CMSC 447

Software Design Description (SDD)

|  |  |  |
| --- | --- | --- |
| Name | Role | Signature |
| Holly Bennett | Customer/Sponsor |  |
| Khaled Elgendy | POC, Student Developer |  |
| Rachael McKenzie | Student Developer |  |
| Ryan Miller | Student Developer |  |
| Aarti Patel | Student Developer |  |
| Connor Thomas | Student Developer |  |
| Jie Zhou | Student Developer |  |

Last Updated: 5/4/2019

[**1. Scope**](#_jcxy2je9lumr) **4**

[1.1 Identification](#_tr7k2dsr5b6a) 4

[1.2 System overview](#_ab69zbar93cj) 4

[1.3 Document overview](#_xekqiw3uhx6e) 4

[**2. Referenced Documents**](#_gsfbmv80vqa4) **4**

[**3. CSCI-wide design decisions**](#_mndcebqvmfq4) **4**

[3.1 Website](#_dv67bydfdm8i) 4

[3.2 Simulation](#_b2xjp21aow48) 5

[**4. CSCI architectural design**](#_ns6lgk8os99a) **5**

[4.1 CSCI components](#_fd4rusevs4hw) 5

[4.1.1 Website](#_jm5wmvhwitg8) 5

[4.1.2 File Parser](#_wnw4i7k5t7ru) 8

[4.1.3 Simulation](#_2gj638s099di) 8

[4.2 Concept of execution](#_35lxpv5ehpqr) 9

[4.2.1 Get user input(No particular order)](#_3ly4yq73e10a) 9

[4.2.1 Validate user input](#_jaf1wdpboooh) 9

[4.2.2 Iteration](#_486gu6xb4hl9) 10

[4.2.3 Termination](#_l3v9smwzsggo) 10

[4.3 Interface design](#_t3xo7ie14tnz) 10

[4.3.1 Message Box](#_emejvmsjxxz2) 10

[4.3.2 Drop Down Lists](#_uz9l6s57rho) 10

[4.4 Data Model](#_sdra0gv5uhla) 10

[4.4.1 Data structure](#_nzeaibyhu35l) 10

[**5. CSCI detailed design**](#_cidym3j3od9p) **11**

[5.1 surviveMin](#_9wvgt21m4a51) 11

[5.2 surviveMax](#_gejpnncaz5w1) 11

[5.3 revive](#_i49mph93xtf2) 11

[5.4 gridW](#_j7w8nkwm0786) 12

[5.5 gridH](#_m4zo87boxwuv) 12

[5.6 shapeSelect](#_ju28d8q9doeo) 12

[5.7 cellColor](#_em951fh81x5c) 12

[5.8 gridColor](#_p047jsc7vjn) 13

[5.9 time](#_mbv8271etrma) 13

[5.10 simDisplay](#_5705cmfd7v96) 13

[5.11 cellsAlive](#_4z8kxd1nfunu) 13

[5.12 iterationCount](#_7pamiu3hvxy1) 14

[5.13 startButton](#_d1eyga3jy8yo) 14

[5.14 fileInput](#_98p7z31z327f) 14

[5.15 iterationInput](#_7pc5ia5ds0bk) 14

[5.16 errors](#_yxscnjqa0unp) 14

[5.17 reset()](#_gqrjdts8mrji) 15

[5.18 inputValidation()](#_y9tzg762hh27) 15

[5.19 file upload listener](#_tgbku1b83wx2) 15

[5.20 togglePlay()](#_lflvgn7qg1mj) 15

[5.21 compare()](#_j5wgy9c9oi4l) 16

[5.22 life.next()](#_j0zr075coirq) 16

[5.23 activate()](#_k6ee1xwau1s9) 16

[5.24 Cell](#_bwd9wm7i08j0) 16

[5.25 Board](#_bnv49hqtfui) 17

[5.26 Life](#_2jrclc1bnuwg) 17

[**6. Requirements Traceability**](#_g74r75y9mbyp) **17**

[**7. Notes**](#_5yeuoh1vl1gw) **21**

[**A. Appendixes**](#_6rhjo1yybzg9) **22**

[A. Assessment of Alternatives](#_z15vc0cu7xb) 22

[B. UML Diagram](#_z15vc0cu7xb) 22

[C. Interface Design Diagram](#_kwq2bo9a6b45) 22

# **1. Scope**

## 1.1 Identification

This document applies to the latest version of the Conway’s Game of Life simulation developed by Team Segmentation Fault and sponsored by Holly Bennett. Currently the software is in the tool analysis and low-level design analysis stage of development.

