

# Module Interface Specification for Software Engineering

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

November 10, 2025

# 1 Revision History

Date	Version	Notes
11/03/2025	1.0	Draft Rev 1

## **2 Symbols, Abbreviations and Acronyms**

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

# Contents

<b>1 Revision History</b>	i
<b>2 Symbols, Abbreviations and Acronyms</b>	ii
<b>3 Introduction</b>	1
<b>4 Notation</b>	1
<b>5 Module Decomposition</b>	1
<b>6 MIS of [Module Name —SS]</b>	3
6.1 Module . . . . .	3
6.2 Uses . . . . .	3
6.3 Syntax . . . . .	3
6.3.1 Exported Constants . . . . .	3
6.3.2 Exported Access Programs . . . . .	3
6.4 Semantics . . . . .	3
6.4.1 State Variables . . . . .	3
6.4.2 Environment Variables . . . . .	3
6.4.3 Assumptions . . . . .	3
6.4.4 Access Routine Semantics . . . . .	3
6.4.5 Local Functions . . . . .	4
<b>7 MIS of Hardware-Hiding Module (M1)</b>	4
7.1 Module . . . . .	4
7.2 Uses . . . . .	4
7.3 Syntax . . . . .	4
7.3.1 Exported Constants . . . . .	4
7.3.2 Exported Access Programs . . . . .	4
7.4 Semantics . . . . .	4
7.4.1 State Variables . . . . .	4
7.4.2 Environment Variables . . . . .	5
7.4.3 Assumptions . . . . .	5
7.4.4 Access Routine Semantics . . . . .	5
7.4.5 Local Functions . . . . .	5
<b>8 MIS of Computer Vision Model (M2)</b>	5
8.1 Module . . . . .	5
8.2 Uses . . . . .	5
8.3 Syntax . . . . .	5
8.3.1 Exported Constants . . . . .	5
8.3.2 Exported Access Programs . . . . .	6

8.4 Semantics . . . . .	6
8.4.1 State Variables . . . . .	6
8.4.2 Environment Variables . . . . .	6
8.4.3 Assumptions . . . . .	6
8.4.4 Access Routine Semantics . . . . .	6
8.4.5 Local Functions . . . . .	6
<b>9 MIS of User Interface (M3)</b>	<b>6</b>
9.1 Module . . . . .	6
9.2 Uses . . . . .	6
9.3 Syntax . . . . .	7
9.3.1 Exported Access Programs . . . . .	7
9.4 Semantics . . . . .	7
9.4.1 Environment Variables . . . . .	7
9.4.2 Assumptions . . . . .	7
9.4.3 Access Routine Semantics . . . . .	7
<b>10 MIS of Game State Manager (M4)</b>	<b>7</b>
10.1 Module . . . . .	7
10.2 Uses . . . . .	7
10.3 Syntax . . . . .	7
10.3.1 Exported Access Programs . . . . .	7
10.4 Semantics . . . . .	8
10.4.1 State Variables . . . . .	8
10.4.2 Assumptions . . . . .	8
10.4.3 Access Routine Semantics . . . . .	8
<b>11 MIS of Reinforcement Learning Environment (M5)</b>	<b>8</b>
11.1 Module . . . . .	8
11.2 Uses . . . . .	8
11.3 Syntax . . . . .	8
11.3.1 Exported Access Programs . . . . .	8
11.4 Semantics . . . . .	8
11.4.1 State Variables . . . . .	8
11.4.2 Assumptions . . . . .	8
11.4.3 Access Routine Semantics . . . . .	9
<b>12 MIS of AI Model (M6)</b>	<b>9</b>
12.1 Module . . . . .	9
12.2 Uses . . . . .	9
12.3 Syntax . . . . .	9
12.3.1 Exported Access Programs . . . . .	9
12.4 Semantics . . . . .	9

12.4.1 State Variables . . . . .	9
12.4.2 Assumptions . . . . .	9
<b>13 MIS of Game State Database (M7)</b>	<b>9</b>
13.1 Module . . . . .	9
13.2 Uses . . . . .	10
13.3 Syntax . . . . .	10
13.3.1 Exported Access Programs . . . . .	10
13.4 Semantics . . . . .	10
13.4.1 Assumptions . . . . .	10
<b>14 MIS of Image Queue (M8)</b>	<b>10</b>
14.1 Module . . . . .	10
14.2 Uses . . . . .	10
14.3 Syntax . . . . .	10
14.3.1 Exported Constants . . . . .	10
14.3.2 Exported Access Programs . . . . .	10
14.4 Semantics . . . . .	11
14.4.1 State Variables . . . . .	11
14.4.2 Environment Variables . . . . .	11
14.4.3 Assumptions . . . . .	11
14.4.4 Access Routine Semantics . . . . .	11
<b>15 Appendix</b>	<b>13</b>

## 3 Introduction

The following document details the Module Interface Specifications for our project RLCatan. This project aims to create a competent reinforcement learning AI agent designed to master the board game Settlers of Catan through autonomous self-play training. The AI will use deep reinforcement learning algorithms to learn optimal decision-making strategies across several game states including resource management, territory expansion and adaptive responses to opponent actions.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at <https://github.com/SY3141/RLCatan>.

