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: 4/23/2019

[**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) **5**

[3.1 Website](#_dv67bydfdm8i) 5

[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) 9

[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 reviveMin](#_i49mph93xtf2) 11

[5.4 gridHorizontal](#_j7w8nkwm0786) 12

[5.5 gridVertical](#_m4zo87boxwuv) 12

[5.6 cellShape](#_ju28d8q9doeo) 12

[5.7 cellColor](#_em951fh81x5c) 12

[5.8 backgroundColor](#_p047jsc7vjn) 13

[5.9 speedSelect](#_mbv8271etrma) 13

[5.10 simDisplay](#_5705cmfd7v96) 13

[5.11 cellCountDisplay](#_4z8kxd1nfunu) 13

[5.12 iterationDisplay](#_7pamiu3hvxy1) 14

[5.13 startButton](#_d1eyga3jy8yo) 14

[5.14 fileInput](#_98p7z31z327f) 14

[5.15 iterationInput](#_7pc5ia5ds0bk) 14

[5.16 rulesValidation()](#_bqluxo4kgfwz) 14

[5.17 inputValidation()](#_tgbku1b83wx2) 15

[5.18 step()](#_lflvgn7qg1mj) 15

[5.19 getList()](#_k6ee1xwau1s9) 15

[5.20 isEquals()](#_46vj7imro4gq) 16

[**6. Requirements Traceability**](#_g74r75y9mbyp) **16**

[**7. Notes**](#_sa0i407prh8j) **16**

[**A. Appendixes**](#_6rhjo1yybzg9) **17**

[A. Assessment of Alternatives](#_z15vc0cu7xb) 17

[B. UML Diagram](#_z15vc0cu7xb) 17

[C. Interface Design Diagram](#_kwq2bo9a6b45) 17

# **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 a proposed Interface Design Diagram. (Some of these may be required and thus not included here?)

# **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 drop down lists to allow the user to select values within a valid range. Manipulating the display speed and adjusting the cell shape will also use drop down menus. Background color and cell color will be manipulated via color picker. The inputs responsible for interaction include a start button, a file upload button, and text field to enter a maximum number of iterations. The file upload button shall use a file finder to accept a .txt file and provide an error notification if any other file extension is provided. 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 | reviveMin | 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 | gridHorizontal | Identify the width of the grid for the simulation. | 2.2.8 | Reengineered |
| Grid Size Y Button | Drop down | gridVertical | Identify the height of the grid for the simulation. | 2.2.8 | Reengineered |
| Cell Shape Button | Drop down | cellShape | 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 | backgroundColor | Identify the background color of the grid for the simulation. | 2.2.5 | Reengineered |
| Speed selector button | Drop down | speedSelect | 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 | cellCountDisplay | Display the number of cells that are alive. | 2.3.2 | Reengineered |
| Iteration Display | Text | iterationDisplay | 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 |

### 4.1.2 File Parser

This component of the website will accept a .txt file as input from the user, then will parse the contents of the file. When the simulation is started the input will be validated . If the input is judged to be invalid, points are outside the range of the grid, an error message will be sent as output to the message box. It will be written in Javascript.

### 4.1.3 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 | rulesValidation() | Ensure that the user’s rules can be implemented. If the rules are valid, set the rules for the object. | 2.2  2.3 | New development |
| Input validation | inputValidation() | Ensure that the input file contains valid coordinates and does not contain garbage input. If the input is valid, initialize the state of the simulation. | 2.2.11 | New development |
| Step | step() | Process one iteration of the simulation and return an indication of if any changes were made. | 2.4.1 | Reengineered |
| Get Live List | getList() | Get the list of living cells | 2.4.2  2.3 | New development |
| Compare Lists | isEquals() | Accepts a list of cells and returns a boolean if they are the same. | 2.4.2 | New development |

## 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 hashmaps 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 | Development Status |
| Current generation | currGen | Used to store the newly determined living nodes in the current iteration(n). Key=row, Value=column | New development |
| Parent Generation | parentGen | Stores the previous iterations(n-1) living cells. Key=row, Value=column  This will be compared against the currGen for steady state detection | New development |
| Grandparent Generation | grandGen | Stores the n-2 generations living cells. This will be compared against the current generation (CurrGen) for oscillation detection | New development |

# 5. CSCI detailed design

## 5.1 surviveMin

surviveMin 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 step() (5.18) can use it to define the rules for iteration. The input is validated through ruleValidation() (5.16) to ensure that surviveMin’s value doesn’t conflict with any other rules.

