Conway's Game of Life Assessment

Assumptions about the rules of the game:

- Conway's Game of Life is a zero-player game where the initial state is a two-dimensional grid of square cells that are either alive or dead. Each cell has eight neighbors (two vertically, two horizontally, and four diagonally) that determine the state of the cell for the next generation. The rules are as follows:
    - A live cell with fewer than two live neighbor dies in the next generation
    - A live cell with two or three neighbor lives on in the next generation
    - A live cell with more than three live neighbors dies in the next generation
    - A dead cell with exactly three live neighbors lives in the next generation

Questions to the Product Owner

- Who will be using the software package? Which client (whether external or internal) will be using the software package? This will determine the project's importance and what priority we will assign to the project.
    - If the project is deemed a high priority project, what is the expected timetable that the client will need delivery of the project.
    - How much engineering resources is allotted to this project? After estimating the effort required to complete the project, we will determine if the timetables are feasible given the engineering resources allotted.

- Conway's Game of Life is dependent on an initial state. Who will provide the initial state? The assumption is that the user of the package will provide the initial state. Along those lines, these are additional questions:
    - Will the package be a standalone app where the user provides the initial state and can observe the state of the Game of Life after *x* number of generations? My assumption is that the user of the package will want to be able to give an input state and get an output based on the final state after *x* number of generations. Our user is responsible for:
        - Initial State of the World
        - Number of Generations to run the Game of Life

- How will the user of the software package provide the input states and retrieve the final output? Will we give them multiple ways to do this? Some ideas are:
    - HTTP Requests
    - Input Files/Output Files – User provides an input file, and the package provides an output file with the results.
    - Direct Method/API calls – Our software package will provide a public method that can be called. It will return a two-dimensional bit/Boolean array
    - If a standalone app, we can allow them to download the final state as a CSV or an image file that has the representation of the grid in its final state.

- Is the size of the grid static or dynamic? In other words, is the size of the grid limited from the initial state of the game.

    - If the size can expand as necessary, is there a hard limit on the size? We should probably cap the grid size to avoid any potential memory leak issues
    - If the size is fixed, do the cells roll over if it overflows. For example, the cells above the topmost rows are in the bottom rows and the cell to the right of the rightmost columns are in the leftmost columns.

- The default rules were specified in the Assumptions about the rules of the game. Should we allow the users of the package to modify the rules of the game?

Subsystems Needed for This Project

- Input Module(s)
    - This module is responsible for giving the user a way to input the initial state of the game as well as other configuration options such as iterations/generations to run the game.
    - The module can be as simple as the ability for the user
        - To provide an input file.
        - To hit an HTTP endpoint
        - To call a public method that will return an output of a two-dimensional array.
    - The module can be as complicated as a GUI where the user can create the size of the grid, specify each individual cell as alive or dead, and the number of iterations/generations to run the game.

- Game of Life Engine
    - The Game of Life Engine should be consistent regardless of the input or output module.
    - The underlying logic of the game is found here.
    - The engine is responsible for
        - Receiving the initial state from the input module.
        - Iterating through the correct number of generations
        - Updating the size of the grid/array if the grid size is dynamic. In addition, the engine should know when the grid size has reached its maximum size and disallow any further expansions.
        - Updating each cellular state after each generation.
        - Provide the final state to the output module.

- Output Module(s)
    - This module is responsible for giving the user a way to extract the final state of the game.
    - The module can be as simple as the ability for the user to gather the final state by
        - Providing an output file.
        - Responding to the HTTP endpoint call
        - Returning an output of a two-dimensional array after a method call.
    - The module can be as complicated as a GUI where the user can view the final state. For the GUI, we should also give the user a way to download the results via a CSV or image file.