System Verification and Validation Plan for Uno-Flip 3D

Team 24, Uno-Flip 3D
Mingyang Xu
Kevin Ishak
Jianhao Wei
Zheng Bang Liang
Zain-Alabedeen Garada

Nov 3, 2024

Revision History

Date	Version	Notes
2024.Oct.29	1.0	Created a shared file
2024.Oct.30	1.1	Wrote section 1-2
2024.Oct.31	1.2	Wrote section 3 and modified sections 1 and 2
2024.Nov.1	1.3	Wrote content for section 4 and reflection

Contents

List of Tables

Table Number	Title	Page Number
1	Summary of tables list	2
2	team member's responsibility	7
3	Traceability table I	33
4	Traceability table II	34

Table 1: Summary of tables list

1 Symbols, Abbreviations, and Acronyms

symbol	description
SRS	System Requirements Specification
PSAG	problem statement and goals

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.

These objectives are chosen based on our resource constraints, prioritizing core functionality and gameplay experience over advanced features.

2.3 Challenge Level and Extras

The challenge level for this project is 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.

Extras included in the project:

- Usability testing to ensure the game provides an enjoyable user experience.
- User documentation to guide players on gameplay rules and controls.
- Design Thinking techniques applied during the design phase to create an engaging interface.

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.UNO-Flip-3D/docs/SRS/SRS.pdf at main · zgarada/UNO-Flip-3D

PSAG: This document defines our problem statement and goals.

UNO-Flip-3D/docs/ProblemStatementAndGoals/ProblemStatement.pdf at main · zgarada/UNO-Flip-3D

Uno-flip : This is a game that can be played online by unoflip, and then we will modify and refer to it according to this project.Uno Flip!

3 Plan

This section provides an overview of the planned verification and validation (VnV) 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. 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 overall VnV process, ensures adherence to the VnV plan, and coordinates the team's efforts.
Jiahao Wei	SRS and Design Reviewer	Conducts in-depth reviews of the SRS and design documents, providing feedback on requirement clarity and design feasibility.
Zheng Bang Liang	Code Verification Specialist	Responsible for implementing unit and integration tests, and conducting code inspections and reviews.

Zain-Alabedeen Garada	System Test Engineer	Designs and executes system tests, including functional and non-functional tests, and reports on test coverage and outcomes.
Kevin Ishak	System Test Engineer	Designs and executes system tests, including functional and non-functional tests, and reports on test coverage and outcomes.

Table 2: team member's responsibility

3.2 SRS Verification Plan

1. **Review Process**:
   - The SRS document will undergo a **formal peer review** with the primary reviewer being Student 2, supported by the external reviewer (supervisor). This review will assess requirement clarity, completeness, and feasibility.
   - **Meeting with Supervisor**: A dedicated meeting will be arranged with our supervisor to go over the SRS document. During this session, we will present the requirements, ask targeted questions to clarify expectations, and gather any suggestions for improvement.
2. **SRS Checklist**:
   - We will develop a detailed **SRS checklist** to ensure that all requirements are specific, measurable, achievable, relevant, and time-bound (SMART). This checklist will cover critical aspects like requirement consistency, clarity, traceability, and testability.
   - The checklist will be shared with the entire VnV team, and each member will verify sections of the SRS relevant to their role, such as functional requirements, non-functional requirements, and any assumptions or dependencies.
3. **Use of Issue Tracker**:
   - All identified issues during the SRS review will be logged in our project's **GitHub issue tracker**. Each issue will include a detailed description, relevant section references, and suggestions for resolution. This approach will facilitate

transparent tracking of revisions and provide a record of feedback incorporated into the final SRS document.

4. **Task-Based Inspection**:
    - We will perform a task-based inspection where each team member reviews a specific portion of the SRS related to their responsibility. For instance, Student 3 will focus on ensuring the requirements are testable from a coding perspective, while Student 4 will confirm that the system tests align with specified functional requirements.
5. **Ad Hoc Feedback**:
    - In addition to the formal review, we will seek **ad hoc feedback** from classmates and peers, particularly those experienced in software design and requirements engineering. This feedback will allow us to address any overlooked areas and improve the SRS further before finalizing.

By combining structured reviews, checklists, and feedback loops, this SRS Verification Plan aims to ensure that the requirements are well-defined and provide a solid foundation for the subsequent design and implementation phases.

3.3 Design Verification Plan

To verify the design of the project, we will implement a structured review process that involves peer feedback, checklists, and verification through multiple review stages:

1. **Peer Reviews**:
    - The design document will undergo thorough reviews by team members and classmates. Each member will review sections related to their responsibilities, ensuring design consistency, feasibility, and alignment with the Software Requirements Specification (SRS).

