Module Guide for UNO Flip Remix

Team 24

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

April 4, 2025

1 Revision History

Date	Version	Notes	Developer
2025 Jan 13	1.0	Start section 3-5	Mingyang Xu
2025 Jan 14	1.1	Start section 5-7	Mingyang Xu
2025 Jan 15	1.2	Start section 7-11	Mingyang Xu
2025 Jan 17	1.3	Fix small error based on TA's feedback	Mingyang Xu
2025 Mar 31	2.0	Comprehensive revision based on TA feedback for final report	Kevin Ishak

Table 1: Revision History

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
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 POIH, 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.

Unlike traditional UNO, UNO Flip introduces additional complexity via the “Flip” card, which causes all players to switch their hands, draw piles, and discard piles to the opposite side (light to dark or vice versa). Each side contains different types of action cards — for example, the dark side includes more punishing cards such as Draw Five, Skip Everyone, and Wild Draw Color. These dynamic rule changes make the game unpredictable and require more advanced rule enforcement logic. Implementing this flipping mechanism digitally — especially while synchronizing state across multiple players in a real-time networked game — significantly increases design complexity.

To ensure a robust and maintainable design, we adhere to the modular design principles established by David L. Parnas

- 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 Volere, 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.

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 2. 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.~~ *Purpose:* Isolates hardware-specific dependencies such as rendering devices, input/output handling, and system memory. Enables the rest of the system to remain platform-independent by leveraging Unity's hardware abstraction layer.

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.~~ *Purpose:* Encapsulates the system-wide logic that connects user interaction with game state, such as UI updates, animation transitions, turn sequencing, card effects, and score tracking. This layer maintains internal logic without exposing implementation details.

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.~~ *Purpose:* Contains logic specific to internal implementation choices, such as communication protocols (e.g., TCP/WebSocket), save/load functionality, and architectural constraints that are not visible to the user. These decisions are not described in the SRS but are essential for system operation.

The module hierarchy table has been updated to improve clarity and consistency with the module decomposition described throughout this document. All module names have been fully spelled out for readability, the Verification Module has been added to align with the SRS and MIS, and the Turn Management Module has been reclassified as an Abstract Object to better reflect its role in implementing logic rather than merely holding data. These revisions address prior inconsistencies and ensure the design more accurately reflects the implemented architecture.

Level 1	**Level 2**
Hardware-Hiding Module	Hardware Abstraction Module: Isolates platform-specific rendering and input/output concerns.
Behaviour-Hiding Module	Game Logic Module: Manages turn flow, rule checking, and state updates during gameplay.
	Turn Management Module: Sequences player turns and handles logic for ”Skip”, ”Reverse”, and ”Draw” cards. (Abstract Object)
	Card Effect Module: Executes effects of special cards like “Flip” and “Draw Five.”
	Score Tracking Module: Calculates and updates scores in real time and at end-of-game.
	UI Module: Manages interface elements such as player HUD, hand display, and animations.
	Animation Module: Controls card transitions, flip animations, and victory/loss feedback.
	Verification Module: Checks rule enforcement, illegal moves, and final game validity.
Software Decision Module	Multiplayer Networking Module: Implements client-server communication, matchmaking, and state syncing.
	Save/Load Module: Manages persistence of game state across sessions.

Table 2: 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 3.

To improve traceability, we now refer to specific module identifiers using `\mref{}` where appropriate. Additionally, details have been added to clarify the enforcement of UNO Flip's special rule logic.

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 (M7.2.3).
- Handling user inputs from hardware devices and abstracting them into standard data formats (M7.1).
- Displaying the game state through a user interface (M7.5.2).
- Ensuring compatibility with hardware platforms via a hardware-hiding module (M7.1).

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 (M7.2, M7.5).
- Implementing special card effects, such as "Flip," "Skip," and "Draw Two," to update the game state accordingly via the M7.2.2 module and enforced by M7.2.3 and M7.2.
- Card effect validation ensures illegal plays are blocked, and effects are triggered using internal enums and state handlers.

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 (M**??**).
- Recording and updating player scores dynamically (M**??**).
- 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 (M7.5.2).
- Managing state transitions and ensuring synchronization between modules (M??).
- 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 (M7.4.3).
- Ensuring special card effects are visually represented and correctly executed by coupling card effect logic (M7.2.2) with animation triggers (M7.4.3).
- Animations are context-aware — for example, the "Flip" animation is activated when a Flip card is played, and player hands are updated accordingly.

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 (M??, M7.5.2, M??).

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 (M7.5.2, M??).
- 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 (M7.4.3, M??).
- 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 (M7.4.2).
- 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 (M7.5.2).
- Ensuring game outputs are consistent with the effects of special cards (M7.2.2).

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 (M**??**).
- Managing game control logic for multiplayer and advanced game modes (M**??**, M**??**).

