Testing Introduction

Overview:

Atlasventure is a multifaceted educational game designed to marry the elements of entertainment with the foundational aspects of geography learning. This game provides an interactive platform for players to engage with various geographical challenges, incorporating a variety of screens, user interactions, and feedback systems to foster an engaging and informative environment. Central to Atlasventure's development has been the implementation of a robust design that supports educational objectives through gameplay mechanics such as scoring systems, progression mechanics, and educational challenges. Ensuring these components function as intended necessitates a comprehensive testing approach, focusing on verifying the seamless operation of game functionalities, the intuitive nature of the user interface, and the educational value delivered through the game's content. Testing documentation is pivotal in outlining the strategies, tools, and methodologies adopted to assess every aspect of Atlasventure, from user interface navigation to game state management and educational content delivery.

Objectives:

The primary objectives of the Atlasventure project encompass a broad spectrum of goals, from providing an engaging educational experience in geography to ensuring the game's development aligns with established software engineering principles. Key objectives include:

- **Educational Enhancement**: To leverage the gaming platform for enhancing geographical knowledge and skills among players, making learning an interactive and enjoyable experience.
- **Software Engineering Application**: To apply and demonstrate comprehensive software engineering principles throughout the game's lifecycle, including design, development, and testing phases.
- **Collaborative Development**: To manage and execute the project through effective team dynamics, communication, and collaborative tools, ensuring a coherent and structured development process.
- **Technical Robustness**: To develop a technically sound application that is scalable, maintainable, and adheres to best coding practices, with a particular focus on the user interface, game mechanics, and data management.
- **Reflective Learning**: To engage in a continuous reflective process, assessing and learning from design and development decisions to recognize successes and identify areas for improvement.

References:

- Servos, D. *CS2212 Group Project Specification*. https://owl.uwo.ca/access/content/group/aa2311c9-4cef-497f-8d9c-b502023be21c/project/CS2212%20Group%20Project%20Specification.pdf
- Requirements Documentation
- Design Documentation

Testing Summary

Our testing documentation outlines a comprehensive strategy for ensuring the strength and dependability of our geography-based educational game. We thoroughly tested each aspect of the software against our project specifications and user requirements using a combination of white box and black box testing methodologies, such as automated unit tests with JUnit and targeted manual testing for user interaction components. The detailed test plan, which includes unit, integration, validation, and system testing, outlines a systematic approach to identifying and resolving potential issues, ensuring a consistent user experience across multiple systems, particularly compatibility with Windows 10. This document reflects our team's dedication to quality and serves as a road map for reaching a high level of software excellence.

Test Plan

Unit Testing

ShowFunFactTest - This JUnit class, `ShowFunFactTest`, is designed to test the functionality of displaying fun facts. It begins by setting up a `QuizGame` instance with predefined fun facts for two questions and then we perform different test cases to check if the fun facts are accurately displayed or not. The first test case checks that the game correctly displays a fun fact alongside when a user answers the question correctly. The second test case ensures that if the user answers incorrectly, the game shows the correct answer followed by a fun fact. And the third test case verifies that no message is displayed when there is no fun fact available

CheckAnswerLevel1GameClassTest - This Junit class, CheckAnswerLevel1GameClassTest, is designed to rigorously test the answer checking functionality within the game. A few examples of the test cases include:

1. **Correct Answer Increases Points:** This test ensures that the system correctly identifies a correct answer and awards the appropriate points, increasing the total score by 10.
2. **Incorrect Answer Does Not Increase Points:** It validates that when an answer is incorrect, the system does not increase the total points by mistake. This check makes sure that the points only increase by 10 when a user gets the answer right.

createQuestionPanel - This test ensures the createQuestionPanel method correctly initializes a question panel with a title, image, options, and hint functionality, adhering to design specifications.

HintSystemTest file

1. **provideHint_WithFreeHint()**: Tests that a hint is provided without deducting player points when a free hint is available.
2. **checkFreeHintAvailability()**: Verifies that the method correctly reports the availability of free hints.
3. **deductPoints()**: Checks whether points are correctly deducted from the player's total points when they have enough points.
4. **confirmPurchase()**: Ensures that the method accurately confirms whether a player has enough points to purchase a hint.
5. **addPoints()**: Tests adding points to the player's total points and verifies if the new total is correct.

