Testing Documentation

Capital Chain - Word Chain Game Which Teaches the Audience About Capitals in North America

Testing Documentation

Team Members

Hailey Pong

Victor Yung Wai Tan

Simran Kaur Kullar

Maneet Kaur Chahal

Kareena Sen

1. Main Page

1.1 Table of Contents

- Capital Chain - Word Chain Game Which Teaches the Audience About Capitals in North America
    - Testing Documentation
- 1. Main Page
    - 1.1 Table of Contents
    - 1.2 Tabular Revision History
    - 2. Introduction
    - 2.1. Overview
        - 2.2 Objectives
        - 2.3 References
    - 3. Test Plan
        - 3.1 Unit Testing
            - 3.1.1 Instructor Class
            - 3.1.2 Leaderboard Class
            - 3.1.3 Player Class
            - 3.1.4 Data Class
            - 3.1.5 GamePlay Class
        - 3.2 Integration Testing
        - 3.3 Validation Testing
        - 3.4 System testing
        - 3.5 Test Detailing Templates for Integration, Validation, and System Testing
    - 4. Summary

1.2 Tabular Revision History

Version	Date	Author(s)	Summary of Changes
1.0 Overview	March 17th, 2024	Hailey Pong	▪ Documentation created and added on to the layout

1.0 Introduction 1.0 Test Plan 2.0 Test Plan	March 20th, 2024 - April 1st, 2024	Simran Kaur Kullar	Introduction: ▪ Overview ▪ Added description to Overview section ▪ Objectives ▪ Added description to Overview section ▪ References ▪ Added references for: ▪ Project Specification ▪ Javadoc Comments Guide ▪ JUnit Testing Guide ▪ Design Documentation ▪ Requirement Documentation ▪ Test Plan ▪ Added descriptions for the Testing Framework Process ▪ Unit Testing ▪ Integration Testing ▪ Validation Testing ▪ System Testing ▪ Added descriptions for Junit testing of Instructor class
3.0 Test Plan	April 1st, 2024	Victor Yung Wai Tan Maneet Kaur Chahal Hailey Pong Kareena Sen Simran Kaur Kullar	▪ Test Plan ▪ Added description for Junit testing of Gameplay, Player, Leaderboard and Data class - Victor ▪ Login and Player, Instructor Dashboard table - Victor ▪ Filled in Table 1.0 for Difficulty testing - Maneet ▪ Filled in Table 2.0 for Easy, Medium, Hard High Scores Table - Kareena ▪ Wrote a detailed summary for the testing documentation - Kareena ▪ Filled in Table 13.0 for Instructor Pin Login ▪ Made the term summary page - Kareena ▪ Validation, system, integration testing tables - Hailey ▪ Progress bar, achievement medals, music files testing tables - Maneet

NOTE: We were dealing with issues of pushing bitbucket, hence, Simran had to push some of our files, however, the @author tag in the files indicate who worked on that file, our code in Bitbucket is also in different branches(master, Data, main, GUI, GamePlay)

Table 1.0 Tabular Revision History

2. Introduction

2.1. Overview

This documentation outlines our strategy of validating our functionality and performance of our Capital Word Chain game. Our game is designed to enhance geographical knowledge, focusing on capital cities in North America with an engaging word chain gameplay. To ensure that our game meets both the engaging and educational objectives, and to provide a seamless and intact user experience, many testing strategies have been implemented. The testing framework has gone through a process starting with unit testing, leading to integration testing, then validation testing, and finishing off with system testing. Each of these components in the process of the testing framework focuses on the different parts of the application and software. From testing the components of our application independently to testing to validating the interaction with the application as a whole with its concurrence with the specified and decided requirements. In which one of such requirements, JUnit Testing, which is a testing framework for Java used to implement unit testing, allowed to check the validation and reliability of the code in small components and its functionality.

2.2 Objectives

The main objective of our testing strategy is to validate the functionality and its correctness by using JUnit for unit testing. Through JUnit, in the unit testing process , each method and class with business logic will be tested to validate and verify the individual components (like the logic for linking the capital cities, the hint component, and the input validation) within the application's functionality, correctness and check if they work as expected. In the integration testing process, the interaction between different components with the application like the user interface, score tracking and game logic will be tested and checked for its accuracy. This allows for the data to be accurately implemented and is intact through the application and the components integrate seamlessly with each other. In the validation testing process, this strategy confirms that the game meets all the educational as well as engagement requirements outlines in the requirements and design documentation. This includes: testing the accuracy of the educational content of the application, how efficiency of the overall learning experience and the features of user engagement. In the system testing process, this strategy will test the game as a whole ensuring its accuracy, stability, ability to perform under different situations and conditions and providing a seamless & consistent user experience. Furthermore, the white box (also known as structural) and the black box (also known as functional) testing strategies allow to discover a vast range of issues that the game can potentially encounter (ranging from logical errors to interruptions in the user experience and functionality). In conclusion, the purpose of implementing these testing strategies are for an iterative method which allows to continuously test and integrate through the construction process. This method detects any issues early on in the construction process ,while supporting the aim delivering an educational game completed and planned in quality which is both educational, informative, and engaging for users.

