Module Guide for UNO-Flip

Team 24

Mingyang Xu
Jiahao Wei
Kevin Ishak
Zain-Alabedeen Garada
Zheng Bang Liang

January 15, 2025

1 Revision History

Date	Version	Notes
2025 Jan 13	1.0	Start section 3-5
2025 Jan 14	1.1	Start section 5-7
2025 Jan 15	1.2	Start section 7-11

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
	Explanation of program name
UC	Unlikely Change
etc.	...

Contents

List of Tables

List of Figures

3 Introduction

Developing software through modular decomposition is a well-accepted practice in software engineering. A module is defined as a work assignment for a programmer or programming team 24. This approach allows the software system to be divided into manageable components, enhancing scalability, maintainability, and adaptability. In this project, we follow the principle of information hiding (**?**), which is essential for accommodating changes during development and maintenance. Given the dynamic and interactive nature of the UNO Flip game, this principle is particularly relevant to ensure flexibility as new features or rules may be incorporated in the future.

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.

To ensure a robust and maintainable design, we adhere to the modular design principles established by **?**:

- System details that are likely to change independently, such as game rules or graphical user interface (GUI) elements, are encapsulated within separate modules.
- Each data structure, such as the card deck, player hands, or game state, is implemented in only one module.
- Inter-module communication is facilitated through well-defined access programs to ensure data encapsulation and minimize dependencies.

After completing the initial design phase, including the Software Requirements Specification (SRS), we created the Module Guide (MG) (**?**). The MG outlines the modular structure of the system and serves as a comprehensive reference for all stakeholders. This document is aimed at the following audiences:

- **New project members:** The MG provides a clear overview of the system's modular structure, enabling new members to understand the architecture and locate relevant modules efficiently.
- **Maintainers:** The hierarchical organization of the MG simplifies the process of identifying and updating the relevant modules when changes are made to the system. Maintainers are encouraged to update the MG to reflect any modifications accurately.
- **Designers:** The MG acts as a verification tool to ensure consistency, feasibility, and flexibility in the system design. Designers can assess the coherence among modules, the viability of the decomposition, and the adaptability of the design to future changes.

The rest of this document is structured as follows. Section 4 outlines the anticipated and unlikely changes in the software requirements, such as the introduction of new game rules or enhancements to the multiplayer functionality. Section 5 describes the module decomposition strategy, constructed based on the principle of likely changes. Section 6 details the connections between the software requirements and the modules. Section 7 provides a comprehensive description of each module, including its responsibilities and dependencies. Section 8 includes traceability matrices to ensure the completeness of the design relative to the requirements and anticipated changes. Finally, Section 9 discusses the use relationships between modules, highlighting their interactions and dependencies.

This modular approach ensures that our UNO Flip software can be developed efficiently while maintaining the flexibility to adapt to future requirements and user feedback.

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.
Explanation: The game should adapt to various hardware platforms, such as desktops, mobile devices, and consoles, by leveraging Unity's cross-platform build tools.

AC2: The format of the initial input data.
Explanation: Unity's Input System allows for flexibility in handling various input methods, such as touch, gamepad, or keyboard input, without impacting the overall game functionality.

AC3: Updating the user interface for accessibility and improved user experience.
Explanation: Changes in UI elements, such as adding new accessibility features or redesigning layouts, can be achieved without affecting the underlying game logic due to Unity's modular UI Toolkit.

AC4: The communication protocol for multiplayer functionality.
Explanation: Switching from an existing protocol (e.g., UDP) to a more secure or efficient one (e.g., WebSockets) should only require modifications to the networking module.

Anticipated changes relate to changes that would be made in requirements, design or implementation choices. They are not related to changes that are made at run-time, like the values of parameters.

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 decisions 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: File and/or Keyboard, Output: File, Memory, and/or Screen).
Explanation: Changing I/O devices is considered unlikely since the Unity Input System already supports a wide range of devices, and most use cases are covered by the current implementation.

UC2: Switching the game engine from Unity to another platform.
Explanation: Rebuilding the game using a different engine would require re-implementation of all assets, scripts, and logic, making this change highly unlikely.

