3. Test Plan

Unit Tests

Our unit tests follow a bottom-up approach. Each test case in the `CardTest` class focuses on testing a specific method or functionality of the `Card` class independently. These tests ensure that individual methods such as `setIdentity`, `flipCard`, `matchCard`, `reset`, etc., perform their intended functionality correctly. By testing each method in isolation, we can identify and address issues at the method level, allowing for targeted debugging and maintenance. This approach facilitates thorough testing of the `Card` class at a granular level before integrating it with other components or modules.

Test Case Name:	***Test Set Identity***
Test Case Description:	This ensures that the identity of the card can be set correctly.
Test Steps:	• Create a `JButton` instance for the card's button representation. • Create a `Card` instance using the `JButton`. • Set the identity of the card to a known value using the `setIdentity` method. • Assert that the identity of the card matches the expected value using `assertEquals`
Pre-Requisites:	None.
Expected Results:	The identity of the card should be set to the specified value.
Test Category:	Unit Test.
Requirement:	Verify that the setIdentity method of the Card class correctly sets the identity of the card.
Automation:	Automated using JUnit.
Date Run:	April 1st, 2024
Pass/Fail:	Pass.
Test Results:	Identity of cards are set up correctly.
Remarks:	None.

Test Case Name:	***Test Flip Card***
Test Case Description:	This tests that flipping the card changes its flipped state from false to true.
Test Steps:	• Create a `JButton` instance for the card's button representation. • Create a `Card` instance using the `JButton`. • Assert that the initial state of the card's flipped status is `false`. • Call the `flipCard` method. • Assert that the flipped status of the card is now `true` using `assertTrue`.
Pre-Requisites:	None.
Expected Results:	Card should change its flipped state from false to true.
Test Category:	Unit Test.
Requirement:	Verify that the flipCard method of the Card class flips the card correctly.
Automation:	Automated using JUnit.
Date Run:	April 1st, 2024
Pass/Fail:	Pass.
Test Results:	Flipped states are changed from false to true.
Remarks:	None.

Test Case Name:	***Test Match Card***
Test Case Description:	Tests the matchCard method to ensure that matching the card

Test Steps:	• Create a `JButton` instance for the card's button representation. • Create a `Card` instance using the `JButton`. • Assert that the initial state of the card's flipped status is `false`. • Call the `matchCard` method. • Assert that the flipped status of the card is now `true` using `assertTrue`.
Pre-Requisites:	None.
Expected Results:	Card should change its flipped state from false to true.
Test Category:	Unit Test.
Requirement:	Verify that the matchCard method of the Card class matches the card correctly.
Automation:	Automated using JUnit.
Date Run:	April 1st, 2024
Pass/Fail:	Pass.
Test Results:	Matched states are changed from false to true.
Remarks:	None.

Test Case Name:	***Test Reset***
Test Case Description:	Tests reset method to ensure that resetting the card changes both its flipped and matched states to false.
Test Steps:	• Create a `JButton` instance for the card's button representation. • Create a `Card` instance using the `JButton`. • Flip the card and match it to set its state. • Assert that the flipped status and matched status are initially `true`. • Call the `reset` method. • Assert that both flipped and matched statuses are now `false`.
Pre-Requisites:	The card should be flipped and matched before resetting.
Expected Results:	After reset, the card's flipped and matched states should be false.
Test Category:	Unit Test.
Requirement:	Verify that the reset method of the Card class resets the card correctly.
Automation:	Automated using JUnit.
Date Run:	April 1st, 2024
Pass/Fail:	Pass.
Test Results:	Resets the cards back to its flipped state if match is false.
Remarks:	None.

Test Case Name:	***Test Identity Persistence After Flip***
Test Case Description:	Tests whether the identity of the card persists after flipping it.
Test Steps:	• Create a `JButton` instance for the card's button representation. • Create a `Card` instance using the `JButton`. • Set the identity of the card to a known value. • Flip the card. • Assert that the identity of the card remains unchanged after flipping using `assertEquals`.
Pre-Requisites:	The identity of the card should be set before flipping.
Expected Results:	The identity of the card should remain unchanged after flipping.

Test Category:	Unit Test.
Requirement:	Verify that the identity of the card persists after flipping.
Automation:	Automated using JUnit.
Date Run:	April 1st, 2024
Pass/Fail:	Pass.
Test Results:	Resets both the flipped and matched states to false.
Remarks:	None.

