CMSC 447

Software Test Description (STD)

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

This document contains an overview of all the tests that will be performed on the current version of the Team Rocket's Game of Life.

There are no security or privacy considerations for the tests.

2 Referenced documents

SRS: Software Requirements Specification

3 Test preparations

3.1 Project-unique identifier of a test

3.1.1 Hardware preparation

Requirement 2.2.1.6:1. - “The Game shall have a right click button to generate a virus, which is represented by a green square.”

Requirement 2.2.1.6:2. - “The Game shall have a left click button to generate a virus, which is represented by a black square

The mouse shall be able to handle these functionalities.

3.1.2 Software preparation

a. Testing the portability of the .exe on various Windows environments

For more information please visit:

Team Rocket’s Software User Manual, **Section 3.3 - Software Environment**

4. Test descriptions

4.1 Ready State Inspection Test

4.1.1 Test purpose

Proof by inspection to see the required elements and buttons of the GUI when the user initially launches the .exe file.

4.1.1.1. Requirements addressed

Requirement 2.2.1.1: “The game shall have a GUI”

Requirement 2.2.1.4: “The game shall have a step button”

Requirement 2.2.1.5: “The game shall have a run button”

Requirement 2.2.1.7: “The game shall have a speed slider”

Full requirement descriptions can be found in the Software Requirements Specification Document.

4.1.1.2 Prerequisite conditions

a. Hardware and software configuration: The game must be run on a Windows

environment.

b. Flags, initial breakpoints, pointers, control parameters, or initial data to be set/reset prior to test commencement: N/A
c. Initial conditions to be used in making timing measurements: N/A
d. Conditioning of the simulated environment: N/A

4.1.1.3 Test inputs

No test inputs are necessary to test the initial layout of the board and program when the .exe is launched, as referenced in 3.1.1 the user should test by inspection.

4.1.1.4 Expected test results

The program will have a visible start button, run button, step button, and speed slider when it is first launched. The initial board size will be of size 40 row x 80 columns.

4.1.1.5 Criteria for evaluating results

When the .exe is launched it will take the user to the working version with the required GUI elements present.

a. There should be no range in output for this test.

The following should be visible near the bottom of the GUI:

- Run button
- Step button
- Speed slider

4.1.1.6 Test procedure

The .exe will be clicked and launched from the user's Window device. Once launched the user will observe the initial board state and GUI. If this works and the required buttons are visible, this test is complete.

4.2 Populating the Board

4.2.1 Test purpose

The purpose of this test is to make sure the board can be populated with a variety of healthy cells and viruses, by left clicking and right clicking on the mouse respectively during various stages throughout the game. The user will also use the mouse to click on already filled cells or viruses to reset them to a blank state.

4.2.1.1 Requirements addressed

Requirement 2.2.1.6: "The user shall be able to turn on/off healthy cells and viruses"

Requirement 2.2.1.6.1: "Game shall have a left click button to generate a healthy cell"

Requirement 2.2.1.6.2: "Game shall have a right click button to generate a virus"

4.2.1.2 Prerequisite conditions

The prerequisite conditions for this test are the same as the ones listed in 4.1.1.2.

4.2.1.3 Test inputs

a. Healthy cells and viruses will be entered, by left or right clicking, into randomly selected rows and columns within the range of the board size. Each tester will randomly choose the cells on the board to turn on and off, making sure to get a variety of different cell locations throughout the board.
b. Real Input
c. The cells will be selected at the start of the game and then tested again after each button on the program has been pressed and executed.
d. No invalid input possible

4.2.1.4 Expected test results

Healthy cells that are generated by left clicking will appear on the board as a black square. Viruses generated by the right clicking will appear on the board as a green square. Cells that are clicked a second time after being turned on will return to the original state of a blank square. A cell can be turned on and off any number of times and will oscillate between the filled or unfilled squares with each click. If these statements are met, the test is complete.

