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

March 7, 2024

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
2023/11/18	Matthew	Resolved most of the team review issues
2024/02/29	Matthew	Created a test case for the usability survey
2024/03/01	Ethan	Removal of Save/Load Game Tests
2024/03/01	Matthew	Description for what the term "database" implies in the test cases
2024/03/06	Sam	Updated Traceability, removed PI7 (updated indexes to match)
2024/03/06	All	Worked on updating the document for revision 0 of VnV Report

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 validator, 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 and 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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

   Requirements being verified: CG1

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

   Id: Test-CR2

   Type: Manual

   Initial State: Create game menu open.

   Input: Game room name inputted. The game room name will be a string consisting of letters, numbers, and ascii characters, and of character length greater than 3 and less than 64.

   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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

Requirements being verified: CG1 CG2 CG5

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

   Id: Test-CR3

   Type: Manual

   Initial State: Game room creation screen open, nothing input for the game room name.

   Input: User tries to create a new game room without inputting a name for the 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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

   Requirements being verified: CG1 CG2 CG5

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 10). Every value from 2 to 10 will be tested.

   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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

Requirements being verified: CG1 CG3 CG5

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. The password length will be between 8 and 64 characters, and consists of letters, number, and ASCII characters arranged in a random order.

   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. Users should also be able to enter a room that requires a password.

   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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

   Requirements being verified: CG1 CG4 CG5 JG1 JG2

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, and hosts a game room.

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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

Requirements being verified: CG1 CG4 CG5

7. **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 the database.

   Test Case Derivation: System should allow users to view all the game rooms.

   How test will be performed: Tester will manually press Join Game button and verify that they are redirected to the join game menu. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

   Requirements being verified: JG1 JG5

   Id: Test-JR2

   Type: Manual

   Initial State: Join game menu open.

Input: User attempts to enter a room that is at its maximum capacity.

Output: The user is not able to enter the room.

Test Case Derivation: System should not allow users to join a full game room.

How test will be performed: Tester will manually attempt to enter a room that is at its maximum capacity. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

Requirements being verified: JG1

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

   Id: Test-JR3

   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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

   Requirements being verified: JG1

9. **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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

Requirements being verified: JG1 JG3

10. **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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

    Requirements being verified: JG1 JG4

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

    Id: Test-RS1

    Type: Manual

    Initial State: Game room menu is present.

    Input: User presses settings button, then processes to change the game room name or password.

    Output: User is redirected to edit game settings menu which displays the current settings of the game room (ie. room name, 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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

Requirements being verified: RS1 RS2

12. **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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

    Requirements being verified: RS1 RS2 RS3

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

    Id: Test-RS3

    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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

Requirements being verified: RS1 RS2 RS6

14. **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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

    Requirements being verified: ER1

15. **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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

Requirements being verified: ER2 ER3

16. **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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

    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, and ST5.

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 enabled and in the scene.

   Test Case Derivation: Assets should be loaded into the game when required.

   How test will be performed: Create automated test that detects if all assets have been loaded at the start of a game. The automated test will check if the required assets have the enabled flag set to true by looping through the puzzle list.

   Requirements being verified: ST1 ST2

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

   Id: Test-ST3

   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 verifies that each user is assigned their respective puzzle and the associated puzzle elements to complete the

puzzle. The puzzles are not unique, every 2-3 players should be given their own puzzle to solve, depending on the number of players in the game room. Ex. with 5 players, a group of 2 and a group of 3 each get their own puzzle.

Requirements being verified: ST1 ST3

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

   Id: Test-ST4

   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 ST4

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

   Id: Test-ST5

   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 ST3 ST5

4.1.3 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: PI4 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 ST4

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:** Puzzle hint request

   Id: Test-PI7

   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

8. **Name:** Close hint display

   Id: Test-PI8

   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

9. **Name:** Skip puzzle

   Id: Test-PI9

   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

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

    Id: Test-PI10

    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

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

    Id: Test-PI11

    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

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

    Id: Test-PI12

    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

4.1.4 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:** Open messaging interface

   Id: Test-MS1

   Type: Manual

   Initial State: User is in a game room.

   Input: User clicks messaging button.

   Output: Messaging interface is displayed.

   Test Case Derivation: The user must be able open the messaging interface to send a message.

   How test will be performed: Tester clicks the messaging button. The test is successful if the messaging interface containing the display window and the input field are displayed.