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. If the entry is "UNO Flip Remix", it means the module will be implemented by the UNO Flip Remix 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: UNO Flip Remix

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: UNO Flip Remix

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: UNO Flip Remix

Type of Module: ~~Record~~ Abstract Object

Secrets: The rules and logic for gameplay mechanics, including valid moves and player actions.

Services: Manages turn resolution, validates moves, updates the game state, and communicates changes to other modules.

Implemented By: UNO Flip Remix

Type of Module: Abstract Object

Secrets: The logic for calculating scores and determining game winners.

Services: Tracks each player's points, updates scores based on remaining cards, and triggers win conditions.

Implemented By: UNO Flip Remix

Type of Module: Abstract Object

Secrets: The conditions for rule legality and move validation.

Services: Prevents illegal moves, checks that actions conform to game rules, and triggers alerts or skips turns as needed.

Implemented By: UNO Flip Remix

Type of Module: Abstract Object

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: UNO Flip Remix

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: UNO Flip Remix

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: UNO Flip Remix

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: UNO Flip Remix

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: UNO Flip Remix

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: UNO Flip Remix

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: UNO Flip Remix

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: UNO Flip Remix

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.

Requirement (Req.)	**Modules**
R1	Hardware-Hiding Module, Input Module, Turn Management Module, User Interface Module
R2	Input Module, Card Effect Module
R3	Verification Module, Verification Output Module, Score Tracking Module
R4	Output Module, Card Effect Module, Control Module, User Interface Module
R5	Card Effect Module, Turn Management Module, Control Module, Animation Module
R6	Output Module, Card Effect Module, Control Module, Turn Management Module, Game Logic Module
R7	User Interface Module, Score Tracking Module, Control Module
R8	User Interface Module, Animation Module, Control Module
R9	Verification Output Module, Save/Load Module
R10	Output Module, Card Effect Module, Control Module
R11	Output Module, Card Effect Module, Score Tracking Module, Control Module

Table 3: Trace Between Requirements and Modules

Anticipated Change (AC)	**Modules**
Hardware Change	Hardware-Hiding Module
Input Change	Input Module
Turn Management	Turn Management Module
UI Update	User Interface Module
Card Effect Update	Card Effect Module
Verification Update	Verification Module, Verification Output Module
Output Update	Output Module
Animation Update	Animation Module
Save/Load Update	Save/Load Module
Control Logic	Control Module, Game Logic Module
Score Tracking	Score Tracking Module

Table 4: 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.

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 an import statement.

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

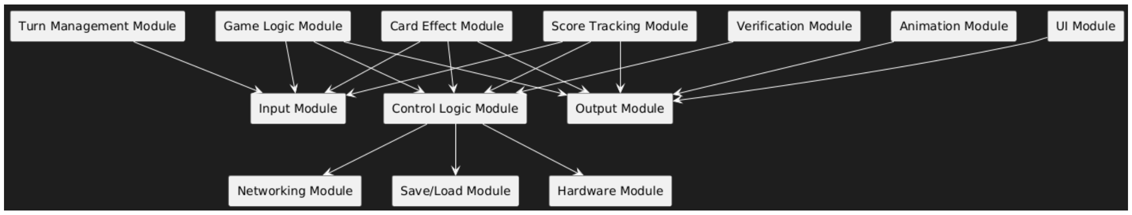

Figure 1: Use hierarchy among modules

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.

