

Aim

To explore the dynamics and behaviors of various cellular automata rules through simulations and visualizations using Python.

Objectives

1. Implement and visualize different 1D and 2D cellular automata rules.
2. Create animations to observe the evolution of the cellular automata over time.
3. Compare the behaviors of different rules and configurations.

Summary

This project involves implementing and simulating various cellular automata rules using the cellpylib library. The simulations include both 1D and 2D cellular automata, applying classic rules such as Rule 110, 30, and Conway's Game of Life. The results are visualized using Matplotlib, with animations to showcase the dynamic evolution of the automata.

Tools and Libraries Used

- **Python:** Programming language.
- **NumPy:** For array and numerical operations.
- **cellpylib:** For cellular automata simulation.
- **Matplotlib:** For plotting and animation.

Procedure

Code 1 & Code 2: 1D Cellular Automata with Rule 110 and Rule 30

CODE:

```
import numpy as np
```

```
import cellpylib as cpl
```

```
import matplotlib.pyplot as plt
```

```
import matplotlib.animation as animation
```

```
# Initialize the cellular automaton
```

```
cell = cpl.init_simple(100)
```

```
# Evolve the cellular automaton using Rule 110 (or Rule 30 for Code 2)
```

```
cell = cpl.evolve(cell, timesteps=30, memoize=True, apply_rule=lambda n, c, t: cpl.nks_rule(n, rule=110))
# Change rule to 30 for Code 2
```

```
# Plot the cellular automaton
```

```
cpl.plot(cell)
```

```
# Setup animation
```

```
fig, ax = plt.subplots()
```

```
mat = ax.matshow(cell, cmap='binary')
```

```
plt.axis('off')
```

```
def animate(i):
```

```
    mat.set_data(cell[:i+1])
```

```
    return [mat]
```

```
ani = animation.FuncAnimation(fig, animate, frames=50, interval=50, blit=True, repeat=True)
```

```
plt.show()
```

Code 3 & Code 4: 2D Cellular Automata with Conway's Game of Life Rule

CODE:

```
import numpy as np
```

```
import cellpylib as cpl
```

```
import matplotlib.pyplot as plt
```

```
import matplotlib.animation as animation
```

```
# Initialize the 2D cellular automaton
```

```
cellular_automaton = cpl.init_simple2d(rows=60, cols=60)
```

```
# (For Code 4) Set initial states to form specific patterns
```

```
cellular_automaton[:, [28, 29, 30, 30], [30, 31, 29, 31]] = 1
```

```

cellular_automaton[:, [40, 40, 40], [15, 16, 17]] = 1
cellular_automaton[:, [18, 18, 19, 20, 21, 21, 21, 21, 20], [45, 48, 44, 44, 44, 45, 46, 47, 48]] = 1

# Evolve the cellular automaton
cellular_automaton = cpl.evolve2d(cellular_automaton, timesteps=250, neighbourhood='Moore',
    apply_rule=cpl.game_of_life_rule, memoize='recursive')

# Setup animation
fig, ax = plt.subplots()
ax.set_xlim((0,60))
ax.set_ylim((0,60))
img = ax.imshow(cellular_automaton[0])

def init():
    img.set_data(cellular_automaton[0])
    return (img,)

def animate(i):
    img.set_data(cellular_automaton[i])
    return (img,)

ani = animation.FuncAnimation(fig, animate, frames=250, interval=50, blit=True, repeat=True)
plt.show()

```

Code 5: 2D Cellular Automata with Totalistic Rule 126

CODE:

```

import numpy as np
import cellpylib as cpl
import matplotlib.pyplot as plt
import matplotlib.animation as animation

```

```
# Initialize the 2D cellular automaton
```

```
cell = cpl.init_simple2d(rows=60, cols=60)
```

```
# Evolve the cellular automaton using Totalistic Rule 126
```

```
cell = cpl.evolve2d(cell, timesteps=30, neighbourhood='Moore', apply_rule=lambda n, c, t:  
cpl.totalistic_rule(n, k=2, rule=126))
```

```
# Animate the cellular automaton
```

```
cpl.plot2d_animate(cell)
```

Highlights

- **Diverse Rules and Configurations:** The project demonstrates various rules (Rule 110, Rule 30, Game of Life, Totalistic Rule 126) and both 1D and 2D cellular automata.
- **Animations:** Matplotlib's FuncAnimation is used to create smooth animations, allowing for an engaging visualization of cellular automata evolution.
- **Custom Initial States:** Code 4 includes custom initial states to form specific patterns, showcasing how initial configurations affect the automaton's behavior.

Conclusion

This project provides a comprehensive overview of cellular automata simulations, illustrating the wide variety of behaviors that can emerge from simple rules and initial conditions. Through visualization and animation, the dynamic and often complex nature of cellular automata is effectively showcased, offering insights into their applications and theoretical significance.