## 1.2 System overview

The purpose of the software is to model Conway’s Game of Life, a cellular automaton that evolves based upon its initial state. The software will take the form of a website that is accessible and executable by a Chrome browser. It will possess the ability to adjust various factors of the simulation, such as initial state, factors for the cell survival, reproduction, death, and be able to identify and halt the simulation when a stable state is reached. The sponsor and user of the software is Holly Bennett, while the developers consist of Khaled Elgendy, Rachael McKenzie, Ryan Miller, Aarti Patel, Connor Thomas, and Jie Zhou, henceforth collectively referenced as Team Segmentation Fault or simply Segmentation Fault.

## 1.3 Document overview

The purpose of this document is to describe the CSCI-wide design decisions, the CSCI architectural design, and the detailed design needed to implement the software of the project, and to maintain a record of the traceability for each requirement.

# **2. Referenced Documents**

This document has appended an Assessment of Alternatives, UML diagramming to represent the system, and the final Interface Design Diagram.

# **3. CSCI-wide design decisions**

## 3.1 Website

The website shall have a variety of inputs conceptually separated into the major categories of rule manipulation, display manipulation, and interaction. Inputs regarding the manipulations of the rules shall use textboxes to allow the user to select values within a valid range. Manipulating the display speed and adjusting the cell shape will use drop down menus. Background color and cell color will be manipulated via color picker. The inputs responsible for interaction include a start checkbox, a file upload button, and a text field to enter a maximum number of iterations. The file upload button shall use a file finder to accept a file. The start button shall submit all of the specified rules and the input file to the simulation CSCI.

The website shall have three major outputs; displaying the simulation, displaying statistics about the game, and input validation. The simulation shall display on the specified grid size such that the cells are a constant size regardless of the grid size and are centered within the display area. The statistics regarding the number of cells that are alive and which iteration the game is on shall be displayed in text below the display area. If the simulation reports invalid input, error messages will display near the start button.

## 3.2 Simulation

The simulation will obtain the rules and starting coordinate input file as input from the website. If no input file is provided, then random values will be used to assign starting cell coordinates. The simulation will then run according to the rules provided until either the specified iterations have been reached, or else a stable state has been reached. When either of these situations occurs, a message will be output to the website describing which of the two situations occurred. In addition, the simulation will output each iteration to the website at the speed which it has been given, which will then be displayed by the website for the user.

# 4. CSCI architectural design

## 4.1 CSCI components

### 4.1.1 Website

The website will be hosted on a as yet to be determined server. It and all of its software components will be written in HTML, CSS, or Javascript as appropriate, unless indicated otherwise. It will consist of several other software components and interfaces, such as a file parser, in order to obtain input from the user. It will output this user data to the simulation, which will in turn provide input that the website will be responsible for displaying to the user as output. It will also consist of a message box that can display error messages and updates to the user as output.

Table 1. Table identifying software units, their purpose, associated requirements, and development status for the front end website. Reengineered should be interpreted to mean that an existing design or software will be used as a base and then will be reengineered.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Type | Unique Identifier | Purpose | Rqrmt | Development Status |
| Survive X Button | Drop down | surviveMin | Identify the minimum number of neighbors that each cell must have to survive. | 2.2.1  2.2.4 | Reengineered |
| Survive Y Button | Drop down | surviveMax | Identify the maximum number of neighbors that each cell can survive. | 2.2.2  2.2.4 | Reengineered |
| Revive X Button | Drop down | revive | Identify the minimum number of neighbors that each cell must have to transition from dead to live. | 2.2.3 | Reengineered |
| Revive Y Button | Drop down | reviveMax | Identify the maximum number of neighbors that a cell must have to transition from dead to live. | 2.2.3 | Reengineered |
| Grid Size X Button | Drop down | gridW | Identify the width of the grid for the simulation. | 2.2.8 | Reengineered |
| Grid Size Y Button | Drop down | gridH | Identify the height of the grid for the simulation. | 2.2.8 | Reengineered |
| Cell Shape Button | Drop down | shapeSelect | Identify the shape of the cells that the website displays. | 2.2.7 | Reengineered |
| Cell Color Button | Color Picker | cellColor | Identify the color of the cells that the website displays. | 2.2.6 | Reengineered |
| Background Color Button | Color Picker | gridColor | Identify the background color of the grid for the simulation. | 2.2.5 | Reengineered |
| Speed selector button | Drop down | time | Adjust the speed at which the simulation displays. | 2.2.13 | Reengineered |
| Simulation Display | Visual | simDisplay | Display the output of the simulation. | 2 | Reengineered |
| Living Cell Display | Text | cellsAlive | Display the number of cells that are alive. | 2.3.2 | Reengineered |
| Iteration Display | Text | iterationCount | Display the number of iterations that have passed. | 2.3.1 | Reengineered |
| Start button |  | startButton | Pass the input to the simulation and initiate the display. | 2 | Reengineered |
| File upload button | File picker | fileInput | Allow the user to specify a starting state for the simulation. | 2.2.11 | Reengineered |
| Iteration count box | Text submission | iterationInput | Allow the user to specify a maximum number of iterations for the simulation. | 2.2.10 | Reengineered |
| Error display | Text display | errors | Specifies errors that occur during input. | 2.2.9 | Reengineered |
| New button | Button | reset() | Takes the current parameters to generate a new simulation. | 2.2.1 - 2.2.4 | Reengineered |

