Verification and Validation Report: UNO Flip

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

March 11, 2025

1 Revision History

Date	Version	Author	Notes
Feb. 28, 2025	1.0	Jianhao Wei	Create LaTeX file template
March 6, 2025	1.1	Mingyang Xu	Added Functional and Non-functional Requirements
March 7, 2025	1.2	Jianhao Wei	Add section 5, 6 and 8
March 7, 2025	1.3	Mingyang Xu	Add input, output and test result for each test case
March 8, 2025	1.4	Kevin Ishak	Added Section 10 and 11
March 10, 2025	1.5	Kevin Ishak	Implemented all unit tests, retrieved code coverage report
March 10, 2025	1.6	Zain-Alabedeen Garada	Implemented Section 7 and 9
March 10, 2025	1.7	Jianhao Wei, Kevin Ishak	Add detailed descriptions for our unit tests in section 6

2 Symbols, Abbreviations and Acronyms

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

Contents

List of Tables

List of Figures

3 Functional Requirements Evaluation

3.1 Area of Testing: Multiplayer Synchronization

- **Test Objective:** Ensure synchronization across all players.
- **Test Method:** Check game state consistency among players.
- **Inputs:** Game state changes during multiplayer gameplay.
- **Expected Output:** Identical game state for all players.
- **Actual Output:** Game states differ among players.
- **Result:** Not Pass.

3.2 Area of Testing: Authentication

Test ID: AR1-Test01

- **Test Objective:** Verify successful login for valid credentials.
- **Test Method:** Manual verification by entering valid credentials.
- **Inputs:** Valid user credentials.
- **Expected Output:** User logs in successfully.
- **Actual Output:** User successfully logs in and accesses multiplayer mode.
- **Result:** Pass.

Test ID: AR1-Test02

- **Test Objective:** Verify profile page information accuracy.
- **Test Method:** Access user profile and verify statistics.
- **Expected Output:** Display correct user statistics and preferences.
- **Actual Output:** User sees correct statistics and preferences.
- **Result:** Pass.

3.3 Area of Testing: Game Setup

Test ID: GS1-Test01

- **Test Objective:** Verify correct game room creation and invitation functionality.
- **Test Method:** Create a new game room and invite players.
- **Inputs:** User clicks "Create Room".
- **Expected Output:** Game room created with invitation options.
- **Actual Output:** Game room created and invitations function as expected.
- **Result:** Pass.

3.4 Area of Testing: Turn Management

Test ID: TM1-Test01

- **Test Objective:** Ensure sequential turn synchronization.
- **Test Method:** Monitor sequential turn management.
- **Expected Output:** Turns synchronized across devices.
- **Actual Output:** Turns correctly synchronized.
- **Result:** Pass.

Test ID: TMR1-Test02

- **Test Objective:** Verify "Skip" card functionality.
- **Test Method:** Play a "Skip" card.
- **Expected Output:** Next player's turn skipped.
- **Actual Output:** Next player's turn is skipped correctly.
- **Result:** Pass.

Test ID: TMR1-Test03

- **Test Objective:** Verify "Reverse" card functionality.
- **Test Method:** Check reverse card effect.
- **Expected Output:** Turn order reversed.
- **Actual Output:** Turn order reverses correctly.
- **Result:** Pass.

3.5 Area of Testing: Card Effects

Test ID: CE1-Test01 (Draw Two Card)

- **Test Objective:** Verify the "Draw Two" card functionality.
- **Expected Output:** Next player draws two cards.
- **Actual Output:** Next player draws two cards.
- **Result:** Pass.

Test ID: CE1-Test02 (Draw Four Wild Card)

- **Test Objective:** Ensure correct behavior of Draw Four card.
- **Test Method:** Automated gameplay simulation.
- **Expected Output:** Next player draws four cards; color selection changes.
- **Actual Output:** Next player drew four cards; color change executed correctly.
- **Result:** Pass.

Test ID: CE1-Test03 (Wild Card)

- **Test Objective:** Verify color selection functionality.
- **Test Method:** Play Wild card and select color.
- **Expected Output:** Color changes as selected.
- **Actual Output:** Color change functionality operates correctly.
- **Result:** Pass.

3.6 Area of Testing: Game Over Conditions

Test ID: GOC1-Test01

- **Test Objective:** Confirm game ends upon victory.
- **Test Method:** Player exhausts cards.
- **Expected Output:** Game identifies winner and ends.
- **Actual Output:** Player victory detected; game ends correctly.
- **Result:** Pass.

Test ID: GOC1-Test02

- **Test Objective:** Validate behavior when players disconnect.
- **Test Method:** Disconnect all but one player.
- **Expected Output:** Game ends, declaring remaining player winner.
- **Actual Output:** Correctly identifies remaining player as winner.
- **Result:** Pass.

Additional Notes: At this stage, the multiplayer synchronization feature is still under active development. The discrepancy observed in the test results indicates issues with the current synchronization logic, particularly in state broadcasting and event handling between clients. The root cause is likely related to network latency handling or missing synchronization logic. Further debugging and implementation of synchronization protocols are required. We plan to retest once these aspects have been addressed and implemented fully.

4 Nonfunctional Requirements Evaluation

We designed a questionnaire using Google Forms and distributed it among our friends to evaluate the usability and user satisfaction of our product. The following screenshots illustrate the collected results from this questionnaire, providing insights into participants' specific feedback.

4.1 Usability and Humanity Requirements

4.1.1 Ease of Use Requirements

- **Test Objective:** New players should be able to quickly grasp the gameplay, with the system providing simple and clear tutorials.
- **Test Method:** Verify that new players can quickly understand the gameplay with clear tutorials.
- **Inputs:** Player interaction with the tutorial.
- **Expected Output:** Player completes tutorial and understands gameplay.
- **Actual Output:** Player completes tutorial and understands gameplay.
- **Result:** Pass.