    - We will hold a peer review meeting with classmates for external feedback on design choices, focusing on areas such as data flow, system architecture, and error handling.
2. **Checklist-Based Verification**:
    - A checklist specific to design verification will be created to ensure the design aligns with the system requirements. The checklist will cover elements like modularity, reusability, scalability, and maintainability.
    - Team members will use this checklist to verify that the design meets required standards before moving forward to implementation.
3. **Review Meeting with Supervisor**:
    - A design review session will be scheduled with our supervisor, where the team will present the high-level architecture, major components, and interactions between modules.
    - During this meeting, the supervisor's feedback will be documented and incorporated into the design.
4. **Issue Tracker**:
    - Any issues or suggested improvements identified during the review process will be documented in the GitHub issue tracker, allowing us to track modifications and address any recurring design challenges.

This Design Verification Plan ensures that the design is feasible, addresses all specified requirements, and is prepared for the implementation phase.

3.4 Verification and Validation Plan Verification Plan

The Verification and Validation (VnV) Plan itself is also an artifact that must be verified to ensure it comprehensively addresses all aspects of the VnV process. We will use the following approaches:

1. **Peer Review and Checklist**:

    - The VnV plan will be reviewed by team members and classmates, focusing on the clarity, completeness, and feasibility of the outlined processes. A VnV checklist will be used to verify that all necessary sections are included and adequately detailed.
2. **Mutation Testing**:
    - We will conduct mutation testing on select test cases outlined in the VnV plan. This process involves intentionally introducing small changes to code or test inputs to evaluate if the VnV plan can detect these errors, thus verifying the effectiveness of the plan.
3. **Classmate Feedback**:
    - Feedback from classmates will be solicited to identify any gaps or areas for improvement in the VnV plan. They will review sections such as system tests, traceability, and the use of tools to confirm our approach's comprehensiveness.
4. **Supervisor Review**:
    - The supervisor will review the VnV plan as an external reviewer, providing insights on the feasibility and thoroughness of our testing and verification processes. Any feedback will be documented and addressed accordingly.
5. **Checklist Creation**:
    - A specific checklist for the VnV Plan verification will be developed, covering all necessary elements like system tests, unit tests, traceability, and non-dynamic testing. This checklist will ensure that each component of the VnV plan meets expected standards and fulfills the project's verification needs.

By following this verification plan for the VnV document, we aim to confirm that the VnV processes are well-defined, achievable, and robust, providing confidence in the project's overall verification strategy.

3.5 Implementation Verification Plan

The Implementation Verification Plan outlines the strategies and methods to verify the correctness and reliability of the software during the implementation phase.

1. **Unit Testing**:
   - Each module and function will be verified through detailed unit tests. These tests will be implemented based on the unit testing plan and aim to cover all possible input conditions, including edge cases.
   - The unit tests will be automated and run continuously to ensure that any new code modifications do not introduce errors.
2. **Static Code Analysis**:
   - Static analyzers will be used to inspect the source code for potential bugs, adherence to coding standards, and areas of improvement. This tool will highlight any syntax errors, unused variables, and logical errors before runtime.
   - The results from the static analysis will be reviewed and issues will be resolved as per priority levels.
3. **Code Walkthroughs and Inspections**:
   - Regular code walkthroughs will be conducted among team members. In these sessions, the team will review each other's code, focusing on readability, structure, and adherence to design.
   - These walkthroughs serve as an additional layer of verification to ensure the code is reliable and maintains consistency with the design specifications.
4. **Final Class Presentation as Code Walkthrough**:
   - The final presentation in CAS 741 will serve as a formal code walkthrough. During this presentation, the team will demonstrate the code structure, explain its functionality, and address any questions from peers and the instructor.
   - Feedback from this presentation will be used to make necessary modifications and improvements to the code.

This Implementation Verification Plan ensures that the software is thoroughly checked for quality and reliability during the coding phase.

3.6 Automated Testing and Verification Tools

We will utilize a suite of automated tools for testing and verification to ensure efficiency and consistency across the project:

1. **Unit Testing Framework**:
   - A unit testing framework will be employed to automate unit tests. This framework allows us to test individual components, ensure they work as expected, and quickly identify any breaking changes during development.
2. **Code Coverage Tools**:
   - Code coverage tools will be used to track the extent of code covered by tests. By monitoring code coverage metrics, we can ensure that critical parts of the code are well-tested and identify any areas that may need additional testing.
3. **Continuous Integration (CI)**:
   - A CI tool (such as GitHub Actions or Jenkins) will be set up to automate the testing process. Every time a new change is pushed to the repository, the CI system will run all tests, ensuring that new changes do not introduce regressions.
4. **Static Analyzer**:
   - A static analyzer will be incorporated into the development process to check for coding standard violations, potential bugs, and security vulnerabilities. For instance, flake8 or pylint could be used for Python projects, while other languages have equivalent tools.
5. **Linters**:
   - Linter tools will be used to enforce code formatting and style standards. Linters help maintain consistency across the codebase, making the code easier to read and less prone to errors.

