CMPT115 - Final Project

In this project, you will create a Python Program that will play Conway's Game of Life.

Game of Life Description

John Horton Conway, a British mathematician, created the cellular automaton known as The Game of Life in 1970. It is the most popular illustration of a cell machine. The "game" is in fact a zero-player game, which means that its development is determined by its initial state and does not require input from humans. The Game of Life can be played by setting up a starting configuration and watching it change over time.

The Game of Life universe consists of an infinite orthogonal grid of square cells in two dimensions. The cells in Conway's Game of Life can be either `alive` or `dead`. To represent whether a cell is `alive` or `dead`, you will use `0` for `dead` and `1` for `alive`.

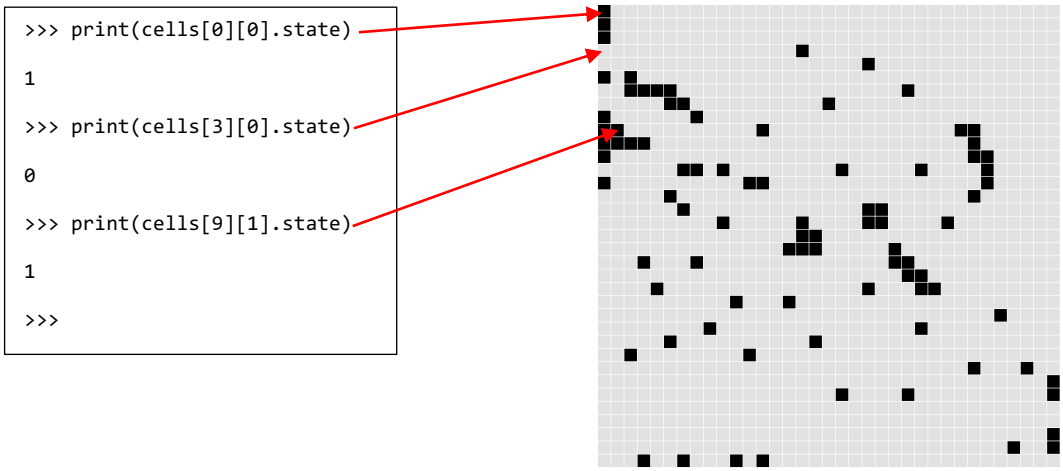
For more information see https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

In this project, you will create a `35 x 35` grid of cells by making a 2-dimentional Python `list` named `cells`. Each element of the `list` which represent one cell is a Python Turtle. The initial state for each cell must be either `0` or `1` (see Figure 1). The goal is to simulate the evolution of the system over time by applying the rules of Conway's Game of Life and to display the state of the game at each time step.

The Grid:

```
wn = turtle.Screen()
wn.setup(width = 35*20 + 100, height = 35*20 + 100)
wn.tracer(0)
```

Figure 1: Example of a universe



Note. For each cell (Turtle), if the state of the cell is `0`, we set its color to `"gray90"` for the color **gray** and if the state of the cell is `1`, we set its color to `"gray0"` for the color **black**.

```
def initializeTheCells():
    for i in range(35):
        cells.append([])
        for j in range(35):
            newCell = turtle.Turtle()
            newCell.penup()
            newCell.shape("square")
            newCell.shapesize(stretch_wid = 0.9, stretch_len = 0.9)
            cells[i].append(newCell)
            newCell.state = 0
            newCell.color("gray90") #gray90 almost white


def selectCells(x, y):
    if onClick:
        if x > -350 and x < 350 and y > -350 and y < 350:
            j = math.floor((x + 350)/20)
            i = math.floor((350 - y)/20)
            if cells[i][j].state == 0:
                cells[i][j].state = 1
                cells[i][j].color("gray0") #black
            else:
                cells[i][j].state = 0
                cells[i][j].color("gray90") #gray90 almost white
    wn.update()
```

Rules of The Game

Every cell interacts with its eight neighbors, which are the cells that are horizontally, vertically, or diagonally adjacent to it (see Figure 2). Any given cell in cells can be accessed through its row `i` and column `j` numbers by `cells[i][j]` (see figure 1).

```
>>>print(type(cells[0][0]))
<class 'turtle.Turtle'>
```