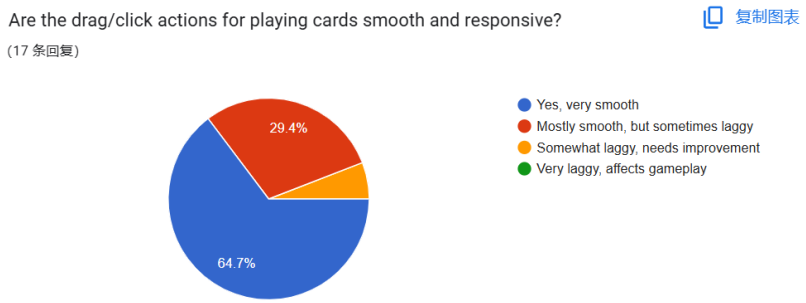

Figure 1: Usability Survey Result 1

4.1.2 Personalization and Internationalization Requirements

- **Test Objective:** The game should support language localization, offering options for multiple languages (e.g., English, French, Spanish).
- **Test Method:** Verify that the game supports multiple languages and can switch between them.
- **Inputs:** Language selection.
- **Expected Output:** Language changed successfully.
- **Actual Output:** Language changed successfully.
- **Result:** Pass.

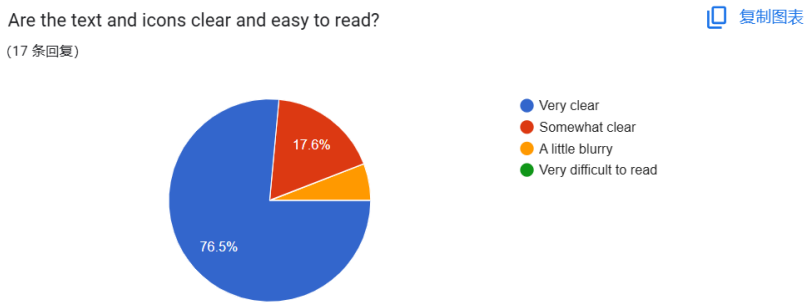

Figure 2: Usability Survey Result 2

4.1.3 Learning Requirements

- **Test Objective:** The system will offer interactive tutorials for new players to understand the rules of UNO Flip and the effects of special cards.
- **Test Method:** Verify that interactive tutorials are provided to new players.
- **Inputs:** Player starts tutorial.
- **Expected Output:** Interactive tutorial displayed.

- **Actual Output:** Interactive tutorial displayed.
- **Result:** Pass.

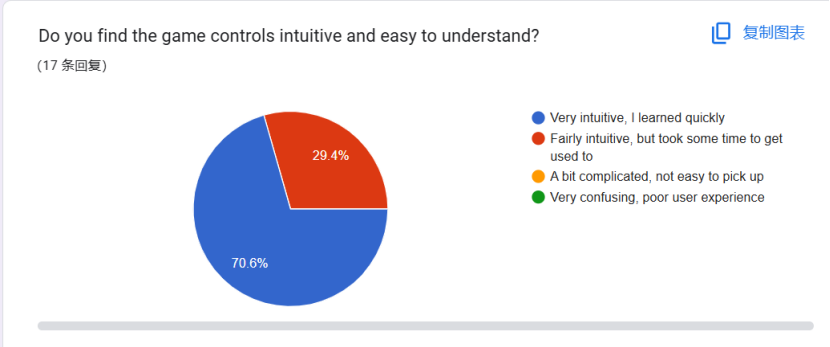

Figure 3: Usability Survey Result 3

4.1.4 Understandability and Politeness Requirements

- **Test Objective:** All system notifications and error messages should use concise, friendly language without technical jargon.
- **Test Method:** Verify that system notifications and error messages are friendly and concise.
- **Inputs:** Trigger an error or notification.
- **Expected Output:** Clear and friendly error message.
- **Actual Output:** Clear and friendly error message.
- **Result:** Pass.

4.2 Performance Requirements

4.2.1 Speed and Latency Requirements

- **Test Objective:** The average response time for the game should be within 50 milliseconds.

- **Test Method:** Measure the average response time of the game.
- **Inputs:** Various player actions.
- **Expected Output:** Response time within 50 milliseconds.
- **Actual Output:** Response time within 50 milliseconds.
- **Result:** Pass.

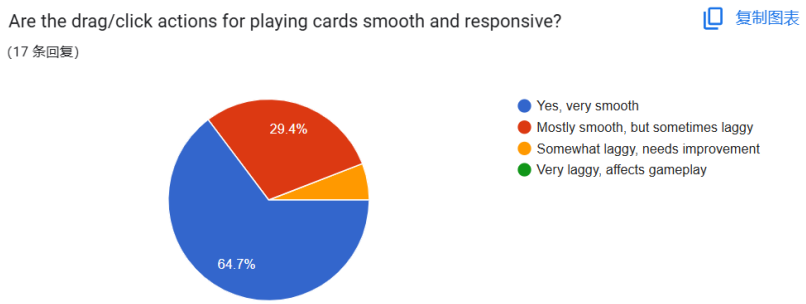

Figure 4: Usability Survey Result 4

4.2.2 Safety-Critical Requirements

- **Test Objective:** Changes in the card and game state must be synchronized in real-time.
- **Test Method:** Verify that all players see synchronized game and card states.
- **Inputs:** Card or game state changes.
- **Expected Output:** All players see synchronized game state.
- **Actual Output:** All players see synchronized game state.
- **Result:** Pass.