### 4.1.2 Simulation

The simulation will accept input from the website in the form of rules and from the file parser in the form of starting coordinates. It will use a combination of these rules and preprogrammed rules to run a simulation, then output the results to the website after each iteration. If an error occurs, or if the simulation concludes (either when a stable state is reached, oscillation state is reached, or the number of iterations is reached), a message will be sent as output to the message box of the website. It will be written in Javascript.

Table 2. Table identifying software units, their purpose, associated requirements, and development status for the simulation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Unique Identifier | Purpose | Rqrmt | Development Status |
| Rule validation | inputValidation() | Ensure that the user’s rules can be implemented. If the rules are valid, set the rules for the simulation. | 2.2  2.3 | New development |
| Coordinate validation | file upload listener | Read the file provided by the user and validate against acceptable inputs. | 2.2.11 | Reengineered |
| Autorun | togglePlay() | Cause the simulation to run and manage the failure conditions. | 2.2.9 | Reengineered |
| Compare Lists | compare() | Accepts two lists of cells and returns a boolean if they are the same. | 2.4.2 | New development |
| Step | life.next() | Process one iteration of the simulation and return an indication of if any changes were made. | 2.2.9 | Reengineered |
| Default | activate() | Present a default set of parameters when the page first loads. | 2.2.9 | Reengineered |
| Cell | Cell | An entity to represent each state on the board. Also has helper methods to facilitate generating the next iteration. | 2.2.9 | Reengineered |
| Board | board | A method to generate new blank boards of a specified size. | 2.2.9 | Reengineered |
| Life | life | An entity which contains the primary methods to generate a new iteration of the simulation. | 2.2.9 | Reengineered |

## 4.2 Concept of execution

### 4.2.1 Get user input(No particular order)

* Upload text file
* Enter grid height and width
* Modify cell shape (default circle)
* Modify background color (default white)
* Modify cell color (default black)
* Modify speed of iteration
* Modify survive parameters
* Modify revive parameters
* Enter number of iterations

### 4.2.1 Validate user input

When the user tries to execute the Start command an error will be displayed if user input is invalid or triggers an error state. Error states are as follows:

* Invalid file type or no file is uploaded
* Coordinates in user file are outside the boundaries of the indicated grid-size
* Background and cell color are the same
* Number of iteration is outside of the range (1,216000) inclusive

### 4.2.2 Iteration

1. The program will execute if no error state is triggered.
   1. Each new table will be compared against the previous one, except for the first, to check for a steady state. If steady state is reached the program will stop and display that a steady state has been reached.
   2. Each new table will be compared against the previous two, except for the first two, to check for an oscillation state. If an oscillation state is reached the program will stop and display that an oscillation has been reached.
2. Each new table will be displayed on screen along with the iteration number and the number of live cells

### 4.2.3 Termination

If a steady state or oscillation is not reached then the program will run until the number of iterations indicated by the user. The program will stop and the final frame will be displayed along with the iteration count and the number of living cells.

## 4.3 Interface design

### 4.3.1 Message Box

The message box will accept input from the file validation and rule validation units. This box will be used to display error messages based on those inputs.

