Module Interface Specification for UnoFlip3D

Team 24

Mingyang Xu
Kevin Ishak
Jianhao Wei
Zain-Alabedeen Garada
Zheng Beng Liang

January 17, 2025

1 Revision History

Date	Developer	Notes
January 12th, 2025	Kevin Ishak	Initialize template, add rough draft of section 3,4 and 5
January 13th, 2025	Jianhao Wei	Add modules into section 6
January 13th, 2025	Zain-Alabedeen Garada	Add modules into section 6
January 17th, 2025	Zheng Bang Liang	Added all modules for Behavior Hiding Modules
January 17th, 2025	Jianhao Wei, Kevin Ishak, Zain-Alabedeen Garada	Modify behavior hiding modules that Zheng added and add the rest of the MIS modules
January 17th, 2025	Jianhao Wei	Modify section 3 and wrote section 4, 5, 16. Communicate with other members and wrote section 17
January 28th, 2025	Jianhao Wei	Make changes based on peer feedbacks. Please see commits and issue trackers for detail

2 Symbols, Abbreviations and Acronyms

See MG document in here.

Contents

3 Introduction

UNO Flip is a modern twist on the traditional UNO card game, incorporating an innovative double-sided card deck with "light" and "dark" sides. Players are challenged to adapt their strategies dynamically as the game flips between these two modes. Our goal for this project is to design and develop a digital version of UNO Flip that emulates the physical gameplay experience while adding features like automated rule enforcement, multiplayer support, and interactive animations.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found in here.

4 Notation

The structure of the MIS for modules comes from **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. Template used from the SFWRENG 4G06 GitHub in here.

The following table summarizes the primitive data types used by UnoFlip3D.

Data Type	**Notation**	**Description**
Boolean	boolean	A variable that represent true or false on a statement
Integer number	int	A number without fractional part with the range between $[-2^{63}, 2^{63} - 1]$
Decimal number	float	A number with fractional part represented by 32-bit single-precision float point
Data Stream	serializedData	A stream of binary data for inter-module or inter-device transmissions

The following table summarizes the derived object data types used by UnoFlip3D

Object Type	Notation	Description
String Object	String	an object with a sequence of unicode characters that represent word or sentences
Generic Array Object	Array[Type]	A object represented by a set of certain type of variable
Dictionary Object	dictionary	A collection of two arrays with each variable in one array correspond to a specific element in another array
Graphics Description Object	GraphicObject	A object contain all the details that the user interface needed for display the game properly

UnoFlip3D 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 description is taken directly from the Module Guide document for this project. The modules are divided into 3 main categories: Hardware-Hiding, Behaviour-Hiding and Software Decision. Below is a detailed description about how each modules is categorized:

5.1 Hardwire-Hiding Modules

- **Backend/Server Module:** Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software, allowing the system to display outputs or accept inputs.

5.2 Behaviour-Hiding Modules:

- **Card Effect Module:** Executes the effects of special cards and updates the game state accordingly
- **Turn Management Module:** Manages the order of player turns, including handling special conditions like "Reverse" or "Skip" cards.
- **User Interface Module:** Displays the game state to the user and accepts user inputs through various interactive elements.
- **Save/Load Module:** Allows saving the current game state and loading it at a later time.

- **Animation Module:** Provides animations for card movements, flips, and game interactions.
- **Output Module:** Provides visual or textual outputs to the user based on the game state

5.3 Software Decision Modules

- **Multiplayer Networking Module:** Handles communication between players, including matchmaking, game state synchronization, and latency management.
- **Verification Output Module:** Validates the final output of the game, ensuring compliance with rules and expected results.
- **Input Module:** Converts the input data into the data structure used by other modules, such as the game logic or UI modules.

6 MIS of Backend/Server Module

6.1 Module

Backend/Server

6.2 Uses

None

6.3 Syntax

6.3.1 Exported Constants

- SUPPORTED_DEVICES: Enumerates the supported hardware devices
- DEFAULT_RESOLUTION: Specifies the default screen resolution for the game.

6.3.2 Exported Access Programs

- initializeHardware()
- captureInput()
- renderOutput(graphicsData: GraphicsObject)
- detectHardware()

6.4 Semantics

6.4.1 State Variables

- connectedDevices: Tracks the list of currently connected input/output devices.
- currentResolution: Stores the current screen resolution of the application.