LevelTest file

1. **addQuestion()**: Verifies that a question is correctly added to the level's list of questions.
2. **getNextQuestion()**: Tests retrieving the next question in sequence and checks the behavior when there are no more questions left.
3. **resetLevel()**: Ensures that resetting the level correctly resets the current question index and the score to their initial states.
4. **calculateScore()**: Checks if the score is correctly calculated and updated based on whether the player's answer is correct or incorrect.
5. **isLevelCompleted()**: Verifies that the method accurately determines whether all questions in the level have been answered.
6. **getLevelNum()**: Confirms that the correct level number is returned for the level.
7. **isReadyToUnlockNextLevel()**: Tests whether the condition to unlock the next level is accurately checked, based on the completion status of the current level.

QuestionTest file

1. **isCorrect_CorrectAnswer()**: Tests if the `isCorrect` method correctly identifies a correct answer to the question.
2. **isCorrect_IncorrectAnswer()**: Ensures that the `isCorrect` method accurately identifies an incorrect answer.
3. **displayQuestion()**: Verifies that `displayQuestion` correctly prints the question text and all answer choices to the console.
4. **revealAnswer()**: Checks if `revealAnswer` properly reveals the correct answer to the console.
5. **displayFunFact()**: Confirms that `displayFunFact` correctly displays the fun fact associated with the question to the console.

GameManagerTest file

1. **setUp()**: set up a Game manager instance and frame for the game
2. **tearDown()**: test the closing of the game
3. **testInitialStateMainMenu()**: test mainmenu startup andmake sure gamemanager switches contexts and starts game music
4. **testChangeGameState_NewGame():** test changing state to new game
5. **testChangeGameState_LoadGame():** test changing state to load game
6. **testChangeGameState_DebugMode():** test changing state to Debug Mode
7. **testChangeGameState_Invalid_State():** test changing state to an Invalid State
8. **testSwitchToLevelSelectionScreen_NullPlayerData():** test changing state to Level Selection Screen
9. **testSwitchToGameModeScreen_NullPlayerDate():** test changing state to GameModeScreen

Integration Testing

Test Case Name	Validate initialize Method for Game Screen Setup

Test Case Description	Ensures the initialize method sets up the game screen with a layout for questions, points display, and a back button correctly.
Test Steps	• Invoke the initialize method. • Verify main layout is BorderLayout and background color is gray. • Check cardsPanel creation and addition with question panels for each entry in questionTitles. • Ensure points panel correctly displays total points and is added to the screen. • Confirm back button configuration and functionality.
Pre-Requisites	• Game data arrays for questions, answers, options, hints, and image paths are initialized. • gameManager, levelSelectionScreen, and AudioManager are properly set up.
Expected Results	• Game screen set with gray background and BorderLayout. • cardsPanel contains question panels for each question, configured correctly. • Points display is accurate with specified styling. • Back button shows correct image and navigates back on click, playing a sound.
Test Category	Integration Test
Requirement	The initialize method integrates various components to set up the game screen according to design specifications.
Automation	Partially (Layout and components can be automated; visual and sound aspects may need manual verification.)
Date Run	01 Apr 2024
Pass/Fail	Passed
Test Results	- Game screen layout and background set as expected. - cardsPanel included a question panel for each title, all correctly configured. - Points label reflected totalPoints with correct styling. - Back button functioned correctly, returning to level selection and playing click sound.
Remarks	- Integration of components within initialize method verified successfully. - Visual elements and audio output were manually verified. - No issues encountered, indicating a cohesive setup process for the game screen.

Validation Testing

Test Case Name:	Validate HighScoreScreen GUI Display
Test Case Description:	This test case verifies that the HighScoreScreen class correctly displays high scores and functions as intended.
Test Steps:	1. Instantiate HighScoreScreen with a mock GameManager and a predefined list of GameData objects for top scores. 2. Call the initialize method of the HighScoreScreen instance. 3. Verify that all components are initialized correctly with the appropriate properties. 4. Call the activate method and ensure the screen is visible. 5. Interact with the back button and confirm navigation.

