Project Title: System Verification and Validation Plan for Chess Connect

Team #4,
Alexander Van Kralingen
Arshdeep Aujla
Jonathan Cels
Joshua Chapman
Rupinder Nagra

November 2, 2022

1 Revision History

Table 1: Revision History

Date	**Developer(s)**	**Change**
2022-10-31	Jonathan Cels	NFR Testing
2022-10-31	Arshdeep Aujla	Added section 3
2022-11-02	Joshua Chapman	Section 4.1, 4.2, 4.3, 4.5
2022-11-02	Alexander Van Kralingen	Completed Section 4.4, 4.6, 4.7

Contents

List of Tables

List of Figures

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test

[symbols, abbreviations or acronyms — you can simply reference the SRS (Author, 2019) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

3 General Information

3.1 Summary

The project name is Chess Connect. It is comprised of software and hardware components. The hardware will consist of a reactive chess set connected to a microcontroller. The microcontroller will relay information on the chess board in the form of LEDs of the possible moves the user can make. The software component of this project will consist of a web application that will reflect all of the chess piece's location on the physical board.

3.2 Objectives

The following objectives are the qualities that are the most important for the project.

- The hardware should reflect relevent information on the LEDs on the chess board
- The software component should reflect the physical chess board in near-real time
- The movement of the chess pieces should be recorded by the hardware

3.3 Relevant Documentation

The following documents are relevent to this project.

- SRS
- Hazard Analysis
- Requirements Document
- Design Document
- VnV Report

- MIS
- MG

4 Plan

This section discusses the general plans for hardware and software testing. The responsibilities of members are assigned and requirements verifications are discussed. Design and implementation verification include high-level plans to execute hardware and software systems. Automated testing techniques and tools are outlined. A detailed software validation plan discusses the external tools required for completeness.

4.1 Verification and Validation Team

Alexander Van Kralingen	Web application connection tests GitHub integration testing Microcontroller code testing
Arshdeep Aujla	Sensor accuracy testing Microcontroller hardware & code testing Touchscreen UI latency tests
Jonathan Cels	Microcontroller code testing Chess API integration testing Communication protocol testing
Joshua Chapman	Power distribution design and integration testing Microcontroller hardware & code testing Communication protocol testing
Rupinder Nagra	Chess API integration testing Web application connection tests Web application functionality testing

4.2 SRS Verification Plan

SRS verification is performed by the teammates throughout the design and testing process. They will reference requirements and consider the results

throughout design. Teammates will periodically verify this completeness throughout the design process. The verification and validation report will reference an associated requirement for each test. This allows for tracking of individual requirement fullfilment throughout the process.

4.3 Design Verification Plan

Hardware Design Verification begins with LTSpice and Multisim design software packages. The designed circuits are simulated in the software and tested accordingly. Testingincludes simulating inputs and verifying that expected outputs are returned. Edge cases are simulated to verify safety and failure conditions of the circuits.

Software Design Verification utilizes the compiler to verify code correctness. Arduino intergrated development environments contain the tools to compile and check for errors. Code walkthroughs are performed by collaborators that did not write the program. This includes detailed inspection and a report describing the function of the code based on their perspective. This tests the readability and functionality of the code.

4.4 Verification and Validation Plan Verification Plan

The Verification and Validation Plan verification will include reviewing the Verification and Validation Plan checklist and adjusting this document accordingly. The verification will also include considering feedback from issues created by another team, as well as from the TA assigned to this project. Comments and improvements will be implemented after completing revision 0 of this document.

4.5 Implementation Verification Plan

Hardware Implementation Verification begins with individual components. First the power supply is tested with a voltemeter for accuracy and precision. Then, the microcontroller is powered on and each individual I/O is tested for correctness. Inputs will be powered via the power supply and readings are verified for correctness using custom testing software installed on the controller. Outputs are verified using the voltemeter to measure correctness relative to the controller value. Finally, sensors and circuits are tested

using the IO of the microcontroller. This allows for accurate and detailed inspection of the components and verifies their correctness.

Software Implementation Verification requires the hardware to be tested and assembled before beginning. The software is downloaded to the controller and the inputs and outputs are configured one-by-one. Functionality of code is tested with unit tests from section 5.1 with the appropriate hardware. Once each section is unit tested, the sub-systems can begin to combine to test completness of full systems and their functionality.

4.6 Automated Testing and Verification Tools

Automated testing will be carried out for the software involved with this project. Unit tests will be created to test the functionality fo each function created for the program. A minimum of one "successful" test will be written to describe the intended program execution, and one or more "unsuccessful" tests will be written to test the robustness of the program. This will ensure complete code coverage for the software. There are 3 main classes of tests that will be run involving the different aspects Chess Connect:

- **Linting:** Performed on Python and Javascript code. This will be integrated into VS Code to assist in local development; this will be run as a Github Workflow as a non-blocking check before building the software.
  - ESLint to be used for Javascript
  - Flake8 to be used for Python
- **Unit Tests:** Detailed in Section 6, unit tests will be performed for Python, Javascript and C code.
  - React Testing Library will be used for Javascript unit tests.
  - PyTest will be used for Python unit tests.
  - AceUnit will be used for C unit tests.
- **Dynamic Analysis Tool:** in addition to creating unit tests for the C code, memory leaks and access errors will be caught using ValGrind.