6.4.2 Environment Variables

- hardwareDrivers: Represents the drivers required to interface with the supported hardware.
- platform: Indicates the operating system or platform the game is running on

6.4.3 Assumptions

- All required hardware drivers are installed and operational.
- The platform supports Unity's hardware abstraction layer.

6.4.4 Access Routine Semantics

- initializeHardware() → void
  Transition: Sets up the required hardware connections and initializes drivers.
- captureInput() → String
  Output: Returns a structured object representing raw input data from connected devices.
- renderOutput(graphicsData: GraphicsObject) → void
  Transition: Translates graphical data into visual outputs using the rendering hardware.
- detectHardware() → Array[String]
  Output: Returns a list of hardware devices detected and compatible with the game.

6.4.5 Local Functions

- verifyDeviceSupport() → boolean
  Description: Checks if the provided device is supported by the game.
- applyDriverSettings() → void
  Description: Configures hardware drivers based on the detected platform.
- fallbackToDefault() → void
  Description: Reverts to default hardware settings if the required device is not detected or initialization fails.

7 MIS of Card Effect Module

7.1 Module

Card Effect

7.2 Uses

Hardwire Hiding

7.3 Syntax

7.3.1 Exported Constants

- DRAW_TWO_EFFECT: Specifies the effect identifier for a "Draw Two" card.
- SKIP_TURN_EFFECT: Specifies the effect identifier for a "Skip" card.
- FLIP_DECK_EFFECT: Specifies the effect identifier for a "Flip" card.

7.3.2 Exported Access Programs

- reverseDirection(playerId: int)
- skipTurn(playerId: int)
- triggerDrawCards(playerId: int, cardCount: Int)
- flipDeck()

7.4 Semantics

7.4.1 State Variables

- currentEffect: Track the current effect being applied
- effectQueue: Store the effects that are waiting to be applied

7.4.2 Environment Variables

- The special card type that current game environment allowed.

7.4.3 Assumptions

- All card effects are predefined.
- The "Flip" card effect toggles the entire game state between "light" and "dark" sides.

7.4.4 Access Routine Semantics

- reverseDirection(playerId: int, previousPlayerId: int): → void
  Transition: Reverse the direction the game is being played to the previous player by re-assigning the previous player next opportunity

- skipTurn(playerId: int, nextPlayerId: int) → void
  Transition: Skip the opportunity for the specified player to play and assignment the opportunity to the next player

- triggerDrawCards(playerId: int, cardCount: Int) → void
  Transition: Let the player specified to draw another card into their database and add the card count to their totalPower variable.

- flipDeck() → void
  Transition: Changes the deckSide variable and updates the game state to reflect the flipped deck.

7.4.5 Local Functions

- calculateNextPlayer(direction: String) → int
  Description: Determines the next player in the turn sequence after applying a "Skip" or "Reverse" effect.

- applyChainEffect(effectQueue: Array[String]) → void
  Description: Resolves multiple card effects in sequence defined in the input string array

- toggleDeckSide() → void
  Description: Switches the game state between "light" and "dark" sides during a "Flip" card effect.

8 MIS of Turn Management Module

8.1 Module

Turn Management

8.2 Uses

Input, Card Effect

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

- validateMove(playerId: int, cardid: int)
- endTurn(playerId: int)
- shuffleDeck()
- drawCard(playerId: int)

8.4 Semantics

8.4.1 State Variables

- currentPlayer: Tracks the player whose turn it is
- deck: Represents the stack of remaining cards in the game.
- discardPile: Stores played cards
- playerHands: Stores each player's cards

8.4.2 Environment Variables

- maxPlayers: Maximum number of players allowed in a game
- flipEnabled: Boolean to toggle the flip functionality

8.4.3 Assumptions

- The number of players, game rules, player restrictions are preloaded
- The game environment is known

8.4.4 Access Routine Semantics

- validateMove(playerId: int, cardId: int) → boolean
  Output: Checks if a move is valid
- endTurn(playerId:int) → void
  Transition: Ends the current player's turn and starts the next

- shuffleDeck() → void
  Transition: Randomizes the card deck

- drawCard(playerId: int) → void
  Transition: Adds a card to the specified player's hand

8.4.5 Local Functions

- shuffleProcess(original: Array[String]) → Array[String]
  Description: Contain the random algorithm to shuffle the deck

- CardModifier(cardId: int) → void
  Description: Contain algorithm to draw different card to screen

9 MIS of User Interface Module

9.1 Module

User Interface

9.2 Uses

Output, Turn Management

9.3 Syntax

9.3.1 Exported Constants

- DEFAULT_THEME: Specifies the default theme for the game UI (e.g., light mode).
- FONT_STYLE: Default font style used across UI elements.
- ASSET_PATH: Directory path where assets are stored
- DEFAULT_CARD_SPRITE: Specifies the default card sprite to use if none is provided.

