CMSC 447
Software Design Description (SDD)

Name	Role	Signature
Mohamed Alkindi	Point of Contact, Documentation Writer	
Eoin Fitzpatrick	Software Developer, Documentation Writer, Program Tester	
Alex Flaherty	Lead Software Developer, Documentation Writer, Program Tester	
Evan Hart	Documentation Writer	
Gabriel Kilungya	Documentation Writer, Program Tester	
Hannah Russell	Documentation Writer, Program Tester	

1 Scope

1.1 Identification

Team Rocket's Game of Life, Team 3

Version 1, first release

Identifications:

Virus: Infectious system designed for gameplay interactions

Neighbor: The closest squares around the current square

Seed: Number used to generate pseudorandom board layouts

.exe: Windows OS executable program

GUI: Graphical user interface

Left click: User mouse left-click button

Right click: User mouse right-click button

1.2 System overview

The purpose of this software is to provide the user with an enhanced experience with Conway's Game of Life. This extension of the game adds features such as generating viruses, alongside living cells and seeing how each living cell and virus thrive or die together in each generation. In addition, users can randomly generate board configurations, change the game speed, and load or save board configurations.

The game comes packed into a single .exe which can be run on windows environments. The user is able to run the application on Windows systems installed with .NET framework 4+.

Team Rocket's Game of Life is sponsored by Ms. Dorothy Kirlew. This project is intended to be used by the public, and is currently being developed by Team Rocket. The open source code is available on github at the link https://github.com/hrussell898/CMSC447.

1.3 Document overview

The purpose of this document is to identify the main aspects of the implemented system design. The software is localized and does not present privacy or security risks towards users.

2 Referenced Documents

Below is a list of documents referenced.

a. .NET Framework system requirements 17 April 2019
b. SRS - Software Requirement Specification

