Key concepts

- Cellular automata (CA) are the most basic objects in a given grid/foundation that serve as variable states which can be used to model physical systems. An automaton is governed by rules and its neighboring automata which determine how it evolves or changes through time.
- The 256 Wolfram 1D-CA models show how some cellular automata rulesets can achieve complexity (class 4) which is commonly seen in nature. This allows us to apply such models to physical systems and do predictions accurate to as much as determinism allows.
- Agent based models are similar to CAs but employ a relatively more complex set of conditions. Rules are set as agent-agent and agent-environment interactions which focuses on an agents' decision making to achieve a certain goal. These allow us to model natural or real-life occurrences and make predictions or even identify emergent properties, if there are any.

Essential code snippets

The initialization section of the code allows us to visualize what our grid looks like and how the generations will stack upon one another. Seeing the trend in how each row of the array evolves can then be used to map out the patterns using graphing.

```
rows = 5
cols = 11
array = np.zeros((rows, cols), dtype=np.uint8)
array[0, 5] = 1
print(array)
```

```
[[ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

```
def step(array, i):
    rows, cols = array.shape
    row = array[i-1]
    for j in range(1, cols):
        elts = row[j-1:j+2]
        array[i, j] = sum(elts) % 2
```

Applying cellular automata to model physical systems requires the pre-set rules for the specific system which determine how each automaton will evolve through each time step. Code that decides this is at the heart of such applications. They allow us to see patterns or properties which could arise from the given system.

```
class Agent:

    def step(self, env):
        self.loc = env.look_and_move(self.loc, self.vision)
        self.sugar += env.harvest(self.loc) - self.metabolism
        self.age += 1
```

For agent-based models, defining what the agents does per time step is important in understanding patterns that arise in future generations. Such is seen with this part of the Sugarscape model which explains why better-fit agents are able to survive and harvest sugar more efficiently. Schelling's model is similarly predictable. However, an additional emergent property of an unnecessarily high degree of segregation is found.

Pitfalls

- Dealing with randomness and properly defining rules that will lead to a desirable CA pattern.
- Having to address emergent properties and whether or not the rules should be modified to better fit the situation being modeled.
- Determinism and arguing semantics of what predictions are valid or acceptable.