SE 3XA3: Module Guide
Supreme Chess

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

April 12, 2021

Contents

List of Tables

List of Figures

Important Information

Text that has been struck through, ~~like this~~, was in the previous revision of the document and has been removed.

Text that is colored red, like this, is new to revision 1, and reflects the most recent information.

Text that is colored green, like this, is a comment to the TA's/professor and should not be included as a part of the content in this document.

Table 1: **Revision History**

Date	**Version**	**Notes**
2021-03-13	1.0	Created document, completed up to anticipated and unlikely changes
2021-03-14	1.1	Finished module hierarchy, connections between requirements and design, and started module decomposition
2021-03-16	1.2	Finished traceability matrix and the rest of the document
2021-03-18	1.3	Minor edits and changes
2021-04-11	2.0	Made changes to modules to match the system, as requirements were altered during the course of the development process.

1 Introduction

1.1 Overview

The SupremeChess project is the re-implementation of the open-source software application "Online-Chess-Game", that allows two users to play a game of chess on a server.

1.2 Context

This document is the Module Guide (MG), which is created after the Software Requirements Specification (SRS).

The purpose of the Software Requirements Specification document is to present a description of the software system to be developed, including the functional and non-functional requirements for the project. The following MG has a different purpose, where it is instead providing a modular decomposition of the system, showing the modular structure of the application. The MG also describes how the requirements in the SRS are met with the modular structure that is described.

Along with the MG, it is also necessary to create a Module Interface Specification (MIS) explaining the semantics and syntax of each module. Examples of such semantics and syntax includes the access routines, state variables, inputs/outputs, and exceptions of the modules. This document will further expand on the information provided in the MG.

1.3 Design Principles

Information Hiding and Encapsulation are some of the design principles being used to build a modular structure of our application. The project should also assess the software metric of coupling and cohesion. Ideally, the project has high cohesion, and a low degree of coupling.

Information Hiding is the process of hiding the details of an object or function. This process disassociates the calling code from the internal workings of the object or function being called. This makes it possible to change the hidden portions without having to also change the calling code. Encapsulation is a design principle that makes it easier to enforce information hiding. Encapsulation simply hides the states of an object of a class by preventing direct access to it.

A high cohesion signifies that the methods and data within a module are closely related. Low coupling means that there is a low degree of interdependence between the modules of the system.

1.4 Document Structure

The document structure is organised as follows:

- Section 2 lists Anticipated and Unlikely Changes to the system's implementation. This list is used for the Traceability Matrices later in the document.

- Section 3 presents the Module Hierarchy, listing all the modules and their hierarchy by levels.
- Section 4 describes the Connection Between Requirements and Design, which details how the software requirements are related to the modules.
- Section 5 describes the Module Decomposition, detailing the module secrets, services, and implementations.
- Section 6 provides the Traceability Matrices. The first matrix connects the functional and nonfunctional requirements to the modules. The second matrix connects anticipated changes from Section 2 to the modules.
- Section 7 presents the Uses Hierarchy diagram for the application. The model shows the uses relations between modules.
- Section 8 presents the design schedule for the project. A link to the Gantt chart created is provided in this section.

2 Anticipated and Unlikely Changes

2.1 Anticipated Changes

AC1: How user moves are verified.

AC2: The format for inputting user moves. Currently click piece and drag to destination, likely to add click piece and click destination.

AC3: The implementation of the data structure that stores the game board state.

AC4: User options to change the font and piece models.

AC5: ~~AI settings to select and scale difficulty.~~

AC6: The GUI and interface design.

AC7: User options to play games with different starting timer values.

2.2 Unlikely Changes

UC1 Input/output devices to/from the system (Input: Keyboard or Touchscreen, Output: Screen).

UC2 The user option to play against another user online.

UC3 The interface functionality of the system.

UC4 The rules and values related to the game itself.

UC5 The purpose of the system to allow users to play a game of chess against an opponent.

