Farming Matters: System Verification and Validation Plan for Farming Matters

Team #14, The Farmers
Brandon Duong
Andrew Balmakund
Namit Chopra
Mohammad Harun
Mihail Serafimovski

March 9, 2023

1 Revision History

Date	Version	Notes
11/02/2022 1	1.0	Finished first version
03/07/2023 1	1.1	Updated Non-function requirement tests to match test implementation

Contents

List of Tables

List of Figures

This document describes the verification and validation plan for Farming Matters. By the end of the course all manual and automated tests will have been executed and passed on the final version.

2 General Information

2.1 Summary

Farming Matters is an interactive and engaging game where the user manages a farm. The goal of the game is to collect research data about risk-reward decisions from the users. The software will mask traditional conventions of conducting surveys and lab experiments allowing users to make genuine decisions. The game decisions made by the users will be logged on the server and saved which can be later used for research.

2.2 Objectives

- Build confidence in the software's correctness to match the gameplay consistency
- Demonstrate suitable usability for users playing the game and data collection for research
- To assure the system has met the requirements provided in the Software Requirement Specification (SRS)
- To assure potential hazards are accounted for

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 Document
- Hazard Analysis

- Verification & Validation Report
- Module Guide
- Module Interface Specification
- User Guide

3 Plan

This section highlights and outlines the plans for the validation and verification of the software requirements, design verification, as well as this document. The section also discusses implementing the verification plan given in this document, automated testing and verification tools, as well as the validation plan.

3.1 Verification and Validation Team

Table 1: Verification and Validation Team Members Table

Name	**Role(s)**	**Responsibilities**
Mohammad Harun	Backend test developer, Manual tester, SRS verifier	Create automated tests for backend functionality, perform manual testing, review SRS
Namit Chopra	Backend test developer, Test framework developer	Create automated tests for backend functionality, set up test framework including code coverage, etc.
Andrew Balmakund	Frontend test developer, Manual tester	Create automated tests for frontend functionality, perform manual testing
Brandon Duong	Full stack test developer, Lead test developer	Create automated tests for frontend and backend functionality, lead existing and new directives in testing and validation
Mihail Serafimovski	Frontend test developer, Test framework developer	Create automated tests for frontend functionality, set up test framework including code coverage, etc.
Prof. Niko Yiannakoulias	Supervisor, Manual tester, Final reviewer	As the project will be delivered to Prof. Yiannakoulias, he will also be the final reviewer in terms of validation. He will also perform manual tests in the process.

Please note that all team members listed on the table will also have the role of code reviewers. This includes the supervisor, as he has to maintain the code in the future and should therefore have some oversight of the code itself.

3.2 SRS Verification Plan

SRS verification will be implemented in a variety of ways. First of all, reviews by classmates and teammates will be used in the form of the creation of GitHub Issues. Although this has been done before, we intend to continue verifying the SRS in this way. We will improve on the previous process

by creating a checklist to guide a reviewer, to help improve the speed of reviews and guide reviewers towards areas that may need more attention. This checklist will be implemented in the form of a GitHub Issues template.

We will also use structured reviews from the project's supervisor to help verify the SRS. This will be carried out in the form of an SRS walkthrough, where all group members and the supervisor will meet and the group will guide the supervisor through the SRS document.

3.3 Design Verification Plan

Design verification will be implemented very similarly to SRS verification.
Like in the preceding section, reviews by classmates and teammates will be used in the form of the creation of GitHub Issues. A checklist to guide reviewers will also be created, to help improve the speed of reviews and guide reviewers towards areas that may need more attention. This checklist will be implemented in the form of a GitHub Issues template.

We will also use structured reviews from the project's supervisor to help verify the design. This will be carried out in the form of a design walkthrough, where all group members and the supervisor will meet and the group will guide the supervisor through the design of the project, including drawing diagrams on a whiteboard and describing concepts in-depth where needed. The supervisor will be encouraged to give feedback during the walkthrough.

3.4 Verification and Validation Plan Verification Plan

Verification and Validation Plan verification will be implemented very similarly to the preceding two sections.
Like in the preceding sections, reviews by classmates and teammates will be used in the form of the creation of GitHub Issues. A checklist to guide reviewers will also be created, to help improve the speed of reviews and guide reviewers towards areas that may need more attention. This checklist will be implemented in the form of a GitHub Issues template.

Due to the nature of the Verification and Validation plan, we elected to use other classmates for structured reviews as well as the supervisor. Since other classmates have a more technical background than our supervisor, these structured reviews will help us improve the VnV plan document. The review from the supervisor will be equally for his purposes, so he is familiar with how we are testing the project. These structured reviews will be carried out in the

form of VnV plan walkthroughs, where all group members and the supervisor will meet and the group will guide the supervisor through the design of the project, including drawing diagrams on a whiteboard and describing concepts in-depth where needed. The supervisor will be encouraged to give feedback during the walkthrough.

