Sprawozdanie z listy 1 - Automat komórkowy - Plamy

Mateusz Wojteczek

Wstęp

Celem tej pracy było zaprojektowanie i stworzenie programu komputerowego symulującego ewolucję plam za pomocą automatu komórkowego, wedle wskazówek z wykładu. Automat ten działa na dwuwymiarowej siatce, gdzie każda komórka może znajdować się w jednym z dwóch stanów: aktywnym lub nieaktywnym (0 - nieaktywna, czarna lub 1 - aktywna, biała). Głównym celem zadania było przygotowanie analizy ewolucji układu w formie graficznej oraz wykresów gęstości plam od czasu (iteracji modelu). Analiza przeprowadzona została dla planszy pierwotnie wypełnionej losowymi wartościami, które wedle przyjętych zasad dotyczących rozmnażania się bądź umierania komórek zgodnie z wynikiem sumy sąsiadów komórki (wraz z komórką), dążą do połączenia się i stworzenia plam.

Kod programu

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

# Ustawienia parametrów wyświetlania
width, height = 600, 600  # Definicja szerokości i wysokości okna Pygame
rows, cols = 100, 100  # Definicja liczby wierszy i kolumn w siatce automatu
komórkowego
cell_size = width // cols, height // rows  # Obliczenie rozmiaru każdej komórki w
siatce
black = (0, 0, 0)
white = (255, 255, 255)

# Inicjalizacja Pygame i okna wyświetlania
pygame.init()
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Patterns")

def create_grid():
    """Tworzenie nowej siatki z losowymi stanami początkowymi."""
    # Losowe inicjowanie siatki zerami i jedynkami, gdzie 0 reprezentuje
nieaktywną komórkę, a 1 aktywną komórkę
    return np.random.choice([0, 1], size=(rows, cols))

def draw_grid(current_grid, prev_grid):
    """Rysowanie siatki, optymalizacja poprzez przerysowywanie tylko zmienionych
komórek."""
    for row in range(rows):
        for col in range(cols):
            # Sprawdzanie, czy stan bieżącej komórki zmienił się w porównaniu do
```

```
poprzedniego stanu
            if current_grid[row][col] != prev_grid[row][col]:
                # Ustawianie koloru w zależności od bieżącego stanu komórki
(aktywna lub nieaktywna)
                color = white if current_grid[row][col] == 1 else black
                # Rysowanie komórki jako prostokąta na ekranie Pygame w określonym
kolorze
                pygame.draw.rect(screen, color, (col * cell_size[0], row *
cell_size[1], cell_size[0], cell_size[1]))


def update_grid(grid):
    N = len(grid)
    new_grid = np.zeros((N, N), dtype=int)

    for i in range(N):
        for j in range(N):
            # Obliczanie całkowitej liczby sąsiadujących aktywnych komórek wokół
bieżącej komórki,
            # uwzględniając tym razem także stan samej komórki.
            total_active_neighbors = sum([grid[(i + x) % N][(j + y) % N] for x in
range(-1, 2)
                                          for y in range(-1, 2)])

            # Zasady przejścia z uwzględnieniem dostosowanej sumy
            # Tutaj adnotacja - pierwotny problem występował we wzorze na sumę,
gdzie sumowałem bez stanu komórki w środku (otoczonej sąsiadami) oraz ze zwykłym
niedopatrzeniem w zbiorach gdzie zdublowałem liczby.
            if total_active_neighbors in {0, 1, 2, 3, 5}:
                new_grid[i][j] = 0
            elif total_active_neighbors in {4, 6, 7, 8, 9}:
                new_grid[i][j] = 1
            else:
                # Jeśli suma nie należy do żadnego z tych zbiorów, stan komórki
pozostaje niezmieniony
                new_grid[i][j] = grid[i][j]

    return new_grid




def main():
    current_grid = create_grid()  # Inicjowanie siatki losowymi stanami
    prev_grid = np.zeros((rows, cols))  # Inicjowanie poprzedniego stanu siatki
dla porównania (początkowo same zera)
    running = True  # Ustawienie flagi kontynuacji symulacji