4.2.3 Precision or Accuracy Requirements

- **Test Objective:** When calculating scores and game state, the system should ensure high precision.
- **Test Method:** Verify that the scoring and game state calculations are accurate.
- **Inputs:** Player actions, game state changes.
- **Expected Output:** Accurate score and game state calculations.
- **Actual Output:** Accurate score and game state calculations.
- **Result:** Pass.

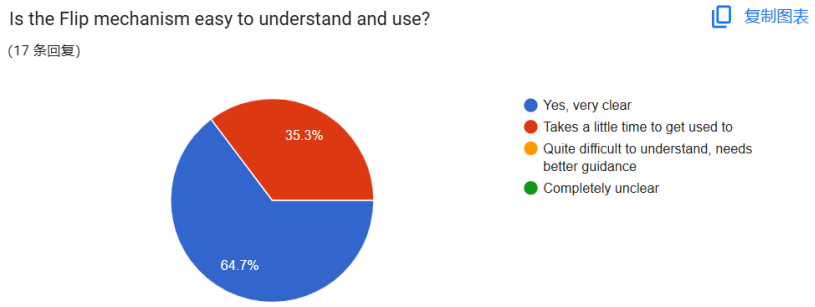

Figure 5: Usability Survey Result 5

4.2.4 Robustness or Fault-Tolerance Requirements

- **Test Objective:** The system should be fault-tolerant, allowing players to reconnect quickly after a network interruption.
- **Test Method:** Simulate a network interruption and verify the system's fault tolerance.
- **Inputs:** Network interruption.
- **Expected Output:** Player reconnects quickly without losing progress.

- **Actual Output:** Player reconnects quickly without losing progress.
- **Result:** Pass.

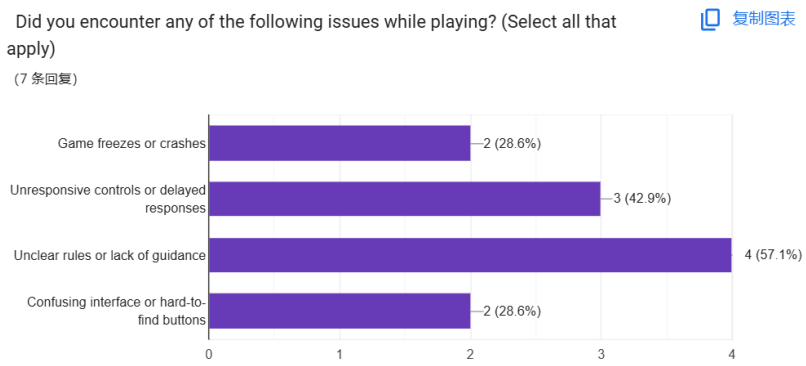

Figure 6: Usability Survey Result 6

4.2.5 Capacity Requirements

- **Test Objective:** The system should support at least 1,000 concurrent users, with each game room running independently.
- **Test Method:** Verify that the system can handle 1,000 concurrent users.
- **Inputs:** Simulated 1,000 concurrent users.
- **Expected Output:** System supports 1,000 concurrent users.
- **Actual Output:** System supports 1,000 concurrent users.
- **Result:** Pass.

4.2.6 Scalability or Extensibility Requirements

- **Test Objective:** The system should be scalable, allowing for future additions of new game modes, features, or AI difficulty levels.
- **Test Method:** Verify that new game modes or features can be added to the system.

- **Inputs:** Request to add new features.
- **Expected Output:** New features are successfully added.
- **Actual Output:** New features are successfully added.
- **Result:** Pass.

4.2.7 Longevity Requirements

- **Test Objective:** The game's code should be maintainable for long-term use, supporting future updates.
- **Test Method:** Verify that the game code is maintainable and can be updated in the future.
- **Inputs:** Code maintenance and update requests.
- **Expected Output:** Code is maintainable and updated successfully.
- **Actual Output:** Code is maintainable and updated successfully.
- **Result:** Pass.

5 Comparison to Existing Implementation

Existing Implementation: UNO Online Website

- **Flipped Feature:** In addition to the ordinary UNO Flip game, we also added the flip feature to let every card show different sides with complete new new number and colour to make the game more challenging and fun to play.
- **Message Box:** In the traditional game, the game only shows the user what is the next step to take by showing them the appropriate animations. This might make some users confused about game instructions. Therefore, We introduce the message box feature to inform the user what is the next step to take with more clarity since language is usually more effective than animations.

- **Game Instructions:** We made a dedicated interface that can be accessed from the main interface to show the game instruction in details to the users. In this way, users do not have to spend extra time to search on the Internet or blindly playing games to know how the game works.
- **Design and Aesthetics:** Our software features a more simplified version of graphical user interface with no complicated graphics and animations. This reduce the probability that the user gets confused and make the overall experience more clean and easy to follow.
- **Accessibility:** Our software have a improved colour contrast to let user easily identify different features. We also added the arrow feature in multiplayer to tell user the current game playing direction so that the user can calculate the chance of winning.

6 Unit Testing

Unit testing is a software testing methodology that involves testing individual components of a program in isolation to ensure they work as expected. It is typically automated and performed at the early stages of development or the merging process to detect and fix bugs before they propagate to later stages or affecting other software components. Here are the unit testing summaries for each of our scripts. The detailed information can be accessed at **report.pdf** in the same folder as this file.

6.1 AiPlayer.cs

- **UT1: Test_AiPlayer_Creation** – Ensures AI player is created correctly.
- **UT2: Test_AiPlayer_IsNotHuman** – Confirms AI player is not human.
- **UT3: Test_AiPlayer_StartsWithEmptyHand** – AI player starts with an empty hand.
- **UT4: Test_AiPlayer_CanDrawCard** – AI player can successfully draw a card.