3.5 Implementation Verification Plan

Implementation verification will be done using the dynamic and static tests outlined in this document. Namely, they consist of the tests for functional requirements in section 4.1 and the tests for nonfunctional requirements in section 4.2. In future revisions of this document, unit tests from section 6 will also be used for implementation verification.

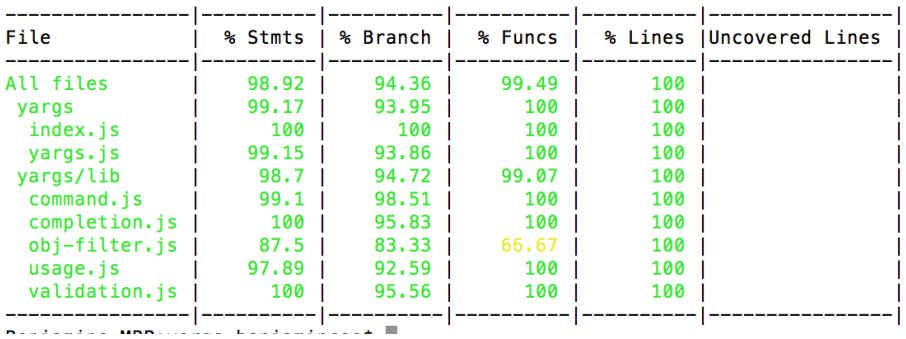
Implementation verification will also be carried out through code reviews. When a group member makes a pull request to the GitHub repository, another member will have to approve it before it can be merged to the main (stable) branch.

Finally, implementation verification will be carried out through a static analyzer. For now, this will just consist of running the linter used for the project (see section 3.6). This can be set up either via GitHub Actions, or since the team size is small it may cost less time to simply require that all devs run the linter on their pull requests before opening them.

3.6 Automated Testing and Verification Tools

- For front-end testing, we will use Jest. For back-end, we will use Mocha.
  - Mocha is the most popular Javascript unit testing framework. It is designed to be simple, extensible, and fast. Mocha provides a lot of useful functionality out of the box such as fixtures, mocking, etc. It also provides a lot of features for async testing, which will be useful for testing back-end components that need to do async operations such as writing to a database (for example).
  - Jest is also very popular, however we will specifically use it for front-end testing as it provides a suite of useful features such as snapshot testing that can be used to ensure the layout of a page doesn't change.

- For code coverage, we will use istanbul.js.
  - Istanbul.js is a very simple, lightweight code coverage tool that bundles and runs all unit tests with a single command. It generates a very readable and understandable output and prints it to the terminal. An example can be seen in the linked website.
  - Code coverage reports for the entire project will be generated as shown in Figure 1 below.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
All files	98.92	94.36	99.49	100	
yargs	99.17	93.95	100	100	
index.js	100	100	100	100	
yargs.js	99.15	93.86	100	100	
yargs/lib	98.7	94.72	99.07	100	
command.js	99.1	98.51	100	100	
completion.js	100	95.83	100	100	
obj-filter.js	87.5	83.33	66.67	100	
usage.js	97.89	92.59	100	100	
validation.js	100	95.56	100	100	

Figure 1: A sample code coverage report

- CI will be set up using GitHub actions. The CI system will run automated tests triggered by a change to the main branch to ensure the main branch is always stable.
  - Mocha and Jest both have the necessary features needed to automate testing with GitHub actions.
- Linter: prettier

3.7 Software Validation Plan

One of the central purposes of this project is to gather research data through a user's gameplay. The supervisor, Prof. Niko Yiannakoulias, has done research before on the same topic, but through traditional survey methods instead of a game. The data obtained from his previous research will be important to validate the software, as it will help show if the software fulfilled

its purpose of generating reliable data.

Another method of validating the software will be review sessions with our primary stakeholder, the supervisor (Prof. Niko Yiannakoulias). We have a weekly meeting cadence set up with him and we can use these to iteratively validate the software as we go through the development process as opposed to validating it just once. In the beginning, it will be best to keep these review sessions less formal but as the software gets closer to the finished product, the review sessions will become formal in nature and the group will implement task based inspection.

4 System Test Description

4.1 Tests for Functional Requirements

Game testing can be broken down into the following sections. The subsections will cover all the different components of the game. These components will work operate separately or work together to fulfill the functional requirements. Some of the components will be tested manually while others will require automated testing.

4.1.1 Account Testing

The account creation testing deals with the test cases related that relate to a user account. FR1, FR2, FR5 and FR15 are the functional requirements that are covered by the subsection. The system requires you to have an account to play the game. The test cases include the following: creating an account, resetting password, logging in, verifying if the user is human and account deletion.

