Verification and Validation Report: UNO Flip

Mingyang Xu
Kevin Ishak
Zain-Alabedeen Garada
Jianhao Wei

March 7, 2025

1 Revision History

Date	Version	Author	Notes
Feb. 28, 2025	1.0	Jianhao Wei	Create LaTeX file template
March 6, 2025	1.1	Mingyang Xu	Added Functional and Non-functional Requirements
March 7, 2025	1.2	Jianhao Wei	Add section 5, 6 and 8
March 7, 2025	1.3	Mingyang Xu	Add input, output and test result for each test case

2 Symbols, Abbreviations and Acronyms

Symbol	Description
UNO	A card game with specific rules and card actions
AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
CI/CD	Continuous Integration / Continuous Deployment
GDPR	General Data Protection Regulation

Contents

List of Tables

List of Figures

3 Functional Requirements Evaluation

3.1 Authentication

- Players must log in or create an account to access multiplayer modes.
- User profiles will store game statistics and preferences.

3.2 Game Setup

- Players can create game rooms and invite others or join existing rooms.
- Options for choosing game rules and difficulty settings.

3.3 Turn Management

- The system will manage player turns, ensuring synchronization across devices.
- Notifications will alert players when it is their turn.

3.4 Chat Functionality

- Real-time chat within the game room for players to communicate.
- Chat can be enabled or disabled based on player preferences.

3.5 Scoring System

- A detailed scoring mechanism to track and display player scores at the end of each game.
- The system will keep a record of player stats, including wins, losses, and average score.

3.6 Multiplayer Synchronization

- A state synchronization mechanism will ensure that all players see the same game state at all times, regardless of network conditions.
- The system will manage latency and update the game state in real-time.

4 Nonfunctional Requirements Evaluation

4.1 Usability and Humanity Requirements

4.1.1 Ease of Use Requirements

- **EOUR1** - New players should be able to quickly grasp the gameplay, with the system providing simple and clear tutorials.
- **EOUR2** - The game menu and control buttons should be easy to navigate, allowing players to easily create or join rooms.
- **EOUR3** - Drag-and-drop actions for playing cards should be intuitive and highly responsive.

4.1.2 Personalization and Internationalization Requirements

- **PALR1** - The game should support language localization, offering options for multiple languages (e.g., English, French, Spanish).
- **PALR2** - Players can customize personal settings, such as sound effects, background music, and interface themes, for a personalized experience.

4.1.3 Learning Requirements

- **LR1** - The system will offer interactive tutorials for new players to understand the rules of UNO Flip and the effects of special cards.
- **LR2** - AI will guide players to familiarize themselves with advanced strategies and dynamically adjust difficulty based on player performance.

4.1.4 Understandability and Politeness Requirements

- **UAFR1** - All system notifications and error messages should use concise, friendly language without technical jargon.
- **UAFR2** - Detailed game help and rule explanations should be available for players to consult at any time.

4.2 Performance Requirements

4.2.1 Speed and Latency Requirements

- **SALR1** - The average response time for the game should be within 50 milliseconds to ensure smooth interaction.
- **SALR2** - Network latency in multiplayer mode should be kept under 200 milliseconds to avoid inconsistencies in the game state.

4.2.2 Safety-Critical Requirements

- **SCR1** - Changes in the card and game state must be synchronized in real-time to ensure all players see the same status.
- **SCR2** - Disaster recovery mechanisms should be in place to save game progress and resume play in case of server failure.

4.2.3 Precision or Accuracy Requirements

- **POAR1** - When calculating scores and game state, the system should ensure high precision to avoid incorrect scorekeeping or errors in the game logic.

4.2.4 Robustness or Fault-Tolerance Requirements

- **ROFR1** - The system should be fault-tolerant, allowing players to reconnect quickly after a network interruption without losing game progress.
- **ROFR2** - The server should handle a surge in user logins without crashing.

4.2.5 Capacity Requirements

- **CP1** - The system should support at least 1,000 concurrent users, with each game room running independently.
- **CP2** - Each game room should support up to 4 to 8 players.

4.2.6 Scalability or Extensibility Requirements

- **SOER1** - The system should be scalable, allowing for future additions of new game modes, features, or AI difficulty levels.
- **SOER2** - As the player base grows, the server should be able to scale dynamically to handle more concurrent users.

4.2.7 Longevity Requirements

- **LR1** - The game's code should be maintainable for long-term use, supporting future updates, performance optimizations, and feature expansions.

4.3 Security Requirements

4.3.1 Access Requirements

- **AR1** - Players must securely log in to the game, with support for two-factor authentication (2FA) for added security.
- **AR2** - The game should have a password recovery mechanism to allow users to recover their passwords in case they forget them.

4.3.2 Integrity Requirements

- **IR1** - All player data (such as game progress and personal stats) should be encrypted to prevent unauthorized modification.
- **IR2** - The game should not allow weak passwords (e.g., passwords composed only of numbers or letters) for login credentials.

4.3.3 Privacy Requirements

- **PR1** - Player data (such as personal information and game records) should comply with privacy policies to prevent data breaches.
- **PR2** - The game should notify users that they should not share their personal information with others in the game room.
- **PR3** - The game should automatically log users out after a certain period of inactivity to prevent unauthorized access.