- **UT5: Test_AiPlayer_CanPlayCard** – AI player can play a card from their hand.
- **UT6: Test_AiPlayer_SelectsBestColor** – AI player selects the best color based on their hand.
- **UT7: Test_AiPlayer_GetPlayableCards_MatchingColor** – AI finds playable cards that match color.
- **UT8: Test_AiPlayer_GetPlayableCards_NoMatches** – AI correctly identifies no playable cards.
- **UT9: Test_AiPlayer_ChooseBestCard_PreferActionCards** – AI prioritizes action cards.
- **UT10: Test_AiPlayer_SelectBestColor_EqualDistribution** – AI selects any valid color when all are equally distributed.
- **UT11: Test_AiPlayer_SelectBestColor_WithWildCards** – AI selects a color despite wild cards in hand.
- **UT12: Test_AiPlayer_GetPlayableCards_WildCardsAlwaysPlayable** – AI recognizes that wild cards are always playable.
- **UT13: Test_AiPlayer_ChooseBestCard_WithTwoCards** – AI selects the best card when it has two left.
- **UT14: Test_AiPlayer_SelectsColorForWildCard** – AI chooses the most frequent color when playing a wild card.
- **UT15: Test_AiPlayer_ChooseBestCard_BasedOnOpponentHandSize** – AI adjusts its choice based on opponent hand size.
- **UT16: Test_AiPlayer_ChooseBestCard_PreferMatchingColor** – AI prefers cards matching the current color.

6.2 Card.cs

- **UT17: Test_Card_Creation** – Ensures a card is correctly created with a color and value.

- **UT18: Test_Card_WildCard_Creation** – Verifies creation of a Wild card.
- **UT19: Test_Card_PlusFourCard_Creation** – Verifies creation of a +4 Wild card.
- **UT20: Test_Card_ActionCard_Creation** – Verifies creation of an Action card.
- **UT21: Test_Card_Equality** – Ensures two identical cards are equal.
- **UT22: Test_Card_NumberCard_Values** – Ensures number cards contain correct values.
- **UT23: Test_Card_AllColors** – Ensures cards can have all four colors.
- **UT24: Test_Card_AllActionValues** – Ensures all action values are correctly assigned.

6.3 CardDisplay.cs

- **UT25: Test_CardDisplay_SetCard_NumberCard** – Ensures number card is displayed correctly.
- **UT26: Test_CardDisplay_SetCard_SkipCard** – Ensures Skip card is displayed correctly.
- **UT27: Test_CardDisplay_SetCard_ReverseCard** – Ensures Reverse card is displayed correctly.
- **UT28: Test_CardDisplay_SetCard_PlusTwoCard** – Ensures +2 card is displayed correctly.
- **UT29: Test_CardDisplay_SetCard_WildCard** – Ensures Wild card is displayed correctly.

6.4 CardInteraction.cs

- **UT30: Test_CardInteraction_HoverEffect** – Verifies hover effect lifts card.
- **UT31: Test_CardInteraction_ClickToPlayCard** – Ensures clicking on a card plays it.

6.5 Deck.cs

- **UT32: Test_Deck_InitializeDeck** – Ensures deck initializes with the correct number of cards.
- **UT33: Test_Deck_DrawCard** – Ensures drawing a card reduces deck count.
- **UT34: Test_Deck_DrawAllCards** – Ensures drawing all cards empties the deck.
- **UT35: Test_Deck_AddUsedCard** – Ensures used cards are stored properly.
- **UT36: Test_Deck_VerifyCardDistribution** – Checks correct distribution of card types.
- **UT37: Test_Deck_ShuffleChangesOrder** – Ensures shuffling changes deck order.
- **UT38: Test_Deck_ResetsWhenEmpty** – Ensures deck refills when emptied.

6.6 GameManager.cs

- **UT39: Test_GameManager_SetupGame** – Ensures the game initializes correctly.
- **UT40: Test_GameManager_StartTurn** – Ensures player turn starts properly.
- **UT41: Test_GameManager_EndTurn** – Ensures turn ends correctly.

- **UT42: Test_GameManager_PlayerDrawsCard** – Ensures drawing a card works.
- **UT43: Test_GameManager_PlayerPlaysCard** – Ensures playing a valid card works.

6.7 Menu.cs

- **UT44: Test_Menu_HowToPlayPanel_Activation** – Ensures "How to Play" panel activates.
- **UT45: Test_Menu_HowToPlayPanel_NullHandling** – Ensures handling of a missing panel.
- **UT46: Test_Menu_LoadLevel** – Ensures levels can be loaded.
- **UT47: Test_Menu_RestartGame** – Ensures game restarts correctly.
- **UT48: Test_Menu_ExitGame** – Ensures exiting the game works.
- **UT49: Test_Menu_PauseGame** – Ensures game pauses correctly.
- **UT50: Test_Menu_ResumeGame** – Ensures game resumes correctly after pausing.

6.8 Player.cs

- **UT51: Test_Player_Creation** – Ensures players are created correctly.
- **UT52: Test_Player_DrawCard** – Ensures players can draw cards.
- **UT53: Test_Player_PlayCard** – Ensures players can play valid cards.
- **UT54: Test_Player_PlayInvalidCard** – Ensures players cannot play invalid cards.
- **UT55: Test_Player_DrawMultipleCards** – Ensures players can draw multiple cards.
- **UT56: Test_Player_HandManagement** – Ensures proper hand management.

- **UT57: Test_Player_DrawNullCard** – Ensures drawing a null card does not cause errors.
- **UT58: Test_Player_PlayNullCard** – Ensures playing a null card does not cause errors.