9.3.2 Exported Access Programs

- updateCardDisplay(playerId: int, cardId: int)
- showTurnIndicator(playerId: int)
- displayMessage(message: String)
- loadScene(type: String)
- loadAsset(assetName: String)

- unloadAsset(assetName: String)
- playSound(effectName: String)

9.4 Semantics

9.4.1 State Variables

- displayedCards: Tracks the cards currently visible for each player.
- turnIndicator: Indicates which player's turn it is.
- messageQueue: Stores pending notifications or chat messages to be displayed.
- theme: Specifies the current visual theme in light mode or dark mode.
- loadedAssets: Tracks assets currently loaded into memory.
- audioSettings: Stores configuration for playing audio
- assetCache: Cache for frequently accessed assets to improve performance.
- assetDirectory: Path to the directory containing all assets

9.4.2 Environment Variables

- The resolution of the device being used.

9.4.3 Assumptions

- The UI module assumes that game state updates from the multiplayer networking and turn management modules are reliable.
- All required assets are preloaded by the Save/Load module.
- Multiplayer synchronization ensures accurate real-time updates across all connected devices.
- All assets are correctly named and stored in the specified directory.
- The module assumes sufficient memory and storage are available for caching assets.
- Dependencies for visual and audio formats are preinstalled on the system.

9.4.4 Access Routine Semantics

- updateCardDisplay(playerId: int, cardId: int) → void
  Transition: Updates the player's visible hand to reflect the current state of their cards.

- showTurnIndicator(playerId: int) → void
  Transition: Highlights the current player's turn using visual indicators.

- displayMessage(message: String) → void
  Transition: Displays a notification or chat message on the game screen.

- loadScene(type: String) → void
  Transition: Load specific type of background with animation to the user interface

- loadAsset(assetName: String) → void
  Transition: Loads the specified asset from the asset directory into memory and returns a reference.

- unloadAsset(assetName: String) → void
  Transition: Removes the specified asset from memory to free up resources.

- playSound(effectName: String) → void
  Transition: Plays the specified sound effect from the audio assets directory.

9.4.5 Local Functions

- applyTheme(themeId: int) → void
  Description: Configures and applies the selected theme for the game UI.

- renderMessageQueue(messages: Array[String]) → void
  Description: Processes and displays pending messages in the queue.

- adjustUILayout() → void
  Description: Dynamically adjusts the layout based on the screen resolution and device type.

- cacheAsset(assetName: String) → void
  Description: Adds the specified asset to the cache for quick retrieval.

- clearCache() → void
  Description: Clears the asset cache to free up memory

- validateAsset(assetName: String) → void
  Description: Checks if the specified asset exists and is accessible.

10 MIS of Save/Load Module

10.1 Module

Save/Load

10.2 Uses

Hardwire Hiding

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

- save(info: String, description: String)
- retrieve(description: String)
- delete(description: String)
- changeDesc(originalDesc: String, updateDesc: String)

10.4 Semantics

10.4.1 State Variables

- ifFull: Track if the database is full
- dict: The dictionary that stores the array index correspond with descriptions
- infoArray: The array that stores all the information

10.4.2 Environment Variables

None

10.4.3 Assumptions

The string and description stored does not contain any special characters

10.4.4 Access Routine Semantics

- save(info: String, description: String) → void
  Transition: Save the information into the database with description
- retrieve(description: String) → String
  Output: Return the information by its description
- delete(description: String) → void
  Transition: Delete the information in the database by its description
- changeDesc(originalDesc: String, updateDesc: String) → void
  Transition: change the description of a piece of information into another

10.4.5 Local Functions

- returnIndex(description: String) → int
  Description: Return the index of the infoArray based on the description.