2.3 References

- Servos, D. (2024). CS2212 Group Project Specification - Version 1.0 - Winter Session 2024. OWL. https://owl.uwo.ca/portal/site/aa2311c9-4cef-497f-8d9c-b502023be21c/tool/c989892c-fe53-47cf-87e8-e30b144f5dd5/ShowPage?sendingPage=183877555&itemId=188127157&path=clear_and_push&title=Project%20Specification&newTopLevel=false
- baeldung. “Introduction to JavaDoc.” *Baeldung*, 28 Jan. 2018, www.baeldung.com/javadoc.
- Bechtold, Stefan, et al. “JUnit 5 User Guide.” *Junit.org*, junit.org/junit5/docs/current/user-guide/.
- JUnit Team. “JUnit 5.” *Junit.org*, 2018, junit.org/junit5/.

- Chahal, Maneet , Tan, Victor, Kullar, Simran, Pong, Hailey, Sen, Kareena. “Login Required - OWL.” *Owl.uwo.ca*, 2024, owl.uwo.ca/access/content/attachment/aa2311c9-4cef-497f-8d9c-b502023be21c/Assignments/b8fbe0f7-4085-4783-a2eb-3bb5c91ec514/Design%20Documentation_339041931d7147e588edfc1721761138-050324-0449-2648.pdf. Accessed 31 Mar. 2024.
- Chahal, Maneet , Tan, Victor, Kullar, Simran, Pong, Hailey, Sen, Kareena. “Login Required - OWL.” *Owl.uwo.ca*, 2024, owl.uwo.ca/access/content/attachment/aa2311c9-4cef-497f-8d9c-b502023be21c/Assignments/a5cff455-35af-4cdb-9cd1-bcf51f6ba526/Requirement%20Documentation_TEAM2.pdf. Accessed 31 Mar. 2024.

3. Test Plan

3.1 Unit Testing

For our application, we use unit testing to test the individual game components like the logic for chaining words, processing user inputs, and generating hints to validate their accuracy and correctness. By testing specific methods and classes which are integral to the games functionality, this can identify any errors and rectify such errors earlier on in the construction process, reducing the risk and cost associated with the complex issues that may arise later on in the construction process. Using JUnit, we automated these tests to ensure consistent and efficient verifications of our applications' foundational components. The scrutiny of the application's foundational components establish a solid foundation for a stable, error-free, and seamless user experience, with subsequent testing phases testing the game's overall performance and user satisfaction.

3.1.1 Instructor Class

The unit testing for Instructor class is automated so that it validates the key components of our application, emphasizing on user authentication, role verification, data integration and session management. The tests ensure that the user entities like the instructors which are correctly instantiated with the valid pin, with a solid foundation for secure user interactions. It verifies data loading components, specifically focusing on the authentication data and user-specific information, ensuring that the data is correctly loaded and accesses within the system. The testing tests user-specific functionalities such as processing account logins and session management to validate the systems ability to recognize and authenticate users based on their roles (player or instructor) and credentials (their id and pin). It examines the integrity of the session control, ensuring that the user sessions are correctly initiated and terminated, protecting the system access and user data. Furthermore, the testing also examines the functionality of the data retrieval of user-specific data (such as their id or if instructor – their pin). It ensures that our application can reliably fetch and display accurate information, enhancing the user experience.

3.1.2 Leaderboard Class

The Leaderboard tests sets up a test environment with predefined players and a leaderboard instance before each test, and cleans up afterwards to ensure test isolation. Tests include adding new players to the leaderboard, verifying the correct identification of the top player based on high scores, and ensuring players can be retrieved or identified as non-existent based on their ID. The tests also tests the leaderboard's ability to accurately reflect changes in players' ranks and to display the leaderboard correctly, capturing and verifying output to ensure it contains the expected player information and scores. Additional checks include validating the handling of invalid operations, such as changing the rank of a non-existent player or to an invalid position, to ensure the leaderboard remains stable and accurate.

3.1.3 Player Class

The unit test for the Player class includes key functionalities tested including updating player attributes, setting and retrieving player details(ID, PIN, high score, progress, and achievements), and administrative checks to ascertain if a player has admin privileges. methods for getting and setting player data are thoroughly tested to validate both the integrity of the data and the accuracy of the class's logic. Additionally, the tests evaluates the Player class's ability to handle high scores and achievements, including the correct assignment and retrieval of these values. Administrative roles are verified through tests that distinguish between regular and admin players. Through a series of assert statements, the tests confirm the Player class's behavior aligns with expected outcomes, ensuring robust and consistent performance across its various methods.

3.1.4 Data Class

The test for the Data class in our game validates the singleton pattern implementation and player data management, including addition, retrievals, and duplication checks. it ensures the data class consistently returns the same instance, maintaining a proper initialization state. It also test the player management functionality, confirming that players are accurately added to, retrieved from, and checked for duplicates within the games data structures. Data persistence and retrieval are examined through file operations, where the suite checks the accurate storage and loading of player data in a file. This process ensures the Data class effectively manages the games data externally and internally, maintaining consistency and reliability in the games data handling. the test cover critical functions from basic initialization to complex data interactions, aiming to ensure the durability of the games data management system.

3.1.5 GamePlay Class

The test suite is designed to validate the functionality of a game, focusing on initialization, data loading, gameplay mechanics, and state management. Initially, it ensures the game object is correctly initialized with a valid starting state, including a non-null first question and proper level setting. It then checks that essential game data, such as locations and facts, are correctly loaded, likely from a CSV file, and that the game's data retrieval methods return accurate and non-empty data sets. The tests assess the game's ability to manage levels and difficulty settings, reflecting changes in the game state. Gameplay mechanics are tested by verifying the accuracy of question and answer handling, including validating the correct scoring and tracking of incorrect answers. The suite also examines the game's ability to save and resume its state, ensuring continuity and a consistent user experience. Additionally, it tests the interaction with external data sources, like CSV files, for loading and saving game-related data, ensuring that the game manages its data correctly both internally and externally. Overall, the suite aims to cover all critical aspects of the game's operation, from basic initialization to complex game state management and data handling.