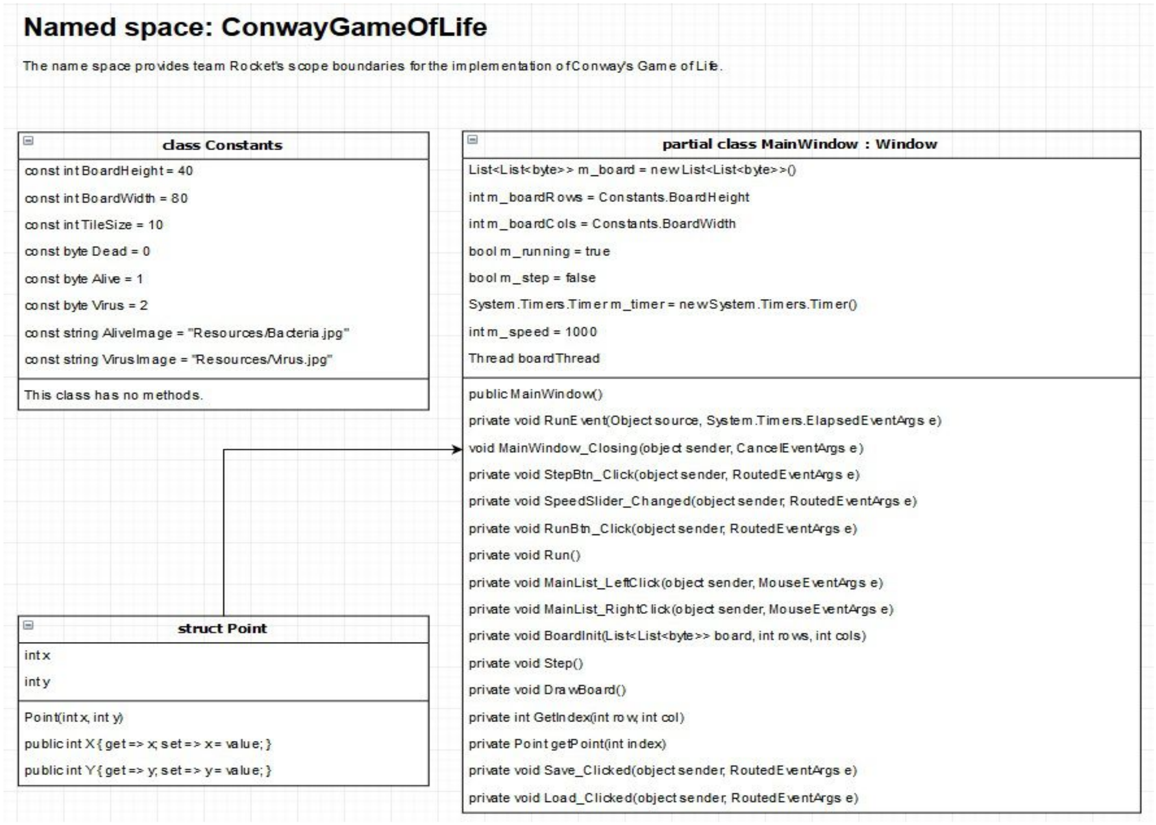
3 CSCI-wide design decisions

a. The CSCI accepts user input from any pointing device. The CSCI accepts user input in the form of a written file. The output from user and file input is writing a state change to the appropriate cell on the board.
b. The CSCI accepts a drag and drop gesture for selecting multiple cells at one time. The CSCI supports a right-click function that generates virus cells. The CSCI will output a state change only if proper input from a pointing device or file is received. The CSCI will update the changed state of the cell(s) selected by the input from file or device. The CSCI outputs random state changes to all cells when the randomizer button is selected by the pointing device. The CSCI ignores inputs from pointing devices that do not select buttons located on the interface. There are no unallowed inputs for pointing devices. Unallowed file input prompts a message stating the error which does not allow the file to be loaded.
c. The files written by the CSCI are written as single files with .gol format.

4 CSCI architectural design

4.1 CSCI components

a. The CSCI is made of the software units shown the following diagram.


Named space: ConwayGameOfLife
The name space provides team Rocket's scope boundaries for the implementation of Conway's Game of Life.
class Constants
partial class MainWindow : Window
struct Point

```
class Constants
const int BoardHeight = 40
const int BoardWidth = 80
const int TileSize = 10
const byte Dead = 0
const byte Alive = 1
const byte Virus = 2
const string AliveImage = "Resources/Bacteria.jpg"
const string VirusImage = "Resources/Virus.jpg"
This class has no methods.

partial class MainWindow : Window
List<List<byte>> m_board = new List<List<byte>>()
int m_boardRows = Constants.BoardHeight
int m_boardCols = Constants.BoardWidth
bool m_running = true
bool m_step = false
System.Timers.Timer m_timer = new System.Timers.Timer()
int m_speed = 1000
Thread boardThread
public MainWindow()
private void RunEvent(Object source, System.Timers.ElapsedEventArgs e)
void MainWindow_Closing(object sender, CancelEventArgs e)
private void StepBtn_Click(object sender, RoutedEventArgs e)
private void SpeedSlider_Changed(object sender, RoutedEventArgs e)
private void RunBtn_Click(object sender, RoutedEventArgs e)
private void Run()
private void MainList_LeftClick(object sender, MouseEventArgs e)
private void MainList_RightClick(object sender, MouseEventArgs e)
private void BoardInit(List<List<byte>> board, int rows, int cols)
private void Step()
private void DrawBoard()
private int GetIndex(int row, int col)
private Point getPoint(int index)
private void Save_Clicked(object sender, RoutedEventArgs e)
private void Load_Clicked(object sender, RoutedEventArgs e)

struct Point
int x
int y
Point(int x, int y)
public int X { get => x; set => x = value; }
public int Y { get => y; set => y = value; }
```

b. The MainWindow object maintains a struct Point utility structure.

c. The purpose of each unit is as follows:
   a. The main window provides the API interface to the program. It also contains the

entry point of the program.

b. The point structure is used as a utility to organize the representation of grid locations on the screen.
c. The constant structure is used to organize the boundaries of functionality for the program. It specifies information about the possible states of a grid element, image resource names, and the window size.

d. Software unit’s development status/type:
   a. The Constants class is at version 1.0. It is depicted in figure 4.1.a. It relies on the string library and references the image resources associated with a virus and a bacteria. All of its data members are public.

   b. The Point structure is at version 1.0. It is depicted in figure 4.1.a. It is a basic container class defined in the MainWindow class and has no dependencies. All of its data and function members are public.

   c. The MainWindow class is at version 1.0. It is depicted in figure 4.1.a. It is derived from the Window's API class Window therefore the base libraries are required. All of its data members are private

