CMSC 447

Software Design Description (SDD)

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Signature** |
| Caroline Cocca | POC/Team Lead |  |
| James Gough | Simulation Dev |  |
| Stuart Reilly | Junior Developer |  |
| Kelley Schmidt | Documentation and  Diagrams Technician |  |
| Ryan Messett | Data Guru |  |
| Sang Nguyen | Visual Specialist |  |
| Dr. Jon Squire | Customer |  |

[1 Scope 3](#_gjdgxs)

[1.1 Identification 3](#_2et92p0)

[1.2 System overview 3](#_tyjcwt)

[1.3 Document overview 3](#_3dy6vkm)

[2 Referenced documents 3](#_4d34og8)

[3 CSCI-wide design decisions](#_17dp8vu) 4

[4 CSCI architectural design](#_26in1rg) 5

[4.1 CSCI components](#_35nkun2) 5

[4.2 Concept of execution](#_2jxsxqh) 6

[4.3 Interface design](#_1y810tw) 8

[4.3.1 Interface identification and diagrams](#_4i7ojhp) 8

[4.3.2 (Project-unique identifier of interface)](#_1ci93xb) 10

A. Appendix A - Programming Language AOA 11

# Scope

## Identification

This document applies to the design of the Scrum-Diddlyumptious Python implementation of Conway’s Game of Life, which we have titled “Game of Life”. Our software application will be able to run on a Windows 10 machine via command prompt. The application is a Python toolkit GUI and there are no specific software requirements needed in order to be able to run the application. Our current rapid prototype we will refer to as version number 1. Identification number pending, release number pending.

## System overview

The purpose of our Game of Life software application is to recreate the classic zero-player cellular automaton developed in 1970 by John Horton Conway. Our implementation has additional requirements that we must meet to make it an interesting project. The Game of Life demonstrates how the evolution of cells on a grid creates patterns. How the cells interact on the grid is determined by a set of rules. The purpose of the system - the Windows 10 machine - is to run our software application. The development of our software application has begun within the last few weeks, as was the development of specific requirements and documentation. The developer of the application is the Scrum-Diddlyumptious team. The acquirers of the software are our customer and our professor. The team, our customer, and our professor are all users of this application.

## Document overview

The purpose of this document is to provide an overview of the design for the Game of Life. This includes behavioral design and architectural design. The behavioral design includes how the user interacts with the software and visually how the application behaves. The architectural design includes software unit identification, concept of execution, and interface design. These are the internal specifications of the software. There are no privacy concerns or security concerns involved with using our application.

# Referenced documents

Conway's Game of Life. (2019, March 24). Retrieved from <https://en.wikipedia.org/wiki/Conway's_Game_of_Life>

Tkinter -- Python interface to Tcl/Tk. (2019). Retrieved from <https://docs.python.org/2/library/tkinter.html>

Pandas: Python Data Analysis Library. (2019). Retrieved from <https://pandas.pydata.org>