Pre-Requisites:	• The GameManager class and GameData class must be defined. • The images and resources must be available in the specified paths. • Mock objects or a testing framework like Mockito to simulate the GameManager behavior.
Expected Results:	The HighScoreScreen displays the list of high scores with correct ranks, player names, and scores. The back button navigates back to the main menu when clicked.
Test Category:	GUI Validation Test
Requirement:	The GUI must correctly display a list of high scores and allow navigation back to the main menu.
Automation:	No (This is a GUI validation test that is best performed manually to verify visual elements and interactions.)
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The HighScoreScreen GUI successfully displayed the predefined list of high scores with correct ranks, player usernames, and scores. The high scores were correctly aligned and formatted in a grid layout as expected. The title "High Scores" was centered at the top of the screen, and the font styling matched the requirements. The back button with the correct icon was present in the upper left corner of the screen, and it functioned correctly when clicked, navigating back to the main menu. The screen's background color was appropriately set to gray, and all components were visible and rendered correctly. The screen was also responsive to the activate and deactivate methods, showing and hiding the screen as expected.
Remarks:	The test was performed manually to ensure that the GUI elements are not only present but also correctly positioned and visually appealing. All components behaved as expected, with no deviations from the expected results. The test environment was set up with mock objects for GameManager and a fixed set of GameData to simulate a realistic scenario. No issues were encountered during the test. The successful execution of this test suggests that the HighScoreScreen class meets the design and functional requirements for displaying high scores. The visual inspection confirmed that the GUI is user-friendly and meets the usability standards set for the game.

Test Case Name	Validate createProfileAction Functionality
Test Case Description	This one just ensures the createProfileAction method successfully creates a new user profile with validation for empty fields and existing usernames, and correctly saves the new game data.

Test Steps	• Trigger createProfileAction with various inputs for username and password. • Check for feedback with empty fields. • Test with an existing username. • Validate successful profile creation for a new unique username.
Pre-Requisites	• UsernameField and passwordField are initialized and accessible • FeedbackLabel is set up for displaying messages. • Game data storage system is functional.
Expected Results	• Feedback for empty fields when either username or password is not provided. • Message indicating username exists for duplicate usernames. • Successful profile creation message and transition to the game mode screen for unique usernames.
Test Category	Functional Test
Requirement	The method must handle user input validation, check for existing usernames, create new user profiles, and save them properly.
Automation	Yes, except for the final screen transition and sound playback, which might require manual verification.
Date Run	01/ Apr 2024
Pass /Fail	Passed
Test Results	• The method correctly identified and provided feedback for empty input fields. • Attempting to create a profile with an existing username correctly triggered the "Username already exists" feedback. • A unique username and password led to successful profile creation, with the new game data saved and a transition to the game mode screen. • Hashing of the password was verified by inspecting the saved game data
Remarks	The test covered key functional aspects of the createProfileAction method, including input validation, username uniqueness checks, password hashing, and data persistence. Manual checks were performed for the auditory feedback from the button click and the screen transition to the game mode screen. No issues were encountered during the test.

Test Case Name:	Validate InstructorDashboardScreen GUI Access and Functionality
Test Case Description:	Ensure that the InstructorDashboardScreen GUI properly restricts access via a password prompt, loads, and displays game-related statistics correctly, and allows navigation back to the main menu.
Test Steps:	1. Attempt to instantiate InstructorDashboardScreen with a mock GameManager. 2. Verify that the password prompt is displayed. 3. Enter the correct password ("123") and verify access to the dashboard. 4. Check that all UI components such as player panels and scroll pane are initialized and added to the screen. 5. Confirm that each player's game data is correctly retrieved and displayed. 6. Interact with the back button and confirm navigation back to the main menu. 7. Attempt to instantiate InstructorDashboardScreen with an incorrect password and verify that access is denied.
Pre-Requisites:	• The GameManager and GameData classes should be defined. • The password for the instructor dashboard is known and set as "123". • The data source for GameData is populated and accessible.