By integrating these automated tools, the project can benefit from continuous feedback, early detection of issues, and improved code quality.

3.7 Software Validation Plan

The Software Validation Plan is designed to validate that the software meets the user requirements and operates as intended within the real-world context.

1. **User Review Sessions**:
   - Regular review sessions will be conducted with stakeholders or users (if available) to verify that the software meets the functional requirements specified in the SRS. These sessions will focus on ensuring the software's usability and alignment with the expected user experience.
2. **Rev 0 Demo for External Supervisor**:
   - For teams with an external supervisor, the Rev 0 demo will provide an opportunity to validate the software's core functionality. This demonstration will showcase the initial implementation of key features, and the supervisor's feedback will be used to guide further development.
   - Any feedback provided will be recorded, and adjustments will be made to the software to better meet user needs.
3. **User Testing**:
   - In the absence of an external supervisor, user testing will serve as an alternative. Users will interact with the system, and their feedback on usability, functionality, and satisfaction will be gathered to validate that the software is fulfilling its intended purpose.
4. **Task-Based Inspection**:
   - Task-based inspections will be conducted to validate that the requirements document captures the right user requirements. This method involves checking if specific tasks, as expected by

the stakeholders, can be completed effectively and efficiently within the software.

5. **Reference to SRS Verification**:
    - The validation process will reference the SRS verification (3.2) to ensure that the software fulfills all documented requirements. Any discrepancies between the SRS and software behavior will be noted and resolved.

This Software Validation Plan ensures that the final product aligns with the stakeholder's expectations and effectively serves the end-users.

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.

4.1 Tests for Functional Requirements

4.1.1 Area of Testing: Authentication

The Authentication tests are critical to ensure that only registered users can access the game and that the security of user credentials is maintained. This area includes tests for standard login operations as well as edge cases, such as handling incorrect credentials and verifying session security. The tests are derived from the SRS, specifically the sections covering security and access control requirements.

1. **Test ID:** `AR1-Test01`
    - **Control:** Manual
    - **State:** No precondition required

    - **Input:** Log in using valid username and password for an existing account.
    - **Output:** User successfully logs in and accesses the multiplayer mode.
    - **Test Case Derivation:** Verifies that the authentication system correctly identifies and validates existing users.
    - **Test Steps:**
        1. Open the login screen.
        2. Enter a valid username and password.
        3. Click the login button.
    - **Validation:** Confirm that the user is directed to the game's main screen and can access the multiplayer mode.
2. **Test ID:** `AR2-Test02`
    - **Control:** Manual
    - **State:** User is logged in and has a profile.
    - **Input:** Access the user profile page.
    - **Output:** User sees game statistics and preferences.
    - **Test Case Derivation:** Verifies that the profile page correctly displays user-specific data.
    - **Test Steps:**
        1. Log in and access the main screen.
        2. Click the "Profile" button.
    - **Validation:** Confirm that the displayed game statistics (e.g., win rate, preferences) match the user's records.

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:** No precondition required
- **Input:** User clicks "Create Room" button.
- **Output:** A new game room is created with options to invite friends.
- **Test Case Derivation:** Verifies that the game room creation functionality works as expected.
- **Test Steps:**
  1. On the main screen, click "Create Room."
  2. Check if the invite option is displayed.
- **Validation:** Confirm that a new game room is created and the user can invite other players.

2. **Test ID:** `GSR2-Test02`
   - **Control:** Manual
   - **State:** User is on the game setup page.
   - **Input:** User selects game rules and difficulty level.
   - **Output:** The selected rules and difficulty are applied to the game.
   - **Test Case Derivation:** Verifies that user-selected settings are correctly applied to the game.
   - **Test Steps:**
     1. Access the game setup page.
     2. Choose specific game rules and difficulty level.
     3. Start the game.
   - **Validation:** Confirm that the game starts with the selected rules and difficulty settings.

4.1.3 Area of Testing: Turn Management

Turn Management is a fundamental part of the game's mechanics, ensuring fair and organized gameplay. This area includes tests that confirm the correct sequencing of turns, proper handling of turn-based effects from special cards, and notification alerts for players. References to the SRS are used to confirm that turn management requirements are met according to the gameplay rules.

