# *Advanced Topic in Computer Science I (420-G40-HR)*

# *Assignment 3 – Game of Life*

Date assigned: October 20, 2021

Date due: November 2, 2021

**Learning Objectives**

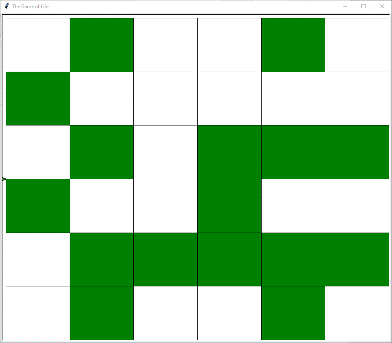
Upon successful completion of this assignment, the student will be able to:

* Model Conway’s Game of Life using Python libraries for displaying

**Conway’s Game of Life**

1. In 1970 the British Mathematician John Conway created his "Game of Life" – which is not really a game, but a set of rules that mimics the chaotic yet patterned growth of a colony of biological organisms. The "game" takes place on a two-dimensional grid consisting of "living" and "dead" cells, and the rules to step from generation to generation are simple:
   1. Overpopulation: if a living cell is surrounded by more than three living cells, it dies.
   2. Stasis: if a living cell is surrounded by two or three living cells, it survives.
   3. Underpopulation: if a living cell is surrounded by fewer than two living cells, it dies.
   4. Reproduction: if a dead cell is surrounded by exactly three cells, it becomes a live cell.

By enforcing these rules in sequential steps, beautiful and unexpected patterns can appear.

1. For this assignment is to create a program called *yourinitials*GameOfLife.py. This will model the game.
2. Clearly, no cells can come to life unless there are already some living cells in the universe. Thus, we have to provide the universe with a seed; i.e., a set of initial living cells. The massive variety and complexity of the results, often starting from simple seeds, is what makes cellular automata so interesting.
3. The rough algorithm is as follows:
   1. Prompt for the number of squares on a side of the grid (universe size).
   2. Create a display the (empty) grid using turtle (empty grid means background colour)
   3. Set up key(press) functions as follows:
      1. ‘r’ sets a random universe of points covering 40% of the universe;
      2. ‘f’ prompts for a file to be read containing the starting point of the universe (this may change the grid size);
      3. ‘s’ starts the ‘game of life’;
      4. ‘p’ pauses the ‘game of life’;
      5. ‘q’ quits the application (optionally can prompt to confirm).
      6. The user has the option of clicking (with the mouse) on specific squares to toggle that square on or off.
   4. If the user presses ‘r’:
      1. Determine how many total squares are in the universe;
      2. Generate a random number between 1 and the size of the universe and turn that square on;
      3. Loop until 40% of the squares have been turned on (Note: if you get the same random number twice, ignore it and do NOT increment the total count of squares);
      4. Display the (initialized) universe using turtle. Universe is ‘paused’.
   5. If the user presses ‘f’
      1. Prompt for the file name to read;
      2. Read the file (see below) and create the universe based on the file data (files will be provided);
      3. Display the (initialized) universe using turtle. Universe is ‘paused’.
   6. If the user clicks on a box, turn the box off if it is on, and on if it is off. Universe is ‘paused’.q
   7. If the user presses ‘q’ quit the application and end the program.
   8. If the user presses ‘s’ call to start the game of life. This works as follows:
      1. For each cell, determine if a cell survives to/dies/is born in the next iteration, based on its neighbours;
      2. NOTE: Survival is based on the grid at the start of the cycle and not on the values in the grid as the cycle progresses;
      3. Once ALL cells survival/death/birth is calculated, redisplay the new generation in the grid;
      4. Wait for some time (start with ½ a second but you may need to adjust). Import the time module and use time.sleep(sleeptimeinseconds) for this;
      5. Repeat until the user chooses to end/pause.
   9. If the user presses ‘p’ stop iterating the game of life and display the current grid.
4. File format:
   1. File has the extension .life (user does not enter the extension) and is in the folder lifeFiles.
   2. Files contain numbers in a grid, 0 meaning the cell is not alive and 1 indicating that the cell is alive.
      1. One line per row of grid
      2. MUST be a square
      3. No spaces, commas or anything between numbers. For example, a file might be as follows with the resulting grid on the right:  
         010010  
         100000  
         010111  
         100100  
         011111  
         010010  
         Ones are ‘live’ spaces and zeroes are ‘dead’ spaces

* 1. Read the file and create a grid based on the values.
  2. Perform error checking on file read and file contents.

POSSIBLE METHOD:

It is not a requirement but I recommend creating a list in a list. You can do this yourself, or you can use numpy. I am not going to have time to teach this before your assignment is due, but it is a specialized library for dealing with matrices and grids. For example, numpy.zeroes(6,6) creates a 6x6 grid of zeroes.

**Final Five**

Completing the above receives a maximum of 95%. The final 5% is available if you complete at least two of the following extra pieces of functionality.

1. Cell colours changing – Make newly alive cells a different colour than stasis colour.
2. Add speed…keys one through five set speed. The higher the number the faster the speed.
3. Use right arrow key to skip to next generation when paused (and stay paused).
4. Implement drag functionality to be able to click with the mouse and toggle as you drag the mouse around the screen.
5. Allow the user to change the size of the grid without having to restart.

**Do’s and Don’ts:**

Do:

1. Create a test case file in Word of all the conditions that you are going to test. Specify what you are testing and how you will test it. For example, you can say you will test errors in the grid size using the file notthere.life from the lifeFiles folder.
2. Display a list of the options for the user somewhere so that they know what their choices are. This can be part of the grid window or (probably better) a separate small window.
3. Make sure you use the efficient methods for outputting strings with the f’ format and comprehensions for all lists wherever possible.
4. Use exception handling with try/except/else blocks
5. Use magic functions appropriately for any classes you create (and you should have classes).
6. Think Pythonically. That is, follow the guidelines established in the [PEP 8 Styling Guide](https://www.python.org/dev/peps/pep-0008/).

Don’t:

1. Be procedural.
2. Put everything in one function.
3. Use arrays (they do exist in Python, but don’t use them).

**To submit**

1. A ZIP format (*initials*G40A03.zip) containing all submitted files on Moodle.