Expected Results:	• Password prompt is shown when the screen is instantiated. • Upon entering the correct password, the screen displays player statistics. • Incorrect password entry should not grant access to the dashboard. • The back button functions correctly to return to the main menu. • The UI components are laid out according to the specifications, with no visual issues.
Test Category:	GUI Validation Test
Requirement:	The screen must restrict access to authorized users via a password, correctly display player statistics, and provide navigation back to the main menu.
Automation:	No (Manual testing is recommended for validating visual elements and interactive behavior.)
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The password prompt appeared upon the screen's initialization. The correct password ("123") granted access to the instructor dashboard. The game data statistics for each player were displayed correctly in their respective panels within the scroll pane. The back button, located at the top left corner of the screen, worked as intended, and navigation to the main menu was successful. Attempting to access with an incorrect password correctly restricted entry, confirming the functionality of the access control. All visual components were rendered according to the specified layout, and the interface was intuitive and easy to navigate.
Remarks:	The test went smoothly with no significant issues. Access control via password worked correctly, and the user interface elements met the expected standards. This validation test confirms that the InstructorDashboardScreen behaves as expected in both authorized access scenarios and unauthorized access attempts. The data was well-structured, and the overall design was user-friendly, adhering to the requirements. The components were all functional, and the password verification process did not present any latency or errors. All success criteria were met.

Test Case Name:	GameManagerTest - Successful Login
Test Case Description:	Verify that all methods in GameManager are working as intended
Test Steps:	1. GameManager Instance. 2. Check if Frame is generating as expected 3. pass game states to the manager to test visual and auditory feedback
Pre-Requisites:	N/A
Expected Results:	A game window should be created and should cycle though all the game states
Test Category:	Functional Test
Requirement:	N/A
Automation:	Yes
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The GameManager successfully generated a game frame and shuffled through the game states
Remarks:	Critical Class to every aspect of the game functions as intended without bugs

Test Case Name:	Validate gameModeScreen
Test Case Description:	This test case verifies the correct display and functionality of the GameModeScreen. This helps to ensure that all of the UI elements are properly laid out, the game mode options are selectable, and navigation buttons like Back to Main Menu function as we intend to.
Test Steps:	1. First instantiate GameModeScreen with GameManager and GameData to simulate the game environment. 2. Then the initialize method is then called so that the correct instantiation is checked for of all UI the elements (Main focus on the title, game mode buttons, and the back button.) 3. Verify the title Game Options is correctly displayed at the top center of the screen. 4. Confirm the presence and proper alignment of game mode buttons ("Classic", "Theme-Based") and their responsiveness to clicks. 5. Test the Back to Main Menu button to ensure it navigates back to the main menu screen effectively. 6. Validate the sound effects are triggered upon button clicks, enhancing user interaction feedback.
Pre-Requisites:	• The gameModeScreen, gameManager, and gameData classes are already included • This class has to correctly instantiate and display all the UI elements described in its design, such as the game mode selection buttons and the 'Back to Main Menu' button, ensuring they align with the intended layout and aesthetics. • The GameModeScreen class represents a specific user interface that allows players to select between different game modes (Classic and Theme-Based). For testing purposes, this class should ensure that the different methods, event listeners, and UI components (buttons, labels, etc.) are coded and functional
Expected Results:	• The screen displays the title "Game Options" and buttons for selecting game modes. • Each button correctly labels the game mode it represents and is interactable. • The 'Back to Main Menu' button is visible, correctly positioned, and navigates back to the main menu when clicked. • Clicking on game mode buttons transitions to the respective game mode selection or level selection screens. • Button click sound effects are played, providing auditory feedback to the user.
Test Category:	GUI Validation Test
Requirement:	• The GameModeScreen has a clear and intuitive interface for selecting game modes, and it is complete with navigational functionality so that the user can be returned to the main menu.
Automation:	No (Manual testing is preferred for UI element verification and interaction testing.)
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The GameModeScreen was successfully initialized with all expected UI components in their correct positions. The game mode buttons (Classic and Theme-Based) were responsive, and their selection led to the correct game mode environment. The 'Back to Main Menu' button functioned correctly, providing a seamless navigation experience back to the main menu. Sound effects for button clicks were properly implemented, adding to the overall user interaction quality.
Remarks:	The test confirmed that the screen's design and functionality are in line with the game's requirements for a mode selection feature. It was also noted that there were no issues were encountered during the manual testing process, indicating a well-implemented GUI screen.

Test Case Name	Validate showFunFact pop up
Test Case Description	This test case verifies that the showFunFact method displays a correct congratulatory message or a correction message along with a fun fact, based on the answer correctness and specified question index.
Test Steps	1. Initialize necessary objects and currentLevelData with predefined fun facts. 2. Invoke showFunFact with isCorrect set to true and a valid questionIndex, observe the dialog. 3. Invoke showFunFact with isCorrect set to false and the same questionIndex, observe the dialog.