   Requirements being verified: SM1 SM3

2. **Name:** Close messaging interface

   Id: Test-MS2

   Type: Manual

   Initial State: Messaging interface composed of the message display port and text input field are displayed.

   Input: User presses the "X" button to close the interface.

   Output: The messaging interface is closed.

Test Case Derivation: The user must be able to leave the messaging interface and any point and return to the puzzle component of the game.

How test will be performed: When the messaging interface is open, the tester checks that the "X" button is visible to close it. The tester then presses the "X" button and the test is successful if the message display port, text input field and "X" button are no longer visible.

Requirements being verified: SM2 SM4 RM3

3. **Name:** Send message

   Id: Test-MS3

   Type: Manual

   Initial State: Messaging interface composed of messaging display port and text input field is displayed.

   Input: User taps the input field and types on the displayed keyboard to compose a message, presses the send button.

   Output: Typed message appears in text input field, then once send button is pressed message is displayed in the message display port. The input field returns to the empty (default) state.

   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 taps the input field and uses the displayed 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 input field and if when they press the send button the sent message is displayed in the messaging display port and the input field is emptied.

   Requirements being verified: SM2 SM3 SM5

4. **Name:** Receive message

   Id: Test-MS4

   Type: Manual

Initial State: User has opened the messaging interface, tapped the input field and typed a message. The message is a string of ASCII characters longer than 0 characters.

Input: User presses send.

Output: The UI of all the other users in the game room display 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. The test is successful if a notification is displayed on the UI of all other users in the game room that a message has been received. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

Requirements being verified: RM1 SM3

5. **Name:** Read message

   Id: Test-MS5

   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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

   Requirements being verified: RM1 RM2 SM3

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, Manual

   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

   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

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: Multiple users 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.

   Test Case Derivation: Users between the age of 20 and 30 are the target demographic, so testing in a larger range for under the target demographic is efficient given the limited resources.

   How test will be performed: Multiple users selected at random 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: Multiple users 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.

Test Case Derivation: Users between the age of 20 and 30 are the target demographic, so testing the target demographic range independently of the other tests will give us important data on how to improve our system for our target

How test will be performed: Multiple users selected at random 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: Multiple users 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.

   Test Case Derivation: Users between the age of 20 and 30 are the target demographic, so testing in a larger range of ages above the target demographic is more efficient given the limited amount of resources and time.

   How test will be performed: Multiple users selected at random 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 UH6

7. **Name:** Internet Disconnect

   Id: Test-UH7

   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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

Requirements being tested: UH5 UH7

8. **Name:** Usability Survey

   Id: Test-UH8

   Type: Non-Functional, Dynamic, Manual

   Initial State: Users have completed playing the game

   Output: Average rating of usability survey is 3/5 or higher

   How test will be performed: The usability survey 6.2, is given to the users that have completed the game to fill out to provide feedback on the game-play experience.

   Requirements being tested: UH1-UH9, LF1-LF2, PR1-PR2, CR1

4.2.3 Performance

Tests for performance requirements.

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

   Id: Test-PR1

   Type: Non-Functional, Manual

   Initial State: The app is connected to the internet.

   Input: A request is made to join a lobby.

   Output: The system responds within 5s.

How test will be performed: The tester will run the app and create a lobby. The server response time for this request will be measured. The test will pass if the response happens within 5 seconds. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

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 and there is no current server outage.

   Input: A request is made to join a lobby.

   Output: The functionality is available.

   Test Case Derivation: 6am - 6pm EST would be the slack time, as most potential users in the main region of North America would be going to work or school instead of using the app.

   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 EST and there is no current server outage.

   Input: A request is made to join a lobby.

   Output: The functionality is available.

   Test Case Derivation: 6pm - 6am EST would be the peak hours for users in the main region of North America as it is right after they get home from work/school.

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. **Name:** Server Outage Warning

   Id: Test-PR4

   Type: Non-Functional, Manual

   Initial State: There is an active server outage.

   Input: A request is made to join a lobby.

   Output: The system will display a messaging notifying the user there is a server outage and the game is currently unavailable.

   Test Case Derivation: If there is a server outage the game won't be available as the lobby and communication systems won't work. The user should be made aware of this so they know why they cannot access the game.

   How test will be performed: The tester will attempt to use the app during a known outage period. The test is successful if the outage message appears.

