CMSC 447

Software Requirements Specification (SRS)

|  |  |  |
| --- | --- | --- |
| **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](#_3znysh7) 3

[1.1 Identification](#_2et92p0) 3

[1.2 System overview](#_tyjcwt) 3

[1.3 Document overview](#_3dy6vkm) 3

2 Referenced Documents 3

3 [Requirements](#_17dp8vu) 4

3[.1 Required states and modes 4](#_26in1rg)

3[.2 CSCI capability requirements](#_35nkun2) 4

3.3 [CSCI environment requirements](#_1hmsyys) 8

3.4 [Computer resource requirements](#_2grqrue) 8

3[.4.1 Computer hardware requirements](#_1v1yuxt) 8

3[.4.2 Computer hardware resource utilization requirements](#_4f1mdlm) 8

3[.4.3 Computer software requirements](#_3tbugp1) 8

3[.4.4 Computer communications requirements](#_28h4qwu) 8

3.5 [Software quality factors](#_37m2jsg) 8

3.6 [Design and implementation constraints](#_46r0co2) 9

3.7 [Personnel-related requirements](#_111kx3o) 9

3.8 Packaging requirements 9

3.9 [Precedence and criticality of requirements](#_1rvwp1q) 9

[A. Appendix](#_34g0dwd) A - Visual Prototype 10

# Scope

## Identification

This document applies to the specific requirements for our Python implementation of Conway’s Game of Life. The application will be able to run on Windows 10 via command prompt. No specific software installs are required other than the files for this project and the libraries chosen to import in the design. The main program will be in the form of a Python file. The Game of Life is a cellular automaton that is a single-player game with the only user input needed being the initial setup of the starting cell(s) and menu item selections. The cells are contained in squares of a grid, with each square of the grid being a specific x-y coordinate. The cells multiply, shrink, and move to form patterns according to specified algorithms based on their placement and the placement of surrounding cells. The goal of the game is to create interesting patterns through various starting cell coordinates. Title of application pending, version number pending, release number pending.

## System overview

The purpose of this application is to demonstrate a classic cellular automaton with client-specific requirements. The input of the initial state of the game is the only input required in order to play the game. The software is a visual program and various aspects can be configured by the user. The system on which the software will run is the Windows 10 operating system. A mouse is required for the system to be able to navigate the software. The users of this software are the developers, graders, and the client. There are no project sponsors as this project will have no monetary cost to produce. The acquirer for our software application is the client, Dr. Jon Squire. The developers consist of our software engineering team: Kelley Schmidt, Caroline Cocca, James Gough, Sang Nguyen, Ryan Messett, and Stuart Reilly. The support agencies for our project would be the teaching assistants, Professor Cain, and the client if support is needed. Planned operating sites not applicable. Other relevant documents are the Software Design Description, Software Test Description, Software User Manual, and Software Test Report.

## Document overview

The purpose of this document is to formally identify the physical and programmable requirements for our software application. The requirements are broken down into different categories based on the type of specification and for ease of readability. There are no security concerns with reading and/or using this document and there are also no privacy issues. All work presented in this document is done by the development team.

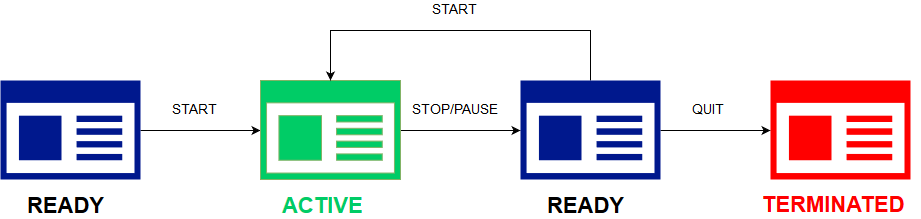
# Referenced Documents

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

# Requirements

## Required states and modes

The CSCI will consist of 2 states: ready and active. Upon launch, the application will be in the “ready” state, awaiting the user input. Once the user input is specified, the application will then go into its “active” state and begin its programmed steps. When the application is paused or stopped, it will enter the ready state again and await user input to start again or to terminate the session.



