SE 3XA3: Test Plan
SupremeChess

Team 2, The Triple Grobs
Jonathan Cels (celsj)
Rupinder Nagra (nagrar5)
Pesara Amarasekera (amarasep)

March 5, 2021

Contents

List of Tables

Table 1: **Revision History**

Date	Version	Notes
2021-02-23	1.0	Created document, completed up to functional requirement tests
2021-03-02	1.1	Finished functional requirements, started non-functional requirement tests
2021-03-03	1.2	Finished non-functional requirements and the rest of the document
2021-03-05	1.3	Minor edits and changes

1 General Information

1.1 Purpose

The purpose of testing this system is to detect and correct software failures, increasing confidence that the system was implemented correctly.

1.2 Scope

The project, SupremeChess, is a chess game played ~~online by one player against an AI opponent, or~~ by two players against each other. The scope of testing for this project will cover three main areas:

1. The implementation of the chess game logic. This includes testing moves, game states, and game termination.
2. Surrounding functionality such as the timers, the menu, the chat feature, ~~and the AI opponent~~.
3. ~~The online multiplayer functionality against an AI opponent or another user.~~

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
POC	Proof of Concept
GUI	Graphical User Interface
SRS	Software Requirements Specification

1.4 Overview of Document

The SupremeChess project plans to re-implement the open source project "Online-Chess-Game". The software will allow users to play chess with anyone on a web server. All software requirements of the project are stated in the Software Requirements Document in the SRS folder of the 3XA3-Group2-Chess Git Repository.

Table 3: **Table of Definitions**

Term	Definition
Static Testing	Testing that does not involve program execution.
Dynamic Testing	Testing that requires program execution.
Manual Testing	Testing that is conducted by people by hand.
Automated Testing	Testing that is done automatically by software.
Structural Testing	Testing that is derived from the structure of the software system.
Functional Testing	Testing that is derived from the functionality of the software system.
Equivalence Testing	Testing that partitions possible inputs based on their behaviour and tests them together.

2 Plan

2.1 Software Description

The software will allow a user to play a game of chess against another user ~~online, or against an AI opponent~~. The implementation will be completed in JavaScript using Node.js and React.

2.2 Test Team

The individuals responsible for testing the system are Jonathan Cels, Rupinder Nagra, and Pesara Amarasekera.

2.3 Automated Testing Approach

~~Automated testing will be done using the software testing tools specified below, Enzyme and Mocha. The team will write unit tests for React components using Enzyme, and write the test suite using Mocha. Further detail on the unit testing strategy can be found in section 6, the Unit Testing Plan. The test suite will be run after each major commit to the master branch, and will be run before and after merging any branches to ensure that nothing is broken on addition of new components.~~

Automated testing will be done using the default jest library installed

with create-react-app. However, there is no automated testing required for this system as manual testing is more effective for highly visual components.

2.4 Testing Tools

1. ~~**Enzyme**: Enzyme will be the tool used to test React components.~~
2. ~~**Mocha**: Mocha will be used to write the automated test suite.~~
3. **Jest** will be the tool used to test React components and will be run using the built-in 'react-test script'.

2.5 Testing Schedule

A Gantt chart detailing the project and testing schedule can be found at the following link.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 ~~Select Opponent~~ Join Chat Room

Connection to server

1. T-FR1-1

   Type: Manual, Dynamic, Structural.

   Initial State: A user is attempting to connect to the web server.

   Input: Valid connection request.

   ~~Output: The user is presented with the options of playing against an AI or online player. A successful connection will verify that the connection was made with a console output message.~~

   Output: A successful connection will verify that the connection was made with a console output message.

   How test will be performed: A team member will manually attempt to connect to the server and will ensure that the server verifies the connection in the console.

2. T-FR1-2

   Type: Manual, Dynamic, Structural.

   Initial State: A user is attempting to connect to the web server.

   Input: Invalid connection request.

   Output: An exception is generated, the user is presented with a connection error, and no connection is made.

   How test will be performed: A team member will manually attempt to connect to the server while the server is not running and will verify that an error is displayed and an exception occurs.

~~Select AI opponent or online player~~ Starting Screen

1. ~~T-FR2-1~~

   ~~Type: Manual, Static, Structural. Initial State: A user is presented with the options of playing against an AI opponent or online player. Input: User selects an online player. Output: An invite link visible to the player. How test will be performed: A team member will choose to play against an online player and verify that an invite link is generated.~~

2. ~~T-FR2-2~~

   ~~Type: Functional, Dynamic, Manual. Initial State: A user is presented with the options of playing against an AI opponent or online player. Input: User selects an AI opponent. Output: The system attempts to start a game against an AI opponent. How test will be performed: A team member will choose to play against an AI opponent and verify that the system attempts to start a game.~~

3. T-FR2-3

   Type: Manual, Dynamic, Structural.