1. **Name:** Successful Login

   Id: Test-AC1

   Control: Automatic

   Initial State: User is not logged in. Start at login page

   Input: All valid information required for account creation

Output: Account corresponding to the input information is created in database with game state initialized to NEW_ACCOUNT_STATE

Test Case Derivation: To fully verify the account creation functionality, it is essential that the account information reflects the input information

How test will be performed: Create automatic test that creates an account and checks that the account can be logged into

2. **Name:** Unsuccessful Login

   Id: Test-AC2

   Control: Automatic

   Initial State: User is not logged in and at login page

   Input: At least 1 information required for account creation is invalid

   Output: Account is not created in database and corresponding error is returned

   Test Case Derivation: The system should not allow a user to create an account if they do not input all the necessary information, or if the information is invalid

   How test will be performed: Create automatic test that inputs invalid information and verifies that a corresponding error is returned

3. **Name:** Reset Password
   Id: Test-AC3

   Control: Manual

   Initial State: User is not logged in

   Input: User's email

   Output: Password was successfully reset

   Test Case Derivation: In the event the user misplaces or forgets their password there needs to be a safeguard to ensure progress is not lost

   How test will be performed: After resetting the password try to login with the new credentials to see if password was successfully reset

4. **Name:** Verify Human User
   Id: Test-AC4

   Control: Manual

   Initial State: User is not logged in. Start at login page

   Input: All information required for account creation is present

   Output: System verifies user through a captcha

   Test Case Derivation: Users must be able to succeed the captcha to ensure no automation of account creation is possible

   How test will be performed: Manual test that fills in all account information and checks that a captcha is required before account creation is attempted

5. **Name:** Successful deletion of user account
   Id: Test-AC5
   Control: Manual

   Initial State: User has an account and is logged in

   Input: User request to have their account information deleted

   Output: Data pertaining to the given username and password is deleted

   Test Case Derivation: This functionality is essential for the ethics board and so must be thoroughly tested

   How test will be performed: This test can be run after the other manual test of successfully creating an account

4.1.2 Game Mechanics Testing

The game mechanics testing deals with the test cases related that relate to the game. FR3, FR4, FR6, FR7, FR8, FR9, FR10, FR11, FR12, FR16, FR17, FR19, FR21 and FR22 are the functional requirements that are covered by the subsection. The game mechanics compose of all the game logic and consist of any actions the player is able to perform in the context of the game. Examples of these test cases include successfully buying and selling crops.

1. **Name:** Successful Land Purchase
   Id: Test-GM1

   Control: Manual

   Initial State: User is logged in and at game page

   Input: User requests to buy selected land area

   Output: User's currency count correctly reflects the price of buying the land, and the user is able to interact with new bought piece of land

   Test Case Derivation: User's should always accurately be aware of how much money they have to ensure they make genuine and calculated decisions within the game, and be able to expand their farm

   How test will be performed: Thorough and iterative manual testing of the land functionality

2. **Name:** Successful planting of seeds
   Id: Test-GM2

   Control: Manual

   Initial State: User is logged in, at game page, and has an empty piece of land

   Input: User requests to plant seeds on selected land area

   Output: The seeds are removed from the user's inventory and planted on the selected area

   Test Case Derivation: User's should always accurately be aware of what is in their inventory and be able to plant crops

   How test will be performed: Thorough and iterative manual testing of the farming functionality

3. **Name:** Successful growing of plant seeds
   Id: Test-GM3

   Control: Manual

   Initial State: User is logged in, at game page, and has planted a seed on a piece of land

Input: User ends turn

Output: All planted seeds grow by 1 turn

Test Case Derivation: User's should be able to plant crops and know of a plant's growth progress

How test will be performed: Thorough and iterative manual testing of the farming functionality

4. **Name:** Successful use of fertilizer on crop
   Id: Test-GM4

   Control: Manual

   Initial State: User is logged in, at game page, and has planted a seed on a piece of land

   Input: User requests to use fertilizer on a selected planted seed

   Output: Planted seed's growth is accelerated by FERTILIZER_EFFECT

   Test Case Derivation: User's should be able to use fertilizer on crops to create a more complex decision making system

   How test will be performed: Thorough and iterative manual testing of the farming functionality

5. Name: Successful visual check of farm area enlarged
   Id: Test-GM5

   Control: Manual

   Initial State: User is logged in and is in the shop

   Input: Amount of land to be purchased

   Output: The land is purchased and the farm area is larger prior to purchase that the user can interact with

   Test Case Derivation: The game should be somewhat expansive and the user should be able to purchase land to enlarge their farm

   How the test will be performed: Manual test to check visually if the farm area has enlarged

6. **Name:** Automatic prompt for a consultant
   Id: Test-GM6

   Control: Manual

   Initial State: User is logged in and at game page

   Input: User progresses to a turn number that is divisible by CONSULTING_INTERVAL

   Output: User is prompted on whether or not they want to pay for consulting advice