11 MIS of Animation Module

11.1 Module

Animation

11.2 Uses

User Interface, Card Effect, Save/Load

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

- move(cardId: int, distance: int, direction: String)
- flip(cardId: int)
- select(cardId: int)
- appear(cardId: int)
- disappear(cardId: int)

11.4 Semantics

11.4.1 State Variables

- cardSide: Track side the card is on
- cardColor: Track the color of the card
- cardPosition: Track the position of the card
- show: Track if the card is shown on the screen

11.4.2 Environment Variables

None

11.4.3 Assumptions

Each card has a unique id

11.4.4 Access Routine Semantics

- move(cardId: int, distance: int, direction: String) → void
  Transition: Move the card with specific id by a set amount of pixels with horizontal or vertical direction
- flip(cardId: int) → void
  Transition: Flip the card with specific id to show the opposite face
- select(cardId: int) → void
  Transition: Show the animation when the card is selected by the user
- appear(cardId: int) → void
  Transition: Show the card with specific id to the user screen
- disappear(cardId: int) → void
  Transition: Make the card with specific id to disappear from the user screen

11.4.5 Local Functions

- getCardInfo(id: int) → void
  Description: Get the info of the card to local state variables
- applyVisualElements(id: int) → void
  Description: Apply the visual effect to the user screen based on the id provided and update local state variables

12 MIS of Output Module

12.1 Module

Output

12.2 Uses

Card Effect

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

- render(info: String, font: int, color: String, location: int)
- showCardEffect(id: int, effectNum: int)

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

None

12.4.3 Assumptions

Each card has a unique id

12.4.4 Access Routine Semantics

- render(info: String, font: int, color: String, location: int) → void
  Transition: Display the information onto the screen with the font, color and location specified
- showCardEffect(id: int, effectNum: int) → void
  Transition: Using Card Effect module to show flip, skip or draw two on specific card

12.4.5 Local Functions

- checkEdge(font: int, location: int) → boolean
  Description: Check if the information displayed exceeds the boundary of the screen

13 MIS of Multiplayer Networking Module

13.1 Module

Multiplayer Networking

13.2 Uses

Verification Output, Save/Load, Animation

13.3 Syntax

13.3.1 Exported Constants

serverID: The serial number of the game room upon user request

13.3.2 Exported Access Programs

- createGameRoom(playerId: int, roomSettings: Array[String])
- joinGameRoom(playerId: int, roomId: int)
- broadcastUpdate(gameId: int, update: String)

13.4 Semantics

13.4.1 State Variables

- activeGames: Tracks all ongoing game sessions.
- connectedPlayers: List of currently connected players.

13.4.2 Environment Variables

- serverIP: IP address of the game server.
- timeoutLimit: Time limit for a player to respond during their turn.

13.4.3 Assumptions

- The connection between server and other machines can be established successfully
- The encryption and decryption methods are known

13.4.4 Access Routine Semantics

- createGameRoom(playerId: int, roomSettings: Array[String]) → void
  Transition: Creates a new game room by a specific user with specific setting
- joinGameRoom(playerId: int, roomId: int, publicKey: int) → int
  Transition: Adds a specific player to an existing room by its ID and public key for encryption and decryption purpose purposes
  Output: Return the public key of the server for encryption and decryption purposes
- broadcastUpdate(gameId: int, update: String) → void
  Transition: Sends game state updates to all players in a room.

13.4.5 Local Functions

- encryption(information: String, publicKey: int) → String
  Description: Contain encryption algorithm to encrypt data before sending using public key from user
- decryption(information: String, privateKey: int) → String
  Description: Contain decryption algorithm to decrypt data after receiving using the private key of game room

14 MIS of Verification Output Module

14.1 Module

Verification Output

14.2 Uses

None

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

- captureOutput(playerId: int, info: String)
- validateOutput(info: String)

14.4 Semantics

14.4.1 State Variables

- outputBuffer: Temporarily store the incoming input received for later use
- validatedOutput: Store the input that has been validated by the module for later transmission

14.4.2 Environment Variables

- the validation algorithm the device is running on

14.4.3 Assumptions

- All output devices conform to Unity's input standard.
- The validation algorithm must make sure that there is no error or discrepancy occurring after the validation

14.4.4 Access Routine Semantics

- captureInput(info: String) → void
  Transition: Capture and save the information into the output buffer
- validateInput(info: String) → String, boolean
  Transition: validate the output from the outputBuffer using existing algorithms
  Output: Return the original output and a boolean indicating if the input can be validated

14.4.5 Local Functions

- algorithmDatabase(input: String, type: int) → String
  Description: Contain the algorithm that converts the input string to the format that can be used by other modules and return the converted input string
- serialization(input: String) → serializedData
  Description: Contain the algorithm to convert the input string into serialized data for inter-module or internet communications

15 MIS of Input Module

15.1 Module

Input

15.2 Uses

Hardwire Hiding

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

- captureInput(playerId: int, info: String)
- validateInput(info: String)
- convertInput(info: String, type: String)