   Requirements being tested: PR2

4.2.4 Operational and Environmental

N/A

4.2.5 Maintainability and Support

1. **Name:** Documentation Verification

   Id: Test-MT1

   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-MT2

   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. This test will be run on a test database with existing lobbies, and on another test database without other existing lobbies, each scenario tested 50% of the time.

   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: The testers will play through the game during the other tests and 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: 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
Test-CR1	X																								
Test-CR2	X	X			X																				
Test-CR3	X		X		X																				
Test-CR4	X		X		X																				
Test-CR5	X			X	X	X	X																		
Test-CR6	X			X	X																				
Test-JR1						X				X															
Test-JR2						X																			
Test-JR3						X		X																	
Test-JR4						X			X																
Test-JR5						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		X		X
Test-PI5																							X		

Table 2: Functional Traceability Matrix Pt. 1

	PI1	PI2	PI3	PI4	PI5	PI6	PI7	HR1	HR2	HR3	SP1	SP2	SP3	SM1	SM2	SM3	SM4	SM5	RM1	RM2	RM3
Test-PI1	X	X																			
Test-PI2	X	X	X	X																	
Test-PI3	X		X																		
Test-PI4				X	X																
Test-PI5	X	X	X	X	X	X	X														
Test-PI6	X	X	X	X	X	X	X														
Test-PI7								X	X												
Test-PI8										X											
Test-PI9	X										X										
Test-PI10	X										X	X									
Test-PI11	X				X						X	X									
Test-PI12											X	X	X								
Test-MS1														X		X					
Test-MS2															X		X				X
Test-MS3															X	X		X			
Test-MS4																X			X		
Test-MS5																X			X	X	
Test-PM1	X					X	X														
Test-PM2	X					X	X														
Test-PM3	X					X	X														
Test-PM4	X					X	X														
Test-PM5	X					X	X														
Test-WI1			X	X	X																
Test-WI2			X	X	X																
Test-WI3			X	X	X																
Test-WI4			X	X	X																
Test-WI5			X	X	X																
Test-MP1			X	X	X																
Test-MP2			X	X	X																
Test-MP3			X	X	X																
Test-MP4			X	X	X																
Test-MP5			X	X	X																
Test-MP6			X	X	X																
Test-MP7			X	X	X																
Test-SP1			X	X	X																
Test-SP2			X	X	X																
Test-SP3			X	X	X																
Test-SP4			X	X	X																
Test-SP5			X	X	X																
Test-SP6			X	X	X																
Test-IP1			X	X	X																
Test-IP2			X	X	X																
Test-IP3			X	X	X																
Test-IP4			X	X	X																
Test-CP1			X	X	X																
Test-CP2			X	X	X																
Test-CP3			X	X	X																
Test-CP4			X	X	X																
Test-CP5			X	X	X																
Test-CP6			X	X	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			X													
Test-UH7							X		X												
Test-UH8										X											
Test-PR1											X										
Test-PR2												X									
Test-PR3												X									
Test-PR4												X									
Test-MT1														X	X						
Test-MT2														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

The scope of the unit testing is limited to functions that have observable effects outside of the unity game scene. Mostly helper functions that implement modules.

5.2 Tests for Functional Requirements

5.2.1 MultiplayerPuzzle Module

The Multiplayer Puzzle Module tests deal with testing the puzzle manager which handles all of the spawning and despawning of puzzles, along with puzzle based communication between client and server.

1. Test-PM1

   Type: Automatic

   Initial State: N/A

   Input: Simple vector of data created

   Output: Data correctly serialized into bytes

   Test Case Derivation: Allows for the server to send simple data to the client

   How test will be performed: Puzzle manager will be created and ObjectToBytes will be called with the data

2. Test-PM2

   Type: Automatic

   Initial State: N/A

   Input: Complex vector of data created

   Output: Data correctly serialized into bytes

Test Case Derivation: Allows for the server to send complex data to the client

How test will be performed: Puzzle manager will be created and ObjectToBytes will be called with the data

3. Test-PM3

   Type: Automatic

   Initial State: N/A

   Input: Simple vector of bytes created

   Output: Data correctly deserialized into bytes

   Test Case Derivation: Allows for the client to read simple data from the server

   How test will be performed: Puzzle manager will be created and BytesToObject will be called with the bytes

