Module Guide for Software Engineering

Team 8, RLCatan
Rebecca Di Filippo
Jake Read
Matthew Cheung
Sunny Yao

November 10th, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
AI	Artificial Intelligence
CV	Computer Vision
DAG	Directed Acyclic Graph
FR	Functional Requirement
M	Module
MG	Module Guide
NFR	Non-Functional Requirement
OS	Operating System
R	Requirement
RL	Reinforcement Learning
SRS	Software Requirements Specification
UC	Unlikely Change
UI	User Interface
VnV	Verification and Validation

Contents

List of Tables

List of Figures

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.

Our design follows the rules layed 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 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.

AC1: The specific architecture, heuristics, and reward functions of the RL agent. These will be iterated on constantly to improve performance.

AC2: The specific CV algorithm or model (e.g., YOLOv9 vs. OpenCV) used for board state detection. This may be changed to improve accuracy or adapt to different lighting conditions.

AC3: The layout and design of the User Interface. This will likely change based on user feedback to improve usability.

AC4: The underlying Catan simulator (Catanatron). This is an external library that may be updated or replaced.

AC5: The specific schema or technology (e.g., SQL vs NoSQL) for the Game State Database, as post-game analysis needs may evolve.

AC6: The communication protocol (e.g., 0MQ) used by the Image Queue. This might be swapped for another low-latency middleware like WebSockets.

4.2 Unlikely Changes

The module design should be as general as possible. However, fixing some design decisions at the system architecture stage can simplify the software design. It is not intended that these decisions will be changed.

UC1: The fundamental rules of the standard *Settlers of Catan* base game. The project is not intended to support expansions.

UC2: The core architectural decomposition. The system will always require separate components for vision, state management, AI, and a user interface.

UC3: The primary technology stack choices of Python for the backend/AI and JavaScript/React for the frontend.

UC4: The fundamental premise of reading from a *physical* game board via a camera, as opposed to being a purely digital application.

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 (OS)

M2: Computer Vision Model

M3: User Interface

M4: Game State Manager

M5: Reinforcement Learning Environment

M6: AI Model

M7: Game State Database

M8: Image Queue (Communication Layer)

Table 1: Module Hierarchy

Level 1	**Level 2**
Hardware-Hiding Module	M1: Hardware-Hiding Module (OS) M2: Computer Vision Model
Behaviour-Hiding Module	M3: User Interface M4: Game State Manager M5: Reinforcement Learning Environment
Software Decision Module	M6: AI Model M7: Game State Database M8: Image Queue

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. To satisfy the requirements for real-time operation (NFR.E.4), low-latency communication (FR.S.6.3), and component decoupling (NFR.S.3), a design decision was made to use a message queue middleware (as implemented in M8) for asynchronous communication between the primary components. This allows the CV, AI, and UI modules to operate independently without blocking each other.

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.

7.1 Hardware Hiding Modules

7.1.1 Hardware-Hiding Module (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware. This includes the drivers for the camera hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware (camera) and the software.

Implemented By: OS

7.1.2 Computer Vision Model (M2)

Secrets: The specific algorithms, libraries (e.g., OpenCV, YOLOv9), and trained models used to interpret a raw video/image feed and detect the physical board layout, pieces, and player actions. Hides the complexity of calibration and error correction.

Services: Provides access programs to process camera input, detect board elements, and translate visual features into a structured digital representation of the game state. Provides diagnostic feedback on detection confidence.

Implemented By: Software Engineering

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 SRS. This module serves as a communication layer between the hardware-hiding module and the software decision module.

Implemented By: –

7.2.1 User Interface (M3)

Secrets: The specific frontend framework (React, JavaScript) and UI/UX design choices used to render the game state and AI suggestions. Hides the details of DOM manipulation and event handling.

Services: Provides an interactive, real-time visualization of the Catan board. Displays player moves, resources, and AI-generated move recommendations. Allows users to provide manual corrections to the board state (to satisfy FR.Sa.1).

Implemented By: Software Engineering

7.2.2 Game State Manager (M4)

Secrets: The internal data structure and logic that represent the "Digital Twin" of the Catan board. Hides the complex logic used to enforce the official Catan ruleset and ensure state synchronization.

Services: Maintains the single source of truth for the current game state. Tracks player assets and turn data. Provides access programs to update the state and validate all player and AI moves against the game rules.

Implemented By: Software Engineering

7.2.3 Reinforcement Learning Environment (M5)

Secrets: The implementation details of the Catan game simulator (Catanatron). Hides the specific reward/penalty functions and opponent logic used during AI self-play and training.