# CSCI-wide design decisions

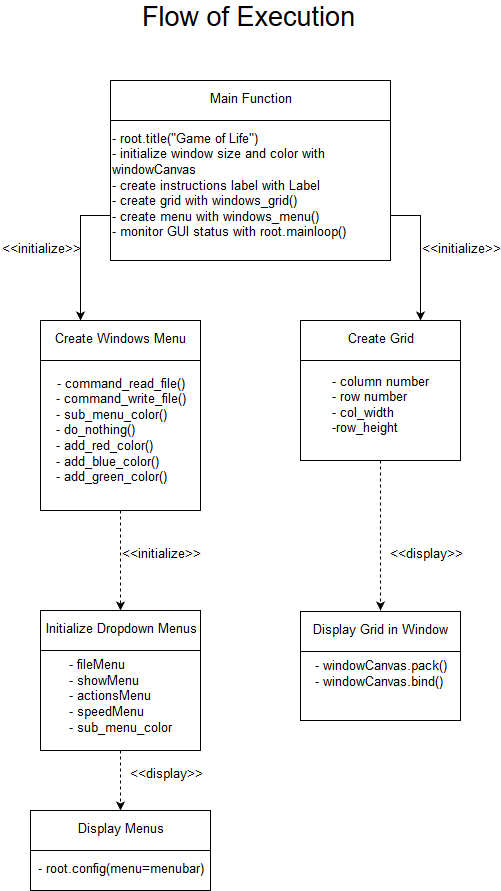
* To meet the requirement to **give the user the ability to add color to a grid square (Requirement 1)**, given the user has clicked on a color option from the submenu “Colors”, subsequently clicking on a grid square will alter it to display the chosen color. This action does not alter any other squares in the grid.
* To meet the requirement for **saving the game state to a file (Requirement 2)**, the user will be able to click the save game button on the file menu and save the game with a specified name. The filename will be checked to be a valid Windows 10 filename before the game is saved.
* To meet the requirement for **loading the game state from a file (Requirement 3)**, the user will be able to click load game. Windows explorer will open, and the user will be able to select the file from the file system. The file will then be checked to be a valid “Game of Life” load file. If the file is valid, the game state will return to the same state as the saved game in the file.
* To meet the requirement for **providing the program with pause capability (Requirement 4)** and **unpause capability (Requirement 11)**, the user will be able to see a ‘Stop’ button to pause the game and a ‘Run’ button to unpause the game. These buttons on the GUI will be connected to a boolean that will be checked before the end of each step, as the program is running. To meet the requirement for **following the rules of Conway’s Game of Life (Requirement 12)**, upon ‘Run’ being selected, cells in the grid will begin to appear, disappear, and move with each step according to the rules specified in the Wikipedia of Conway’s Game of Life.
* To meet the requirement to **allow the user to change the speed of the rate of gameplay (Requirement 5)**, there will be a ‘faster’ and a ‘slower’ button, visible on the GUI. The game will start at a rate of loop iteration, artificially slowed by a time wait variable. Clicking the ‘Slower’ button will increase the time wait variable by a factor of two. Clicking the ‘Faster’ button will reduce the time wait variable’s value by a factor of two.
* To meet the requirement for **displaying life count (Requirement 6)** and the requirement for **displaying step count (Requirement 7)**, the user will be able to view displays of either count in the upper right corner of the Graphical User Interface. Before the end of each step, both counts will be updated to reflect the change in game steps and squares alive in the grid. Both counts begin at zero, and return to zero when the game is cleared.
* To meet the requirement for **toggling the appearance of grid lines (Requirement 8)**, the user will be able to make a selection from a drop down menu. From the user’s perspective, when the application is running, there will be a selection box named “Grid/Speed.” When the box is clicked, a drop down menu appears with options “No Grid”, “Show Grid”, “Faster”, “Slower”, and “Color.” When the user selects the “No Grid” option when the grid lines are showing, then the appearance of grid lines will be turned off. If the user selects the “Show Grid” option when the grid lines are not showing, then the appearance of grid lines will be turned on. If the user selects “No Grid” when there are no grid lines or “Show Grid” when there are already grid lines, then nothing will happen.
* To meet the requirement to **display three color options to the user (Requirement 9)**, there will be a dropdown menu on the top of the program including “File”, “Actions”, and “Show”. There will be a parent menu “Grid/Speed” with the child submenu named “Colors” to show the desired colors. There will be three colors that the user can choose from: red, green and blue.
* To meet the requirement to **detect stable patterns (Requirement 10)**, the word “Stable” will appear in the top right corner of the GUI when the entire grid has stable patterns oscillating (patterns of squares which repeat in cycles and stay in place). This word will appear after one complete stable oscillation in the entire grid and remain as long as the entire grid continues to have stable oscillations. If there is a stable pattern in one area of the grid but other active squares are *not* part of a stable pattern, the word “Stable” will not appear.