4.2.1.5 Criteria for evaluating results

a. There should be no range in output for this test. Test results are based on observation data. Observation data is acquired through binary test results - each test result has either a true or false outcome observable to the tester.
b. N/A
c. N/A
d. No interrupts, halts, or other system breaks may occur while populating the board.
e. No processing errors are allowed to occur.
f. The result is inconclusive and re-testing is to be performed, if the cells fail to update to the correct state after clicked at any point in the program.
g. There are no conditions under which the outputs are to be interpreted as indicating irregularities in input test data, in the test database/data files, or in test procedures
h. Ready for next test case if board is populated with a variety of healthy cells and viruses.

4.2.1.6 Test procedure

a. Test operator will launch the executable and proceed to left click on a minimum of 10 cells and right click on a minimum of 10 cells.

1) The condition of the cells will be inspected to ensure each cell has turned the correct color representing its respective cell type.
2) Test operator will click on a minimum of 3 cells that are turned on in order to return them to the off state.
3) Record data.
4) Repeat the test case if unsuccessful.
5) Apply alternate modes by choosing different cell locations to test.

6) Terminate the test case.

b. The expected results of step 1, are 10 green squares and 10 black squares generated to the board. The expected results of step 3, are 17 colored squares (black or green) populating the board.

c. Requirement traceability is covered below in **Section 5**.

d. In the event that the program is stopped, record the results and relaunch the executable to return to testing.

4.3 Running State Inspection Test

4.3.1 Test purpose

The purpose of this test is to analyze integrity and function of the game's running state. The running state includes testing if the run button, step button, speed slider work correctly. In addition, the board cell functions will be analyzed to make sure cells respond to Conway's Game of Life rules.

4.3.1.1 Requirements addressed

Requirement 2.2.1.2: "The game shall follow the rules listed in Conway's Game of life."

Requirement 2.2.1.3: "The game shall generate viruses"

Requirement 2.2.1.4: "The game shall have a step button that advances the cells forward one generation."

Requirement 2.2.1.5: "The game shall have a run button that continuously advances to the cells next generation."

Requirement 2.2.1.7: "Game shall have a speed slider that adjusts the speed each generation is generated at."

Full requirement descriptions can be found in the **Software Requirements Specification Document.** All requirements that are derived from the ones listed are also tested in this section.

4.3.1.2 Prerequisite conditions

a. The program .exe must be running.
b. The "run" button must be pressed, and the board must be moving.
c. Preferably, a "sufficient" board layout is provided by either loading or using the "random" button.
d. See 4.1.1.2. for remaining prerequisite conditions.

4.3.1.3 Test inputs

1. Cell with no neighbors

Purpose: follows first given rule for Conway's Game of Life and step correctly executes.

a. Source of test input: user clicking on cells.

b. Real Input
c. Test Operator left clicks on numerous cells that have no neighbors, then presses step.
d. Test in a variety of locations. No invalid input is possible.

2. Cell with four or more neighbors

Purpose: follows second given rule for Conway's Game of Life and step correctly executes.

a. Source of test input: user clicking on cells
b. Real Input
c. Test Operator left clicks a cell and left clicks 4 more times to give it four neighbors, then presses step.
d. Test in a variety of locations and with different neighbors. No invalid input is possible.

3. Cell with 2-3 neighbors

Purpose: follows third given rule for Conway's Game of Life and step correctly executes.

a. Source of test input: user clicking on cells
b. Real Input
c. Test Operator left clicks a cell and left clicks two of its neighbors, then presses step.
d. Test in a variety of locations and with different neighbors. No invalid input is possible.

4. Virus with no surrounding healthy neighbors

Purpose: follows second rule for virus and step correctly executes.

a. Source of test input: user clicking on cells
b. Real Input
c. Test Operator right clicks a cell with no neighbors, then presses step.
d. Test in a variety of locations and with different neighbors. No invalid input is possible.

5. Virus populates a healthy neighbor

Purpose: follows first and third rule for virus and step correctly executes.

a. Source of test input: user clicking on cells
b. Real Input
c. Test Operator right clicks a cell and left clicks 2 - all of its neighbors, then presses step.
d. Test in a variety of locations and with different neighbors. No invalid input is possible.

