The Second Project
CA (Cellular Automata)

Smart Systems

Faculty of Computer and Data Sciences,
Alexandria University.

Authored by: Ahmed Ehab Abdelhalem
ID:2203147
Authored by: Youssef Mohamed Helmy
ID:2203163

Introduction

Our Project simulates a simple ecosystem with different types of organisms, represented by cells in a grid. The organisms include Alpha, Beta, and Gamma, each with specific characteristics and behaviors. They tried to survive by food . The simulation is based on a set of rules governing the interactions among these organisms, and it visualizes the evolution of the ecosystem over multiple generations using Matplotlib library.

The illustration of Code

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def initialize_population(size):
    population = np.random.choice([0, 1, 2, 3, 4], size=(size, size), p=[0.4, 0.3, 0.15, 0.1, 0.05])
    Alpha_age = np.zeros_like(population)
    Beta_age = np.zeros_like(population)
    Gamma_age = np.zeros_like(population)
    return population, Alpha_age, Beta_age, Gamma_age

population, Alpha_age, Beta_age, Gamma_age = initialize_population(100)
```

First , We Start with importing some of essential libraries like Numpy and Matplotlib. Then we Create a function that initialize a random population from each organism with certain size ,so we assign each cell with certain value :- Empty → 0 , Food →1 , Gamma →2 , Beta → 3, Alpha →4 with probabilities (0.4, 0.3, 0.15, 0.1, 0.05) respectively . And we create an age for each organism (Gamma, Beta, Alpha) with age = Zero as default

```python
def count_neighbors(population, x, y, value):
    count = 0
    for i in range(-1, 2):
        for j in range(-1, 2):
            if (i, j) != (0, 0) and 0 <= x + i < population.shape[0] and 0 <= y + j < population.shape[1] and population[x + i, y + j] == value:
                count += 1
    return count

```

Then , we create a Function that calculate the number of neighboring cells around a specified cell . The function uses two nested loops to iterate over a 3x3 neighborhood centered around the specified cell .

1-The condition (i, j) != (0, 0) ensures that the central cell itself is not included in the count.

2-The condition 0 <= x + i < population.shape[0] and 0 <= y + j < population.shape[1] checks if the neighboring cell is within the boundaries of the grid to avoid index out-of-bounds errors.

3- If the conditions are met, the function checks if the value of the neighboring cell (population[x + i, y + j]) is equal to the specified value. If true, it increases the count.

```python
    for i in range(population.shape[0]):
        for j in range(population.shape[1]):
            neighbors_2 = count_neighbors(population, i, j, 2)
            neighbors_3 = count_neighbors(population, i, j, 3)
            neighbors_4 = count_neighbors(population, i, j, 4)

            if population[i, j] == 0:  # Empty cell
                if np.random.rand() <= 0.25:  # Generate food in random empty cells with probability 25%
                    new_population[i, j] = 1

            elif population[i, j] == 1: # Food
                if neighbors_2 >= 1 and neighbors_3 >= 1 and neighbors_4 >= 1: #if food have neighbors of all types, alpha will eat food
                    new_population[i, j] = 4
                elif neighbors_2 == 0 and neighbors_3 == 0 and neighbors_4 >= 1: #if food have neighbors of only alpha, alpha will eat food
                    new_population[i, j] = 4
                elif neighbors_2 == 0 and neighbors_3 >= 1 and neighbors_4 >= 1: #if food have neighbors of alpha and beta, alpha will eat food
                    new_population[i, j] = 4
                elif neighbors_2 >= 1 and neighbors_3 == 0 and neighbors_4 >= 1: #if food have neighbors of alpha and gamma, alpha will eat food
                    new_population[i, j] = 4
                elif neighbors_2 >= 1 and neighbors_3 >= 1 and neighbors_4 == 0: #if food have neighbors of beta and gamma, beta will eat food
                    new_population[i, j] = 3
                elif neighbors_2 == 0 and neighbors_3 >= 1 and neighbors_4 == 0: #if food have neighbors of only beta, beta will eat food
                    new_population[i, j] = 3
                elif neighbors_2 >= 1 and neighbors_3 == 0 and neighbors_4 == 0: #if food have neighbors of only gamma, gamma will eat food
                    new_population[i, j] = 2
                else:
                    new_population[i, j] = 0

