System Verification and Validation Plan for Uno-Flip Remix

Team 24
Mingyang Xu
Kevin Ishak
Zain-Alabedeen Garada
Jianhao Wei
~~Andy Liang~~

April 2, 2025

Contents

List of Tables

Revision History

Date	Version	Notes
2024-10-29	1.0	Created shared file
2024-10-30	1.1	Wrote Sections 1–2
2024-10-31	1.2	Wrote Section 3 and modified Sections 1 and 2
2024-11-01	1.3	Added Section 4 and Reflection
2025-03-26	2.0	Kevin Ishak - Revised document to align with course standards. Improved formatting, added references to Hazard Analysis, removed JavaScript/Java tool mentions, and clarified ambiguities.

1 Symbols, Abbreviations, and Acronyms

Table 2: Symbols, Abbreviations, and Acronyms

Term	**Definition**
SRS	Software Requirements Specification
PSAG	Problem Statement and Goals
V&V	Verification and Validation
FR	Functional Requirement
NFR	Non-Functional Requirement
UI	User Interface
AI	Artificial Intelligence (used in opponent behavior)
SUT	System Under Test
ID	Identifier used to label specific test cases
Rev 0	Initial version of the game build used for early demonstration
Unity	Game engine used to implement UNO Flip Remix

2 General Information

2.1 Summary

The software being tested is an UNO Flip game application, developed to provide an engaging, digital version of the popular card game with additional features for enhanced user experience. The game includes functionalities such as switching between two sides of the cards (light and dark), maintaining score, tracking player moves, and handling various card effects. This software aims to capture the essence of the physical game while offering interactive elements that make it enjoyable on a digital platform.

2.2 Objectives

The primary objective of the Verification and Validation (V&V) plan is to build confidence in the correctness and stability of the game software, ensuring a smooth user experience with minimal bugs. Key objectives include:

- **Verifying functionality**: Testing to ensure that core game mechanics, such as card flipping, turn-taking, and scoring, work as expected.
- **Ensuring performance**: Assessing the software's responsiveness and ensuring minimal lag or delay in gameplay.
- **Evaluating user experience**: Conducting usability tests to confirm that the interface is intuitive and enhances engagement.

Out of scope:

- **Accessibility testing**: Due to limited resources, we are not performing detailed accessibility testing. Our focus is on verifying core gameplay features.
- **Advanced AI opponent testing**: Since the game primarily targets player-vs-player mode, sophisticated AI functionality is not within our current scope.

2.3 Challenge Level and Extras

The challenge level for this project is classified as **general**, as agreed with the course instructor. This level reflects the game's moderate complexity, focusing on implementing and testing the main gameplay mechanics without advanced AI or complex networking features.

The two extras selected for grading are:

- **Usability Testing**: Ensures that the game interface is intuitive and accessible to users with varying levels of gaming experience. This is especially relevant in a multiplayer setting, where ease of interaction directly affects player enjoyment.
- **GenderMag Analysis**: Used during the design phase to evaluate how different cognitive styles (especially across gender dimensions) interact with the interface. This helps reduce potential biases in design and increases inclusivity, aligning well with our project's user-centric goals.

2.4 Relevant Documentation

- **SRS**: This document outlines the game's functional and non-functional requirements, providing the foundation for the test cases designed in this V&V plan. Available at: `https://github.com/simon-0215/UNO-Flip-3D/blob/main/docs/SRS-Volere/SRS.pdf`
- **PSAG**: Defines our problem statement and goals. Available at: `https://github.com/simon-0215/UNO-Flip-3D/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf`

3 Plan

This section provides an overview of the planned verification and validation (V&V) activities for our project. The plan outlines the roles and responsibilities of the team members, strategies for verifying the SRS, design, and implementation, as well as the testing approaches for both functional and non-functional requirements. Additionally, this plan traces back to hazards identified in the Hazard Analysis document to ensure critical risks are verified and mitigated during testing. By following this plan, we aim to ensure that our software meets all specified requirements and maintains high reliability, usability, and performance standards.

3.1 Verification and Validation Team

Team Member	Role	Responsibilities
Mingyang Xu	Lead Validator	Oversees the V&V process, ensures adherence to the plan, and coordinates team efforts.
Jianhao Wei	SRS and Design Reviewer	Reviews the SRS and design documents, providing feedback on requirement clarity and feasibility.
~~Zheng Bang Liang~~ Kevin Ishak	Code Verification Specialist	Implements unit and integration tests using C# and Unity Test Framework; conducts code inspections and static analysis.
Zain-Alabedeen Garada	System Test Engineer	Designs and executes system tests, covering functional and non-functional requirements, and reports outcomes.
Kevin Ishak	System Test Engineer	Assists in designing and executing system tests, and reporting on test coverage and outcomes.

Table 3: Team member responsibilities

3.2 SRS Verification Plan

1. **Review Process**: Formal peer review led by Jianhao Wei with supervisor support. Evaluates clarity, completeness, and feasibility.
2. **Meeting with Supervisor**: Review SRS in detail; clarify requirement intent and gather improvement suggestions.
3. **SRS Checklist**: Ensure requirements meet SMART criteria and are traceable to tests.
4. **Use of Issue Tracker**: Log all issues on GitHub for transparency and revision tracking.
5. **Task-Based Inspection**: Each team member validates a section aligned with their test domain.
6. **Ad Hoc Feedback**: Peer review by classmates experienced in game or Unity projects.

3.3 Design Verification Plan

1. **Peer Reviews**: Examine consistency, modularity, feasibility of Unity architecture.
2. **Checklist-Based Verification**: Ensure modularity, maintainability, Unity best practices.

3. **Review Meeting with Supervisor**: Present high-level Unity scene and component layout.
4. **Issue Tracker**: Track issues/improvements via GitHub.

