Test Case Documentation

For

Interactive Chess Game

Khanh Tran, Tuan Phan

Table of Contents

1. Introduction

1.1. Purpose of the document

The purpose of this document is to emphasize that the Chess Game design and functionality meets the requirements, which are specified in the previous requirement document. This document consists of the validation and outcomes of possible cases.

1.2. Scope of the document

The scope of this document is to ensure that the three stages of developing the Chess Game are complete and ready to be used. The three stages are starting game, gameplay validation, and ending game. These stages are tested in different testing phases to make sure the application meets all of the requirements.

2. Testing Environment

The test suite was performed on several browsers including Google Chrome, Firefox, Safari, Microsoft Edge and on different operating systems including Windows 10, MacOS, and Ubuntu.

3. Testing Environment Setup

For the testing purpose, our program can be tested on the local machine or the heroku domain with a stable Internet connection. However, if we want to test on our local machine, we would have to open a different browser which acts as a different device since the application uses cookies to validate the user.

4. Test Phases

4.1 Unit Testing

As described in the design documentation, our application is divided into 2 main parts - server and client. The server consists of 3 components: Server Request Handler, Game Manager, Room Manager, and Chess API. The client consists of 3 components: Client Request Handler, Screen Renderer, and Event Handler. Since the Server Request Handler and the Client Request Handler are the central points of contact which connect components in the server and client together, these components will be tested in integration testing phase.

Each component in the server and the client will be tested independently in unit test modules before gluing together in integration and system tests. To decouple the dependencies between different components in the system, the test case will have to create mock objects which act as an alternative for the actual objects from other components.

4.1.1 Room Manager

Room Manager is responsible for managing rooms and users throughout the lifetime of the application. It supports the following operations: create new user, authenticate user, create new room, join a room, quit a room, and get room state. The test cases make sure that different combinations of the scenarios will be covered.

Case Number	**Description**	**Expected Output**

1	Create new user	A random unique user id number
2	Authenticate a valid user id number	True with a success message
3	Authenticate an invalid user id number	False with an error message
4	Valid user creates new room for the first time	True with a success message and a random unique room number
5	Invalid user creates new room	False with an error message
6	Valid user rejoins the room that the user has already joined another room	True with a success message and a random unique room number
7	Valid user joins a full room	False with an error message
8	Valid user creates new room when the user has already another room	False with an error message
9	Valid user joins a valid room when the user has already another room	False with an error message
10	Valid user joins a valid room number	True with a success message
11	Valid user joins an invalid room number	False with an error message
12	Invalid user joins a valid room	False with an error message
13	Valid user who has already joined a room tries to get room number	the room number
14	Valid user who hasn’t joined any room tries to get room number	null
15	Invalid user tries to get room number	null
16	Gets room state from a valid room number	Room state object
17	Get room state from an invalid room number	null
18	Quit room if already joined	True with a success message, remove user from the room
19	Quit room if has not joined (invalid and valid user)	False with an error message

4.1.2 Game Manager

The Game Manager is responsible for managing the game. In the context of this application, it supports operation to make a chess move on the current room. The Game Manager assumes that the user is valid and the room is valid. On a valid move, the Game Manager calls the Chess API to make the actual move and update the state. In the unit test, the Chess API mock objects will be created to support this operation.

Case Number	Description	Expected Output
1	The user makes a valid chess move on their turn	True with a success message
2	The user makes a valid chess move, but not on their turn	False with an error message
3	The user makes tries to move the other player's chess piece (invalid move)	False with an error message
4	The user makes tries to move to an out of bound position (invalid move)	False with an error message
5	The user makes a valid move when the game is paused (the other player hasn't joined)	False with an error message

4.1.3 Chess API

Chess API is responsible for validating the chess move and updating the game state according to the chess rule.

Case Number	Description	Expected Output
1	The pawn makes a forward move if there's no piece blocking the way	True with a success message. Update game state
2	The pawn makes a forward move if there's a piece blocking the way	False with an error message
3	The user's pawn attacks the other player's pieces	True with a success message. Update game state
4	The user's pawn attacks his own pieces	False with an error message
5	The pawn moves two steps forward at the first time	True with a success message. Update game state
6	The pawn moves two steps forward after the first time doing that, or tries to move to the positions which are not valid	False with an error message.
7	The rook moves forward, backward, left, and right at any valid steps	True with a success message. Update game state
8	The user's rook attacks the other player's pieces	True with a success message. Update game state
9	The user's rook attacks his own pieces	False with an error message
10	The bishop moves diagonally at any valid steps	True with a success message. Update game state
11	The user's bishop attacks the other player's pieces	True with a success message. Update game state
12	The user's bishop attacks his own pieces	False with an error message
13	The knight moves two squares vertically and one square horizontally, or two squares	True with a success message. Update game state

	horizontally and one square vertically	
14	The user’s knight attacks the other player’s pieces	True with a success message. Update game state
15	The user’s knight attacks his own pieces	False with an error message
16	The queen should moves forward, backward, left, right, and diagonally at any valid steps	True with a success message. Update game state
17	The user’s queen attacks the other player’s pieces	True with a success message. Update game state
18	The user’s queen attacks his own pieces	False with an error message
19	The king should moves forward, backward, left, right, and diagonally at one step	True with a success message. Update game state
20	The user’s king attacks the other player’s pieces	True with a success message. Update game state
21	The user’s king attacks his own pieces	False with an error message
22	The pieces move to positions which are not valid in possible movements	False with an error message
23	The user makes move if the game is checkmate or stalemate	False with a message that indicates the game is over
24	The user moves after finishing their turn	False with an error message