Pre-Requisites	• currentLevelData initialized with fun facts. • Valid questionIndex within fun facts array bounds. • GUI environment for JOptionPane dialogs.
Expected Results	• With isCorrect true: Dialog shows "Correct! Fun Fact: [fun fact]". • With isCorrect false: Dialog shows "Incorrect. The correct answer is [correctAnswer]. Fun Fact: [fun fact]".
Test Category	GUI Validation Test
Requirement	The showFunFact method should appropriately display a message and fun fact for both correct and incorrect answers based on the provided index.
Automation	No (Manual observation of GUI elements and interactions is required.)
Date Run	01 Apr 2024
Pass /Fail	Passed
Test Results	The method displayed correct messages and fun facts for both scenarios. For a correct answer, "Correct! Fun Fact: [fun fact]" was shown. For an incorrect answer, "Incorrect. The correct answer is [correctAnswer]. Fun Fact: [fun fact]" was displayed. The fun facts corresponded correctly to the questionIndex, and messages were formatted as expected.
Remarks	The test was carried out manually to ensure the accuracy of dynamic content and user interactions within the GUI. All behaviors matched expected outcomes with no deviations. The test setup included a mock environment with predefined fun facts to simulate realistic application data. The GUI elements were verified for correctness and alignment with design standards.

Test Case Name:	NewGameScreen - Create Profile Existing Username
Test Case Description:	Check the system's response to a profile creation attempt using an existing username.
Test Steps:	1. Launch NewGameScreen. 2. Enter an existing username and a valid password. 3. Click the "Create Profile" button.
Pre-Requisites:	The game_data.json contains the entered username.
Expected Results:	An error message indicating the username already exists is displayed.
Test Category:	Validation Test
Requirement:	The system should prevent the creation of a profile with a username that already exists.
Automation:	No
Date Run:	02 Apr 2024
Pass/Fail:	Passed
Test Results:	Feedback indicated "Username already exists. Please choose another."
Remarks:	Profile creation correctly prevents duplicate usernames, ensuring user uniqueness.

Test Case Name:	NewGameScreen - Create Profile Existing Username
Test Case Description:	Check the system's response to a profile creation attempt using an existing username.
Test Steps:	1. Launch NewGameScreen. 2. Enter an existing username and a valid password. 3. Click the "Create Profile" button.
Pre-Requisites:	The game_data.json contains the entered username.
Expected Results:	An error message indicating the username already exists is displayed.
Test Category:	Validation Test

Requirement:	The system should prevent the creation of a profile with a username that already exists.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	Feedback indicated "Username already exists. Please choose another."
Remarks:	Profile creation correctly prevents duplicate usernames, ensuring user uniqueness.

Test Case Name:	TutorialScreen - Tooltip Text
Test Case Description:	Confirm the question icon button and tooltip text for all options is displayed correctly
Test Steps:	1. Hover over the question icon buttons. 2. Observe the tooltip.
Pre-Requisites:	TooltipManager is initialized, and the question icon buttons are displayed with the correct tooltip set.
Expected Results:	The tooltip detailingthe corresponding instruction appears instantly upon hover.
Test Category:	UI/UX Test
Requirement:	Tooltips should display instantly and provide helpful information on how to use the game features.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The tooltips were all displayed correctly and instantly.
Remarks:	The tooltip functionality is performing as expected, enhancing user understanding the gameplay.

Test Case Name:	TutorialScreen - Back Button Functionality
Test Case Description:	Test the back button functionality on the TutorialScreen.
Test Steps:	1. Launch TutorialScreen. 2. Click the back button.
Pre-Requisites:	The screen is initialized and the back button is displayed.
Expected Results:	The game transitions back to the main menu.
Test Category:	Navigation Test
Requirement:	The back button should return the user to the main menu.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	Clicking the back button navigated to the main menu correctly.
Remarks:	The back navigation is working correctly, providing a smooth transition back to the main menu.

Test Case Name:	ThemeBasedModeSelectionScreen - Back Button Functionality
Test Case Description:	Test the back button functionality on the ThemeBasedModeSelectionScreen.
Test Steps:	1. Launch ThemeBasedModeSelectionScreen. 2. Click the back button.
Pre-Requisites:	The screen is initialized and the back button is displayed.
Expected Results:	The game transitions back to the game mode selection screen.