Hardware and software requirements:

d. The hardware requirements, in addition to the basic user terminal devices, for the software are derived from the the requirements for the .Net framework:

Processor	1 GHz
RAM	512 MB
Disk space (minimum)	
32-bit	4.5 GB
64-bit	4.5 GB

e. The minimum software requirement is Windows Vista SP2.
f. Source: .NET Framework system requirements 17 April 2019 https://docs.microsoft.com/en-us/dotnet/framework/get-started/system-requirements

4.2 Concept of execution

4.2.1 Conway’s Game of Life is based on a handful of rules:

4.2.1.1 A cell with one or more neighbors dies.

4.2.1.2 A cell with more than four neighbors dies.

4.2.1.3 A cell with 2-3 neighbors lives.

4.2.1.4 A cell with 3 neighbors becomes populated.

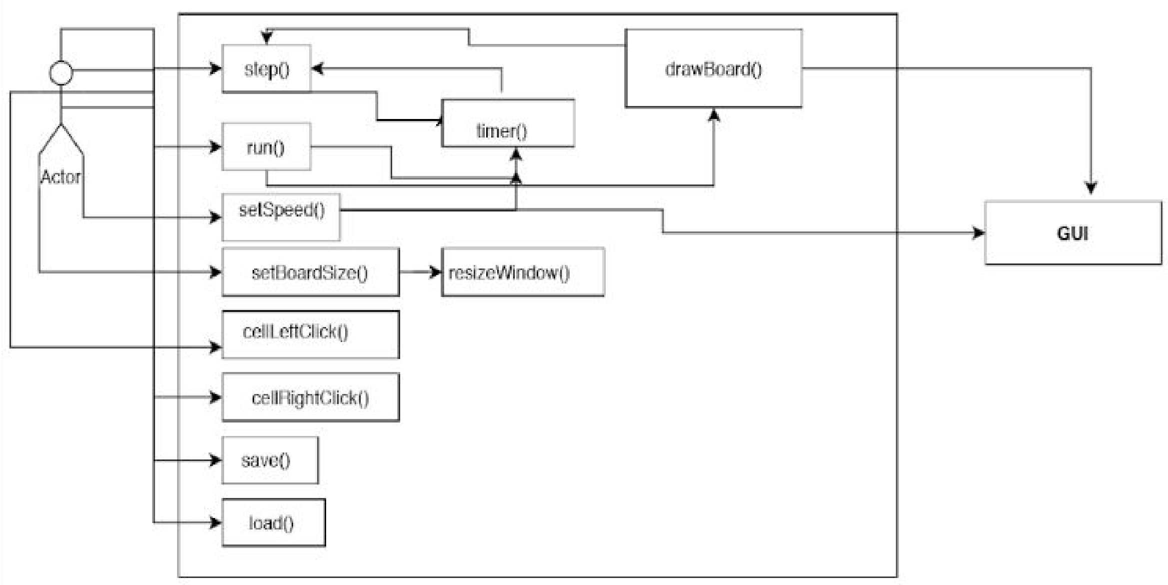
The following are rules for virus cells based on given requirements:

4.2.1.5 A virus populates one neighbor at random if it has more than one non-infected neighbor.

4.2.1.6 A virus with non-infected neighbors dies

4.2.2 The executable maintains a single execution thread (boardThread from Figure 4.1.a) within the window class that responds to changes in the simulation speed/step, continuous play, the initial conditions of the board, and periodically updates the board's cells status. During gameplay, if we select to load a game file, the current state of the board will switch to that of the loaded game and selecting step or run buttons, the game will proceed. If you select to save the game, the current state of the game will be saved.