   Test Case Derivation: This prompt is essential to the research study aspect of the system, which must be functional and consistent in order for the research to propose a conclusion

   How test will be performed: Manual testing of playing the overall game and ensuring the consultant prompt would be present every CONSULTING_INTERVAL turns

7. **Name:** Consultant's advice
   Id: Test-GM7

   Control: Manual

   Initial State: User is logged in and at game page

   Input: User plays the game for multiple turns and is prompted on whether or not they want to pay for consulting advice

   Output: The consulting advice is randomly given as either deterministic or probabilistic

   Test Case Derivation: This distinction between deterministic and probabilistic information is essential to the research study aspect of the system, which must be functional and consistent in order for the research to propose a conclusion. The same type of advice should be given a each player throughout the game

   How test will be performed: Manual testing of playing the overall game and ensuring the consultant information is either deterministic or probabilistic on average 50% of the time for each

8. **Name:** Successful prompt for insurance upon purchasing a crop
   Id: Test-GM8

   Control: Manual

   Initial State: User is logged in and in shop

   Input: User is prompted on whether or not they will purchase insurance with the crop.

   Output: Insurance is added to crop in case of damage

   Test Case Derivation: The research also aims to understand if user's are willing to pay for insurance and under what circumstance.

   How test will be performed: Manual test to check whether prompt appears to purchase insurance when purchasing a crop.

9. **Name:** Successful occurrence of random events/decisions happening in one season
   Id: test-GM9

   Control: Manual

   Initial State: User is logged in and at game page

   Input: User plays out 1 season, of which is equivalent to SEASON_LENGTH amount of turns

   Output: User is prompted with a random event EVENT_OCCURRENCE amount of times within that season

   Test Case Derivation: This functionality is essential to the study as these decoy decisions/events hide the decisions that matter to the research. These decoy decisions ensure participants are unaware of what they actual study is about and so must be thoroughly tested

   How test will be performed: Manual testing of playing the overall game and ensuring these random events/decisions happen twice a season

10. **Name:** Successful season change
    Id: Test-GM10

Control: Manual

Initial State: User has played the game for some time

Input: User has played SEASON_LENGTH turns

Output: the state of the season change

Test Case Derivation: This is significant as it will encourage the user to plan ahead and try to optimize their decisions in the game for the best outcomes.

How the test will be performed: Manual test to check whether after SEASON_LENGTH the season has changed.

4.1.3 Database Testing

The database testing deals with the test cases related that relate to the game. FR13 and FR14 are the functional requirements that are covered by the subsection. The database is critical as it will log decisions and the game state. The test cases ensure that the game decisions and game state will be successfully logged.

1. **Name:** Logging game decisions
   Id: Test-DB1
   Control: Manual

   Initial State: User is logged in and at game page

   Input: Any action the user performs while playing the game

   Output: All actions will be logged

   Test Case Derivation: The research is highly dependent on decisions which are the actions that will be logged by the game.

   How test will be performed: Manual test by checking the server logs to see if all actions are being logged.

2. **Name:** Successful saving of game state
   Id: Test-DB2
   Control: Manual

   Initial State: User is logged in and makes a turn to change the game state.

   Input: The most recent game state

Output: The new game state will be saved

Test Case Derivation: The user needs to have the option of being able to continue where they left off in case they wish to play at a later time.

How the test will be performed: The tester will play three turns (one season) of the game trying to maximize profit. After three turns, the tester ensures the game state table contains the most recent game state in the corresponding account.

3. **Name:** Successful loading of game state
   Id: Test-DB3
   Control: Manual

   Initial State: User is logged in and performs actions to change the game state. Then, the user logs out of the game.

   Input: The user logs back into the game

   Output: The new game state is same as prior to logout

   Test Case Derivation: The user needs to have the option of being able to continue where they left off in case they wish to play at a later time.

   How the test will be performed: The tester will play three turns (one season) of the game trying to maximize profit. The user will logs out of the game and logs back in. The game state prior to logout must be the same as the state after login.

4.1.4 Pre-Game Testing

The Pre-game testing deals with the test cases related that relate to the user before they can start playing the game. FR18 is the only functional requirement that is covered by the subsection. To play the game, the user must agree to the terms and conditions in the consent form.