3.4 V&V Plan Verification

1. **Peer Review and Checklist**: Review clarity and coverage.
2. **Mutation Testing**: Validate the strength of test cases using Unity-based mutation strategies.
3. **Classmate Feedback**: Additional external feedback for clarity.
4. **Supervisor Review**: Confirm thoroughness and feasibility.
5. **Checklist Creation**: Verify test coverage and traceability.

3.5 Implementation Verification Plan

1. **Unit Testing**: Use NUnit with Unity Test Framework to validate scripts.
2. **Static Code Analysis**: Use Visual Studio's built-in static code analysis.
3. **Code Walkthroughs and Inspections**: Regular code reviews to catch logical errors.
4. **Final Class Presentation**: Showcase core test cases and inspection walkthroughs.

3.6 Automated Testing and Verification Tools

- **NUnit with Unity Test Framework**: Used to write and automate unit and integration tests for C# scripts in Unity.
- **Unity Profiler**: Monitors frame rate, memory usage, and performance bottlenecks.
- **Code Coverage Tools**: Use Unity's Code Coverage package to identify untested parts of the codebase.
- **GitHub Actions for CI**: Runs tests on pull requests to catch regressions.
- **Visual Studio Analyzer**: Performs static checks for style and bug patterns.
- ~~JavaScript and Jest tools have been removed to reflect Unity/C# usage.~~

3.7 Software Validation Plan

1. **User Review Sessions**: Gather informal feedback from players during playtests.
2. **Rev 0 Demo for Supervisor**: Walkthrough of gameplay and key functionality.
3. **User Testing**: Validate gameplay flow, difficulty balance, and satisfaction.
4. **Task-Based Inspection**: Match user stories to requirement fulfillment.
5. **Reference to Hazard Analysis**: Validation activities will be cross-referenced with the Hazard Analysis document to ensure that identified risks are adequately mitigated. This helps confirm that safety and reliability concerns are addressed during the testing process.

4 System Tests

This section outlines the testing procedures designed to verify that the system meets its functional and non-functional requirements. The tests will cover various areas to ensure comprehensive validation of the system's functionality, performance, and usability. Each area will address specific requirements and reference the Software Requirements Specification (SRS) for detailed test criteria.

This section has been revised to address TA feedback by ensuring consistent formatting across all test cases and clarifying ambiguous test elements such as "Input" vs. "State." Additionally, each test case now includes detailed, step-by-step procedures to eliminate surface-level descriptions and improve traceability.

4.1 Tests for Functional Requirements

4.1.1 Area of Testing: Game Initialization

These tests ensure that the game session begins correctly when players enter the lobby and start a new game. The tests verify that player names are correctly registered, the lobby displays connected players, and cards are dealt when the session starts.

1. **Test ID: GI1-Test01**
   - **Control:** Manual
   - **State:** Application has been launched.
   - **Input:** User selects "Start Game" and enters a valid player name.
   - **Output:** Player enters lobby with their name displayed on screen.
   - **Test Case Derivation:** Verifies player name selection and lobby registration.
   - **Test Steps:**
     (a) Launch the UNO Flip Remix application.
     (b) Click "Start Game."

(c) Enter a player name (e.g., "Player1").

(d) Observe the lobby screen.

- **Validation:** Confirm that the entered name is displayed and the player is visible in the lobby.

2. **Test ID: GI1-Test02**

- **Control:** Automatic
- **State:** Two players are connected to the session.
- **Input:** N/A (automatic transition)
- **Output:** Game screen is displayed and cards are dealt to the players.
- **Test Case Derivation:** Verifies automatic game session start upon establishing a multiplayer connection.
- **Test Steps:**

  (a) Player 1 launches the game and enters a name.

  (b) Player 2 launches the game and enters a name.

  (c) Wait for the multiplayer session to establish.

  (d) Observe transition to game screen.

- **Validation:** Confirm that both players are transitioned into a game session and each receives a hand of 7 cards.

4.1.2 Area of Testing: Game Setup

The Game Setup area is essential for validating the creation and configuration of game rooms, as it directly impacts the player experience. Tests in this area verify that game rooms can be created with the specified rules, difficulty levels, and invite options. Reference to the SRS is made to ensure alignment with the functional requirements related to room creation, rule selection, and multiplayer setup.

1. **Test ID: GSR1-Test01**

- **Control:** Manual
- **State:** Application is launched and main screen is visible.
- **Input:** User clicks the "Play Game" button.
- **Output:** A new game room is created, and the player can see options to invite others.
- **Test Case Derivation:** Verifies that the game correctly initiates a room creation process from the main menu.
- **Test Steps:**
  - Launch the UNO Flip Remix application.
  - Wait for the main screen to load.

- Click the “Match Opponent” button.
- Observe the resulting interface.

- **Validation:** Confirm that a new room is instantiated, and the player can press the ”Play Game” button to begin the game.

4.1.3 Area of Testing: Turn Management

The Turn Management tests verify the correct sequencing of player turns and handling of turn-based actions such as the effects of special cards. This is crucial for ensuring fair gameplay and adherence to game rules as specified in the SRS.

1. **Test ID: TMR1-Test01**

   - **Control:** Automatic
   - **State:** Multiplayer game is in progress with two or more players.
   - **Input:** A player completes their turn.
   - **Output:** The next player’s turn begins and is reflected across all devices.
   - **Test Case Derivation:** Ensures real-time synchronization of turn order across connected clients.
   - **Test Steps:**
     - Start a multiplayer session with at least two players.
     - Player 1 plays a card and ends their turn.
     - Observe Player 2’s device.
   - **Validation:** Confirm that Player 2 sees it is now their turn and can proceed with gameplay.

2. **Test ID: TMR1-Test02**

   - **Control:** Automatic
   - **State:** Player A has played a “Skip” card.
   - **Input:** System recognizes the Skip card effect.
   - **Output:** Player B’s turn is skipped; turn moves to Player C.
   - **Test Case Derivation:** Validates Skip card functionality and impact on turn sequencing.
   - **Test Steps:**
     - Player A plays a Skip card.
     - Observe whether Player B is bypassed and Player C becomes active.
   - **Validation:** Confirm that Player B’s turn is skipped and the turn indicator moves to Player C.