4. Test-PM4

   Type: Automatic

   Initial State: N/A

   Input: Complex vector of bytes created

   Output: Data correctly deserialized into bytes

   Test Case Derivation: Allows for the client to read complex data from the server

   How test will be performed: Puzzle manager will be created and BytesToObject will be called with the bytes

5. Test-PM5

   Type: Automatic

   Initial State: N/A

   Input: Vector of data created

Output: Data the same after full cycle of serialization and deserialization

Test Case Derivation: Simulates full cycle of the server and the client communicating with each other

How test will be performed: Puzzle manager will be created and objectToBytes, then BytesToObject will be called with the bytes

5.2.2 MazePuzzle Module

The Maze Puzzle Module tests deal with testing the non-multiplayer portions of the code that are crucial to the helping the module perform all the required actions. This module is separated from the other modules, and the functions in the module are not decencies from other modules.

1. Test-MP1

   Type: Automatic

   Initial State: Empty scene, no objects spawned

   Input: Spawn in the maze puzzle

   Output: The maze puzzle is successfully spawned in, and mazePuzzle.activeSelf is true.

   Test Case Derivation: When we spawn in a given game object, we should expect them to be available for use for the user/code, and this test will test if when we create an object we can interact with it, and it is active on the scene.

   How test will be performed: In an NUnit test case, create an empty scene, and instantiate the maze puzzle using a pre-loaded maze puzzle prefab, and check if the maze puzzle is active using the mazePuzzle.activeSelf attribute.

2. Test-MP2

   Type: Automatic

   Initial State: Empty scene, no objects spawned

   Input: Instantiate maze puzzle at (0,0,0) with rotation (0,0,0)

Output: The maze puzzle object is not null

Test Case Derivation: Checking that the object is non-null implies that the object exists in the scene at the location and rotation specified when spawning it in.

How test will be performed: In an NUnit test case, create an empty scene, and instantiate the maze puzzle using a pre-loaded maze puzzle prefab. Set the object location and rotation at spawn to (0,0,0), and (0,0,0) respectively. Then Assert that the object instance is not null.

3. Test-MP3

   Type: Automatic

   Initial State: N/A

   Input: A two dimensional array of size nxn, where $1 \leq n$, $n \in \mathbb{Z}$ input into the To1DArray function

   Output: A one dimensional array arranged row by row containing all the elements from the two dimensional array input

   Test Case Derivation: This method is used in the code to convert a two dimensional array to a one dimensional array so it can be transmitted over the server/client RPC

   How test will be performed: In an NUnit test case, create a random 2D array and another 1D array with the same elements, and then call the To1DArray with the 2D array as input, and test to see if the output is the same as our 1D array. Test with successful cases and unsuccessful cases.

4. Test-MP4

   Type: Automatic

   Initial State: N/A

   Input: A one dimensional array of size n^2, where $1 \leq n$, $n \in \mathbb{Z}$ input into the Make2DArray function

   Output: A two dimensional array of size nxn containing all the elements from the one dimensional array input row by row.

Test Case Derivation: This method is used in the code to convert a one dimensional array to a two dimensional array so it can take the one dimensional array given by the clientRPC and convert it to a useable format.

How test will be performed: In an NUnit test case, create a random 2D array and another 1D array with the same elements, and then call the Make2DArry with the 1D array as input, and test to see if the output is the same as our 2D array. Test with successful cases and unsuccessful cases.

5. Test-MP5

   Type: Automatic

   Initial State: N/A

   Input: Two adjacent maze cube are input into the ClearWalls function

   Output: Both walls have their adjacent walls removed i.e. the wall closest to each other

   Test Case Derivation: The maze generation algorithm uses a wall clearing method to generate the random maze every time, so the wall clearing needs to be validated to ensure the correct walls are being cleared.

   How test will be performed: In an NUnit test case, create two walls adjacent to each other by Instantiating two maze cells with different positions. Then call the ClearWalls function with both the cells as input, and check the walls properties on the maze cells to see if they have been removed.

6. Test-MP6

   Type: Automatic

   Initial State: N/A

   Input: A random array of size 100 with elements between 0 and 3 input into the convertLayoutToGrid function

   Output: A maze that matches the removed elements

Test Case Derivation: To synchronize each user's maze, a maze layout array is sent to each user, and this function takes the array and converts it into the maze grid, so this function is very crucial to ensuring consistency between users.

How test will be performed: In an NUnit test case, create an array of size 100, and fill each elements with a random integer between 0 and 3, and then set the size of the maze grid to 10x10 and call the convertLayoutToGrid function with the array, then compare the output to the array to ensure the correct maze was generated.

7. Test-MP7

Type: Automatic

Initial State: N/A

Input: A completely filled maze is massed into the GetUnvisitedCells function

Output: No unvisited cells found / empty list

Test Case Derivation: The maze generation algorithm guarantees that every single maze cell will be visited, so GetUnvisitedCells should give an empty list after the maze has been generated.

How test will be performed: In an NUnit test case, create an array of size 100, and fill each elements with a random integer between 0 and 3, and then set the size of the maze grid to 10x10 and call the convert-LayoutToGrid function with the array. Then call GetUnvisitedCells and check to see if it's an empty list.

5.2.3 IsometricPuzzle Module

1. Test-IP1

Type: Automatic

Initial State: Empty scene, no objects spawned

Input: Spawn in Isometric Puzzle

Output: Isometric successfully spawns in

Test Case Derivation: When the Isometric is spawned, users should see the Isometric puzzle in the scene

How test will be performed: NUnit test that instantiates the Isometric puzzle prefab into the scene and checks if it is active using the activeSelf attribute of the GameObject

2. Test-IP2

   Type: Automatic

   Initial State: Empty scene, no objects spawned

   Input: Spawn in Isometric Puzzle

   Output: Isometric Puzzle object is not null

   Test Case Derivation: When Isometric puzzle is spawned, need to make sure that the GameObject exists

   How test will be performed: NUnit test that instantiates the Isometric Puzzle prefab into the scene and checks that GameObject is not null

3. Test-IP3

   Type: Automatic

   Initial State: Isometric puzzle is spawned

   Input: Call isometric puzzle InitializePuzzle without first setting up a multiplayer environment

   Output: Null Reference Exception

   Test Case Derivation: When there is no multiplayer element, the puzzles should not work, as they depend on the multiplayer module

   How test will be performed: Nunit test will spawn the Isometric puzzle and call the Initialize function and when it catches the NullReferenceException, it will succeed the test case

4. Test-IP4

   Type: Automatic

Initial State: Isometric puzzle is spawned

Input: Set the cube layout of the isometric puzzle to one of the letter-number combinations

Output: Cube layout changes to match the input combination

Test Case Derivation: When we switch between Isometric cubes, we want to ensure the right cubes are showing

How test will be performed: Nunit test will spawn the Isometric puzzle and call a helper function called TestingSetup to setup the class for testing executions, and then call setCubes with one of the letter-number combinations.

5.2.4 SimonSaysPuzzle Module

1. Test-SP1

   Type: Automatic

   Initial State: Empty scene, no objects spawned

   Input: Spawn in Simon Says Puzzle

   Output: Simon Says Puzzle successfully spawns in

   Test Case Derivation: When the Simon Says puzzle is spawned, users should see the puzzle prefab present in the scene

   How test will be performed: NUnit test that instantiates the SimonSaysPuzzle prefab into the scene and checks if it is active using the activeSelf attribute of the GameObject

2. Test-SP2

   Type: Automatic

   Initial State: Empty scene, no objects spawned

   Input: Spawn in Simon Says Puzzle

   Output: Simon Says Puzzle object is not null

   Test Case Derivation: When Simon Says puzzle is spawned, need to make sure that the GameObject exists

How test will be performed: NUnit test that instantiates the SimonSaysPuzzle prefab into the scene and checks that GameObject is not null

3. Test-SP3

   Type: Automatic

   Initial State: Spawn in Simon Says Puzzle

   Input: Increment level function called

   Output: Simon Says Puzzle level increases by 1

   Test Case Derivation: Important to test that the IncrementLevel() function called in the Simon Says Puzzle is working correctly

   How test will be performed: NUnit test that calls IncrementLevel() function and checks that level increases from 1 to 2

4. Test-SP4

   Type: Automatic

   Initial State: Spawn in Simon Says Puzzle

   Input: Colour sequence list contains blue. TrackUserInput(blue) is called which represents user pressing blue button

   Output: Colour sequence list and Player Sequence list are equal

   Test Case Derivation: Important to test that the TrackUserInput function is working as its purpose is to store what button the user has pressed

   How test will be performed: NUnit test that adds blue to the Simon Says Puzzle colour sequence list, then calls TrackUserInput on blue button and checks that the colour sequence list and the player input match

5. Test-SP5

   Type: Automatic

   Initial State: Spawn in Simon Says Puzzle

Input: Player input list contains colour values. BeginSimonSays() function is called.

Output: Player input list is cleared

Test Case Derivation: Important to test that each time the BeginSimonSays() function is called, the current player input list is cleared since a new colour sequence will be generated.

How test will be performed: NUnit test that adds colour values to the player input list, then calls BeginSimonSays(). Tests confirms that player sequence list is empty.

6. Test-SP6

Type: Automatic

Initial State: Spawn in Simon Says Puzzle

Input: Colour sequence list contains colour values. BeginSimonSays() function is called.

Output: Colour sequence list is cleared

Test Case Derivation: Important to test that each time the BeginSimonSays() function is called, the current colour sequence is cleared since a new colour sequence will be generated.

How test will be performed: NUnit test that adds colour values to the colour sequence list, then calls BeginSimonSays(). Tests confirms that colour sequence list is empty.

5.2.5 WiresPuzzle Module

The Wires Puzzle Module tests deal with testing the wires puzzle and all of its constituent components.

1. Test-WI1

Type: Automatic

Initial State: Puzzle Spawned

Input: Wires initialized with trivial sequence of wires and nodes

Output: Puzzle is updated to save the new expected wire sequence

Test Case Derivation: Important to test that the puzzle can be properly initialized with a given code

How test will be performed: Automated NUnit test that will construct a trivial 1:1 list of wire and node connections, and then call Init with the list as a parameter

2. Test-WI2

   Type: Automatic

   Initial State: Puzzle Spawned

   Input: Wires initialized with complex sequence of wires and nodes

   Output: Puzzle is updated to save the new expected wire sequence

   Test Case Derivation: Important to test that the puzzle can be properly initialized with a complex code

   How test will be performed: Automated NUnit test that will construct a complex list associating wires and nodes, and then call Init with the list as a parameter

3. Test-WI3

   Type: Automatic

   Initial State: Puzzle Spawned

   Input: Init is called

   Output: Puzzle automatically initializes with impossible connections, indicating that no connections are currently formed

   Test Case Derivation: Important to remove the risk of a default array initialization causing the puzzle to automatically complete

   How test will be performed: Automated NUnit test that spawn in the puzzle and check it's values upon initialization

4. Test-WI4

   Type: Automatic

Initial State: Puzzle spawned and initialized

Input: New connection is created

Output: Puzzle state updates to reflect the new connection

Test Case Derivation: Simulates the proccess of the user working their way through the puzzle, ensuring state changes are saved

How test will be performed: Automated NUnit test that spawn in the puzzle, initialize it, then call UpdateSequence with a new connection to check that it saves

5. Test-WI5

   Type: Automatic

   Initial State: Puzzle spawned and initialized

   Input: Single new connection is created

   Output: Puzzle is marked as incomplete

   Test Case Derivation: Important to ensure that only the correct sequence will complete the puzzle

   How test will be performed: Automated NUnit test that spawn in the puzzle, initialize it, then call UpdateSequence with a new connection that doesn't fully match the final set of connections

5.2.6 CombinationPuzzle Module

1. Test-CP1

   Type: Automatic

   Initial State: Empty scene, no objects spawned

   Input: Spawn in Combination Puzzle

   Output: Combination Puzzle successfully spawns in

   Test Case Derivation: When the Combination puzzle is spawned, users should see the puzzle prefab present in the scene

   How test will be performed: NUnit test that instantiates the CombinationPuzzle prefab into the scene and checks if it is active using the activeSelf attribute of the GameObject

2. Test-CP2

   Type: Automatic

   Initial State: Empty scene, no objects spawned

   Input: Spawn in Combination Puzzle

   Output: Combination Puzzle object is not null

   Test Case Derivation: When Combination puzzle is spawned, need to make sure that the GameObject exists

   How test will be performed: NUnit test that instantiates the CombinationPuzzle prefab into the scene and checks that GameObject is not null

3. Test-CP3

   Type: Automatic

   Initial State: Combination puzzle has been spawned and initialized

   Input: Simulate pressing the "3" button on the keypad

   Output: The first character of the current code is returned as "3"

   Test Case Derivation: When a button corresponding to the correct digit of the combination is pressed, the current code should update accordingly storing that correct entry.