   Initial State: A user is attempting to connect to the web server.

   Input: The system will run the welcome screen.

Output: The user is able to see the starting screen.

How test will be performed: A team member will manually attempt to join this page of the application.

Create Chat Room

1. T-FR3-1

   Type: Manual, Dynamic, Structural.

   Initial State: The user is presented with a welcome screen.

   Input: The system shall present the user with options to input a username and room name.

   Output: The user shall join the specified chat room.

   How test will be performed: Members of the team will manually attempt to be granted access to join a chat room.

Chat Room Requirements

1. T-FR4-1

   Type: Manual, Functional, Dynamic.

   Initial State: The user attempts to join a chat room after inputting a username and room name.

   Input: The system shall send the user input data to the server.

   Output: The system shall connect the user to the specified chat room.

   How test will be performed: Two members of the team will manually attempt to join a chat room to interact with one another.

1. T-FR4-2

   Type: Manual, Functional, Dynamic.

   Initial State: The user attempts to join a chat room after inputting a username and room name.

   Input: The user sends the incorrect input data to the server.

Output: The system shall not connect the user to the specified chat room.

How test will be performed: Two members of the team will manually attempt to join a chat room to interact with one another.

3.1.2 ~~Play Against Live Opponent~~

~~Pair with Invite Link~~

1. ~~T-FR5-1~~

   ~~Type: Manual, Dynamic, Structural. Initial State: A user has connected to the web server. Input: User chooses the option of playing against a live opponent and inputs a valid invite link. Output: A successful invitation will grant access for the player to join the game and play. The system will verify that access was granted with a console output message. How test will be performed: Two members of the team will manually attempt to be granted access to play one another.~~

2. ~~T-FR5-2~~

   ~~Type: Manual, Dynamic, Structural. Initial State: A user has connected to the web server. Input: User chooses the option of playing against a live opponent and inputs an invalid invite link. Output: An unsuccessful invitation will not provide access for a player to join a game and play against another opponent. The system will show that access was not granted with an exception. How test will be performed: Two members of the team will manually attempt to be granted access to play with one another, but fails.~~

~~Options while Awaiting Player~~

1. ~~T-FR6-1~~

   ~~Type: Manual, Functional, Dynamic. Initial State: The user has a valid invite link and is awaiting player. Input: The user chooses to quit the game. Output: The invite link will expire the player will no longer be awaiting a player. How test will be performed: Two members~~

~~of the team will manually attempt to play with one another, but one member quits the game while awaiting the other player to join.~~

2. ~~T-FR6-2~~

   ~~Type: Manual, Functional, Dynamic. Initial State: The user has a valid invite link and is awaiting player. Input: The user chooses to play against the AI. Output: The invite link will expire and the system will now set up the game to play against the AI. How test will be performed: Two members of the team will manually attempt to play with one another, but one member will choose to play against the AI while awaiting the other player to join.~~

Terminate Game

1. ~~T-FR7-1~~

   ~~Type: Manual, Functional, Dynamic. Initial State: The user has a valid invite link and is awaiting player. The user has input a room name and has connected to the server. Input: The user chooses to quit the game. Output: The current game shall be terminated. How test will be performed: Two members of the team will manually attempt to play with one another, but one member leaves the game. while awaiting the other player to join.~~

~~User Selects the Quit Option~~

1. ~~T-FR8-1~~

   ~~Type: Manual, Functional, Dynamic. Initial State: The user has a valid invite link and is awaiting player. Input: The user chooses to quit the game. Output: The current game shall be terminated. How test will be performed: Two members of the team will manually attempt to play with one another, but one member leaves the game while awaiting the other player to join.~~

~~**User Selects the Play Against AI Option**~~

1. ~~T-FR9-1~~

   ~~Type: Manual, Functional, Dynamic. Initial State: The user has a valid invite link and is awaiting player. Input: The user chooses to play against the AI. Output: The invite link will expire and the system will now set up the game to play against the AI. How test will be performed: Two members of the team will manually attempt to play with one another, but one member will choose to play against the AI while awaiting the other player to join.~~

~~**Present Loading Screen**~~

1. ~~T-FR10-1~~

   ~~Type: Manual, Functional, Dynamic. Initial State: Both users have joined the invite link. Input: The users join the game and the loading screen is initialized. Output: The system presents the players a view of the board. How test will be performed: Two members of the team will join a game, and system shall check if the loading screen has been initialized.~~