Services: Simulates the game rules and state transitions of Catan. Provides an interface for the AI model to interact with, perform self-play, and receive feedback for learning. Supports model evaluation and benchmarking.

Implemented By: Software Engineering (as a wrapper around Catanatron)

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 AI Model (M6)

Secrets: The specific DRL model architecture (e.g., deep neural network parameters), heuristics, and trained weights used to determine the optimal move. Hides the complex mathematical formulation of the policy $\pi(a \mid s)$.

Services: Analyzes a given digital game state to predict optimal player strategies. Provides access programs to request move recommendations, confidence scores, and textual reasoning.

Implemented By: Software Engineering

7.3.2 Game State Database (M7)

Secrets: The specific database technology (e.g., SQL, NoSQL) and schema used to store and retrieve game history. Hides the implementation of data persistence, transactions, and indexing.

Services: Stores and maintains complete and consistent records of all game states and player profiles. Provides access programs for efficient read/write operations and structured access to historical game data for post-game analysis or replay.

Implemented By: Software Engineering

7.3.3 Image Queue (M8)

Secrets: The specific data transfer protocol and middleware (e.g., 0MQ/imageMQ) used for low-latency, asynchronous communication between modules. Hides the details of message serialization, authentication, and error handling.

Services: Enables reliable, low-latency data exchange between the CV Model, Game State Manager, AI Model, and UI. Supports asynchronous message-passing mechanisms.

Implemented By: Software Engineering

8 Traceability Matrix

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

Table 2: Trace Between Requirements and Modules

Requirement(s)	**Modules**
FR.S.1.x (CV Model)	M2
FR.S.2.x (RL Environment)	M5
FR.S.3.x (AI Model)	M6
FR.S.4.x (User Interface)	M3
FR.S.5.x (Game State DB)	M7
FR.S.6.x (Image Queue)	M8
FR.S.7.x (Game State Mgr)	M4
NFR.S.1 (Scalability)	M6, M8, M7
NFR.S.2 (Usability)	M3
NFR.S.3 (Maintainability)	M6, M2, M4, M3 (All)
NFR.S.4 (Installability)	M1
NFR.S.5 (Data Integrity)	M7, M4
NIndentingNFR.S.6 (Availability)	M8, M7
FR.Sa.1, FR.Sa.2, FR.Sa.4, FR.Sa.7, FR.Sa.8, NFR.Sa.1 (UI/Feedback Safety)	M3
FR.Sa.3 (AI Validation)	M4, M6
FR.Sa.5 (DB Safety)	M7
FR.Sa.6, NFR.Sa.2 (Connection Safety)	M8
FR.Sa.9, FR.Sa.11 (CV Safety)	M2
FR.Sa.10 (AI Rollback)	M6
FR.Sa.12 (Env. Validation)	M5

Table 3: Trace Between Anticipated Changes and Modules

Anticipated Change (AC)	Modules
AC1	M6
AC2	M2
AC3	M3
AC4	M5
AC5	M7
AC6	M8

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. A module A *uses* a module B if the correct execution of A depends upon the availability of a correct implementation of B. For our system, the 'uses' relation is not a simple hierarchy, as the main components communicate in a decoupled manner through the Communication Layer (M8). The relationships are shown in Figure **??**. This graph is a directed acyclic graph (DAG), as required.

10 User Interfaces

The detailed design of the User Interface (M3) will be documented using Figma wireframes and mockups. These design artifacts are attached as an appendix to this document (or will be available at a shared link). The design focuses on satisfying NFR.S.2 (Usability) and the specific UI-related safety requirements (e.g., FR.Sa.1, FR.Sa.7).

11 Design of Communication Protocols

The system's modules communicate via the M8 module. This module implements a publish-subscribe protocol. The design of this protocol is defined by the message topics and data structures (schemas) that are shared between modules.

Key communication topics will include:

- `cv.board.update`: Published by M2, consumed by M4. Carries structured data of detected board changes.
- `state.game.update`: Published by M4, consumed by M3 and M6. Carries the new authoritative game state.
- `ai.request.move`: Published by M3 (or other clients), consumed by M6. Requests a move suggestion for the current state.

- `ai.response.move`: Published by M6, consumed by M3. Provides the suggested optimal move.

The precise JSON or Protobuf schemas for these messages will be defined in the Module Interface Specification (MIS).

12 Timeline

The schedule of tasks, milestones, and team member responsibilities is maintained on our GitHub Project Kanban board, as specified in the Development Plan. This allows for dynamic tracking of progress. The board can be accessed here: github.com/users/SY3141/projects/1

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

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.