < 0.0) * 1
    return major

if __name__ == "__main__":
    N = 50
    town = np.concatenate((np.zeros(int(N*N*0.1)), np.ones(int(N*N*0.45)), np.ones(int(N*N
        *0.45)) * (-1)))
    np.random.shuffle(town)
    town = town.reshape((N,N))

    town0 = np.copy(town)


    T = 100000
    a_happy = np.zeros(T)
    b_happy = np.zeros(T)
    moving  = np.zeros(T)

    plotting = HAPPINESS

    for t in trange(T):
        majorities = get_majorities(town)
        happy = majorities * town

        a_happy[t] = np.count_nonzero(np.logical_and(happy < 0, town > 0)) / int(N*N*0.45)
        b_happy[t] = np.count_nonzero(np.logical_and(happy < 0, town < 0)) / int(N*N*0.45)
        # Find non-empty house
        while True:
            i = np.random.randint(0, N)
            j = np.random.randint(0, N)
            if town[i,j] != 0:
                break

        if happy[i, j] == 1:
            # Find empty house to move to
            moving[t] = 1
            while True:
                i_move = np.random.randint(0, N)
                j_move = np.random.randint(0, N)
                if town[i_move,j_move] == 0 and i_move != i and j_move != j:
                    break

            town[i_move,j_move] = town[i,j]
            town[i,j] = 0

    if plotting == IMAGE:
        plt.subplot(1, 2, 1)
        plt.imshow(town0, cmap="bwr")
        plt.title("$t=1$")

        plt.subplot(1, 2, 2)
        plt.imshow(town, cmap="bwr")
```

```
        plt.title(f"$t={T}$")
    elif plotting == HAPPINESS:
        plt.plot(np.arange(T), a_happy, label="Happiness (A)")
        plt.plot(np.arange(T), b_happy, label="Happiness (B)")
        plt.plot(np.arange(T), (a_happy + b_happy) / 2, label="Happiness (total)")
        plt.plot(np.arange(T)[1000:], (fftconvolve(moving, np.ones(1000), mode="same")/1000)
            [1000:], label="Moves (rolling mean, window=1000)")
        plt.legend()
        plt.xlabel("$t$")
        plt.ylabel("$p$")

    plt.show()
```

/Frustration/main.py

```
import numpy as np

from matplotlib import pyplot as plt
from scipy.signal import fftconvolve
from tqdm import trange

IMAGE = 0
HAPPINESS = 1

def get_happiness(town: np.ndarray) -> int:
    kernel = np.ones((3, 3))
    conv = fftconvolve(town, kernel, mode="same")
    # Family A are happy when in the minory
    # Family B are happy when in the large majority
    major = (conv < 0.0) * 1 - (conv > 2.0) * 1
    return major

if __name__ == "__main__":
    N = 50
    town = np.concatenate((np.zeros(int(N*N*0.1)), np.ones(int(N*N*0.45)), np.ones(int(N*N
        *0.45)) * (-1)))
    np.random.shuffle(town)
    town = town.reshape((N,N))

    town0 = np.copy(town)


    T = 100000
    a_happy = np.zeros(T)
    b_happy = np.zeros(T)
    moving  = np.zeros(T)

    plotting = IMAGE

    for t in trange(T):
        happy = get_happiness(town) * town

        a_happy[t] = np.count_nonzero(np.logical_and(happy > 0, town > 0)) / int(N*N*0.45)
        b_happy[t] = np.count_nonzero(np.logical_and(happy > 0, town < 0)) / int(N*N*0.45)
        # Find non-empty house
        while True:
            i = np.random.randint(0, N)
            j = np.random.randint(0, N)
            if town[i,j] != 0:
                break

        if happy[i, j]:
            # Find empty house to move to
            moving[t] = 1
            while True:
                i_move = np.random.randint(0, N)
                j_move = np.random.randint(0, N)
                if town[i_move,j_move] == 0 and i_move != i and j_move != j:
                    break

            town[i_move,j_move] = town[i,j]
            town[i,j] = 0

    if plotting == IMAGE:
        plt.subplot(1, 2, 1)
        plt.imshow(town0, cmap="bwr")
        plt.title("$t=1$")

        plt.subplot(1, 2, 2)
        plt.imshow(town, cmap="bwr")
```

```
        plt.title(f"$t={T}$")

        print(np.count_nonzero(town0), np.count_nonzero(town))
    elif plotting == HAPPINESS:
        plt.plot(np.arange(T), a_happy, label="Happiness (A)")
        plt.plot(np.arange(T), b_happy, label="Happiness (B)")
        plt.plot(np.arange(T), (a_happy + b_happy) / 2, label="Happiness (total)")
        plt.plot(np.arange(T)[1000:-1000], (fftconvolve(moving, np.ones(1000), mode="same")
            /1000)[1000:-1000], label="Moves (rolling mean, window=1000)")
        plt.legend()
        plt.xlabel("$t$")
        plt.ylabel("$p$")

        plt.ylim((0, 1))

    plt.show()
```

/Sugar-Scape/main.py

```
import numpy as np
import random

from matplotlib import pyplot as plt
from tqdm import trange

def place_sugar(grid: np.ndarray, row: int, col: int, radius: float, sugar_level: float):
    for i in range(grid.shape[0]):
        for j in range(grid.shape[1]):
            if np.sqrt((row - i) ** 2 + (col - j) ** 2) <= radius:
                grid[i,j] = sugar_level

def place_agents(agents: np.ndarray, N: int):
    agent_grid = np.zeros((N, N))
    for a in range(agents.shape[0]):
        while True:
            i = np.random.randint(0, N)
            j = np.random.randint(0, N)
            if agent_grid[i,j] == 0:
                break

        agents[a,0] = i
        agents[a,1] = j

        agent_grid[i,j] = 1
    return agent_grid

IMAGE = 0
DISTS = 1
W_HIS = 2

if __name__ == "__main__":
    A = 400
    N = 50
    T = 500

    sugar_grid = np.zeros((N, N))

    place_sugar(sugar_grid, 40, 10, 20, 1)
    place_sugar(sugar_grid, 40, 10, 15, 2)
    place_sugar(sugar_grid, 40, 10, 10, 3)
    place_sugar(sugar_grid, 40, 10,  5, 4)

    place_sugar(sugar_grid, 10, 40, 20, 1)
    place_sugar(sugar_grid, 10, 40, 15, 2)
    place_sugar(sugar_grid, 10, 40, 10, 3)
    place_sugar(sugar_grid, 10, 40,  5, 4)

    sugar_capacity = sugar_grid.copy()