Call "Start Game" use case

1. ~~T-FR11-1~~

   ~~Type: Manual, Functional, Dynamic. Initial State: The system presents the players a view of the board. Input: The game will begin to call the "Start Game" use case. Output: The system initializes the game after calling the "Start Game" use case, and both players may now play. How test will be performed: Two members of the team will join a game, and system shall check if the "Start Game" use case has been called.~~

Accept or Reject Game

1. ~~T-FR12-1~~

   ~~Type: Manual, Functional, Dynamic. Initial State: The game calls the "Start Game" use case. Input: The current server information and concurrent users are checked in the "Start Game" use case. Output: The system shall reject the request and return to the "Select Opponent" use case. How test will be performed: Two members of the team will join a game, and system shall check if the "Start Game" use case successfully completes with less and more than 15 concurrent users.~~

3.1.3 ~~Play Against AI~~

~~Call "Start Game" use case~~

1. ~~T-FR13-1~~

   ~~Type: Manual, Functional, Dynamic. Initial State: The user is faced with the option of playing a Live Opponent or an AI. Input: The user selects to "Play Against AI". Output: The system initializes the game by calling the "Start Game" use case. How test will be performed: One member of the team will join a game, and system shall check if the "Start Game" use case has been called.~~

~~No Chat Option~~

1. ~~T-FR14-1~~

   ~~Type: Manual, Functional, Dynamic. Initial State: The game calls the "Start Game" use case. Input: The game between the player and AI has begun. Output: The system shall restrict the chat window if the player is playing against the AI. How test will be performed: One member of the team will join a game, and system shall check if the user is able to talk to the AI player.~~

~~Accept or Reject Game~~

1. ~~T-FR15-1~~

~~Type: Manual, Functional, Dynamic. Initial State: The game calls the "Start Game" use case. Input: The current server information and concurrent users are checked in the "Start Game" use case. Output: The system shall reject the request and return to the "Select Opponent" use case. How test will be performed: Two members of the team will join a game, and system shall check if the "Start Game" use case successfully completes with less and more than 15 concurrent users.~~

3.1.4 Start Game

~~**User Chooses "AI"**~~

1. ~~T-FR16-1~~

~~Type: Manual, Functional, Dynamic. Initial State: The user is faced with the option of playing a Live Opponent or an AI. Input: The user chooses "AI" from the "Select Opponent" use case Output: The server shall connect to an AI player and arbitrarily assign black or white to each player. How test will be performed: One member of the team will join a game with an "AI" and check if a game is successfully started.~~

~~**User Chooses "Online Player"**~~ **User Starts Game**

1. ~~T-FR17-1~~

~~Type: Manual, Functional, Dynamic. Initial State: The user is faced with the option of playing a Live Opponent or an AI. The user is presented the option to input a username and room name. Input: The user chooses "Online Player" from the "Select Opponent" use case. The user connects to a room and starts a game. Output: A shareable link shall be created with which a player can join the server. If a player joins, each player shall be arbitrarily assigned black or white. The user shall join the room specified by the input room name and that player shall have a game started on their machine. How test will be performed: One member of the team will join a game and check if a game is successfully started.~~

Initialize Board

1. T-FR18-1

   Type: Manual, Functional, Dynamic.

   Initial State: ~~The user is faced with the option of playing a Live Opponent or an AI.~~ The user is presented the option to input a username and room name.

   Input: ~~The user chooses "AI" or "Online Player" from the "Select Opponent" use case.~~ The user connects to a room and starts a game.

   Output: The board and pieces shall be initialized, and the orientation of the board will be determined based on the colour of the users pieces.

   How test will be performed: One member of the team will start a game and check to see if the board has been appropriately initialized.

Initialize Timer

1. T-FR19-1

   Type: Manual, Functional, Dynamic.

   Initial State: ~~The user chooses "AI" or "Online Player" from the "Select Opponent" use case.~~ The user has input a username and room name.

   Input: The board and pieces have been initialized.

   Output: The timer shall be initialized with a countdown of 10 minutes for each player, and the game control is dependent on the player with the white pieces.

   How test will be performed: One member of the team will start a game and check to see if the timers have been appropriately initialized.

Give Control to White Pieces

1. T-FR20-1

   Type: Manual, Functional, Dynamic.

   Initial State: The board and pieces have been initialized.

Input: The timers have been initialized.

Output: The system shall give exclusive control to the player with the white pieces and start that player's timer.