Test Category:	Navigation Test
Requirement:	The back button should return the user to the previous screen without selecting a theme.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	Clicking the back button correctly navigated to the game mode selection screen.
Remarks:	The back button is operational and allows the player to navigate to the previous menu seamlessly.

Test Case Name:	ThemeBasedModeSelectionScreen - CultureTraditionsSelection
Test Case Description:	Confirm the selection of the "Culture/Traditions" theme.
Test Steps:	1. Launch ThemeBasedModeSelectionScreen. 2. Click the "Culture/Traditions" button.
Pre-Requisites:	The screen is initialized and the "Culture/Traditions" button is displayed.
Expected Results:	The game enters the gameplay screen with the "Culture/Traditions" theme active.
Test Category:	Functional Test
Requirement:	"Culture/Traditions" should be selectable and initiate the correct gameplay theme.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The selection of the "Culture/Traditions" theme led to the expected gameplay state with the theme active.
Remarks:	The "Culture/Traditions" theme was selected and applied correctly, with the system providing appropriate feedback.

Test Case Name:	ThemeBasedModeSelectionScreen - Traditional Clothing Selection
Test Case Description:	Validate the functionality of the "Traditional Clothing" theme option.
Test Steps:	1. Launch ThemeBasedModeSelectionScreen. 2. Click on the "Traditional Clothing" button.
Pre-Requisites:	The screen is initialized and the "Traditional Clothing" button is displayed.
Expected Results:	The game transitions to the gameplay screen with the "Traditional Clothing" theme.
Test Category:	Functional Test
Requirement:	The "Traditional Clothing" option should successfully set the theme for gameplay.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The "Traditional Clothing" theme was correctly applied to the game.
Remarks:	Theme application for "Traditional Clothing" worked without issues, confirming the functionality is correct.

Test Case Name:	ThemeBasedModeSelectionScreen - Landmarks Selection
Test Case Description:	Verify selecting the "Landmarks" theme option.
Test Steps:	1. Launch ThemeBasedModeSelectionScreen. 2. Click on the "Landmarks" button.
Pre-Requisites:	The screen is initialized and the "Landmarks" button is displayed.

Expected Results:	The game transitions to the gameplay screen with the "Landmarks" theme.
Test Category:	Functional Test
Requirement:	Selecting "Landmarks" should initiate gameplay with the chosen theme.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The "Landmarks" theme was successfully selected and the game transitioned to the appropriate gameplay state.
Remarks:	The selection process for the "Landmarks" theme is functioning as expected, providing the correct feedback and transition.

Test Case Name:	NewGameScreen - Back Navigation
Test Case Description:	Verify the "Back" button functionality on the NewGameScreen.
Test Steps:	1. Launch NewGameScreen. Click the "Back" button.
Pre-Requisites:	None
Expected Results:	The screen transitions back to the main menu.
Test Category:	Navigation Test
Requirement:	Users should be able to return to the main menu without creating a new profile.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	Clicking "Back" navigated to the main menu correctly.
Remarks:	The back navigation is working correctly, providing a good user experience for navigation.

Test Case Name:	NewGameScreen - Create Profile Existing Username
Test Case Description:	Check the system's response to a profile creation attempt using an existing username.
Test Steps:	1. Launch NewGameScreen. 2. Enter an existing username and a valid password. 3. Click the "Create Profile" button.
Pre-Requisites:	The game_data.json contains the entered username.
Expected Results:	An error message indicating the username already exists is displayed.
Test Category:	Validation Test
Requirement:	The system should prevent the creation of a profile with a username that already exists.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	Feedback indicated "Username already exists. Please choose another."
Remarks:	Profile creation correctly prevents duplicate usernames, ensuring user uniqueness.

Test Case Name:	NewGameScreen - Create Profile Empty Fields
Test Case Description:	Confirm that creating a profile with empty username or password fields is not allowed.

Test Steps:	1. Launch NewGameScreen. 2. Leave the username and password fields empty. 3. Click the "Create Profile" button.
Pre-Requisites:	None
Expected Results:	An error message is displayed, and no profile is created.
Test Category:	Validation Test
Requirement:	Empty fields should not be accepted and should prompt the user to enter valid information.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	Correct feedback was provided, stating "Username and password cannot be empty."
Remarks:	The validation checks for empty fields are functioning correctly, preventing profile creation.

