Game of Life Project

By Malaika Awoko

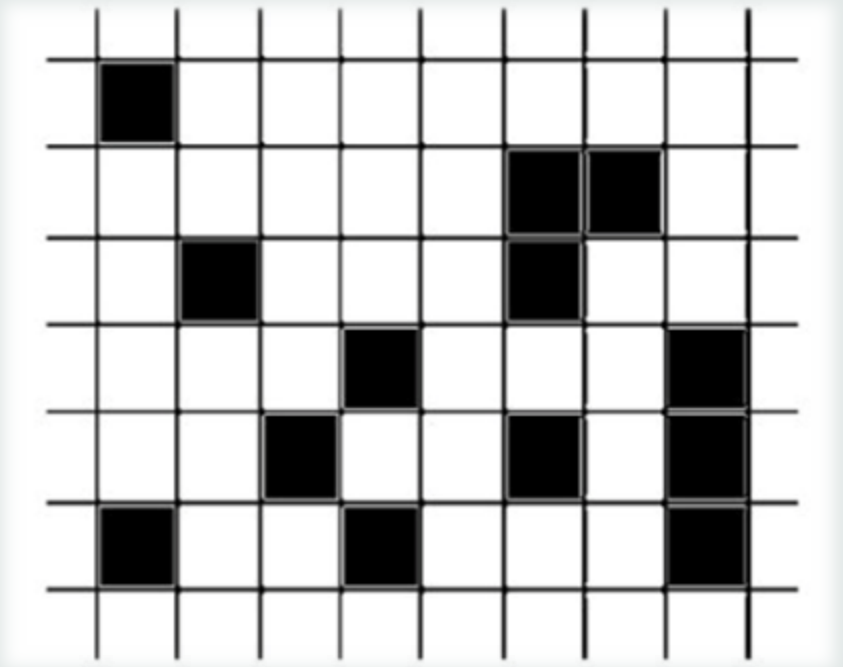
Antanya Lindsay, and Abigail Graham

Project Goals

- Original/ Unique
  - Different color and shape
  - Create new and future generations
  - Count population

What the code looks like

- Square grid
- Shows current generation of the population
- Each square is either
  - Alive or dead



Cell Class

```
class Cell:
    def __init__(self,state):
        self.state = state
        self._future_state = state

    @property
    def state(self):
        return self._state

    @state.setter
    def state(self, new_state):
        self._state = new_state

    def switch_state(self):
        self._state = not self._state

    def update_future_state(self, new_state):
        self._future_state = new_state

    def next_gen(self):
        self._state = self._future_state

    def __str__(self):
        if self._state:
            return "*"
        else:
            return " "
```

Game class

```
class Cell:
    def __init__(self,state):
        self.state = state
        self._future_state = state

    @property
    def state(self):
        return self._state

    @state.setter
    def state(self, new_state):
        self._state = new_state

    def switch_state(self):
        self._state = not self._state

    def update_future_state(self, new_state):
        self._future_state = new_state

    def next_gen(self):
        self._state = self._future_state

    def __str__(self):
        if self._state:
            return "*"
        else:
            return " "
```

Board Class

```
class Board:
    def __init__(self, height, width):
        self._matrix = [
            [Cell(state=False) for _ in range(width)] for _ in range(height)
            ]

    def __str__(self):
        row_repr = ""
        for row in self._matrix:
            row_repr += "".join([str(cell) for cell in row] + ["\n"])
        return row_repr

    def get_cell(self, y, x):
        return self._matrix[y][x]

    def get_cell_state(self, y, x):
       return self._matrix[y][x].state

    def set_cell_state(self, y, x, new_state):
        self._matrix[y][x].state = new_state
```

How it works?

- Each cell can either be dead or alive
- The status of each cell changes how the game will play out
- Each cell's status depends on the status of its 8 neighbors

Rules as the different cell shapes

- We did three different cell shapes
- Oscillators
- Gliders
- Blinkers