How test will be performed: One member of the team will start a game and check to see if white pieces have been given control and their time has begun.

3.1.5 Make Move

Await Opponent Move

1. T-FR21-1

   Type: Manual, Functional, Dynamic.

   Initial State: The board and pieces have been initialized.

   Input: The system has given exclusive control to the player with the white pieces and has started that player's timer.

   Output: The system shall not allow the player to make a move until their opponent has played a move, with the exception of the first move of the game.

   How test will be performed: One member of the team will start a game and check to see if they are allowed to make a move before their opponent if they have the black pieces.

Move Piece

1. T-FR22-1

   Type: Manual, Functional, Dynamic.

   Initial State: The system has given exclusive control to the player with the white pieces and has started that player's timer.

   Input: The system shall not allow the player to make a move until their opponent has played a move, with the exception of the first move of the game.

Output: If a player clicks a piece, holds the mouse down, drags the piece to a square, and releases the mouse, the system shall register an attempted move.

How test will be performed: One member of the team will start a game and check to see if they are allowed to drag and drop a piece to make a move on the board.

Check if the Move is Legal

1. T-FR23-1

   Type: Manual, Functional, Dynamic.

   Initial State: The system shall not allow the player to make a move until their opponent has played a move, with the exception of the first move of the game.

   Input: If a player clicks a piece, holds the mouse down, drags the piece to a square, and releases the mouse, the system shall register an attempted move.

   Output: The application shall confirm that the move is legal as defined by the rules of the game.

   How test will be performed: One member of the team will start a game and check to see if the "Check Legal Move" use case is triggered after moving a piece.

The Move is Legal

1. T-FR24-1

   Type: Manual, Functional, Dynamic.

   Initial State: If a player clicks a piece, holds the mouse down, drags the piece to a square, and releases the mouse, the system shall register an attempted move.

   Input: The application shall confirm that the move is legal as defined by the rules of the game.

Output: If the attempted move is legal, the move shall be sent to the server and the opposite player shall be given control.

How test will be performed: One member of the team will make a legal move and ensure that it is reflected in the board state and that they are unable to make a second move with the same pieces.

The Move is Not Legal

1. T-FR25-1

   Type: Manual, Functional, Dynamic.

   Initial State: If a player clicks a piece, holds the mouse down, drags the piece to a square, and releases the mouse, the system shall register an attempted move.

   Input: The application shall confirm that the move is legal as defined by the rules of the game.

   Output: If the attempted move is not legal, the move shall not be sent to the server and the player shall retain control.

   How test will be performed: One member of the team will make an illegal move and ensure that it is not reflected in the board state and that they are able to attempt another move with the same pieces.

Check "Terminate Game" use case

1. T-FR26-1

   Type: Manual, Functional, Dynamic.

   Initial State: If a player clicks a piece, holds the mouse down, drags the piece to a square, and releases the mouse, the system shall register an attempted move.

   Input: The application shall confirm that the move is legal as defined by the rules of the game.

   Output: The 'Terminate Game' use case shall be checked after every move is made.

   How test will be performed: One member of the team will check to see if the "Terminate Game" use case is triggered after moving a piece.

Resume Game Timer

1. T-FR27-1

   Type: Manual, Functional, Dynamic.

   Initial State: The application shall confirm that the move is legal as defined by the rules of the game.

   Input: The "Terminate Game" use case shall be checked after every move is made.

   Output: If the game is not terminated, the timer for the other player shall resume.

   How test will be performed: One member of the team will check to see if the timers swap successfully while the game is not terminated.

3.1.6 Check Legal Move

Verify move

1. T-FR28-1

   Type: Manual, Dynamic.

   Initial State: A user has made a move.

   Input: A valid move based on the current game state.

   Output: A positive signal will be returned to signify that the game state should be updated to reflect the input move.

   How test will be performed: The system will input a valid move into the check legal move module and verify that a positive signal is returned.

2. T-FR28-2

   Type: Manual, Dynamic

   Initial State: A user has made a move.

   Input: An invalid move based on the current game state.

   Output: A negative signal will be returned to signify that the game state should not be updated.

How test will be performed: The system will input an invalid move into the check legal move module and verify that a negative signal is returned.

Update Board State

1. T-FR29-1

   Type: Manual, Functional, Dynamic.

   Initial State: A move has been made and a signal has been passed returned.

   Input: An inputted move and a positive or negative signal representing if the move was legal.

   Output: An update to the board state data representing a legal move made.

   How test will be performed: The system will input a valid move and verify that the same move is reflected on the board state.