1. **Test ID:** `TMR1-Test01`
    - **Control:** Automatic
    - **State:** Game in progress with multiple players
    - **Input:** System manages player turns.
    - **Output:** System synchronizes player turns across all devices.
    - **Test Case Derivation:** Ensures that turn management functions in real-time across devices.
    - **Test Steps:**
        1. Start a multiplayer game with at least two players.
        2. Each player completes their turn.
    - **Validation:** Confirm that each device displays the current turn consistently.
2. **Test ID:** `TMR2-Test02`
    - **Control:** Automatic
    - **State:** Player's turn is approaching.
    - **Input:** System sends a notification for the player's turn.
    - **Output:** Player receives a notification when it is their turn.
    - **Test Case Derivation:** Verifies that the notification system works as intended for turn alerts.
    - **Test Steps:**
        1. Player joins the game.
        2. System notifies the player when it's their turn.
    - **Validation:** Confirm that the player receives a notification when their turn arrives.

4.1.4 Area of Testing: Chat Functionality

The Chat Functionality area is important for enhancing user interaction in multiplayer sessions. Tests in this area validate that chat messages are correctly transmitted in real-time, that messages are received without delays, and that users can toggle chat functionality. SRS references ensure that all chat-related requirements are fulfilled.

1. **Test ID:** `CFR1-Test01`
    - **Control:** Manual
    - **State:** Player is in the game room.
    - **Input:** Player sends a chat message.
    - **Output:** Message is displayed in real-time for all players in the chat window.
    - **Test Case Derivation:** Verifies that chat messages are instantly delivered to all players.
    - **Test Steps:**
        1. Player types and sends a message in the chat window.
    - **Validation:** Confirm that the message appears in the chat window for all players in real-time.
2. **Test ID:** `CFR1-Test02`
    - **Control:** Manual
    - **State:** Chat feature is disabled.
    - **Input:** Attempt to enable or disable the chat feature.
    - **Output:** Chat is toggled on or off according to user preference.
    - **Test Case Derivation:** Ensures that players can control the chat feature's availability.
    - **Test Steps:**
        1. In settings, enable or disable the chat feature.
    - **Validation:** Confirm that the chat is available when enabled and inaccessible when disabled.

4.1.5 Area of Testing: Scoring System

The Scoring System is essential for competitive gameplay, and accurate scoring is critical for player satisfaction. This area of testing includes validating score calculation, score display at the end of each game, and proper tracking of win/loss records. The SRS serves as a reference to verify that the scoring functionality meets specified accuracy requirements.

1. **Test ID:** `SSR1-Test01`
    - **Control:** Automatic

- **State:** Game has ended.
- **Input:** System triggers score display.
- **Output:** Final scores of all players are displayed accurately.
- **Test Case Derivation:** Ensures that the scoring system calculates and displays final scores correctly.
- **Test Steps:**
  1. Complete a round of the game.
  2. Check the final score display.
- **Validation:** Confirm that the final scores match the game outcomes.

2. **Test ID:** `SSR2-Test02`
   - **Control:** Automatic
   - **State:** Game has ended.
   - **Input:** Access game history.
   - **Output:** Displays player's win/loss records and average scores.
   - **Test Case Derivation:** Verifies that historical statistics are accurately maintained.
   - **Test Steps:**
     1. Finish a game round.
     2. Access the player's record page.
   - **Validation:** Confirm that the historical data matches the actual game results.

4.1.6 Area of Testing: End Game Handling

End Game Handling tests ensure that the game ends correctly, a winner is declared based on the game rules, and players are presented with options to continue or exit. This area confirms that the game state transitions smoothly and that end-game results are displayed as specified in the SRS.

1. **Test ID:** `EGHR1-Test01`
   - **Control:** Automatic

- **State:** Game ends according to rules.
- **Input:** Trigger end game event.
- **Output:** System declares a winner and displays a summary of results.
- **Test Case Derivation:** Verifies that end-game conditions are properly implemented.
- **Test Steps:**
  1. Complete a game with a clear outcome.
  2. Verify that the system displays the winner and result summary.
- **Validation:** Confirm that the system correctly declares the winner and provides a summary.

2. **Test ID:** `EGHR2-Test02`
   - **Control:** Manual
   - **State:** Game has ended.
   - **Input:** Player selects replay or exit.
   - **Output:** System allows player to either start a new round or exit the game.
   - **Test Case Derivation:** Verifies that players can choose to continue or leave after the game ends.
   - **Test Steps:**
     1. Finish a game round.
     2. Click "Replay" or "Exit."
   - **Validation:** Confirm that the system responds to the player's choice by starting a new game or exiting.

4.1.7 Area of Testing: Multiplayer Synchronization

Multiplayer Synchronization is vital for a smooth, consistent multiplayer experience. This area of testing verifies that all game state updates are reflected for each player without delay, even under challenging network conditions. SRS references confirm that synchronization meets the requirements for real-time multiplayer gameplay.