3. **Test ID: TMR1-Test03**

- **Control:** Automatic
- **State:** Player A has played a “Reverse” card.
- **Input:** System processes the Reverse card action.
- **Output:** Turn order changes direction (e.g., clockwise to counter-clockwise).
- **Test Case Derivation:** Verifies correct change of turn order when a Reverse card is used.
- **Test Steps:**
  - Start a game with at least three players.
  - Player A plays a Reverse card.
  - Observe the direction of turns before and after the card is played.
- **Validation:** Confirm that subsequent turns proceed in the opposite direction from before the Reverse card was played.

4.1.4 Area of Testing: Card Effects (4.1.4)

The Card Effects tests verify that the special effects of various cards (such as Draw, Skip, and Reverse) work as expected within the game rules. These tests ensure that each card's effect is correctly applied to the game state and impacts player actions as intended.

1. **Test ID: CE1-Test01**
   - **Control:** Automatic
   - **State:** Game is in progress with at least two players.
   - **Input:** Player 1 plays a "Draw Two" card.
   - **Output:** Player 2 is required to draw two cards; turn proceeds to the next player.
   - **Test Case Derivation:** Verifies correct application of the "Draw Two" card effect.
   - **Test Steps:**
     - Player 1 plays a "Draw Two" card.
     - Observe Player 2's hand and turn state.
   - **Validation:** Confirm that Player 2 draws two cards and the game advances to the next player's turn.
2. **Test ID: CE1-Test02**
   - **Control:** Automatic
   - **State:** Game is in progress with multiple players.
   - **Input:** Player 1 plays a "Draw Four" Wild card and selects a new color.
   - **Output:** Player 2 draws four cards, and the play color changes to the selected color.

- **Test Case Derivation:** Verifies the "Draw Four" effect and color selection behavior.
- **Test Steps:**
  - Player 1 plays a "Draw Four" Wild card.
  - Player 1 selects a new color.
  - Observe Player 2's hand and game state.
- **Validation:** Confirm that Player 2 draws four cards and the play color is updated correctly.

3. **Test ID: CE1-Test03**
   - **Control:** Automatic
   - **State:** Game is in progress.
   - **Input:** Player plays a "Wild" card and selects a new color.
   - **Output:** Play color changes to the one selected.
   - **Test Case Derivation:** Verifies that the "Wild" card allows color selection.
   - **Test Steps:**
     - Player plays a "Wild" card.
     - Player selects a new color.
     - Observe the updated play color.
   - **Validation:** Confirm that the new color is applied and the game proceeds with that color.

4.1.5 Area of Testing: Game Over Conditions (4.1.5)

The Game Over Condition tests verify that the game correctly identifies when a player has won or when the game ends under specific conditions. These tests ensure proper detection of a winning condition and proper handling of game termination.

1. **Test ID: GOC1-Test01**
   - **Control:** Automatic
   - **State:** Game is in progress with at least two players. Player 1 has only one card left.
   - **Input:** Player 1 presses the UNO button and plays their final card.
   - **Output:** The game identifies Player 1 as the winner and ends the session.
   - **Test Case Derivation:** Verifies that the system enforces correct win conditions, including pressing the UNO button before the last card is played.
   - **Test Steps:**
     - Player 1 reduces their hand to one card.

  - Player 1 presses the on-screen UNO button.
  - Player 1 plays their final card.
  - Observe whether the win condition is triggered.
- **Validation:** Confirm that the game ends, displays a victory screen, and announces Player 1 as the winner. If the UNO button is not pressed before playing the final card, verify that the player is prevented from winning.

2. **Test ID: GOC1-Test02**

- **Control:** Automatic
- **State:** Multiplayer game is active with more than one player.
- **Input:** All players except one disconnect or exit the game.
- **Output:** The remaining player is automatically declared the winner.
- **Test Case Derivation:** Verifies that the game handles unexpected player exits by properly ending the session.
- **Test Steps:**
  - Start a multiplayer game with 3 or more players.
  - Have all players except one exit or disconnect.
  - Observe the game's behavior.
- **Validation:** Confirm that the game ends automatically and the remaining player is shown a winning message.

4.1.6 Area of Testing: Scoring System (4.1.6)

The Scoring System tests ensure that points are calculated and awarded correctly based on the cards remaining in each player's hand when the game ends. This area includes tests for verifying score calculation accuracy and ranking players based on their scores.

1. **Test ID: SS1-Test01**

- **Control:** Automatic
- **Initial State:** Game has just ended.
- **Input:** The system calculates the final scores based on remaining cards.
- **Output:** Players are ranked according to their scores, with points accurately reflecting card values.
- **Test Case Derivation:** Ensures that the scoring system accurately calculates points and ranks players accordingly.
- **Test Steps:**
  (a) End a game with various cards remaining in each player's hand.
  (b) Verify that the system calculates points for each player based on the values of the remaining cards.

(c) Confirm that players are ranked in descending order of scores.

- **Validation:** Confirm that the scoring is accurate and players are ranked correctly based on their scores.

4.1.7 Area of Testing: Multiplayer Synchronization (4.1.7)

The Multiplayer Synchronization tests verify that all players in a game room experience real-time updates and that actions taken by one player are correctly reflected for all other players. This is critical for ensuring a consistent multiplayer experience.