   How test will be performed: NUnit test that calls the KeyPadPress function with an input of "3" and checks the updated currentCode first digit value against the character "3" to check that they are equal

4. Test-CP4

   Type: Automatic

   Initial State: Combination puzzle has been spawned and initialized

   Input: Simulate pressing the "4" button on the keypad

   Output: The first character of the current code is returned as "_"

   Test Case Derivation: When a button not corresponding to the current current combination digit is pressed the currentCode value should

reset to it's default as the users must restart their entry of the combination

How test will be performed: NUnit test that calls the KeyPadPress function with an input of "4" and checks the updated currentCode first digit value against the character "_" to check that they are equal

5. Test-CP5

   Type: Automatic

   Initial State: Combination puzzle has been spawned and initialized

   Input: Simulate pressing the buttons corresponding to the correct code

   Output: The completePuzzle value is returned as True

   Test Case Derivation: When the correct combination is entered the puzzle should be marked as complete in order to proceed in the room.

   How test will be performed: NUnit test that calls the KeyPadPress function for each of the four digits corresponding to the correct code. Then assert that the value of completePuzzle is True

6. Test-CP6

   Type: Automatic

   Initial State: Empty scene, no objects spawned

   Input: Spawn in and initialize the Combination Puzzle

   Output: Instruction page text is not the default value

   Test Case Derivation: When the combination puzzle is spawned in and initialized the instruction page should be populated with the instruction information for the corresponding user. If the page still contains the default information after initializing the puzzle, the test has failed.

   How test will be performed: NUnit test that instantiates and initializes the CombinationPuzzle prefab and checks that the text field of the instruction page game object does not still only contain the test "Instructions" which is the default value.

5.3 Tests for Nonfunctional Requirements

Due to constraints with the Unity environment, all non-functional are to be performed manually.

5.4 Traceability Between Test Cases and Modules

Module descriptions can be found in section 5 of the MG. Module M10 (Math Puzzle module) will not be implemented, and therefore does not have any traceability to tests. In the future, documents will be revised to update module numbering.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
Test-CR1		X												
Test-CR2		X												
Test-CR3		X												
Test-CR4		X												
Test-CR5		X												
Test-CR6		X												
Test-JR1		X												
Test-JR2		X												
Test-JR3		X												
Test-JR4		X												
Test-RS1		X												
Test-RS2		X												
Test-RS3		X												
Test-ER1		X												
Test-ER2		X												
Test-ER3		X												
Test-ST1		X												
Test-ST2		X			X									
Test-ST3					X									
Test-ST4					X									
Test-ST5					X									
Test-PI1					X									
Test-PI2					X									
Test-PI3					X									
Test-PI4					X									
Test-PI5					X									
Test-PI6					X									
Test-PI7					X									
Test-PI8					X									
Test-PI9					X									
Test-PI10					X									
Test-PI11					X									
Test-PI12					X									
Test-MS1			X											
Test-MS2			X											
Test-MS3			X											
Test-MS4			X											
Test-MS5			X											
Test-MS6			X											
Test-PM1					X									
Test-PM2					X									

Table 5: Module Traceability Matrix

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
Test-PM3					X									
Test-PM4					X									
Test-PM5					X									
Test-MP1									X					
Test-MP2									X					
Test-MP3									X					
Test-MP4									X					
Test-MP5									X					
Test-MP6									X					
Test-MP7									X					
Test-SP1						X								
Test-SP2						X								
Test-SP3						X								
Test-SP4						X								
Test-SP5						X								
Test-SP6						X								
Test-WI1								X						
Test-WI2								X						
Test-WI3								X						
Test-WI4								X						
Test-WI5								X						
Test-IP1							X							
Test-IP2							X							
Test-IP3							X							
Test-IP4							X							
Test-CP1											X			
Test-CP2											X			
Test-CP3											X			
Test-CP4											X			
Test-CP5											X			
Test-CP6											X			
Test-LF1		X												
Test-LF2		X			X									
Test-LF3		X			X									
Test-LF4		X			X									
Test-UH1		X											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-PR1		X										X		
Test-PR2		X	X	X								X		
Test-PR3		X	X	X								X		
Test-PR4		X	X	X								X	X	
Test-MT1														X
Test-MT2														X
Test-SR1												X		
Test-SR2												X		
Test-CU1														X
Test-CU2														X
Test-CU3														X
Test-LR1														X
Test-HS1		X											X	

Table 6: Module Traceability Matrix Pt. 2

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.