3 Module Hierarchy

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Module Output Module Board Module Shared Data Module Piece Module Game Module
Software Decision Module	App Module Chat Module Timer Module ~~AI Module~~

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design satisfies the system requirements developed in the SRS. The app module will handle all interactions from the inputs provided by the user and the outputs the program produces. This module will cover all requirements that have a component of the user directly interacting with the system. The ~~AI and~~ chat module covers the requirements of the ~~AI player, and~~ chat functionality. The board and game modules cover the requirements necessitated (such as checking for legal moves, and end game conditions). The timer module will contain timer classes that will provide the timer information for the game. The look and feel requirements are satisfied by the input and output modules as they define the core areas of user interaction. The usability and humanity requirements are again satisfied by the input and output modules which define the user interaction. Performance requirements are to be satisfied by the combination of all modules, but the key modules that impact performance are software decision hiding modules. The operational and environmental requirements, and the maintainability and support requirements are achieved by all modules. This is because all modules are designed to be operational for the environment (the server and client) and maintained for a set time (end of semester). Security and legal requirements are covered by the entirety of the system. This is because the accessibility of the repository and the JavaScript standard in use affects all modules.

5 Module Decomposition

5.1 Hardware Hiding Module

Secrets: The design, data structures, and algorithms used in the implementation of the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. The system can use it to display outputs or to accept inputs.

Implemented By: OS

5.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviour hiding modules.

Services: These modules describe the application's visible behaviour. They facilitate communication between the Hardware Hiding Module and the Software Decision Modules.

Implemented By: –

5.2.1 Input Module

Secrets: Input data.

Services: Allows the user to input game moves into the system.

Implemented By: Node.js libraries.

5.2.2 Output Module

Secrets: Graphics output data.

Services: Takes in data on board state, timers, user selection options, and chat messages, and displays it visually.

Implemented By: Node.js libraries.

5.2.3 Board Module

Secrets: Board data.

Services: Stores and modifies board state information.

Implemented By: Node.js libraries.

5.2.4 Shared Data Module

Secrets: Data.

Services: Stores and shares data needed by other modules and between users.

Implemented By: JavaScript code.

5.2.5 Piece Module

Secrets: Piece data.

Services: Stores piece data and allows for user input to move the piece.

Implemented By: Node.js libraries.

5.2.6 Game Module

Secrets: Game logic and data.

Services: Handles game termination, move handling, and storing previous moves.

Implemented By: Node.js libraries.

5.3 Software Decision Module

Secrets: Data structures.

Services: Provides the data structures to store the information from application.

Implemented By: –

5.3.1 App Module

Secrets: Module output data.

Services: Compiles and renders all incoming module data.

Implemented By: Node.js libraries.

5.3.2 Timer Module

Secrets: Timer data.

Services: Stores and processes game timer data.

Implemented By: Node.js libraries.

5.3.3 Chat Module

Secrets: Chat data, such as messages and room name.

Services: Stores chat messages inputted by players in specific rooms.

Implemented By: Node.js libraries.

5.3.4 ~~AI Module~~

Secrets: ~~AI algorithm data.~~

Services: ~~Outputs and performs possible moves by algorithm.~~

Implemented By: ~~Node.js libraries.~~

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Table 3: Trace Between Requirements and Modules

Req.	**Modules**
R1	M5.3, M5.3.1, M5.3.3
R2	~~M5.2~~, ~~M5.2.1~~, M5.3, M5.3.1, ~~M5.3.4~~
R3	M5.2, M5.2.1, M5.2.2, ~~M5.2.6~~
R4	M5.2, M5.2.1, M5.2.2, M5.2.4, ~~M5.2.6~~, M5.3, M5.3.3
~~R5~~	M5.2, M5.2.6
~~R6~~	~~M5.2, M5.2.6, M5.3, M5.3.1~~
~~R7~~	~~M5.2, M5.2.1, M5.3, M5.3.4~~
~~R8~~	~~M5.2, M5.2.3, M5.2.6, M5.3, M5.3.1~~
~~R9~~	~~M5.2, M5.2.2, M5.2.4~~
~~R10~~	~~M5.2, M5.2.6, M5.3, M5.3.1~~
~~R11~~	~~M5.2, M5.2.6, M5.3, M5.3.1~~
~~R12~~	~~M5.3, M5.3.3~~
~~R13~~	~~M5.2, M5.2.6, M5.3, M5.3.1~~