1. **Test ID: MS1-Test01**
   - **Control:** Automatic
   - **State:** Multiplayer game session is active with at least two players connected.
   - **Input:** Player 1 plays a valid card.
   - **Output:** The card is displayed immediately on all players' screens.
   - **Test Case Derivation:** Ensures that actions performed by one player are reflected on all clients in real time.
   - **Test Steps:**
     (a) Player 1 plays a green "3" card.
     (b) Observe other players' screens to see if the green "3" card is on top of the discard pile.
   - **Validation:** Confirm that the card appears simultaneously across all connected devices with minimal delay.
2. **Test ID: MS1-Test02**
   - **Control:** Automatic
   - **State:** Multiplayer game session is active with at least two players connected.
   - **Input:** Player 1 disconnects from the session (intentionally or due to network loss).
   - **Output:** Remaining players receive a disconnection notice, and the game updates its state accordingly.
   - **Test Case Derivation:** Verifies proper handling and communication of player disconnection events.
   - **Test Steps:**
     (a) Start a multiplayer game with multiple players.
     (b) Disconnect Player 1.
     (c) Observe whether a notification appears for remaining players.
     (d) Confirm game state (pause, continue, or default behavior).
   - **Validation:** Ensure the game handles the disconnection without crashing and notifies all active players appropriately.

4.2 Tests for Nonfunctional Requirements (4.2)

The tests for nonfunctional requirements ensure that the game meets standards for performance, usability, and appearance as specified in the SRS. These tests do not focus on specific functional behavior but rather on the overall quality of the system's user experience and performance metrics.

4.2.1 Area of Testing: Appearance Requirements (4.2.1)

1. **Test ID: AR1-Test01**

   - **Control:** Manual
   - **State:** Game interface has loaded on a desktop or laptop device.
   - **Input:** User navigates through the application UI, including the main menu, lobby, and in-game screen.
   - **Output:** All UI elements are consistently styled, readable, and follow a unified visual design.
   - **Test Case Derivation:** Ensures the UI adheres to modern design standards and provides a visually cohesive experience.
   - **Test Steps:**
     (a) Launch the UNO Flip Remix application.
     (b) Navigate through the main menu, lobby, and in-game screen.
     (c) Observe layout consistency, font readability, button styling, and color themes.
   - **Validation:** Tester confirms that no UI elements are misaligned, cut off, or styled inconsistently. Visual hierarchy and contrast must align with the criteria defined in the SRS.

4.2.2 Area of Testing: Speed and Latency Requirements (4.2.2)

1. **Test ID: SALR1-Test01**

   - **Control:** Manual with timing tool
   - **State:** Game is active in multiplayer mode with at least two connected players.
   - **Input:** A player performs an action such as playing a card or drawing from the deck.
   - **Output:** The system reflects the action within 50 milliseconds on all connected players' screens.
   - **Test Case Derivation:** Verifies that in-game interactions produce minimal delay, ensuring responsive gameplay.
   - **Test Steps:**
     (a) Start a multiplayer session with two players.
     (b) Player 1 plays a card.

(c) Use a response time measurement tool to record the time taken for the action to appear on Player 2's screen.

- **Validation:** Tester confirms that the delay is less than 50ms and no lag or jitter occurs during card play interactions.

2. **Test ID: SALR1-Test02**

- **Control:** Automated network load simulation
- **State:** Game is running with the maximum number (4) of players supported.
- **Input:** Players perform simultaneous in-game actions under simulated high network traffic.
- **Output:** The system maintains less than 100ms response time despite the load.
- **Test Case Derivation:** Ensures robustness of the networking system under stress.
- **Test Steps:**
  (a) Simulate network congestion using a tool like Clumsy or a throttled Wi-Fi environment.
  (b) All players perform actions like playing cards or drawing cards.
  (c) Measure the delay experienced for action propagation between clients.
- **Validation:** Tester validates that actions are still reflected in under 100ms, with no game desync or player timeout.

4.2.3 Usability Requirements (4.2.3)

These tests confirm that the user interface is intuitive and easy to navigate, ensuring a positive experience for players.

1. **Test ID: UR1-Test01**

- **Control:** Manual
- **State:** User has launched the game and is viewing the main menu.
- **Input:** New user attempts to navigate the menu and start a game without external guidance.
- **Output:** User is able to start a game without assistance or confusion.
- **Test Case Derivation:** Ensures that the main menu and navigation are intuitive for first-time users.
- **Test Steps:**
  (a) Ask a new user to open the game application.
  (b) Observe the user as they attempt to reach the lobby or start screen.
  (c) Note if any instructions or assistance are required.

- **Validation:** Tester records any difficulty or confusion and confirms whether the user can begin a game without guidance.

2. **Test ID: UR1-Test02**
   - **Control:** Manual
   - **State:** User is in an active game session with cards in hand.
   - **Input:** User interacts with game elements such as drawing and playing cards, and using the pause menu.
   - **Output:** User performs actions smoothly, indicating ease of control interaction.
   - **Test Case Derivation:** Ensures that in-game interactions are accessible and understandable to all users.
   - **Test Steps:**
     (a) Observe the user as they draw and play cards during a live game.
     (b) Ask them to pause and resume the game.
     (c) Record their reaction and ease of completing these tasks.
   - **Validation:** Tester assesses if user actions are intuitive and whether any hesitation or confusion occurs.

4.2.4 Reliability Requirements (4.2.4)

These tests confirm the stability and reliability of the game, ensuring that it can handle prolonged usage without crashes or data loss.

1. **Test ID: RR1-Test01**
   - **Control:** Manual
   - **State:** Game has been continuously running for an extended session (e.g., 2 hours).
   - **Input:** Observe game performance and interactions during and after extended runtime.
   - **Output:** Game remains stable without crashes, freezes, or memory-related issues.
   - **Test Case Derivation:** Ensures the system maintains consistent behavior and performance over time.
   - **Test Steps:**
     (a) Launch the game and enter a session.
     (b) Allow the game to remain active for 2 hours with periodic interaction.
     (c) Attempt standard in-game actions such as drawing, playing, and flipping cards.
   - **Validation:** Tester confirms that no crashes, major performance drops, or data corruption occur during or after prolonged use.

4.2.5 Availability Requirements (4.2.5)

These tests verify that the game is available for access as required, particularly in multiplayer mode, ensuring minimal downtime.