### 4.3.2 Drop Down Lists

These lists will accept input from the user, specifically by allowing them to choose from a predetermined range of values. This choice will then be sent as output to the simulation.

## 4.4 Data Model

### 4.4.1 Data structure

For this project we will be using arrays of cells as our main data-structure for computing the currently living cells and for storing previous generations used for steady-state and oscillation detection

Table 3: Table identifying data-structures used in the backend. Table shows name, unique identifier, and purpose.

|  |  |  |
| --- | --- | --- |
| Name | Unique Identifier | Purpose |
| Current generation | board | Used to store the newly determined living nodes in the current iteration(n). |
| Parent Generation | parent | Stores the previous iterations(n-1) cells. This generation is stored so that it can be compared to the next generation (n+1). |
| Grandparent Generation | grandparent | Stores the n-2 generations cells. This will be compared against the current generation (board) for oscillation detection. |

# 5. CSCI detailed design

## 5.1 surviveMin

The surviveMin element designates the lower bound on the number of living neighbors a cell must have to stay alive. It is implemented as an HTML input element of type text to allow the users to input the lower bound themselves. By default, the element will hold the value ‘2’. This lower bound will be stored so that life.next() (5.22) can use it to define the rules for iteration. The input is validated through inputValidation() (5.18) to ensure that surviveMin’s value is an integer.

If the user tries to run the game with a value that is not an integer, the simulation will not run, and an error will tell the user that the must input an integer for the minimum survival number.

## 5.2 surviveMax

The surviveMax element determines the upper bound on the number of living neighbors a cell must have to stay alive. It works nearly identically to surviveMin (5.1). The only difference is that it has a default value of ‘3’. It is sent to inputValidation() (5.18) as the upper bound on survival in life.next() (5.22). It works identically to surviveMin outside of those differences.

## 5.3 revive

The revive element sets the number of living neighbors a cell must have to transition from dead to live. Its implementation is nearly identically to surviveMin (5.1). It varies in that its default value is ‘3’. It is sent to inputValiation() (5.18) as the value necessary to revive a cell in life.next() (5.22). Like surviveMin and surviveMax (5.2), error handling will verify that it is an integer and provide an error message if it is not.

## 5.4 gridW

The gridW element is used to determine the number of columns that the simDisplay (5.10) has. It is implemented as an HTML input element of type text, meaning it will allow the user to enter in any text to the input element on the screen. This input will be stored and used to help create the board that simDisplay will use when displaying the game. The user is expected to input an integer that has a lower bound of 1 and no upper bound. Before creating the board, the website checks for whether the input follows these rules. To ensure this, the website first checks if the input is an integer. If the input is an integer, the website verifies that it is in the interval [1, 100]. If both those checks are successful, then simDisplay is updated to have the desired number of columns. If either check fails, an error is displayed to the user telling them that they must input an integer with a value in the interval [1, 100] and the number of columns displayed returns to the default value of 15.

## 5.5 gridH

The gridH element sets the number of rows that the simDisplay (5.10) has. It is implemented identically to gridW (5.4) except it designates the number of rows that the board will have. It also verifies that the value provided by the user is an integer on the range [1, 40].

## 5.6 shapeSelect

The shapeSelect element is a drop-down menu that determines the shape each living cell in the board is displayed as. It has three options: a sphere, a square, and a triangle (by default, sphere will be selected). The drop-down menu will be displayed as an HTML select element with three options (one for each shape). The shapes will be displayed as a Unicode character. The Unicode values for each shape will be stored with each respective shape’s option in the drop-down menu (Sphere: “&#9679;”, Square: “&#9632;”, Triangle: “&#9650”). The simDisplay (5.10) determines when and where the living cell will be displayed.

## 5.7 cellColor

The cellColor element is used to identify the color of the living cells. It is implemented with an HTML input element of type color. This element provides a user interface that lets the user choose a color by visually choosing one or inputting RGB values, and then returning a hexadecimal version of the RGB color. This hexadecimal color is then sent to the simDisplay (5.10) to be used to set the color of the Unicode characters. This HTML element will always return an accurate hexadecimal RGB color, so no error checking is required.

## 5.8 gridColor

The gridColor element is used to identify the background color for the simDisplay (5.10). It works similarly to cellColor in nearly every respect. It too uses an HTML input element of type color to get hexadecimal color values from the user, but instead it sets background color for the board in simDisplay.