# CSCI architectural design

## CSCI components

* **refresh\_life(state):** This function will take in a list recording the current state of the grid (the coordinates of each square currently alive). This will use the rules of Conway’s Game of Life to determine which squares remain alive, which dead squares come to life, and which squares die. The function then updates the state of the grid as well as the current life count. If any squares exit one side of the grid as a result, the square will be wrapped around to re-enter the opposite side of the grid. The step count is increased by one to reflect another step has completed. This fulfills the requirement that a life count be displayed to the user, as well as the requirement that a step count be displayed to the user.
* **run\_game( ):** This function will set the boolean variable that allows the game to continue its running loops to True, so that the game may be unpaused. It will be triggered by pressing the ‘run’ button on the GUI. This function helps to fulfill the pause and unpause capability requirement.
* **pause\_game( ):** This function will set the boolean variable that permits running of the game to false, so that the loops allowing movement will not iterate again until the boolean is set back to True by the **run\_game( )** function. It will be triggered by pressing the ‘stop’ button on the GUI. This function helps to fulfill the pause and unpause capability requirement.
* **change\_speed(factor):** This function will scale the wait time variable by the passed factor. In so doing, the game loops will be able to iterate faster or slower, thus helping to fulfill the requirement for the user to be able to speed up or slow down the rate of gameplay. This function will be triggered by the user pressing the ‘faster’ or ‘slower’ button on the GUI.
* **save\_game():** This function will open windows explorer for a user to save the file with a valid name in the specified location. The function will be triggered by the user clicking File->Save Game on the GUI.
* **load\_game():** This function will open windows explorer for a user to select a valid save file. The function will be triggered by the user clicking File->Load Game on the GUI.
* **show\_grid( ):** This function will control making the grid of cells appear on the application. This function helps fulfill the requirement to be able to turn on and off the appearance of the grid.
* **hide\_grid( ):** This function will control turning off the appearance of the cells that appear on the application. This function helps to fulfill the requirement to turn on and off the appearance of the grid.
* **show\_color():** This function will allow the user to choose from 3 different colors by displaying the options on the GUI. This function will be triggered on the “Color” submenu from the parent menu “Grid/Speed”. It will help fulfill the requirement to show the colors the user can choose.
* **change\_color():** This function will change the color of the selected grid square according to the selected color. This function will be triggered once the user clicks on the desired color from the submenu “Color” and subsequently clicks a grid square. It will help fulfill the requirement to change the grid color.
* **check\_stable():** This function will detect if the entire grid has reached a stable state. This function will be triggered at the end of each step to check whether the grid has reached the end of one stable oscillation. If so, the word “Stable” is placed in the top right of the GUI. If not and the grid was previously marked stable, the word “Stable” is taken off of the top right of the GUI. Otherwise, no action is taken.