There is also error handling to ensure that the inputted value is in fact an integer. It is done by utilizing javascript functions which allow the programmer to convert a string to some number format and check if that number is an integer or not. If the string the user inputs has no decimal places and no characters, it will determine that this is an integer and allow the simulation to run. 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

surviveMax 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 differences are: it has a default value of ‘3’, it is sent to ruleValidation() (5.16) in such a way so it knows it is the upper bound on survival and step() (5.18) will use it as an upper bound on survival. It works identically to suriveMin outside of those differences.

## 5.3 reviveMin

reviveMin sets the lower bound on 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 ruleValiation() (5.16) in such a way so it knows it is the upper bound on revival and step() (5.18) will use it as the lower bound on revival. Like surviveMin and surviveMax (5.2), it does the same error handling.

## 5.4 gridHorizontal

gridHorizontal 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 by attempting to convert the inputted string into an int and checking for success. After that check, if the input is an integer, it is checked if it is less than 1. 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 greater than or equal to zero and the number of columns displayed remain unchanged.

## 5.5 gridVertical

gridVeritcal sets the number of rows that the simDisplay (5.10) has. It is implemented identically to gridHorizontal (5.4) except it designates the number of rows that the board simDisplay creates will have. It has the same error checking as gridHorizontal too.

## 5.6 cellShape

cellShape 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

cellColor 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 backgroundColor

backgroundColor 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 speedSelect

speedSelect allows the user to select the speed at which the simulation will execute. The user is given three options: slow, normal and fast. Slow, the default option, makes the simulation run at 1Hz, normal makes it run at 3.33Hz, and fast makes it run at 10Hz. 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. Each option holds the number of milliseconds in between steps. This value is utilized in the “$interval” AngularJS function to set the frequency of how often step() (5.18) is called per second. 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 and an errors indicating that the user may not do this is displayed.

## 5.10 simDisplay

simDisplay is the interactive 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 gridHorizontal (5.4) and gridVertical (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 cellShape (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 and updated in three ways. First, by running the simulation. Each simulation, the board is sent to step() (5.18) and step returns the new board. This new board replaces the old one. Next, it can be given a valid file of alive cells through fileInput (5.14) and create a new board based off that file. Lastly, each cell in the table can be clicked on, which toggles the element in the board with that cells row and column from alive to dead or vice versa.

## 5.11 cellCountDisplay

cellCountDisplay displays the number of living cells at the current iteration in the simulation. It is implemented as a variable that is updated each time step() (5.18) 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 iterationDisplay

iterationDisplay displays the number of iterations the simulation is currently at to the user. It is implemented as a variable that is initialized to zero and is iterated by one each time step() (5.18) 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

startButton 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 step() (5.18). The user can toggle the box as many times as they wish to pause and resume the simulation, except for in the case where the max iteration is reached. More information about this case is described in section 5.15.

## 5.14 fileInput

fileInput 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 (5.17) to be validated and applied to the board.

## 5.15 iterationInput

iterationInput 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 step() (5.18). If the current iteration number is equal to the max iteration value the user inputted, the simulation is paused and the startButton (5.13) is restricted from being toggled until the simulation is reset.

## 5.16 rulesValidation()

rulesValidation() is utilized to ensure 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 rules the user defines are collected by surviveMin (5.1), surviveMax (5.2) and reviveMin(5.3). rulesValidation() checks for two things. First, it makes sure surviveMin, surviveMax, and reviveMin are all greater than or equal to 0. If anyone of them is not, it returns an error specifying which one is less than zero, and it will untoggle the startButton (5.13) to prevent the simulation from running. Second, it checks to ensure that suviveMin is not greater than surviveMax. If surviveMin is greater than surviveMax, then an error is returned stating this and, again, it will untoggle the startButton to prevent the simulation from running.

