SOFTWARE TESTING REPORT

contest

1.1 Explanation of the purpose and methods of software testing.

Software testing is a crucial process in the software development lifecycle that aims to identify defects, errors, or bugs in a software application to ensure it meets the specified requirements and functions correctly. The primary purpose of software testing is to improve the quality, reliability, and performance of the software by finding and fixing issues before the application is deployed to users.

Here are the key purposes and methods of software testing:

Purpose of Software Testing:
1. Identifying Defects: Testing helps in identifying defects or bugs in the software, which can impact its functionality, user experience, and overall performance.

2. Validating Requirements: Testing ensures that the software meets the specified requirements and behaves as expected by stakeholders.

3. Improving Quality: By identifying and fixing defects early in the development process, testing helps in improving the quality and reliability of the software.

4. Enhancing User Experience: Testing helps in ensuring that the software provides a seamless and user-friendly experience to its end-users.

5. Reducing Risks: Testing helps in reducing risks associated with software failures, security vulnerabilities, and performance issues.

Methods of Software Testing:
1. Manual Testing: Manual testing involves human testers executing test cases without using any automation tools. Testers interact with the software application as end-users to identify defects and validate functionality.

2. Automated Testing: Automated testing involves using tools and scripts to automate the execution of test cases. Automated tests can be run repeatedly to check for regressions and save time compared to manual testing.

3. Unit Testing: Unit testing focuses on testing individual units or components of the software in isolation to ensure they work correctly. Developers typically perform unit testing during the coding phase.

4. Integration Testing: Integration testing verifies that different modules or components of the software work together seamlessly when integrated. It tests the interactions between various parts of the system.

5. System Testing: System testing tests the entire software application as a whole to validate its compliance with requirements, functionality, performance, and security.

6. Regression Testing: Regression testing involves retesting the software after changes or updates to ensure that new code modifications have not adversely affected existing functionalities.

7. Performance Testing: Performance testing evaluates the responsiveness, stability, scalability, and speed of the software under different load conditions to ensure it performs optimally.

8. Security Testing: Security testing assesses the software's ability to protect data and resources from unauthorized access, vulnerabilities, and potential threats.

By employing a combination of these testing methods and techniques, software development teams can ensure that their applications are thoroughly tested, reliable, and deliver a high-quality user experience.

There are various types of manual testing, including white-box testing, black-box testing, and gray-box testing.

Let's use a card game as an example to explain these types of manual testing:

WHITE-BOX TESTING	BLACK-BOX TESTING	GRAY-BOX TESTING
The tester has access to the internal structure and code of the software application being tested. In the context of a card game, a white-box tester would examine the game's code and logic to understand how the cards are shuffled, dealt, and played. The tester would design test cases based on this knowledge to ensure that the game functions correctly according to the rules and logic defined in the code.	The tester does not have access to the internal code or structure of the software application and focuses on testing its functionality. - In the context of a card game, a black-box tester would test the game without knowing how it is implemented internally. The tester would interact with the game as an end-user, playing different scenarios and verifying that the game behaves as expected.	Gray-box testing combines elements of both white-box and black-box testing, where the tester has partial knowledge of the internal code and structure of the software application. - In the context of a card game, a gray-box tester might have some understanding of how the game is implemented but not full access to all details. The tester would use this partial knowledge to design test cases that cover both functional aspects (black-box) and internal logic (white-box) of the game.
For example, a white-box tester might test if the cards are shuffled properly before dealing them to players, ensuring that the game follows a specific algorithm for shuffling.	For example, a black-box tester might test if the game accurately calculates scores, follows the rules of the game, and handles player inputs correctly.	For example, a gray-box tester might test if certain game features are implemented correctly based on their partial knowledge of the code while also verifying user interactions and overall functionality.

By applying white-box, black-box, and gray-box testing techniques to a card game example, testers will ensure thorough testing coverage to identify defects, validate functionality, and improve overall quality before the game is released to users.