    # Ustawienie wykresu dystrybucji gęstości za pomocą Matplotlib
    plt.ion()  # Włączenie trybu interaktywnego rysowania
    fig, ax = plt.subplots()
    ax.set_title('Rozkład gęstości w czasie')  # Ustawienie tytułu wykresu
    ax.set_xlabel('Iteracja')
    ax.set_ylabel('Gęstość')
```

```
    densities = []  # Inicjowanie listy do przechowywania wartości gęstości w
czasie

    iteration = 0  # Inicjowanie licznika iteracji

    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        draw_grid(current_grid, prev_grid)  # Rysowanie bieżącego stanu siatki
        pygame.display.flip()  # Aktualizacja wyświetlania Pygame, aby pokazać
nowe rysowanie

        # Aktualizacja siatki i zapisanie poprzedniego stanu do porównania w
następnej iteracji
        prev_grid = current_grid.copy()
        current_grid = update_grid(current_grid)

        # Obliczanie i rysowanie gęstości aktywnych komórek w siatce
        density = np.mean(current_grid)  # Obliczanie średniej wartości siatki
(gęstość aktywnych komórek)
        densities.append(density)  # Dodawanie gęstości do listy
        if iteration % 10 == 0:  # Aktualizacja wykresu co 10 iteracji dla
efektywności
            ax.clear()  # Czyszczenie poprzedniego wykresu
            ax.plot(densities)  # Rysowanie nowych wartości gęstości
            ax.set_title('Rozkład gęstości w czasie')  # Resetowanie tytułu
(usuniętego przez ax.clear())
            ax.set_xlabel('Iteracja')  # Resetowanie etykiety osi X
            ax.set_ylabel('Gęstość')  # Resetowanie etykiety osi Y
            plt.pause(0.01)  # Krótka pauza, aby zaktualizować wykres

        iteration += 1  # Inkrementacja licznika iteracji

    pygame.quit()
    plt.ioff()
    plt.show()  # Pokazanie ostatecznego wykresu gęstości