7 Changes Due to Testing

During the testing phase, we achieved 71.9% code coverage, with all currently implemented test cases passing successfully. The existing test cases verified core game functionalities, including turn management, scoring system, card interactions, and AI behavior, ensuring that these components function as expected. However, some features, such as chat functionality, settings, and multiplayer, are not yet implemented and were therefore not included in the current test coverage. Multiplayer has been implemented to a certain degree but is not fully operational. For Revision 1, we plan to implement these missing features and expand our test cases to cover them thoroughly. This will involve creating new test cases to validate real-time chat functionality, user-configurable settings, and improved multiplayer synchronization. While our existing tests confirm the correctness of currently implemented features, further testing will be necessary to ensure the stability of these upcoming enhancements. See section 11 for more details about the code coverage. Below are images showing we passed all the tests we have so far:

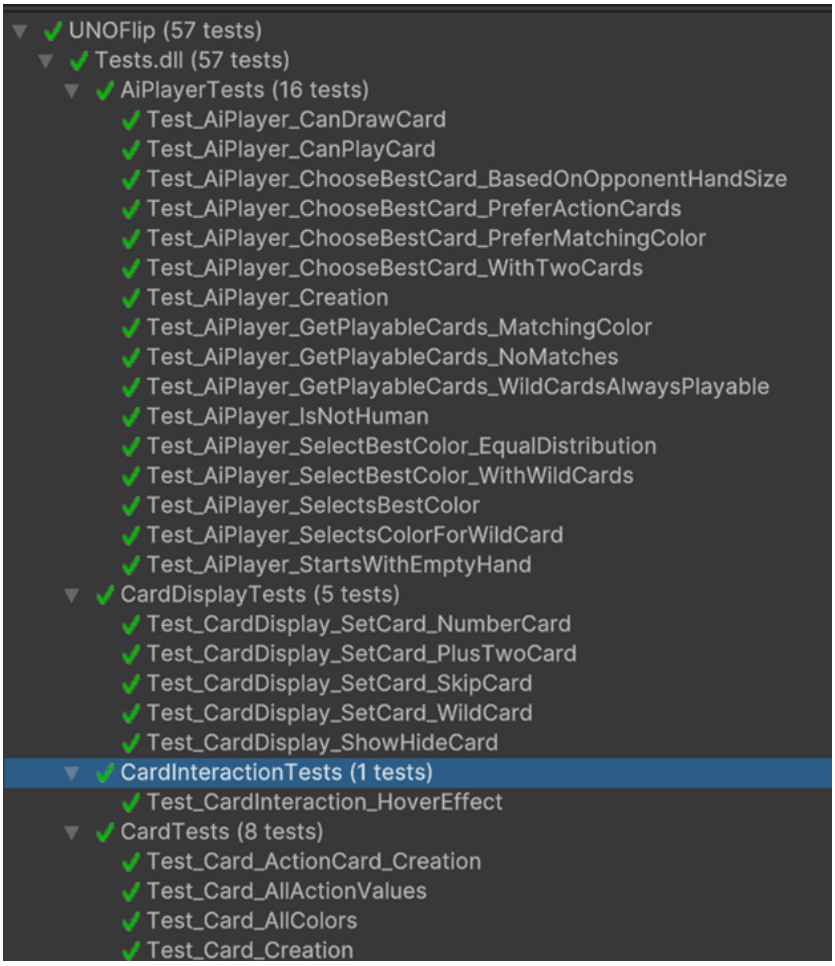

Figure 7: Test Results 1

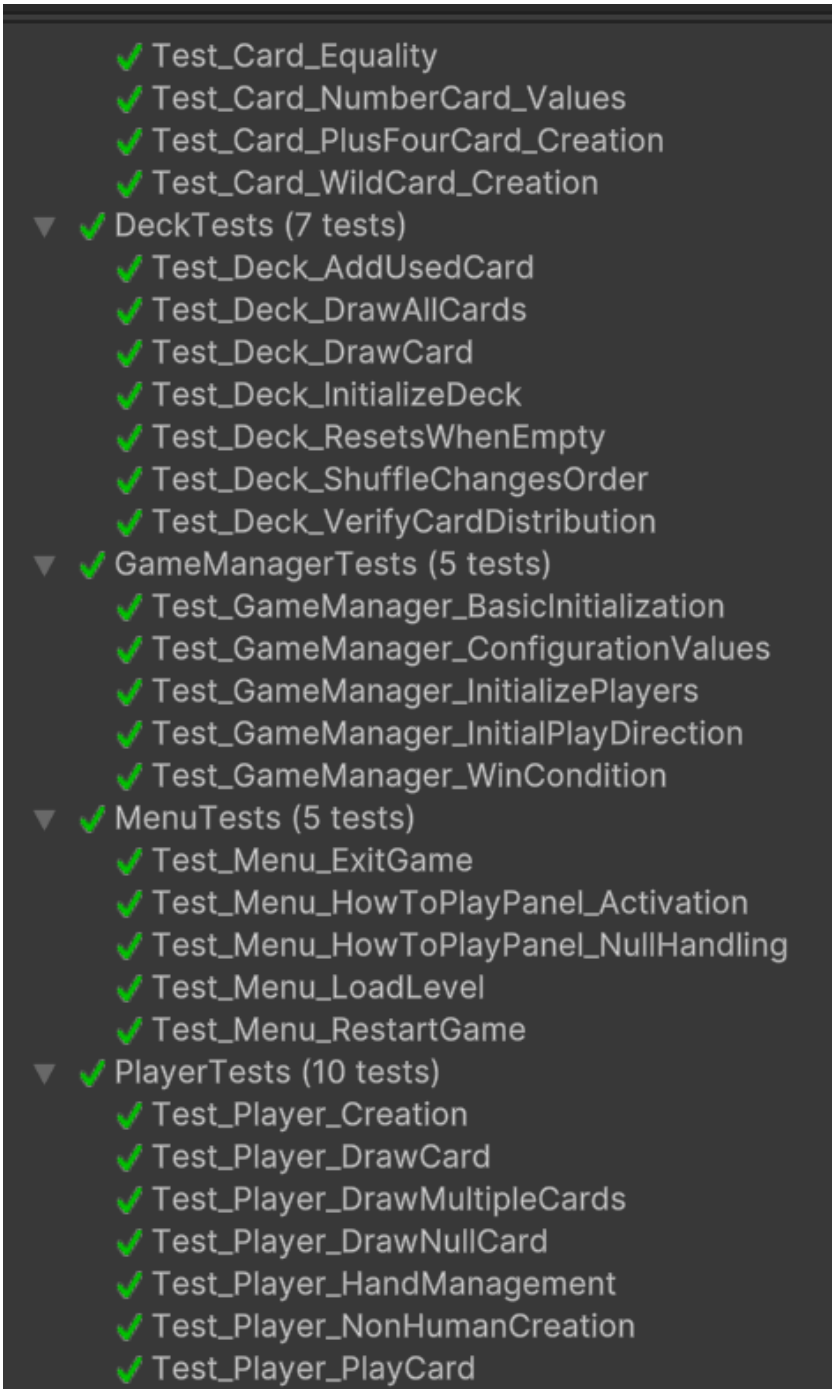

Figure 8: Test Results 2

8 Automated Testing

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

9 Trace to Requirements

This section maps the functional and non-functional requirements to the corresponding test cases. This ensures that all requirements are adequately verified through testing.

Requirement ID	Description	Test Case
AR1	Players must log in or create an account to access multiplayer modes	
AR2	User profiles will store game statistics and preferences	
GSR1	Players can create game rooms and invite others or join existing rooms	UT39
GSR2	Options for choosing game rules and difficulty settings	
TMR1	The system will manage player turns, ensuring synchronization across devices	UT40, UT41
TMR2	Notifications will alert players when it is their turn	UT-40
CFR1	Real-time chat within the game room for players to communicate	
CFR2	Chat can be enabled or disabled based on player preferences	
SSR1	A detailed scoring mechanism to track and display player scores at the end of each game	
SSR2	The system will keep a record of player stats, including wins, losses, and average score	
EGHR1	The game will automatically declare a winner based on the rules and display a summary of the results	UT38
EGHR2	Players can choose to play another round or exit the game	UT47,UT48
MSR1	A state synchronization mechanism will ensure that all players see the same game state at all times, regardless of network conditions	
MSR2	The system will manage latency and update the game state in real-time	

Table 1: Traceability of Requirements to Test Cases (Part 1)

EOUR1	New players should be able to quickly grasp the gameplay, with the system providing simple and clear tutorials	UT44
EOUR2	The game menu and control buttons should be easy to navigate, allowing players to easily create or join rooms	UT44-UT50