4.1.4 Screen Renderer

The Screen Renderer is responsible for rendering new changes to the screen. It can properly render the chessboard, other information related to the game, and in-game messages or errors from servers.

Case Number	**Description**	**Expected Output**
1	Render Board	-When the page is loading, a board of size 8*8 with all chess pieces will be rendered on the screen. -If applicable, the last move will be displayed on the screen
2	Render Room	The information about the room (such as room ID, players' names, and scores) will display
3	Display Message	The messages, which are sent from the server, will be loaded to the client

4.1.5 Event Handler

The Event Handler is responsible for handling the events that the users make from the screen (clicking mouse, pressing enter, etc.) The Event Handler interprets these events from the users and makes a corresponding request to the client request handler.

Case Number	**Description**	**Expected Output**
1	Button is clicked	When the button is clicked, the requests are initiated and sent to the server for handling.
2	Create Room Button	The screen will render a new room with a unique ID
3	Join Room Button	When clicking the button, if the room ID is matched with any existing room ID from the server, it will let the second player join the game. If the room ID matches, but there are more than two players in the same room, or the room ID does not match at all, an error message will display.
4	Quit Room Button	When clicking this button, the screen will go to the home page
5	A position is clicked for the first time	When a piece is clicked, the clicked position is saved and all available moves associated with the piece are displayed.
6	Another position is clicked for the second time	When clicking the second time, the first position and the second position are combined to be a move. This move is sent to the request handler to make a move to the server

4.2 Integration Testing

4.2.1 Server

As mentioned above, the Server Request Handler acts as the central unit to connect all components in the server together. In the integration testing phases, other than all tests that we have mentioned in the unit test phase, we will test the Server Request Handler to make sure it correctly integrates Game Manager and Room Manager, and Chess API together.

Case Number	**Description**	**Expected Output**
1	Authenticate User	True for valid user, False for invalid user
2	Create Room	True only for authenticated user, False otherwise
3	Join Room	True only for authenticated user and valid room to join (valid room ID and room is not full), False otherwise
4	Get Room	Return the correct room state associated with the user (user can only join 1 room), return null otherwise (invalid user, user does not join the room)
5	Quit Room	True only for authenticated user who already joined a room, False otherwise
6	Make Chess Move	-True only when user is allowed to move (game is not paused, it is their turn, etc.) and the move is valid according to chess rules, False otherwise -The game state is updated if return true

4.2.2 Client

Similarly, all components in the client are connected by the Client Request Handler. Although there are little dependencies between the client components in the current design, the integration tests are still important to make sure the client works correctly as the whole.

Case Number	**Description**	**Expected Output**
1	Update Room	The page in the browser should be updated correctly with the provided room state including chess board, message from server, and other metadata such as the turn, the room number, etc.
2	Create Room	Upon create a new room function is invoked, the handler send a request to the server to create the room
3	Join Room	Upon join a room function is invoked, the handler sends a request to the server to join a room with the input room id
4	Quit Room	Upon quit the room function is invoked, the handler sends the request to the server to quit the room
5	Make Move	Upon a new move function is invoked, the handler sends a request to the server to make move with the corresponding move input

4.3 System Testing

4.3.1 Functionality Testing

After the integration testing phase with client and server, we now test the application as a whole. The test includes the complete flow of the application from creating or joining a room, making a chess move, quitting a room, etc. The application will be tested in the communication between the client and the server. For different operations that are required for the app, the client should send the proper request to the server, and the server should handle the request accordingly and send the response back to one or both players depending on the operation.

- **User creates or join a room**

  If the operation succeeds, the user should be redirected to the game page with the rendered board and other metadata. If not, the user should be notified with a proper error message.

- **User makes a move**

  If the operation succeeds, the new board with updated state should be rendered on the screen for both players and a message should be displayed saying the move is completed. If not, the user should be notified with a proper error message.

- **User quits a move**

  The user should be redirected back to the launch to create or join a new game. The other player in the game should be notified about this.

4.3.2 Load Testing

The application will be tested under multiple users and rooms simultaneously. We will simulate the load of multi-users by repeatedly making multiple requests to the server with different operations (create room, join room, make move, etc.). The application is expected to properly and correctly respond to the requests with a reasonable response time.

The load testing will also help us locate the bottleneck and software design issues in our application