## 4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from [Hoffman and Strooper \(1995\)](#), with the addition that template modules have been adapted from [Ghezzi et al. \(2003\)](#). The mathematical notation comes from Chapter 3 of [Hoffman and Strooper \(1995\)](#). For instance, the symbol  $\Rightarrow$  is used for a multiple assignment statement and conditional rules follow the form  $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$ .

The following table summarizes the primitive data types used by Software Engineering.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in $(-\infty, \infty)$
natural number	$\mathbb{N}$	a number without a fractional component in $[1, \infty)$
real	$\mathbb{R}$	any number in $(-\infty, \infty)$

The specification of Software Engineering uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Software Engineering uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

## 5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

<b>Level 1</b>	<b>Level 2</b>
Hardware-Hiding Module	Hardware-Hiding Module (OS) Computer Vision Model
Behaviour-Hiding Module	User Interface Game State Manager Reinforcement Learning Environment
Software Decision Module	AI Model Game State Database Image Queue

## 6 MIS of [Module Name —SS]

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R???. —SS]

[It is also possible to use L<sup>A</sup>T<sub>E</sub>X for hyperlinks to external documents. —SS]

### 6.1 Module

[Short name for the module —SS]

### 6.2 Uses

### 6.3 Syntax

#### 6.3.1 Exported Constants

#### 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

### 6.4 Semantics

#### 6.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

#### 6.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

#### 6.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

#### 6.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

#### 6.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

## 7 MIS of Hardware-Hiding Module (M1)

### 7.1 Module

Hardware-Hiding Module (OS)

### 7.2 Uses

None (this module provides the hardware abstraction layer used by other modules).

### 7.3 Syntax

#### 7.3.1 Exported Constants

- **CAMERA\_DEVICE\_ID**: Identifier for the connected camera.

#### 7.3.2 Exported Access Programs

Name	In	Out	Exceptions
initializeCamera	-	Boolean (success/failure)	(succ- CameraConnectionError
captureFrame	-	Image	CameraReadError
releaseCamera	-	-	-

### 7.4 Semantics

#### 7.4.1 State Variables

cameraConnected: Boolean

#### **7.4.2 Environment Variables**

Camera device connected via USB or onboard webcam.

#### **7.4.3 Assumptions**

The camera hardware is available and permissions for access are granted.

#### **7.4.4 Access Routine Semantics**

initializeCamera():

- transition: connects to camera
- output: returns True if connection succeeds
- exception: CameraConnectionError if unavailable

captureFrame():

- output: returns latest frame from camera stream
- exception: CameraReadError if capture fails

releaseCamera():

- transition: releases hardware resources

#### **7.4.5 Local Functions**

None.

## **8 MIS of Computer Vision Model (M2)**

### **8.1 Module**

Computer Vision Model

### **8.2 Uses**

M1 (Hardware-Hiding Module), M8 (Image Queue), M4 (Game State Manager)

### **8.3 Syntax**

#### **8.3.1 Exported Constants**

- **MODEL\_PATH**: Path to trained detection model.
- **CONFIDENCE\_THRESH**: Minimum detection confidence.

### 8.3.2 Exported Access Programs

Name	In	Out	Exceptions
processFrame	Image	GameStateUpdate	DetectionError
getConfidence	-	Float	-
calibrateCamera	CalibrationData	Boolean	CalibrationError

## 8.4 Semantics

### 8.4.1 State Variables

model: trained CV model instance

confidence: Float

### 8.4.2 Environment Variables

Image feed from M1.

### 8.4.3 Assumptions

Camera feed is correctly initialized and lighting conditions are adequate.

### 8.4.4 Access Routine Semantics

processFrame():

- output: structured board state update
- exception: DetectionError if pieces or tiles cannot be identified

### 8.4.5 Local Functions

imagePreprocessing(), objectDetection(), boardMapping()

## 9 MIS of User Interface (M3)

### 9.1 Module

User Interface

### 9.2 Uses

M4 (Game State Manager), M6 (AI Model), M8 (Image Queue)

## 9.3 Syntax

### 9.3.1 Exported Access Programs

Name	In	Out	Exceptions
renderBoard	GameState	-	RenderError
displayAIMove	MoveSuggestion	-	-
submitFeedback	FeedbackData	Boolean	NetworkError

## 9.4 Semantics

### 9.4.1 Environment Variables

User browser window and input devices.