if __name__ == "__main__":
    main()
```

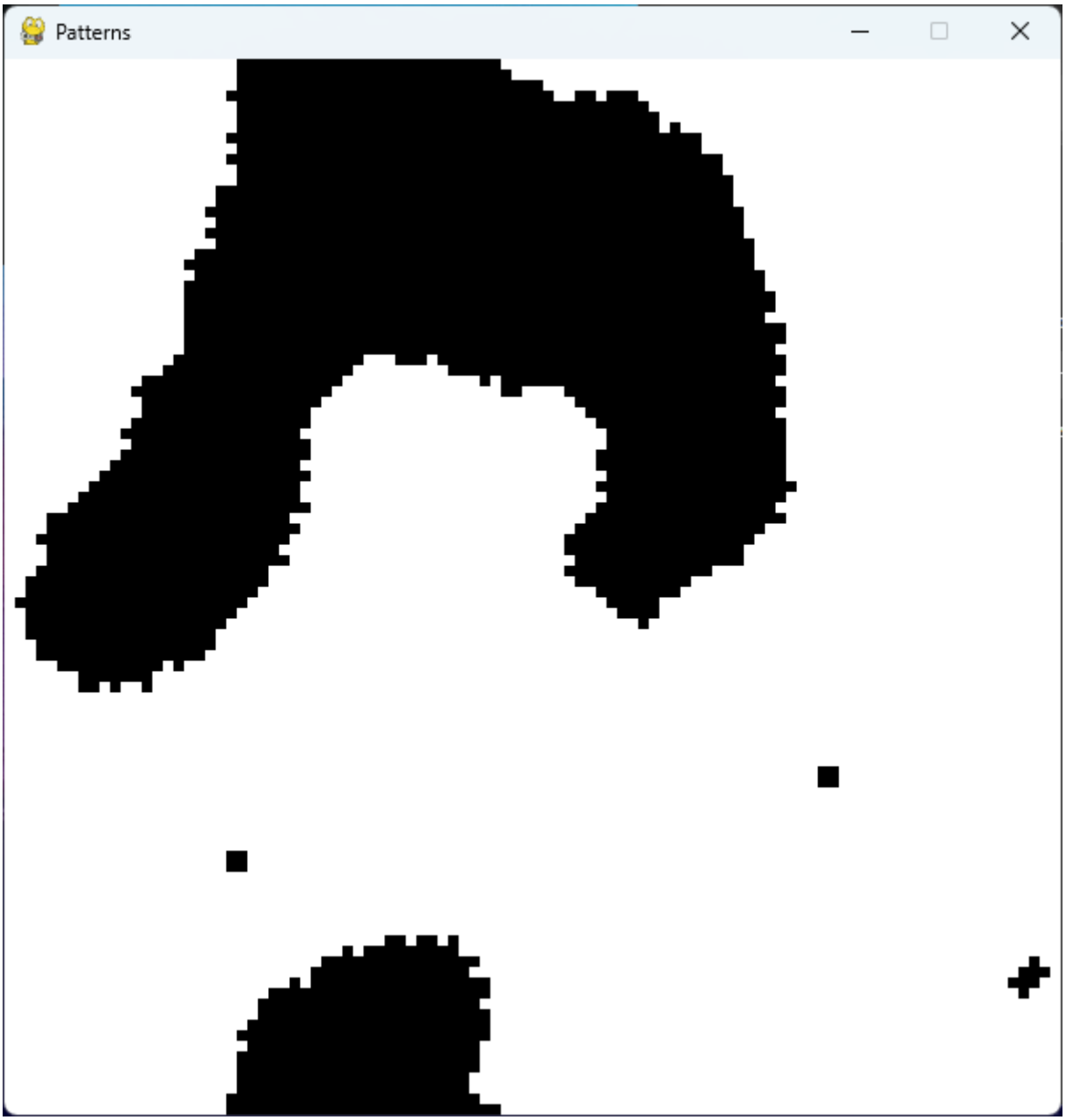
Analiza wyników - Rozkład gęstości

W moim kodzie gęstość w rozkładzie obliczana jest jako średnia wartość siatki, co odpowiada średniej liczbie aktywnych komórek (o wartości 1 - kolor biały) na całej planszy.

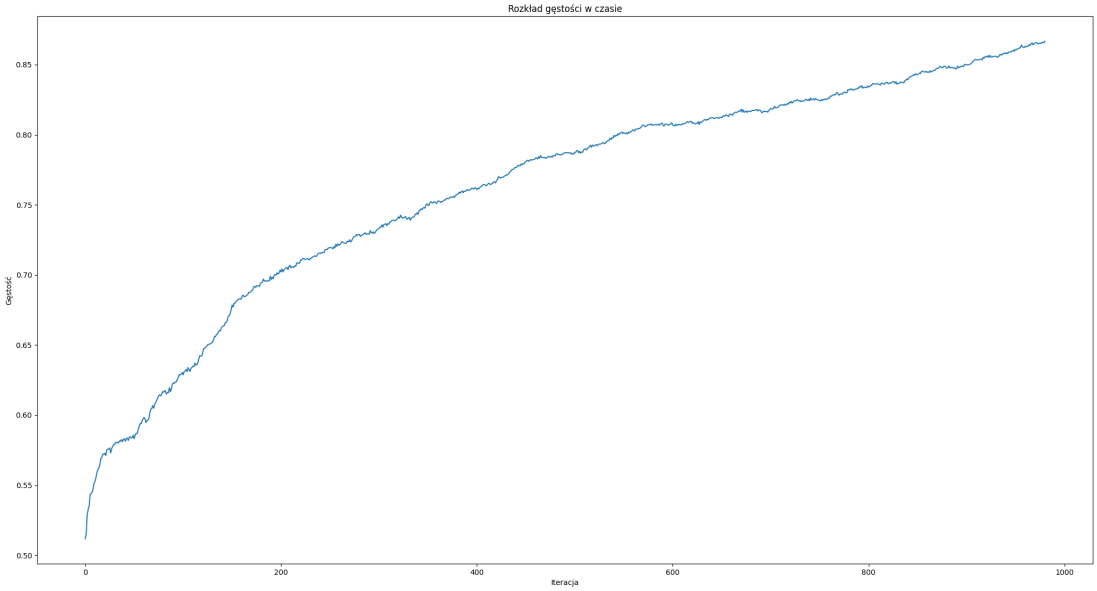
Przeprowadziłem dwie próby, by zaprezentować losowość układu i jego faktyczną umiejętność do formowania plam, wychodząc z początkowych założeń i z losoweo wypełnionej planszy początkowej.