1. **Test ID:** `MSR1-Test01`

- **Control:** Automatic
- **State:** Game in progress with multiple players.
- **Input:** Synchronize game state across players.
- **Output:** All players see the same game state regardless of network conditions.
- **Test Case Derivation:** Ensures synchronization consistency across all players under varying network conditions.
- **Test Steps:**
    1. Start a multiplayer game.
    2. Check the game state on each player's device.
- **Validation:** Confirm that all players see the same game state at the same time.

2. **Test ID:** `MSR2-Test02`
    - **Control:** Automatic
    - **State:** Game in progress with network delay.
    - **Input:** System manages latency.
    - **Output:** Smooth, real-time game interaction is maintained despite network delays.
    - **Test Case Derivation:** Verifies that the system remains responsive under network delay.
    - **Test Steps:**
        1. Simulate network delay.
        2. Observe the synchronization and response time.
    - **Validation:** Confirm that gameplay remains smooth and synchronized despite delays.

4.2 Tests for Nonfunctional Requirements

4.2.1 Area of Testing: Appearance Requirements

Title for Test: Interface Modernity and Consistency

The interface modernity and consistency testing ensures the game provides the best up-to-date visual experience for the users without inconsistency or inclarity, which could cause confusion and uncomfortableness when the user navigates through the game. These tests confirm the appearance requirements in the SRS document

1. Test ID: AR1-Test01
    - Type: Functional, Manual, Static
    - Initial State: Game interface is loaded on various screens.
    - Input/Condition: User navigates through different game screens.
    - Output/Result: All screens have a modern, visually appealing design.
    - How test will be performed: Tester inspects each screen for visual appeal and modern design.
2. Test ID: AR2-Test02
    - Type: Functional, Manual, Static
    - Initial State: Game elements are loaded.
    - Input/Condition: Inspect color schemes, fonts, buttons, and animations.
    - Output/Result: Consistent color schemes, fonts, button styles, and animations across all screens.
    - How test will be performed: Tester navigates all screens, ensuring UI consistency.
3. Test ID: AR3-Test03
    - Type: Functional, Manual, Static
    - Initial State: Card designs rendered on the game screen.
    - Input/Condition: Review clarity of card designs in 3D rendering.
    - Output/Result: All card designs are easily recognizable and clear in 3D view.
    - How test will be performed: Tester examines cards during gameplay to ensure clarity.

4.2.2 Area of Testing: Style Requirements

Title for Test: Responsive Animations and Adaptivity

The responsive animations and adaptivity is vital for the overall user experience as problems such as lagging or incompatible interface could cause the game to be unrealistic and uncomfortable on the users side. These tests ensure all elements are displayed on different screens correctly and all animations have the desired quality. These tests confirm the style requirement in the SRS document.

1. Test ID: SR1-Test01
   - Type: Functional, Dynamic, Manual
   - Initial State: Interface loaded on different device resolutions.
   - Input/Condition: Open the game on mobile, tablet, and desktop devices.
   - Output/Result: Interface adapts properly on each device type.
   - How test will be performed: Tester verifies if the interface elements are displayed correctly on different screens.
2. Test ID: SR2-Test02
   - Type: Functional, Dynamic, Manual
   - Initial State: Game is in progress.
   - Input/Condition: Trigger animations for actions like flipping or playing a card.
   - Output/Result: Animations are smooth and responsive to player actions.
   - How test will be performed: Tester performs actions in-game and observes animation quality.

4.2.3 Area of Testing: Ease of Use Requirements

Title for Test: Tutorial Clarity and Navigation Simplicity

Since learning always takes effort, and no one wants to do extensive learning. The clarity in the tutorial and the simplicity of the navigation is important for the user because it takes less effort for the user to achieve familiarity with the game setup. These tests are set up to ensure the user has a positive experience with the tutorial and manu functionality. These tests confirm the ease of use requirement in the SRS document.

1. Test ID: EOUR1-Test01
    - Type: Functional, Manual, Dynamic
    - Initial State: Tutorial screen open for new players.
    - Input/Condition: Follow the tutorial instructions.
    - Output/Result: New players can easily understand basic gameplay.
    - How test will be performed: Tester follows the tutorial and evaluates clarity.
2. Test ID: EOUR2-Test02
    - Type: Functional, Manual, Dynamic
    - Initial State: Game menu open.
    - Input/Condition: Attempt to navigate the menu and control buttons.
    - Output/Result: User can easily create or join rooms through the menu.
    - How test will be performed: Tester navigates the menu and checks for ease of access to each function.

