Module Interface Specification for Software Engineering

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

November 12, 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

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 := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

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 (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

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.

Level 1	**Level 2**
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 LaTeXfor hypperlinks 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

[Use labels for cross-referencing —SS]

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

7.1 Module

Hardware-Hiding Module (M1)

7.2 Uses

Provides a hardware abstraction layer for other modules. Interfaces with OS-level drivers and the camera hardware or simulation layer to provide consistent input streams.

7.3 Syntax

7.3.1 Exported Constants

- `CAMERA_RESOLUTION` - resolution of captured frames
- `FRAME_RATE` - frames per second captured by the hardware layer

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
initializeHardware	-	Boolean	HardwareInitError
captureFrame	-	FrameData	CaptureError
shutdownHardware	-	Boolean	HardwareShutdownError

7.4 Semantics

7.4.1 State Variables

- `hardwareStatus`: Boolean - indicates if hardware is initialized
- `frameBuffer`: FrameData - stores the latest captured frame

7.4.2 Environment Variables

- `CameraDevice` - physical or virtual camera input
- `DisplayInterface` - optional screen interface for visualization

7.4.3 Assumptions

The module assumes that the camera or simulated device is available and that OS drivers are functional.

7.4.4 Access Routine Semantics

initializeHardware():

- transition: sets `hardwareStatus` to True on successful initialization
- output: returns True on success
- exception: raises `HardwareInitError` if initialization fails

captureFrame():

- transition: updates `frameBuffer` with latest captured frame
- output: returns `FrameData`
- exception: raises `CaptureError` if frame cannot be captured

shutdownHardware():

- transition: sets `hardwareStatus` to False and releases resources
- output: returns True on successful shutdown
- exception: raises `HardwareShutdownError` if cleanup fails

7.4.5 Local Functions

- `checkDeviceConnection()`: ensures camera is available
- `allocateBufferMemory()`: manages memory for frame storage
- `releaseResources()`: frees hardware resources on shutdown

8 MIS of Computer Vision Module

8.1 Module

Computer Vision Module (M2)

8.2 Uses

Processes raw camera input and translates it into structured board state data. Provides confidence metrics for detection and error correction.

8.3 Syntax

8.3.1 Exported Constants

- `DETECTION_THRESHOLD` - minimum confidence for object recognition
- `MODEL_PATH` - path to trained CV model (YOLOv9/OpenCV)

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
processFrame	FrameData	GameStateData	ProcessingError
detectBoardElements	FrameData	List[Elements]	DetectionError
getConfidenceMetrics	-	Dict	-

8.4 Semantics

8.4.1 State Variables

- `currentGameState`: structured representation of board state
- `lastFrame`: last processed frame

8.4.2 Environment Variables

- Access to `Hardware-Hiding Module` frame buffer

8.4.3 Assumptions

Assumes that frames provided by M1 are valid and that model files are correctly loaded.

8.4.4 Access Routine Semantics

processFrame(frame):

- transition: converts frame into structured game state
- output: returns `GameStateData`
- exception: raises `ProcessingError` if parsing fails

detectBoardElements(frame):

- transition: identifies tiles, pieces, and player actions in the frame
- output: returns list of detected elements
- exception: raises `DetectionError` if detection fails

getConfidenceMetrics():

- output: returns confidence scores for last processed frame

8.4.5 Local Functions

- `calibrateCamera()`: adjusts image for lens distortion
- `filterNoise()`: removes spurious detections

9 MIS of User Interface Module

9.1 Module

User Interface (M3)

9.2 Uses

Provides interactive visualization of board state and AI move suggestions. Allows users to correct misdetections from the CV module.

9.3 Syntax

9.3.1 Exported Constants

- `REFRESH_RATE` - frequency of UI updates

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderBoard	GameStateData	-	RenderError
displayAIMove	AIMove	-	-
getUserCorrections	-	List[Corrections]	-

9.4 Semantics

9.4.1 State Variables

- `uiState`: current UI rendering data

9.4.2 Environment Variables

- Display device or browser

9.4.3 Assumptions

Assumes the rendering environment is compatible with frontend framework and supports required update rate.

9.4.4 Access Routine Semantics

renderBoard(state):

- transition: updates UI elements to reflect the current game state
- output: -
- exception: raises `RenderError` if rendering fails

displayAIMove(move):

- transition: overlays AI recommended move on board
- output: -
- exception: -

getUserCorrections():

- output: returns user corrections for misdetected board state

9.4.5 Local Functions

- `updateDOM()`: handles DOM updates
- `highlightElements()`: highlights tiles, pieces, or moves

10 MIS of Game State Manager Module

10.1 Module

Game State Manager (M4)

10.2 Uses

Maintains the "Digital Twin" of the Catan board, validates moves, and enforces rules.

10.3 Syntax

10.3.1 Exported Constants

- `MAX_PLAYERS` = 4

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
updateState	MoveData	Boolean	InvalidMoveError
getState	-	GameStateData	-
validateMove	MoveData	Boolean	-

10.4 Semantics

10.4.1 State Variables

- `currentGameState`
- `playerAssets`: resources, settlements, roads

10.4.2 Environment Variables

- None

10.4.3 Assumptions

Assumes moves are provided in standard MoveData format and game rules are well-defined.