Próba nr. 1

Snapshoty układu


Patterns

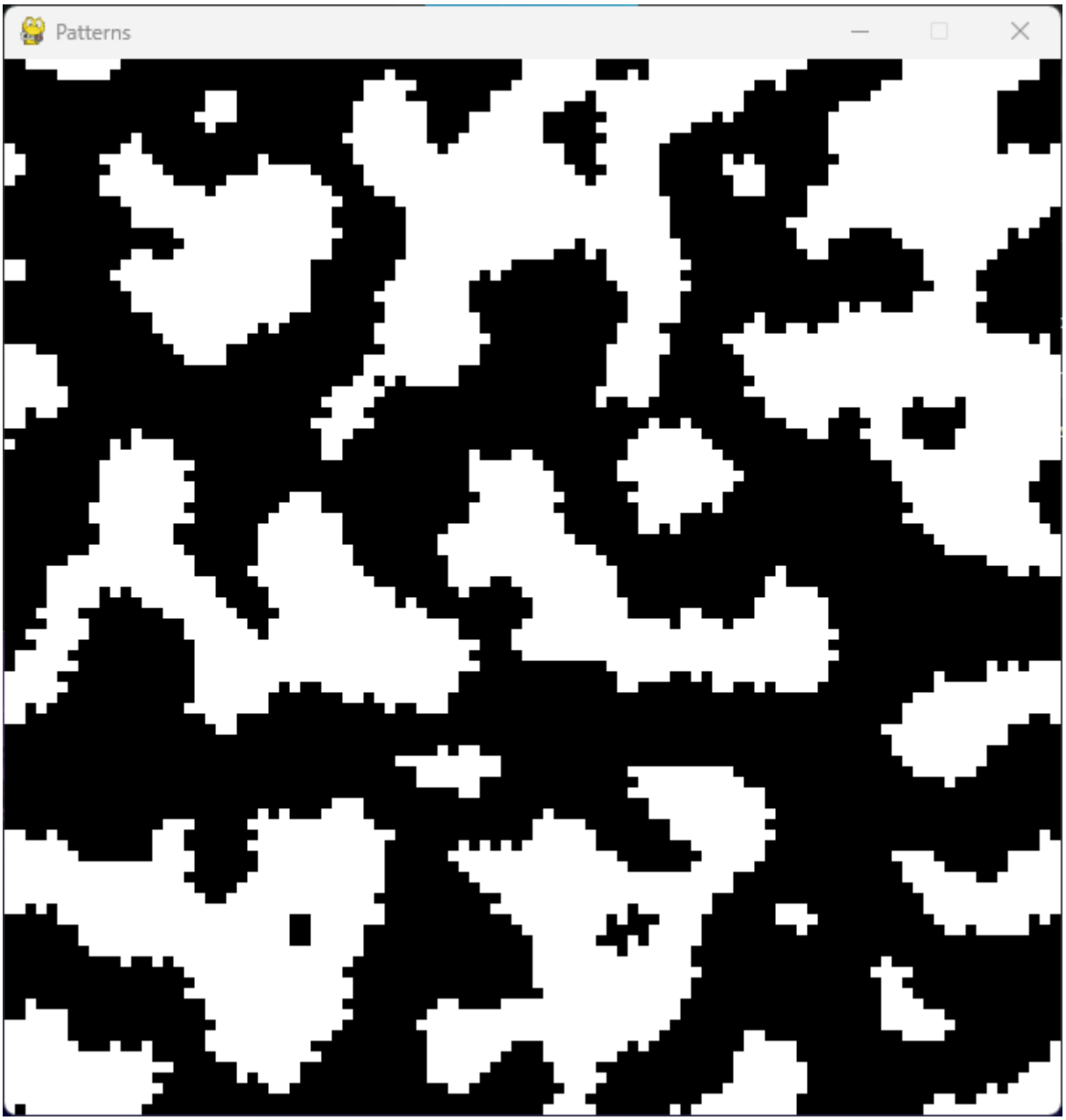
Rozkład gęstości tego układu



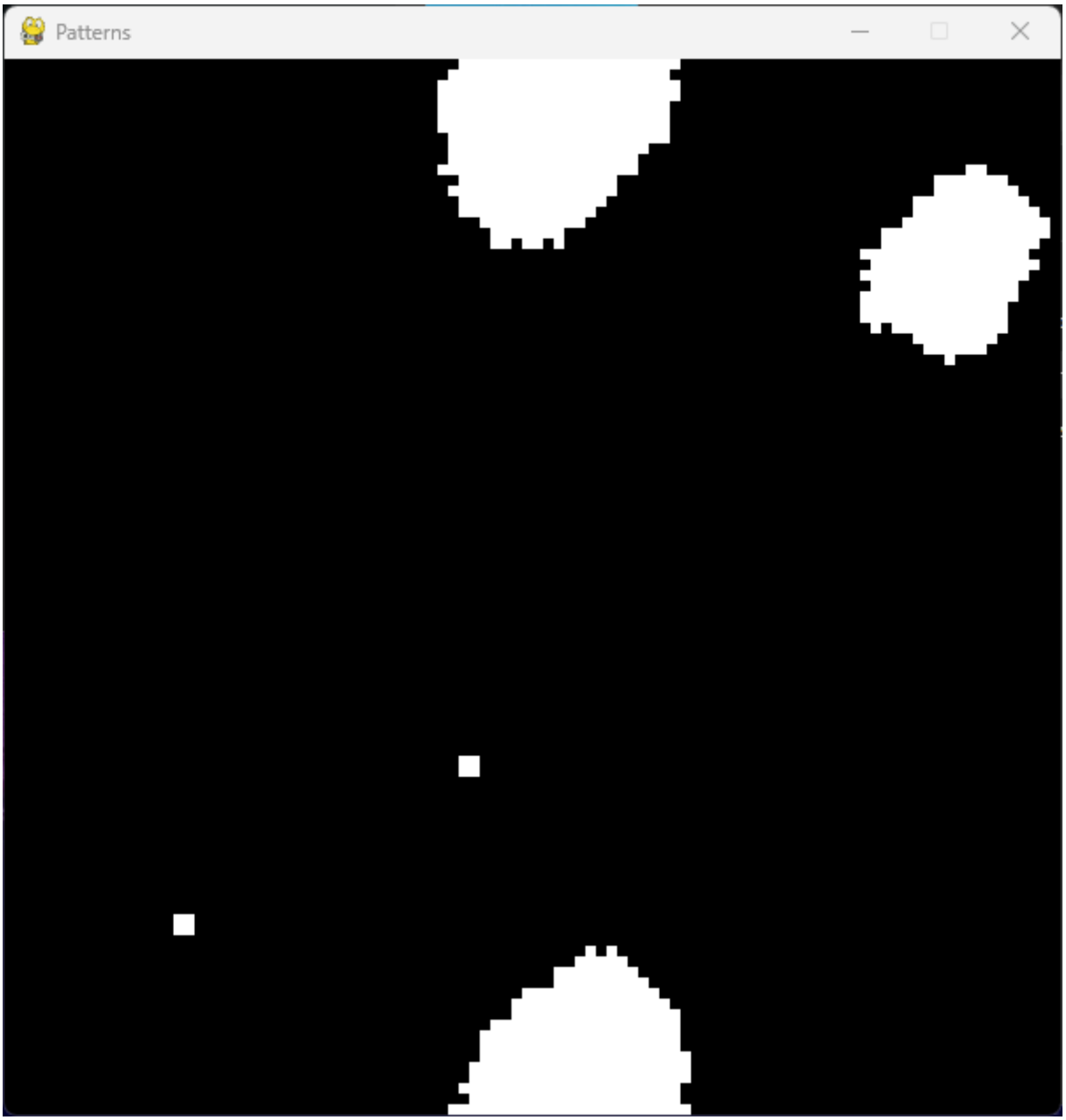
Jak możemy zauważyć dla tego przypadku, plamy zaczęły się formować z czarnych, martwych komórek, co skutkowało wzrostem gęstości wraz ze wzrostem iteracji, który byłby w takiej sytuacji oczekiwanym wynikiem.

Próba nr. 2

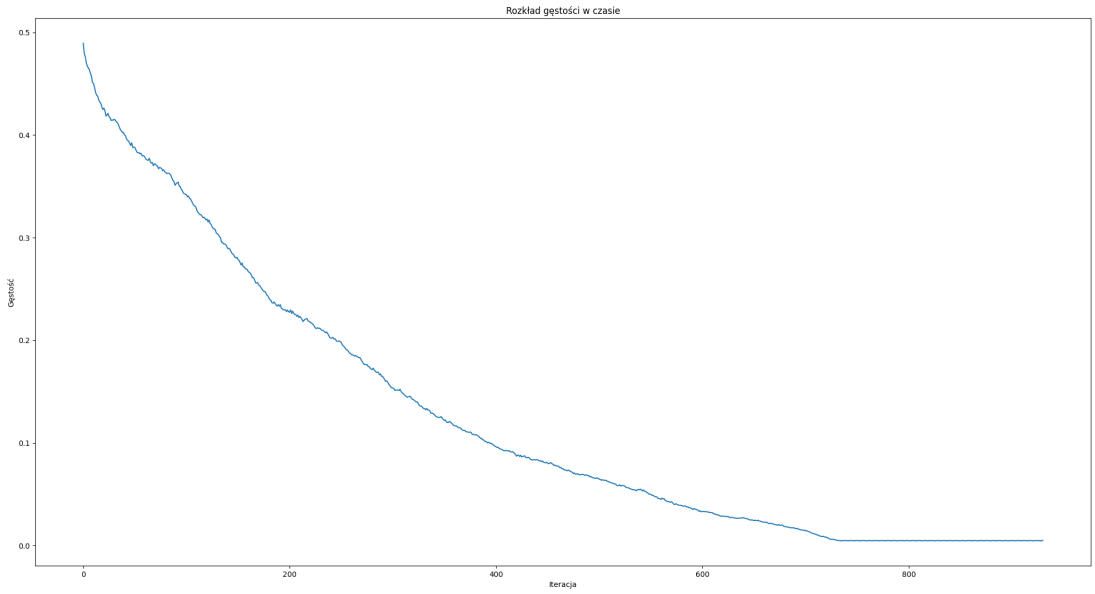
Snapshoty układu


Patterns


Patterns



Rozkład gęstości tego układu



Tutaj mamy analogiczną sytuację dla drugiego, skrajnego przypadku, gdzie tym razem plamy zaczęły się formować z komórek żywych, co skutkowało stopniowym zmniejszaniem się gęstości układu (komórek żywych). W takim wypadku mozemy stwierdzić, że kod działa prawidłowo, wedle przyjętych założeń i w pełni losowo, co za każdym razem może stworzyć nam inne, ciekawe struktury.

Wnioski

Podsumowując, udało się uzyskać korzystne wyniki i potwierdzić działanie programu, jak i przydatność wykorzystania automatów komórkowych do symulacji cieakwych układów. Warto zaznaczyć, że wyniki z obu prób podkreślają znaczący wpływ początkowego rozkładu aktywnych i nieaktywnych komórek na dalszą ewolucję układu. To może prowadzić do wniosku, że w systemach opartych na automatach komórkowych, takich jak ten, warunki początkowe mają kluczowe znaczenie dla przewidywania długoterminowego zachowania systemu.