1. **Test ID: AV1-Test01**
    - **Control:** Automated
    - **State:** Game server is deployed and running continuously over a 24-hour observation window.
    - **Input:** Automated script attempts to access the server at regular intervals (e.g., every 5 minutes).
    - **Output:** Game is accessible with no more than 5 minutes of downtime in a 24-hour period.
    - **Test Case Derivation:** Ensures the game meets availability targets defined in the SRS for minimal service interruption.
    - **Test Steps:**
        (a) Deploy the game server and monitor it over 24 hours.
        (b) Run an automated script that pings the server every 5 minutes.
        (c) Log and analyze any failed connection attempts.
    - **Validation:** Confirm that downtime does not exceed 5 minutes total during the 24-hour period.

4.2.6 Security Requirements (4.2.6)

These tests ensure that user data is secure and that unauthorized access is prevented.

1. **Test ID: SR1-Test01**
    - **Control:** Manual
    - **State:** User is in an active game session with a valid player name.
    - **Input:** Attempt to trigger unauthorized access to another player's screen or in-game data using in-game interactions or console commands.
    - **Output:** Access is denied, and no data is leaked or manipulated.
    - **Test Case Derivation:** Ensures that the multiplayer environment protects each player's game state and restricts unauthorized access.
    - **Test Steps:**
        (a) Launch two instances of the game and enter different player names.
        (b) Attempt to read or manipulate another player's card hand or turn data using unauthorized means.
    - **Validation:** Confirm that all unauthorized attempts are blocked and no external player data is accessible or visible.

4.2.7 Maintainability Requirements (4.2.7)

These tests verify that the code is maintainable and can be updated or debugged with ease.

1. **Test ID: MR1-Test01**

   - **Control:** Manual (Code Review)
   - **State:** Codebase is accessible in the team's GitHub repository with full source and documentation.
   - **Input:** Conduct a review focusing on modularity, naming conventions, documentation quality, and adherence to Unity C# best practices.
   - **Output:** Code is well-documented, modular, and follows consistent conventions, supporting ease of future maintenance and debugging.
   - **Test Case Derivation:** Ensures that future contributors or developers can easily understand, modify, and extend the codebase.
   - **Test Steps:**
     (a) Open the Unity project and inspect folder structure and scripts.
     (b) Verify that each script handles a single responsibility (e.g., PlayerManager.cs, CardEffectHandler.cs).
     (c) Review presence and clarity of inline comments and function headers.
     (d) Evaluate use of consistent naming, spacing, and formatting across files.
   - **Validation:** Confirm that code meets defined maintainability criteria with no major violations or undocumented sections.

4.2.8 Portability Requirements (4.2.8)

These tests ensure that the game operates correctly across different platforms and devices.

1. **Test ID: PR1-Test01**

   - **Control:** Manual
   - **State:** The final build is available for Windows and macOS platforms.
   - **Input:** Launch and play the game on both Windows and macOS to observe platform behavior.
   - **Output:** Game runs smoothly and consistently on all supported platforms without platform-specific issues.
   - **Test Case Derivation:** Ensures the game delivers a consistent experience across different operating systems, as outlined in the SRS portability requirement.
   - **Test Steps:**
     (a) Download and install the game on Windows and macOS devices.
     (b) Launch the game on each device.

(c) Play through a full match on each platform.

(d) Observe UI rendering, performance, and responsiveness.

- **Validation:** Confirm that gameplay, visuals, and performance are consistent and error-free on all tested platforms.

4.2.9 Scalability Requirements (4.2.9)

These tests confirm that the game can handle increasing numbers of players without performance degradation.

1. **Test ID: SCR1-Test01**

   - **Control:** Automated
   - **State:** The game server is running with a baseline load.
   - **Input:** Simulate increasing numbers of players joining and playing in real-time.
   - **Output:** Server maintains performance as player count increases up to the maximum supported level.
   - **Test Case Derivation:** Ensures the game maintains responsiveness and functionality as concurrent users increase, aligning with the scalability requirement in the SRS.
   - **Test Steps:**

     (a) Launch game server.

     (b) Use a simulation tool to incrementally add player sessions (e.g., from 2 to 20).

     (c) Monitor CPU usage, memory usage, and response time after each increment.
   - **Validation:** Confirm that the game handles the increased load without degradation in response time, game state synchronization, or connection stability.

4.2.10 Performance Requirements (4.2.10)

These tests assess the game's performance under typical and peak conditions, ensuring that response times remain within acceptable limits.

1. **Test ID: PRF1-Test01**

   - **Control:** Manual
   - **State:** Game is launched and in progress with standard gameplay conditions.
   - **Input:** Measure gameplay response time and frame rate over the course of a full match.
   - **Output:** Response time is below 100ms, and frame rate is consistent (at least 30 FPS).

- **Test Case Derivation:** Ensures that the game maintains performance under normal operating conditions, as specified in the SRS.
- **Test Steps:**
  (a) Start a multiplayer game session.
  (b) Perform normal actions (play cards, draw cards, flip sides).
  (c) Use a profiler or performance monitor to log response delays and FPS.
- **Validation:** Confirm that average input latency stays under 100ms and the frame rate remains stable during gameplay.

4.2.11 Accessibility Requirements (4.2.11)

These tests verify that the game is accessible to users with disabilities and adheres to relevant accessibility standards.

1. **Test ID: ACC1-Test01**
   - **Control:** Manual
   - **State:** Game interface is fully loaded with accessibility features enabled.
   - **Input:** Navigate the UI using screen reader software and keyboard-only inputs.
   - **Output:** Interface is fully navigable using accessibility tools, and key actions are accessible without a mouse.
   - **Test Case Derivation:** Ensures compliance with accessibility best practices to support users with visual or motor impairments.
   - **Test Steps:**
     (a) Launch the game.
     (b) Enable a screen reader (e.g., NVDA or VoiceOver).
     (c) Navigate the main menu, start a game, and attempt to play using only the keyboard.
   - **Validation:** Confirm that all elements are readable and operable through the accessibility tools without hindrance.