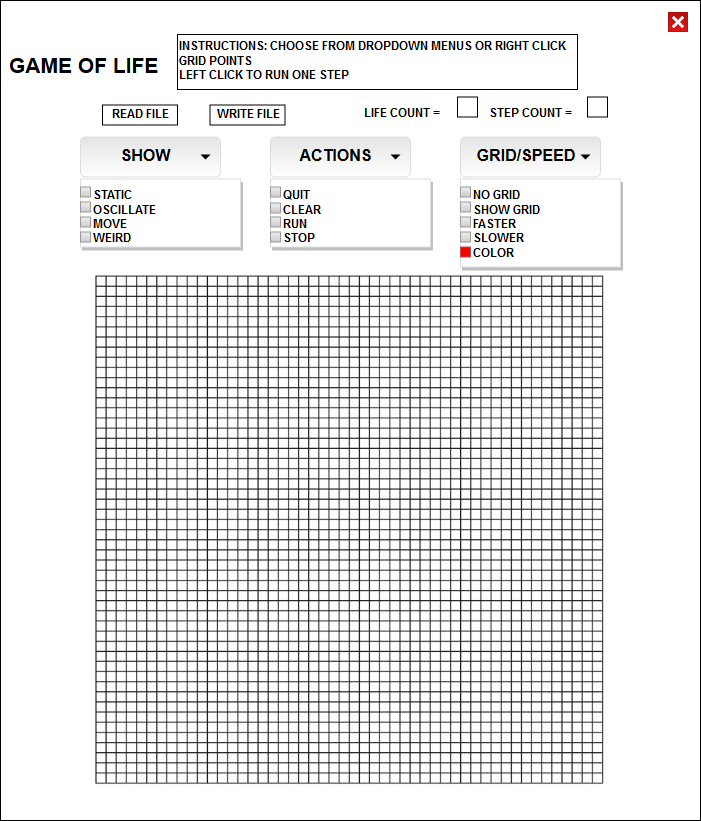
## Concept of execution



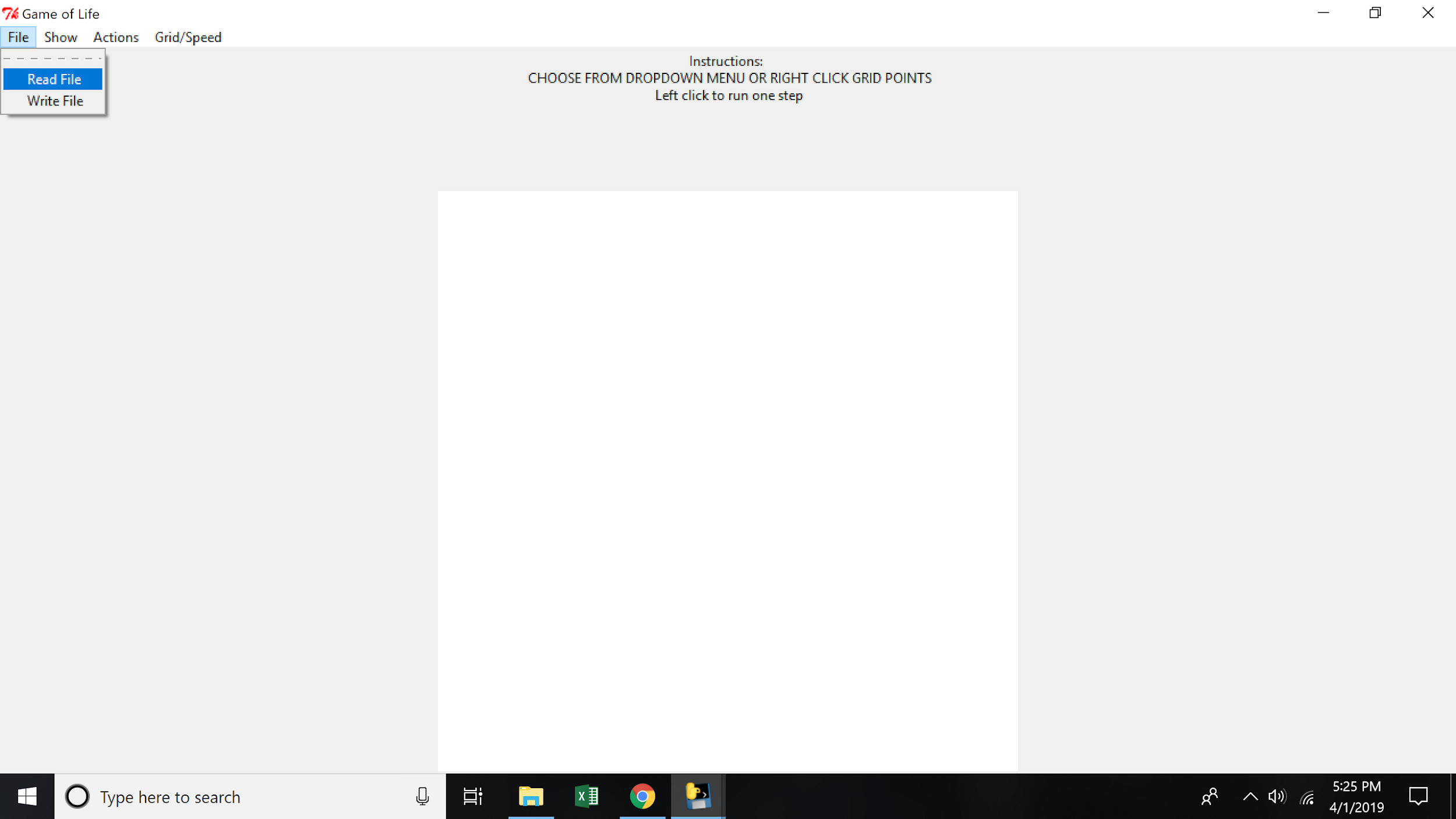
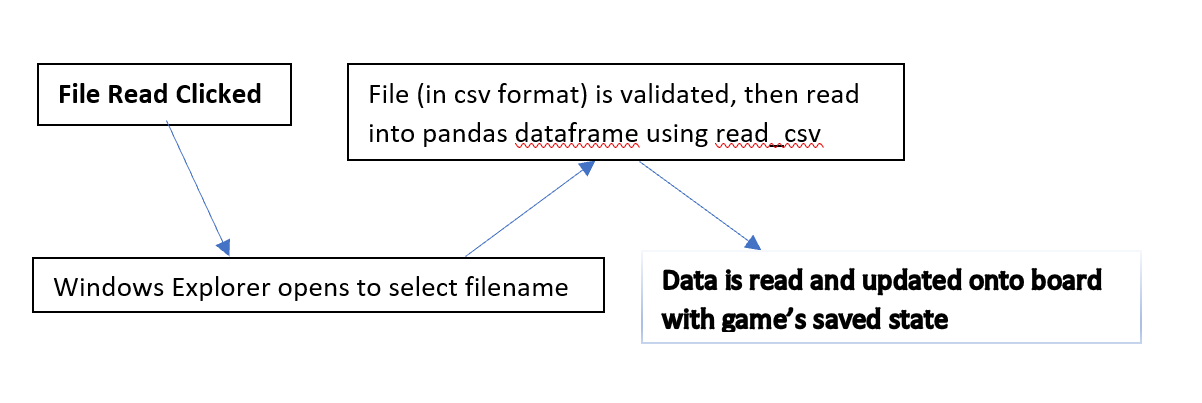
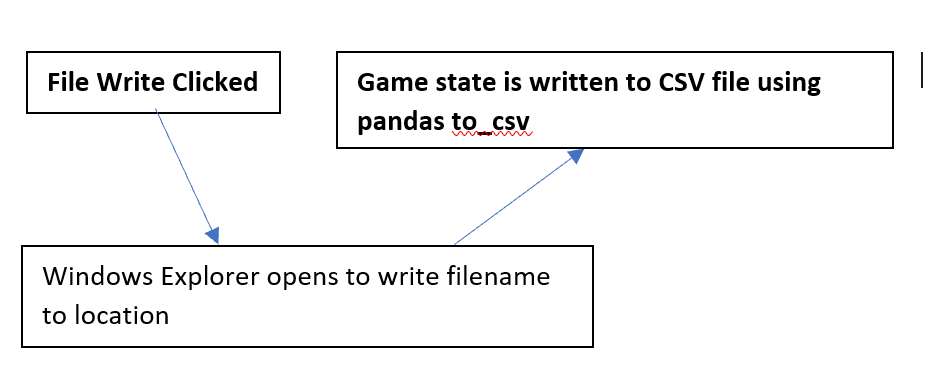
## Interface design