Debuggers: Dynamic analysis will also be performed on the code through debuggers to verify the system and unit tests, as well as normal operation. These tools will be used extensively to bring all unit tests to a successful state, and to determine the root cause of any inconsistent behaviour from the hardware.

- The broswer's (Chrome, Firefox, etc.) built in debugger will be used for stepping through the Javascript code on the web application.
- pdb will be used for Python debugging.
- Arduino Zero Built-in Debugger Interface will be used for debugging embedded C code.

4.7 Software Validation Plan

Software will be validated by reviewing the requirements in the Software Requirement Specifications document and ensuring the software matches the expected performance and behaviour. The chess engine software integration will be validated by comparing it alongside an established online chess engine running the same Stockfish build as is selected for Chess Connect, and comparing the recommended moves. One webside that may be used is 365Chess.com.

Feedback from members of the team that are assigned to another area of the system will be called to verify the software is performing adequately. For example, a member working on the web app may come to test the hardware to ensure the sequences and behaviour is intuitive and user friendly. Feedback and comments from the professor and TA assigned to this project will also be considered when making ajustments to the software.

Unit tests will be created that capture both normal-use and failure modes to validate the software as well. Unit tests will be integrated in a Github workflow to ensure that changes are never made to the software that conflict with other areas of the code. Failing unit tests will be used as direction to change parts of the software back to a valid state of operation.

5 System Test Description

5.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

5.1.1 Area of Testing1

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Title for Test

1. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

5.1.2 Area of Testing2

...

5.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

5.2.1 Look and Feel

Style

1. NFT1
2. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

3. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

Input:

Output:

How test will be performed:

5.2.2 Usability and Humanity

Learnability

1. NFT2

   Type: Structural, Static, Manual

   Initial State: Product is in normal mode, a game has started, and the users have not interacted with the product before.

   Input/Condition: Users are asked to use the product and move one specified piece from one square to another specified square on the board.

   Output/Result: The majority of users understand which piece they moved and to where, and are able to identify that the web application has reflected their move in the virtual model within 30 seconds of studying the visual representation of the board state.

   How test will be performed: A test group of people who do not play chess regularly are asked to move pieces on the chessboard. They are then asked to identify the move they just made as reflected in the web application model. The subjects must identify that the piece they moved on the board has also been moved on the web application's virtual model within 30 seconds or less, averaged over the number of people in the test group.

2. NFT3

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State: The product shall be representing the state of the game that is in progress. The pieces shall be in a legal position according to the rules of chess. The pieces shall not all be in their starting positions, and there is at least one of each type of piece (pawn, knight, bishop, rook, queen, king) on the board.

   Input: Users are asked to identify the names of different pieces and squares based on their visual appearance in both the physical product and the web application.

Output: The majority of users are able to identify the names of pieces and squares based on their likeliness to historically used symbols and shapes.

How test will be performed: A test group of people who have played chess in the past or play chess regularly are asked to identify each of the pieces and squares from an in-progress game of chess using the system. The justification for this is to avoid piece identification based on their starting positions. The majority of the group should be able to visually identify every piece and square within 2 minutes of seeing the position for the first time.

5.2.3 Performance

Speed and Latency

1. NFT4

   Type: Structural, Static, Manual

   Initial State: The product is in normal mode, and a game has started.

   Input/Condition: Users are asked to pick up a piece and suspend it midair without placing it down.

   Output/Result: The board shall visually indicate where the held piece is able to move according to the rules of chess within a specific time frame.

   How test will be performed: A test group of people who have played chess in the past or play chess regularly are asked to pick up a specific piece and hold it. The system will give visual indicators of where the held piece is able to move according to the rules of chess. The time between when they pick up the piece and when the visual response occurs is measured and recorded. This process is repeated 5 times per user. The individual and average times are recorded. The response times are then averaged over the entire test group. The average response time of the entire test group must be less than 0.5 seconds.

2. NFT5

   Type: Structural, Static, Manual

Initial State: The results of the previous test, NFT4, have been measured and recorded.

Input/Condition: The results of NFT4.

Output/Result: The maximum recorded time of any individual response is within a specific time frame.

How test will be performed: The times measured in the previous test, NFT4, will be inspected. The maximum recorded individual time must be less than 1 second.

3. NFT6

   Type: Structural, Static, Manual

   Initial State: The product is in normal mode, and a game has started.

   Input/Condition: Users are asked to pick up a piece and legally move it to a square according to the rules of chess.