4.2.12 Compatibility Requirements (4.2.12)

These tests ensure that the game is compatible with a variety of operating systems, browsers, and hardware configurations.

1. **Test ID: COMP1-Test01**
   - **Control:** Manual
   - **State:** The game has been built and exported for different platforms.
   - **Input:** Run the game on Windows, macOS, and different browser engines (if applicable).

- **Output:** Game functions as expected across all supported configurations without graphical or functional issues.
- **Test Case Derivation:** Ensures platform consistency and browser compatibility where applicable, as outlined in the compatibility requirement in the SRS.
- **Test Steps:**
  (a) Install and run the game on Windows and macOS.
  (b) (If applicable) Load the game on Chrome, Firefox, or Safari.
  (c) Interact with the full game cycle on each platform.
- **Validation:** Confirm that gameplay, graphics, input handling, and performance remain consistent across tested platforms and browsers.

4.3 Traceability Between Test Cases and Requirements (4.3)

The following tables illustrate the traceability between test cases and the requirements specified in the Software Requirements Specification (SRS). This mapping ensures that all requirements have been addressed by at least one test case, providing comprehensive coverage and validating that the system meets all specified needs. The tables are split into 4 tables for better visibility in the document, consider it 1 big table.

Requirement ID	**Requirement Description**	**Test Case ID(s)**
FR1: Game Initialization	The game must initialize a session with player name input and enter a game lobby when started.	GI1-Test01, GI1-Test02
FR2: Game Room Setup	The game must allow creation and setup of a multiplayer session with invited players.	GSR1-Test01
FR3: Turn Management	The game must manage player turns and correctly process skip/reverse actions.	TMR1-Test01, TMR1-Test02, TMR1-Test03
FR4: Card Effects	Card effects like Draw Two, Draw Four, Wild, Skip, and Reverse must perform correctly.	CE1-Test01, CE1-Test02, CE1-Test03
FR5: Game Over Conditions	The game must detect when a player wins by playing their last card and pressing UNO.	GOC1-Test01, GOC1-Test02

Table 4: Traceability Table between Functional Requirements and Test Cases (Part 1)

Requirement ID	**Requirement Description**	**Test Case ID(s)**
FR6: Multiplayer Synchronization	All actions must be synchronized across players in real time, including disconnections.	MS1-Test01, MS1-Test02
NFR1: Appearance	The user interface must follow modern design principles and be visually appealing.	AR1-Test01
NFR2: Speed and Latency	Game response time must remain below 50-100ms in standard and high-traffic conditions.	SALR1-Test01, SALR1-Test02
NFR3: Usability	The interface must be intuitive for new users and support smooth control interactions.	UR1-Test01, UR1-Test02
NFR4: Reliability	The game must remain stable under prolonged use without crashing or performance drops.	RR1-Test01

Table 5: Traceability Table between Requirements and Test Cases (Part 2)

Requirement ID	Requirement Description	Test Case ID(s)
NFR5: Availability	The game server should have uptime of at least 99.65%, with ¡5 minutes of downtime per day.	AV1-Test01
NFR6: Security	The system must prevent unauthorized access to other players' data or gameplay.	SR1-Test01
NFR7: Maintainability	The code should be modular, documented, and follow Unity/C# standards.	MR1-Test01
NFR8: Portability	The game must function correctly on Windows and macOS builds.	PR1-Test01
NFR9: Scalability	The game must support increasing player load up to a maximum threshold without lag.	SCR1-Test01

Table 6: Traceability Table between Requirements and Test Cases (Part 3)

Requirement ID	Requirement Description	Test Case ID(s)
NFR10: Performance	The game must maintain response times under 100ms and stable FPS during play.	PRF1-Test01
NFR11: Accessibility	Game UI should be operable using screen readers and keyboard-only navigation.	ACC1-Test01
NFR12: Compatibility	The game must run correctly across Windows/macOS platforms and compatible browsers (if applicable).	COMP1-Test01

Table 7: Traceability Table between Requirements and Test Cases (Part 4)

5 Unit Test Description

This section describes the scope of unit testing for the UNO Flip Remix project, including the modules under test, their key responsibilities, and the specific test cases used to validate them. It also outlines traceability between unit test cases and the system modules defined in the Module Interface Specification (MIS). This whole section has been edited but for ease of readability, the text will remain in black.

5.1 Unit Testing Scope

Unit testing in this project is designed to validate the correctness of individual components within the Unity C# codebase. The primary focus is on testing modules that handle game logic, player interactions, and state transitions. Tests are implemented using the NUnit framework and are executed regularly during development. Each test is designed to:

- Verify the correctness of the module logic.
- Ensure each function behaves as expected under normal and edge case conditions.
- Catch regressions introduced by future changes.

5.2 Tests for Functional Requirements

This section describes unit tests for the system's functional modules. Each test targets the internal logic of a specific class or function and ensures correctness of behavior under typical and edge-case conditions. These tests were implemented using the NUnit testing framework in Unity and aim to cover gameplay logic, deck and card operations, AI decisions, player actions, and menu flow. Below the test names match the names of the unit tests implemented in our project.

