

Module Guide for Game of Continuous Life

Baptiste Pignier

March 25, 2025

1 Revision History

Date	Version	Notes
03/16/25	1.0	First Release

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Game of Continuous Life	Explanation of program name
UC	Unlikely Change
US	User settings
FI	Functionnal inputs

Contents

1	Revision History	i
2	Reference Material	ii
2.1	Abbreviations and Acronyms	ii
3	Introduction	1
4	Anticipated and Unlikely Changes	2
4.1	Anticipated Changes	2
4.2	Unlikely Changes	2
5	Module Hierarchy	2
6	Connection Between Requirements and Design	3
7	Module Decomposition	4
7.1	Hardware Hiding Modules (M1)	4
7.2	Behaviour-Hiding Module	4
7.2.1	US View Module (M3)	4
7.2.2	US Control Module (M4)	5
7.2.3	FI View Module (M6)	5
7.2.4	FI Control Module (M7)	5
7.2.5	Simulation View Module (M9)	5
7.2.6	Simulation Control Module (M10)	5
7.2.7	Window Manager Module (M11)	6
7.3	Software Decision Module	6
7.3.1	US Model Module (M2)	6
7.3.2	FI Model Module (M5)	6
7.3.3	Simulation Model Module (M8)	6
8	Traceability Matrix	7
9	Use Hierarchy Between Modules	7
10	User Interfaces	8
11	Design of Communication Protocols	9
12	Timeline	9

List of Tables

1	Module Hierarchy	3
2	Trace Between Requirements and Modules	7
3	Trace Between Anticipated Changes and Modules	7

List of Figures

1	Use hierarchy among modules	8
2	User interface	8

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the “secrets” that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module’s data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers’ understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The programming language used for the implementation.

AC3: Adding a stochastic evolution feature.

AC4: The arrangement of graphic components on the simulation interface.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: Keyboard, Output: Memory and Screen).

UC2: The geometry of the simulation will remain a two-dimensional grid.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: US Model Module

M3: US View Module

M4: US Control Module

M5: FI Model Module

M6: FI View Module

M7: FI Control Module

M8: Simulation Model Module

M9: Simulation View Module

M10: Simulation Control Module

M11: Window Manager Module

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	M3
	M4
	M6
	M7
	M9
	M10
	M11
Software Decision Module	M2
	M5
	M8

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

7 Module Decomposition

Modules are decomposed according to the principle of “information hiding” proposed by [Parnas et al. \(1984\)](#). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Game of Continuous Life* means the module will be implemented by the Game of Continuous Life software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1 US View Module (M3)

Secrets: The view of the settings entered by the user

Services: Displays user settings selection objects on the GUI.

Implemented By: Game of Continuous Life

Type of Module: Abstract Object

7.2.2 US Control Module (M4)

Secrets: Management of settings entered by the user

Services: Retrieves the value of user settings selectors and makes them available for calculated functional inputs.

Implemented By: Game of Continuous Life

Type of Module: Abstract Object

7.2.3 FI View Module (M6)

Secrets: The view of the calculated functional inputs.

Services: Displays the result of calculated functional inputs on the GUI.

Implemented By: Game of Continuous Life

Type of Module: Abstract Object

7.2.4 FI Control Module (M7)

Secrets: Management of the calculated functional inputs.

Services: Retrieves user settings and makes the calculated functional inputs available to the simulation.

Implemented By: Game of Continuous Life

Type of Module: Abstract Object

7.2.5 Simulation View Module (M9)

Secrets: The view of the simulation result.

Services: Displays the simulation result on the GUI.

Implemented By: Game of Continuous Life

Type of Module: Abstract Object

7.2.6 Simulation Control Module (M10)

Secrets: Management of the simulation.

Services: Manages the simulation animations (play/pause/reset/zoom)

Implemented By: Game of Continuous Life

Type of Module: Abstract Object

7.2.7 Window Manager Module (M11)

Secrets: Management of the GUI.

Services: Manages the launching and stopping of the software and the arrangement of the different views on the graphical interface

Implemented By: Game of Continuous Life

Type of Module: Abstract Object

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 US Model Module (M2)

Secrets: Representation of user settings.

Services: Validates and saves user settings values.

Implemented By: Game of Continuous Life

Type of Module: Abstract Object

7.3.2 FI Model Module (M5)

Secrets: Representation of calculated functional inputs.

Services: Validates, calculate and saves functional inputs.

Implemented By: Game of Continuous Life

Type of Module: Abstract Object

7.3.3 Simulation Model Module (M8)

Secrets: Representation of the simulation state.

Services: Saves the simulation state and calculate the next one.