4.2.4 Area of Testing: Personalization and Localization Requirements

Title for Test: Language Localization and Personal Settings

It is important that the app displays the language that the user from different countries knows, otherwise they won't be able to communicate with the game. The personal setting also needs to be tested to make sure the user can save the preference they like. So these tests are set up to ensure the users can switch to different languages smoothly and their personal setting can be saved without extra effort. These tests confirm the personalization and localization requirements in our SRS document.

1. Test ID: PALR1-Test01
    - Type: Functional, Manual, Static
    - Initial State: Game settings open.
    - Input/Condition: Change language setting to other languages like French or Spanish.
    - Output/Result: All text displays correctly in the chosen language.
    - How test will be performed: Tester switches languages and verifies translations.
2. Test ID: PALR2-Test02
    - Type: Functional, Manual, Dynamic
    - Initial State: Game settings open.
    - Input/Condition: Customize sound effects, background music, and themes.
    - Output/Result: User's personalization preferences are applied and saved.
    - How test will be performed: Tester changes settings and confirms they are saved and reflected in gameplay.

4.2.5 Area of Testing: Learning Requirements

Title for Test: Tutorial Effectiveness

It is important that tutorials we create in this app do not consume too much of the users' valuable time and makes the user impatient about the rules of the game. So this test is set up to make sure the tutorial effectiveness is vital for the overall experience of the user. This test confirms the learning requirement in our SRS document.

1. Test ID: LR1-Test01
    - Type: Functional, Manual, Dynamic
    - Initial State: Interactive tutorial open.
    - Input/Condition: Follow the tutorial to understand game rules.
    - Output/Result: Player can understand rules and effects of special cards.
    - How test will be performed: Tester follows tutorial steps and confirms understanding.

4.2.6 Area of Testing: Understandability and Friendliness Requirements

Title for Test: Friendly Language for Notifications

It is important for the notification of the app to be concise and without errors as the most users will skim through the notification. This test checks the correctness of notifications to ensure the language is friendly to the user without jargon. This test confirm the understandability and friendliness requirements in our SRS documents

1. Test ID: UAFR1-Test01
    - Type: Functional, Manual, Static
    - Initial State: Notification triggered.
    - Input/Condition: Review the content of system notifications or error messages.
    - Output/Result: Messages are concise and use friendly language without jargon.
    - How test will be performed: Tester triggers various notifications and evaluates their clarity.

4.2.7 Area of Testing: Accessibility Requirements

Title for Test: Accessibility Settings Functionality

The accessibility setting functionality is vital for users with access disability, and it can greatly reduce the difficulties they encounter while accessing the game so they could also enjoy the game just like everybody else. This test will test the effect of color blind mode, text enlargement mode and soundless mode to see whether they will take effect immediately and efficiently. This test confirms the accessibility requirements in our SRS document

1. Test ID: AR1-Test01
    - Type: Functional, Manual, Dynamic
    - Initial State: Accessibility settings open in game options.
    - Input/Condition: Enable colorblind mode, text enlargement, and soundless mode.
    - Output/Result: Accessibility adjustments take effect immediately.
    - How test will be performed: Tester enables each accessibility feature and verifies the expected changes.

4.2.8 Area of Testing: Speed and Latency Requirements

Title for Test: Response and Network Latency

It is important that the game responds to the user's and other players' actions swiftly to ensure the user has a positive experience. This test is set up to test whether the time that the system responds to user input is within an acceptable range, and also whether the game can synchronize other players' action in a reasonable amount of time. The SRS serves as a reference to verify the response time of the system meets the speed and latency requirements begin defined.

1. Test ID: SALR1-Test01
    - Type: Functional, Dynamic
    - Initial State: Game in multiplayer mode.
    - Input/Condition: Measure response time between player actions and game reaction.
    - Output/Result: Average response time is within 50 milliseconds.
    - How test will be performed: Tester performs actions in-game while measuring response time.
2. Test ID: SCR1-Test01
    - Type: Functional, Dynamic
    - Initial State: Multiplayer game with multiple players.
    - Input/Condition: Change card or game state.
    - Output/Result: All players see synchronized, real-time updates.
    - How test will be performed: Tester simulates gameplay with multiple players to observe synchronization.

4.2.9 Area of Testing: Safety-Critical Requirements

Title for Test: Crush handling mechanism

It is inevitable that the game crashes during the time of use because of the the software internal issues. So this test is set up to test whether the game can handle the crush properly and save user data and states. This test confirms the safety-critical requirements in the SRS document.