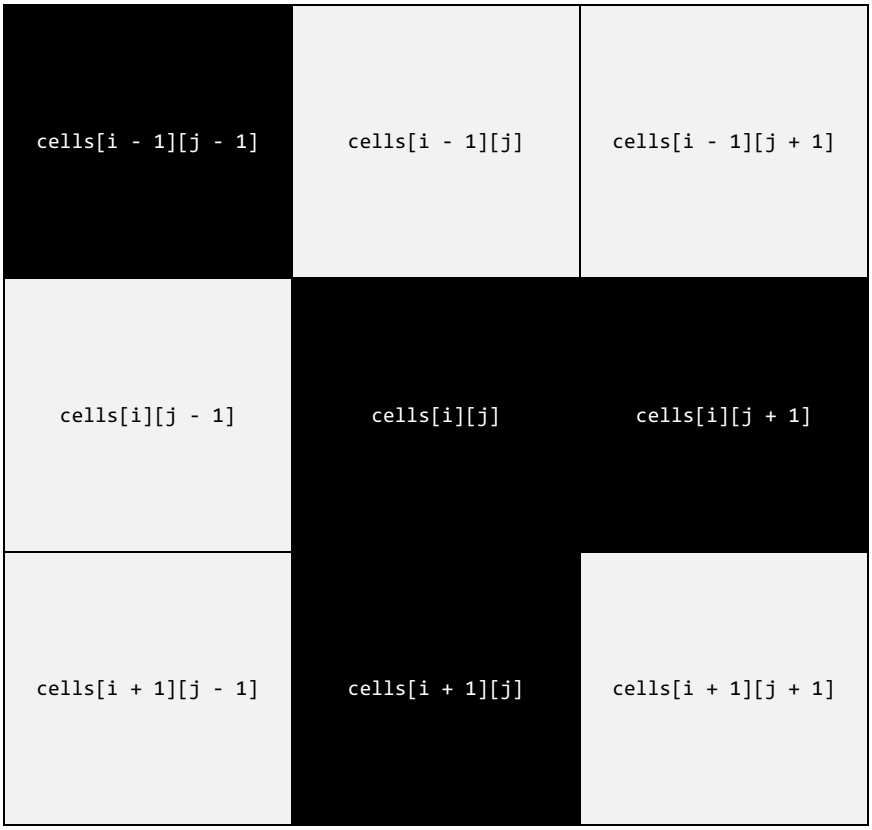
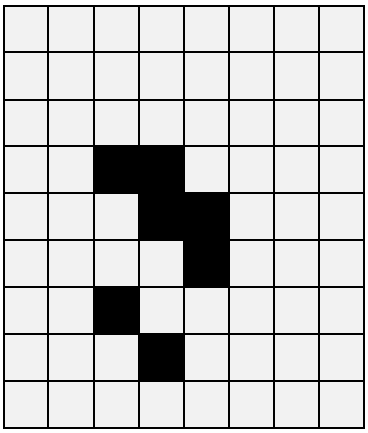


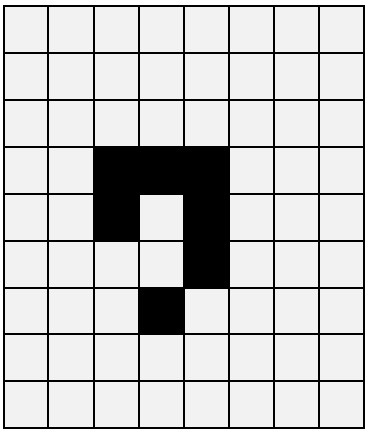
Figure 2: Eight neighboring cells to a cell at row `i` and column `j`.

The following shifts take place at each time step (see Figure 3):

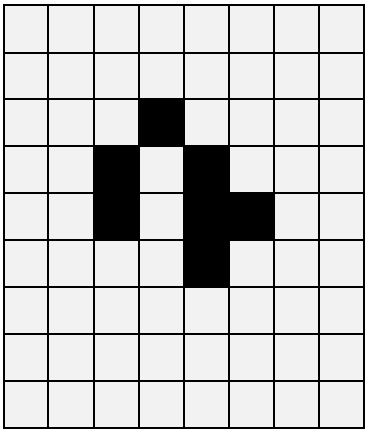
- If a state of a cell is `1` and has fewer than two neighbors that have states `1`, it changes to `0`.
- If a state of a cell is `1` and has either two or three neighbors that have states `1`, it remains `1`.
- If a state of a cell is `1` and has more than three neighbors that have states `1`, it changes to `0`.
- If a state of a cell is `0` and has exactly three neighbors that have states `1`, it changes to `1`.



Time step 1



Time step 2



Time step 3

Figure 3: An example showing the cells change from one generation to the next.

Boundary Conditions

How to deal with cells at the edge of the grid? Which cells are their neighbors? The answers to this question are what are known as **Boundary Conditions**. In this project we are going to look at two of them: **The Constant Boundary Condition** and **Periodic Boundary Condition**.