UC3: Fundamental changes to the core gameplay mechanics.
Explanation: Altering the basic rules of UNO Flip, such as removing the "Flip" mechanic, would require significant rewrites across multiple modules.

UC4: Replacing all graphical assets with a new visual theme.
Explanation: Although possible, replacing all assets would involve modifying Unity scenes, prefabs, and animations extensively, which is not a likely requirement.

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
Purpose: This module isolates hardware-specific dependencies, such as rendering devices, input devices, and memory management. It allows the rest of the system to remain platform-independent by using Unity's built-in hardware abstraction.

M2: Behaviour-Hiding Module
Purpose: Encapsulates the core gameplay logic, such as player actions, turn mechanics, and rule enforcement, ensuring that other modules can interact with game behavior through a unified interface.

M3: Software Decision Module

Purpose: Handles system-wide decisions, including UI updates, multiplayer network protocols, and game state transitions. This module integrates decisions that impact user interaction and system-wide consistency.

Level 1	**Level 2**
Hardware-Hiding Module	*Hardware Abstraction module: Isolate hardware-specific dependencies, such as rendering devices and input/output handling.*
Behaviour-Hiding Module	Game Logic Module: Handles player actions, turn mechanics, and card rules enforcement. Turn Management Module: Ensures correct sequencing of turns, including handling "Skip" or "Reverse" cards. Card Effect Module: Implements special card effects such as "Flip" or "Draw Two." Score Tracking Module: Manages player scores and determines game-winning conditions. Animation Module: Handles animations for card movements, flips, and effects using Unity's animation system.
Software Decision Module	UI Module: Manages user interface components, including menus, HUD, and accessibility options. Multiplayer Networking Module: Handles online player interactions, matchmaking, and game state synchronization. Save/Load Module: Provides functionality for saving and loading game progress.

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.

The intention of this section is to document decisions that are made "between" the requirements and the design. To satisfy some requirements, design decisions need to be made. Rather than make these decisions implicit, they are explicitly recorded here. For

instance, if a program has security requirements, a specific design decision may be made to satisfy those requirements with a password.

6.1 R1: Basic Game Interaction and Hardware Support

The system must provide basic interaction capabilities, including:

- Managing player turns through a turn management system.
- Handling user inputs from hardware devices and abstracting them into standard data formats.
- Displaying the game state through a user interface.
- Ensuring compatibility with hardware platforms via a hardware-hiding module.

6.2 R2: Input Processing and Special Card Effects

The system must process user inputs and execute card effects accurately, including:

- Standardizing user inputs into a format usable by the game logic.
- Implementing special card effects, such as "Flip," "Skip," and "Draw Two," to update the game state accordingly.

6.3 R3: Verification and Score Tracking

The system must verify the correctness of game outcomes and track player scores, including:

- Ensuring compliance with game rules and expected results.
- Recording and updating player scores dynamically.
- Providing a summary of scores at the end of the game.

6.4 R4: Output Rendering and Game Control

The system must render game outputs and manage overall control flow, including:

- Displaying game outputs, such as scoreboards and game results.
- Managing state transitions and ensuring synchronization between modules.
- Handling error states gracefully.

6.5 R5: Animation and Special Card Interaction

The system must support animations and ensure proper handling of special card interactions, including:

- Providing smooth animations for card movements and flips.
- Ensuring special card effects are visually represented and correctly executed.

6.6 R6: Advanced Game Outputs and Interactions

The system must handle complex interactions and advanced output scenarios, including:

- Supporting additional game scenarios with detailed outputs.
- Ensuring consistency in interactions across all modules.

6.7 R7: User Interface and Score Integration

The system must integrate user interface elements with score tracking, including:

- Displaying dynamic score updates during gameplay.
- Providing interactive elements for players to review their scores.

6.8 R8: Animation and User Interaction Synchronization

The system must synchronize animations with user interactions, including:

- Ensuring animations are consistent with game logic.
- Allowing user inputs to interrupt or modify animations as needed.