1. T-FR29-2

   Type: Manual, Functional, Dynamic.

   Initial State: A move has been made and a signal has been passed returned.

   Input: An inputted move and a positive or negative signal representing if the move was legal.

   Output: An update to the board state data representing a legal move made.

   How test will be performed: The system will input an invalid move and verify that the same move is not reflected on the board state.

Lock Player Control

1. T-FR30-1

   Type: Manual, Functional, Dynamic.

Initial State: A legal move has been made and the board state has been updated.

Input: Both players attempt to make a move simultaneously.

Output: Only one move will occur.

How test will be performed: The system will input a valid move from both players simultaneously and verify that only the move of the player with control is reflected on the board state.

3.1.7 Initiate Chat

Present Start Chat Option

1. T-FR31-1

   Type: Manual, Functional, Dynamic.

   Initial State: A game has been started between two players.

   Input: None.

   Output: An option to start chat is presented to the players.

   How test will be performed: Two team members will start a game with each other and verify that they both have an option to start chat.

Open Chat Window

1. T-FR32-1

   Type: Manual, Dynamic.

   Initial State: A game has been started between two players.

   Input: One player has selected the start chat option.

   Output: A chat window is opened for both players.

   How test will be performed: Two team members will start a game with each other and verify that one of them starting the chat opens a chat window for both players.

Open Chat Window Simultaneously

1. T-FR33-1

   Type: Manual, Dynamic.

   Initial State: A game has been started between two players.

   Input: Both players have selected the start chat option simultaneously.

   Output: A chat window is opened for both players.

   How test will be performed: Two team members will start a game with each other and verify that both of them starting the chat simultaneously opens a chat window for both players.

Present End Chat Option

1. T-FR34-1

   Type: Manual, Functional, Dynamic.

   Initial State: A chat has been started between two players.

   Input: The chat window has been opened between both players.

   Output: An option to end chat is presented to the players.

   How test will be performed: Two team members will start a game and chat with each other and verify that they both have an option to end chat.

3.1.8 End Chat

Present End Chat Option

1. T-FR35-1

   Type: Manual, Functional, Dynamic.

   Initial State: A chat has been started between two players.

   Input: The chat window has been opened between both players.

   Output: An option to end chat is presented to the players.

How test will be performed: Two team members will start a game and chat with each other and verify that they both have an option to end chat.

Close Chat Window

1. T-FR36-1

   Type: Manual, Functional.

   Initial State: A chat has been started between two players.

   Input: One player has selected the end chat option.

   Output: The chat window has disappeared for both players and an option to start chat is presented to the players.

   How test will be performed: Two team members will start a chat with each other, one will end the chat, and they will verify that the chat window has disappeared and both players have an option to start chat.

Close Chat Window Simultaneously

1. T-FR37-1

   Type: Manual, Functional.

   Initial State: A chat has been started between two players.

   Input: Both players have selected the end chat option simultaneously.

   Output: The chat window has disappeared for both players and an option to start chat is presented to the players.

   How test will be performed: Two team members will start a chat with each other, both will end the chat simultaneously, and they will verify that the chat window has disappeared and both players have an option to start chat.

Clear Chat Data

1. T-FR38-1

   Type: Manual, Functional, Dynamic.

   Initial State: A chat has been ended between two players.

   Input: The chat window has been closed between both players.

   Output: A new chat window will have no previous chat data.

   How test will be performed: Two team members will start a chat with each other, type some message in the chat, and then end the chat. One will then start the chat again and verify that no previous messages are in the chat window for ~~either player.~~ that player. The other team member will verify that all previous messages are still in their chat window.

3.1.9 Start Timers

Start Timer On Game Start

1. T-FR39-1

   Type: Manual, Functional, Dynamic, Structural.

   Initial State: A game has been started between two players.

   Input: None.

   Output: One timer has been initialized for each player. The timer for the player with the white pieces has started counting down.

   How test will be performed: Two team members will start a game and verify that they each have a timer and that the timer for the player with the white pieces has started counting down.

Initialize Timer Values

1. T-FR40-1

   Type: Functional, Dynamic.

   Initial State: A game has been started between two players.

Input: None.

Output: One timer has been initialized for each player with 10:00 minutes of total time.

How test will be performed: Two team members will start a game and verify that they each have a timer with 10:00 minutes of total time.

Pause and Resume Timers

1. T-FR41-1

   Type: Manual, Functional, Dynamic.

   Initial State: A game has been started between two players.

   Input: Both players input valid moves in valid sequence.