## 5.17 inputValidation()

inputValidation() ensures that the input file received from fileInput (5.14) contains valid input. If the input is valid, it gets the board from the simDisplay (5.10), sets all cells to dead, and sets the cells the input file says are alive as alive. If they are not valid, the board is not updated and an error describing why its invalid is displayed. Two errors are possible. That being if it does not follow the format that will be described in the forthcoming paragraph, or the file specifies a column or row greater than the number of columns or rows.

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 one and the first row is the bottommost and begins at one. Below is an example of what the file would contain if the user would want to place an alive cell in the bottom most row and left most column and another alive cell at the 5th row from bottom and the 6th column from the left.

1,1

5,6

## 5.18 step()

Step() 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 reviveMin (5.3). Along with this, it examines the current iteration number and the max iteration number as defined by iterationInput (5.15) and prevent the step from occurring if they are equal. If they are not equal, the simulation continues, that being updating the board and iterating the number of iterations by 1.

## 5.19 getList()

getList() returns a list of the living cells. It loops through the current board and create a list that contains the row and column of each living cell. It is used for the purpose of steady state detection. It acts as the portion of the steady state detections which gets a representation of the board at multiple iterations.

## 5.20 isEquals()

isEqual() accepts two lists of cells and returns a boolean that tells whether the two states are the same. Similar to getList() (5.19), it is used for the purpose of steady state detection. It acts as the way to compare boards to see if the simulation ever returns to the same state.

# 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 Button | 3.2.1 Conway’s Game of Life Rules and Implementation |
| Survive Y Button | 3.2.1 Conway’s Game of Life Rules and Implementation |
| Revive X Button | 3.2.1 Conway’s Game of Life Rules and Implementation |
| Revive Y Button | 3.2.1 Conway’s Game of Life Rules and Implementation |
| Grid Size X Button | 3.2.2.8 Parameter Tuning |
| Grid Size Y Button | 3.2.2.8 Parameter Tuning |
| Cell Shape Button | 3.2.2.7 Parameter Tuning |
| Cell Color Button | 3.2.2.6 Parameter Tuning |
| Background Color Button | 3.2.2.5 Parameter Tuning |
| Speed selector button | 3.2.2.12 Parameter Tuning |
| Simulation Display | 3.2.3 Data Display |
| Living Cell Display | 3.2.3 Data Display |
| Iteration Display | 3.2.3 Data Display |
| Start button | 3.2.11 Parameter Tuning |
| File upload button | 3.2.2.8 Parameter Tuning |
| Iteration count box | 3.2.3 Data Display |
| File Parser | 3.2.2.8 Parameter Tuning |
| Simulation | 3.2.1 Conway’s Game of Life Rules and Implementation  3.2.2.11 Parameter Tuning |
| Rule validation | 3.2.1 Conway’s Game of Life Rules and Implementation |
| Input validation | 3.2.2.11 Parameter Tuning |
| Step | 3.2.1 Conway’s Game of Life Rules and Implementation |
| Get Live List | 3.2.4 Stable State Detection |
| Compare Lists | 3.2.4 Stable State Detection |