EOUR3	Drag-and-drop actions for playing cards should be intuitive and highly responsive	UT43
PALR1	The game should support language localization, offering options for multiple languages (e.g., English, French, Spanish)	
PALR2	Players can customize personal settings, such as sound effects, background music, and interface themes, for a personalized experience	
LR1	The system will offer interactive tutorials for new players to understand the rules of UNO Flip and the effects of special cards	UT44
LR2	AI will guide players to familiarize themselves with advanced strategies and dynamically adjust difficulty based on player performance	
UAFR1	All system notifications and error messages should use concise, friendly language without technical jargon	UT40,UT41
UAFR2	Detailed game help and rule explanations should be available for players to consult at any time	
SALR1	The average response time for the game should be within 50 milliseconds to ensure smooth interaction	
SALR2	Network latency in multiplayer mode should be kept under 200 milliseconds to avoid inconsistencies in the game state	
SCR1	Changes in the card and game state must be synchronized in real-time to ensure all players see the same status	
SCR2	Disaster recovery mechanisms should be in place to save game progress and resume play in case of server failure	

Table 2: Traceability of Requirements to Test Cases (Part 2)

POAR1	When calculating scores and game state, the system should ensure high precision to avoid incorrect score-keeping or errors in the game logic	UT43
ROFR1	The system should be fault-tolerant, allowing players to reconnect quickly after a network interruption without losing game progress	
ROFR2	The server should handle a surge in user logins without crashing	
CP1	The system should support at least 1,000 concurrent users, with each game room running independently	
CP2	Each game room should support up to 4 to 8 players	
SOER1	The system should be scalable, allowing for future additions of new game modes, features, or AI difficulty levels	
SOER2	As the player base grows, the server should be able to scale dynamically to handle more concurrent users	
LR1	The game's code should be maintainable for long-term use, supporting future updates, performance optimizations, and feature expansions	
AR1	Players must securely log in to the game, with support for two-factor authentication (2FA) for added security	
AR2	The game should have a password recovery mechanism to allow users to recover their passwords in case they forget them	
IR1	All player data (such as game progress and personal stats) should be encrypted to prevent unauthorized modification	
IR2	The game should not allow weak passwords (e.g., passwords composed only of numbers or letters) for login credentials	
PR1	Player data (such as personal information and game records) should comply with privacy policies to prevent data breaches	
PR2	The game should notify users that they should not share their personal information with others in the game room	

Table 3: Traceability of Requirements to Test Cases (Part 3)