## CSCI capability requirements

**Rank key: 1 = Most important, 2 = Moderately important, 3 = Least important**

**Requirement 1:** The color of a square clicked in the game grid shall change to the color most recently selected on the color menu.

* **Error handling:** None. Only the options in the drop down box shall be available for picking, and through the qualification provision listed below, this shall be verified before the software is in a production environment.
* **Response time**: Upon selecting a color and consequently clicking on a square in the grid, the square shall be filled with the selected color in less than one second
* **Qualification provision**: To test this requirement, we will select each individual color and click five random squares after the selection to validate that the color fills each of the squares.
* **Traceability:** This requirement traces back to our system requirement that the user shall be able to select a color and click a square on the board to change its color.
* **Rank:** 3

**Requirement 2:** The program shall allow for the saving of the current grid state to a file.

* **Error handling:** An error that could occur upon saving the game’s state could be error in file I/O from the host operating system. A disallowed file name could also be specified in windows. The file name shall be checked to ensure proper allowable windows file names and the output file shall be checked against the current game state.
* **Response time:** The game shall save the board’s state to a file, with a name specified by the user, in less than 2 seconds.
* **Qualification provision:** First, saved game file names shall be checked in the specified location on the host operating system to ensure that file names are being recorded properly. Improper windows file names shall be entered into the “save game” option to check for proper handling of improper usage. Save games shall be checked against the current game board(s) to ensure accuracy of each color, x, and y values.
* **Traceability:** The “save game” function traces back to our system requirement that the user shall be able to save the game state to return to a specific point in play at a later point in time.
* **Rank:** 2

**Requirement 3:** The program shall allow for the loading of a saved grid state from a file.

* **Error handling**: If a nonexistent file name is given, or a file is given with improperly formatted contents, an error message will be displayed to the user clearly stating why the file failed to be loaded. The user will be re-prompted for another file.
* **Response time**: The game shall load the board’s state from a file, with a name specified by the user, in less than 2 seconds.
* **Qualification provision**: Each possible error shall be tested by inputting a nonexistent file name and inputting a file with improperly formatted contents. In each case, the error handling requirement will be fulfilled if an error message is displayed and the user is re-prompted. Three existing files with properly formatted contents will also be input to test the program ability to load game states. If the saved coordinate and color information is accurately recreated in the grid for each file, this requirement is fulfilled.
* **Traceability:** This traces back to our system requirement that the game shall be able to load games from file and continue at whatever state the game file contains inside.
* **Rank:** 2

**Requirement 4:** The program shall allow for the user to pause the game.

* **Error handling: None**
* **Response time:** Upon clicking the pause button, the program shall change from the Run state that it was previously in to the paused state within 1 second.
* **Qualification provision:** To test this requirement, we will have a test case with the game in the Run state. We will click the pause button, and immediately record the time from the click of the pause button. Within 1 second movement on the board should freeze/cease.
* **Traceability:** This function traces back to the requirement that our game should include a pause functionality.
* **Rank**: 1

**Requirement 5:** The program shall have an option to change the speed of the rate of gameplay, while the game is running.

* **Error handling: None**
* **Response time:** Upon clicking on any of the options to speed up gameplay, the rate of the board’s movement/play shall increase its speed by that factor, within 1 second from the time that the speedup option is clicked on.
* **Qualification provision:** To test this requirement, we will select each speedup option, one at a time. After having selected a particular speedup option, the rate of movement on the board must increase its speed by the factor displayed on the button, within 1 second of having clicked on the aforementioned button.
* **Traceability:** This function traces back to the requirement that our game should include a speed up game rate functionality.
* **Rank**: 2