   Output: After each valid move, the player who made the move will have their timer paused and the other timer will resume.

   How test will be performed: Two team members will start a game with each other and make valid moves in a valid sequence. They will verify that the timers are pausing and resuming after each valid move.

1. T-FR41-2

   Type: Manual, Functional, Dynamic.

   Initial State: A game has been started between two players.

   Input: Both players input invalid moves in valid sequence.

   Output: After each invalid move, the player who made the move will still have their timer going, and the other player will still have their timer paused.

   How test will be performed: Two team members will start a game with each other and make invalid moves. They will verify that the timers are not pausing and resuming after any invalid moves.

3.1.10 Terminate Game

Establish System Checks

1. T-FR42-1

   Type: Manual, Functional, Dynamic.

   Initial State: The ‘Terminate Game‘ use case shall be checked after every move is made.

   Input: The system will have checks to establish if the game shall be terminated with the appropriate use case possibilities.

   Output: These checks include the Checkmate, Stalemate, Draw, Resignation, and Timeout use cases.

   How test will be performed: One member of the team will start a game and check to see if the “Terminate Game” use case is triggered.

3.1.11 Checkmate

Check if a Checkmate has occurred

1. T-FR43-1

   Type: Manual, Functional, Dynamic.

   Initial State: The ‘Terminate Game‘ use case shall be checked after every move is made.

   Input: The system shall determine if the opposing player is in ‘Check‘ (state where the King is under attack) and has no legal moves for the King.

   Output: If true, this shall result in a victory with ‘Checkmate‘ for the current player.

   How test will be performed: One member of the team will play a game and check to see if they have achieved a ’Checkmate’.

3.1.12 Stalemate

Check if a Stalemate has occurred

1. T-FR44-1

   Type: Manual, Functional, Dynamic.

   Initial State: The ‘Terminate Game‘ use case shall be checked after every move is made.

   Input: The system shall determine if the opponent player is not in ‘Check‘ and has also no legal moves.

   Output: If true, this shall result in a draw with ‘Stalemate‘ for both players.

   How test will be performed: One member of the team will play a game and check to see if they have achieved a ’Stalemate’.

3.1.13 Draw

Check if a Draw has occurred

1. T-FR45-1

   Type: Manual, Functional, Dynamic.

   Initial State: The ‘Terminate Game‘ use case shall be checked after every move is made.

   Input: The system shall present the user with the ‘Offer Draw‘ button to offer a draw to the opponent.

   Output: If true, the draw must be accepted to end the game.

   How test will be performed: One member of the team will play a game and check to see if they have achieved a ’Draw’.

3.1.14 Resignation

Check if a Resignation has occurred

1. T-FR46-1

   Type: Manual, Functional, Dynamic.

   Initial State: The ‘Terminate Game‘ use case shall be checked after every move is made.

   Input: The system shall present the user with a decision to end the game with the ‘Resign‘ button.

   Output: If true, this shall immediately end the game.

   How test will be performed: One member of the team will play a game and check to see if they have achieved a ’Resignation’.

3.1.15 Timeout

Check if a Timeout has occurred

1. T-FR47-1

   Type: Manual, Functional, Dynamic.

   Initial State: The ‘Terminate Game‘ use case shall be checked after every move is made.

   Input: The system shall determine if the user is out of time.

   Output: If true, this causes that player to lose the game.

   How test will be performed: One member of the team will play a game and check to see if they have achieved a ’Timeout’.

3.2 Tests for Nonfunctional Requirements

3.2.1 Usability and Humanity

Ease of Use

1. T-UH2-1

   Type: Manual, Functional(Usability), Dynamic

   Initial State: The game is started and presented to a ten 10 year old children.

Input/Condition: The children are presented with the interface to play the game. The children should not have played the game previously before-hand.

Output/Result: Around 90% of the test subjects should be able to play a move within five minutes of starting the presented program without any assistance.

How test will be performed: The test will be performed by presenting the game to random 10-year-olds in the populace.

2. ~~T-UH5~~

~~Type: Functional(Learning), Dynamic, Manual Initial State: The program is started and presented to a panel of 10 year old children. Input: The children are presented with the interface to play the game. The children should not have played the game previously or interacted with the system in any way. Output/Result: At least 90% of the test subjects should be able to start a game and play a move within five minutes of starting the presented program without any assistance. How test will be performed: The test will be performed by presenting the game to a selected panel of 10-year-olds in the populace. They will be presented with the system on the main page and will be asked to start playing. A team member will measure the time between each person being presented with the system and that person making their first move.~~

3.2.2 Performance Requirements

System Response Time

1. T-PR1-1

Type: Functional, Dynamic

Initial State: A user is presented with a game that has started.