## 5.9 time

The time element allows the user to select how long of a delay is added between each iteration. The user is given three options: slow, normal and fast. Normal, the default option, makes the simulation wait 300ms between each iteration, slow makes it wait 1000s, and fast makes it wait 100ms between iterations. The user can select the speed through a drop-down menu, which is implemented through the HTML select element with the three options described above. This value is utilized in the “$interval” AngularJS function to set the frequency of how often life.next() (5.22) is called. The user is only able to make changes to the speed while the simulation is paused. Any attempt to change the speed while the simulation is running is ignored.

## 5.10 simDisplay

The simDisplay element is the display of the simulation. It stores a 2d array of each cell, indexed by its row and column, current status (that being alive or dead). This array is referred to as the board. The board is displayed and interacted with by the user with a HTML table element. This table, and board, have the dimensions specified by gridW (5.4) and gridH (5.6). The table has the background color specified by the hexadecimal value given by backgroudColor (5.8) and displays every alive cell with the Unicode character determined by the Unicode value given by shapeSelect (5.6). That Unicode character has the color determined by hexadecimal value given by cellColor (5.7). It detects alive cells by taking the current row and column and examining that position in the board for its status.

Each time the board is changed, the display is updated. The board can be changed by running the simulation. Each simulation, the board is sent to life.next() (5.22) and life.next() updates the board. It can be given a valid file of alive cells through fileInput (5.14) and create a new board based off that file.

## 5.11 cellsAlive

The cellsAlive element displays the number of living cells at the current iteration in the simulation. It is implemented as a variable that is updated each time life.next() (5.22) is called, and each time the user adds new living cells. This value is displayed by printing out the variable to the screen below, and keeping the variable updated for each iteration.

## 5.12 iterationCount

The iterationCount element displays the number of iterations that the simulation has processed to the user. It is implemented as a variable that is initialized to zero and is iterated by one each time life.next() (5.22) is called. If the board is ever reset, the variable is set back to zero. This variable is printed out below the simDisplay (5.10) and updated each iteration.

## 5.13 startButton

The startButton element is a checkbox that the user can select to start or continue the simulation. It is implemented as an HTML input element of type checkbox, which, when checked, calls a function to begin calling life.next() (5.22). The user can toggle the box as many times as they wish to pause and restart the simulation.

## 5.14 fileInput

The fileInput element allows the user to select a .txt file that describes the currently alive cells on the board. The file is accepted through the HTML input element of type file. It will only accept .txt and will not allow the user to choose anything but a .txt file. Once the file is selected, it is sent to inputValidation (5.18) to be validated and applied to the board.

## 5.15 iterationInput

The iterationInput element allows the user to select the maximum number of iterations for the current simulation. Its default and max value are both 216,000. When the user tries to input a value of less than or equal to zero or greater than 216,000, the input is prevented and an error is displayed to tell the user they must input a value in the range of (0, 216,000]. The user may input the value through an HTML input element of type text. The value is then compared to the current iteration number before each life.next() (5.22).

## 5.16 errors

The errors element displays any text to explain why the simulation is not running. It will display errors if the user inputs invalid data. It will display errors if the input coordinates are not coordinates or if they are malformed coordinates. It will also display errors if the user enters a non-integer value into any of the text inputs. It will also display a message if the simulation has reached a steady state.

## 5.17 reset()

The reset() element is responsible for preparing the simulation for a set of new settings. It resets iterationCount and cellsAlive to 0 and resets any error messages. After that, it triggers inputValidation() and creates a new board with the user specified rules.

## 5.18 inputValidation()

The inputValidation() function ensures that the rules the user defines are valid. In this case, valid means that the simulation is capable of being run and there are no impossible rules to implement. The function determines that all of the textbox inputs except iterationInput are integers and returns an appropriate error message to identify a field which is not an integer. Additionally, the function verifies that the grid sizes are between [1,100] horizontally and [1,40] vertically.

## 5.19 file upload listener

The file upload listener ensures that the input file received from fileInput (5.14) contains valid input. The file upload listener should trigger when a file is successfully uploaded. After the file is read, the listener should call reset() and end the game, if it is currently running. Failing to do so would cause unpredictable behaviors. Next, the listener should create a new board via methods in board. After that, it should parse the text of the file. It will generate an error message to the user if the text is not integers, has malformed coordinates, and if the coordinates are outside the specified grid. If no errors are generated, the listener will set each coordinate to alive. If no errors are generated for any coordinate, it will then set the newly generated board as the current board.