3.2 Integration Testing

Following unit testing, the Word Chain application goes through integration testing, where integrated components like word-linking logic and user interface is tested. This phase is significant for identifying any disturbances in how the components communicate, function, and are intact with one another, ensuring the game works cohesively. This testing strategy starts by testing the connections between closely related components to testing the integration of broader components of the system. This approach allows to pinpoint certain interfaces/components which may be causing integration issues, ensuring the game's components work together to provide a seamless and cohesive user experience.

3.3 Validation Testing

After completing integration testing, validation testing is used to test that the Word Chain game meets the educational and engagement objectives outlined during the planning phase. This stages included testing the educational effectiveness of the game, the user engagement, and overall usability by involving real-world scenarios and getting feedback from end-users. Validation testing is significant for confirming that the game functions as intended in a technical manner as well as fulfilling the motive of of enhancing players' geographical knowledge in an engaging manner. This phase ensures that delivers with all the requirements which where decided upon and meets users expectations.

3.4 System testing

The final stage in the testing process, system testing is used to test fully integrated Word Chain game in environment used in deployment phase. This testing evaluates the game's functionality, performance under various conditions & situations, and its behavior. The method helps to identify any issues which potentially could impact the user's experience or the stability of the system in real-world like situations. The system testing allowed us to ensure that all the functional and non-functional components performed accurately and aligned with the requirements specified in the planning stage. Using this approach allowed out game to be prepared for a successful deployment with high quality and bug-free education user experience.

3.5 Test Detailing Templates for Integration, Validation, and System Testing

Test Case Name:	**Game Difficulties and Play**
Test Case Description:	Ensure that the game adjusts its difficulty level based on the user's choice. Verify that the number of lives and point increments vary according to the selected difficulty level.
Test Steps:	1. Create fake variables within the Play GUI Main class. 2. Initialize GamePlay objects with different level integers through the constructor. 3. Call Play GUI to test the display and gameplay. 4. Ensure that the points are being incremented and decremented accurately. 5. Validate that GamePlay methods are functioning correctly. 6. Confirm that the number of lives is displayed according to the level. 7. Verify that points are incremented differently according to the level. 8. Check if the screen color changes when an incorrect input is made. 9. Ensure that the game ends when the user runs out of lives. 10. Validate that the game allows the user to complete it successfully.
Pre-Requisites:	• GamePlay methods should be fully functional. • Play should display all the required information (user ID, input box, scores, high score, number of lives)
Expected Results:	• The game adjusts its difficulty level based on the user's choice. • Number of lives displayed corresponds to the selected difficulty level. • Points are incremented and decremented accurately. • Screen color changes appropriately for incorrect inputs. • Game ends when lives run out. • User can successfully complete the game.
Test Category:	Integration Testing
Requirement:	The game should adjust its difficulty level according to the user's selection, with differences in the number of lives and point increments.
Automation:	Manually ran by human
Date Run:	*March 28st, 2024*
Pass /Fail:	*Passed*

Test Results:	The game successfully adjusted the difficulty level according to the user's choice, displaying the correct number of lives and adjusting point increments accordingly. Additionally, the screen color changed appropriately when an incorrect input was made, and the game ended when the user ran out of lives. Users were able to complete the game successfully.
Remarks:	The test results indicate that the game's difficulty levels are implemented correctly, with the appropriate adjustments made to the number of lives and point increments. This ensures a challenging and engaging gameplay experience for users across different difficulty levels. Further testing may be required to ensure consistency and reliability across various scenarios and user interactions.

Table 1.0 Game Difficulties and Play

Test Case Name:	**Easy, Medium, Hard Mode High Score Table**
Test Case Description:	*Users with the highest scores can be accessed in the varying levels (Easy, Medium, and Hard) in the high score table leaderboards.*
Test Steps:	*1. Play as multiple users (so multiple data values) at all the levels (Easy, Medium, and Hard) and check if the highest scores update in descending order on the three high-score tables.* *2. Check if the data is saving and updating for each user that plays the game* *3. Ensure that more than 5 entries are stored in each high-score table*
Pre-Requisites:	*1. User has an account that is created, enabling them to play the game* *2. To be able to play the varying levels, so that the user's high score can be updated in the high score leaderboards*
Expected Results:	*1.* Each high score table shows the top high scores for users only for those specific levels *2.* The high scores for each mode are valid and correct *3.* The high scores on the high score table leaderboard are ordered in descending order
Test Category:	*Validation test*
Requirement:	Users must be able to view the high-score table that maintains the list of top-scoring players.
Automation:	*Not automated, was manually tested while testing the game*
Date Run:	*April 1st*
Pass/Fail:	*Passed*
Test Results:	*Results = working as expected to be*
Remarks:	*Showed that the data was not saving/updating - we fixed these issues by having it save CapitalGameData.csv*