Functional testing for object-oriented programs typically includes the following types of testing:

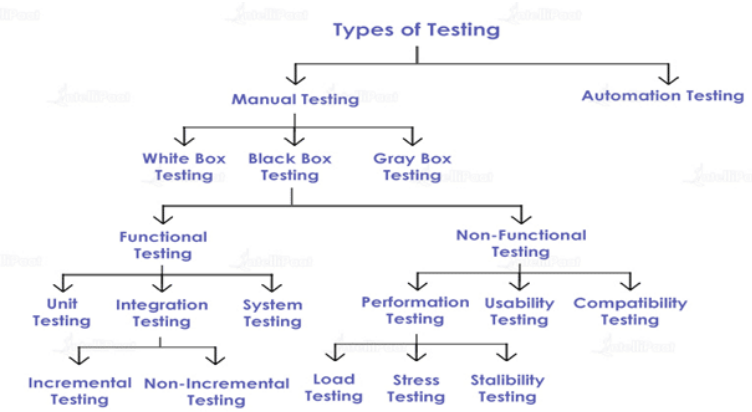
1. **Unit** Testing: Testing individual classes and methods to ensure they work as expected.
2. **Integration** Testing: Testing the interactions between different classes and components to ensure they work together correctly.
3. **System** Testing: Testing the entire system to verify that it meets the specified requirements.
4. **Acceptance** Testing: Testing the system with real users to ensure it meets their needs and expectations.

Non-functional testing for object-oriented programs may include the following types of testing:

1. Performance Testing: Testing the program's speed, scalability, and resource usage under different conditions.
2. Security Testing: Testing the program for vulnerabilities and ensuring data protection measures are in place.
3. Usability Testing: Testing the program's user interface and overall user experience.
4. Compatibility Testing: Testing the program on different platforms, browsers, and devices to ensure it works as intended.
5. Reliability Testing: Testing the program's stability and ability to recover from failures.
6. Maintainability Testing: Testing the program's ease of maintenance and ability to accommodate future changes.
7. Compliance Testing: Ensuring the program complies with relevant standards, regulations, and best practices.

By conducting a combination of these functional and non-functional testing types, developers can ensure that an object-oriented program meets its requirements and performs effectively in various scenarios.

1.2 Describe the different stages and types of software testing.



Software testing is a crucial process in the software development lifecycle that helps identify defects and ensure the quality and reliability of the software. The different stages and types of software testing include:

	STAGE	TYPE
Unit Testing	This is the first level of testing where individual units or components of the software are tested in isolation.	Developers write test cases to verify that each unit works as expected and meets its design specifications.
Integration Testing	In this stage, multiple units or components are combined and tested together to check their interactions and interfaces.	Different integration testing approaches include top-down, bottom-up, and sandwich testing to ensure that the integrated components work correctly
System Testing	The entire system is tested as a whole to verify that it meets the specified requirements.	Functional testing, non-functional testing, regression testing, and user acceptance testing are common types of system testing.
Acceptance Testing	This stage involves testing the software with real users to ensure it meets their needs and expectations	Alpha testing, beta testing, and usability testing are common types of acceptance testing.
Performance Testing	This type of testing evaluates the software's speed, scalability, and stability under various conditions.	Load testing, stress testing, and scalability testing are examples of performance testing.
Security Testing	Security testing focuses on identifying vulnerabilities in the software and ensuring data protection measures are in place	Type: Penetration testing, vulnerability scanning, and security auditing are common types of security testing
Regression Testing	This stage involves retesting the software after changes to ensure that existing functionalities have not been affected.	Type: Automated regression testing and manual regression testing are commonly used methods.
User Acceptance Testing (UAT)	UAT is conducted by end-users to validate the software's functionality against business requirements	Alpha testing (in-house) and beta testing (external) are common types of UAT.

1.3 Explain how automation is used in software testing.

Automation is widely used in software testing to improve efficiency, reduce human error, and accelerate the testing process. Here's how automation is typically used in software testing:

1. Test Case Execution:

Automation tools are used to execute predefined test cases without human intervention. This includes running regression tests, smoke tests, and other repetitive test cases to ensure that the software behaves as expected after changes or updates.

2. Data Generation:

Automation can be used to generate test data for various scenarios, including positive and negative test cases. This ensures that a wide range of inputs and conditions are tested without the need for manual data entry.

3. Test Result Analysis:

Automation tools can analyze test results and compare them with expected outcomes. This helps in identifying discrepancies and automatically flagging failed test cases for further investigation.

4. Integration Testing:

Automation is used to perform integration testing by simulating interactions between different modules or components of the software. This helps in identifying integration issues early in the development cycle.

5. Load and Performance Testing:

Automation tools are used to simulate large numbers of users or transactions to assess the performance and scalability of the software under different load conditions.

6. Continuous Integration/Continuous Deployment (CI/CD):

Automation is an integral part of CI/CD pipelines, where automated tests are triggered after each code change to ensure that new features or bug fixes do not introduce regressions.