1. Test ID: SCR1-Test01
    - Type: Functional, Dynamic
    - Initial State: Multiplayer game with multiple players.
    - Input/Condition: Intentionally make the game crush.
    - Output/Result: The game prompt to the user about the crash information, save all user information and current state, and restarts
    - How test will be performed: Tester make the game crush and verify the critical error handling mechanism

4.2.10 Area of Testing: Precision or Accuracy Requirements

Title for Test: Accurate Score Calculation

The score calculation is crucial for determining the win or lose of the user, so it is very important for the game to calculate the score accurately. This test is set up to determine whether the score is calculated in an accurate manner and no error is generated when the number is piled up. This test confirms the precision or accuracy requirements in SRS document

1. Test ID: POAR1-Test01
    - Type: Functional, Automatic
    - Initial State: Game score calculation triggered.
    - Input/Condition: Play through and complete a game round.
    - Output/Result: Scores are calculated with high precision.
    - How test will be performed: Tester completes a round and verifies score accuracy.

4.2.11 Area of Testing: Reliability and Availability Requirements

Title for Test: System Uptime and Reliability

Too much down time for the system could lead to negative user experience. We must ensure the system is up running for 99% of the time. This test ensures the system is running within 99% of the time by monitoring the system activity. This test confirms the reliability and availability requirement in SRS document.

1. Test ID: RAAR1-Test01
    - Type: Non-Functional
    - Initial State: Server is online.
    - Input/Condition: Simulate gameplay over an extended period.
    - Output/Result: System uptime meets 99% reliability standard.
    - How test will be performed: System monitored continuously to measure uptime.

4.2.12 Area of Testing: Robustness or Fault-Tolerance Requirements

Title for Test: Network Interruption Resilience

It is important for the game to save user information during times of disconnecting the internet to ensure the game can resume properly. This test ensures that all user data and game progress can be saved during internet disruption so the game can resume properly afterward. This test confirms the robustness and fault-tolerance requirements in the SRS document. For the fault-tolerance part, please see section 4.2.9.

1. Test ID: ROFR1-Test01
    - Type: Functional, Dynamic
    - Initial State: Game in progress with an active network connection.
    - Input/Condition: Simulate a temporary network disconnection.
    - Output/Result: Player can reconnect without losing game progress.
    - How test will be performed: Tester disconnects network and reconnects, verifying game progress continuity.

4.3 Traceability Between Test Cases and Requirements

Requirement ID	Requirement Description	Test Case ID
AR1	The game interface should have a modern design that is visually appealing.	AR1-Test01
AR2	Consistency in color schemes, fonts, buttons, and animation effects should be maintained.	AR2-Test02
AR3	Card designs should be easily recognizable, with clarity during 3D rendering.	AR3-Test03
SR1	The interface should be adaptive for different screen resolutions on mobile, tablet, and desktop.	SR1-Test01
SR2	Actions like flipping or playing a card should feature smooth and responsive animations.	SR2-Test02
EOUR1	New players should quickly grasp gameplay with clear tutorials.	EOUR1-Test01

EOUR2	Game menu and control buttons should be easy to navigate.	EOUR2-Test02
PALR1	The game should support multiple language options.	PALR1-Test01
PALR2	Players can customize settings like sound effects, music, and themes.	PALR2-Test02
LR1	System offers interactive tutorials for new players to understand UNO Flip rules.	LR1-Test01
UAFR1	All system notifications should use concise, friendly language without technical jargon.	UAFR1-Test01
AR1	Support features like colorblind modes, text enlargement, and soundless mode for accessibility.	AR1-Test01
SALR1	Average response time should be within 50 milliseconds.	SALR1-Test01
SCR1	Card and game state changes must synchronize in real-time for all	SCR1-Test01

	players.	
SCR1-Test01	Ensure high precision in score calculations and game state.	POAR1-Test01
RAAR1	Guarantee 99% system uptime.	RAAR1-Test01
ROFR1	Allow players to reconnect after network interruption without losing progress.	ROFR1-Test01
CP1	Support at least 1,000 concurrent users, with each game room operating independently.	CP1-Test01
SOER1	System should be scalable for future game modes or AI difficulty levels.	SOER1-Test01

Table 3.Traceability table I

MR1	Code should be well-documented and modular for future updates and fixes.	MR1-Test01
SR1	Offer online technical support for reporting	SR1-Test01

	issues in-game or via the website.	
IR1	Encrypt all player data to prevent unauthorized modifications.	IR1-Test01
PR1	Comply with privacy policies to prevent data breaches.	PR1-Test01
AR1	Log all critical backend operations for auditing and troubleshooting.	AR1-Test01
IR1	System should be immune to common network attacks like DDoS and SQL injection.	IR1-Test02
CR1	Adapt the game for global markets, avoiding cultural sensitivities.	CR1-Test01
PR1	Comply with the laws and regulations of different regions.	PR1-Test01
CR1	Comply with international standards and local data protection laws like GDPR.	CR1-Test02
SR1	Adhere to industry best practices for code	SR1-Test02

	quality and testing.	
HASR1	Avoid elements like flashing lights that could trigger health issues.	HASR1-Test01

Table 4.Traceability table II

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.