4. **Name:** User agrees to the consent form
   Id: Test-PG1

Control: Automatic

Initial State: User is not logged in, has not played the game yet, and is presented the consent form

Input: The user accepts the terms and conditions to the consent form

Output: User is re-directed to the login screen

Test Case Derivation: This is mandatory to conduct the study and without the user's consent their data will not be able to be collected

How the test will be performed: Automated test to check if all terms and conditions to play the game have been met

5. **Name:** User does not agree to consent form
   Id: Test-PG2

   Control: Automatic

   Initial State: User is not logged in, has not played the game yet, and is presented the consent form

   Input: The user rejects the terms and conditions to the consent form

   Output: The user is unable to access the login screen and play the game

   Test Case Derivation: This is mandatory to conduct the study and without the user's consent their data will not be able to be collected

   How the test will be performed: Automated test will reject the consent form and try to access the login screen

4.2 Tests for Nonfunctional Requirements

The non-functional system requirements that require non-functional testing. These tests can be broken down into different subsections that cover the areas of testing. These subsections for the non-functional tests include: Look and Feel, Usability and Humanity, Performance, Operational and Environmental, Maintainability and Support Requirements, Security, Access and Integrity. Area of Testing:

- Client Side & Server Side Performance Testing

- Security & Data Integrity Testing
- Look and Feel
- Usability

4.2.1 Look and Feel

1. **Name:** Survey for feedback on different interface elements
   Id: Test-LF1

   Type: Non-functional, Dynamic, Manual

   Initial State: The user is at home page

   Input: The user registers account, logs in and plays out one season

   Output/Result: Get feedback and verify interface is not overwhelming

   How test will be performed: A survey will be given to the players where they will give feedback to different elements of the interface.

2. **Name:** Test the system on different screen resolutions
   Id: Test-LF2

   Type: Non-functional, Dynamic, Manual

   Initial State: The user is at home page

   Input: The user registers account, logs in and plays out one season

   Output: Get feedback and verify interface is clear at different SCREEN_RESOLUTION

   How test will be performed: The test will be performed manually with the tester playing one season at different SCREEN_RESOLUTION in which they are to navigate to different screens and access different interface features.

3. **Name:** Testing User Engagement
   Id: Test-LF3

   Type: Non-functional, Dynamic, Automated

Initial State: The user is logged into the game

Input: The user plays the game

Output: Have metric of how engaged the player is depending on how many turns they play in a session

How test will be performed: The user decisions and the number of turns played are logged and the total number of turns played will be calculated for every player and compared to MIN_TURNS. If player has played greater than or equal to MIN_TURNS, we say they were engaged.

4.2.2 Usability and Humanity

1. **Name:** Testing Learnability
   Id: Test-UH1

   Type: Non-functional, Dynamic, Manual

   Initial State: The user is at home page

   Input: The user registers account, logs in and plays the game

   Output: Get feedback and verify authorization, game logic, and interface is easy/fast to learn

   How test will be performed: A survey will be given to players to provide feedback on how easy it is to learn the different aspects of the system. This includes the game logic and interactions between the different menus

4.2.3 Performance

1. **Name:** Test the load times for logging the game
   Id: Test-P1

   Type: Non-functional, Manual, Dynamic

   Initial State: The game server is online and running, user is at home page

   Input: The user logs into the game

Output: The user is able to log into the game within LOAD_TIME

How test will be performed: The team will use a performance testing tool of choice to measure loading times of logging in to the game.

2. **Name:** Test response time of the system based on user input
   Id: Test-P2

   Type: Non-functional, Manual, Dynamic

   Initial State: The user is logged into the game

   Input: The user makes a decisions within the game and interacts with the user interface

   Output: The application provides a response to the request within RESPONSE_TIME

   How test will be performed: The team will measure the response times of different decisions within the game.

3. **Name:** Test for performance on using different image assets
   Id: Test-P3

   Type: Manual, Dynamic

   Initial State: The user is logged into the game

   Input: An existing image asset is replaced

   Output: The new image can be seen in place of the replaced image asset and the application response does not exceed RESPONSE_TIME

   How test will be performed: The team will use a performance tool of choice to automate replacing an image asset and measuring the response time of different interface interactions.

4.2.4 Operational and Environmental

1. **Name:** Testing if system can run on different browsers
   Id: Test-OE1

   Type: Non-functional, Manual, Dynamic

Initial State: The user is at home page

Input: The user registers account, logs in and plays the game on different modern web browsers

Output: The system shall be functional with no errors

How test will be performed: The system will be tested with various features of the system in order to determine the system is working correctly on different modern web browsers. This will ensure the system can be supported by that specific web browser.

2. **Name:** Test for all working versions of a web browser up to supported working version
Id: Test-OE2

Type: Non-functional, Manual, Dynamic

Initial State: The user is at home page

Input: The user registers account, logs in and plays the game on different versions of different modern web browsers

Output: The system shall be functional with no errors

How test will be performed: The system will be tested with various features of the system in order to determine the system is working correctly on different versions of that web browser. The system will test from the SUPPORTED_VERSION to the latest version of that web browser

4.2.5 Maintainability and Support Requirements

1. **Name:** Testing audio change in current game session and its affects
Id: Test-MS1

Type: Non-functional, Manual, Dynamic

Initial State: The user is logged into the game

Input: The game audio is changed to a different track

Output: The change is game audio does not affect other aspects of the game

How test will be performed: A member of the team will switch the game audio to a different track and monitor the effect on the game after the change.