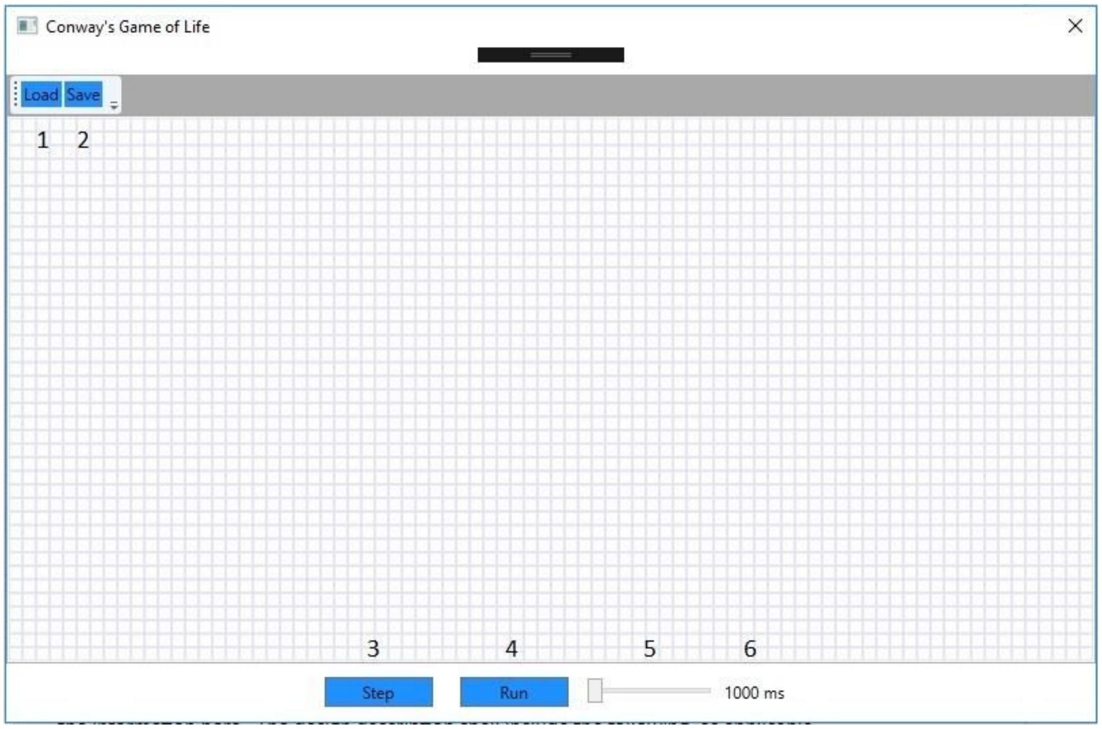
4.2.3 The UML diagram below shows the high-level interaction of software units used by the CSCI.



4.3 Interface design

4.3.1 **Interface identification and diagrams**

Below is a diagram showing the project-unique identifiers within the interface.



4.3.2 **(Project-unique identifier of interface)**

4.3.2.1 The load button is used to load a configuration from secondary storage prior to running a simulation.

4.3.2.2 The save button is used to save a particular configuration to secondary storage for later use.

4.3.2.3 The step button is used to step the program through iteration steps one at a time.

4.3.2.4 The run button progresses the simulation using a timer based on the value of the slider.

4.3.2.5 The slider controls the interim between steps of the program.

4.3.2.6 The time length label provides information about the amount of time between program steps.

4.3.2.7 The squares of the play board are used to initialize cells (live/dead) to a specific configuration prior to running the simulation.

5 CSCI detailed design

5.1 Team Rocket's implementation of Conway's Game of Life

5.1.1 Once the run button has been selected the program implements the 'rules' for the game of life described in section 4.2.1 at time intervals specified by the slider control described in section 4.3.2.5 and controlled by an execution thread as described in 4.2.2.

5.1.2 The language used by the CSCI is C#. This language fulfils our requirement for portability, while having other benefits like familiarity for our developers.

5.1.3 The CSCI outputs data once a save state as requested by the use of software unit 4.3.2.2. This software unit referred to in 4.3.2.2 utilizes the current state changes of each cell on the board and writes it to a .gol file.

5.1.4 All software units using logical operations follow based on the rules defined in 4.2.1.

6 Requirements traceability

6.1 Team Rocket's requirement/software unit traceability

6.1.1 Interface requirements are traced against the components names given in section 4.1.

6.1.2 Functionality requirements are traced against the execution logic outlined in section 4.2.

6.1.3 All other requirement traceability can be found in **SRS Section 4.**

7 Notes

7.1 Acronymns

7.1.1 API -- Application Programmer Interface

7.1.2 CSCI

7.1.3 SDD -- Software Design Document

7.1.4 SP2 -- Service Pack 2

7.1.5 UML -- Unified Modeling Language