7. Cross-platform Testing:

Automation tools can be used to run tests on different platforms and devices, ensuring that the software functions correctly across a variety of environments.

8. Maintenance Testing:

Automation is used to re-run tests after software updates or changes to verify that existing functionality has not been impacted.

By leveraging automation in these ways, software testing teams can increase test coverage, reduce testing time, and improve overall software quality. However, it's important to note that while automation can greatly enhance testing efficiency, it should be complemented with manual testing to cover aspects that cannot be automated, such as usability, user experience, and exploratory testing.

1.4 Describe functional and structural testing.

	Functional Testing	Structural Testing / white-box testing
Definition	A type of software testing that verifies that the software functions according to the specified requirements and meets the user's expectations.	Also known as white-box testing or code-based testing, is a type of software testing that examines the internal structure of the software code.
Purpose	The main goal of functional testing is to ensure that the software performs the functions it is supposed to perform correctly	The main goal of structural testing is to verify the correctness and completeness of the software code by analyzing its structure, logic, and implementation.
Focus	On the external behavior of the software without considering its internal structure or code implementation.	On evaluating the code paths, branches, statements, and conditions to identify potential defects or errors in the code.
Types	Different types of functional testing include unit testing, integration testing, system testing, acceptance testing, and regression testing.	Different types of structural testing include statement coverage, branch coverage, path coverage, condition coverage, and mutation testing.
Techniques	Functional testing techniques include black-box testing, white-box testing, gray-box testing, equivalence partitioning, boundary value analysis, and decision table testing.	Structural testing techniques include code walkthroughs, code inspections, static analysis, dynamic analysis, and code coverage tools.

In summary, functional testing assesses the software's external behavior and functionality based on user requirements, while structural testing examines the internal code structure and logic to ensure its correctness and completeness. Both types of testing are essential for ensuring the quality and reliability of software products.

2.1 Design appropriate test data.

For generating testing data, we can use different methods and tools such as manual input, automated scripts, data extraction, data masking, data anonymization, etc. Let's describe how each of these methods can be used to generate test data for the card game example:

1. Manual Input:

- Testers can manually input player names, card colors, and numbers to create specific test scenarios. For example, testers can create test cases where players have specific combinations of cards to test different game outcomes.

2. Automated Scripts:

- Test automation tools can be used to generate randomized test data for the game. Scripts can be created to automatically generate player names, card colors, and numbers in various combinations to simulate different game scenarios efficiently.

3. Data Extraction:

- Existing data from previous game plays can be extracted and used as test data. For example, data on player scores, card combinations, and game outcomes from previous game sessions can be extracted and reused to create new test cases.

4. Data Masking:

- In cases where sensitive or personal information is involved, data masking techniques can be applied to anonymize the data. For the card game, player names or any other sensitive information can be masked to ensure privacy while still providing realistic test data.

5. Data Anonymization:

- Data anonymization techniques can be used to replace real player names with anonymized identifiers. This ensures that the test data remains realistic without exposing personal information. Anonymized player names can still be used to simulate different player interactions in the game.

By using a combination of these methods and tools, testers can generate a diverse set of test data for the card game that covers various scenarios and ensures thorough testing of the game's functionality.

As a tester I've chosen the method that suits me most - **manual input**. As far as we do not have a lot of input information it is an easier way to check this game. In requirements we do not have a mention about how long should be name and how strong should be password, just requirements it must be in list (file). So, it would be

good to set same requirements about username, for example Username no longer than 12 characters and password longer than 3 characters.

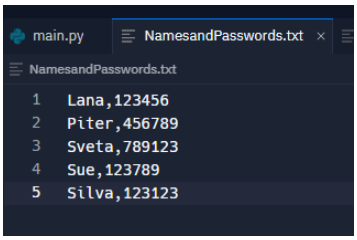
Here are some appropriate test data for the LOUISE'S CARD GAME:

Player Authorization:

Test Case 1: Verify that an unauthorized player cannot access the game.
Test Case 2: Verify that an authorized player can access the game.