The format the file parser expects is that each line in the file specifies a row, followed by a comma, followed by the column. In addition to that, it will expect that the first column is the leftmost and begins at zero and the first row is the topmost and begins at zero. Below is an example of what the file would contain if the user would want to place an alive cell in the top most row and left most column and another alive cell at the 6th row from top and the 7th column from the left.

1,1

5,6

## 5.20 togglePlay()

The togglePlay() function will manage the end conditions and how many iterations the simulation will run. It will first ensure that if the simulation is running and it should not (e.g. if isStarted (5.13) is no longer checked) be, that it is terminated. Next it will validate that the iterationInput (5.15) is an integer between 1 and 216000. It will empty the current fileInput (5.14) and begin running the simulation for the specified number of iterations with the delay specified by time (5.9). After the first iteration, the function will preserve a copy of the grandparent generation and for each generation, it will preserve a copy of the parent generation. Then it will call life.next() (5.22), effectively advancing the board state one iteration. Then it will call compare (5.21) to verify that the grandparent generation is not identical to the current generation. Since life.next() returns how many cells are currently alive or -1 if none have changed, it’s output can be used to verify whether the game is in a stable state. If the game is in a stable state, the rest of the iterations are canceled.

## 5.21 compare()

The compare() function does a deep comparison of two generations of board states. For each cell, it verifies that they are either both alive or dead. The first time that a difference is detected, the function returns a 0 and if no differences are detected, it returns a 1.

## 5.22 life.next()

life.next() processes one iteration of the simulation based off the current board and returns an updated board. It determines what the next iteration will be based off the rules defined in surviveMin (5.1), surviveMax (5.2), and revive (5.3). It returns the number of cells that are currently alive or -1 if none of the cells changed states from the last generation.  
 The next board state is generated by calculating how many alive neighbors each cell had in the previous board and comparing it to the specified rules in a helper function (life.newCellState). If the cell was not alive in a past iteration and is now alive, the current number of alive cells is incremented and a change flag is set. If the cell was alive in the past and is not in the current generation, the current number of living cells is decremented and a change flag is set. When all cells have been processed, if the change flag has not been set, -1 is returned because the board is in a stable state.

## 5.23 activate()

The activate() function generates a board using default values so that a grid is displayed when the page loads.

## 5.24 Cell

The Cell entity contains a representation of each cell on the board and a helper method called getAliveNeighbors. This helper function calculates how many neighbors the cell has alive by examining each of its current neighbors and returns that value. The Cell entity also contains a couple of methods to set the current state of the cell to either alive or dead.

## 5.25 Board

The Board entity contains one method called createNew that takes two integer parameters and generates a new board with those dimensions. Specifically, the first parameter is the horizontal size and the second is the vertical size.

## 5.26 Life

The Life entity contains two methods. The first, next(), is detailed above (5.22) due to the integral nature it has to the simulation and the second is newCellState. The second function takes a reference to a previous board state, and a coordinate. It uses this information to compute whether the cell is in question should be alive. It calculates this by comparing the number of neighbors it has to the minimum and maximum number to survive or the number to revive, as is appropriate for its status in the previous board.

# 6. Requirements Traceability

Table 3. Table identifying each of the software unit’s traceability to the CSCI requirements allocated to it.

|  |  |
| --- | --- |
| Name | CSCI Requirement |
| Survive X Textbox | 2.1 Conway’s Game of Life Rules and Implementation |
| Survive Y Textbox | 2.1 Conway’s Game of Life Rules and Implementation |
| Revive X Textbox | 2.1 Conway’s Game of Life Rules and Implementation |
| Grid Size X Textbox | 2.2.8 Parameter Tuning |
| Grid Size Y Textbox | 2.2.8 Parameter Tuning |
| Cell Shape Button | 2.2.7 Parameter Tuning |
| Cell Color Button | 2.2.6 Parameter Tuning |
| Background Color Button | 2.2.5 Parameter Tuning |
| Speed selector button | 2.2.12 Parameter Tuning |
| Simulation Display | 2.3 Data Display |
| Living Cell Display | 2.3 Data Display |
| Iteration Display | 2.3 Data Display |
| Start button | 2.11 Parameter Tuning |
| File upload button | 2.2.8 Parameter Tuning |
| Iteration count box | 2.3 Data Display |
| Error Display | 2.2.9 Parameter Tuning |
| New Button | 2.2.1 - 2.2.4 Parameter Tuning |
| Rule validation | 2.1 Conway’s Game of Life Rules and Implementation |
| Coordinate validation | 2.2.11 Parameter Tuning |
| Autorun | 2.1 Conway’s Game of Life Rules and Implementation |
| Compare Lists | 2.4 Stable State Detection |
| Step | 2.1 Conway’s Game of Life Rules and Implementation |
| Default | 2.1 Conway’s Game of Life Rules and Implementation |
| Cell | 2.1 Conway’s Game of Life Rules and Implementation |
| Board | 2.1 Conway’s Game of Life Rules and Implementation |
| Life | 2.1 Conway’s Game of Life Rules and Implementation |

