# **Table of Contents**

Table of Contents1

Revisions1

1. Introduction2

1.1 Purpose2

1.2 System Overview2

1.3 Definitions, Acronyms and Abbreviations3

1.4 Supporting Materials3

1.5 Document Overview4

2. Architecture5

2.1 Views: (Display)5

2.2 Controller: (Actions)5

2.3 Models: (Business Logic)5

3. High-Level Design7

3.1 View / Model Component7

3.1.a Model7

3.1.b View7

3.1.c Controller8

3.2 System Flow9

**Revisions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author(s)** | **Description of Version** | **Date Completed** |
| Final | Ruben | Finishing Touches | 10/01/2020 |

1. **Introduction**

1.1 Purpose

1.1.1 The purpose of this Software Design Document (SDD) is to inform the developers about the functionality and requirements for Team Zoidberg’s version Decapod-10 of *John Conway’s game Life* program. This SDD will give full details regarding system requirements, functionality, and the interactions with the system. Details shall be provided to describe the necessary information in order to code and test the functionality of each component outlined in the SRS.

1.2 System Overview

1.2.1 Our software shall comprise three sections

1. The main menu
2. The rules and product description
3. Game grid with setup functions.

1.2.2 The main menu will offer the user three buttons: Start Game, Exit, and Rules.

1.2.3 The Start game option will open the main game window. This window will display the game grid, game setup buttons, sliders, drop down menus, and game control buttons.

1.2.3.1 The game grid will consist of a square grid and the user can select a grid size from 10 by 10 to 50 by 50. Upon entering the game grid, the controller shall create a default grid of 20 by 20. In single player mode, the grid will display one of two colors or a background color and a sprite icon. The background color which represents a dead cell will be white. The second color which represents a live cell will be user selectable color (USC). If the user selects a sprite icon, the sprite will replace the USC cell to represent a live cell. The grid’s buttons will allow the user to click on them in order to change the color of a single cell. When the user clicks on a cell, that cell will change colors from white to USC/sprite or USC/sprite to white. In Player vs Player (PvP), the grid will allow two different colors or sprites to represent a live cell. Players cannot change the other player’s cell during setup. Changes to the cell’s status will be managed by the controller

1.2.3.2 The game grid will contain several setup buttons: Single/Multiplayer pulldown menu, iterations text window w/ up-down arrows, a speed slider, a grid size slider, a preset patterns pulldown menu, and three buttons for sprite icons.

1.2.3.3 The game grid will contain four control buttons: Start (run for preselected iterations), Next ( stop and single step to the next generation), Stop, and Reset (clear the grid to the same state as start)

1.2.4 The Rules button will open a window that describes the game rules, and two buttons 1) to return to the main menu and 2) Next.

1.2.4.1 The Next button will open a functionality window that describes the additional features Team Zoidberg added to John Conway’s game of Life.

1.3 Definitions, Acronyms, and Abbreviations

Software Design Document SDD

Software Requirements Specification SRS

User Selectable Color USC

Millisecond ms

Player vs. Player PvP

Model View Controller MVC

1.4 Supporting Materials

Wikipedia

Model-View-Controller

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

John Conway

<https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life>

Website dedicated to John Conway and his game of Life.

