Sprawozdanie z listy 1 - Automat komórkowy

Mateusz Wojteczek

Wstęp

Celem tej pracy było zaprojektowanie i stworzenie programu komputerowego symulującego Game of Life wedle Conway'a, która jest szczególnym przypadkiem automatu komórkowego. Gra w Życie działa na zasadzie prostej symulacji, w której każda komórka na dwuwymiarowej siatce może znajdować się w jednym z dwóch stanów: aktywnym (żywa) lub nieaktywnym (martwa) Stan komórki w następnym kroku czasowym zależy od liczby jej aktywnych sąsiadów. Celem symulacji jest obserwacja ewolucji układu początkowego, który jest generowany losowo z określonym prawdopodobieństwem początkowej aktywacji komórek p0.

Kod programu

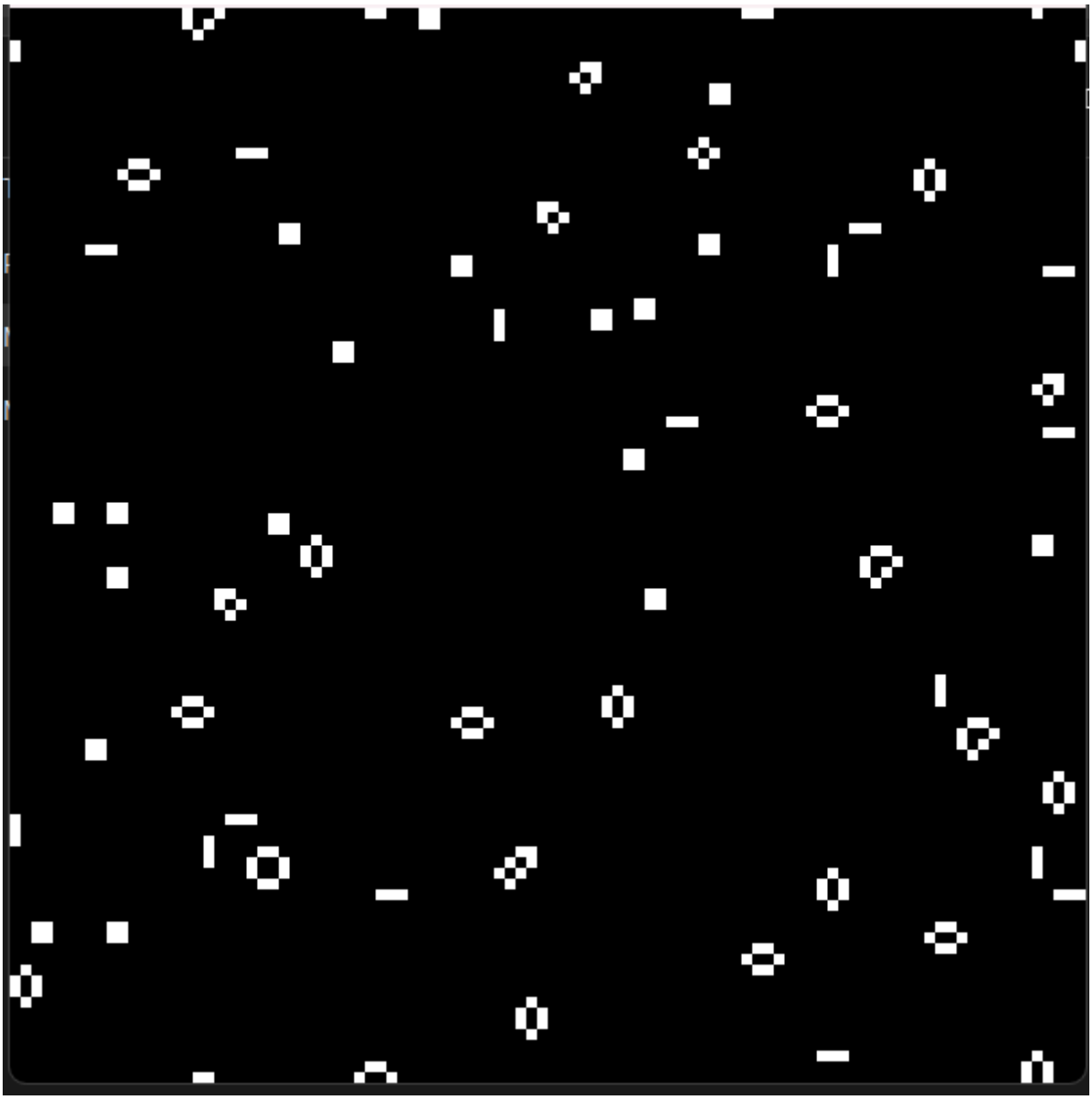
```
import pygame
import numpy as np
import matplotlib.pyplot as plt

# 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("Game of Life")

def create_grid(p0=0.1):
    """Tworzenie nowej siatki z losowymi stanami początkowymi oraz z zadanym
prawdopodobieństwem p0."""
    return np.random.choice([0, 1], size=(rows, cols), p=[1-p0, p0])


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):
    """Aktualizacja siatki na podstawie reguł automatu komórkowego."""
    new_grid = grid.copy()  # Tworzenie kopii bieżącej siatki do przechowywania
nowych stanów
    for row in range(rows):
        for col in range(cols):
            # Obliczanie całkowitej liczby aktywnych sąsiadów wokół bieżącej
komórki (bez aktualnej komórki)
            total = sum([grid[(row + i) % rows][(col + j) % cols] for i in
range(-1, 2) for j in range(-1, 2)]) - grid[row][col]

            # Stosowanie reguł automatu komórkowego do określenia następnego stanu
komórki
            if grid[row][col] == 1:  # Jeśli komórka jest obecnie aktywna
                # Komórka staje się nieaktywna, jeśli ma mniej niż 2 lub więcej
niż 3 aktywnych sąsiadów (przeludnienie lub osamotnienie)
                if total < 2 or total > 3:
                    new_grid[row][col] = 0
            else:
                # Komórka staje się aktywna, jeśli ma dokładnie 3 aktywnych
sąsiadów (rozmnażanie)
                if total == 3:
                    new_grid[row][col] = 1

    return new_grid

def main():
    current_grid = create_grid(0.5)  # Inicjowanie siatki losowymi stanami
    prev_grid = np.zeros((rows, cols))  # Inicjowanie poprzedniego stanu siatki
dla porównania (początkowo same zera)
    running = True

    # 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', fontsize = 16)  #
Resetowanie tytułu (usuniętego przez ax.clear())
            ax.set_xlabel('Iteracja', fontsize = 12)  # Resetowanie etykiety osi X
            ax.set_ylabel('Gęstość', fontsize = 12)  # 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()
```

