Software Design Document

for

Project 1

Prepared by Ryan Hiner, Dillen Kelton, Ricardo Reyna, Nicholas Shishkoff, and Kaylee Williams

CS3398

10/01/2020

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this SDD is to describe the design details and functionality requirements of Team Lrr, Ruler of planet Omicron Persei 8’s implementation of Conway’s Game of Life (hereinafter referred to as Game of Life) program. This document is intended to provide a clear understanding of the desired architecture, implementation, functionality, and development path of the program for use by the development team. This document should provide an overall understanding of the necessary information and details to develop a Game of Life program.

1.2 System Overview

The Game of Life program shall be written in C++ using the OpenGL library to provide GUI features to the user. The Game of Life program is intended to run on Windows operating systems. The implementation of the Game of Life program shall incorporate the Model-View-Controller design pattern to allow the user to interact with GUIs, which in turn will control the behavior of the simulation.

1.3 Definitions, Acronyms, and Abbreviations

SDD: Software Design Document
GUI: Graphical User Interface
MVC: Model-View-Controller (Design Pattern)

1.4 Supporting Materials

Team Lrr’s SRS for Conway’s Game of Life
Link to State Diagram: https://photos.app.goo.gl/Bxj3BwsJ5Lxz4qx99

1.5 Document Overview

Section 1:

Provides an overview of the information and material contained within the SDD.

Describes the design pattern to be used to implement the Game of Life.

Section 2:

Defines how the MVC design pattern is incorporated into the Game of Life program.

Section 3:

Provides detailed diagrams exhibiting user interaction with the program.

Details the functionality and operation of the program's components.

2. Architecture

2.1 Overview

The application will be constructed using the MVC pattern, while commonly the MVC pattern is used in applications which run multiple instances, it still is one of the best choices for this application in that it's design philosophy is very useful for compartmentalizing the application and thus making updates and maintenance easier and based on what components the application has, this makes the most sense.