5.2.1 AIPlayer Module

1. **Test ID: AI1-Test01**
    - **Control:** Automated
    - **State:** AI player has a valid hand of cards
    - **Input:** Request AI to draw a card
    - **Output:** AI hand size increases by 1
    - **Test Case Derivation:** From method: `Test_AIPlayer_CanDrawCard`
    - **Validation:** Confirm that AI's hand has one more card after the draw
2. **Test ID: AI1-Test02**
    - **Control:** Automated
    - **State:** AI player has at least one playable card
    - **Input:** Request AI to select the best card
    - **Output:** AI selects a valid card based on heuristics
    - **Test Case Derivation:** From method: `Test_AIPlayer_ChooseBestCard_PreferActionCards`
    - **Validation:** Confirm that selected card is an action card if available

5.2.2 Player Module

1. **Test ID: PL1-Test01**
    - **Control:** Automated
    - **State:** New player instance is created
    - **Input:** Instantiate Player class
    - **Output:** Player has empty hand and correct attributes
    - **Test Case Derivation:** From method: `Test_Player_Creation`
    - **Validation:** Confirm player's hand is initialized empty and name is assigned
2. **Test ID: PL1-Test02**
    - **Control:** Automated
    - **State:** Player has a card that matches the top card
    - **Input:** Request player to play a card
    - **Output:** Card is removed from hand and placed on pile
    - **Test Case Derivation:** From method: `Test_Player_PlayCard`
    - **Validation:** Confirm correct card is played and game state updates

5.2.3 Deck Module

1. **Test ID: DK1-Test01**
    - **Control:** Automated
    - **State:** Deck is initialized
    - **Input:** Call shuffle method
    - **Output:** Deck order is randomized
    - **Test Case Derivation:** From method: `Test_Deck_ShuffleChangesOrder`
    - **Validation:** Confirm deck order is different after shuffling
2. **Test ID: DK1-Test02**
    - **Control:** Automated
    - **State:** Deck is empty
    - **Input:** Draw card request is made
    - **Output:** Deck is reset from used pile
    - **Test Case Derivation:** From method: `Test_Deck_ResetsWhenEmpty`
    - **Validation:** Confirm deck is refilled and draw proceeds successfully

5.2.4 Card Module

1. **Test ID: CD1-Test01**
    - **Control:** Automated
    - **State:** New card object created
    - **Input:** Create ActionCard with color RED and action SKIP
    - **Output:** Card stores correct type and metadata
    - **Test Case Derivation:** From method: `Test_Card_ActionCard_Creation`
    - **Validation:** Confirm card properties match initialization values
2. **Test ID: CD1-Test02**
    - **Control:** Automated
    - **State:** Two cards with same color and value are compared
    - **Input:** Compare cards using equality method
    - **Output:** Cards are confirmed equal
    - **Test Case Derivation:** From method: `Test_Card_Equality`
    - **Validation:** Assert that equality operator returns true

5.2.5 GameManager Module

1. **Test ID: GM1-Test01**
    - **Control:** Automated
    - **State:** GameManager has been initialized
    - **Input:** Run setup with player list
    - **Output:** Player objects and game state are initialized
    - **Test Case Derivation:** From method: `Test_GameManager_InitializePlayers`
    - **Validation:** Confirm each player is initialized and dealt cards
2. **Test ID: GM1-Test02**
    - **Control:** Automated
    - **State:** A win condition has been triggered
    - **Input:** Player finishes last card and clicks UNO
    - **Output:** Win screen is triggered
    - **Test Case Derivation:** From method: `Test_GameManager_WinCondition`
    - **Validation:** Confirm game ends with proper winner message displayed

5.3 Tests for Nonfunctional Requirements

This section provides detailed unit tests for verifying nonfunctional requirements such as usability, maintainability, and correctness of internal module behavior. Each test aligns with specific modules and ensures software quality, robustness, and code consistency. Below the test names match the names of the unit tests implemented in our project.

5.3.1 Module 3: PlayerManager.cs

This module manages player identity, hand status, and in-game actions related to player behavior. It is essential for maintaining correct player state across game sessions.

1. **Test ID: UT-PM01**
    - **Control:** Automated
    - **State:** PlayerManager object is initialized with a given player name.
    - **Input:** Call `GetPlayerName()`.
    - **Output:** The correct player name string is returned.
    - **Test Case Derivation:** Verifies proper assignment and retrieval of player name during initialization.
    - **Validation:** `Assert.AreEqual("Player1", GetPlayerName())`
    - **Associated Test:** `Test_Player_Creation`
2. **Test ID: UT-PM02**
    - **Control:** Automated
    - **State:** Player hand is initialized with 7 cards.
    - **Input:** Player plays one card via `PlayCard()`.
    - **Output:** Hand size decreases by one.
    - **Test Case Derivation:** Ensures card is removed from internal hand list after a valid play.
    - **Validation:** `Assert.AreEqual(6, player.Hand.Count)`
    - **Associated Test:** `Test_Player_PlayCard`

5.3.2 Module 4: Card.cs

The Card module defines attributes and equality logic for UNO Flip cards. It ensures that all card properties—color, type, and value—are valid and support comparisons and game logic.

1. **Test ID: UT-CARD01**
    - **Control:** Automated

- **State:** A card is created using constructor parameters for color and type.
- **Input:** Invoke `ToString()` or inspect color/type fields.
- **Output:** The card reflects expected attributes (e.g., Red Draw Two).
- **Test Case Derivation:** Ensures the constructor stores the correct card metadata.
- **Validation:** `Assert.AreEqual(CardType.DrawTwo, newCard.Type)`
- **Associated Test:** `Test_Card_DrawFourCard_Creation`

2. **Test ID: UT-CARD02**

- **Control:** Automated
- **State:** Two cards are instantiated with matching values.
- **Input:** Invoke `card1.Equals(card2)`.
- **Output:** Returns `true` if the cards match in all attributes.
- **Test Case Derivation:** Ensures equality checks are implemented correctly for game logic.
- **Validation:** `Assert.IsTrue(card1.Equals(card2))`
- **Associated Test:** `Test_Card_Equality`

5.3.3 Module 5: MenuManager.cs

This module handles the game's menu UI including navigation and screen transitions.

1. **Test ID: UT-MM01**

- **Control:** Automated
- **State:** MenuManager is loaded and the main menu is active.
- **Input:** Call `ExitGame()` method.
- **Output:** Application quits or exits to desktop (in runtime or simulated test environment).
- **Test Case Derivation:** Confirms correct response to user pressing "Exit" in menu.
- **Validation:** `Assert.True(Application.isQuitting)` (or test mock behavior)
- **Associated Test:** `Test_Menu_ExitGame`

2. **Test ID: UT-MM02**

- **Control:** Automated
- **State:** How-To-Play panel component is present but inactive.
- **Input:** Call `ToggleHowToPlayPanel()`.