    # One agent: [ j, k, s, v, m, alive ]
    agents = np.zeros((A, 6))

    agents[:,2] = np.random.randint(5, 26, size=(A,))
    agents[:,3] = np.random.randint(1, 7, size=(A,))
    agents[:,4] = np.random.randint(1, 5, size=(A,))
    agents[:,5] = np.ones((A,))

    agent_grid = place_agents(agents, N)

    plotting = W_HIS

    if plotting == IMAGE:
        plt.subplot(1, 2, 1)
        plt.imshow(sugar_grid.copy(), cmap="summer")
        plt.plot(agents[:,0], agents[:,1], 'r.')
```

```
    plt.title("$t=0$")

if plotting == DISTS:
    v_hist, v_bins = np.histogram(agents[:,3], bins=[1, 2, 3, 4, 5, 6, 7])
    m_hist, m_bins = np.histogram(agents[:,4], bins=[1, 2, 3, 4, 5])
    plt.subplot(1, 2, 1)
    plt.bar(v_bins[:-1] - 0.2, v_hist, width=0.4, label="Initial")
    plt.xlabel("$v$")
    plt.ylabel("$N$")
    plt.subplot(1, 2, 2)
    plt.bar(m_bins[:-1] - 0.2, m_hist, width=0.4, label="Initial")
    plt.xlabel("$m$")


for t in trange(T):
    # Update agents
    agent_order = list(range(agents.shape[0]))
    random.shuffle(agent_order)

    if plotting == W_HIS:
        if t % 20 == 0 and t < 80:
            alive_agents = agents[agents[:,5] > 0]
            s_hist, s_bins = np.histogram(alive_agents[:,2], bins=20)
            plt.plot(s_bins[:-1], s_hist, label=f"t={t}", alpha=0.5)

    for i in agent_order:
        if agents[i,5] == 0:
            continue

        j = int(agents[i,0])
        k = int(agents[i,1])
        # Sugar
        agents[i,2] += (sugar_grid[j,k] - agents[i,4]) * agents[i,5]
        agents[i,5] = (agents[i,2] > 0) * 1
        sugar_grid[j,k] = 0

        # Move
        max_sugar = -1

        candidates = []

        vision = int(agents[i,3]+1)

        # Right
        for v in range(1,vision):
            if agent_grid[min(j+v,N-1),k] == 0:
                if sugar_grid[min(j+v,N-1),k] > max_sugar:
                    candidates.clear()
                    candidates.append( (min(j+v,N-1),k) )
                    max_sugar = sugar_grid[min(j+v,N-1),k]
        # Left
        for v in range(1,vision):
            if agent_grid[max(j-v,0),k] == 0:
                if sugar_grid[max(j-v,0),k] > max_sugar:
                    candidates.clear()
                    candidates.append( (max(j-v,0),k) )
                    max_sugar = sugar_grid[max(j-v,0),k]
        # Up
        for v in range(1,vision):
            if agent_grid[j,max(k-v,0)] == 0:
                if sugar_grid[j,max(k-v,0)] > max_sugar:
                    candidates.clear()
                    candidates.append( (j,max(k-v,0)) )
                    max_sugar = sugar_grid[j,max(k-v,0)]
        # Down
        for v in range(1,vision):
            if agent_grid[j,min(k+v,N-1)] == 0:
```

```
                if sugar_grid[j,min(k+v,N-1)] > max_sugar:
                    candidates.clear()
                    candidates.append( (j,min(k+v,N-1)) )
                    max_sugar = sugar_grid[j,min(k+v,N-1)]

        if len(candidates) > 0:
            max_j, max_k = random.choice(candidates)
        else:
            max_j, max_k = j, k

        agent_grid[j,k] = 0
        agent_grid[max_j,max_k] = 1
        agents[i,0] = max_j
        agents[i,1] = max_k

    # Update sugar scape
    sugar_grid += 1
    sugar_grid = np.minimum(sugar_grid, sugar_capacity)

alive_agents = agents[agents[:,5] > 0]

print(alive_agents.shape)

if plotting == IMAGE:
    plt.subplot(1, 2, 2)
    plt.imshow(sugar_grid, cmap="summer")
    plt.plot(alive_agents[:,0], alive_agents[:,1], 'r.')
    plt.title(f"$t={T}$")

if plotting == DISTS:
    v_hist, v_bins = np.histogram(alive_agents[:,3], bins=[1, 2, 3, 4, 5, 6, 7])
    m_hist, m_bins = np.histogram(alive_agents[:,4], bins=[1, 2, 3, 4, 5])
    plt.subplot(1, 2, 1)
    plt.bar(v_bins[:-1] + 0.2, v_hist, width=0.4, label="Final")
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.bar(m_bins[:-1] + 0.2, m_hist, width=0.4, label="Final")
    plt.legend()

if plotting == W_HIS:
    plt.xlabel("$s$")
    plt.xlabel("$N$")
    plt.legend()
plt.show()
```