Constant Boundary Condition:

We consider all the cells adjacent to those on the edge of the grid are considered `dead` (see Figure 4). You can assume there are imaginary cells surrounding the grid where all of them are `0`.

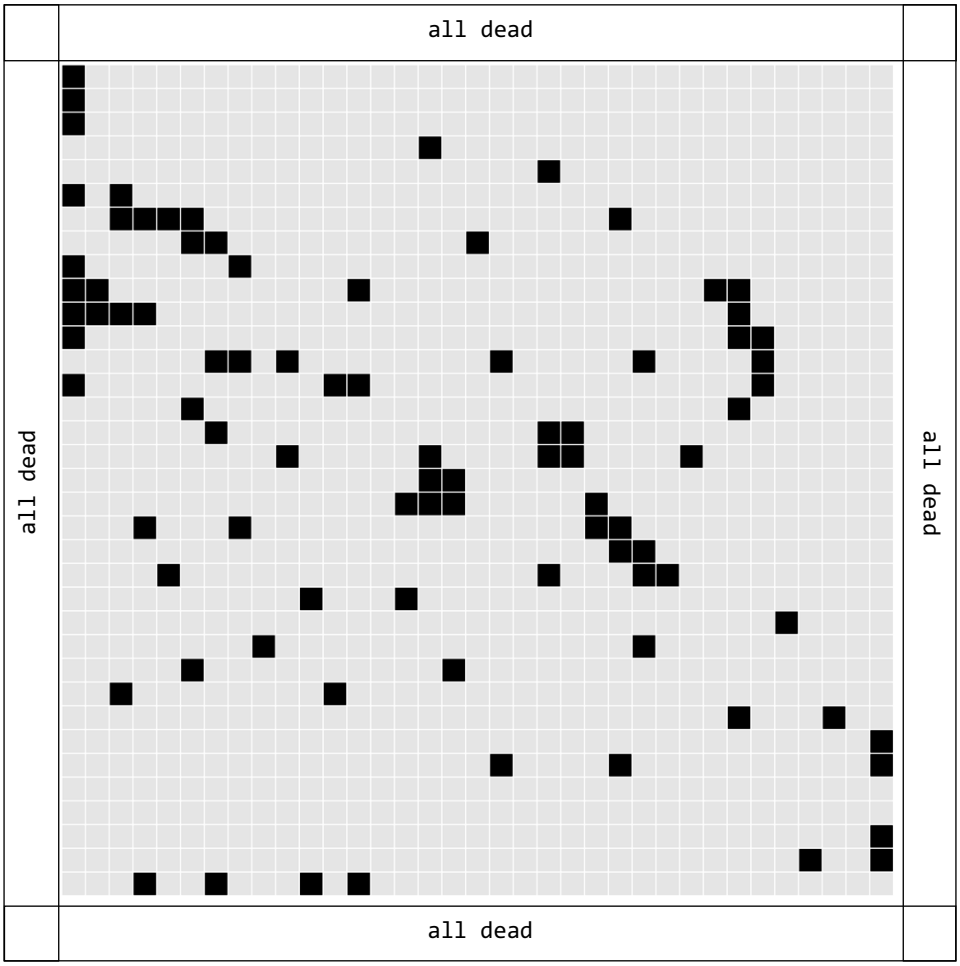


Figure 4.

Periodic Boundary Condition:

We are going to assume that the neighbors to a cell at the edge of the grid are those cells at the opposite edge of the grid (The cells on the left edge of the grid are considered to be **adjacent** to the cell on the right edge and the cells on the top edge of the grid are considered to be **adjacent** to the bottom edge of the grid. (see Figure 5)).