<https://www.conwaylife.com/wiki/Conway%27s_Game_of_Life>

1.5 Document Overview

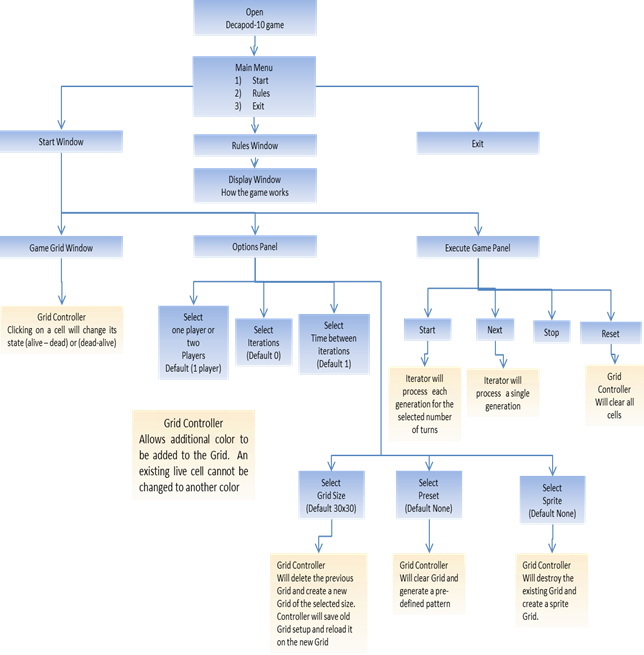


Figure 1 Overview

2. Architecture

2.1. Views: (Display)

1. Game View – Displays the game interface. This will accept the user’s inputs on the grid and optional toggles. The player can start/stop/step/reset the game interface which shall send the grid’s current setup (activated cells and toggles) to the controller. The game interface shall send information to the controller and receive information back with updates to the grid’s display at each iteration.
2. Title View – Displays the Title Menu interface. The player can ‘Start Game’ to navigate to the Game view, ‘Rules’ to navigate to the Rules view, or ‘Exit’ to close the application. This will be handled by the application controller.
3. Rules View – Displays information about game rules. The player can ‘Return to Title’ to navigate back to the Title view and ‘Functionality’ to navigate to the Functionality view. This will be handled by the application controller.
4. Functionality View – Displays information about additional functionality. The player can ‘Start Game’ to navigate to the Game view and ‘Rules’ to navigate back to the Rules view. This will be handled by the application controller.

2.2. Controller: (Actions)

1. Application Controller: This controller handles which view to display.
2. Iterator/Grid Controller: This controller bridges the flow of data between the game interface and the respective models and vice versa.

2.3. Models: (Business Logic)

1. Iterator Model: The iterator model accepts the array data sent from the controller. The data is used to determine which cells are activated on the grid for each generation. The grid size, preset, selected squares, iterations, and PvP toggles will affect this model.
2. Grid Model: The grid model accepts the data sent from the controller. The data is used to display which cells are activated on the grid for each generation. The grid size, sprite, speed, and PvP toggles will affect this model.

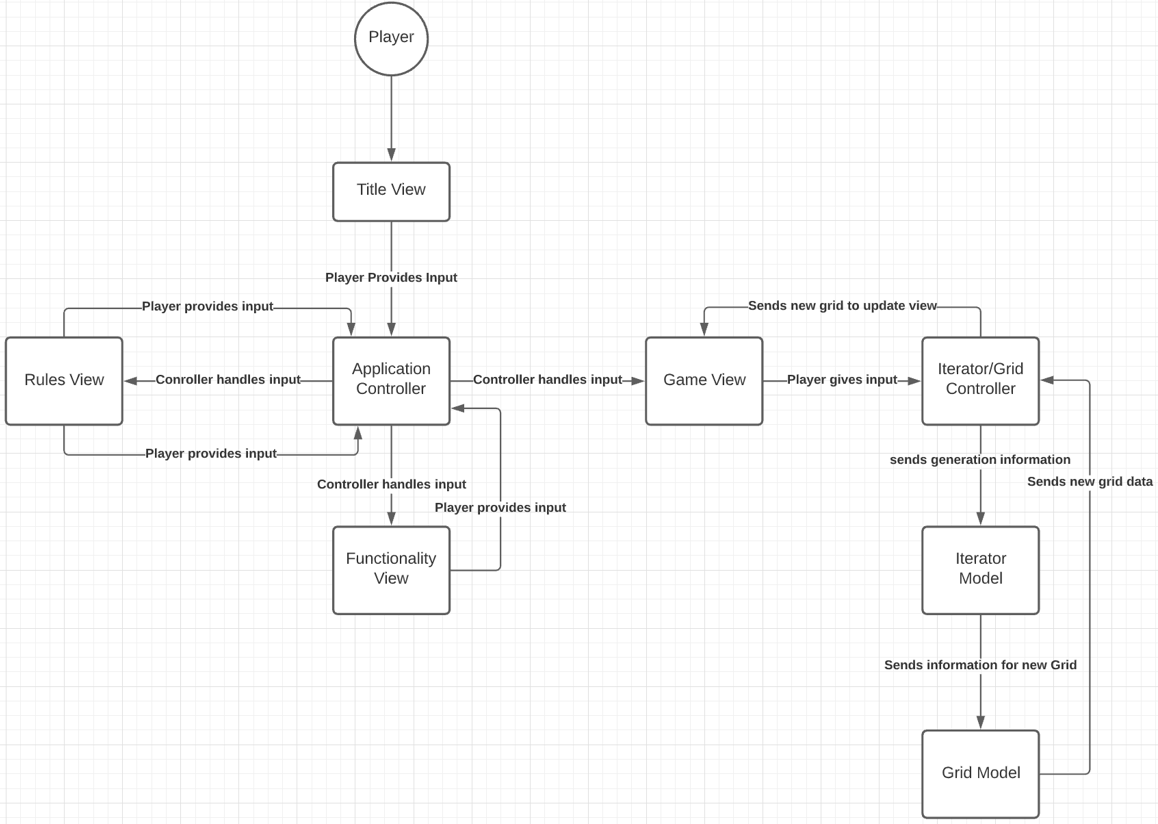


Figure 2 Game Model

3. High Level Design

3.1 View / Model Components

3.1.1 - The Conway’s Game of Life application will operate following the MVC architecture. The MVC components of the application shown in the sequence diagram are as followed:

1. Model - The model component of the application will be the iterator module and the grid module. These two models will actually handle executing the methods that will make the game function properly. The iterator will receive the number of iterations input by the player and will also return the new generation data for each iteration back to the controller. The grid module will handle executing methods for changing the grid. The grid module will receive the new generation data and update the grid based on that data. After the grid is updated, the grid module sends the updated grid data back to the controller.