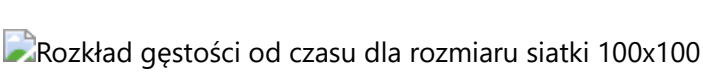
Analiza wyników

Rozkład gęstości od czasu dla p0 równego 10%



W przypadku podjęcia próby analizy wykresu sporządzonego dla rozmiaru siatki 60 x 60 możemy zauważyć, że przede wszystkim rozkład ten jest dość chaotyczny, występują nagłe spadki ilości, a następnie rozmnażania komórek, jednakże wedle oczekiwań stopniowo maleje aż do całkowitego wygaszenia i unormowania się przy około 1700 iteracjach. Zachowanie to było jak najbardziej oczekiwane, przy przyjętych założeniach automatu komórkowego.

Rozkład gęstości od czasu dla rozmiaru siatki 100x100



Natomiast w tym przypadku, mimo nadal występującej tendencji spadkowej, widzimy, że układ ten jest znacznie mniej chaotyczny, nie zauważymy tu nagłych spadków i wzrostów jak w poprzednim wykresie, jest zdecydowanie bardziej unormowany. Jednakże całkowite wygaszenie i unormowanie występuje dopiero po ponad 2000 iteracjach, także potrzebuje on więcej czasu dla takiego rozmiaru siatki.

Wnioski

Praca ta pokazuje, że automaty komórkowe mogą być użytecznym narzędziem do modelowania złożonych systemów dynamicznych, takich jak ewolucja plam. Układ zachowywał się wedle oczekiwań i przyjętych

założeń, co pokazują wykresy oraz nagrania ewolucji graficznej układu, które umieściłem w folderze ze sprawozdaniem.