            elif population[i, j] == 4:  # Alpha
                new_Alpha_age[i, j] += 1

                if new_Alpha_age[i, j] >= 4 or neighbors_4 == 8:  #alpha dies after reaching age 4 or from over population
                    new_population[i, j] = 0
                    new_Alpha_age[i, j] = 0

            elif population[i, j] == 3:  # Beta
                new_Beta_age[i, j] += 1

                if new_Beta_age[i, j] >= 24 or neighbors_3 == 8:  #beta dies after reaching age 24 or from over population
                    new_population[i, j] = 0
                    new_Beta_age[i, j] = 0

            elif population[i, j] == 2:  # Gamma
                new_Gamma_age[i, j] += 1

                if new_Gamma_age[i, j] >= 48 or neighbors_2 == 8:  #gamma dies after reaching age 48 or from over population
                    new_population[i, j] = 0
                    new_Gamma_age[i, j] = 0

    return new_population, new_Alpha_age, new_Beta_age, new_Gamma_age
```

Then we create a function that determines the rules for each organism . The rules governing the evolution of the population are as follows:-

Empty cells (state 0) may randomly become food cells (state 1) with a probability of 25%.

Food cells may be consumed by individuals of type Alpha (state 4) under various conditions, depending on the types of neighbors. If certain conditions are met, the food cell is replaced by an Alpha cell.

Individuals of type Alpha, Beta, and Gamma age over time. If their age exceeds a certain threshold or if they are surrounded by a certain number of neighbors, they die and are replaced by an empty cell.

Alpha individuals (state 4) die after reaching an age of 4 or if surrounded by 8 neighbors.

Beta individuals (state 3) die after reaching an age of 24 or if surrounded by 8 neighbors.

Gamma individuals (state 2) die after reaching an age of 48 or if surrounded by 8 neighbors.

```python
generations = 101
colors = ['white', 'gold', 'green', 'blue', 'red']
cmap = plt.cm.colors.ListedColormap(colors)

plt.imshow(population, cmap=cmap, interpolation='nearest')
plt.grid(True, color='black', linewidth=0.5)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Generation 0')
plt.colorbar(ticks=[0, 1, 2, 3, 4], label='Cell State (0: Empty, 1: Food, 2: Gamma, 3: Beta, 4: Alpha)')
plt.gca().set_aspect('equal', adjustable='box')

```

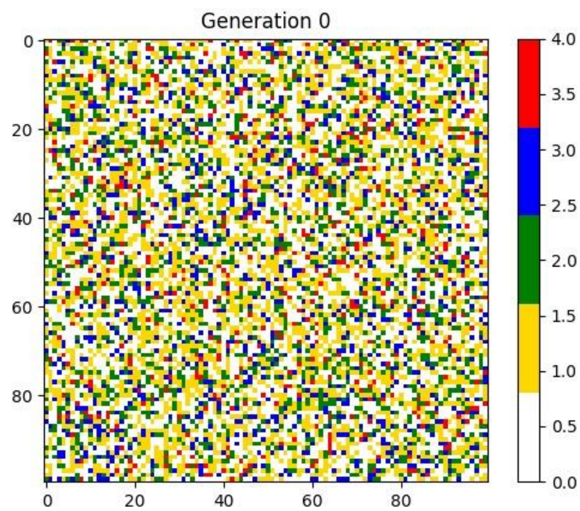
We initialize generations with 101 and create a list of colors for each cell state to use it in the colormap