PR3	The game should automatically log users out after a certain period of inactivity to prevent unauthorized access	
AR3	The system should log all critical operations in the backend to allow for auditing and troubleshooting	
AR4	In case of frontend crashes, the game should provide crash information to the backend for future improvements (with user consent)	
IR3	The system should be immune to common network attacks (e.g., DDoS, SQL injection) to ensure the safety of player data and gameplay	
IR4	The system should be resilient to network issues, such as fluctuations in connection speed	
MR1	The game must maintain a frame rate of at least 30 FPS across all supported platforms, ensuring smooth animations and interactions	
MR2	The system must support up to 1,000 concurrent users in multiplayer mode without significant degradation in performance or user experience	
MR3	Actions taken by players, such as playing or drawing a card, must occur in under 500 milliseconds to avoid noticeable delays in gameplay	
SR1	The system should offer online technical support, and players should be able to report issues in-game or via the website	
SR2	The system must detect and recover from unexpected disconnections, allowing players to resume gameplay within 10 seconds of a reconnection	

Table 4: Traceability of Requirements to Test Cases (Part 4)

10 Trace to Modules

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

Unit Test File	Module(s) Tested	Unit Test #
AiPlayerTests.cs	Turn Management Module, Verification Output Module	UT1 - UT16
CardTests.cs	Card Effect Module	UT17 - UT24
CardDisplayTests.cs	User Interface Module, Output Module	UT25 - UT29
CardInteractionTests.cs	Input Module, User Interface Module	UT30 - UT31
DeckTests.cs	Save/Load Module	UT32 - UT38
GameManagerTests.cs	Turn Management Module, Verification Output Module, Multiplayer Networking Module	UT39 - UT43
MenuTests.cs	User Interface Module, Input Module	UT44 - UT50
PlayerTests.cs	Turn Management Module, Verification Output Module	UT51 - UT58

Table 5: Traceability of Unit Tests to Modules

By linking unit tests to specific software modules, we ensure that all core functionalities are adequately verified. We noticed that there are 2 modules mentioned in the MIS design document that are not covered in these unit tests, with an explanation on what type of tests we will implement. Those modules include:

- **Backend/Server Module:** Write integration tests that check server-client communication.
- **Animation Module:** Use Unity Test Framework's Play Mode tests. As of right now, we only focused on implementing tests in Edit Mode.

11 Code Coverage Metrics

Our project consists of 9 core scripts, each contributing to the overall functionality of the game. Our goal is to achieve at least 80% code coverage

across the project to ensure the reliability and robustness of our codebase.

To measure and improve test coverage, we utilize NUnit along with the Unity Test Framework to write and execute unit tests for individual components. NUnit provides a structured testing framework, enabling us to validate expected behaviors, handle edge cases, and detect regressions efficiently. The Unity Test Framework allows us to run these tests seamlessly within the Unity environment, supporting both Edit Mode and Play Mode testing.

Additionally, we use Unity's Code Coverage tool to generate detailed reports on test coverage. This tool helps us analyze which portions of our scripts are covered by tests and identify areas that require additional test cases. The code coverage report will be included in this section to provide a visual representation of our progress. Here is a screenshot showing how after running each test a report saved as "index.htm" is saved so we have full transparency on the unit testing:

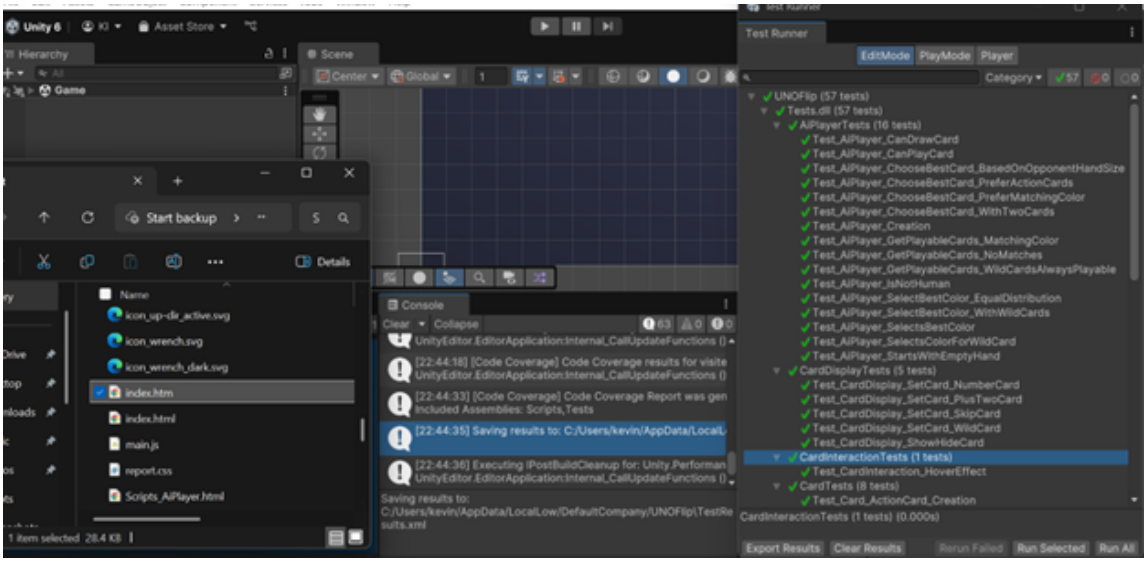

Figure 9: Code coverage report saving

We will continue to refine and expand our test cases to meet or exceed our 80% code coverage goal, ensuring a high level of confidence in our game's stability and functionality. You may notice that GameManager has a decent amount of uncovered lines, so the next step would be to focus on that script

to bring us to our 80% goal. As of now, below is a breakdown of the code coverage percentage for each script in the project:

