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: 4/4/2019

[**1. Scope**](#_jcxy2je9lumr) **3**

[1.1 Identification](#_tr7k2dsr5b6a) 3

[1.2 System overview](#_ab69zbar93cj) 3

[1.3 Document overview](#_xekqiw3uhx6e) 3

[**2. Referenced Documents**](#_gsfbmv80vqa4) **3**

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

[3.1 Website](#_dv67bydfdm8i) 4

[3.2 Simulation](#_b2xjp21aow48) 4

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

[4.1 CSCI components](#_fd4rusevs4hw) 4

[4.1.1 Website](#_jm5wmvhwitg8) 4

[4.1.2 File Parser](#_wnw4i7k5t7ru) 7

[4.1.3 Simulation](#_2gj638s099di) 7

[4.2 Concept of execution](#_35lxpv5ehpqr) 8

[4.3 Interface design](#_t3xo7ie14tnz) 8

[4.3.1 Message Box](#_emejvmsjxxz2) 8

[4.3.2 Drop Down Lists](#_uz9l6s57rho) 8

[**5. CSCI detailed design**](#_cidym3j3od9p) **8**

[**7. Notes**](#_gjddhjhynlbe) **8**

TODO: CSCI-wide Design Decisions, CSCI Architectural Design, CSCI Detailed Design, Requirements Traceability, Notes

# **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

**6. Requirements Traceability**  
**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. If no start file, generate random state object
   1. Two proposed methods: Randomly generate a (potentially random) number of starting cells
   2. Or randomly generate each node at a fixed chance
   3. I think a is probably faster. I have no idea how we will test if this feature is working well.
5. Step time
   1. Change flag = 0
   2. Compute neighbors
   3. Apply rules for each node
   4. Change flag = 1 if any change
6. Iteration check to terminate
7. 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)
8. Display new state
   1. New state is displayed
9. 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