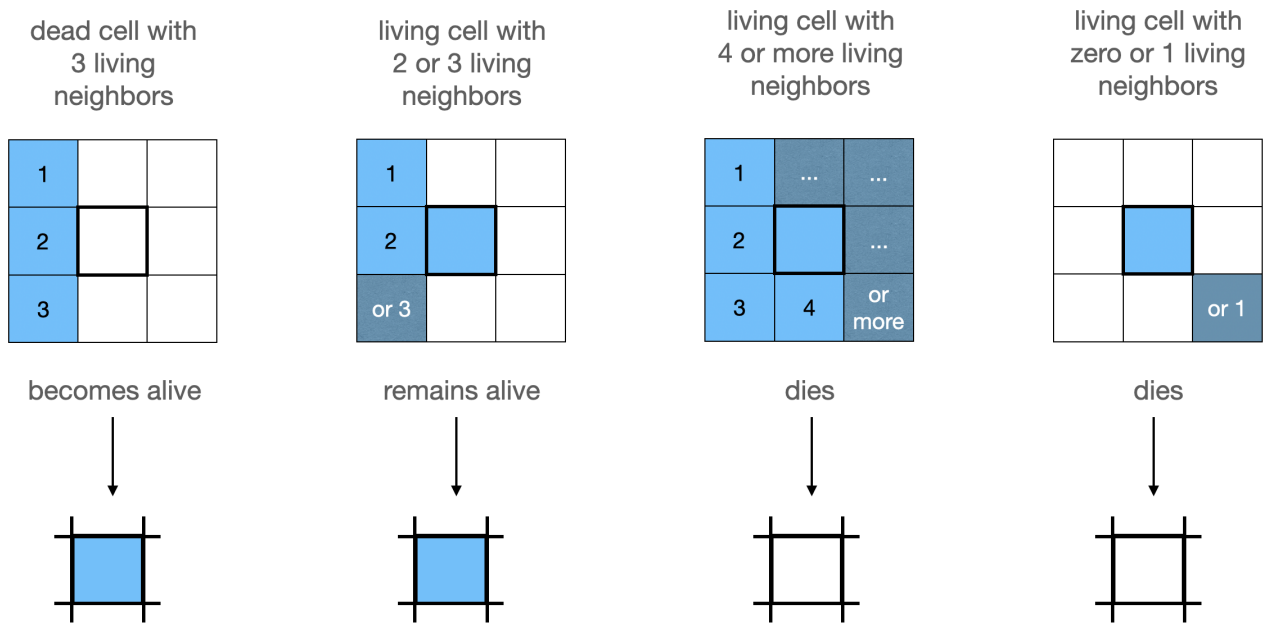
Conway's Game of Life: C++ Implementation

In this project, we'll develop an implementation of Conway's Game of Life.

History of the Game

Conway's Game of Life - Wikipedia is a cellular automaton devised by the British mathematician John Horton Conway in 1970. Its roots can be traced back to the 1940s when John von Neumann and Stanislaw Ulam were developing simulation theories at Los Alamos National Laboratory. They did this work not long after working with Oppenheimer to develop the first nuclear bomb. The game, which started as a mathematical concept, has applications in various fields including biology, economics, and computer science.

The Game of Life simulation operates on a grid of squares (cells) where each cell can be alive or dead, and evolves in steps from generation to generation, following a set of simple rules based on the number of living neighbors a cell has.



Your job will be to implement the Life class. Life makes use of a few concepts we've been learning, such as maintaining a dynamic array, using a flat array to represent a 2D array, as well as implementing the "rule of three" methods: copy constructor, assignment operator and destructor.

To-Do

- Familiarize yourself with **main.cpp**. In it, you'll find some static variables and a function that enables the user to select a figure file. Figure files describe the initial conditions for the simulations. Also, please review one or two `/figures` files which describe the initial state of the simulations.
- Implement the `Life` class according to the interface described in **Life.h**. `Life` manages an array of bools called `_cells`. A cell has a true value when it is alive, and a false value otherwise. The array of bools will represent a 2D grid of cells.The game of `Life` class has three main capabilities:

    - Read itself from a file. See the `Life(ifstream &ifs)` constructor.
    - Render its state on a `ConsoleGrid*` by setting tiles corresponding to cells. See the `draw(ConsoleGrid *grid)` method.
    - Return a new instance (a new generation) of itself with new cell states modified according to the rules of Conway's Game of Life. See the `nextLife()` method.

- Compile and run. Try each of the figure files, and feel free to find or design your own. Just add a file to the figures folder and add its file name to the `figures[]` array.

This Project Requires a Lot of Console Space

Be sure to stretch your console window side-to-side if necessary, and use (cmd or cntrl) + the minus key to zoom out your browser window.

Further Study (optional)

If you enjoy this subject as much as I do, you may be interested to study it further. Given an instance of Life in an arbitrary initial state, can predict how it will look in some future generation without running the game of life rules over and over (calling nextLife())? Is there any other way to predict how a Life will turn out without running it? **1 Think about it.

In fact, there is no way -- apart from running the simulation -- to know how a Life with a given initial state will turn out. The technical term for this is "computational irreducibility" **2.

There's a related question which you might try to answer: Will a given instance of Life ever repeat its state? If so, we know that instance will loop forever. How might one go about identifying a repeated state? You could, for example, try the following:

- Implement the `toString()` instance method on the Life class. Decide on a representation that completely encodes the instance as a string. This representation must be complete and reasonably compact.
- In the main(), keep track of each instance of Life created at each generation. Consider using a vector of strings.
- When a new Life instance is created, determine if it matches any saved instances. If it does, you've found a cycle.
- Upon finding a cycle, call the `PrintStatusMessage()` function with the optional `append` parameter indicating that a cycle was found, and stop the simulation.

Resources

- Conway's Game of Life - Wikipedia
- Learn CPP

Footnotes

- **1 Fun coincidence that this question seems equally salient to real life, too.
- **2 Like life in our simulation, is *your* behavior determined at the outset while still being computationally irreducable? Stanford professor Robert Sapolsky seems to think so.