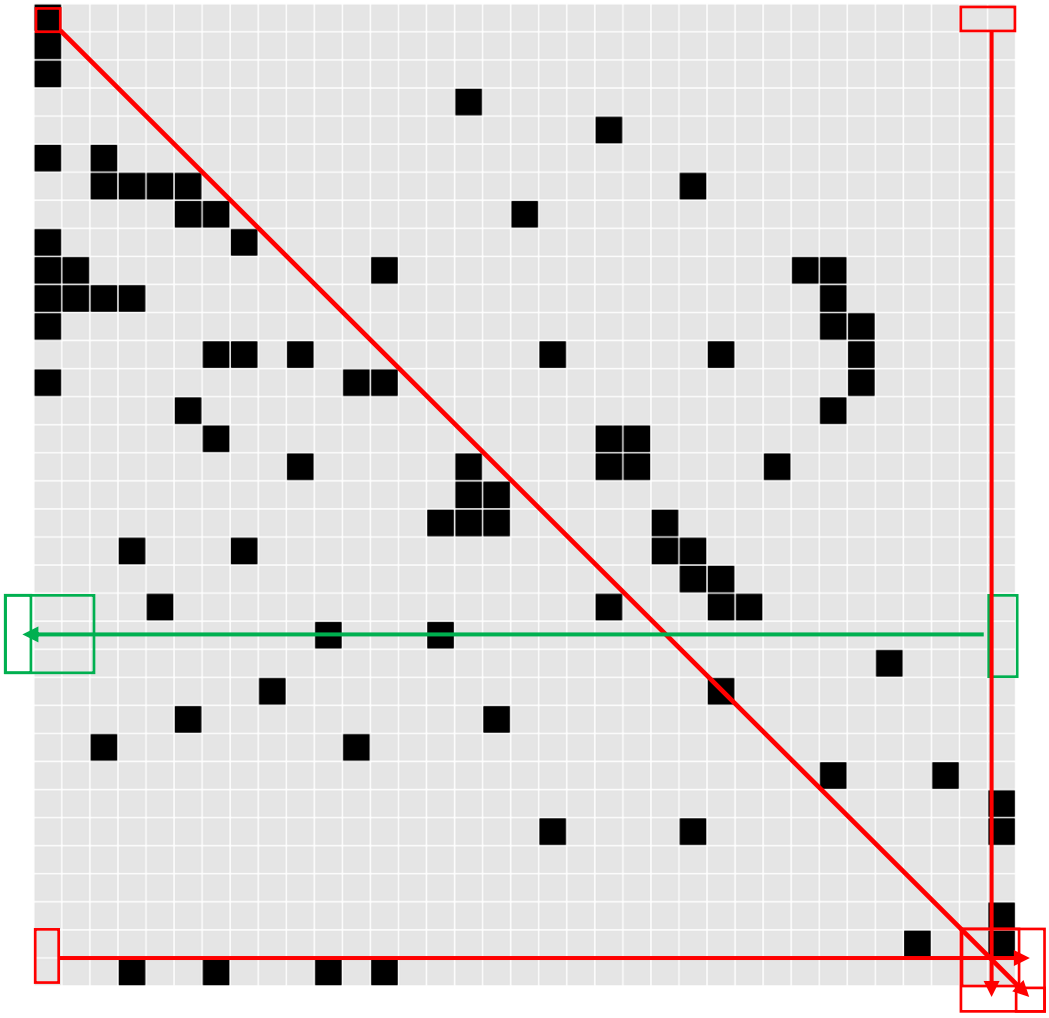


Figure 5: Three examples showing the periodic boundary condition.

The Python Program

I have included a starter Python program for this project on Moodle. You **must** use the provided starter code to complete the project.

Algorithm:

1. Initialize the cells in the grid to all dead.
2. Using mouse click(s) to change cells' states (alive to dead or dead to alive)
3. Ask the user to choose a boundary condition (enter 1 for *constant* and 2 for *periodic*)
4. At each iteration (time step), the next generation of cells is calculated for each cell at row `i` and column `j`, in two passes:

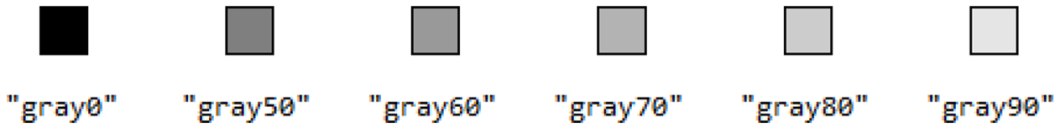
   Pass 1: For each cell record its number of live neighbors by considering the boundary condition.

   Pass 2: Update each cell to alive/dead based on the rules.

Bonus for Extra Credit

You do not need to complete the bonus section, however students who manage to successfully complete the bonus section will receive extra credit.

We can add more colors into the game by using the following color palette.



Rules on how to use the color palette:

1. Only use the color `"gray0"` to indicate the cell is **alive** and all other shades of gray indicate a **dead** cell.
2. If a living cell dies, its color is set to `"gray50"` and as the game progresses the color gradually fades out to `"gray90"`, provided that the cell remains dead.

Submitting Your Project

You will find a submission link on the course Moodle page under "**Final Project**". You are required to submit your project as one `your_full_name.py` file that contains the given starter program and **all** the required custom functions.

Submission Due Date and Time

The due date and time to submit your project is Monday April 8th, 2024, at 4:59 **4M**. You are required to submit your project through Moodle before submission time expires. **If you don't submit your project through Moodle on time, then it will be considered as if you didn't attempt the final project in which case, you will get a letter grade N for the course. No email submission will be accepted for any reason.**