Req.	**Modules**
~~R14~~	~~M5.2, M5.2.1, M5.3, M5.3.4~~
~~R15~~	~~M5.2, M5.2.1~~
~~R16~~	~~M5.2, M5.2.3, M5.2.6~~
~~R17~~	~~M5.2, M5.2.6, M5.3, M5.3.2~~
R18	M5.2, M5.2.3, M5.2.5, M5.2.6
R19	M5.2, M5.2.4, M5.2.6, M5.3, M5.3.2
R20	M5.2, M5.2.1, M5.2.6, M5.3, M5.3.2
R21	M5.2, M5.2.1, M5.2.4, M5.2.6
R22	M5.2, M5.2.1, M5.2.5, M5.2.6
R23	M5.2, M5.2.5, M5.2.6
R24	M5.2, M5.2.5, M5.2.6
R25	M5.2, M5.2.5, M5.2.6
R26	M5.2, M5.2.6
R27	M5.2, M5.2.6, M5.3, M5.3.2
R28	M5.2, M5.2.4, M5.2.6
R29	M5.2, M5.2.3, M5.2.6
R30	M5.2, M5.2.3, M5.2.4
R31	M5.3, M5.3.3
R32	M5.3, M5.3.3
R33	M5.2, M5.2.1, M5.3, M5.3.3
R34	M5.2, M5.2.1, M5.3, M5.3.3
R35	M5.2, M5.2.1, M5.3, M5.3.3
R36	M5.2, M5.2.1, M5.3, M5.3.3
R37	M5.2, M5.2.1, M5.3, M5.3.3
R38	M5.2, M5.2.1, M5.3, M5.3.3
R39	M5.2, M5.2.6, M5.3, M5.3.2
R40	M5.2, M5.2.6, M5.3, M5.3.2
R41	M5.2, M5.2.6, M5.3, M5.3.2
R42	M5.2, M5.2.2, M5.2.3, M5.2.6
R43	M5.2, M5.2.2, M5.2.3, M5.2.6
R44	M5.2, M5.2.2, M5.2.3, M5.2.6
R45	M5.2, M5.2.2, M5.2.3, M5.2.6
R46	M5.2, M5.2.2, M5.2.3, M5.2.6
R47	M5.2, M5.2.2, M5.2.3, M5.2.6
UH2	M5.2, M5.2.6
UH8	M5.2, M5.2.6
PR1	M5.2, M5.2.1, M5.2.2
PR2	M5.2, M5.2.1, M5.2.2
PR3	M5.2, M5.2.1, M5.2.2
PR8	M5.2, M5.2.1, M5.2.2

Table 4: Trace Between Anticipated Changes and Modules

Req.	**Modules**
AC1	M5.2, M5.2.6
AC2	M5.2, M5.2.1, M5.2.5
AC3	M5.2, M5.2.3, M5.2.5
AC4	M5.2, M5.2.2
~~AC5~~	~~M5.3, M5.3.4~~
AC6	M5.2, M5.2.2
AC7	M5.3, M5.3.2