**Requirement 6:** The program shall display to the user a life count.

* **Error handling: None**
* **Qualification provision**: To test this requirement, we will use the test cases from the Conway’s Game of Life wikipedia page. These tests will include “Block”, “Beehive”, “Glider”, “Middleweight spaceship”, and “Beacon”. If the life count displays an accurate report for 10 steps for each test case, the life count requirement will be considered fulfilled.
* **Response time:** The life count shall be updated before each step has ended.
* **Traceability:** This function traces back to the requirement that our game should include the display to the user life count.
* **Rank:** 2

**Requirement 7:** The program shall display to the user a step count.

* **Error handling:** If a player has a step count greater than 9,223,372,036,854,775,807(“nine quintillion, two hundred twenty-three quadrillion, three hundred seventy-two trillion, thirty-six billion, eight hundred fifty-four million, seven hundred seventy-five thousand, eight hundred seven”), we shall notify the user that the current game session will be restarted in order to avoid integer overflow.
* **Qualification provision**: To test this requirement we shall have two test cases: one to test integer overflow and another for negative numbers.
* **Response time:** The step count shall be updated before each step has ended.
* **Traceability:** This function traces back to the requirement that our game should include the display to the user step count.
* **Rank**: 2

**Requirement 8:** The program shall allow the user to toggle the appearance of grid lines.

* **Error handling: None**
* **Response time:** Upon clicking on the button for the grid lines the appearance of grid lines shall transition from the on or off state, to the opposite state, within 2 seconds time.
* **Qualification provision:** To test this requirement, we will shall start with a game that has the grid lines on turned on. The button to turn off the grid lines shall then be pressed. Within 2 seconds of having pressed the said button, the grid lines will disappear from the user’s screen view. Then, the button to turn the grid lines back on shall be pressed. Within 2 seconds of having pressed the said button, board grid lines should reappear on the user’s view screen.
* **Traceability:** This function traces back to the requirement that our game should include functionality to adjust the appearance of grid lines.
* **Rank**: 3

**Requirement 9:** The program shall allow for 3 different colors to be chosen by the user for grid coloring.

* **Error handling:** None. Drop down box shall ensure that only specific colors are able to be selected.
* **Response time:** Color shall change in less than 1 second.
* **Qualification provision:** This requirement shall be verified via observation by checking the Grid/Speed menu and ensuring there are three Color options presented to the user.
* **Traceability:** This requirement traces back to the customer’s requirement that there shall be at least 3 different colors available for selection.
* **Rank**: 3

**Requirement 10:** The program shall display to the user the word “Stable” if the entire grid contains only stable oscillating patterns.

* **Error handling**: **None**
* **Response time**: The word “Stable” shall appear within one second after one complete cycle of stable patterns across the entire grid. The word “Stable” shall disappear within one second after patterns become unstable, which could occur if the user paused to enter more color squares which disrupted existing stable patterns.
* **Qualification provision**: This requirement shall be tested by using three separate stable pattern test cases from the Game of Life Wiki to ensure their stability is detected. The requirement shall also be tested by having one stable and one non-stable pattern in the grid, as well as only one non-stable pattern in the grid, to ensure “Stable” is not displayed.
* **Traceability**: This requirement traces back to the customer’s requirement that the user shall be notified of existing stable patterns in the grid.
* **Rank:** 1

**Requirement 11:** The program shall allow for the user to unpause the game.

* **Error handling: None**
* **Response time:** Upon clicking the Run button, the program shall change from the paused state that it was previously in to the Run state within 1 second.
* **Qualification provision:** To test this requirement, we will have a test case with the game in the paused state. We will click the Run button, and immediately record the time from the click of the pause button. Within 1 second the first step should begin execution.
* **Traceability:** This function traces back to the requirement that our game should include unpause functionality.
* **Rank**: 1

**Requirement 12:** The game shall follow the rules of Conway’s Game of Life.

* **Error handling: None**
* **Response time:** N/A
* **Qualification provision:** To test this requirement, we will use several different test cases given in the Wikipedia of Conway’s Game of Life (specifically, the Examples of Patterns section). We will verify that “still life” patterns remain static, “oscillators” cycle in the specified shapes, and “spaceships” move about the screen in the specified shapes.
* **Traceability:** This function traces back to the requirement that our game shall be modeled after Conway’s Game of Life.
* **Rank:** 1

## CSCI environment requirements

The CSCI must operate in a Windows 10 environment. The computer hardware must include one screen to view the program, one mouse, and one keyboard to interact with the program.