6. Run button

Purpose: The run function continuously steps through generations and upholds the rules of

Conway's Game of Life and the customer derived viruses.

a. Source of test input: loaded file, or random button.
b. Simulated Input
c. Test Operator clicks run button on a loaded board.
d. Test with a variety of board layouts including an empty board and full board. No invalid input is possible.

7. Slider

Purpose: Test that the slider adjusts the speed that the next generation of cells is shown at.

Conway's Game of Life and the customer derived viruses.

a. Source of test input: loaded file, or random button.
b. Simulated Input
c. Test Operator clicks run button on a loaded board, then pauses adjusts speed and runs again.
d. Test with a variety of board layouts including an empty board and full board. No invalid input is possible.

4.3. 1.4 Expected test results

1. All cells will die returning a blank board.

2. The cell will die returning to a blank space.

3. The cell lives remaining as a current black square.

4. The virus will die returning to a blank space.

5. The virus will spread to one of the neighboring cells at random making it green.

6. Program will continuously generate the next generation of cells without interruption.

7. The slider will cause the game to run slower when moved to the right and faster when moved to the left.

4.3.1.5 Criteria for evaluating results

a. Tests results are based on observation data. Observation data is acquired through binary test results - each test result has either a true or false outcome observable to the tester.
b. N/A
c. N/A
d. After pressing the step button, the board should go to the next generation of cell. In this process, no interrupts, halts, or other system breaks may occur.
e. No processing errors are allowed to occur.
f. The result is inconclusive and re-testing is to be performed, if the cells fail to match the expected output listed in section 2.1.1.1.
g. There are no conditions under which the outputs are to be interpreted as indicating irregularities in input test data, in the test database/data files, or in test procedures
h. Ready for next test case if board is populated with a variety of healthy cells and viruses, the step function follows rules and advances one generation, the run function follows rules and continuously advances generations.

4.3.1.6 Test procedure

a. Test operator will launch the executable and produce the specified board set up that is listed for the test input.

1) The condition of the cells will be inspected to ensure each cell has turned the correct color representing its respective cell type after each step or each iteration when using run.

2) When using run, pause every so often to check that the board is updating correctly.
3) Record data
4) Repeat the test case if unsuccessful.
5) Apply alternate modes by choosing different cell locations to test the inputs and generating new random boards. Then test the step and run buttons with the new boards.
6) Terminate the test case

b. The expected results vary depending on the input entered. The results should follow the rules of Conway's Game of Life and the customer derived viruses at all times. The expected results for each input can be found in 4.3.1.4.
c. Requirement traceability is covered in **Section 5.**
d. In the event that the program is stopped, record the results and relaunch the executable to return to testing.

4.4 Loading and Saving Inspection Test

4.4.1. Test Purpose

The purpose of this test is to analyze integrity and function of the game's load and save buttons. The save button should save the game board to a binary file in a user-selected file directory. The load button should load binary containing the board state configuration from a user-selected file.

4.4.1.1 Requirements addressed

Requirement 2.2.1.8: "Game shall have the ability to save a current boards state"

Requirement 2.2.1.9: "Game shall have the ability to load a board"

4.4.1.2 Prerequisite conditions

A. The program .exe must be open
B. Preferably, a "sufficient" board layout is provided by either loading or using the "random" button
C. See **section 4.1.1.2** for other prerequisite requirements.

4.4.1.3 Test inputs

1. Save and Load

Purpose: verifies that a current board configuration can be saved to Windows file explorer and that a file can be loaded from Windows file explorer.

a. Source of test input: a randomly generated board
b. Simulated Input
c. Test operator clicks on the save button in top left corner to store the current board configuration. Next the test operator alters the board in some way and clicks the load button in the top right corner. The test operator then selects one of the previously save boards.

d. Test in a variety of locations. No invalid input is possible.