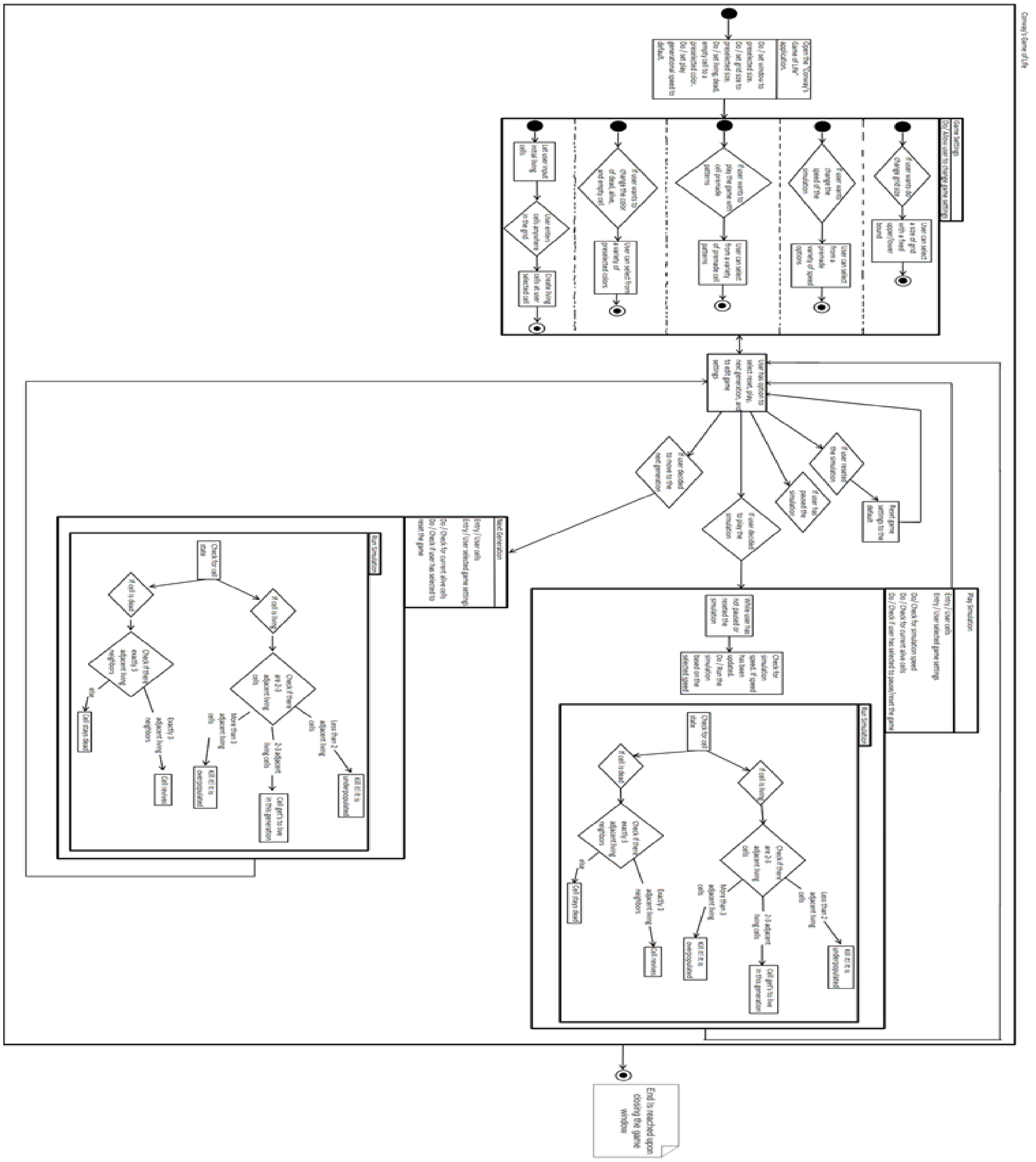
2.2 Components

2.2.1 Model: True to the pattern, the model in this application will be handling the data storage, this includes storing presets and holding relevant information of the current state.

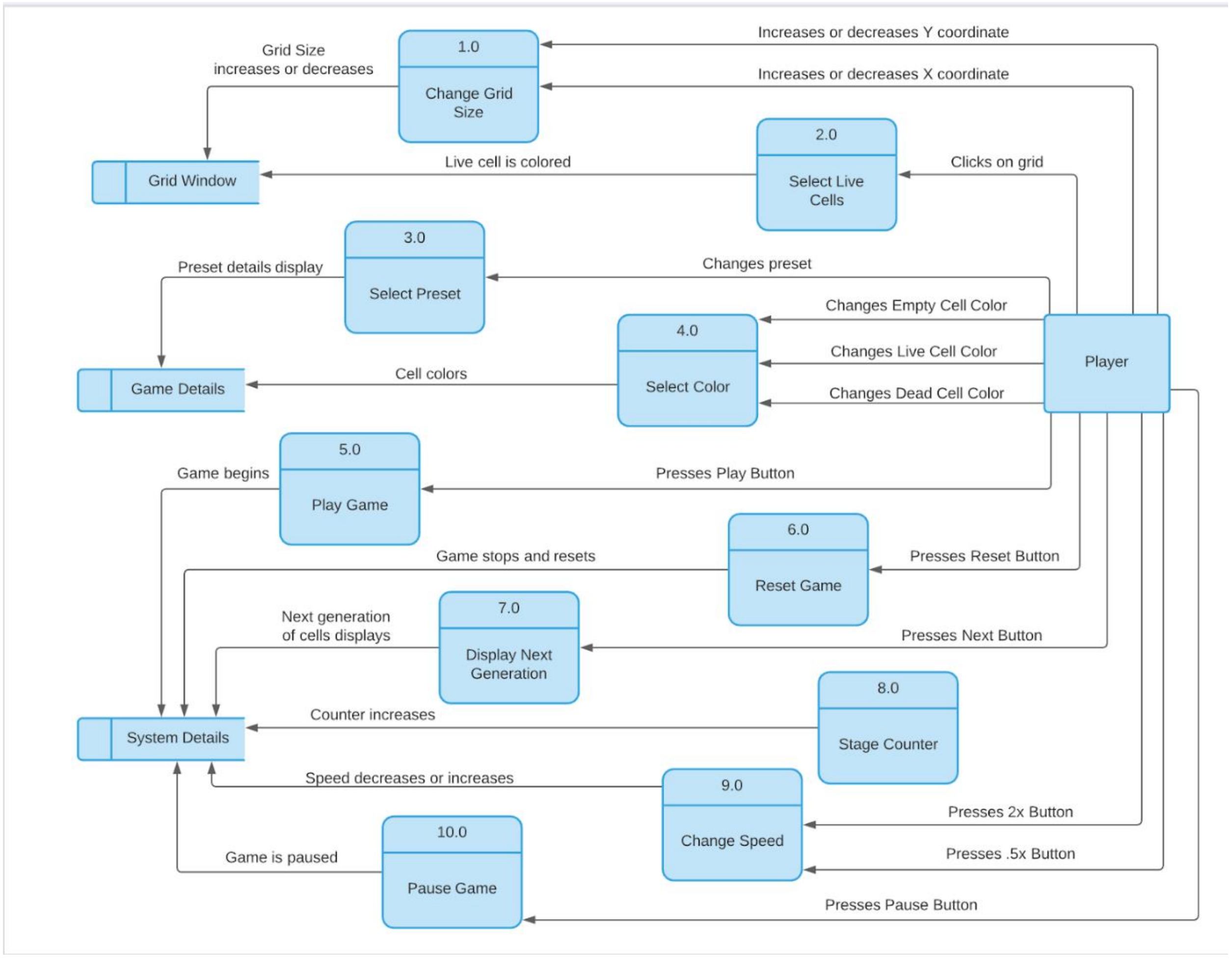
2.2.2 View: The view for the application is going to be relatively simple, consisting of only one window which will be displaying all of the UI elements for the application as well as displaying some state changes from the model such as a change in cell state.