### 9.4.2 Assumptions

Frontend is served and connected to backend.

### 9.4.3 Access Routine Semantics

renderBoard():

- transition: updates display with new game state

displayAIMove():

- output: highlights AI suggestion on board

## 10 MIS of Game State Manager (M4)

### 10.1 Module

Game State Manager

### 10.2 Uses

M2 (CV Model), M7 (Game State DB), M8 (Image Queue)

### 10.3 Syntax

#### 10.3.1 Exported Access Programs

Name	In	Out	Exceptions
updateState	GameStateUpdate	Boolean	InvalidMoveError
getState	-	GameState	-
validateMove	Move	Boolean	RuleViolationError

## 10.4 Semantics

### 10.4.1 State Variables

currentState: GameState

### 10.4.2 Assumptions

All updates are serialized through the Image Queue.

### 10.4.3 Access Routine Semantics

updateState():

- transition: applies detected move to internal state
- exception: InvalidMoveError if move is inconsistent

# 11 MIS of Reinforcement Learning Environment (M5)

## 11.1 Module

Reinforcement Learning Environment

## 11.2 Uses

M4 (Game State Manager), M6 (AI Model)

## 11.3 Syntax

### 11.3.1 Exported Access Programs

Name	In	Out	Exceptions
reset	-	InitialState	-
step	Action	(NextState, Reward, Done)	InvalidActionError
renderSim	-	Visualization	-

## 11.4 Semantics

### 11.4.1 State Variables

envState: GameState

### 11.4.2 Assumptions

Catanatron environment is available and configured.

### **11.4.3 Access Routine Semantics**

step():

- transition: applies action to environment and updates state
- output: returns reward and next state

## **12 MIS of AI Model (M6)**

### **12.1 Module**

AI Model

### **12.2 Uses**

M5 (RL Environment), M7 (Database)

### **12.3 Syntax**

#### **12.3.1 Exported Access Programs**

Name	In	Out	Exceptions
predictMove	GameState	MoveSuggestion	ModelErrorNotLoaded
train	TrainingData	Metrics	TrainingError
evaluate	TestSet	Score	-

### **12.4 Semantics**

#### **12.4.1 State Variables**

weights: Matrix

optimizerState: Object

#### **12.4.2 Assumptions**

Model has been initialized and loaded from storage.

## **13 MIS of Game State Database (M7)**

### **13.1 Module**

Game State Database

## 13.2 Uses

None (accessed by other modules)

## 13.3 Syntax

### 13.3.1 Exported Access Programs

Name	In	Out	Exceptions
storeGameState	GameState	Boolean	DBWriteError
fetchGameState	GameID	GameState	DBReadError
queryHistory	PlayerID	GameHistory	DBReadError

## 13.4 Semantics

### 13.4.1 Assumptions

Database connection is active and schema is initialized.

# 14 MIS of Image Queue (M8)

## 14.1 Module

Image Queue (Communication Layer)

## 14.2 Uses

All communicating modules (M2, M3, M4, M6)

## 14.3 Syntax

### 14.3.1 Exported Constants

- **TOPIC\_BOARD\_UPDATE** = "cv.board.update"
- **TOPIC\_GAME\_UPDATE** = "state.game.update"
- **TOPIC\_AI\_REQUEST** = "ai.request.move"
- **TOPIC\_AI\_RESPONSE** = "ai.response.move"

### 14.3.2 Exported Access Programs

Name	In	Out	Exceptions
publishMessage	(Topic, Payload)	Boolean	NetworkError
subscribeTopic	Topic	Stream	NetworkError
receiveMessage	-	Payload	TimeoutError

## **14.4 Semantics**

### **14.4.1 State Variables**

connection: QueueConnection

### **14.4.2 Environment Variables**

Network socket for inter-module communication.

### **14.4.3 Assumptions**

Broker service (e.g., ZeroMQ or WebSocket server) is active.

### **14.4.4 Access Routine Semantics**

publishMessage():

- output: confirms message successfully published to topic

## References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL <http://citeseer.ist.psu.edu/428727.html>.

## **15 Appendix**

[Extra information if required —SS]

## Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?
2. What pain points did you experience during this deliverable, and how did you resolve them?
3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?
4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), if any, needed to be changed, and why?
5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO\_ProbSolutions)
6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO\_Explores)