4.4.1.4 Expected test results

1. After clicking save, a Windows file explorer window is opened and a file is saved to an arbitrary location. After clicking load, a Windows file explorer is opened and a file is loaded with the updated board configuration.

4.4.1.5 Criteria for evaluating results

a. Tests results are based on observation data. Observation data is acquired through binary test results - each test result has either a true or false outcome observable to the tester.
b. N/A
c. N/A
d. No interrupts, halts, or other system breaks may occur.
e. No processing errors are allowed to occur.
f. The result is inconclusive and re-testing is to be performed, if the program loads or saves the board configuration incorrectly, or if the program crashes.
g. If the user tries to upload a file that is not the right file type they will be presented with an error message.
h. Ready for next test case if board data is saved without errors and a file can be loaded without errors.

4.4.1.6 Test procedure

a. Test operator will launch the executable and populate board with some cells.
1) Test operator will save this configuration by clicking the save button in the upper left corner
2) Make changes to the board by populating cells.
3) Load the original set up to the board by clicking the load button in the upper left corner and selecting the file that was saved in step 1.
4) There is no need for interim evaluations of test results
5) Record data.
6) Repeat the test case if unsuccessful
7) Attempt saving and loading files at various stages of the game.
8) Terminate the test case.

b. File loaded or saved matches the expected board.
c. Requirement traceability is covered below in **Section 5**.
d. In the event that the program is stopped, record the results and relaunch the executable to return to testing.

4.5 Seeding/Randomization Inspection Test

4.5.1. Test Purpose

The purpose of this test is to analyze the functionality of entering a seed value, entering a percent chance, and clicking the Randomize button, which will populate the board with live cells and viruses.

4.5.1.1 Requirements addressed

Requirement 2.2.1.11 "Game shall have option to generate cells automatically"

4.5.1.2 Prerequisite conditions

The prerequisite conditions for this test are the same as the ones listed in 4.1.1.2.

4.5.1.3 Test inputs

1. Percent Chance Feature

Purpose: percent chance feature correctly fills the board with the right amount of cells.

a. Source of test input: user types a percent value int the percent chance box.
b. Real Input
c. Test Operator types in percent value and presses enter, then the test operator will press the randomize button.
d. Invalid Inputs: Any negative that the tester enters will be treated as a positive number. Any other invalid text that is entered will be disregarded by the program.

2. Seed Feature

Purpose: seed feature correctly allows user to choose the seed for the programs random generator.

a. Source of test input: user types a seed value into the seed box.
b. Real Input
c. Test operator will type in a seed value and press enter, then the test operator will press the randomize button.
e. Invalid Inputs: Any negative that the user enters will be treated as a positive number. Any other invalid text that is entered will be disregarded by the program.

4.5.1.4 Expected test results

1. The board will be full with the correct percentage of cells that the user chose to enter. So, if the user chooses 100% chance then clicks randomize, every cell in the board shall be populated. If the user chooses 0% chance then clicks randomize then every cell in the board will be blank.
2. Each seed entered should result in a different board configuration.

The randomize button will populate the board with both healthy cells and viruses.

4.5.1.5 Criteria for evaluating results

a. The configuration of the board will vary with each test run. However, the number of cells that are populated should always match the percentage entered by the test operator.
b. N/A
c. N/A
d. No interrupts, halts, or other system breaks may occur.
e. No processing errors are allowed to occur.
f. Result is inconclusive and re-testing is to be performed, if the board does not generate a random board with both healthy cells and viruses.
g. The outputs should not be interrupted.
h. Ready for next test case if entering a seed works, percentage works as expected, and board is randomly populated with healthy and virus cells.