4.2.6 Security

1. **Name:** Test for exploits of current technology used in the system
   Id: Test-SR1

   Type: Non-functional, Manual, Dynamic

   Initial State: The user is not logged in

   Input: User creates account

   Output: The system shall prevent the creation of accounts in suspicion of automated attacked

   How test will be performed: Use common exploit vulnerabilities with the current versions or past versions of the technology used within the system that can lead to automated attacks. This will be used on a separate test server to avoid any complications with the development server.

2. **Name:** Test for basic SQL injection attacks on the login input form
   Id: Test-SR2

   Type: Non-functional, Manual, Dynamic

   Initial State: The user is not logged in

   Input: User logs in

   Output: The system shall prevent the user from logging in through a malicious attack or from damaging the user database.

   How test will be performed: Basic cross site scripting (XSS) techniques will be used to see if the user is able to by-pass the login screen. Some techniques include persistent, non-persistent and DOM-based cross-site scripting.

3. **Name:** Test to see if plain text of password exist in database
   Id: Test-SR3

Type: Non-functional, Manual, Dynamic

Initial State: Only one user's password is encrypted in data base and admin access to data base

Input: Query passwords for passwords in database

Output: Encrypted user password

How test will be performed: Manually query the passwords in the data base and check to see if the user password is returned in plaintex

4.2.7 Access

1. **Name:** Test for accessing the system on different internet accesses (i.e public and private wifi)
   Id: Test-ACR1

   Type: Non-functional, manual, Dynamic

   Initial State: The user is not logged in

   Input: User accesses system

   Output: The system successfully loads on users system

   How test will be performed: The system will be opened on multiple systems in multiple locations (i.e public, private internet access and check to see if the system is able to be accessed by the user. However, only one user session is active at a time.

2. **Name:** Test to see if only login functionality is only provided when user is not logged in
   Id: Test-ACR2

   Type: Non-functional, Manual, Dynamic

   Initial State: The user is not logged in and at login page

   Input: N/A

   Output: The system will only provide functionality relating to logging in. The system should not allow any functionality relating to the in-game subsystem when they are not logged in.

How test will be performed: The system checked to see which request are provided when presented at the login screen when the user is not signed in. There will be checks to see if any game related logic is stored on the user's client side to see if they have access to functionality they should not be having access to while not signed in.

3. **Name:** Test to see if only in game functionality is provided when user is logged in (i.e functionality for logging should not be available)
   Id: Test-ACR3

   Type: Non-functional, Manual, Dynamic

   Initial State: The user is logged in and at game page

   Input: N/A

   Output: The system will provide all functionality relating to the game upon after logging. All functionality related to the login should not be present

   How test will be performed: The system checked to see which request are provided after the user is logged in and at the game page. There will be checks to see if all game related logic is stored on the user's client side after they are logged in and playing the game. There should be no functionality related to the login system

4. **Name:** Test to see if only one user session is activated each time a new log in appears
   Id: Test-ACR4

   Type: Non-functional, Manual, Dynamic

   Initial State: The user is logged in and at game page in one session

   Input: User attempts to login into a separate session

   Output: The system shall only allow the new session created by the user upon login to a new session to be active while deactivating the previous user session

   How test will be performed: Have one session logged in and sign in again in a different tab/window on the same machine and testing on a separate machine. Each instance of this test, only one session should be active while deactivating the previous user session

4.2.8 Integrity

1. **Name:** Testing API Response Time
   ~~**Id:** IR1~~
   ~~**Type:** Non-functional, Manual, Dynamic **Initial State:** The user is logged in **Input:** User plays the game **Output:** The system shall respond with user input request **How test will be performed:** The system shall be able to receive all input request provided by the user and each one takes at most API_RESPONSE_TIME~~

2. **Name:** Testing Database Response Time
   Id: IR2

   Type: Non-functional, Manual, Dynamic

   Initial State: N/A

   Input: System makes request to database

   Output: The database shall respond to system request

   How test will be performed: The database will receive all types of input requests provided by the system and each one takes at most DATABASE_RESPONSE_TIME

3. **Name:** Testing if progress is saved when connection is lost
   Id: IR3

   Type: Non-functional, Manual, Dynamic

   Initial State: User logged in

   Input: User disconnects from internet

   Output: The system shall keep user progress prior to loss of connection and resume gameplay upon an established connection with the server

   How test will be performed: The user will disconnect their game mid session (i.e disconnect internet, close tab, web browser client) and then reconnect and log back into their game. Check to see if progress prior to loss of connection was saved and resumed