Test Case Name:	NewGameScreen_CreateProfileValid
Test Case Description:	Verify creating a new profile with a valid and unique username and password.
Test Steps:	1. Launch NewGameScreen. 2. Enter a new, unique username and a valid password. 3. Click on the "Create Profile" button.
Pre-Requisites:	The game_data.json does not contain the entered username.
Expected Results:	A new game profile is created, and the game transitions to the game mode selection screen.
Test Category:	Functional Test
Requirement:	Users should be able to create a new profile with unique credentials.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	Profile creation was successful, and transition to game mode selection was smooth.
Remarks:	The user experience was intuitive, and the profile creation process worked as intended.

Test Case Name:	MainMenuScreen - Exit Option
Test Case Description:	Validate the "Exit" functionality from the main menu.
Test Steps:	1. Launch MainMenuScreen. 2. Click on "Exit".
Pre-Requisites:	None
Expected Results:	The game application closes.
Test Category:	Functional Test
Requirement:	The game should close cleanly when "Exit" is selected.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The game closed as expected without errors.
Remarks:	The exit process was smooth, with all resources released appropriately.

Test Case Name:	MainMenuScreen -Debug Mode Option
Test Case Description:	Test the "Debug Mode" option in the main menu.
Test Steps:	1. Launch MainMenuScreen. 2. Click on "High Scores".
Pre-Requisites:	Debug mode is an implemented feature.
Expected Results:	The game enters debug mode.
Test Category:	Functional Test
Requirement:	Debug mode should be enabled when selected from the menu.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	Debug mode was engaged successfully.
Remarks:	Debug mode functionality checked and verified.

Test Case Name:	MainMenuScreen - High Scores Option
Test Case Description:	Validate the "High Scores" button functionality.
Test Steps:	1. Launch MainMenuScreen. 2. Click on "High Scores".
Pre-Requisites:	None
Expected Results:	The high scores screen is displayed.
Test Category:	Functional Test
Requirement:	Should display the high scores list when selected.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	High scores were displayed accurately.
Remarks:	Score display was accurate, and formatting was correct.

Test Case Name:	MainMenuScreen - Instructor Dashboard Option
Test Case Description:	Confirm access to the "Instructor Dashboard".
Test Steps:	1. Launch MainMenuScreen. 2. Click on "Instructor Dashboard".
Pre-Requisites:	The instructor dashboard is implemented and accessible.
Expected Results:	Transitions to the instructor dashboard.
Test Category:	Functional Test
Requirement:	The instructor dashboard should be accessible for authorized users.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The instructor dashboard loaded without issues.

Remarks:	Access control and dashboard functionality were as expected.

Test Case Name:	MainMenuScreen - Tutorial Option
Test Case Description:	Test the "Tutorial" button on the main menu.
Test Steps:	1. Launch MainMenuScreen. 2. Click the "Tutorial" button.
Pre-Requisites:	None
Expected Results:	The game transitions to the tutorial state.
Test Category:	Functional Test
Requirement:	Tutorial mode should be accessible from the main menu.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	Clicking "Tutorial" led to the tutorial state as expected.
Remarks:	Tutorial loaded successfully, the transition was smooth.

Test Case Name:	MainMenuScreen - Load Saved Game Option
Test Case Description:	Confirm the functionality of the "Load Saved Game" option.
Test Steps:	1. Launch MainMenuScreen. 2. Click on the "Load Saved Game" button.
Pre-Requisites:	At least one saved game exists.
Expected Results:	Game state changes to the "LOAD_GAME" state, ready to load a game.
Test Category:	Functional Test
Requirement:	The "Load Saved Game" option should bring up saved game states.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The saved games were presented correctly.
Remarks:	The loading mechanism is working properly, no errors reported.

Test Case Name:	MainMenuScreen - New Game Option
Test Case Description:	Validate the "New Game" option in the main menu.
Test Steps:	1. Launch MainMenuScreen. 2. Click on the "New Game" button.
Pre-Requisites:	The game is at the initial start screen.
Expected Results:	Game state changes to the "NEW_GAME" state.
Test Category:	Functional Test
Requirement:	The "New Game" option should initiate a new game.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed

Test Results:	Clicking "New Game" correctly transitioned the game to the "NEW_GAME" state.
Remarks:	Transition to the new game state was seamless.