10.4.4 Access Routine Semantics

updateState(move):

- transition: applies move to `currentGameState`
- output: returns True if successful

- exception: raises `InvalidMoveError` if move is illegal

getState():

- output: returns `currentGameState`

validateMove(move):

- output: returns True if move is valid according to game rules

10.4.5 Local Functions

- `calculateResources()`, `updateScores()`, `checkVictory()`

11 MIS of Reinforcement Learning Environment Module

11.1 Module

Reinforcement Learning Environment (M5)

11.2 Uses

Simulates Catan rules and opponent behavior for AI training and evaluation.

11.3 Syntax

11.3.1 Exported Constants

- `MAX_TURNS`
- `REWARD_SCALE`

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
step	AIMove	GameStateData, Reward	StepError
reset	-	GameStateData	-
render	-	-	RenderError

11.4 Semantics

11.4.1 State Variables

- `simulatedState`
- `turnCount`

11.4.2 Environment Variables

- None

11.4.3 Assumptions

Assumes AI actions are in valid AIMove format and that simulated rules match official Catan rules.

11.4.4 Access Routine Semantics

step(action):

- transition: applies action to `simulatedState`, increments `turnCount`
- output: returns new state and reward
- exception: raises `StepError` if action is invalid

reset():

- transition: resets environment to initial state
- output: returns initial `simulatedState`

render():

- transition: visualizes `simulatedState`
- exception: raises `RenderError` on failure

11.4.5 Local Functions

- `applyOpponentLogic()`, `calculateReward()`

12 MIS of AI Model Module

12.1 Module

AI Model (M6)

12.2 Uses

Analyzes game states to recommend optimal moves using deep reinforcement learning. Provides move confidence scores and textual reasoning.

12.3 Syntax

12.3.1 Exported Constants

- `MODEL_PATH` - path to trained neural network weights

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
predictMove	GameStateData	AIMove	PredictionError
getMoveConfidence	AIMove	Float	-
explainMove	AIMove	String	-

12.4 Semantics

12.4.1 State Variables

- `modelWeights` - neural network parameters
- `policyNetwork` - DRL policy network

12.4.2 Environment Variables

- None

12.4.3 Assumptions

Assumes input game states are valid and model weights are loaded correctly.

12.4.4 Access Routine Semantics

predictMove(state):

- transition: evaluates state using policy network
- output: returns optimal move (`AIMove`)
- exception: raises `PredictionError` if model fails

getMoveConfidence(move):

- output: returns confidence score of move

explainMove(move):

- output: returns textual reasoning for move selection

12.4.5 Local Functions

- `preprocessState()`, `postprocessOutput()`

13 MIS of Game State Database Module

13.1 Module

Game State Database (M7)

13.2 Uses

Stores complete history of games and player actions. Provides access for analysis, replay, or post-game evaluation.

13.3 Syntax

13.3.1 Exported Constants

- `DB_PATH` - database file or server location
- `MAX_ENTRIES` - maximum stored states

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
writeState	GameStateData	Boolean	DBWriteError
readState	String	GameStateData	DBReadError
queryHistory	String	List[GameStateData]	DBReadError

13.4 Semantics

13.4.1 State Variables

- `dbConnection` - current database connection
- `gameRecords` - cached records in memory

13.4.2 Environment Variables

- File system or database server

13.4.3 Assumptions

Assumes database is accessible and schema matches expected structure.

13.4.4 Access Routine Semantics

writeState(state):

- transition: stores state in database
- output: returns True on success
- exception: raises `DBWriteError` if write fails

readState(gameID):

- output: returns `GameStateData` for specified game
- exception: raises `DBReadError` if retrieval fails

queryHistory(playerID):

- output: returns list of `GameStateData` for given player
- exception: raises `DBReadError` if query fails

13.4.5 Local Functions

- `serializeState()`, `deserializeState()`

14 MIS of Image Queue Module

14.1 Module

Image Queue (M8)

14.2 Uses

Provides low-latency asynchronous communication between CV, Game State Manager, AI Model, and UI modules.

14.3 Syntax

14.3.1 Exported Constants

- `QUEUE_SIZE` - maximum buffer size
- `TIMEOUT` - maximum wait time for enqueue/dequeue

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
enqueue	FrameData	Boolean	QueueFullError
dequeue	-	FrameData	QueueEmptyError
peek	-	FrameData	-

14.4 Semantics

14.4.1 State Variables

- `queueBuffer` - stores frames in order
- `head`, `tail` - pointers for buffer management

14.4.2 Environment Variables

- None

14.4.3 Assumptions

Assumes frames are correctly formatted and queue size is sufficient for intended throughput.

14.4.4 Access Routine Semantics

enqueue(frame):

- transition: adds frame to `queueBuffer`
- output: returns True if successful
- exception: raises `QueueFullError` if buffer is full

dequeue():

- transition: removes frame from `queueBuffer`
- output: returns next `FrameData`
- exception: raises `QueueEmptyError` if queue is empty

peek():

- output: returns next frame without removal

14.4.5 Local Functions

- `isEmpty()`, `isFull()`, `resetQueue()`

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), it 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)