Table 2.0 Easy, Medium, Hard Mode High Score Table

Test Case Name:	**Achievement Medals**
Test Case Description:	A medal for each level should be unlocked when the user completes the level without getting any incorrect answers. The color of the medal icon will change when Boolean achievement array in the level is unlocked.
Test Steps:	*1.* Created a test player profile *2.* Play the game on easy mode without any incorrect answers *3.* Verify that the bronze achievement medal is unlocked upon successful completion *4.* Exit the game and then re-login with the same player profile *5.* Check that the bronze medal remains unlocked
Pre-Requisites:	*1. The game and player profiles should be set up and ready for testing* *2. The game must have different levels of difficulty, each with a corresponding achievement medal*
Expected Results:	*1.* A bronze medal is unlocked after completing the easy level without any incorrect answers *2.* The unlocked medals state is saved and persists across game sessions

Test Category:	*Integration Test*
Requirement:	*Play, Player class should be successfully updating achievement when unlocked*
Automation:	*Manually ran by a human.*
Date Run:	*March 31st, 2024*
Pass/Fail:	*Pass*
Test Results:	*The picture of the medal is changed when the selected achievement is unlocked.*
Remarks:	*No issues were encountered during the test. The achievement system works as expected, and the state preservation across sessions is functioning properly.*

Table 3.0 Achievement Medal Integration Test

Test Case Name:	**Login and Player, Data Integration**
Test Case Description:	*Test if the backend allows players to login successfully and retrieve player-specific data.*
Test Steps:	1. Navigate to the login screen of the application. 2. Enter valid user credentials (username). 3. Click on the login button. 4. After successful login, fetch the player-specific data from the backend.
Pre-Requisites:	- The application is installed and accessible. - The user has a valid account with correct credentials. - The backend service is up and running.
Expected Results:	- The user is successfully logged in without any errors. - The player-specific data is correctly retrieved and displayed on the frontend.
Test Category:	Integration Test
Requirement:	Users must be able to log in to their accounts and access their player data.
Automation:	*Yes (JUnit)*
Date Run:	*March 31th, 2024*
Pass/Fail:	Pass
Test Results:	*Login functionality worked as expected. Player data was successfully retrieved and matched the database records.*
Remarks:	No issues were encountered during the test. Backend and frontend integration for the login and player data retrieval is functioning correctly.

Table 4.0 Login and User Integration Case

Test Case Name:	**User Update Data After Game play 1.0**
Test Case Description:	This test case verifies whether the user's information, such as high scores, achievements, and progress, updates correctly in the data stored in the "CapitalGameData.csv" file after the user plays the game.
Test Steps:	1. Play the game as a user and achieve a high score. 2. Unlock achievements during gameplay. 3. Progress through the game by completing levels or stages. 4. Exit the game or end the gameplay session.
Pre-Requisites:	• The game must be operational and capable of updating user data during gameplay. • The "CapitalGameData.csv" file must exist and contain relevant user data fields.

Expected Results:	After gameplay, the "CapitalGameData.csv" file should be updated to reflect the following changes: • User's high score should be updated to the new achieved score. • Achievements unlocked during gameplay should be recorded in the appropriate field. • Progress made by the user, such as completed levels or stages, should be updated accordingly.
Test Category:	Integration Test
Requirement:	This test validates the requirement for the game to accurately update and store user data in the "CapitalGameData.csv" file after game play through Play GUI.
Automation:	• Manually ran by a team member
Date Run:	*March 30th, 2024*
Pass/Fail:	*Failed*
Test Results:	*Does not update the user information. The same information after the user creates an account (0s and null) remains.*
Remarks:	- The test could not be executed as planned due to the absence of a method in the data system for updating the current user's process. - Lack of this method prevented accurate verification of user information updates, such as high scores, achievements, and progress, in the "CapitalGameData.csv" file after gameplay. - It highlights the need to implement a method or mechanism within the data system to enable updating of user processes during gameplay. - Once this functionality is implemented, the test can be rerun to validate the accuracy of user data storage.

Table 5.0 Updating User's Data Integration Test

Test Case Name:	**User Update Data After Game play 2.0**
Test Case Description:	The objective of this test case is to verify whether the user's gameplay data updates correctly in the CSV file after the game play.
Test Steps:	1. Simulate user gameplay by playing the game and achieving a certain score. 2. Confirm completion of gameplay and data update. 3. Check the content of the CSV file to verify the presence and accuracy of the updated user gameplay data.
Pre-Requisites:	• The game system must be operational. • Login, player and data class must be creating a data list of users and append it on CSV file.
Expected Results:	• After gameplay, the CSV file should be updated with the user's gameplay data, including score and any other relevant information. • The updated data should be appended to the CSV file without duplicating existing records.
Test Category:	*Integration test*
Requirement:	Validation of the system's capability to update user gameplay data accurately in the CSV file.
Automation:	• Manually ran by a human and used JUnit test to test just the data methods.
Date Run:	*March 30th, 2024*
Pass/Fail:	*Fail*
Test Results:	• The test failed as it does not check if the user already exists and appends the data again to the CSV file.
Remarks:	To ensure data integrity and prevent duplication, the system should first check if the user already exists in the CSV file before appending new gameplay data. Implementing this check will prevent redundant data entries and maintain the CSV file's integrity.