Type of Data	Username	Password
Normal data	Lana Piter Sveta Sue Silva	123456 456789 789123 123789 123123
Invalid data (unacceptable username not on the list)	&@@@!	777777777
Enter username/password which contains spaces	Lana Lana	789 789
Boundary inputs Non-acceptable username		
Non-acceptable username (Username / password 1 character)	L	1
Non-acceptable username (Username No longer then 12) (password les then 4)	Usernametest123	789
Empty username		
Erroneous inputs (Acceptable username not on the list)	Theo Maggie	Qw787 Ere7879

In my case I have a file with this list of names



Deck Initialization:

- Test Case 3: Verify that the deck contains 30 unique cards.
- Test Case 4: Verify that each card has a unique color and number.
- Test Case 5: Verify that the deck is shuffled and stored in a random order.

Data Storage and Display:

- Test Case 10: Verify that the system lists all the cards held by the winning player after each round.
- Test Case 11: Verify that the system stores the name and quantity of cards of the winning player in an external file.
- Test Case 12: Verify that the system displays the name and quantity of cards held by the top 5 players with the highest quantity of cards from the external file.

These test cases cover various aspects of the card game, including authorization, deck initialization, game play mechanics, and data storage/display requirements.

2.2 Develop a test plan in line with requirements.

FUNCTIONAL REQUIREMENTS for card game:	NON-FUNCTIONAL REQUIREMENTS for card game:
WHAT	**HOW**
The game must allow for 2 players to participate, and only authorized players are allowed to play the game	The game should ask for a password and name, so only authorized players should be able to access and play the game The game must be secure to protect player data and prevent unauthorized access.
There must be a deck of 30 shuffled cards, each unique. Each card must have one of three colors: red, black, or yellow. Each card must have a number from 1 to 10 for each color. Player 1 takes the top card from the deck, and Player 2 takes the next card.	The game should provide a user-friendly interface for easy gameplay as well as show general roles before start of game The game should have efficient performance and response time.
If both players have a card of the same color, the player with the highest number wins. If players have cards of different colors, the winning color is determined by the rule: RED beats BLACK, BLACK beats YELLOW, YELLOW beats RED. The winner of each round keeps both cards and Player 2 takes the next card. Players continue playing until there are no cards left in the deck.	The game should display the result of each round The game should store and display the cards held by the winning player accurately
The results of playing a game between 2 players have to be saved.	The external file storing player information should be secure and accessible only to authorized users.

The results of the winners should be saved and sorted in descending order	The game should display the name and quantity of cards held by the top 5 players accurately based on data from the external file.

All software work adheres to the IEEE Standard for Software and System Test Documentation (IEEE 829-2008), which sets out the guidelines for creating a software test plan document. This standard defines the necessary format and content for thorough test planning in software development projects. For my project, "Louise's game," I will focus on explaining a portion of the 19 paragraphs required by this standard.

My project's testing plan you can find below. This test plan provides a structured approach to testing the card game application, ensuring that all functional and non-functional requirements are thoroughly validated before release.

	Name of paragraph IEEE 829-2008	Describing related to Louise's game
1	**INTRODUCTION**	This test plan outlines the testing approach for the card game application based on the provided requirements[1]
2	**REFERENCES** All documents that support this test plan	• Functional and Requirement Specifications • Design Specifications • Prototype • Data Model
3	**TEST ITEMS / OBJECTIVES** These are things I intend to test within the scope of this test plan. Essentially, something I will test, a list of what is to be tested.	The test items include the game software, user interface, player authentication system, card deck management, game logic, result storage, and data display functionalities. - Ensure that the game functions as per the rules and requirements. - Verify that only authorized players can access and play the game. - Confirm that card shuffling and dealing are done correctly. - Validate the game logic for determining the winner based on card color and number. - Test the functionality to list and store winning player's cards in an external file. - Verify the display of top 5 players with the highest quantity of cards from the external file.
4	**SOFTWARE RISK ISSUES**	Risks related to data security, performance issues, or functionality gaps will be identified, assessed, and mitigated during testing
5	**FEATURES TO BE TESTED** This is a listing of what is to be tested from the USERS viewpoint of what the system does.	The following features will be tested **Authorization Testing** • Verify that only authorized players are able to access and play the game. **Card Shuffling and Dealing**

[1] For more information about the requirements, you could read the technical documentation part «SOFTWARE REQUIREMENTS»