4.3 Traceability Between Test Cases and Requirements

4.3.1 Traceability Matrix for Functional Requirement Test Cases

	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FR12	FR13	FR14	FR15	FR16	FR17	FR18	FR19	FR20	FR21	FR22
AC1	X																					
AC2	X																					
AC3		X																				
AC4					X																	
AC5															X							
GM1			X				X			X							X					
GM2				X			X										X					
GM3							X															
GM4							X		X								X					
GM5										X						X						
GM6											X											
GM7											X									X		
GM8												X								X		
GM9																					X	
GM10																						X
GM11			X			X																
GM12			X			X																
GM13			X			X																
GM14								X														
GM15								X														
GM16								X														
GM17								X														
GM18																						X
DB1													X									
DB2														X								
DB3														X								
PG1																		X				
PG2																		X				

Table 2: Traceability Matrix for Functional Test Cases

4.3.2 Traceability Matrix for Non-Functional Requirement Test Cases

	LF1	LF2	LF3	LF4	LF5	LF6	UH1	PR1	PR2	OE1	OE2	MS1	SR1	SR2	ACR1	ACR2	ACR3	ACR4	IR1	IR2	IR3
Test-LF1	X	X	X	X																	
Test-LF2					X																
Test-LF3						X															
Test-UH1							X														
Test-P1								X													
Test-P2									X												
Test-P3									X			X									
Test-OE1										X											
Test-OE2											X										
Test-MS1												X									
Test-SR1													X								
Test-SR2														X							
Test-SR3														X							
Test-ACR1															X						
Test-ACR2																X					
Test-ACR3																	X				
Test-ACR4																		X			
Test-IR1																			X		
Test-IR2																				X	
Test-IR3																					X

Table 3: Traceability Matrix for Non-functional Test Cases

5 Unit Test Description

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS] [This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 GameLogic

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. **Name:** Successful first item purchase
   Id: Test-GM11

   Control: Automatic

   Initial State: User is logged in and in the shop. User has enough in-game currency to purchase an item but does not have the item in their inventory

   Input: User is logged in. The user buys an item

   Output: The item appears in the user's inventory and the user's currency correctly reflects the outcome of buying items

   Test Case Derivation: The system must be able to verify that users can purchase a new item they do not currently own using in-game currency

   How test will be performed: Automated test to check if purchased item exists in inventory and the user's currency is correctly reflected after the purchase, in both the database and front-end

2. **Name:** Unsuccessful item purchase
   Id: Test-GM12

   Control: Automatic

   Initial State: User is logged in and in the shop. User does not have enough in-game currency to purchase an item but does not have the item in their inventory

Input: The user buys an item

Output: The item does not appear in the user's inventory and the user's currency is not changed

Test Case Derivation: The system must be able to verify that users can not purchase an item using in-game currency when they do not have enough in-game currency

How test will be performed: Automated test to check if the user has enough in-game currency to purchase that item which does not change after an unsuccessful purchase, in both the database and front-end

3. **Name:** Successful second or more item purchase of the same item
   Id: Test-GM13

   Control: Automatic

   Initial State: User is logged in and in the shop. User has enough in-game currency to purchase an item and already has the item in their inventory

   Input: The user buys an item they already have in their inventory

   Output: The quantity of the same item is increased by 1 in the user's inventory and the user's currency correctly reflects the outcome of buying items

   Test Case Derivation: The system must be able to verify that users can purchase items to use for farming purposes

   How test will be performed: Automatic test to check if purchased item count in inventory has increased and the user's currency is correctly reflected after the purchase, in both the database and front-end How test will be performed:

4. **Name:** Selling the same crop in different seasons
   Id: Test-GM14

   Control: Automatic

   Initial State: User is logged in, user is at the shop, and the user owns at least two of the same crop

Input: User sells the one of the crops in current season and the other in the next season

Output: Profit earned from selling the two of the same crops in different seasons should vary

Test Case Derivation: The user should be able to somewhat simulate a real market where crops can be sold at varying prices at different times and market

How the test will be performed: Automated test to sell two of the same crops during different time periods and ensure the prices are the different to check for random sell prices

5. **Name:** Selling the same crop in same seasons
   Id: Test-GM15

   Control: Automatic

   Initial State: User is logged in, user is at the shop, and the user owns at least two of the same crop

   Input: User sells the one of the crops in current season and the other in the next season

   Output: Profit earned from selling the two of the same crops in the same seasons should be the same

   Test Case Derivation: The user should be able to somewhat simulate a real market where crops can be sold at varying prices at different times and market

   How the test will be performed: Automated test to sell two of the same crops during the same time periods and ensure the prices are the same to check for random sell prices

6. **Name:** Successful balance increase in selling crop
   Id: Test-GM16

   Control: Automatic

   Initial State: User is logged in and at the shop

   Input: Crop to be sold

Output: Money earned through selling crop. User's currency count correctly reflects the increase in user's currency count