Table 6.0 User Update Data After Game Play 2.0

Test Case Name:	**User Update Data After Game play 3.0**
Test Case Description:	This test case verifies whether the user's information (high score, progress and achievements) updates correctly in the "CapitalGameData.csv" file after gameplay.
Test Steps:	1. Play the game as a user and achieve a certain score. 2. Confirm completion of gameplay and data update. 3. Check the content of the "CapitalGameData.csv" file to verify the presence and accuracy of the updated user information.
Pre-Requisites:	• The game system must be operational. • Login, player and data class must be creating a data list of users and append it on CSV file.
Expected Results:	• After gameplay, the "CapitalGameData.csv" file should be updated with the user's information, including score and any other relevant data. • The updated data should be incorporated into the CSV file without altering other users' information.
Test Category:	*Integration test*
Requirement:	Validation of the system's capability to update user information accurately in the "CapitalGameData.csv" file after gameplay.
Automation:	• Manually ran by a human and used JUnit test to test just the data methods.
Date Run:	*March 30th, 2024*
Pass/Fail:	*Pass*
Test Results:	The test successfully updates the "CapitalGameData.csv" data with the user's updated information without altering other users' info.
Remarks:	The system effectively handles the updating of user data in the CSV file after gameplay, maintaining data integrity and ensuring accurate records for each user.

Table 7.0 User Update Data After Game Play 3.0

Test Case Name:	**Play GUI and GamePlay Saving current game process on CSV file**
Test Case Description:	Test if GamePlay saves the user's last chain in csv that will be needed for loading the game when the user wants to play it later and Play implements calling saving game methos correctly to automatically save game every time the user finishes a word chain.
Test Steps:	1. Create a fake user ID and Player object for running the test. 2. Play the word chain to test if the game works, which it did, and successfully updated the variables such as high score, score, lives left on Play GUI. 3. SaveGame successfully created a CSV with the user ID. 4. Verify that the user level is stored and updated successfully but deletes the other levels, hence only has one row of one of the level game play, not all three levels played. 5. Manually edited the CSV file to include user's process for other levels: a. 0, 43, New York, 5 (easy mode game play) b. 1, 3, Alberta, 3 (normal mode game play) c. 2, 13, Ohio, 1 (hard mode game play) 6. Play the game again which updated the variables accordingly on easy level. 7. Check CSV file, only has one row of current game. a. 0, 23, Ottawa, 4

Pre-Requisites:	Main methods in GamePlay (backend) and Play GUI (frontend) must be working. The word chain mechanics should be successfully updating the user's score and other information (unlocking a medal, etc.). Determine whether the user's input is valid. Play GUI should be displaying the frame when run.
Expected Results:	Create UserName.CSV file if not already existed on your computer, and store/update the last game played. It should include the level number, current score, last chain word the user was on, and lives left in this order. It should have a maximum of three rows, in the order of easy (0), normal (1), hard (2) on each row with the game information as mentioned previously. When the user plays the next game, it should still have the same previously played game on the other level. If the user has not played the next level yet, it should be empty. E.g., A user who has played easy and normal mode should have the following output stored on their CSV file: 0, 14, Edmonton, 4 1, 29, Alberta, 3
Test Category:	*Integration test*
Requirement:	Game's saving mechanism efficiency and accuracy in preserving user progress in a CSV file.
Automation:	• *Manually ran by team member*
Date Run:	*March 26th, 2024*
Pass /Fail:	*Failed*
Test Results:	• Overwrites on CSV file. • Only writes one row of current process on the first row CSV and deletes the previously played game level.
Remarks:	- Critical flaw identified in the game's saving mechanism. - User's progress not accurately stored in the CSV file. - CSV file overwritten with only the latest game progress. - Failure in the game's logic for updating and appending data. - Thorough investigation required to rectify the issue. - Consideration of separating user data and gameplay progress for efficiency.

Table 8.0 Game Save on CSV Integration Case

Test Case Name:	**Play/Load GUI and GamePlay Save Game and Load Game**
Test Case Description:	Test if GamePlay saves the user's last chain in csv that will be needed for loading the game when the user wants to play it later and Play implements calling saving game methos correctly to automatically save game every time the user finishes a word chain.

Test Steps:	Case 1) New User - No Previously Played Game 1. Create a fake user ID and Player object for running the test. 2. Play the word chain to test if the game works, which it did, and successfully updated the variables such as high score, score, lives left on Play GUI. 3. SaveGame successfully created userID.CSV. 4. Verify that the user level is stored and updated successfully. 5. Manually edited the CSV file to include user's process for other levels: a. 0, 102, Regina, 3 (easy mode) – Completed easy mode b. 1, 32, Toronto, 4 (normal mode) 6. Save the level game without deleting the previous level data. 7. Load GUI displayed the level, score and lives left on designated slot. Case 2) Existing User - Load Game & Save Game 1. Logged in as a previously existing user. 2. Load previously played data from a slot. Successfully loaded the game with previous scores, chain and lives left on Play GUI. 3. Finish normal level and played on hard level. 4. Save normal level and hard level on CSV without erasing other data. The game form easy level also existed on CSV file.
Pre-Requisites:	• GamePlay (backend) and Play GUI (frontend) must be working correctly. • GamePlay should successfully update scores, high score, achievements, and deduct lives as required. • Word chaining mechanism should be operational.
Expected Results:	*CSV with the name of user's id is created and stores the current level, scores, current chain, lives left in order from left to right.*
Test Category:	*Integration Test*
Requirement:	• Play GUI - calls GamePlay methods to save the current chain • Load GUI - calls GamePlay reading and storing method to load the data. • GamePlay reading and storing (loading game), saveCSV (saving current process) methods
Automation:	• *Manually ran by team members*
Date Run:	*March 30th, 2024*
Pass/Fail:	*Passed*
Test Results:	*Create a csv file with the name of user's id and saves the current process on csv. Easy mode on row 0, Normal mode on row 1, Hard on row 2.*
Remarks:	The test failure stems from the game's inability to accurately save the user's progress within the CSV file. The issue lies in the game's failure to update the CSV file with the pertinent game data. Consequently, the CSV file is disregarding the existing information and replacing it with the current process, resulting in only one row reflecting the gameplay of the current level. This indicates a critical flaw in the game's data persistence mechanism, specifically with regards to CSV handling. A thorough rewrite of this functionality is imperative to ensure accurate progress tracking and retention of gameplay data.