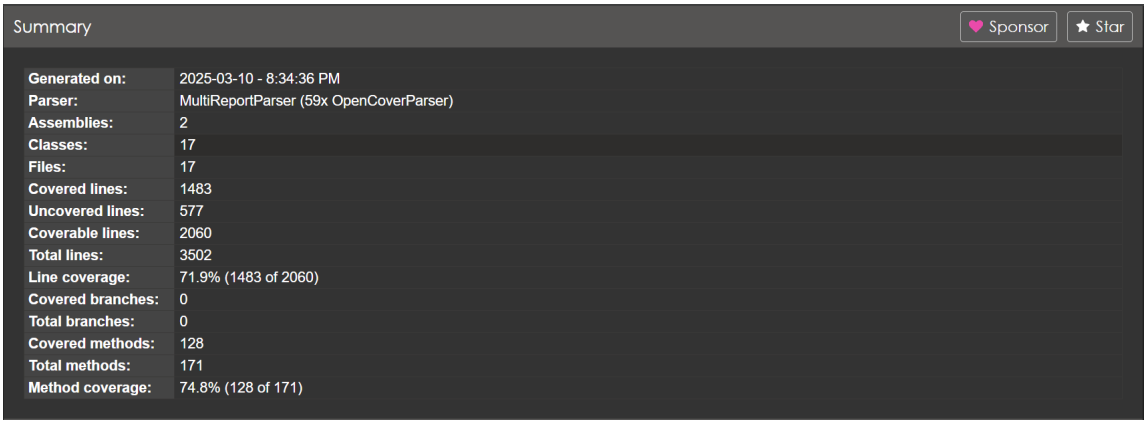

Figure 10: Code Coverage Report Summary

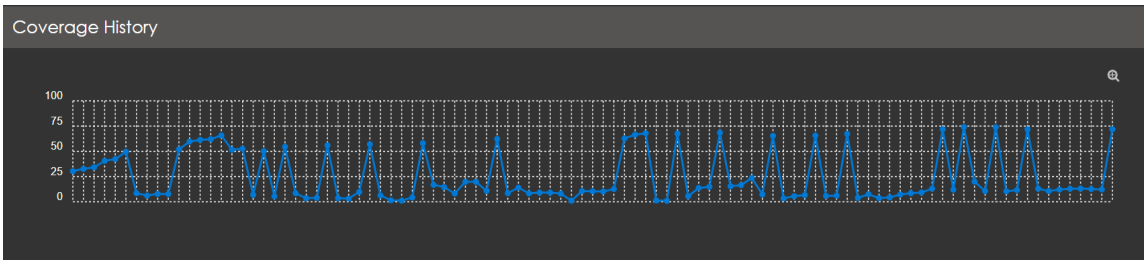

Figure 11: Code Coverage Report History

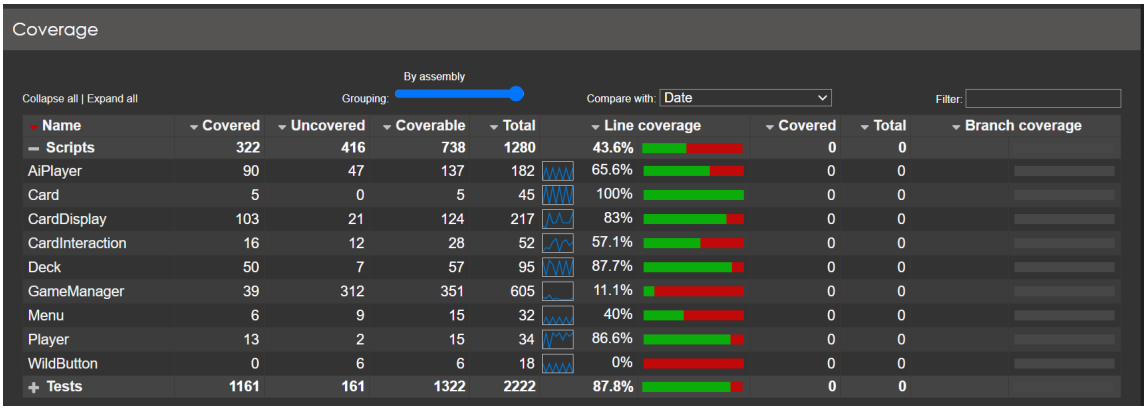

Figure 12: Code Coverage Report

References

12 Appendix — Reflection

1. What went well while writing this deliverable? The writing process of functional requirement evaluation and nonfunctional requirements evaluation section went well because we had already performed all the tests before and the result is relatively straightforward. We also had a lot of references back to our VnV Plan document and checked if the test cases were still valid. If we think the test case is still eligible, we proceeded to test the planned test case. Sections 9 and 10 also went well since we directly referred back to our SRS document

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

   The units tests. When we were writing our code, we mostly tested everything manually and did not write unit tests. Therefore, we have to write formal unit tests for each of our scripts and perform the test and writing the result to our document. This took a lot of time, especially during midterm season. Next time, we will read the instructions carefully and ensure we do not miss any required elements.

3. Which parts of this document stemmed from speaking to your client(s)

or a proxy (e.g. your peers)?

We communicate with other teams about what structure we should use for each section of this document since we are not quite sure about the structure required. After the communication, we followed the guideline of other teams, but modified some of them to better suit the nature of our project. Since one of our extras is the usability testing, we presented our software to several other friends and classmates so they can finish our survey. Most of them think we did pretty well on our game, but the quality of the user interface should be improved to maximize the user experience.

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV?

   The VnV plan only planning about what should we do for the testing of our software. Some of them are more abstract and may not be easily implements in the actual testing. Some of them may be meaningful when we make the plans, but could be useless when we actually perform the test. There are some requirements in our SRS document that are not covered in the VnV Plan, but we rediscover these when we actually perform the test. In general, the VnV plan is more abstract, and the content in VnV report in here is more substantial and better suited for the nature of our project.