# 

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

# 

|  |  |
| --- | --- |
| CSCI Requirement | Software Unit |
| 3.1.1 Chrome Accessibility | N/A |
| 3.1.2 Mac Accessibility | N/A |
| 3.2.1.1 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 |
| 3.2.1.2 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 |
| 3.2.1.3 The game shall implement the rule, by default, “If a dead cell has 3 living neighbors, it will be revived.” | Rule validation |
| 3.2.1.4 The game shall implement the rule, by default, “If a living cell has 2 or 3 neighbors, it survives.” | Rule validation |
| 3.2.1.5 The game shall be implemented on a square grid. | Grid Size X Button  Grid Size Y Button |
| 3.2.1.6 The game shall display in a 1080p resolution. | Simulation Display |
| 3.2.1.7 The game shall display nodes at a fixed size, regardless of the size of the world. | Simulation Display |
| 3.2.1.8 The game shall feature nodes which are visually distinguishable from one another. | Simulation Display |
| 3.2.1.9 The game shall parse grid coordinates provided by a user as cells that are living at the start of the game. | Input Validation |
| 3.2.2.1 The website shall implement the ability to adjust the number of neighbors for a cell to die from solitude. | Survive X Button  Survive Y Button |
| 3.2.2.2 The website shall implement the ability to adjust the number of neighbors for a cell to die from overpopulation. | Survive X Button  Survive Y Button |
| 3.2.2.3 The website shall implement the ability to adjust the number of neighbors for a cell to be revived. | Revive X Button  Revive Y Button |
| 3.2.2.4 The website shall implement the ability to adjust the number of neighbors for a cell to survive. | Survive X Button  Survive Y Button |
| 3.2.2.5 The website shall implement the ability to adjust the background color of the game using 16 bit colors. | Background Color Button |
| 3.2.2.6 The website shall implement the ability to adjust the color of cells using 16 bit colors. | Cell color button |
| 3.2.2.7 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 |
| 3.2.2.8 The website shall implement the ability to change the size of the grid in the game. | Grid Size X Button  Grind Size Y Button |
| 3.2.2.9 The website shall run the game to a maximum of 216,000 iterations. | Simulation |
| 3.2.2.10 The website shall implement the ability to change the default number of maximum iterations to a value between 1 and 216,000. | Iteration Count Box |
| 3.2.2.11 The website shall accept a .txt text file containing grid coordinates as input for the game. | Start Button |
| 3.2.2.12 The website should be able to run the game at multiple speeds. | Speed Selector Button |
| 3.2.2.13 The website should allow the user to adjust the speeds between selectable options. | Speed Selector Button |
| 3.2.3.1 The website shall display how many iterations the game has been running. | Iteration Display |
| 3.2.3.2 The website shall display a count of how many cells are alive during each iteration. | Living Cell Display |
| 3.2.4.1 The game shall stop if it is in the same state for two iterations. | getList  isEquals |
| 3.2.4.2 The game shall stop if it is oscillating between two states. | getList  isEquals |

# 

# 7. Notes

Conceptual Flow:

1. Get input from user
2. Validate user input
   1. What are we considering invalid input? (please provide some input about how you perceive this)
      1. For cells to revive (X-Y); Y-X < 0 means none will revive. Is this an error?
      2. For cells to survive (X-Y); Y-X < 0 means none will survive. Is this an error?
      3. If both are < 0, is that an error?
      4. For the first two cases, there are use cases, so do we want to consider a warning instead of a hard error?
3. Validate start file into state object
   1. Are duplicates errors?
   2. Coordinates outside the grid will cause an error state and an error message will be displayed to the user.
   3. Invalidly formed coordinates
      1. Either things that do not contain numbers
      2. Or things that contain too many/few numbers
4. Step time
   1. Change flag = 0
   2. Compute neighbors
   3. Apply rules for each node
   4. Change flag = 1 if any change
5. Iteration check to terminate
6. State check & Oscillation check to terminate
   1. Check change flag (If no change, display final state and terminate)
   2. Check if state n == state n - 2 (if no change, display final state and terminate)
7. Display new state
   1. New state is displayed
8. Repeat from 5 if no steady state or oscillation was detected

Data Model:

1. Use a map with keys (rows) and values (columns) to represent only the cells that are alive.
2. For cell computations, make a map with map1key, map1value tuples as the key and the number of living neighbors as a value.
3. Update map1 with the calculations from rules + map2.

I think this way should be faster and lighter than trying to represent using an array

# A. Appendixes

## A. Assessment of Alternatives

## B. UML Diagram

## C. Interface Design Diagram