Table 9.0 Play GUI and GamePlay Save Game and Load Game

Test Case Name:	***Data and GamePlay Regression Test***
Test Case Description:	Check if the change in Data class and GamePlay changes
Test Steps:	1. Update the Data class with the latest changes. 2. Modify the GamePlay functionality according to the updated Data class. 3. Execute a series of test scenarios, including typical gameplay actions and edge cases. 4. Record any deviations or unexpected behavior observed during testing.
Pre-Requisites:	• Ensure that the development environment is set up correctly. • Verify that the Data class modifications have been properly documented and reviewed. • Confirm that the GamePlay changes are aligned with the updated Data class requirements.
Expected Results:	• Successful execution of the test scenarios without any errors or exceptions. • Consistent and expected behavior observed during gameplay actions. • Proper integration between the Data class and GamePlay functionality.
Test Category:	*Integration Test*
Requirement:	The Data and GamePlay GUI components must be seamlessly integrated to ensure efficient data management and smooth user interaction.
Automation:	Manually run by a human.
Date Run:	*April 1st*
Pass /Fail:	*Pass*
Test Results:	The changes made to the Data class and GamePlay were successfully integrated, and all test scenarios executed without errors. The gameplay actions produced expected outcomes, and there were no deviations observed during testing.
Remarks:	The regression test was conducted thoroughly, covering various gameplay scenarios and edge cases. The modifications to the Data class and GamePlay were implemented seamlessly, indicating good code compatibility and integration. Moving forward, it's recommended to continue monitoring the performance of these components and conducting periodic regression tests to ensure ongoing stability and reliability.

Table 10.0 Play and GamePlay Regression Test

Test Case Name:	**Progress Bar**
Test Case Description:	Assesses the functionality and accuracy of the progress bar in displaying current status to the user
Test Steps:	1. *Start new game file* 2. *Complete Each Level and observe the bar increase as each level is completed* 3. *Continue process until complete* 4. *Note progress bars final state when each level is completed*
Pre-Requisites:	1. User must have an account 2. User must complete the lower level before having access to higher levels

Expected Results:	1. progress bar begins empty 2. progress bar increments according to the games progression 3. progress bar becomes full when all levels are completed
Test Category:	*Integration*
Requirement:	*The progress bar should accurately reflect the current status of the game, providing feedback to the user from start to finish.*
Automation:	*Manual*
Date Run:	*April 1st*
Pass/Fail:	*Pass*
Test Results:	*Progress bar accurately reflects the game status of the current user*
Remarks:	No issues were encountered during the test. Backend and frontend integration for the progress bar is functioning correctly.

Table 11.0: Progress Bar Testing

Test Case Name:	**Instructor Dashboard**
Test Case Description:	This test case verifies that the Instructor Dashboard correctly displays student progress metrics, including current scores and progression. Issues to be tested include the orientation of data presentation (horizontal vs. vertical), accurate storage of levels, and correct retrieval of player scores.
Test Steps:	1. Log in to the Instructor Dashboard with the correct pin. 2. Verify that the dashboard displays player data in a horizontal format. 3. Check that the levels are stored and displayed properly for each player. 4. Confirm that the player scores are accurately retrieved and displayed. 5. (Optional) Verify additional metrics like number of attempts, time spent, and aggregated statistics, if implemented. 6. Log out and ensure that access to the dashboard is secured and requires re-authentication.
Pre-Requisites:	- The Instructor Dashboard feature is implemented and accessible. - There are existing player profiles with progress and scores to display. - The system has predefined test data to display on the dashboard.
Expected Results:	- Data on the dashboard is displayed horizontally. - Levels are correctly stored and reflected for each player in the dashboard. - Player scores are accurately fetched and shown. - Additional metrics, if present, are correctly calculated and displayed. - Dashboard access is securely controlled by a password.
Test Category:	*System testing*
Requirement:	- The Instructor Dashboard should accurately display each player's current score and progression, including completed levels/stages. - The dashboard should be pin-protected to restrict access to instructors only.
Automation:	*Manual*
Date Run:	*April 1st, 2024*
Pass/Fail:	*Pass*
Test Results:	Initally there were many errors/bugs with the dashboard, however after editing code, changing path files, the dashboard is now working as expected. It will display the progress of students (or users who are stored in data file).
Remarks:	The data was printing vertically not horizontally, it was not storing the levels properly, and it was not getting the player's scores. To fix this, the array was fixed by flipping the conditions around and fixing the getters that it was calling.