```python
def update_plot(frame):
    global population, Alpha_age, Beta_age, Gamma_age
    population, Alpha_age, Beta_age, Gamma_age = apply_rules(population, Alpha_age, Beta_age, Gamma_age)
    plt.imshow(population, cmap=cmap, interpolation='nearest')
    plt.title(f'Generation {frame}')

animation = FuncAnimation(plt.gcf(), update_plot, frames=generations, repeat=False, interval=100)
plt.show()

```

The update_plot function is repeatedly called for each generation, showing how the population evolves over time. The color of each cell represents its state, and the title is updated to indicate the current generation.

The output

Generation Zero:



We can observe that the cells is distributed by our selected probabilities where :-

Empty→40%

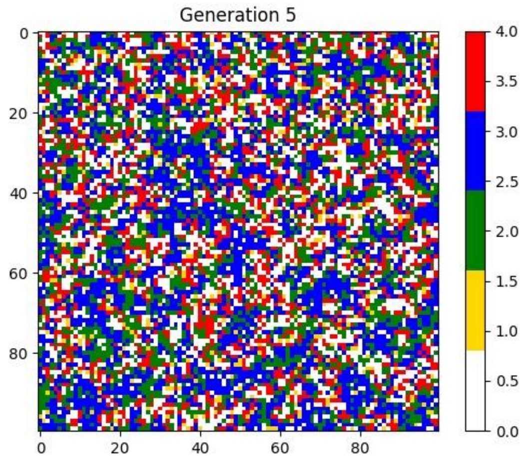
Food→30%

Gamma→15%

Beta→10%

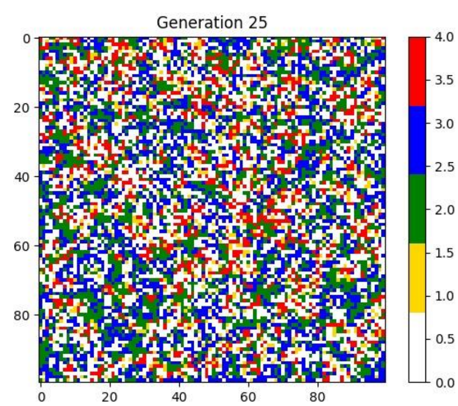
Alpha→5%

Generation Five:



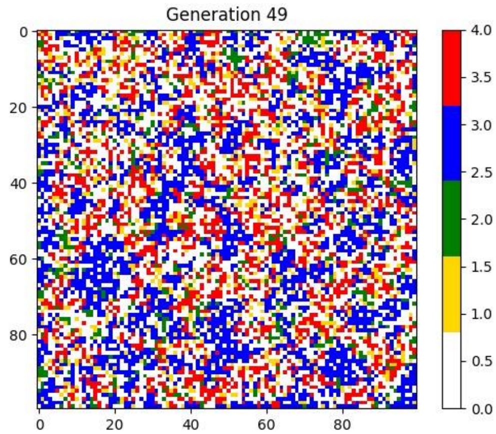
We can observe that Beta has grown very fast because she has the second biggest initial probability after gamma, and she can obtain food from gamma according to our rules

Generation 25:



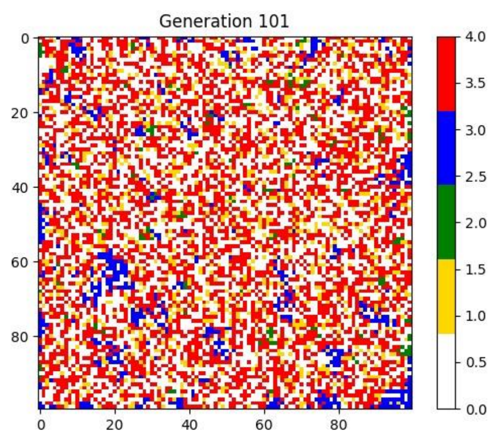
We can observe that beta has been decreased because the alpha has been grown (alpha is considered the strongest organism according to the rules)

Generation 49 :-



We can observe that the beta has been decreasing and the alpha has been increasing.
Gamma has been decreasing (The weakest organism) according to the rules

Generation 101:-



We can observe that the alpha has been dominating over the system as we predict because she has the strongest rules , we also observe that Gamma has been absolutely decreasing