		• test to ensure that all 30 cards are shuffled and stored in the deck. **Game Logic Testing** • test scenarios where both players have cards of the same color and different numbers. • test scenarios where both players have cards of different colors to validate winning color logic. **Result storage and sorting** **External File Interaction** • Verify that the game correctly lists all cards held by the winning player in an external file. • Validate the storage of the name and quantity of cards of the winning player in the external file. **Display of Top 5 Players.** Test to ensure that the application displays the name and quantity of cards of the 5 players with the highest quantity from the external file. **User interface design and usability** **Performance and response time**
6	**FEATURES NOT TO BE TESTED** This is a listing of what is NOT to be tested from both the Users viewpoint of what the system does and a configuration management/version control view. This is not a technical description of the software, but a USERS view of the functions.	• Create new player / user • Forgotten username / password
7	**APPROACH**	Testing will involve just manual methods to verify the functionality and performance of the game
8	**ITEM PASS/FAIL CRITERIA** A failure is the result of a defect as seen by the User, the system crashes, etc.	**EXIT CRITERIA** • Testing is finished and there are no functional bugs • All remaining bugs have low severity • No more than 10% of medium-severity bugs are open
9	**SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS**	• Critical Bugs are open and they are blocking testing • All remaining test cases are blocked by an open bug
	ROLES AND RESPONSIBILITIES -	The testing team will consist of testers responsible for executing test cases, reporting bugs, and ensuring the quality of the game. 1
10	**TEST DELIVERABLES**	• Test Cases • Bugs Report • Test Summary Report
11	**REMAINING TEST TASKS**	By completing remaining test tasks for Luise's Card Game as outlined in the software test plan document, the testing team can ensure that the game meets quality standards, is free of

		critical issues, and provides an engaging and enjoyable experience for players. Execute remaining test cases for all game features, including gameplay mechanics, card interactions, scoring systems, and user interface elements. Test closure activities, such as preparing a test summary report, conducting a post-mortem meeting to discuss lessons learned, and archiving test artifacts for future reference.
12	**ENVIRONMENTAL NEEDS**	• Test Data: Sample set of 30 unique cards with different colors and numbers. • Operating System: Windows 10 • Mobile system: Android • Browser: Google Chrome The last available version • Access to internet connection

Here is template for testing with all necessary information.

Project name	**LOUISE'S CARD GAME**					Date	
Test Case Description						Test Type	Manual
Covering requirements							
Dependencies							
Process	**N**	**Step**	**Expected result**	**Pass/Fail**	**Data Input**	**Actual outcome**	

2.3 Implement a test plan and record results.

A test strategy is a comprehensive plan that defines the overall approach to testing for a project or product. While the test strategy provides strategic guidance for the entire testing process, the test log focuses on documenting specific testing activities.

Test Log is a detailed record of individual test executions and their outcomes (details of each test execution, including the test cases executed, their results, any defects found, and other relevant information)

Usage: Test logs are used for tracking the progress of testing, identifying patterns in test results, documenting the history of testing activities, and providing a clear audit trail of the testing process.

Content: The test log typically includes details such as test case IDs, descriptions, actual results, expected results, timestamps, the tester's name, and any comments or notes related to the test execution.

Project name	**LOUISE'S CARD GAME**					Date	
Test Case Description	Player Authentication System Test					**Test Type**	**Manual**
Covering requirements	The game must allow for 2 players to participate, and only authorized players are allowed to play the game						
Dependencies							
	Step		**Expected result**	**Pass/Fail**	**Data Input**	**Actual outcome**	**Comments**
Verifying that an authorized player can access the game.	1	Launch the game Enter valid player Username	Player should be authenticated and allowed to access the game	Pass	**Username** Lana Piter Sveta Sue Silva	Access was allowed after entering credentials, but no confirmation message was given.	Display some kind of text like "Hello!" + name of player
	2	Enter valid player Password		Pass	**Password** 123456 456789 789123 123789 123123		
	3	Verify if the player is authenticated successfully		Pass			
Verifying that an unauthorized player cannot access the game.	1	Launch the game Enter acceptable player username not on the list		Pass	Theo Maggie	an error message was displayed	
	2	Enter valid player Password		Pass	Qw787 Ere7879	an error message was displayed	
	3	Verify that an error message is displayed or the game does not start.		Pass		game does not start	

For the card game based on the provided requirements here are some other test cases that should be transformation on a requirement template for testing, but unfortunately due to lack of time there are just main description, the resalt of this testing was developed as report in the next paragraph.

By executing these test cases, can be ensured that the card game functions correctly according to the specified requirements.

DECK INITIALIZATION TESTING

Test Case 2: Card Deck Creation and Shuffling

Description: Verify that the deck is initialized with 30 unique cards of different colors and numbers.

Test Steps:

1. Initialize the deck.
2. Check if the deck contains 30 unique cards with
3. Check if the card has a valid color (red, black, or yellow) and number (1-10).
4. Verify that the deck is shuffled and stored in a random order.