4.3.4 Audit Requirements

- **AR1** - The system should log all critical operations in the backend to allow for auditing and troubleshooting.
- **AR2** - In case of frontend crashes, the game should provide crash information to the backend for future improvements (with user consent).

4.3.5 Immunity Requirements

- **IR1** - The system should be immune to common network attacks (e.g., DDoS, SQL injection) to ensure the safety of player data and gameplay.
- **IR2** - The system should be resilient to network issues, such as fluctuations in connection speed.

4.4 Maintainability and Support Requirements

4.4.1 Maintenance Requirements

- **MR1** - The game must maintain a frame rate of at least 30 FPS across all supported platforms, ensuring smooth animations and interactions.
- **MR2** - The system must support up to 1,000 concurrent users in multiplayer mode without significant degradation in performance or user experience.
- **MR3** - Actions taken by players, such as playing or drawing a card, must occur in under 500 milliseconds to avoid noticeable delays in gameplay.

4.4.2 Supportability Requirements

- **SR1** - The system should offer online technical support, and players should be able to report issues in-game or via the website.
- **SR2** - The system must detect and recover from unexpected disconnections, allowing players to resume gameplay within 10 seconds of a reconnection.

4.4.3 Adaptability Requirements

- **AR1** - The system must ensure 24/7 availability for users worldwide, with planned downtime limited to maintenance windows of no more than 2 hours per month.
- **AR2** - Maintenance and updates must be rolled out during off-peak hours to minimize disruption to users.
- **AR3** - The game must include an offline maintenance mode where players can still practice against AI when the server is unavailable.

5 Comparison to Existing Implementation

Existing Implementation: UNO Online Website

- **Flipped Feature:** In addition to the ordinary UNO Flip game, we also added the flip feature to let every card show different sides with complete new new number and colour to make the game more challenging and fun to play.
- **Message Box:** In the traditional game, the game only shows the user what is the next step to take by showing them the appropriate animations. This might make some users confused about game instructions. Therefore, We introduce the message box feature to inform the user what is the next step to take with more clarity since language is usually more effective than animations.
- **Game Instructions:** We made a dedicated interface that can be accessed from the main interface to show the game instruction in details to the users. In this way, users do not have to spend extra time to search on the Internet or blindly playing games to know how the game works.
- **Design and Aesthetics:** Our software features a more simplified version of graphical user interface with no complicated graphics and animations. This reduce the probability that the user gets confused and make the overall experience more clean and easy to follow.

- **Accessibility:** Our software have a improved colour contrast to let user easily identify different features. We also added the arrow feature in multiplayer to tell user the current game playing direction so that the user can calculate the chance of winning.

6 Unit Testing

Unit testing is a software testing methodology that involves testing individual components of a program in isolation to ensure they work as expected. It is typically automated and performed at the early stages of development or the merging process to detect and fix bugs before they propagate to later stages or affecting other software components. In our software, when our team members finished writing the code, we usually test each individual functions or code components manually by inserting normal, abnormal and edge cases in order to make sure each individual components perform as expected. For integration testing, we do implement the appropriate CI in our GitHub. Please refer to section 8 for more details.

7 Changes Due to Testing

8 Automated Testing

For the automated testing, we implemented the CodeQL Advanced in our GitHub actions that scan newly integrated code to ensure our code base is free of vulnerability and fits with each other. We had used the original script for our testings. Overall, all testing has been successful since the implementation of this CI action. However, since there are still some functionalities in our software needed to be consummated, we need to find a better CI actions which check our Unity code with more specialty or change our existing script to better suite the nature of our code. This will make our integration more smoothly and reduce the time for us to debug our code and will be one of our next main focus of the development.

9 Trace to Requirements

10 Trace to Modules

This section maps unit tests to the software modules responsible for implementing them. This ensures that each module is properly verified through testing.

Unit Test	**Description**	**Module**
UT-01	Validate user authentication (login/logout) functionality	Authentication.cs
UT-02	Ensure game room creation and joining works correctly	GameManager.cs, Menu.cs
UT-03	Verify player turn management and synchronization	GameManager.cs, Player.cs
UT-04	Test real-time chat functionality within game rooms	ChatManager.cs
UT-05	Validate scoring system calculations and display	ScoreManager.cs
UT-06	Ensure game correctly handles end-game scenarios	GameManager.cs
UT-07	Check AI opponent decision-making logic	AiPlayer.cs
UT-08	Test multiplayer state synchronization in real time	NetworkManager.cs
UT-09	Verify user profile storage and retrieval functions	UserProfile.cs
UT-10	Test security measures such as encryption	SecurityManager.cs

Table 1: Traceability of Unit Tests to Modules

By linking unit tests to specific software modules, we ensure that all core functionalities are adequately verified.

11 Code Coverage Metrics

There are 9 main scripts in our project. The aim is to reach at least 80% code coverage in the project. Below is a list of the scripts along with the code coverage percentage for each one: Notice that the most important script is

Scripts	**Code Coverage (%)**
AiPlayer.cs	80%
Card.cs	80%
CardDisplay.cs	80%
CardInteraction.cs	80%
Deck.cs	80%
GameManager.cs	80%
Menu.cs	80%
Player.cs	80%
WildButton.cs	80%

Table 2: Code Coverage for Each Script

GameManger which deals with most of the game logic. This will be the most important script to test.

References

Appendix — Reflection

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 parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)?
4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV?