Project Title: System Verification and Validation Plan for Software Eng

Team #11, Mac-AR
Student 1 Matthew Collard
Student 2 Sam Gorman
Student 3 Ethan Kannampuzha
Student 4 Kieran Gara

November 18, 2023

Revision History

Date	Developers	Notes
2023/10/30	Matthew	Work on Part 1+2
2023/10/30	Ethan	Work on Part 3
2023/11/01	Sam	Work on non-functional tests
2023/11/01	Kieran	Work on functional tests
2023/11/01	Ethan	Work on functional tests
2023/11/01	Matthew	Work on non-functional tests
2023/11/02	All	Traceability and final revisions
2023/11/03	All	Reflections

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

All of our symbols, abbreviations, and acronyms are in section 1.4 in the SRS.

This document will record all the plans of the MacAR group with respect to verification and validation. It will list all the objectives of the verification and validation as well as the plan going forward and the system level tests and unit tests we are going to perform.

2 General Information

2.1 Summary

The purpose of this document is to outline the plan to evaluate and verify the MacAR augmented reality game. The game consists of the users creating a lobby together, and entering into a virtual escape room using their camera. All the puzzles in the escape room will be cooperative and require multiple users and perspective to solve.

2.2 Objectives

The main objective of the VnV plan is to plan out how the system's correctness will be tested. It outlines the plan to ensure that the system implementation matches the project specifications that are stated in the other documents(listed in 2.3). The plan for verifying the system's non-functional requirements and usability is also in this document. These objectives are important so the stakeholders can trust the system, and enjoy their gaming experience. Additionally, external libraries used in the implementation of the application will be assumed to have already been verified by their implementation team.

2.3 Relevant Documentation

The Project consists of various other documentation that contain contents discussed and mentioned in this document

- Problem Statement
- Development Plan
- Software Requirements Specification (SRS) Document
- Hazard Analysis(HA)

- Verifcation and Validation Plan (VnV Plan)

The problem statement explains the initial problem this program intends to solve, as well as the overall goals and intended features of the game. This document is relevant because the original goals and intentions of the program are listed, and the rest of the documents have to be in line with this original document, or it should be updated to reflect the changes in other documents.

The development plan will be referenced for clarity on team member roles and responsibilities when it comes to testing.

The SRS document will be used to ensure all of our functional and non-functional requirements are accounted for and have a place in the verification and validation plan.

The HA document will be used to further verify that all the safety requirements are met, and the verification and validity of the program adheres to the concerns referenced here.

The VnV Plan will be used to make a plan to verify and validate every design decision we make.

3 Plan

This section describes the members of the verification and validation team as well as plans for SRS, Design, and Verification and Validation verification. Furthermore, this section explains how the verification plan will be implemented along with the testing tools and plans for software validation.

3.1 Verification and Validation Team

Table 1: Verification and Validation Team Members Table

Name	**Role(s)**	**Responsibilities**
Matthew Collard	Lead test developer, Fullstack test developer, Manual tester	Lead test development process, Create automated tests for backend code, Manually test UI and game functionality. Main verification reviewer for the SRS document.
Sam Gorman	Fullstack test developer, Manual tester, Code Verifier	Create automated tests for backend code, Manually test UI and game functionality, Make sure source code follows project coding standard. Main verification reviewer for the Hazard Analysis document.
Ethan Kannampuzha	Fullstack test developer, Manual tester, Code Verifier	Create automated tests for backend code, Manually test UI and game functionality, Make sure source code follows project coding standards. Main verification reviewer for the Verification and Validation document.
Kieran Gara	Fullstack test developer, Manual tester	Create automated tests for backend code, Manually test UI and game functionality. Main verification reviewer for MIS document.
Dr. Irene Yuan	Supervisor, SRS verifier, Final reviewer	Make sure SRS meets requirements of the project, Validate documentation. Because Dr. Yuan is the supervisor of this project, she will be able to validate the requirements.

The following roles and responsibilities are a starting point for how testing process will be done. If there are any updates to member roles, the table will be updated to reflect this.

3.2 SRS Verification Plan

The current plan to verify the SRS involves using the ad hoc feedback from peer reviewers (ie. Team 2 and Team 18) and our TA, and making changes based on the feedback from their static testing (documentation walk through). The peer reviewers file GitHub Issues for any issues they see, and it is our teams responsibility to go through the issues and make changes to the SRS based on the suggestions they have made. Furthermore, our team will create issues related to feedback we receive from the TA. When an issue has been dealt with, it will subsequently be closed on GitHub Issues. Additionally, the supervisor of the project, Dr. Irene Yuan, will also be involved in the SRS verification process. The team will spend some time doing an documentation walk through with Dr. Yuan through the requirements document in order to get her feedback on it. GitHub Issues will be created to address the issues brought up by Dr. Yuan, and once these issues are dealt with they will be closed.

3.3 Design Verification Plan

The plan for design verification involves using feedback received from peer reviewers and TA and making changes based on their feedback. This will be done through the peer reviewers creating GitHub Issues and our team creating issues based on TA feedback. Our team will then go through the issues and make changes accordingly. Additionally, the design of the system will be verified using the MG and MIS. We will compare the modules listed in these documents with the requirements from the SRS in order to make sure that all requirements are implemented in one or more modules.

3.4 Verification and Validation Plan Verification Plan

Similarly to the previous plans, the plan for verification and validation plan verification also involves using feedback from peer reviewers and TA, and making changes based on their feedback. This will be done through GitHub Issues and once issues are resolved, the subsequent GitHub Issue will be closed. Additionally, in order to make sure that the verify that the verification and validation plan relates to all requirements, the requirements in the SRS will be mapped to the tests listed in this document. This will ensure that all requirements have a test that is associated with them.

3.5 Implementation Verification Plan

To verify the implementation, we will run through only our automated test cases in 4.1, and 4.2 with every version of the code to verify that everything is working as intended. We only run automated to preserve resources and time. For major version updates, all the manual and automated test cases in 4.1 and 4.2 will be performed. This full test suite will cover all of our requirements and verify that our program is working as intended by documentation.

For every major version update, the developers and Dr. Irene Yuan will perform a code walk-through and go into detail with what the documentation intentions are versus what is written in the code, and what the stakeholders want. At the same time the other static tests mentioned in Test-MS1 and Test-LR1 will be performed.

3.6 Automated Testing and Verification Tools

Refer to Section 6 of the Development Plan for an up-to-date list of tools.

3.7 Software Validation Plan

Weekly meetings with the project supervisor, Dr. Irene Yuan, will serve to validate the software and its driving requirements (See section 3.2). The meetings will serve to iteratively check that new software follows what was expected from the project.

4 System Test Description

4.1 Tests for Functional Requirements

Tests for functional requirements will be broken up into the following subsections due to tests and functionality being related: Game Room/Lobby Testing, Start Game Testing, Save/Load Game Testing, Puzzle Interaction Testing, and Messaging Testing. The tests in these subsections can each be mapped to functional requirements described in the SRS.

4.1.1 Game Room/Lobby Testing

The following section deals with tests for creating and joining game rooms, as well as editing game room settings. The functional requirements that are covered by this subsection are CG1, CG2, CG3, CG4, CG5, JG1, JG2, JG3, JG4, JG5, RS1, RS2, RS3, RS4, RS5, RS6, ER1, ER2, ER3.

1. **Name:** Create game room menu present

   Id: Test-CR1

   Type: Manual

   Initial State: User has application open.

   Input: User presses Host button.

   Output: User redirected to create game menu which has entries to set game room settings (game room name, game room capacity, game room password).

   Test Case Derivation: System should allow user to have the option to create a game room with specified settings, and the menu being present is the first step for this.

   How test will be performed: Tester will manually press Host button and verify that they are redirected to the create game menu.

   Requirements being verified: CG1

2. **Name:** Successful creation of game room with name

   Id: Test-CR2

   Type: Automated

   Initial State: Create game menu open.

   Input: Game room name inputted.

   Output: Game room with specified name is created in database.

   Test Case Derivation: System should allow users to set a name for their game room.

   How test will be performed: Create automated test that creates a game room name and verify that the game room is present in the database.

   Requirements being verified: CG1 CG2

3. **Name:** Unsuccessful creation of game room without name

   Id: Test-CR3

   Type: Automated

   Initial State: Create game menu open.

   Input: Nothing inputted for game room name.

   Output: Game room is not created and error message tells user that game room name is not valid.

   Test Case Derivation: System requires users to specify a game room name so that other users can find the game room.

   How test will be performed: Create automated test that creates a game room with empty string and verify that the game room is not created and error message is present.

   Requirements being verified: CG1 CG2

4. **Name:** Creation of game room with certain capacity

   Id: Test-CR4

   Type: Manual

   Initial State: Create game menu open.

   Input: User slides game room capacity slider to adjust game room capacity (min 2, max currently unknown).

   Output: Game room with specified capacity is created in database.

   Test Case Derivation: System should allow users to create a game room with a certain capacity to allow users to set the amount of users in their game room.

   How test will be performed: Tester will manually select a capacity for the game room in one instance, and in another instance check to see if the game room exists and the capacity matches the set capacity.

   Requirements being verified: CG1 CG3

5. **Name:** Creation of game room with password

   Id: Test-CR5

   Type: Manual

Initial State: Create game menu open

Input: User inputs game room password.

Output: Game room requiring password specified by user is created in database.

Test Case Derivation: System should allow users to set a password to their game room to prevent random users from joining.

How test will be performed: Tester will manually input game room password on one instance, and in another instance check if the game room exists and that they need to correctly enter the password to enter the game room.

Requirements being verified: CG1 CG4

6. **Name:** Creation of game room with empty password

   Id: Test-CR6

   Type: Manual

   Initial State: Create game menu open

   Input: User does not input anything for game room password.

   Output: Game room requiring no password is created in database.

   Test Case Derivation: System should allow users to create a game room without a password if they would like to (allows for random people to join).

   How test will be performed: Tester will manually input nothing for game room password on one instance, and in another instance attempt to join, and check if it requires them to enter a password.

   Requirements being verified: CG1 CG4

7. **Name:** Game room menu present after creating game room

   Id: Test-CR7

   Type: Manual

   Initial State: Create game menu open and all valid settings set.

   Input: User presses create game room.

   Output: Game room is created and user is redirected to game room menu which displays the users present in the game room.

Test Case Derivation: System should redirect users to game room menu once they have inputted their settings in the create game room menu.

How test will be performed: Tester will manually press create game room and verify that they are redirected to the game room menu.

Requirements being verified: CG1 CG5

8. **Name:** Join game room menu present

   Id: Test-JR1

   Type: Manual

   Initial State: User has application open.

   Input: User presses Join Game button.

   Output: User redirected to join room menu which lists all present game rooms in database that are available to be joined and are under capacity.

   Test Case Derivation: System should allow users to join game rooms that are not at max capacity.

   How test will be performed: Tester will manually press Join Game button and verify that they are redirected to the join game menu.

   Requirements being verified: JG1 JG5

9. **Name:** Join game room requiring no password

   Id: Test-JR2

   Type: Manual

   Initial State: Join game menu open.

   Input: User enters room name of a created room that has no password set.

   Output: User joins game room.

   Test Case Derivation: System should allow users to join game rooms that have no password set.

   How test will be performed: Tester will manually enter room name and verify that they can join without needing to enter a password.

   Requirements being verified: JG1

10. **Name:** Join game room requiring password

    Id: Test-JR3

    Type: Manual

    Initial State: Join game menu open.

    Input: User enters room name of a created room that has a password set.

    Output: User joins game room and is prompted to enter the game room password.

    Test Case Derivation: System should allow people to join game rooms that have a password set if they enter the correct password.

    How test will be performed: Tester will manually enter room name and verify that they are prompted to enter the game room password.

    Requirements being verified: JG1 JG2

11. **Name:** Game room info updated when user joins

    Id: Test-JR4

    Type: Automated

    Initial State: Join game menu open.

    Input: User enters room name and password of created game room.

    Output: User joins game and available capacity decreases by 1.

    Test Case Derivation: System should update game room available capacity based on the current amount of people in the game room.

    How test will be performed: Create automated test that simulates a user joining a room and checks that available capacity decreases by 1.

    Requirements being verified: JG1 JG3

12. **Name:** Game room menu present after joining game

    Id: Test-JR5

    Type: Manual

    Initial State: Join game menu open.

    Input: User enters room name and password of created game room.

Output: User is redirected to game room menu which displays the users present in the game room.

Test Case Derivation: System should redirect users to game room menu once they have joined the game room.

How test will be performed: Tester will manually join a game room and verify that they are redirected to the game room menu.

Requirements being verified: JG1 JG4

13. **Name:** Edit game room settings menu present

    Id: Test-RS1

    Type: Manual

    Initial State: Game room menu is present.

    Input: User presses settings button.

    Output: User is redirected to edit game settings menu which displays the current settings of the game room (ie. room capacity, and password).

    Test Case Derivation: System should allow users to correct their game room settings in case they put in incorrect values initially.

    How test will be performed: Tester will manually enter a game room and press the settings button to verify that they are redirected to the edit game settings menu.

    Requirements being verified: RS1 RS2

14. **Name:** Edit game room password

    Id: Test-RS2

    Type: Manual

    Initial State: Edit game room settings menu present.

    Input: Password to game room is updated.

    Output: Game room is updated to require the new password.

    Test Case Derivation: System should allow users to edit the password to their game room to ensure security.

How test will be performed: Tester will manually update game room password in edit game room settings menu and verify that the game room in the database now requires the new password to join.

Requirements being verified: RS1 RS2 RS3

15. **Name:** Edit game room capacity

    Id: Test-RS3

    Type: Manual

    Initial State: Edit game room settings menu present.

    Input: Capacity value for game room is updated.

    Output: Game room capacity is updated.

    Test Case Derivation: System should allow user to change the number of users that can join the game room in case they want to add more or less users.

    How test will be performed: Tester will manually update game room capacity value in edit game room settings menu and verify that the game room in the database now allows capacity value of users to join the game room.

    Requirements being verified: RS1 RS2 RS4

16. **Name:** Remove user from game room

    Id: Test-RS4

    Type: Manual

    Initial State: Host user (creator of game room) is in game room menu with one or more other users, and edit game room settings menu is present.

    Input: Host user selects users name they want to remove, and presses kick button.

    Output: User selected by host is removed from the game room and current number of users present in game room decreases by 1.

    Test Case Derivation: Host user should be able to remove any user from the game room.

How test will be performed: Tester will manually select user they want to remove, press kick button, and verify that the user has been removed from the game room.

Requirements being verified: RS1 RS2 RS5

17. **Name:** Display updated game settings

    Id: Test-RS5

    Type: Manual

    Initial State: Edit game room settings menu present.

    Input: Game room settings are changed.

    Output: Game room settings menu is updated to show current setting values.

    Test Case Derivation: When changing a setting, all users in a game room should should be able to see the updated settings.

    How test will be performed: Tester will manually update game settings and verify that game room setting menu has been updated to show updated values.

    Requirements being verified: RS1 RS2 RS6

18. **Name:** Able to exit game room

    Id: Test-ER1

    Type: Manual

    Initial State: User creates/joins a game room.

    Input: Game room menu is present and user presses exit button.

    Output: User is removed from game room.

    Test Case Derivation: If user wants to leave the game room, they should be allowed to.

    How test will be performed: Tester will manually press exit button in game room menu and verify that they are no longer in the game room.

    Requirements being verified: ER1

19. **Name:** Game room menu updated when user exits room

    Id: Test-ER2

    Type: Manual

    Initial State: Multiple users present in game room.

    Input: User exits game room.

    Output: Game room menu updated to no longer display the user in the game room.

    Test Case Derivation: The game room should be updated to show the current users present in the game room.

    How test will be performed: Tester will manually verify that after a user exits the game room, the game room menu is updated to longer display that user in the game room.

    Requirements being verified: ER2 ER3

20. **Name:** Current number of users in game room decreases when user exits

    Id: Test-ER3

    Type: Automated

    Initial State: Multiple users present in game room.

    Input: User exits game room.

    Output: Total number of users in game room decreases by one.

    Test Case Derivation: If a user leaves the game room, the total number of users present in the game room should be decreases by 1.

    How test will be performed: Create automated test which removes a user from a game room and verify that the total number of users in the game room database is one less than previous.

    Requirements being verified: ER3

4.1.2 Start Game Testing

The following section goes over tests related to starting a game, loading game assets, and determining/displaying puzzles. The functional requirements that are covered by this section are ST1, ST2, ST3, ST4, ST5, ST6.

1. **Name:** Able to start game

   Id: Test-ST1

   Type: Manual

   Initial State: Game room menu open.

   Input: User presses start button.

   Output: Game is started.

   Test Case Derivation: User needs to be able to start the game.

   How test will be performed: Tester will manually press start button and verify that the game starts.

   Requirements being verified: ST1

2. **Name:** Load all assets when starting game

   Id: Test-ST2

   Type: Automated

   Initial State: Game room menu open.

   Input: User starts game.

   Output: All assets used in the game are loaded.

   Test Case Derivation: All assets should be loaded at the start of the game.

   How test will be performed: Create automated test that detects if all assets have been loaded at the start of a game.

   Requirements being verified: ST1 ST2

3. **Name:** Inform user to complete calibration setup when starting game

   Id: Test-ST3

   Type: Manual

   Initial State: Game room menu open.

   Input: User starts game.

   Output: User is informed to complete calibration setup.

   Test Case Derivation: AR aspect of game requires scan of environment of room in order to place objects.

How test will be performed: Tester will manually start game and verify that they are prompted to complete the calibration setup.

Requirements being verified: ST1 ST3

4. **Name:** Order of puzzles determined upon start

   Id: Test-ST4

   Type: Automated

   Initial State: Game room menu open.

   Input: User starts game.

   Output: Order of puzzles is determined and each user is assigned a part of the puzzle.

   Test Case Derivation: System should give each user a certain part of a puzzle which requires them to work together to solve.

   How test will be performed: Create automated test that simulates users starting the game and completing calibration setup, and verifies that each user is assigned a certain puzzle number/type.

   Requirements being verified: ST1 ST4

5. **Name:** Display progress bar based on current progress

   Id: Test-ST5

   Type: Manual

   Initial State: Game room menu open.

   Input: User starts game.

   Output: Progress bar displayed on screen to let user know current progress.

   Test Case Derivation: System should give user indicator of current puzzle progress.

   How test will be performed: Tester will manually start game and verify that the progress bar is present.

   Requirements being verified: ST1 ST5

6. **Name:** Display puzzle to user

   Id: Test-ST6

   Type: Manual

   Initial State: Game room menu open.

   Input: User starts game.

   Output: Puzzles are displayed to each user and game commences.

   Test Case Derivation: Users need to be able to see the puzzles that they are assigned.

   How test will be performed: Tester will manually start game and verify that the puzzle is displayed to the user.

   Requirements being verified: ST1 ST2 ST4 ST6

4.1.3 Save/Load Game Testing

The following section will go over tests related to saving game progress and loading game data. The functional requirements that are covered by this section are SG1, SG2, SG3, SG4, SG5, SG6, SG7, SG8, LG1, LG2, LG3, DS1, DS2.

1. **Name:** Able to save game

   Id: Test-SG1

   Type: Automated

   Initial State: User has started game.

   Input: Save game action is triggered.

   Output: Game pauses for user who triggered save game action, while game state is saved to device.

   Test Case Derivation: User should be able to save their current progress on a puzzle and continue playing at a later time.

   How test will be performed: Create automated test that triggers save game functionality and checks that the current game state has been saved to the device.

   Requirements being verified: SG1 SG2 SG3 SG7

2. **Name:** Continue actions during save

   Id: Test-SG2

   Type: Manual

   Initial State: User has started game.

   Input: User presses save game button.

   Output: Users who did not press the save game button are able to act during saving process.

   Test Case Derivation: One user saving the game should not stop the other users from continuing playing.

   How test will be performed: Tester will verify that when the save game button is pressed they are able to continue doing actions.

   Requirements being verified: SG4

3. **Name:** Return to play after saving back-end

   Id: Test-SG3

   Type: Automated

   Initial State: User has started game.

   Input: Save game action is triggered.

   Output: Once save game action has completed the user is able to return to interacting with the game.

   Test Case Derivation: The user should be able to save the game data and return to playing the game. Their game should not remain paused and uninteractable once the save process has completed.

   How test will be performed: Create an automated test that triggers the save game functionality. Once the data saving process has completed, the automated test will simulate interacting with a puzzle and will be deemed successful if this interaction is registered.

   Requirements being verified: SG3 SG5

4. **Name:** Return to play after saving front-end

   Id: Test-SG4

   Type: Manual

Initial State: User has started game.

Input: User presses the save game button.

Output: Once save game action has completed the main camera view and AR overlay return to updating in real time.

Test Case Derivation: The user should be able to save the game data and return to playing the game. Once the game is un-paused they should be able to use their camera to look around and have this information updated on their UI, along with the components of the AR overlay mapped to their surroundings.

How test will be performed: Tester will trigger the save game action. They will turn 180 degrees to point their camera in the direction opposite its original direction, and watch to see if the UI updates to match this new view. The test is successful if the UI update is perceived as instantaneous by the user. This metric is based on PR1.

Requirements being verified: SG5

5. **Name:** Repeated saves

   Id: Test-SG5

   Type: Automated

   Initial State: User has started game.

   Input: Save game action is triggered 10 times in succession.

   Output: The game data is saved 10 times with the final saved data representing the system state at the timestamp of the most recent save.

   Test Case Derivation: The system should be capable of handling a user saving the game multiple times. This represents the extreme case of if the user is clicking the save button multiple times in a row. The number of saves is limited to 10 to limit data transfer usage during testing, keeping in mind the limit imposed by Unity.

   How test will be performed: Create an automated test that triggers the save game action 10 times in a loop. The time each save action finishes is recorded. The system shall then read the timestamp of the saved data from the device. If the timestamp on the saved data is between the second last and last recorded save action finish times, the test is successful.

Requirements being verified: SG6 SG7

6. **Name:** Data saved to device

   Id: Test-SG6

   Type: Manual

   Initial State: User has started game.

   Input: User presses the save game button.

   Output: The game files on the device are updated with the current game state data.

   Test Case Derivation: The game files are being stored on the local device so that a separate database is not needed. As such, the user should be able to see on their device that the game files have been updated after pressing the save game button.

   How test will be performed: Tester will press the save game button. They will then exit the app, locate the game files on their phone and check that the time modified matches the time of the save.

   Requirements being verified: SG7

7. **Name:** Saved data encoded

   Id: Test-SG7

   Type: Automated

   Initial State: User has started game and triggered save game action at least once.

   Input: Encoded files containing saved data.

   Output: Error message stating files cannot be loaded.

   Test Case Derivation: To prove that the save data is being encoded when stored, it must be proven that it is not in its original format. This can be shown by the file information not being able to be loaded back into the game if it is not first decoded.

   How test will be performed: Create an automated test that loads the saved game data from the device without decoding it. This test must contain error handling that produces an error message if the data cannot be properly loaded. The test is successful if this error message is produced.

Requirements being verified: SG8

8. **Name:** Saved data decoded

   Id: Test-SG8

   Type: Automated

   Initial State: User has started game and triggered save game action at least once.

   Input: Encoded files containing saved data.

   Output: Game loads into a game room without error.

   Test Case Derivation: To prove that the data can be decoded it must be shown that the game can decode the saved game files and load them into a valid state. That is, it can load the files without error.

   How test will be performed: Create an automated test that loads the saved game data from the device with decoding it. The test is successful if no errors are produced during the game loading process.

   Requirements being verified: SG8 LG2

9. **Name:** Load saved game

   Id: Test-SG9

   Type: Manual

   Initial State: User has started game and pressed the save game button at least once. They have then closed the game and are in the create/load game scene.

   Input: User presses load button on a saved game.

   Output: Game room is loaded in the same state as the original game room was in when it was saved.

   Test Case Derivation: When a saved game is loaded it must be in the same state as when it was saved. Specifically, it must have the same puzzle progress and overall room progress.

   How test will be performed: Tester will save an existing game and record all relevant information on the game state manually, such as puzzle progress, room progress and current AR overlay locations. They will then close the game room, returning to the main menu. They will

then load that saved game and observe the resulting game room. The test is successful if this game room contains all the same information as the original game room.

Requirements being verified: SG1 SG8 LG1 LG2 LG3

10. **Name:** Load saved game

    Id: Test-SG10

    Type: Manual

    Initial State: User has started game and pressed the save game button at least once. They have then closed the game and are in the create/load game scene.

    Input: User presses load button on a saved game.

    Output: Game room is loaded in the same state as the original game room was in when it was saved.

    Test Case Derivation: When a saved game is loaded it must be in the same state as when it was saved. Specifically, it must have the same puzzle progress and overall room progress.

    How test will be performed: Tester will save an existing game and record all relevant information on the game state manually, such as puzzle progress, room progress and current AR overlay locations. They will then close the game room, returning to the main menu. They will then load that saved game and observe the resulting game room. The test is successful if this game room contains all the same information as the original game room.

    Requirements being verified: SG1 SG8 LG1 LG2 LG3

11. **Name:** Delete saved game UI

    Id: Test-SG11

    Type: Manual

    Initial State: User has started game and pressed the save game button at least once. They have then closed the game and are in the create/load game scene.

    Input: User presses the delete button on a saved game

    Output: Saved game is removed from list of loadable games.

Test Case Derivation: When a saved game is deleted it should no longer be available to be loaded.

How test will be performed: Tester will delete a saved game from the list of loadable games. The test is successful if it no longer appears as a loadable game.

Requirements being verified: DS1

12. **Name:** Deleted game not available in new session

    Id: Test-SG12]

    Type: Manual

    Initial State: User has started game and pressed the save game button at least once. They have then closed the game and are in the create/load game scene.

    Input: User presses the delete button on a saved game. User then closes and re-opens the application and returns to the load game screen.

    Output: Saved game is still removed from list of loadable games.

    Test Case Derivation: A saved game that has been deleted should not reappear after the application has been restarted.

    How test will be performed: Tester will delete a saved game from the list of loadable games. They will then close and reopen the application and return to the load games scene. The test is successful if the deleted game has not reappeared in the list of loadable games.

    Requirements being verified: DS1 DS2

13. **Name:** Deleted game removed from device

    Id: Test-SG13

    Type: Automated

    Initial State: There is an existing game save file.

    Input: Trigger to delete saved game.

    Output: Saved game file is removed from device.

    Test Case Derivation: A saved game that has been deleted should no longer be stored on the device.

How test will be performed: Create an automated test to trigger the deletion of a save file and scan the device files to check that file no longer exists on the device.

Requirements being verified: DS2

4.1.4 Puzzle Interaction Testing

The following section will go over tests related to interacting with puzzles. *Many of these tests are left intentionally vague as specific puzzle actions are not yet known. These tests will also need to be performed for each distinct type of puzzle so they must remain general until specific puzzle types are known. As such, the puzzle specifications mentioned have not yet been created.* The functional requirements that are covered by this section are PI1, PI2, PI3, PI4, PI5, PI6, PI7, HR1, HR2, HR3, SP1, SP2, SP3.

1. **Name:** Puzzle selection

   Id: Test-PI1

   Type: Manual

   Initial State: User is in a game room with unfinished puzzles.

   Input: User selects puzzle.

   Output: Puzzle enlarges on UI.

   Test Case Derivation: The user should receive feedback that they have selected a puzzle. This feedback comes in the form of the puzzle enlarging in order for them to more easily interact with the puzzle.

   How test will be performed: Tester shall select a puzzle. The test is successful if the puzzle selected grows to an enlarged size as a result of this selection.

   Requirements being verified: PI1 PI2

2. **Name:** Puzzle actions UI

   Id: Test-PI2

   Type: Manual

   Initial State: User has selected a puzzle.

   Input: User performs an action on the puzzle.

Output: Puzzle UI responds appropriately according to the action.

Test Case Derivation: The user must be able to interact with the puzzle they have selected. The system UI must respond accordingly.

How test will be performed: Tester shall interact with one of the interactable components of the puzzle. The test is successful if the system displays the correct puzzle state response in the UI according to the specification for that puzzle. Should be performed for each puzzle type.

Requirements being verified: PI1 PI2 PI3 PI4

3. **Name:** Puzzle actions back-end

   Id: Test-PI3

   Type: Automated

   Initial State: Puzzle has been selected.

   Input: User performs an action on the puzzle.

   Output: Puzzle backend information responds appropriately according to the action.

   Test Case Derivation: When the user interacts with the puzzle they've selected, this change must be represented in the back-end information about the puzzle such as puzzle progress.

   How test will be performed: Create an automated test that shall simulate an interaction with one of the interactable components of the puzzle. The test is successful if the backend representation of the puzzle responds accordingly. Should be performed for each puzzle type.

   Requirements being verified: PI1 PI3

4. **Name:** Puzzle actions showing for other users UI

   Id: Test-PI4

   Type: Manual

   Initial State: User has selected a puzzle.

   Input: User performs an action on the puzzle.

   Output: Puzzle UI for other members in the game room interacting with the same puzzle responds appropriately according to the action.

Test Case Derivation: When a user interacts with a puzzle other users in the game room working on the puzzle must see these changes reflected on their own UIs so that the puzzle is in sync for all members at all times.

How test will be performed: Tester shall interact with one of the interactable components of the puzzle. The test is successful if the system displays the correct puzzle state response in the UI of another tester in the game room interacting with the same puzzle according to the specification for that puzzle. Should be performed for each puzzle type.

Requirements being verified: PI5

5. **Name:** Puzzle completion UI

   Id: Test-PI5

   Type: Manual

   Initial State: At least two users are have selected the same puzzle.

   Input: Users performs entire sequence of actions needed to complete puzzle according to puzzle specification.

   Output: UI produces notification to tell users they have completed the puzzle. Game room progress increases.

   Test Case Derivation: When users have performed all the necessary actions to complete a puzzle they should be notified that the puzzle is complete and the overall progress of the room should update.

   How test will be performed: Testers shall perform the entire required sequence of tasks to complete the puzzle, as specified by the puzzle specification. When complete all testers working on the puzzle should be notified the puzzle has been completed and the game room progress increase should be represented in the UI. Should be performed for each puzzle type.

   Requirements being verified: PI1 PI2 PI3 PI4 PI5 PI6 PI7

6. **Name:** Room progress update upon puzzle completion

   Id: Test-PI6

   Type: Manual

Initial State: At least two users have selected the same puzzle and one user has not selected the puzzle.

Input: Users performs entire sequence of actions needed to complete puzzle according to puzzle specification.

Output: Game room progress UI representation for user not collaborating on puzzle updates.

Test Case Derivation: Users who are not collaborating on a puzzle that is completed should also see the update of the overall game room progress when the puzzle is completed.

How test will be performed: Testers shall perform the entire required sequence of tasks to complete the puzzle, as specified by the puzzle specification. The tester not collaborating on the puzzle should observe the game room progress before and after the puzzle is complete. The test is successful if they see the game room progress UI update once the puzzle is completed. Should be performed for each puzzle type.

Requirements being verified: PI1 PI2 PI3 PI4 PI5 PI6 PI7

7. **Name:** Hint for each puzzle

   Id: Test-PI7

   Type: Automated

   Initial State: A puzzle has been selected.

   Input: Request a hint functionality triggered.

   Output: Hint for current puzzle is provided.

   Test Case Derivation: Each puzzle must have at least one hint for if users get stuck on the puzzle.

   How test will be performed: Create an automated test that triggers the request hint functionality. The test if successful if the system successfully returns a hint. Should be performed for each puzzle type.

   Requirements being verified: HR1

8. **Name:** Puzzle hint request

   Id: Test-PI8

Type: Manual

Initial State: A puzzle has been selected.

Input: User presses hint request button.

Output: Hint for current puzzle is displayed.

Test Case Derivation: When the user presses the button to request a hint, they should be provided a hint for the selected puzzle.

How test will be performed: Tester presses the hint button. Test is successful if a hint for the current puzzle is displayed.

Requirements being verified: HR1 HR2

9. **Name:** Close hint display

   Id: Test-PI9

   Type: Manual

   Initial State: A puzzle hint is being displayed.

   Input: User presses close button on hint display.

   Output: Hint display is removed.

   Test Case Derivation: When the user presses the button to close the hint, the hint display should be removed.

   How test will be performed: Tester presses the close button on the hint display. Test is successful if the hint display is closed.

   Requirements being verified: HR3

10. **Name:** Skip puzzle

    Id: Test-PI10

    Type: Manual

    Initial State: User has selected a puzzle.

    Input: User presses skip puzzle button.

    Output: Puzzle UI shrinks back down, user is returned to main game scene.

    Test Case Derivation: When a user skips a puzzle they should be returned to the main game scene where they can look around and select

other puzzles. This involves returning the puzzle UI to its original size from when it was not selected.

How test will be performed: Tester presses the skip button. Test is successful if the puzzle returns to its original size and the user is able to select another puzzle.

Requirements being verified: SP1 PI1

11. **Name:** Save skipped puzzle progress

    Id: Test-PI11

    Type: Manual

    Initial State: User has skipped a puzzle.

    Input: User selects the skipped puzzle.

    Output: Puzzle state is the same as when the puzzle was closed, given that no other users have interacted with the puzzle.

    Test Case Derivation: When a puzzle is skipped the progress on that puzzle should be saved, such that when the puzzle is opened again the state of the puzzle is the same as when it was skipped, as long as no other users have interacted with the puzzle.

    How test will be performed: Tester selects the puzzle they have skipped. The test is successful if the state of the puzzle is the same as when it was closed.

    Requirements being verified: SP1 SP2 PI1

12. **Name:** Skipped puzzle progress maintained for other users

    Id: Test-PI12

    Type: Manual

    Initial State: User has skipped a puzzle.

    Input: Different user selects the skipped puzzle.

    Output: Puzzle state for second user is the same as it was for the initial user when they skipped it, given that no other users have interacted with the puzzle.

    Test Case Derivation: When a puzzle is skipped the progress on that puzzle should be saved, such that when the puzzle is opened again by

another user the state of the puzzle is the same as when it was skipped by the initial user, as long as no other users have interacted with the puzzle.

How test will be performed: Tester selects a puzzle that was started and then skipped by another tester. The test is successful if the state of the puzzle is the same as when it was skipped.

Requirements being verified: SP1 SP2 PI1 PI5

13. **Name:** Notify puzzle collaborators of puzzle skip

    Id: Test-PI13

    Type: Manual

    Initial State: Multiple users have the same puzzle selected at the same time.

    Input: One user presses the skip puzzle button.

    Output: The UI of the other user displays a message stating that a collaborator has skipped the puzzle.

    Test Case Derivation: If two users are working on a puzzle and one of them skips the puzzle, the other user must be notified so that they are not waiting on an action from the first user.

    How test will be performed: Two testers have the same puzzle selected at the same time. The first presses the skip puzzle button. The test is successful if a message is displayed for the second user that a collaborator has skipped the puzzle.

    Requirements being verified: SP1 SP2 SP3

4.1.5 Messaging

The following section will go over tests related to sending and receiving messages. The functional requirements that are covered by this section are SM1, SM2, SM3, SM4, SM5, RM1, RM2, RM3.

1. **Name:** View users available for messaging

   Id: Test-MS1

   Type: Manual

Initial State: User is in a game room with at least one other user.

Input: User clicks messaging button.

Output: List of all other users in the game room is displayed.

Test Case Derivation: A user should be able to see all the other users in the game room so they can select which user to send a message to.

How test will be performed: Three testers are in a game room. The first tester clicks the messaging button. The test is successful if the other two testers names are displayed.

Requirements being verified: SM1 SM3

2. **Name:** Select user for messaging

   Id: Test-MS2

   Type: Manual

   Initial State: List of all other users in the game room is displayed within messaging interface.

   Input: User selects another user to send a message to from the list.

   Output: An interface is displayed containing an interactable keyboard and a text-box for the message.

   Test Case Derivation: The user must be presented with an interface that allows them to compose a message once they have selected the recipient.

   How test will be performed: Tester selects another tester from the list of users in the game room. The test is successful if a keyboard and textbox appear on the interface.

   Requirements being verified: SM2 SM3

3. **Name:** Cancel message

   Id: Test-MS3

   Type: Manual

   Initial State: Message composition interface composed of interactable keyboard and text-box is displayed.

Input: User types on keyboard to compose a message, presses the cancel button.

Output: Typed message appears in text-box, then once cancel button is pressed message composition interface is closed and user is returned to list of users in game room.

Test Case Derivation: The user must be able to see the message they are typing, and then be able to cancel this message if they decide they don't want to send it.

How test will be performed: Tester uses the keyboard to type a message, and then presses the cancel button to cancel this message. The test is successful if the tester can see the message they are typing in the text-box and if when they press the cancel button the keyboard and text-box are removed and they are returned to the list of users in the room.

Requirements being verified: SM2 SM4

4. **Name:** Send message

   Id: Test-MS4

   Type: Manual

   Initial State: Message composition interface composed of interactable keyboard and text-box is displayed.

   Input: User types on keyboard to compose a message, presses the send button.

   Output: Typed message appears in text-box, then once send button is pressed message is displayed in separate un-editable text box and fillable text-box is emptied.

   Test Case Derivation: The user must be able to see the message they are typing. Once the message has been sent they should be able to see the message they sent, and also be provided an empty text box to send a new message.

   How test will be performed: Tester uses the keyboard to type a message, and then presses the send button to send this message. The test is successful if the tester can see the message they are typing in the text-box and if when they press the send button the sent message is displayed and the fillable text box is emptied.

Requirements being verified: SM2 SM3 SM5

5. **Name:** Receive message

   Id: Test-MS5

   Type: Manual

   Initial State: User has selected a recipient and typed a message.

   Input: User presses send.

   Output: The UI of the selected recipient displays a notification that a message has been received.

   Test Case Derivation: When a message has been sent the recipient should receive a notification that they've received a message.

   How test will be performed: Tester sends a message to the selected recipient. The test is successful if a notification is displayed on the recipient's UI that a message has been received.

   Requirements being verified: RM1 SM3

6. **Name:** Read message

   Id: Test-MS6

   Type: Manual

   Initial State: User has a notification that they've received a message.

   Input: User touches notification.

   Output: The notification is removed. The received message is displayed.

   Test Case Derivation: When a user receives a message they should be able to press the notification to see the message and remove the notification.

   How test will be performed: Tester presses the notification for the received message. The test is successful if the message is displayed, the notification is removed and the message matches the message that was originally sent.

   Requirements being verified: RM1 RM2 SM3

7. **Name:** Close read message

   Id: Test-MS7

   Type: Manual

   Initial State: User has pressed the notification for a message they received.

   Input: User presses close button.

   Output: The received message is removed from the display, the user is returned to their state before pressing the notification.

   Test Case Derivation: Once a user has read the message they received, they must be able to close the display of that message and return to what they were doing before.

   How test will be performed: Tester presses the close button of the received message display. The test is successful if the display of the message is closed and the tester is returned to the game state they were in before pressing the notification.

   Requirements being verified: RM3

4.2 Tests for Nonfunctional Requirements

The following tests are related to testing for non-functional requirements. Furthermore a usability survey will be provided to users to test certain aspects of usability.

4.2.1 Look and Feel

Tests for look and feel requirements.

1. **Name:** Elements visible on screen

   Id: Test-LF1

   Type: Non-Functional, Automated

   Initial State: Program is run with simulated screen dimensions h (height) and w (width).

   Input: Title screen is launched.

Output: Menu elements are positioned with x and y coordinates within the h and w bounds.

How test will be performed: Test will generate random h and w dimensions, run the program title screen scripts, then check the x and y coordinates of the elements that need to be accessed by the user to ensure they have been scaled to fit within the bounds.

Requirements being verified: LF1

2. **Name:** App is visible on iPhone

   Id: Test-LF2

   Type: Non-Functional, Manual

   Initial State: iPhone SE with minimum iOS version 16 is loaded with the app.

   Input: The program is run.

   Output: Elements are scaled and positioned in a way that the user finds readable.

   How test will be performed: The tester will manually run the app on their phone, going through the startup sequence of entering a game. The test will be considered passed if the user feels that all elements interacted with in this process are in reasonable locations and easy to interact with.

   Requirements being verified: LF1, LF2, UH1, OE1

3. **Name:** App is visible on Android

   Id: Test-LF3

   Type: Non-Functional, Manual

   Initial State: Android phone with minimum Android version 11 is loaded with the app.

   Input: The program is run.

   Output: Elements are scaled and positioned in a way that the user finds readable.

   How test will be performed: The tester will manually run the app on their phone, going through the startup sequence of entering a game.

The test will be considered passed if the user feels that all elements interacted with in this process are in reasonable locations and easy to interact with.

Requirements being verified: LF1, LF2, UH1, OE1

4. **Name:** App is readable in poor conditions

   Id: Test-LF4

   Type: Non-Functional, Manual

   Initial State: App is launched in bright location.

   Input: The program is run.

   Output: All text is able to be read with minimal effort.

   How test will be performed: The test will be performed in a bright room/ outside during the day to test readability in poor conditions. The tester will manually run the app on their phone, going through the startup sequence of entering a game. The test will be considered passed if the user feels that all text elements are able to be read without much effort or changing location.

   Requirements being verified: LF2, UH3

4.2.2 Usability and Humanity

Tests for usability and humanity requirements.

1. **Name:** User is not connected to internet

   Id: Test-UH1

   Type: Non-Functional, Automated

   Initial State: The user is not connected to the internet.

   Input: The user attempts to connect to a lobby.

   Output: A message appears to prompt the user to connect to the internet and no lobby is joined.

   How test will be performed: The app will be configured to detect no internet connection then call the script to join a lobby. It will then check that an element has appeared on screen with text asking the user to connect to the internet.

Requirements being verified: UH2

2. **Name:** User disconnects while in lobby

   Id: Test-UH2

   Type: Non-Functional, Automated

   Initial State: The user is in a lobby and connected to the internet.

   Input: The user loses connection to the internet.

   Output: A message appears to prompt the user to connect to the internet.

   How test will be performed: The app will be initially be configured to detect an internet connection and have the user in a lobby. It will then simulate losing the connection, and check that an element has appeared on screen with text asking the user to connect to the internet.

   Requirements being verified: UH2, UH4

3. **Name:** User disconnects while in game

   Id: Test-UH3

   Type: Non-Functional, Automated

   Initial State: The user is in a game and connected to the internet.

   Input: The user loses connection to the internet.

   Output: A message appears to prompt the user to connect to the internet.

   How test will be performed: The app will be initially be configured to detect an internet connection and have the user in a game. It will then simulate losing the connection, and check that an element has appeared on screen with text asking the user to connect to the internet.

   Requirements being verified: UH2, UH4

4. **Name:** App usable for users below the age of 20

   Id: Test-UH4

   Type: Non-Functional, Manual

   Initial State: A user below the age of 20 will install the app.

   Input: The program is run.

Output: The user determines that the app is able to be navigated and used.

How test will be performed: A user in the demographic of below the age of 20 will go through the startup sequence and initial section of the app. The test will be considered passed if they determine that they are able to operate the app without major issues.

Requirements being verified: UH3

5. **Name:** App usable for users between ages of 20 and 30

   Id: Test-UH5

   Type: Non-Functional, Manual

   Initial State: A user between the ages of 20-30 will install the app.

   Input: The program is run.

   Output: The user determines that the app is able to be navigated and used.

   How test will be performed: A user in the demographic of between the ages of 20-30 will go through the startup sequence and initial section of the app. The test will be considered passed if they determine that they are able to operate the app without major issues.

   Requirements being verified: UH3

6. **Name:** App usable for users above the age of 30

   Id: Test-UH6

   Type: Non-Functional, Manual

   Initial State: A user above the age of 30 will install the app.

   Input: The program is run.

   Output: The user determines that the app is able to be navigated and used.

   How test will be performed: A user in the demographic of above the age of 30 will go through the startup sequence and initial section of the app. The test will be considered passed if they determine that they are able to operate the app without major issues.

   Requirements being verified: UH3

7. **Name:** Internet connection becomes weak

   Id: Test-UH7

   Type: Non-Functional, Automated

   Initial State: The program detects a strong internet connection

   Input: The detected connection changes to become weaker

   Output: A message will be created prompting the user to find a stronger connection

   How test will be performed: The system will be configured to detect a weak internet connection. The test will be considered passed if an element is detected with text finding the user to find a better connection.

   Requirements being verified: UH5

8. **Name:** Environment Prompt

   Id: Test-UH8

   Type: Non-Functional, Dynamic, Manual

   Initial State: The user is in a lobby

   Input: Game starts

   Output: The system will prompt the user to check their environment and make sure its suitable for playing the game

   How test will be performed: A tester will open the game and check to see if the environment message pops up when they start the game.

   Requirements being tested: UH6

9. **Name:** Internet Disconnect

   Id: Test-UH9

   Type: Functional, Dynamic, Manual

   Initial State: The user is connected to the internet.

   Input: The user disconnects from the internet then reconnects.

   Output: The user should be prompted to rejoin the game after their connection is restored.

How test will be performed: The tester will join a game with an internet connection, then they will turn off the internet on their phone and wait to get disconnected from the lobby, then turn on the internet and see if they can rejoin.

Requirements being tested: UH7

10. **Name:** Dangerous Conditions Warning

    Id: Test-UH10

    Type: Functional, Dynamic, Manual

    Initial State: User is playing the game.

    Input: Potential dangerous conditions is reported.

    Output: The system will tell the user to go inside or stop playing the game if dangerous conditions are detected.

    How test will be performed: The tester will simulate a signal of dangerous conditions getting input to the user, then see if the dangerous conditions message appears.

    Requirements being tested: UH8

4.2.3 Performance

Tests for performance requirements.

1. **Name:** Program responds in a reasonable amount of time

   Id: Test-PR1

   Type: Non-Functional, Automated

   Initial State: The app is connected to the internet.

   Input: A request is made to join a lobby.

   Output: The system responds within 100ms.

   How test will be performed:A script will be run to create a lobby, and the server response time will be measured. The test will pass if the response happens within 100ms.

   Requirements being tested: PR1

2. **Name:** App is available during the day

   Id: Test-PR2

   Type: Non-Functional, Manual

   Initial State: The time is between 6am and 6pm.

   Input: A request is made to join a lobby.

   Output: The functionality is available.

   How test will be performed: A user will attempt to use the app during the day and ensure that functionality isn't hindered.

   Requirements being tested: PR2

3. **Name:** App is available during the night

   Id: Test-PR3

   Type: Non-Functional, Manual

   Initial State: The time is between 6pm and 6am.

   Input: A request is made to join a lobby.

   Output: The functionality is available.

   How test will be performed: A user will attempt to use the app during the night and ensure that functionality isn't hindered.

   Requirements being tested: PR2

4.2.4 Operational and Environmental

N/A

4.2.5 Maintainability and Support

1. **Name:** Documentation Verification

   Id: Test-MS1

   Type: Non-Functional, Static, Document Review Walk through

   How test will be performed: All of the developers will group together and go through the documentation line by line and verify that every requirement is met by the current code.

   Requirements being verified: MS1, MS2

2. **Name:** Comments Code Analysis

   Id: Test-MS2

   Type: Non-Functional, Static

   How test will be performed: One person not involved with MacAR with experience in coding will look at the code and comments and give feedback on what is understood.

   Requirements being verified: MS1, MS2

4.2.6 Security

1. **Name:** External Database Query

   Id: Test-SR1

   Type: Non-Functional, Dynamic, Manual

   Initial State: One User's IP address is on the server's database.

   Input: External request for user's IP address.

   Output: No IP address returned.

   How test will be performed: Externally querying the database through code injection attacks to see if they can obtain another user's IP address.

   Requirements being verified: SR1

2. **Name:** Hidden Data

   Id: Test-SR2

   Type: Non-functional, Manual, Dynamic

   Initial State: The user is using the application.

   Input: N/A

   Output: The system will provide all the functionality of the game without showing the user data they do not need to see. All internal functionality will be hidden to the user.

   How test will be performed: External tester tests the application, and afterwards is asked if they can see any unintended features.

   Requirements being verified: SR2

4.2.7 Cultural

1. **Name:** Offensive Assets

   Id: Test-CU1

   Type: Non-Functional, Dynamic, Manual

   Initial State: N/A

   Input: N/A

   Output: No offensive images, text, or sound are displayed or heard during playing the game.

   How test will be performed: A group of testers of various ethnicity, religion, and culture will be selected to play-test the game, and to report any potentially miss-leading, offensive, or sensitive subject matter during the game play.

   Requirements being verified: CR1

2. **Name:** Canadian English

   Id: Test-CU2

   Type: Non-Functional, Dynamic, Manual

   Initial State: N/A

   Input: N/A

   Output: All available text and audio clues are in Canadian English

   How test will be performed: The testers will play through the game during the other tests and verify that the text and audio clues are all in Canadian English.

   Requirements being verified: CR1, CR2

3. **Name:** Canadian English Walkthrough

   Id: Test-CU3

   Type: Non-Functional, Static, Code Inspection

   How test will be performed: The developers will listen to every sound asset, and inspect all the text elements in the code, and check that all data the users can see is in Canadian English.

   Requirements being verified: CR2

4.2.8 Legal

1. **Name:** Asset Code Walk-through

   Id: Test-LR1

   Type: Non-Functional, Static, Code Walk through

   Initial State: Project is updated on GitHub

   Input: N/A

   Output: All the assets in the game are either created by MacAR developers or are open source resources that are properly attributed to.

   How test will be performed: All the members will run through a code coverage session and analyze all the code and assets we are using and verify if they are either our assets or assets that have been properly attributed to their creators

   Requirements being verified: LR1

4.2.9 Health and Safety

Tests for health and safety requirements.

1. **Name:** App warms user to be aware of their environment

   Id: Test-HS1

   Type: Non-Functional, Automated

   Initial State: The user is in a lobby

   Input: Game starts

   Output: The system will prompt the user to check their environment and make sure its suitable for playing the game

   How test will be performed: The script to join a lobby will be called, and check that an element has been created with a message to be aware of surroundings.

   Requirements being verified: HS1

4.3 Traceability Between Test Cases and Requirements

	CG1	CG2	CG3	CG4	CG5	JG1	JG2	JG3	JG4	JG5	RS1	RS2	RS3	RS4	RS5	RS6	ER1	ER2	ER3	ST1	ST2	ST3	ST4	ST5	ST6	SG1	SG2	SG3	SG4	SG5	SG6	SG7	SG8	LG1	LG2	LG3
Test-CR1	X																																			
Test-CR2	X	X																																		
Test-CR3	X		X																																	
Test-CR4	X		X																																	
Test-CR5	X			X																																
Test-CR6	X			X																																
Test-CR7	X				X																															
Test-JR1						X				X																										
Test-JR2						X																														
Test-JR3						X	X																													
Test-JR4						X		X																												
Test-JR5						X			X																											
Test-RS1											X	X																								
Test-RS2											X	X	X																							
Test-RS3											X	X		X																						
Test-RS4											X	X			X																					
Test-RS5											X	X				X																				
Test-ER1																	X																			
Test-ER2																		X	X																	
Test-ER3																			X																	
Test-ST1																				X																
Test-ST2																				X	X															
Test-ST3																				X		X														
Test-ST4																				X			X													
Test-ST5																				X				X												
Test-ST6																				X	X		X		X											
Test-SG1																										X	X	X				X				
Test-SG2																													X							
Test-SG3																												X		X						
Test-SG4																														X						
Test-SG5																															X	X				
Test-SG6																																X				
Test-SG7																																	X			
Test-SG8																																	X		X	
Test-SG9																										X							X	X	X	X
Test-SG10																										X							X	X	X	X

Table 2: Functional Traceability Matrix Pt. 1

	DS1	DS2	PI1	PI2	PI3	PI4	PI5	PI6	PI7	HR1	HR2	HR3	SP1	SP2	SP3	SM1	SM2	SM3	SM4	SM5	RM1	RM2	RM3
Test-SG11	X																						
Test-SG12	X	X																					
Test-SG13		X																					
Test-PI1			X	X																			
Test-PI2			X	X	X	X																	
Test-PI3			X		X																		
Test-PI4							X																
Test-PI5			X	X	X	X	X	X	X														
Test-PI6			X	X	X	X	X	X	X														
Test-PI7										X													
Test-PI8										X	X												
Test-PI9												X											
Test-PI10			X										X										
Test-PI11			X										X	X									
Test-PI12			X				X						X	X									
Test-PI13													X	X	X								
Test-MS1																X		X					
Test-MS2																	X	X					
Test-MS3																	X		X				
Test-MS4																	X	X		X			
Test-MS5																		X			X		
Test-MS6																		X			X	X	
Test-MS7																							X

Table 3: Functional Traceability Matrix Pt. 2

	LF1	LF2	UH1	UH2	UH3	UH4	UH5	UH6	UH7	UH8	PR1	PR2	OE1	MS1	MS2	SR1	SR2	CR1	CR2	LR1	HS1
Test-LF1	X																				
Test-LF2	X	X	X										X								
Test-LF3	X	X	X										X								
Test-LF4		X			X																
Test-UH1				X																	
Test-UH2				X		X															
Test-UH3				X		X															
Test-UH4					X																
Test-UH5					X																
Test-UH6					X																
Test-UH7							X														
Test-UH8								X													
Test-UH9									X												
Test-UH10										X											
Test-PR1											X										
Test-PR2												X									
Test-PR3												X									
Test-MS1														X	X						
Test-MS2														X	X						
Test-SR1																X					
Test-SR2																	X				
Test-CU1																		X			
Test-CU2																		X	X		
Test-CU3																			X		
Test-LR1																				X	
Test-HS1																					X

Table 4: Non-Functional Traceability Matrix

5 Unit Test Description

This section will be filled out once the MIS is completed.

5.1 Unit Testing Scope

TBD

5.2 Tests for Functional Requirements

TBD

5.3 Tests for Nonfunctional Requirements

TBD

5.4 Traceability Between Test Cases and Modules

TBD

References

Joseph Keebler, Willian Shelstad, Dustin Smith, Barbara Chaparro, and Mikki Phan. Validation of the guess-18: A short version of the game user experience satisfaction scale (guess). https://uxpajournal.org/validation-game-user-experience-satisfaction-scale-guess/, 2023.

6 Appendix

This section contains symbolic parameters and usability survey questions.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS once more information is known about implementation specifics of the system. Their values will be defined in this section for easy maintenance once the MIS has been created.

6.2 Usability Survey Questions?

This list will include all of the survey questions we are going to ask users for usability. The survey will initially be provided to at least 20 potential users who have tried Revision 0 of the game. The goal of this survey is not to test specific requirements, but rather to validate the overall system from a user experience standpoint. This will allow us to test whether the right system is being built for a satisfying user experience, or whether new/modified requirements are needed. The game user experience satisfaction scale (GUESS-18) was used as a source for questions (Keebler et al., 2023). The user's will input their score from 1-5(1=bad, 5=good) for each of the follow statements:

- I find the controls of the game to be straightforward
- I find the game's interface to be easy to navigate
- I feel detached from the outside world while playing the game.
- I do not care to check events that are happening in the real world during the game.
- I think the game is fun
- I find the game supports social interaction between players
- I enjoy playing this game with other users
- I feel the game allows me to be imaginative
- I feel creative while playing the game

- I feel like the puzzles are challenging
- I feel like the puzzle interactions are fun
- I feel the game's audio enhances my gaming experience
- I want to do as well as possible during the game
- I enjoy the game's graphics
- I think the game is visually appealing

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning:

1. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage etc. You should look to identify at least one item for each team member.

    - Sam - Static testing and code walkthroughs are an important part of validating that what was created is correct. Since we have stakeholders invested in the project, being able to look through the code himself, along with walk relevant stakeholders through important sections, will ensure that the project is on the correct track. As a result of this, interpersonal and investigative skills will need to be developed throughout the projct.
    - Kieran - nUnit is the primary method of testing C# programs, along with many other popular languages. Kieran will learn how to properly leverage this software to create efficient, meaningful tests.
    - Ethan - Manual testing skills will definitely be a knowledge/skill that is acquired to successfully complete the verification and validation of the project. Because we are making a game, most of the functional tests are dynamic and manual as any user interaction with the application needs to be tested. As a result of this, skills regarding manual testing will need to be developed throughout the project in order to be able to successfully test all aspects of the software.
    - Matthew - Automation testing skills are essential, and are needed to complete our verification. Matthew will learn how to automate as many of the test cases as possible using industry standard methods. As a result of this, the automated test cases can be implemented, and potentially more manual tests can become automated.

2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

    - Static Testing/ Code Walkthroughs:
        - Frequent walking through code with supervisor - Sam: Frequent walkthroughs will allow for practicing how to explain the code to another individual. Doing it with the supervisor, who is already aware of what the code should be doing, will make it easier to repeat the proccess with secondary stakeholders.
        - Reading through different snippets of code to determine functionality - Kieran: Being able to identify intent and functionality is an important part of efficient static analysis. Reading through existing code snippets and talking to the person who created them is an easy way to practice this skill.
    - nUnit Development:
        - Online tutorials - Ethan: nUnit is a popular tool and there are lots of videos walking through how to use the tool. Incorporating other peoples ideas into our own will improve our tests.
        - Official documentation - Matthew: In a similar vein, Microsoft has exensive documentation online, which will help us determine the best way to use the provided functions.
    - Manual Testing:
        - Developing Checklists - Sam: Doing multiple manual runthroughs and keeping a checklist of things to manually test is a good way to keep track of what works and what doesn't in terms of testing. By the end of the project, we should have an in depth checklist that any tester could follow to verify functionality.
        - Trying different devices - Matthew: Getting a feel for how to manually test on different devices will allow us to account for testing in any setting.
    - Automated Testing:

– Online tutorials - Kieran: There are lots of videos online for the best ways to automate the testing of C# scripts, which will be a good place to start.
– Comparing with the rest of the team - Ethan: Some users might identify good patterns to follow when constructing automated tests. Ensuring that someone collects and distributes this knowledge will improve this skill for the whole team.