15.4 Semantics

15.4.1 State Variables

- inputBuffer: Temporarily store the incoming input received for later use
- validatedInput: Store the input that has been validated by the module for later transmission

15.4.2 Environment Variables

- The version of supporting device that the software is running on
- the validation algorithm the device is running on

15.4.3 Assumptions

- All input devices conform to Unity's input standard.
- The validation algorithm must make sure that there is no error or discrepancy occurring after the validation

15.4.4 Access Routine Semantics

- captureInput(playerId: int, info: String) → void
  Transition: Capture and save the information into the input buffer

- validateInput(info: String) → String, boolean
  Transition: validate the input from the inputBuffer using existing algorithms
  Output: Return the original input and a boolean indicating if the input can be validated

- convertInput(action: String, type: String) → String
  Output: Convert the input from validatedInput into specific format that can be used by other modules

15.4.5 Local Functions

- algorithmDatabase(input: String, type: int) → String
  Description: Contain the algorithm that converts the input string to the format that can be used by other modules and return the converted input string

- serialization(input: String) → serializedData
  Description: Contain the algorithm to convert the input string into serialized data for inter-module or internet communications

16 Exception Handling Strategies

The exception handling is critical for our software since it directly impacts the user experience of our software. It is our responsibility to ensure that our customers have a good experience with our software. To prevent exception from happening in our software, we implement the following 4 strategies:

- **Limit Erroneous User Input:** We design the user interface such that the user input is bounded within a certain range to limit erroneous user input that might crash the software. We also include the input verification in Input module to ensure all the information that passed to the software are legitimate

- **Wrap External Resources:** We have design all of our function in our modules to wrap the resources and libraries they use from the global space of the software. This ensures that the exception in third-party software does not impact the integrity of own software.

- **Cleaning up resources:** We have implemented the mechanism to clean up unused resource promptly and reliably to make sure the exceptions do not occurs due to cache overload

- **Limiting Errors Instead of Handling Errors:** Instead of designing exception handling mechanism, we make sure our software is carefully designed and tested to reduce the chance of exception happening.

By implementing these strategies, we can reduce the chance for exception happening and limit the need of the exception handling mechanisms.

References

1. M. Xu, "UNO Flip 3D - SRS Volere Documentation," *GitHub Repository*, 2023.
2. M. Xu, "UNO Flip 3D - Software Architecture Document," *GitHub Repository*, 2023.

17 Appendix

Appendix — Reflection

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

1. What went well while writing this deliverable?

   Our team is able to divide the task very fast, and everyone is working more collaboratively than before. We also started this deliverable earlier than before and be able to present the rough draft during the informal TA meeting. We get lots of feedback from our TA and we are confident that we can get a higher grade than before. In terms of documents, the module guide is relatively quick to do. We are able to gather information very fast and the communication went really well.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   The pain in this deliverable is the implementation of MIS document. The function and variables in every module are hard to visualize because the relationship between the module are very complex and everyone have their own opinion about how the implementation. We have to have extended meeting session to discuss about concept and resolve the conflict between different team members. We tried to absorb the advantages from the opinions from different team member and coming to an integrated idea. But this take a lot of time.

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?

   The MIS are mainly coming from communicating with our peers and our supervisor. But for software architecture part of our decision, we simply reading the materials on the internet such as past project to get a better feeling about the most abstract part of our implementation.

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?

   The document requirements need to be change because we discovered new idea while implementing our project. There are some requirement in the original document doesn't fit with the software architecture we chose. Some of the requirement are also too vague to be implemented properly, some of the requirement are too hard to implement in the actual software. We have to change our SRS document according to the

software architecture and modules we implemented, and remove the unnecessary and vague requirement to make our documentation more consistent and integrate.

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)

   The limitations of our solution is that we never implemented similar software before and this is our first time experience with this kind of project. Even though we have internet but our view is still limited. If the resource and time is unlimited, we would consult with professional people (such as professionals from game companies) to get a better understanding about how should this project be implemented and what is the more efficient way to build this kind of project and managment teams.

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)

   We had considered about design the same game using AI powered opponent with a single player. The AI solution will more cool and this solution will become more convenient with people who are lonely and don't have the access of the internet. But, the AI solution is also much more challenging and the chance of failure is much greater. The single player might also be less popular than multiplayer game for the public since multiplayer game are more fun and engaging. We select our current solution because it achieve the trade-off between implementation difficulties and public popularity.