5 Appendix

We also create a specific testing plan involves a mix of automated and manual testing to cover functionality, performance, compatibility, and security. Tools like **JUnit**, **Jest**, **Selenium**, **Cypress**, **JMeter**, and **BrowserStack** will help us execute tests efficiently and ensure a high-quality game experience for players. Regularly re-running automated tests, especially after updates, will also support continuous improvement and stability.

Unit Testing

- **Goal**: Test individual functions or methods (e.g., card drawing, shuffling, rule enforcement) to ensure correct outputs.
- **Tool**:
    - **Java**: JUnit for automated unit testing.
    - **JavaScript**: Jest for creating and running unit tests.

Integration Testing

- **Goal**: Ensure different game components (e.g., card management and scoring) work together seamlessly.
- **Tool**:
    - **Java**: JUnit and Mockito to test the integration of different modules.
    - **JavaScript**: Jest combined with Sinon.js for mocking dependencies.

System Testing

- **Goal**: Validate the entire game's functionality, including game flow, player turns, and win/loss conditions.
- **Tool**: Selenium WebDriver for end-to-end testing, simulating user actions to verify complete functionality.

User Interface (UI) Testing

- **Goal**: Test the user interface elements such as buttons, menus, and in-game animations for usability and responsiveness.
- **Tool**:
  - **JavaScript**: Cypress or Puppeteer to automate UI tests and ensure elements behave as expected.

Performance Testing

- **Goal**: Evaluate game performance, including load times, memory usage, and responsiveness during peak usage.
- **Tool**:
  - **Java**: Apache JMeter to simulate multiple users and test performance under load.
  - **JavaScript**: Lighthouse or WebPageTest to analyze load speed and responsiveness.

Compatibility Testing

- **Goal**: Ensure the game runs smoothly across different browsers, devices, and screen resolutions.
- **Tool**: BrowserStack or Sauce Labs for cross-browser and cross-device testing.

Regression Testing

- **Goal**: Verify that new updates or changes do not break existing functionality.
- **Tool**: Selenium WebDriver to automate key test cases and re-run them after each update.

User Experience (UX) Testing

- **Goal**: Collect feedback on game flow, design, and enjoyment to enhance the player experience.
- **Tool**: Conduct manual playtesting sessions with real users and gather feedback via surveys or interviews.

Security Testing

- **Goal**: Identify and mitigate potential security vulnerabilities, ensuring safe user data handling.
- **Tool**: OWASP ZAP for testing vulnerabilities such as data leaks or injection attacks.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

   Mingyang ：We used a shared document platform during the writing process, which helped us achieve real-time collaboration. Each member can view and provide feedback on other members' work at any time. This transparent process allows us to discover and solve some minor problems in a timely manner, avoid duplication of work, and ensure the coherence between each part. Communication between team members is also very smooth. Everyone will actively share the problems they encounter and their own solutions, forming a good atmosphere of cooperation. This positive interaction has greatly helped the smooth completion of the deliverables.

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

   Mingyang ：The main pain point I encountered while writing this deliverable was a lack of clarity on the difference between functional requirements and non-functional requirements. When defining requirements, it is sometimes difficult to distinguish between functional requirements (e.g., specific features and behaviors) and non-functional requirements (e.g., aspects such as performance, reliability, and maintainability). After we received feedback on the previous SRS, I revised the requirements based on the feedback and clearly distinguished between the two types of requirements, which helped me understand and define the requirements more clearly.

To overcome this, I referred to the specific suggestions provided in the feedback and read more about functional and non-functional requirements. This allowed me to better grasp the criteria for distinguishing between the two and clearly present each type of requirement in the deliverable. Although this process was a bit challenging, after the revision, the final document was more in line with the requirements.

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

   Jianhao: The first skill we need to acquire is the static testing for our functional requirements, since our document is written very clearly about what we should expect about each functional requirement test and how to test them (ex. input parameters, etc). The second skill we need to acquire is the dynamic testing for our non-functional requirement. In these tests, we need to analyze the performance and if the performance is not ideal, we need to use dynamic analysis and optimize the code and function to ensure they achieve required performance. Automatic testing is not required if there is not too cumbersome. For each team member, we can let them choose a section to test based on their skills and prior knowledge. We can allocate easier tests to member who don't have much prior knowledge, and heavier or automatic tests to members with a lot of knowledge and experiences.

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

   Jianhao: We had a course in third year about software testing, and I think we can still refer to the course note from that course if we are unsure about how dynamic or static testing works. For automatic testing tools, I think we can look for online tutorials for the setup and how they work. We can also teach and help each other within the team to achieve a better efficiency of the learning since we are doing the same project.