Expected Result: Card deck should contain the correct number of cards and should be shuffled effectively.

GAME MECHANICS TESTING

Test Case 3, 4, 5, 6 : Game Logic Testing

Test 3 Description: Verify the game determines the winner correctly when both players have cards of the **same colour** with different numbers.

Test Steps:

1. Simulate a game round with two players.
2. Test scenarios where both players have cards of **the same colour**
3. Verify the correct winner based on number comparison.

Test 4 Description: Verify the game play scenarios where both players have cards of **different colours** to validate winning colour logic.

Test Steps:

1. Simulate a game round with two players.
2. Test scenarios where both players have cards of different colours
3. Verify the correct winner based on colour comparison.

Test 4 Description: Verify that the winning player keeps both cards after each round.

Test 5 Description: Verify that the game ends when there are no cards left in the deck and have option to display resalt as a list of winner's cards

Test Steps:

1. Simulate playing the game until all cards are drawn.
2. Check if the game ends correctly when the deck is empty.

3. Enter an answer as game require.
4. Check option short resalt.
5. Check option short full.

Test Case 6: Verify Card Distribution:

Description: Verify that each player receives cards in the correct order (Player 1, Player 2).

Test Steps:

1. Start a new game and track the order in which cards are distributed to players.

2. Ensure that Player 1 receives the top card from the deck and Player 2 receives the next card.

DATA STORAGE AND DISPLAY

- Test Case 10: Verify that the system lists all the cards held by the winning player after each round.
- Test Case 11: Verify that the system stores the name and quantity of cards of the winning player in an external file.
- Test Case 12: Verify that the system displays the name and quantity of cards held by the top 5 players with the highest quantity of cards from the external file.

USER INTERFACE DESIGN AND USABILITY

Test Case 4: User Interface Design and Usability

Test Steps:

1. Interact with the game interface.
2. Check for user-friendly design elements.
3. Evaluate ease of navigation and interaction.

Expected Result: User interface should be intuitive, visually appealing, and easy to use.

Test Case 5: Performance and Response Time

Test Steps:

1. Simulate multiple players accessing the game simultaneously.
2. Measure response time for various actions within the game.
3. Monitor system performance under load.

Expected Result: Game should respond promptly to user actions and maintain stable performance under load.

These test cases cover various aspects of the card game software, including player authorization, deck initialization, game play mechanics, end conditions, and card distribution. By executing these test cases, can be ensured that the card game functions correctly according to the specified requirements.

Test Report for "Louise's Game"

Test Date: 18.06.2024
Tester: SVITLANA RADCHENKO

1. Introduction
The purpose of this test report is to provide an overview of the testing conducted on the card game application. The application is designed for 2 players, with specific rules for gameplay and additional requirements for recording and displaying player data.

2. Test Environment
- Operating System: Windows 10
- Device: Computer
- Test Tools: just manual

3. Test Cases
The following test cases were executed to ensure the functionality and performance of the card game application:

3.1 Authorization
- Test Case 1: Verify that only authorized players are able to access the game.
- Test Result: Pass

3.2 Deck Shuffling
- Test Case 2: Confirm that the 30 cards are shuffled and stored in the deck.
- Test Result: Pass

3.3 Card Attributes
- Test Case 3: Validate that each card is unique and has one color (red, black, or yellow) and a number (1-10) for each color.
- Test Result: Pass

3.4 Gameplay
- Test Case 4: Execute the gameplay sequence as per the provided rules.
- Test Result: Pass

3.5 Winner Details
- Test Case 5: Ensure that the winning player's cards are listed after each round.
- Test Result: Pass

3.6 Data Storage
- Test Case 6: Verify that the name and quantity of cards of the winning player are stored in an external file.
- Test Result: Pass

3.7 Leaderboard Display

- Test Case 7: Check if the names and quantities of cards of the top 5 players are displayed from the external file.
- Test Result: Pass

4. Conclusion
The card game application has been tested according to the provided requirements and additional specifications. The test results indicate the overall functionality and performance of the application.

5. Approval
This test report is approved by:
[Name and Signature of Approver]

[Additional Comments or Notes]

Cases passing without any critical issues. The "Louise's Game" demonstrated robust functionality, adherence to rules, user-friendly interface, and satisfactory performance under load. The results indicate that the game is ready for deployment to users. Please let me know if you need further information or assistance with testing.