7 Use Hierarchy Between Modules

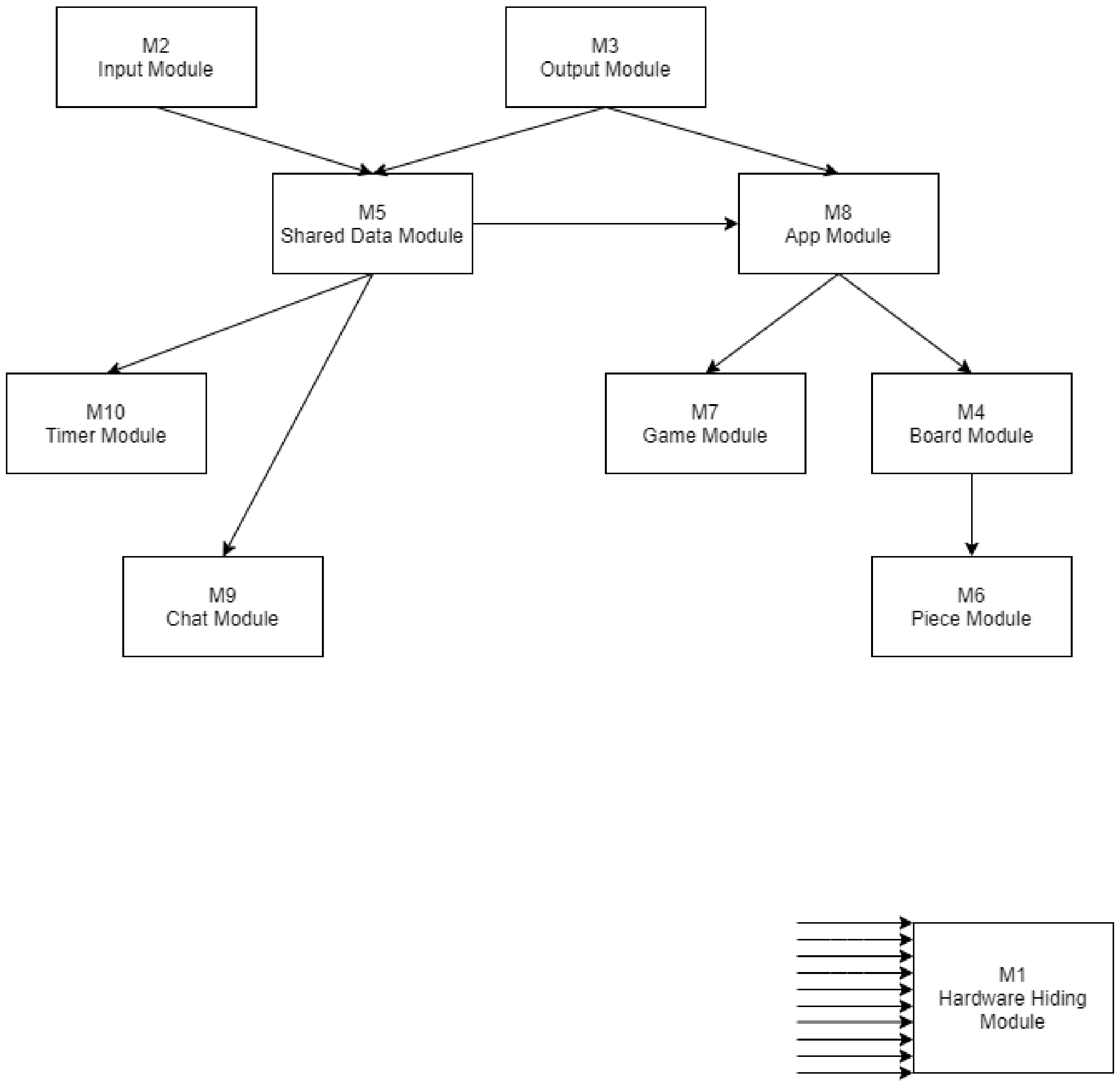

Figure 1: Use Hierarchy Among Modules

8 Design Schedule

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

Many marks were lost in this section of revision 0 for a lack of Git issues being used to track modules. As we cannot go back to create these we hope that we will not lose the same marks again for revision 1. Rather than using Git issues, work was split into Git branches by functionality and assigned individually.