# 

Table 4. Table identifying traceability from each CSCI requirement to the software units to which it is allocated.

|  |  |
| --- | --- |
| CSCI Requirement | Software Unit |
| 1.1 Chrome Accessibility | N/A |
| 1.2 Mac Accessibility | N/A |
| 2.1.1 The game shall implement the rule, by default, “If a living cell has 1 or 0 living neighbors, it will die from solitude.” | Rule validation |
| 2.1.2 The game shall implement the rule, by default, “If a living cell has 4 or more living neighbors, it will die from overpopulation.” | Rule validation |
| 2.1.3 The game shall implement the rule, by default, “If a dead cell has 3 living neighbors, it will be revived.” | Rule validation |
| 2.1.4 The game shall implement the rule, by default, “If a living cell has 2 or 3 neighbors, it survives.” | Rule validation |
| 2.1.5 The game shall be implemented on a square grid. | Grid Size X Textbox  Grid Size Y Textbox |
| 2.1.6 The game shall display in a 1080p resolution. | Simulation Display |
| 2.1.7 The game shall display nodes at a fixed size, regardless of the size of the world. | Simulation Display |
| 2.1.8 The game shall feature nodes which are visually distinguishable from one another. | Simulation Display |
| 2.1.9 The game shall parse grid coordinates provided by a user as cells that are living at the start of the game. | Input Validation |
| 2.2.1 The website shall implement the ability to adjust the number of neighbors for a cell to die from solitude. | Survive X Textbox  Survive Y Textbox |
| 2.2.2 The website shall implement the ability to adjust the number of neighbors for a cell to die from overpopulation. | Survive X Textbox  Survive Y Textbox |
| 2.2.3 The website shall implement the ability to adjust the number of neighbors for a cell to be revived. | Revive Textbox |
| 2.2.4 The website shall implement the ability to adjust the number of neighbors for a cell to survive. | Survive X Textbox  Survive Y Textbox |
| 2.2.5 The website shall implement the ability to adjust the background color of the game using 16 bit colors. | Background Color Button |
| 2.2.6 The website shall implement the ability to adjust the color of cells using 16 bit colors. | Cell color button |
| 2.2.7 The website shall implement the ability to select the shape of the cells. The default shape shall be circles. Additional selectable shapes shall include triangles and squares. | Cell shape button |
| 2.2.8 The website shall implement the ability to change the size of the grid in the game. | Grid Size X Textbox  Grid Size Y Textbox |
| 2.2.9 The website shall run the game to a maximum of 216,000 iterations. | Simulation |
| 2.2.10 The website shall implement the ability to change the default number of maximum iterations to a value between 1 and 216,000. | Iteration Count Textbox |
| 2.2.11 The website shall accept a .txt text file containing grid coordinates as input for the game. | Start Checkbox |
| 2.2.12 The website should be able to run the game at multiple speeds. | Speed Selector Button |
| 2.2.13 The website should allow the user to adjust the speeds between selectable options. | Speed Selector Button |
| 2.3.1 The website shall display how many iterations the game has been running. | Iteration Display |
| 2.3.2 The website shall display a count of how many cells are alive during each iteration. | Living Cell Display |
| 2.4.1 The game shall stop if it is in the same state for two iterations. | compare()  togglePlay()  life.next() |
| 2.4.2 The game shall stop if it is oscillating between two states. | compare()  togglePlay()  life.next() |

# 

# 7. Notes

# A. Appendixes

## A. Assessment of Alternatives

## B. UML Diagram

## C. Interface Design Diagram