Implemented By: Game of Continuous Life

Type of Module: Abstract Object

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
R1	M1, M4, M3
R2	M2, M4, M7, M10
R3	M5, M7, M6, M4, M3
R4	M8, M10
R5	M9

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M2, M3, M4, M5, M6, M7, M8, M9, M10
AC3	M2, M8
AC4	M11

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

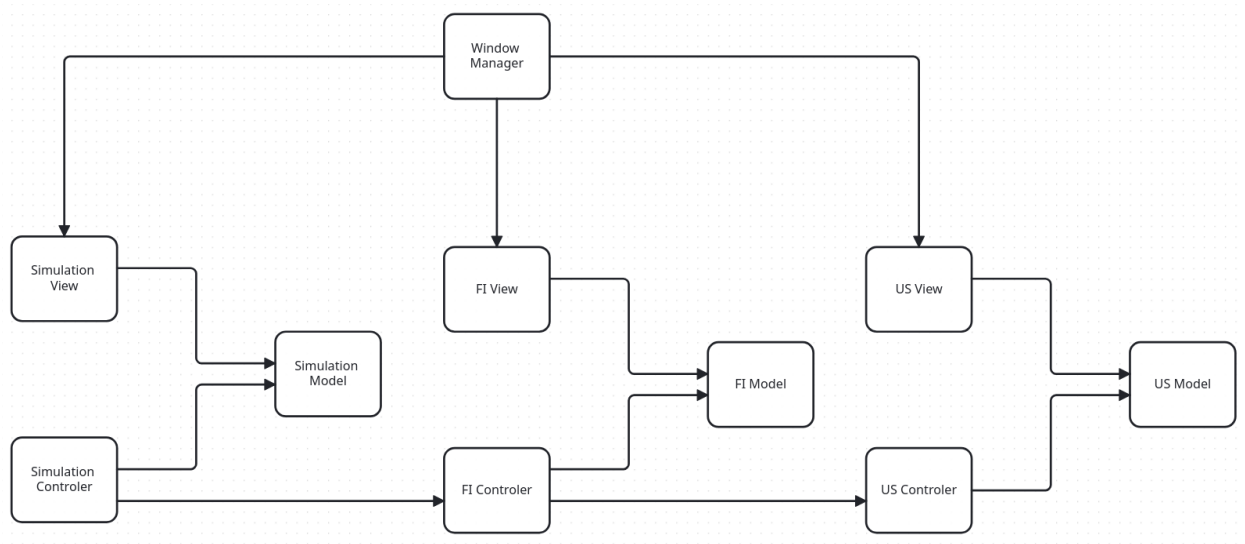


Figure 1: Use hierarchy among modules

10 User Interfaces

Figure 2 illustrates the user interface diagram.

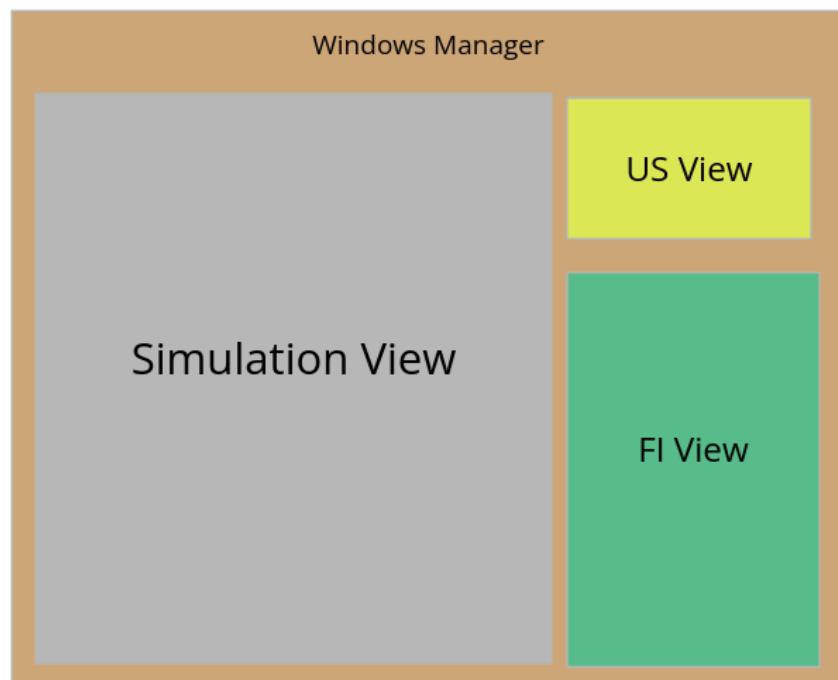


Figure 2: User interface

11 Design of Communication Protocols

Communication protocols are highly dependent on the programming language used. Modules will be implemented using the object-oriented programming paradigm. Therefore, information exchanges between modules will be done using getters and setters. The data types exchanged will be model instances or types specific to the chosen language.

12 Timeline

The project timeline can be found on the github page. <https://github.com/users/BaptistePignier/projects/3>

References

- David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.
- David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.
- D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.