System Verification and Validation Plan for Game of Continuous Life

Baptiste Pignier

March 25, 2025

Revision History

Date	Version	Notes
14/02/2025	1.0	First release

Contents

List of Tables

Remove this section if it isn't needed

List of Figures

Remove this section if it isn't needed

1 Symbols, Abbreviations, and Acronyms

All symbols, abbreviations and acronyms used here are the same as in the SRS document. Please refer to the relevant section of the SRS for definitions. In this document, we will use the xSyB notation to define the generation rule used in discrete simulation. This notation means "a cell survives if and only if there are x cells in its neighborhood, and a cell is born in a cell that has y other cells in its neighborhood."

A Verification and Validation (V&V) plan is a comprehensive guide outlining the goals, scope, techniques, and standards for confirming and validating a software product. Verification ensures that the software aligns with specified requirements and design criteria, while validation confirms that it fulfills user needs and expectations. This plan is crucial for maintaining software, dependability, quality, and functionality, and for identifying and resolving issues before release.

The following document presents the V&V plan for Game of Continuous Life. It begins with the general information in Section 2, followed by a detailed discussion of the methods to achieve these objectives in Section 3. Finally, Section 4 provides the test descriptions.

2 General Information

2.1 Summary

The software to be tested is Game of Continuous Life. It can be used to simulate a continuous cellular automaton, allowing simulation parameters to be modified on the fly. In this way, the user can assess the effect of his modifications on the cell automaton, its evolution and its environment.

2.2 Objectives

The aim is to check the accuracy of the software, in the sense that the modifications applied to the simulation must be faithful to the modifications made by the user. Also, the graphical representation of the functions derived from the input parameters must be correct. For example, if the user wants the simulation to run twice as fast, then the simulation must run twice as fast. If the user wants to add a diffusion constraint to the automaton, this constraint must be taken into account.

The quality of the code will also be assessed using static analysis such as linter. The software depends on a graphics library, which will be assumed to be correct for the rest of the project. No checks will therefore be made on this library.

2.3 Challenge Level and Extras

The level of difficulty is admittedly intermediate, as a few mathematical models are involved in developing an unpredictable simulation. Also, particular attention will be paid to the user interface, which increases the complexity of the project.

2.4 Relevant Documentation

The SRS is crucial for VnV as it outlines what the software is supposed to do. It defines the functional and non-functional requirements that the software must meet. During verification, we check that the software meets these requirements. Validation ensures that the software fulfills its intended use as specified in the SRS.

Pignier (2025)

3 Plan

This section outlines the plan for verifying the documentation and software for Game of Continuous Life. The key elements to be verified include are SRS, design, V&V, and implementation.

3.1 SRS Verification Plan

The SRS will be verified via feedback from a domain expert by using the SRS checklist designed by Dr.Smith. The feedback will be collected by an issue tracker. The author will be responsible for dealing with each issue by discussing it with the sender and applying a corrective measure or justifying its absence.

3.2 Design Verification Plan

3.3 Verification and Validation Plan Verification Plan

The V&V Plan will be verified via feedback from a domain expert by using the V&V checklist designed by Dr.Smith. The feedback will be collected by an issue tracker. The author will be responsible for dealing with each issue by

discussing it with the sender and applying a corrective measure or justifying its absence.

3.4 Implementation Verification Plan

Verification of the implementation will be carried out by manual review and application of the unit tests listed below. Their results will be compared with those expected and potential differences will be justified. Automatic tests will also be used.

3.5 Automated Testing and Verification Tools

The automatic tests will be carried out mainly by the pylint tool, which enables static verification of the code, in particular to check that it complies with the PEP8 python programming standard. Pylint can also be used to check good code practices such as function atomicity and execution consistency, and can be used to optimise performance. The use of pylint will be incorporated into the project's workflow and continuous integration.

3.6 Software Validation Plan

As part of this project, the software will be verified by comparing the resulting simulations with those of other continuous cellular automaton simulation software.

4 System Tests

The following section presents the tests to which the functional and non-functional requirements will be subjected, as well as a traceability matrix.

4.1 Tests for Functional Requirements

Test on inputs

1. Simulation test

   Control: Automatic

Initial State: Initially, the grid will be in any state

Input: entries will be those corresponding to the 3B2S rule

Output: The output will be the simulation as it would have been with the same grid on a classic Game of Life simulator.

How test will be performed:

Inputs will be entered manually and the output will be automatically compared to that of another Game of Life simulation software through a simple comparison operation.

2. Function test

Control: Manual

Initial State: The function graphs will be in any state

Input: The input will be an arbitrary modification of the function parameters

Output: The output will be the modified graph of the function

How test will be performed:

Inputs will be entered manually, and output will be compared visually with that of another function visualization software such as Geogebra.

4.2 Tests for Nonfunctional Requirements

Nonfunctional Test

1. Usability test

Type: Manual

Initial State: The software will be launched for the first time, in an empty state.

Condition: Several third parties, who will never have used the software before, will be asked to run a simulation and modify its parameters, without any other help.

Result: The test will be passed if 80% of the third parties successfully complete the task.

2. Portability test

   Type: Manual

   Initial State: The software is to be downloaded onto a machine running any operating system.

   Condition: The software will be started and the simulation run.

   Result : The test will be passed if the simulation runs correctly, with no problems attributable to the operating system.

4.3 Traceability Between Test Cases and Requirements

	Simulation test	Function test	Usability test	Portability test
R1	X			
R2	X			
R4	X			
R3		X		
NFR2			X	
NFR4				X

Table 1: Traceability Matrix Showing the Connections Between Test Cases and Requirements

5 Unit Test Description

This section should not be filled in until after the MIS (detailed design document) has been completed.

Reference your MIS (detailed design document) and explain your overall philosophy for test case selection.

To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here,

you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests.

5.1 Unit Testing Scope

As previously stated, the graphics library is assumed to be correct. Thus, no unit tests will be imposed on it. Unit tests are therefore limited to what the author has implemented.

5.2 Tests for Functional Requirements

Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section.

5.2.1 Module 1

Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected.

1. test-id1

   Type: Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic

   Initial State:

   Input:

   Output: The expected result for the given inputs

   Test Case Derivation: Justify the expected value given in the Output field

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic

Initial State:

Input:

Output: The expected result for the given inputs

Test Case Derivation: Justify the expected value given in the Output field

How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant.

These tests may involve collecting performance data from previously mentioned functional tests.

5.3.1 Module ?

1. test-id1

   Type: Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input:

Output:

How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

Provide evidence that all of the modules have been considered.

References

Baptiste Pignier. System requirements specification. https://github.com/BaptistePignier/CAS741-GameOfLife/blob/main/docs/SRS/SRS.pdf, 2025.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

This is a section that would be appropriate for some projects.