6.9 R9: Save and Load Functionality

The system must provide save and load functionality, including:

- Saving the current game state to a file or database.
- Loading a saved game state and resuming gameplay seamlessly.

6.10 R10: Output and Special Card Integration

The system must integrate output rendering with special card effects, including:

- Displaying the outcomes of special card effects clearly.
- Ensuring game outputs are consistent with the effects of special cards.

6.11 R11: Advanced Scoring and Control Logic

The system must handle advanced scoring scenarios and control logic, including:

- Supporting complex scoring rules and tie-breaking scenarios.
- Managing game control logic for multiplayer and advanced game modes.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. 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. means the module will be implemented by the 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 HH (Hardware-Hiding Module)

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, allowing the system to display outputs or accept inputs.

Implemented By: OS

7.2 BH (Behaviour-Hiding Module)

Secrets: The contents of the required behaviours, including turn management, game logic, and card effects.

Services: Includes programs that provide externally visible behaviours 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 IM (Input Module)

Secrets: The format and structure of the input data, including data serialization for communication.

Services: Converts the input data into the data structure used by other modules, such as the game logic or UI modules.

Implemented By:

Type of Module: Abstract Data Type

7.2.2 CE (Card Effect Module)

Secrets: The implementation details of special card effects, such as "Flip," "Skip," and "Draw Two."

Services: Executes the effects of special cards and updates the game state accordingly.

Implemented By:

Type of Module: Abstract Object

7.2.3 TM (Turn Management Module)

Secrets: The sequence and rules for determining the current player.

Services: Manages the order of player turns, including handling special conditions like "Reverse" or "Skip" cards.

Implemented By:

Type of Module: Record

7.3 OM (Output Module)

Secrets: The formatting and rendering of game outputs, such as scoreboards and game results.

Services: Provides visual or textual outputs to the user based on the game state.

Implemented By:

Type of Module: Abstract Object

7.4 CM (Control Module)

Secrets: The logic governing the control flow of the game, including state transitions and error handling.

Services: Manages the overall game control, ensuring synchronization between modules.

Implemented By:

Type of Module: Library

7.4.1 VO (Verification Output Module)

Secrets: The methods used to verify the correctness of game outputs.

Services: Validates the final output of the game, ensuring compliance with rules and expected results.

Implemented By:

Type of Module: Abstract Object

7.4.2 SL (Save/Load Module)

Secrets: The structure and format of saved game data.

Services: Allows saving the current game state and loading it at a later time.

Implemented By:

Type of Module: Library

7.4.3 AM (Animation Module)

Secrets: The implementation details of animations, including transitions and visual effects.

Services: Provides animations for card movements, flips, and game interactions.

Implemented By:

Type of Module: Abstract Object

7.5 SD (Software Decision Module)

Secrets: The design decisions based on performance optimizations, networking protocols, and user interaction considerations. These secrets are *not* described in the SRS.

Services: Includes data structures and algorithms used in the system that do not provide direct interaction with the user, such as multiplayer matchmaking and game state synchronization.

Implemented By:

7.5.1 MN (Multiplayer Networking Module)

Secrets: The implementation details of real-time communication, including protocols like UDP or WebSocket.

Services: Handles communication between players, including matchmaking, game state synchronization, and latency management.

Implemented By:

Type of Module: Library

7.5.2 UI (User Interface Module)

Secrets: The layout and design of user interface components, such as menus, HUD, and accessibility features.

Services: Displays the game state to the user and accepts user inputs through various interactive elements.

Implemented By:

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	HH, IM, TM, UI
R2	IM, CE
R3	VM, ST
R4	OM, CM, UI
R5	CE, TM, CM, AM
R6	OM, CE, CM, TM
R7	UI, ST, CM, TM
R8	UI, AM, CM, TM
R9	VO, SL
R10	OM, CE, CM
R11	OM, CE, ST, CM

Table 2: Trace Between Requirements and Modules

AC	Modules
Hardware Abstraction Change	HH
Input Format Change	IM
Turn Management Change	TM
UI Adjustment	UI
Card Effect Modification	CE
Verification Update	VM, VO
Output Adjustment	OM
Animation Update	AM
Save/Load Mechanism Update	SL
Control Logic Update	CM
Score Tracking Update	ST

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** 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.