General Pseudocode from Model perspective:

If user sets a number of iterations to run {

Iterator takes current cell generation data and calculates the generation

set by the user;

Grid gets updated with new cell composition data;

}

If user modifies the current live cell/grid composition{

Grid gets updated with new cell composition data

}

1. View - The view component of the application is the GUI module. This component will handle all the visual and interactive aspects of the game such as viewing the title screen, viewing the rules, toggling buttons, and dropdown menus. When the player has started the game and has entered their input, the GUI module will send the player’s desired number of iterations as well as the data from any buttons clicked to the controller component so execution of iterations can begin once the data reaches the model. Once the iterations are executed, the new grid data is produced by the model and sent back to the controller and then back to the view for the player to see.

General Pseudocode from View Perspective:

If application is opened {

GUI (view component) displays main menu in window;

If 2 Player mode is chosen {

Player 1 clicks desired cell color;

Player 1 clicks desired starting live cells or preset;

Player 1 inputs number of iterations;

Player 2 clicks desired cell color;

Player 2 clicks desired starting live cells or preset;

}

else {

Player 1 clicks desired cell color;

Player 1 clicks desired starting live cells or preset;

Player 1 inputs number of iterations;

}

If “Start” button is clicked {

GUI (view) send all players’ click data and number iterations to controller;

}

If new updated grid data is received {

GUI updates the live and dead cells according to updated grid data;

}

}

1. Controller - The controller component of the application is the game module in the diagram below. The controller will act as the game manager. The controller relays data between the model and view components. Any data that must be passed to another component must first go through the controller.. When the GUI (view) sends the player’s initial input to the controller, the controller will send those number of iterations to the iterator model so that the new generation data can be calculated and returned to the controller. Then the controller sends the new generation data to the grid model and is returned newly updated grid data. Once the controller has the updated grid data, the controller will send the info back to the view component.

Pseudocode from Controller Perspective:

if Controller receives click data and number of iterations(player input) from GUI {

Controller forwards player input iterations to the Iterator;

Controller forwards updated cell data to the Grid;

}

If Controller receives data from the Iterator{

Controller forwards cell data to Grid to update;

}

If Controller receives data from the Grid{

Controller forwards cell data to the View;

}

3.2 System Flow

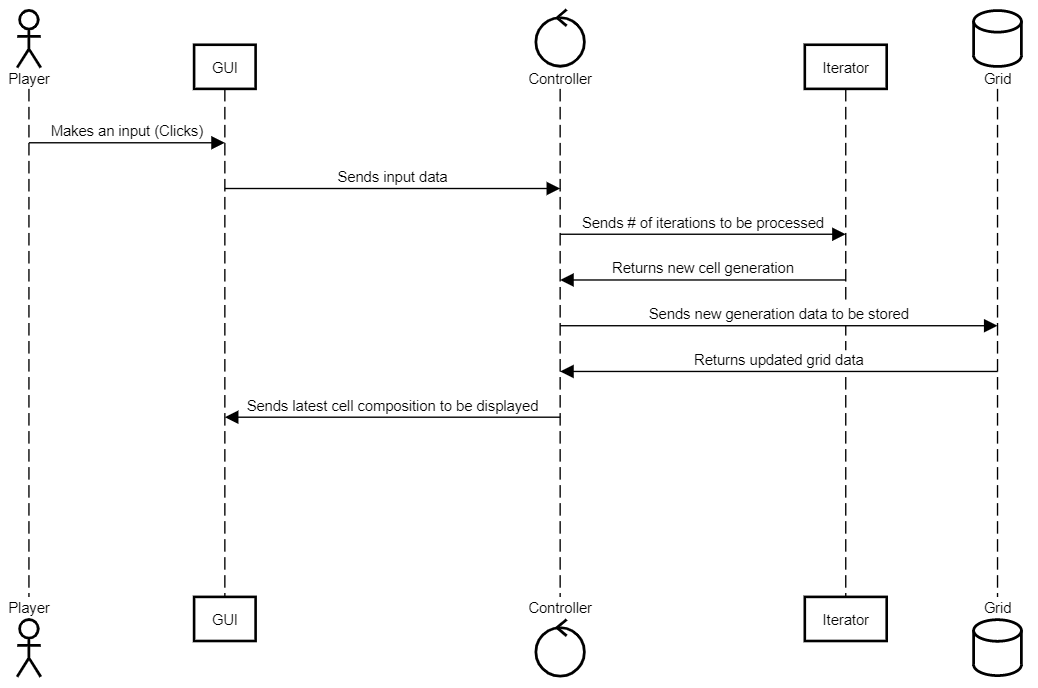


Figure 3 Sequence Diagram