Input: The user will try to drag and drop a piece in place. An internal clock specifically for testing will start every time a user selects a piece and ends when the visual display updates.

Output: The output time measured, on average (taking the mean and mode), should be less than 0.25 seconds

How test will be performed: Games will be played by various test users, and measurements will be taken from all of them (at least 10 games). These times will be averaged by a program, which will verify that the average is less than 0.25 seconds.

2. T-PR2-1

   Type: Functional, Dynamic

   Initial State: A user is presented with a game that has started.

   Input: The user will try to drag and drop a piece in place. A timer, an internal clock specifically for testing, will start every time a user starts a motion and will end when the visual display updates.

   Output: The maximum response time measured (out of all trials) should be less than 2 seconds

   How test will be performed: Games will be played by various test users, and measurements will be taken from all of them (at least 10 games) and the maximum response time will be taken from the data points. The system will then run through every response time and verify that they are all less than 2 seconds.

3. T-PR3-1

   Type: Functional, Dynamic

   Initial State: A user is presented with a game that has started.

   Input: The user will try to drag and drop a piece in place. An internal clock specifically for testing will start every time a user selects a piece and will end when the opposing player's turn timer resumes.

   Output: The maximum response time measured (out of all trials) should be less than 0.1 seconds.

   How test will be performed: Games will be played by various test users, and measurements will be taken from all of them (at least 10 games) and the maximum response time will be taken from the data points.

The system will then run through every response time and verify that they are all less than 0.1 seconds.

Ram Usage

1. T-PR8-1

   Type: Functional, Dynamic, Manual

   Initial State: A user is presented with a game that has started. A program, such as Windows Task Manager, will run on the background to help take measurements.

   Input: Multiple games (at least 10) will be played by various users on the same computer.

   Output: The amount of RAM consumed when the game is running should be checked to be less than 2GB.

   How test will be performed: A team member will observe the program used for bench marking while a user uses the program and will verify that the RAM usage never exceeds 2GB.

3.3 Traceability Between Test Cases and Requirements

Requirement-driven test cases were named with the following naming convention: T-Req#-Test#. T stands for test, Req# represents a specific requirement, such as FR1 or UH2, and Test# represents the numbering for multiple tests on the same requirement. Eg. T-FR1-2 represents the second test for functional requirement with ID FR2.

3.3.1 Traceability Matrix

Table 4: **Traceability Matrix**

Requirement ID	Description	Test ID
FR1	Server Connection	T-FR1-1
FR1	Server Connection	T-FR1-2

FR2	~~Select AI Opponent or Online Player~~	T-FR2-1
FR2	~~Select AI Opponent or Online Player~~	T-FR2-2
FR2	Starting Screen	T-FR2-3
FR3	Create Chat Room	T-FR3-1
FR4	Chat Room Requirements	T-FR4-1
FR4	Chat Room Requirements	T-FR4-2
FR5	~~Pair with invite link~~	T-FR5-1
FR5	~~Pair with invite link~~	T-FR5-2
FR6	~~Options while Awaiting Player~~	T-FR6-1
FR6	~~Options while Awaiting Player~~	T-FR6-2
FR7	Terminate Game	T-FR7-1
FR8	~~User Selects the Quit Option~~	T-FR8-1
FR9	~~User Selects the Play Against AI Option~~	T-FR9-1
FR10	~~Present Loading Screen~~	T-FR10-1
FR11	Call Start Game	T-FR11-1
FR12	Accept or Reject Game	T-FR12-1
FR13	~~Call Start Game~~	T-FR13-1
FR14	~~No Chat Option~~	T-FR14-1
FR15	~~Accept or Reject Game~~	T-FR15-1
FR16	~~User Chooses "AI"~~	T-FR16-1
FR17	~~User Chooses Online Player~~	T-FR17-1
FR18	Initialize Board	T-FR18-1
FR19	Initialize Timer	T-FR19-1
FR20	Give Control to White Pieces	T-FR20-1
FR21	Await Opponent Move	T-FR21-1
FR22	Move Piece	T-FR22-1
FR23	Check if the Move is Legal	T-FR23-1
FR24	The Move is Legal	T-FR24-1
FR25	The Move is Not Legal	T-FR25-1
FR26	Check "Terminate Game" use case	T-FR26-1
FR27	Resume Game Timer	T-FR27-1
FR28	Verify move	T-FR28-1
FR29	Update Board State	T-FR29-1
FR29	Update Board State	T-FR29-2
FR30	Lock Player Control	T-FR30-1
FR31	Present Start Chat Option	T-FR31-1