   Output/Result: The web application shall reflect their move in the virtual model within a specific time frame.

   How test will be performed: A test group of people who have played chess in the past or play chess regularly are asked to pick up a specific piece and legally move it to another square according to the rules of chess. The web application will reflect their move in the virtual model. The time between when they place down the piece and when the web application response occurs is measured and recorded. This process is repeated 5 times per user. The individual and average times are recorded. The response times are then averaged over the entire test group. The average response time of the entire test group must be less than 2 seconds.

4. NFT7

   Type: Structural, Static, Manual

   Initial State: The results of the previous test, NFT6, have been measured and recorded.

   Input/Condition: The results of NFT6.

   Output/Result: The maximum recorded time of any individual response is within a specific time frame.

How test will be performed: The times measured in the previous test, NFT6, will be inspected. The maximum recorded individual time must be less than 5 seconds.

Health and Safety

1. NFT8

   Type: Structural, Static, Manual

   Initial State: The product is in normal mode, and a game has started.

   Input/Condition: 10 wires are chosen as a sample.

   Output/Result: The maximum power on any single wire shall be within the required limit.

   How test will be performed: A sample of 10 wires are chosen arbitrarily from across the entire system. The voltage and amperage of each wire in the sample are measured and recorded. The power shall then be calculated and recorded. The maximum power of any wire in the sample must not exceed the safe limits determined in the Canadian Electrical Code CSA (2021).

Precision and Accuracy

1. NFT9

   Type: Structural, Static, Manual

   Initial State: The product is in the initial game state.

   Input/Condition: Users are instructed to play a full game of chess using the Chess Connect system.

   Output/Result: The web application will properly reflect the moves made on the physical product the majority of the time.

   How test will be performed: A test group of people who have played chess in the past or play chess regularly are asked to play a full game of chess. Their moves and the web application response will be recorded and compared against each other. The number of discrepancies between the physical moves and moves made on the web application will be recorded. The number of discrepancies averaged over the entire test group must be less than or equal to 1.

Capacity

1. NFT10

   Type: Structural, Static, Manual

   Initial State: The product is in the initial game state and is in engine mode.

   Input/Condition: Moves are made until the game state is in one of a set of predetermined computationally complicated chess positions.

   Output/Result: The level of memory used by the web application shall be no more than 1 Gigabyte (GB) at any measured point.

   How test will be performed: Moves will be made until the game state is in one of a set of predetermined computationally complicated chess positions. The engine will then be using the maximum amount of memory to compute the best possible moves for the position. The amount of memory used will be measured and recorded using windows task manager. The amount of memory must never exceed 1 GB at any measured point.

5.2.4 Security

Integrity

1. NFT11

   Type: Structural, Static, Manual

   Initial State: The product is in normal mode and a chess game is in progress on the system.

   Input/Condition: The Bluetooth connection is severed between the web application and the product.

   Output/Result: The web application indicates that the Bluetooth connection has been lost.

   How test will be performed: A game of chess is being played when the Bluetooth option is switched off on the server, severing the connection. The web application must display an alert that the Bluetooth connection has been lost.

1. NFT12

   Type: Structural, Static, Manual

   Initial State: The product is in normal mode and a chess game is in progress on the system.

   Input/Condition: The power connection to the system is severed.

   Output/Result: The system stores the game state in local memory until power is restored.

   How test will be performed: A game of chess is being played when the power is switched off. The power is then restored after 5 or more seconds. A single move is made and the state of the game is tested against the web application. The state of the game should be unchanged from before power was lost.

5.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

6 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]

6.1 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]

6.2 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]

6.2.1 Module 1

[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. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

6.2.2 Module 2

...

6.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

6.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

6.3.2 Module ?

...

6.4 Traceability Between Test Cases and Modules

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

References

Author Author. System requirements specification. https://github.com/..., 2019.

CSA. *Canadian Electrical Code*. CSA Group, 2021.

Appendix — Reflection

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

1. **Arduino Zero Built-in Debugger Interface** is necessary to simulate and test the arduino code on the microcontroller. Learning the software requires reading documentation for installing and operating the debugger. As well, interpreting the outputs from the debugger requires practice. Joshua Chapman will be taking this responsibility.

2. **React Testing Library** is used for testing web application code. All coding in the web application will be done using javascript and react. Documentation for using the library will be read and experimented to develop a thorough understanding. Rupinder Nagra and Jonathan Cels will be taking this responsibility.

3. **Pytest** is used to perform unit tests on python code for custom microcontroller programming and bluetooth connection. Documentation to use the desired functionality will be read. Additionally, a learning period will be required to properly learn the outputs of the unit tester. Jonathan Cels and Arsheep Aujla will be taking this responsibility.

4. **Flake8** simplifies the Github workflow by performing non-blocking checks on integrated code. Integrating it into Github will require reading documentation and practicing before important deadlines. Alexander Van Kralingen will be taking this responsibility.