Table 12.0: Instructor Dashboard Testing

Test Case Name:	**Instructor Pin Login**

Test Case Description:	*This test was issued to ensure that the instructor can access the dashboard through a pin exclusive to the instructor.*
Test Steps:	1. Enter the initialized pin (4351) 2. Enter an uninitialized pin (ex. 43518) to ensure that the error popup message shows 3. If the correct pin is entered, then it should lead the instructor to the instructor dashboard popup window
Pre-Requisites:	- The instructor knows the proper pin - The backend service is up and running.
Expected Results:	- The instructor is successfully logged in without any errors. - The instructor can access the instructor dashboard with no errors
Test Category:	Integration Test
Requirement:	Instructors must have an exclusive method of logging in through a special username or pin. The instructor should also be able to access the student dashboard.
Automation:	*Manual*
Date Run:	*April 1st, 2024*
Pass /Fail:	Pass
Test Results:	*Login functionality worked as expected. Instructor data was successfully retrieved and matched the database records from the initialized backend.*
Remarks:	Small issues were encountered during the test. For instance, the pin can only be entered with the buttons. Also, the instructor couldn't access the instructor dashboard due to a NullPointer exception. Our team was able to fix these issues and now the backend and frontend integration for the instructor data retrieval is functioning correctly.

Table 13.0 Instructor Pin Login

Test Case Name:	**Music Files**
Test Case Description:	Ensuring that the music files were working and the sound played as it was intended to.
Test Steps:	1. Open game and start the game 2. Ensure that the main menu music plays 3. Enter a correct word to check if the sound file will play if answer is correct 4. Enter an incorrect word to check if the sound file will play if the answer is incorrect 5. Test on all modes (easy, medium, hard)
Pre-Requisites:	1. User must have an account 2. User must be logged in
Expected Results:	*Music should work in all cases (incorrect and correct answers entered), and main menu should present music*
Test Category:	*System testing*
Requirement:	*Feedback Systems: The UI must provide visual or auditory feedback in response to user actions,* *such as clicking buttons or entering commands, to confirm that the desired action has been taken. Music responses to correct or incorrect answers.*
Automation:	*Manually run by human*
Date Run:	*April 1st, 2024*
Pass/Fail:	*Pass*
Test Results:	*Passed, the sound works as intended do*

Remarks:	*Remarks and comments regarding the last time the test was executed (e.g. explaining why the test failed).*

Table 14.0: Music Files Testing

Test Case Name:	**Final Project Validation Test 1.0**
Test Case Description:	This test case is designed to validate the final project deliverable by ensuring that all the specified features and functionalities meet the requirements and expectations. It covers the comprehensive testing of the entire system to ensure its correctness, reliability, and usability. The test will verify that the final project fulfills all the defined objectives and criteria set forth during its development.
Test Steps:	*1.* **Functional Testing:** • Verify that all features listed in the project requirements are implemented correctly. • Test each feature to ensure its functionality meets the expected behavior. • Confirm that user interactions with the system produce the intended results. *2.* **UI/UX Testing:** • Evaluate the user interface design for clarity, consistency, and intuitiveness. • Test the responsiveness of the UI across different devices and screen sizes. • Ensure that user inputs are validated and appropriate error messages are displayed when necessary. • Validate that all UI elements are properly aligned and visually appealing. *3.* **Performance Testing:** • Assess the system's performance under normal and peak loads. • Measure response times for key functionalities and transactions. • Check for any memory leaks or performance bottlenecks that could affect the system's responsiveness. *4.* **Security Testing:** • Verify that user data is securely stored and transmitted. *5.* **Documentation Review:** • Review all project documentation, including user manuals, installation guides, and technical specifications. • Ensure that the documentation is comprehensive, accurate, and up-to-date with the final implementation. • Verify that any external dependencies or third-party libraries are properly documented.
Pre-Requisites:	• Complete development and integration of all project features. • Finalization of UI/UX design and implementation. • Deployment of the project to a testing environment.
Expected Results:	• All test cases pass without any critical issues or defects. • The final project meets all specified requirements and objectives. • The system performs reliably, efficiently, and securely under different conditions. • Documentation is comprehensive and provides clear guidance for users and developers. • CapitalGameData.csv should be created to store user data. • userID.csv should be created to store current process. • UI should update the display whenever necessary (scores, high score, etc) • Successfully runs the game mechanics without facing errors.
Test Category:	*Validation Test*
Requirement:	The final project must include a functional leaderboard feature, debugging mode, educational abilities, save/load game functionality, achievement system, user information display, tutorial availability, progress visualization, and multiple difficulty stages. Additionally, comprehensive documentation, including user manuals, installation guides, and technical specifications, must be provided. The project should store user data in `CapitalGameData.csv` and save user progress in `userID.csv`. The user interface should be intuitive, visually appealing, and responsive across different devices and screen sizes.

Automation:	• Manually ran by the team
Date Run:	*April 1th, 2024*
Pass/Fail:	*Pass*
Test Results:	- Leaderboard functionality orders players by high scores for each level - Debugging mode is available and allows the admin to navigate through the game without obstacles, with all levels unlocked - Educational abilities are integrated, enhancing geographical knowledge - Ability to save and load previously played games - Achievements are unlocked when criteria are met, providing a sense of completion to the user - Lives are deducted successfully when the user inputs the wrong answer - User information is displayed throughout the game - Tutorial is available for users with no experience - Progress is visually displayed - Game information is saved for each user - Different stages of difficulty are implemented to provide varying levels of challenge
Remarks:	- The final project validation test revealed that most of the functional requirements have been successfully met. - A clean and intuitive user interface (UI) was implemented, enhancing user experience and usability. - While most functional requirements are fulfilled, it's essential to conduct a thorough review to ensure all aspects of the project align with the specified objectives and criteria. - Any remaining functionalities that have not been fully implemented should be addressed to achieve completeness and user satisfaction. - Overall, the project shows promise and readiness for deployment, pending the resolution of any outstanding issues identified during testing.