FR32	Open Chat Window	T-FR32-1
FR33	Open Chat Window Simultaneously	T-FR33-1
FR34	Present End Chat Option	T-FR34-1
FR35	Present End Chat Option	T-FR35-1
FR36	Close Chat Window	T-FR36-1
FR37	Close Chat Window Simultaneously	T-FR37-1
FR38	Clear Chat Data	T-FR38-1
FR39	Start Timer On Game Start	T-FR39-1
FR40	Initialize Timer Values	T-FR40-1
FR41	Pause and Resume Timers	T-FR41-1
FR41	Pause and Resume Timers	T-FR41-2
FR42	Establish System Checks	T-FR42-1
FR43	Check if a Checkmate has occurred	T-FR43-1
FR44	Check if a Stalemate has occurred	T-FR44-1
FR45	Check if a Draw has occurred	T-FR45-1
FR46	Check if a Resignation has occurred	T-FR46-1
FR47	Check if a Timeout has occurred	T-FR47-1
UH2	Ease of Use	T-UH2-1
~~UH5~~	~~Ease of Use~~	~~T-UH5-1~~
PR1	System Response Time	T-PR1-1
PR2	System Response Time	T-PR2-1
PR3	System Response Time	T-PR3-1
PR8	RAM Usage	T-PR8-1

4 Tests for Proof of Concept

The testing for the proof of concept will be focused around validation of game control logic as well as functionality of other components, such as the chat feature.

4.1 Validation of Player Control

Player Control

1. T-PC1

   Type: Functional, Dynamic

   Initial State: Both player boards have been initialized and the game is in play.

   Input: The current player has the first move, or the opponent has made a move.

   Output: The current player is given complete control to the board, and can choose to make any legal move they want. The opponent may not make a move during this period.

   How test will be performed: The system will make the first move for the player with the white pieces. The test is deemed successful if the player is able to do so.

2. T-PC2

   Type: Functional, Dynamic, Manual

   Initial State: Both player boards have been initialized and the game is in play.

   Input: The opponent player is to make the first move, or the current user has made their move.

   Output: The current player has been restricted all control to the board, and cannot make any legal move on the board.

   How test will be performed: The system will make the opponent start with the white pieces. The test is deemed successful if the current player, with the black pieces, is not able to make a move on the board before the opponent makes a move.

4.2 Functionality of Supporting Components

Functionality of Chat Option

1. T-PC3

   Type: Functional, Dynamic, Manual.

Initial State: A chat has been started between two players.

Input: A valid input string representing a message sent by one of the players.

Output: The message should be visible by both players in the chat window.

How test will be performed: Two team members will start a game with each other and start a chat. One player will send a message in the chat and they will verify that both players can see the message in the chat window. They will also verify that additional messages do not delete previous messages and remain ordered.

5 Comparison to Existing Implementation

A test that compares the system implementation against the existing implementation is as follows:

Board State Comparison

1. T-EI-1

   Type: Functional, Dynamic, Manual.

   Initial State: A game has been started on the system as well as on the existing implementation.

   Input: The same set of valid moves is input into both systems.

   Output: Both systems have the same game state response after each input.

   How test will be performed: Two team members will start a game on the system as well as on the existing implementation. The team members will play the same moves on both systems and ensure the same response occurs on both systems.

6 Unit Testing Plan

The Mocha & Enzyme frameworks will be used to do unit testing for this project. Enzyme will be the utility used to for React that makes it easier to

work with React. Mocha will be used to write the test suite.

6.1 Unit testing of internal functions

To create unit tests for the internal functions of the program, the methods with return values can be tested. This process would involve using the methods by providing appropriate input values. Using the expected outputs of these methods, several unit tests will be developed for the system. Such unit tests will attempt to also generate exceptions, along with valid inputs. No stubs or drivers are necessary for testing the system. Enzyme & Mocha both come with functionality to make imports and tests extremely easy to use. Anything that needs to be imported will already be done by the individual classes. The team goal is to achieve 80-100% statement coverage and greater than 50% path coverage.

6.2 Unit testing of output files

To develop unit tests that will test the accuracy of our output, the system will provide various chess positions and moves, where they will be compared to the actual and the expected output. To complete this task, the chess application will present a chess position, and a method will confirm if the set of moves provided match the current chess position output. For the tests to be deemed successful, all positions should match the corresponding set of moves. The required library for performing unit testing of outputs files is chess.js.