- **Output:** Panel becomes visible and interactive.
- **Test Case Derivation:** Verifies help/instruction UI elements can be toggled as designed.
- **Validation:** `Assert.IsTrue(panel.activeSelf)`
- **Associated Test:** `Test_Menu_HowToPlayPanel_Activation`

5.4 Traceability Between Test Cases and Modules

This section provides a traceability matrix that maps unit test cases to the corresponding software modules they verify. This mapping ensures complete coverage of both functional and nonfunctional aspects of the UNO Flip Remix game implementation.

Module	**Requirement Description**	**Unit Test Case(s)**
`GameManager.cs`	Handles overall game logic, including player initialization, direction, and win condition tracking.	UT-GM01, UT-GM02
`DeckManager.cs`	Manages deck setup, shuffling, drawing, and recycling of used cards.	UT-DM01, UT-DM02
`CardDisplay.cs`	Handles visual representation of cards based on type and properties.	UT-CD01, UT-CD02
`CardInteraction.cs`	Handles mouse hover effects and interaction visuals.	UT-CI01
`PlayerManager.cs`	Tracks player identity, hand size, and in-game actions.	UT-PM01, UT-PM02
`Card.cs`	Models UNO Flip cards, including types, colors, and equality checks.	UT-CARD01, UT-CARD02
`MenuManager.cs`	Controls main menu behavior, UI transitions, and help screens.	UT-MM01, UT-MM02

Table 8: Traceability Table between Unit Test Cases and Modules

6 References

1. Uno Flip Online Game, "Play Uno Flip Online," [Online]. Available: https://unoonlinegame.io/uno-flip-online. Accessed: Nov. 1, 2024.

2. Software Testing Help, "Test Plan Sample – Software Testing and Quality Assurance Templates," [Online]. Available: https://www.softwaretestinghelp.com/test-plan-sample-softwaretesting-and-quality-assurance-templates/. Accessed: Nov. 1, 2024.

3. McMaster University, "Integrated Stroke Test Plan," GitLab, [Online]. Available: https://gitlab.cas.mcmaster.ca/courses/capstone/-/blob/main/SamplesOfStudentWork/VnVPlan/IntegratedStrokeTestPlan.pdf. Accessed: Nov. 1, 2024.

6.1 Symbolic Parameters

6.2 Usability Survey Questions

The following usability survey was distributed to evaluate the overall user experience of the UNO Flip Remix game. Questions focused on interface clarity, controls, responsiveness, accessibility, and enjoyment. The survey was administered through Google Forms.

1. What is your age group?
   - Under 18
   - 18–24
   - 25–34
   - 45 or older
2. Have you played UNO Flip before?
   - Yes
   - No
3. How would you rate the game interface design? (1 = Very Poor, 5 = Excellent)
   - 1, 2, 3, 4, 5
4. Are the text and icons clear and easy to read?
   - Very clear
   - Somewhat clear
   - A little blurry
   - Very difficult to read
5. Do the colors and visual effects feel comfortable to you?

- Yes, very comfortable
- Neutral
- No, the colors feel messy
- No, the colors are too harsh on the eyes

6. What aspects of the game interface do you like the most? (Select all that apply)
   - Color scheme
   - Card design
   - Button layout
   - Fonts and text arrangement
7. Do you find the game controls intuitive and easy to understand?
   - Very intuitive, I learned quickly
   - Fairly intuitive, but took some time to get used to
   - A bit complicated, not easy to pick up
   - Very confusing, poor user experience
8. Are the drag/click actions for playing cards smooth and responsive?
   - Yes, very smooth
   - Mostly smooth, but sometimes laggy
   - Somewhat laggy, needs improvement
   - Very laggy, affects gameplay
9. Is the Flip mechanism easy to understand and use?
   - Yes, very clear
   - Takes a little time to get used to
   - Quite difficult to understand, needs better guidance
   - Completely unclear
10. Did you encounter any of the following issues while playing? (Select all that apply)
    - Game freezes or crashes
    - Unresponsive controls or delayed responses
    - Unclear rules or lack of guidance
    - Confusing interface or hard-to-find buttons
11. Did you find playing UNO Flip enjoyable?
    - Very fun, I loved it

- Pretty good, but not outstanding
- Average, a little boring
- Not enjoyable, I wouldn't play again

12. How would you rate the difficulty level?
    - Too easy, not challenging enough
    - Just right, enjoyable to play
    - Too difficult, a bit frustrating

Appendix — Reflection

The following reflections evaluate the team's engagement during the creation of this VnV Plan.

1. **What went well while writing this deliverable?**
    - Mingyang: We used a shared document platform for real-time collaboration. This allowed us to continuously review and build on each other's work, ensuring consistent formatting and avoiding duplicated efforts.
    - Kevin: Everyone contributed evenly and on time, which made the writing process more manageable and less stressful.
2. **What pain points did you experience during this deliverable, and how did you resolve them?**
    - Mingyang: Distinguishing between functional and non-functional requirements was initially confusing. We resolved this by revisiting course resources and applying detailed feedback from the TAs.
    - Zain: Understanding how to properly format test cases in LaTeX took some trial and error, but collaborating with teammates and referencing examples helped fix the issues.
3. **What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project?**
    - Jianhao: We need to improve in static testing for requirements traceability and dynamic testing for performance and multiplayer features. Familiarity with Unity's built-in testing tools is essential.
    - Kevin: We should get better at writing testable code from the beginning to reduce debugging later in the development process.
4. **For each of the knowledge areas and skills identified, what are at least two approaches to acquiring the knowledge or mastering the skill?**

- Jianhao: We plan to review documentation for Unity Test Framework and watch tutorials on performance profiling. Additionally, we'll do peer learning sessions to share experience with test case writing and automated test setup.
- Zain: I will explore Unity Learn tutorials and ask teammates to walk through their test cases with me so I can learn the structure and logic behind them.