Table 15.0 Final Validation Test

Test Case Name:	***Final Project System Testing***
Test Case Description:	The objective of this test case is to perform comprehensive system testing to ensure the final project meets all requirements and functions as intended.
Test Steps:	1. Navigate through all features and functionalities of the final project. 2. Test user interactions with the system, including input validation and error handling. 3. Test performance under normal and peak loads. 4. Review security measures and test for vulnerabilities. 5. Evaluate compatibility with different browsers, operating systems, and devices. 6. Test accessibility features for compliance with accessibility standards. 7. Verify documentation for completeness and accuracy.
Pre-Requisites:	• The final project must be fully developed and deployed. • Test environment should be set up and ready for testing. • Packaged all the classes/codes.
Expected Results:	• All features and functionalities of the final project should work correctly without any critical issues. • The system should perform reliably, efficiently, and securely under various conditions. • Compatibility should be ensured across different platforms and devices. • Accessibility features should comply with relevant standards. • Documentation should provide clear guidance for users and developers.

Test Category:	*System Test*
Requirement:	All classes within the system must be operational and perform their designated functions accurately to produce desired output.
Automation:	Manual testing by human testers.
Date Run:	*April 1st, 2024*
Pass /Fail:	*Pass*
Test Results:	The test passed overall, but with one exception. The system was able to successfully navigate through all features and functionalities, including user interactions, input validation, error handling, and performance testing under normal and peak loads. Security measures were reviewed, and no vulnerabilities were detected. Compatibility was ensured across different browsers, operating systems, and devices. Accessibility features complied with relevant standards. Documentation was found to be complete and accurate. However, the Leader dashboard feature did not function as expected.
Remarks:	The failed functionality with the Leader dashboard warrants further investigation and resolution. It's important to identify the root cause of the issue and implement the necessary fixes to ensure that all aspects of the final project are fully functional. Additionally, thorough testing and validation should continue to be conducted to maintain the reliability and effectiveness of the system.

Table 16.0 Final System Testing

Test Case Name:	***Window System Testing***
Test Case Description:	Tests if .JAR file of project is executable on Window 10
Test Steps:	1. *On Window 10 laptop, download WorldChain.JAR file.* 2. *Follow the steps on README for installation detail.* 3. *Create/login and test each software page/UI and functionalities.* 4. *Exit the game.* 5. *Login again and check if the user data is saved and updated.*
Pre-Requisites:	• Must be running on Window 10 • Downloaded WorldChain.JAR file
Expected Results:	• Successfully run the software without facing errors.
Test Category:	*System test*
Requirement:	• *All the codes completed and tested* • *Executable WorldChain.JAR file*
Automation:	*Manually ran by human*
Date Run:	*April 2nd, 2024*
Pass/Fail:	*Pass*
Test Results:	*The output or results of the test.*
Remarks:	*Remarks and comments regarding the last time the test was executed (e.g. explaining why the test failed).*

Table 17.0 Window System Testing

4. Summary

The document details the development and testing for the "Capital Chain", an educational word chain game focusing on North American capital cities and states. In this document, we outline the objectives, emphasizing the testing framework: unit, integration, validation, and system testing. Our unit testing ensures the individual component functionalities and the integration testing checks any component interaction. The validation testing supports educational and engagement requirements, and any system testing validates overall performance. The detailed test plan maps classes like Instructor, Leaderboard, Player, Data, and Gameplay, with specific test classes outlined. Our key testing ensures that terminologies are condensed for clarity. In summary, this document serves as a strategy for developing and testing Capital Chains to ensure a high-quality educational gaming experience for learners.

Terms	Definition
Unit Testing	• A way of testing a unit (the smallest part of the code that can be isolated within a system) • Unit testing is a software technique where individual units or components of a software application are tested • It focuses on small-scale testable parts of a software application, isolating it from the rest of the code • It ensures and verifies that each unit of the software performs functionally and as intended • Unit testing enables the unit to be tested in pure isolation, ensuring that any external factors do not affect the isolated units • Unit tests also serve as regression tests, which ensure that refactoring or changes in the codebase do not introduce any new bugs • Various testing frameworks support unit testing such as JUnit or NUnit
Integration Testing	• Validation testing is a type of software testing that confirms that the product meets the projected requirements and specifications • It illuminates on testing the end-to-end functionality and usability of the software from the perspective of the user • Validation testing ensures that the software meets the stakeholder's expectations, requirements, and validating it so that it provides proper performance and value as intended • It includes acceptance testing, system testing, usability testing, and user acceptance testing
Regression Testing	• Regression testing is a type of software testing that ensures whether any recent code changes affect existing features or functionalities • It illuminates on retesting the fixed parts of the software along with the unaffected areas to make sure there are no new defects that have been introduced • It also helps with maintaining any software quality and stability by making sure that any changes do not lead to unintended regressions or problems • The technique for regression testing will include manual regression testing, automated regression testing, and selective regression testing
JAR	A JAR file is a type of package used in Java programming. It bundles together multiple Java class files, along with related data and resources, into a single file for easy distribution. JAR files are similar to zip files and always have a .jar extension.

Table 17.0 Terminology