The uses relation is not a data flow diagram. In the code there will often be an import statement in module A when it directly uses module B. Module B provides the services that module A needs. The code for module A needs to be able to see these services (hence the import statement). Since the uses relation is transitive, there is a use relation without

an import, but the arrows in the diagram typically correspond to the presence of import statement.

If module A uses module B, the arrow is directed from A to B.

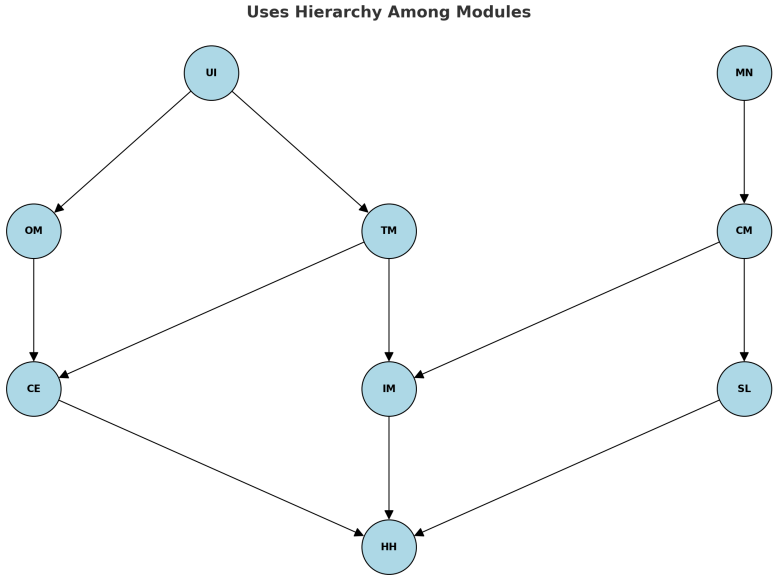

Figure 1: Use hierarchy among modules

10 User Interfaces

The user interface (UI) design for this system focuses on creating an intuitive and accessible experience for players. The UI consists of both software and hardware elements that enable seamless interaction with the game. Below are the primary components of the UI design:

10.1 Software User Interface

- **Main Menu:** The main menu provides options for starting a new game, loading a saved game, viewing instructions, and accessing settings.
- **Game Screen:** The game screen displays the game state, including player scores, active cards, and the current turn. Interactive elements allow players to perform actions such as playing a card or drawing from the deck.
- **Pop-Up Notifications:** Notifications are used to inform players about special events, such as a "Skip" card being played or a turn being reversed.

10.2 Hardware User Interface

- **Input Devices:** Supports mouse, keyboard, and touchscreens for input.

- **Output Devices:** Designed for compatibility with various screen sizes and resolutions.

11 Design of Communication Protocols

The communication protocols in this system ensure reliable and efficient data exchange between modules, especially for multiplayer functionality. These protocols are designed to handle various aspects such as real-time data synchronization, error handling, and security.

11.1 Protocol Overview

- **Transport Layer:** Uses WebSocket protocol for real-time communication in multiplayer games. Ensures low latency and high reliability.
- **Message Format:** Messages are serialized using JSON for simplicity and compatibility. Each message includes a header (e.g., message type) and a body (e.g., card played, turn updated).

11.2 Error Handling and Recovery

- **Connection Loss:** Implements a reconnection mechanism to recover lost connections without disrupting the game state.
- **Invalid Messages:** Validates incoming messages to ensure compliance with the protocol specification. Invalid messages are logged and discarded.

11.3 Security

- **Encryption:** All data exchanged between clients and the server is encrypted using TLS to ensure data security.
- **Authentication:** Players are authenticated via a token-based system to prevent unauthorized access.

Appendix **??** contains detailed diagrams and examples of protocol specifications.

12 Timeline

Schedule of tasks and who is responsible

You can point to GitHub if this information is included there

References