2.2.3 Controller: The controller will be the intermediary between these two, the controller will be responsible for registering inputs and then communicating those to the model, which in turn gets displayed by the view.

State Diagram - For a better view of the diagram, the image has an attached link that directs to the original file. This link is also in 1.4 Supporting Materials.


Conway's Game of Life
Open the "Conway's Game of Life" application.
Do / set window to predefined size.
Do / set grid size to predefined size.
Do / set living, dead, empty cell to a predefined color.
Do / set play generational speed to default.
Game Settings
Do / Allow user to change game settings
If user wants to change grid size
User can select a size of grid with a fixed upper/lower bound
If user wants to change the speed of the simulation
User can select from a variety of speed options
If user wants to play the game with cell premade patterns
User can select from a variety of premade cell patterns
If user wants to change the color of dead, alive, and empty cell
User can select from a variety of predefined colors
Let user input initial living cells
User enters cells anywhere in the grid
Create living cells at user selected cell
User has option to select reset, play, next generation, and to edit game settings
If user decided to move to the next generation
If user decided to play the simulation
If user has paused the simulation
If user resets the simulation
Reset game settings to the default.
Next Generation
Entry / User cells
Entry / User selected game settings
Do / Check for current alive cells
Do / Check if user has selected to reset the game
Play Simulation
Entry / User cells
Entry / User selected game settings
Do / Check for simulation speed
Do / Check for current alive cells
Do / Check if user has selected to pause/reset the game
While user has not paused or reseted the simulation
Check for simulation speed, if speed has been updated.
Do / Run the simulation based on the selected speed
Run Simulation
Check for cell state
If cell is dead
If cell is living
Check if there exactly 3 adjacent living neighbors
Check if there are 2-3 adjacent living cells
else
Cell stays dead
Exactly 3 adjacent living neighbors
Cell revives
More than 3 adjacent living cells
Kill it; it is overpopulated
2-3 adjacent living cells
Cell get's to live in this generation
Less than 2 adjacent living cells
Kill it; it is underpopulated
End is reached upon closing the game window

3. High-Level Design



Data Flow Diagram

3.1 Model

The Model component contains all of the game’s variables and presets which are written to by the Controller component, and read from by the View component.

The Model will contain a dynamic array which has pointers to predefined graphical objects and all their relevant data (location, texture, size, etc.) This is what the View component will read from.

The Model will also contain a 2D array of integers which represents the state of the grid. Whenever the state of the game needs to be changed, this array is used as a reference for the current state and is then updated. Whenever this array is updated, the dynamic graphical array must also be updated so that the View component may reflect these changes. For example, if the grid is set to 3x2 the 2D array may look like following, where 0s may be empty cells, 1s alive cells, and 2s dead cells:

```
Pseudocode {

grid_array = { 0, 1, 1

                  2, 1, 0 };

}
```

3.2 View

The View component will consist of two major parts, a GUI panel at The View component will consist of two major parts, a GUI panel at the bottom of the window, and a game grid above that.

The GUI panel will display current game settings as well as show buttons that can be interacted with through the Controller component to change some of these settings.

The game grid will display colored cells based on their settings and state within the game (alive, dead, or empty).

To achieve all of the above, the View component reads from an array of graphical objects held within the Model component and extracts from them the necessary data to render them graphically.

The View component has no impact on the state of the game and cannot modify any internal variables. All functions that belong within the View component can be treated as read-only as they only exist to read from existing variables within the Model component and render them onto the user’s screen.

3.3 Controller

The Controller component handles communication between the View and Model components and allows the user to interact with the game. The Controller is also responsible for handling the internal logic of the game and processing each generation of the simulation.

The Controller has event listeners which watch for events like mouse clicks and updates the state of the game appropriately.

When a mouse click event has fired, the Controller component checks where the mouse was when it clicked, and if those coordinates lie within any existing graphical objects. To do this, the Controller reads from an array of graphical objects within the Model component. These graphical objects have event handlers within them, which are triggered if the mouse cursor intersected with them at the time of the click event.

The Controller will include a timer which will process the next generation in the game upon specified intervals and update the relevant variables which the View component reads from. The user will be able to modify this interval through the Controller component, either speeding it up or slowing it down. First, the 2D grid array within the Model component will be read from and used to determine an updated grid array based upon the rules of the game, then that updated array will be used to update the graphical array.

If the user has paused, or not started the simulation, then the Controller will not change the game’s state unless an event is raised by the user.

The Controller component is responsible for handling events raised by the user, such as button presses and modifying the game’s state or the game’s settings according to the button that was pressed. For example, if the user has clicked on a different color to display living cells, the Controller will change current living cell colors through the graphical object array as well as set that color as the default for any new living cells.