Test Case Name:	LoadGameScreen - Back Navigation
Test Case Description:	Verify that the back button on LoadGameScreen returns the user to the main menu.
Test Steps:	1. Launch LoadGameScreen with a mock GameManager. 2. Click the back button.
Pre-Requisites:	The GameManager should be set to track navigation changes.
Expected Results:	The GameManager should record a state change back to the "MAIN_MENU".
Test Category:	Navigation Test
Requirement:	The screen must provide an option to return to the previous menu without logging in.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	Clicking the back button successfully navigated back to the main menu. No unintended behavior occurred during the navigation process.
Remarks:	The back navigation functionality is working correctly, allowing the user to return to the main menu smoothly and quickly.

Test Case Name:	LoadGameScreen - Empty Username/ Password
Test Case Description:	Confirm that the LoadGameScreen prompts the user when the username/password field is left empty.
Test Steps:	1. Launch LoadGameScreen. 2. Leave the username field empty and fill in the password field, or vice versa. 3. Click the login button.
Pre-Requisites:	None
Expected Results:	The feedback label should display "Username and password cannot be empty."
Test Category:	Functional Test
Requirement:	The screen must prevent login attempts with empty fields.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed
Test Results:	The feedback label correctly displayed "Username and password cannot be empty." The login button did not initiate any navigation or transition.
Remarks:	The system correctly prevents login with an empty username. The feedback was immediate and correctly prevented any further action.

System Testing

Test Case Name	Validate goToNextQuestion Level Completion and Scoring

Test Case Description	Ensures the goToNextQuestion method accurately processes level completion, updates player scores and hints used, and navigates based on the player's total points.
Test Steps	• Initialize questionTitles, playerData, gameManager, and levelSelectionScreen. • Increment currentQuestionIndex to simulate reaching the end of the level. • Test with player data set for various scenarios: first completion with >= 30 points, < 30 points, and level replay.
Pre-Requisites	• questionTitles array contains questions. • playerData includes initial level completion status and score. • gameManager and levelSelectionScreen are properly configured for navigation. • JOptionPane is able to display messages.
Expected Results	• For first-time completion with >= 30 points: Show congratulatory message, update level and score, navigate to level selection. • For first-time completion with < 30 points: Show message about insufficient points, no score update, navigate. • For replay: Show message, no score update, navigate.
Test Category	Functional Test
Requirement	The method should correctly manage game progression based on question completion, total points, and player data, updating the player's status and navigating accordingly.
Automation	Yes, with the use of UI automation tools for handling JOptionPane dialogs.
Date Run	01 Apr 2024
Pass/Fail	Passed
Test Results	The method functioned as expected across all scenarios: - First-time completion with >= 30 points resulted in a congratulatory message, level and score were updated correctly, and redirection to level selection occurred. - First-time completion with < 30 points displayed the correct message and redirected without updating the score. Replaying the level displayed the appropriate message and redirected without additional score updates. - The levelSelectionScreen refreshed correctly in all cases.
Remarks	The test covered all intended scenarios, and the method behaved as expected. Edge cases, such as incorrect currentQuestionIndex settings, were handled gracefully. The test ensures robust level completion logic and accurate player progress tracking. No issues were encountered during testing.

Test Case Name:	LoadGameScreen - Successful Login
Test Case Description:	Verify that entering a correct username and password results in successful authentication and navigates to the game mode screen.
Test Steps:	1. Launch LoadGameScreen with a mock GameManager. 2. Enter valid username and password into the respective fields. 3. Click the login button.
Pre-Requisites:	Valid mock user credentials are set up in the mocked GameDataManager.
Expected Results:	The feedback label should display "Login successful." and the GameManager should transition to the game mode screen.
Test Category:	Functional Test
Requirement:	Upon valid login, the user should be authenticated and taken to their game state.
Automation:	No
Date Run:	01 Apr 2024
Pass/Fail:	Passed

Test Results:	The feedback label correctly accepted the input. The GameManager successfully transitioned to the game mode screen after login.
Remarks:	The login functionality works as expected with valid credentials. Transition to game mode screen was smooth and without delay.