## Computer resource requirements

### Computer hardware requirements

The computer hardware must include one screen to view the program as well as one mouse to interact with the program. The computer must have 1 GB of RAM available for running the program.

### Computer hardware resource utilization requirements

Depending on the computer CPU(central processing unit), RAM(random access memory), GPU(graphical processing unit) the utilization of the computer internals may vary. The amount of memory that the program use will be less than 500 megabytes. The computer should have 1 GB of RAM or more for the program to perform optimally. The GPU utilization would be negligible if played on original resolution. If the monitor resolution is 4K, it might be higher than original resolution. Example would be that if the computer utilized a motherboard onboard graphics, such as the Intel HD 630, then it would be slightly higher compared to a computer that has a dedicated GPU, such as the Nvidia RTX 2080 ti. The CPU utilization would be negligible and would be used if the player were to interact with the program (clicking on menu items, saving the game, changing the color selected).

### Computer software requirements

The operating system that is needed to execute the program is Windows 10 with at least python version 3 or higher installed along with Python package management system (“pip”). The Python package management system, pip, will be necessary to install any libraries imported by the program (specifically, TkInter and Pandas).

### Computer communications requirements

The program does not need any geographic locations to be linked or internet connection in order to execute successfully.

## Software quality factors

The program will perform Conway’s Game of Life simulation either by manually stepping through each generation or running continuously through each generation. If the program is running continuously, then there will be an option to increase the speed of the simulation. The program will have the option to turn off the grid. The program will have a GUI with options to change the color of boxes in the grid. The program will be robust enough to handle high step counts. When the step count gets too high, the user will be prompted to reset the simulation to avoid integer overflows. The program will be maintained through a GitHub repository accessible by all members in the Scrum-Diddlyumptious team. The program will also easily be available through the GitHub repository. The program will be flexible by use of the GitHub repository. If a requirement is changed, a branch can be made from the repository to implement the missing requirement, then once the code is thoroughly tested, the branch will be merged into the main branch in order to complete the missing functionality. The program will run on Windows 10. Ensuring that the program will run or can be modified in a new environment will not be a main focus of the program. The program will be available on GitHub for reuse in other applications. The program is a stand alone application and ensuring that the program will be able to be reused in other applications will not be the main focus of the project. The program will be thoroughly tested through the use of unit testing and code reviews from other members of the Scrum-Diddlyumptious team. Each component of the code will be thoroughly tested before it is merged into the main branch of the repository. The program will be easily usable through the use of a GUI. Additionally, a user manual will be provided with the program.

## Design and implementation constraints

1. Pandas will be used for data manipulation in the program. The graphical part of the program will be coded using TkInter.
2. The Conway’s Game of Life application will be written using Python version 3.7.
3. The program will be maintained using a GitHub repository. If requirements are changed, the modified requirement will be coded and tested in a separate branch. Once the additional code is approved by team members, the separate branch will be merged with the main branch to complete implementation of the changed requirement.

## Personnel-related requirements

Human error may occur in the loading of files to resume a game state. This error can include inputting a filename that does not exist, as well as inputting a file name whose contents are not in the expected format of saved game states. Upon receival of an improper filename or file, a message clearly defining the error shall be displayed to the user.

## Packaging requirements

The product shall be delivered as a Python program. The product shall also be delivered with a user manual detailing how to install and use the program.

## Precedence and criticality of requirements

Requirements criticality are specified with the “Rank” section underneath each requirement in Section 3.2

# Appendix A - Visual Prototype