4.5.1.6 Test procedure

a. Test operator will launch the executable and automatically populate board.
    1) The test operator will enter the value 0 for % chance.
    2) The test operator then clicks the Randomize button.
    3) Observe that the state of the board has not changed from the initial state.
    4) Repeat step 1, with value 50.
    5) Repeat step 2.
    6) Now, observe that the board is filled, but not completely. Record results.
    7) Repeat step 1, with value 100.
    8) Now, observe that the board is completely filled. Record results.
    9) Repeat step 1, with value 90. Record results.
    10) Now, observe, a few cells are still unused.
    11) Repeat steps 1 - 10 with a variety of seed values and make sure new boards are generated.
    12) Terminate the test case
b. Expected result and evaluation criteria for each step Randomize.
c. Requirement traceability is covered below in **Section 5**.
d. In the event that the program is stopped or error occurs, record the result of current test and relaunch the executable to restart the testing with the error in mind.

4.6 Board Sizing Inspection Test

4.6.1. Test Purpose

The purpose of this test is to analyze the boards ability to change row size and column size with the users input. The test will also analyze if the window size scales when the rows and columns change.

4.6.1.1 Requirements addressed

Requirement 2.2.1.10 "User shall have the ability to change the board size"

Full requirement descriptions can be found in the **Software Requirements Specification Document**. All requirements that are derived from the one listed are also tested in this section.

4.6.1.2 Prerequisite conditions

The prerequisite conditions for this test are the same as the ones listed in 4.1.1.2.

4.6.1.3 Test inputs

a. Source of test input: user types a row number and column number into their boxes.
b. Real Input
c. Test operator will type in an integer for row number, then press enter. Next test operator will type in an integer for the column number, then press enter.
d. Invalid Inputs: Negative numbers will be treated as a positive number. All other text input will be disregarded.

4.6.1.4 Expected test results

The board will adjust to the correct number of rows and columns once enter is pressed. The window will also scale with the board.

4.6.1.5 Criteria for evaluating results

a. The board will have specified rows and columns.
b. N/A
c. N/A
d. No interrupts, halts, or other system breaks may occur
e. No processing errors are allowed to occur
f. Result is inconclusive and re-testing is to be performed, if the board does not adjust to the correct rows or columns.
g. The outputs should not be interrupted.
h. Ready for next test case if board correctly adjusts to new size.

4.6.1.6 Test Procedure

a. Test operator will launch the executable and resize the board.
    1. The test operator will enter a number between 10 and 60 for the rows
    2. The test operator presses the enter key in the row box
    3. The test operator enters a value between 10 and 140 in the columns box

4. The test operator presses enter in the column box
5. Observe that the board has adjusted to the size input
6. Repeat step 1, using a number less than 10 or greater than 60
7. Repeat step 2
8. Observe that a pop-up window appears, telling the user that this input is invalid
9. Repeat step 3, using a number less than 10 or greater than 140
10. Repeat step 4
11. Observe that a pop-up window appears, telling the user that this input is invalid
12. Repeat step 1, using a negative number
13. Repeat step 2
14. Observe that the board treats the negative value as a positive value
15. Repeat step 3, using a negative number
16. Repeat step 4
17. Observe that the board treats the negative value as a positive value
18. Terminate test case

b. Board size is displayed as expected.
c. Requirement traceability is covered below in **Section 5**.
d. In the event that the program is stopped, record the results and relaunch the executable to return to testing.

5. Requirements traceability

The following table data is extracted from the SRS, section 2.2.
The numbers in the boxes represent the subset of requirements met. If “All”, then all subsets are met.

Capability Requirement

Test	2.2.1.1	2.2.1.2	2.2.1.3	2.2.1.4	2.2.1.5	2.2.1.6	2.2.1.7	2.2.1.8	2.2.1.9	2.2.1.10	2.2.1.11
4.1	All			All	All		All				
4.2			All			All	All				
4.3		All	All	All		All					
4.4								All	All		
4.5											All
4.6										All	

Environment Requirement

Test	2.3.1	2.3.2	2.3.3
4.1	All		All
4.2			All
4.3			All
4.4			All
4.5			All
4.6		All	All

Computer Resource Requirement

Test	2.4.1.1	2.4.1.2	2.4.2.1	2.4.3
4.1	All		All	All
4.2	All		All	All
4.3	All		All	All
4.4	All	All	All	All
4.5	All		All	All
4.6	All		All	All