Test Case Name:	***Test Identity Persistence After Reset***
Test Case Description:	Tests if the ID of the card persists after resetting.
Test Steps:	• Create a `JButton` instance for the card's button representation. • Create a `Card` instance using the `JButton`. • Set the identity of the card to a known value. • Reset the card. • Assert that the identity of the card remains unchanged after resetting using `assertEquals`.
Pre-Requisites:	The identity of the card should be set before resetting.
Expected Results:	The identity of the card should remain unchanged after resetting.
Test Category:	Unit Test.
Requirement:	Verify that the identity of the card persists after resetting.
Automation:	Automated using JUnit.
Date Run:	April 1st, 2024
Pass/Fail:	Pass.
Test Results:	The ID of the card persists after resetting.
Remarks:	None.

Test Case Name:	***Test Multiple Cards***
Test Case Description:	Tests multiple instances of the Card objects to ensure their individual states are maintained correctly.
Test Steps:	• Create two `JButton` instances for representing two cards. • Create two `Card` instances using the `JButton` instances. • Set identities for both cards. • Assert the initial flipped and matched states of both cards. • Flip one card and match the other. • Assert the flipped and matched states of both cards individually.
Pre-Requisites:	None.
Expected Results:	The flipped and matched states of multiple cards should be maintained correctly.
Test Category:	Unit Test.
Requirement:	Verify that multiple instances of the Card class maintain their individual states correctly.
Automation:	Automated using JUnit.
Date Run:	April 1st, 2024
Pass/Fail:	Pass.
Test Results:	Multiple instances of Card object are maintained correctly.
Remarks:	None.

Test Case Name:	***Test Null Identity***
Test Case Description:	Tests setting the identity of a card to null and ensures that the identity remains null.
Test Steps:	• Create a `JButton` instance for the card's button representation. • Create a `Card` instance using the `JButton`. • Set the identity of the card to `null`. • Assert that the identity of the card is `null` using `assertNull`
Pre-Requisites:	None.
Expected Results:	The ID of the card should be set to null.
Test Category:	Unit Test.
Requirement:	Verify that setting the ID of the card to null works as expected.
Automation:	Automated using JUnit.
Date Run:	April 1st, 2024
Pass/Fail:	Pass.
Test Results:	Cards are set to null.
Remarks:	None.

Integration Testing

Test Case Name:	***Test Match Card***
Test Case Description:	Tests the matchCard method to ensure that matching the card
Test Steps:	• Create a `JButton` instance for the card's button representation. • Create a `Card` instance using the `JButton`. • Assert that the initial state of the card's flipped status is `false`. • Call the `matchCard` method. • Assert that the flipped status of the card is now `true` using `assertTrue`.
Pre-Requisites:	None.
Expected Results:	Card should change its flipped state from false to true.
Test Category:	Unit Test.
Requirement:	Verify that the matchCard method of the Card class matches the card correctly.
Automation:	Automated using JUnit.
Date Run:	April 1st, 2024
Pass/Fail:	Pass.
Test Results:	Matched states are changed from false to true.
Remarks:	None.

Validation Testing

Test Case Name:	***Test are all cards matched method in GameBoardTest()***
Test Case Description:	Tests if all cards are on the gameboard are matched.

Test Steps:	1. Create a game board with a known size and difficulty. 2. Verify that initially, all cards on the board are not matched. 3. Flip all cards on the board to matched. 4. Check if all cards on the board are now matched
Pre-Requisites:	Game board must be instantiated. Size and difficulty parameters must be specified. Card objects on the game board must be created and initialized.
Expected Results:	All cards on the gameboard are matched.
Test Category:	Unit Test.
Requirement:	All cards should be in an unmatched state. After flipping and matching all cards, system should correctly identify that all cards are matched.
Automation:	Automated using JUnit.
Date Run:	April 1st, 2024
Pass/Fail:	Pass.
Test Results:	All cards on the gameboard are matched.
Remarks:	None.

System Tests

Test Case Name:	***Test all main functionalities on Windows***
Test Case Description:	Make sure all functionality works as intended on Windows, such that a Windows 10 user could seamlessly use the program to fulfill their needs.
Test Steps:	*On a Windows machine:* Perform the **Test all functional requirements** validation test.
Pre-Requisites:	*The program should be accessed from the distribution ZIP file.*
Expected Results:	*All functionalities should perform as intended and without major differences from other operating systems.*
Test Category:	*System Test*
Requirement:	Compile everything before running.
Automation:	*Manual*
Date Run:	*April 1th, 2023*
Pass/Fail:	*Pass*
Test Results:	*All functionalities work as intended.*
Remarks:	*None.*