Test Case Derivation: The user should be able to somewhat simulate a real market where crops can be sold at varying prices depending on the time and market

How the test will be performed: Automatic test to check if user's money/balance has increased after selling a crop

7. **Name:** Successful decrease in item quantity after selling item
   Id: Test-GM17

   Control: Automatic

   Initial State: User is logged in, at the shop, and holds at least one crop in inventory

   Input: User sells one crop

   Output: The inventory count decreases by one and money earned through the sale is added to the user's in-game currency balance

   Test Case Derivation: The user should no longer have the same number of items once its sold

   How the test will be performed: Automatic test to check if crop that has been sold has been removed from the inventory

8. **Name:** Successful visual season change
   Id: test-GM18

   Control: Automated

   Initial State: User has played the game for some time

   Input: Any decisions the user has made ?

   Output: Season change

   Test Case Derivation: This is significant as it will encourage the user to plan ahead and try to optimize their decisions in the game for the best outcomes.

   How the test will be performed: Manual test by playing the game for some time and visually checking whether the season is changing or remains constant.

5.3 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

Table 4: **Symbolic Parameter Table**

Symbolic Parameter	**Description**	**Value**
NEW_ACCOUNT_STATE	The initial state of a new account	Turn 0, 9 tiles of land owned, $1000
FERTILIZER_EFFECT	The amount of turns fertilizer speeds up a crop's growth	1 turn
CONSULTING_INTERVAL	The number of turns between each consultant visit	3 turns
EVENT_OCCURRENCE	The number of turns between each random event	2/season
SEASON_LENGTH	The number of turns per season	3 turns
SCREEN_RESOLUTIONS	The list of screen resolutions that should be supported by the user interface; given as a width and height	1920 pixels x 1080 pixels, 1366 pixels x 768, pixels, 1536 pixels x 864 pixels, 1440 pixels x 900 pixels, 1280 pixels x 720 pixels
MIN_TURNS	The minimum amount of turns played needed for a study participant to be a significant data point	12 turns
LOAD_TIME	The maximum time allowed for the application to successfully load	5 seconds
RESPONSE_TIME	The maximum time allowed for the application to respond to user input	5 seconds
SUPPORTED_VERSION	The oldest supported version of browsers	1 Year

6.2 Usability Survey Questions

User Experience Survery						
The following survey will be completed upon playing the game after 20 minutes						
Look and Feel						
Minimalistic Design	0	1	2	3	4	5
	[0 = too much clutter of elements, 5 = no clutter and minimal feel]					
Consistent Color Theme	0	1	2	3	4	5
	[0 = inconsistent color theme, 5 = consistent color theme]					
Engaging Audio	0	1	2	3	4	5
	[0 = audio is terrible to listen to, 5 = audio is enjoyable to listen to]					
Engaging Graphics	0	1	2	3	4	5
	[0 = graphics are not pleasing, 5 = graphics are pleasing and comfortable]					
Usability						
Age Group	0	1				
	[0 = age less than 18, 1 = age is 18 or above]					
Easy to understand	0	1	2	3	4	5
	[0 = hard to understand, 5 = easy enough to understand]					

Appendix — Reflection

6.3 Knowledge and skills - Reflection

1. **Jest**: Two approaches to acquiring this knowledge would be to complete an in-depth tutorial on Jest Tutorial – JavaScript Unit Testing Using Jest Framework and watching a video of a react testing tutorial using React Testing Tutorial (Jest + React Testing Library). The team member will pursue following the in-depth tutorial as they will be able to learn the basics and apply these concepts when testing the backend.

    - Student: Mihail

2. **Mocha**: Two approaches to acquiring this knowledge would be to read documentation on Mocha and following an in-depth tutorial Introduction to Mocha. The team member will pursue following the in-depth tutorial as they will be able to learn the basics and apply these concepts when testing the front-end. They may refer to the documentation if they require knowledge not acquired in the tutorial.

    - Student: Brandon

3. **Code Walkthroughs**: Two approaches to mastering and acquiring this knowledge would be to read a tutorial on code walkthroughs and

to try to apply this knowledge during a work session. The team member will read the tutorial to get a better understanding of the concept as there might not be sufficient time to properly perform code walkthroughs.

- Student: Mohammad

4. **Static Testing**: Two approaches to mastering this skill would be to refer back to SFWRENG 3S03(Software Testing) notes on static testing and reading up on a static testing tutorial. The team member will refer to course notes as there are many examples covered in their notes.

   - Student: Andrew

5. **Dynamic Testing**: Two approaches to mastering this skill would be to refer back to SFWRENG 3S03(Software Testing) notes on dynamic testing and reading up on dynamic testing tutorial. The team member will refer to the tutorial as their notes do not contain as much information as the tutorial.

   - Student: Namit