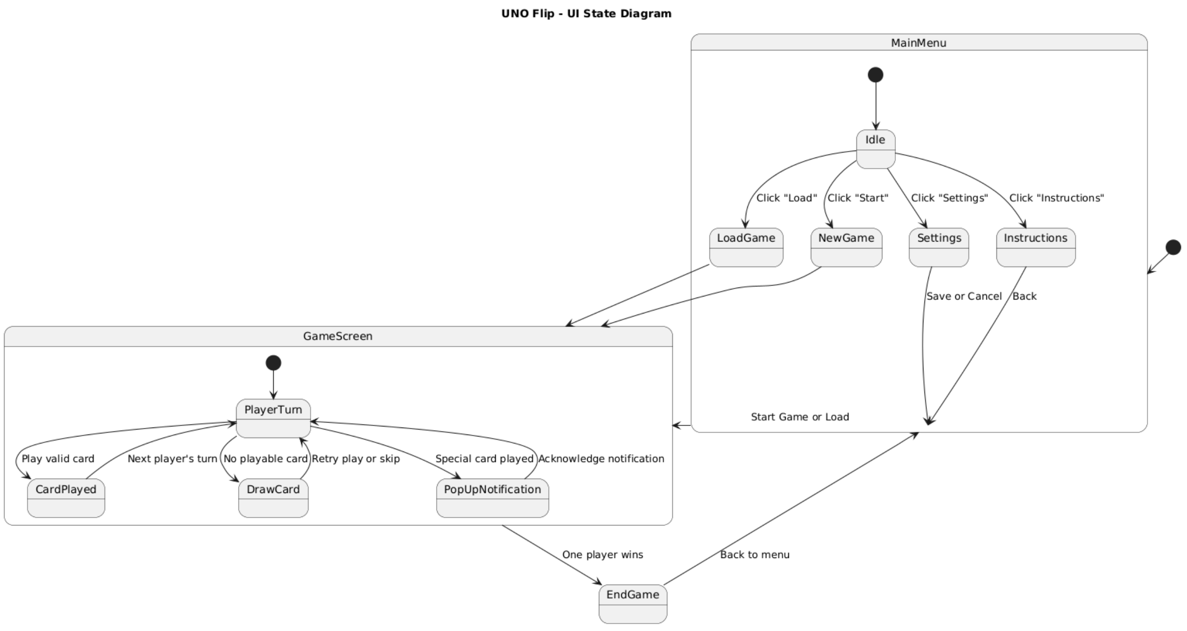

Figure 2: UI State Diagram showing menu navigation and in-game transitions

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 a custom TCP socket-based server to facilitate real-time communication between Unity clients. TCP ensures reliable packet delivery and strict message ordering.
- **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).
- Each message is parsed by the Multiplayer Networking Module, which ensures it adheres to the defined protocol and updates the game state accordingly.

11.2 Error Handling and Recovery

- **Connection Loss:** The server supports reconnection attempts from clients. A basic handshake is re-established to restore session state with minimal disruption.
- **Invalid Messages:** The server validates incoming messages to ensure compliance with the expected structure. Invalid or malformed data is logged and rejected to maintain state consistency.
- Keep-alive messages (heartbeats) may be used to monitor client activity and prevent dropped connections from going unnoticed.

11.3 Security

- **Encryption:** This implementation does not currently use TLS or any form of encrypted messaging. All traffic between clients and the TCP server is unencrypted, which is acceptable for local testing or internal usage.
- **Authentication:** No authentication system is in place. Players connect to the server via raw TCP, and once connected, they are prompted to enter a player name before beginning matchmaking.
- **Session Control:** The server tracks connected clients and matches them in pairs to initiate a game. Game sessions are maintained server-side until completion.

12 Timeline

- **January 20 – January 25: Project Planning and Requirements Gathering**
  Responsible: Jianhao Wei
- **January 26 – February 5: Game Core Development (Deck, Turn, Card Effect Systems)**
  Responsible: Mingyang Xu, Zheng Bang Liang
- **February 6 – February 12: UI and Game State Integration**
  Responsible: Zain-Alabedeen Garada, Kevin Ishak
- **February 13 – February 20: Pre-Rev 0 Testing and Game Logic Validation**
  Responsible: Jianhao Wei, Kevin Ishak
  This phase included unit testing for turn logic and card interactions. A playable offline version was finalized.
- **February 21 – February 28: Rev 0 Demo and Internal Feedback Cycle**
  Responsible: All Members
  We demonstrated offline gameplay, gathered TA feedback, and finalized design documents.
- **February 29 – March 8: Multiplayer Functionality and Networking Implementation**
  Responsible: Mingyang Xu, Zain-Alabedeen Garada
  Client-server syncing, player connection, and turn-by-turn multiplayer flow were introduced post-Rev 0.
- **March 9 – March 16: Rev 1 Finalization and Advanced Features**
  Responsible: Kevin Ishak, Zain-Alabedeen Garada
  We completed AI logic, multiplayer improvements, and bug fixes for Rev 1 submission.
- **March 17 – March 24: Full Game Completion and Final Documentation**
  Responsible: All Members
  Final implementation included card animations, game-end flow, and all UNO Flip rules. We completed all major documents: MG, MIS, SRS, VnV Plan, VnV Report, Hazard Analysis.
- **March 25 – March 30: EXPO Preparation and Presentation Rehearsal**
  Responsible: All Members
  Prepared pitch-style final demo, poster design, and rehearsed Q&A.

References

1. D. L. Parnas, ”On the Criteria to Be Used in Decomposing Systems into Modules,” *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

2. D. L. Parnas, ”Designing Software for Ease of Extension and Contraction,” *IEEE Transactions on Software Engineering*, vol. SE-5, no. 2, pp. 128–138, 1978.

3. C. Richardson, ”Decompose by Subdomain,” *Microservices.io*, 2023.

4. Stack Overflow, ”Abstraction vs Information Hiding vs Encapsulation,” *Stack Overflow*, 2023.

5. M. Xu, ”UNO Flip 3D - SRS Volere Documentation,” *GitHub Repository*, 2023.

6. M. Xu, ”UNO Flip 3D - Software Architecture Document,” *GitHub Repository*, 2023.

7. Mattel Games, ”UNO Flip! Game Rules,” [Online]. Available: https://www.unorules.org/uno-flip-rules/. Accessed: Mar. 31, 2025.

8. IETF, ”The Transport Layer Security (TLS) Protocol Version 1.3,” RFC 8446, Aug. 2018. [Online]. Available: https://tools.ietf.org/html/rfc8446