### Interface identification and diagrams

* **GUI -** Graphical User Interface



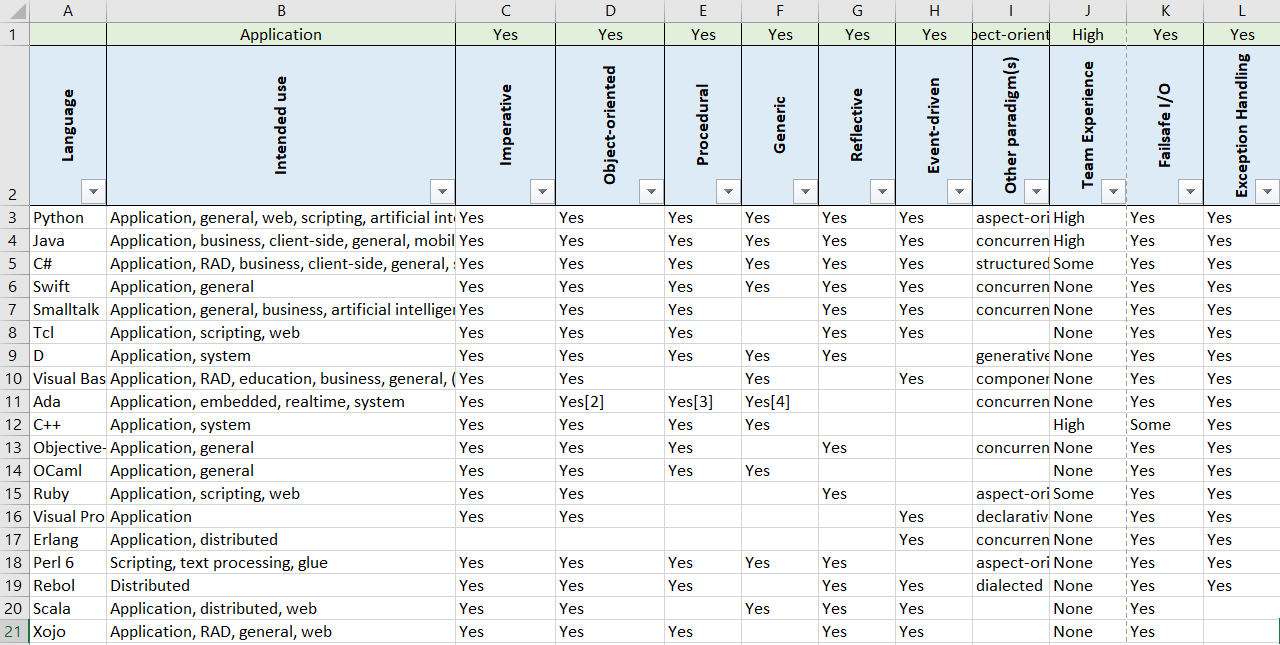
* **I/O -** File read and write

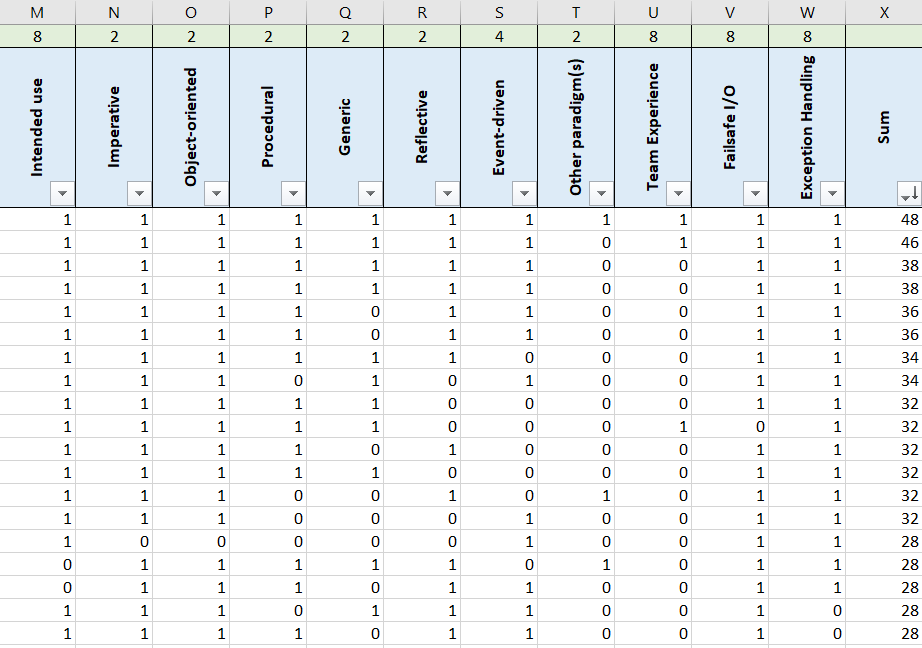
1. 
2. 
3. 

### (Project-unique identifier of interface)

* **GUI -** This interface is the Graphical User Interface, which serves as a visual way for the user to interact with the Game of Life program. This interface will utilize Tkinter library default colors, layouts, fonts, icons, menus, and other display elements. This interface does not have any audio components. The GUI accepts input in the form of clicks from the user which trigger functions linked to menu items and the grid.
* **I/O -** This interface is the interface between the operating system’s file storage system and the program for File I/O. The file name shall be a string of characters except (/ \ : \* ? " < > |). The filename shall be less than or equal to 255 characters. The coordinates of the grid shall be stored with integer values (x, y) for the first two columns of a dataframe and the third column shall specify color of each coordinate in character non-numeric string format. The other game data shall be stored in the first row of the file: move\_count, life\_count, and NULL (nothing needed in 3rd column). All of this data shall be stored in a pandas dataframe (on save) because of the convenient to\_csv function that exists, and the simplicity/readability of the code. The csv file shall be loaded into a pandas dataframe on reading a file and then all the data will be checked and converted into the actual game’s board data. This identifier is called File Read and Write. See the above diagrams for further process visualization.

**Appendix A - Programming Language AOA**

****

****