# SE 3XA3: Module Guide
# Tetris Tussle


| Nicholas Lobo | Matthew Paulin | David Carrie |
|---|---|---|
| lobon3 | paulinm | carriedd |
| 400179304 | 400187147 | 000661652 |


March 19, 2021

# Contents



# List of Tables

# List of Figures



Table 1: **Revision History**

| Date | Version | Notes |
|---|---|---|
| 16/03/2021 | 1.0 | Finished sections 1 and 7 |
| 17/03/2021 | 1.1 | Finished sections 2, 3, and 4 |
| 18/03/2021 | 2.0 | Completed MG |

# 1 Introduction

## 1.1 Overview

Tetris took the world by storm in the 1980's becoming the first hit game ever created by selling over 202 million copies. The aim of the development of Tetris Tussle is to take a bare bones version of this game and update it to better suit the modern standards of gaming through updated visuals, extra features, and player versus player (PvP) interaction

## 1.2 Context

This document is the Module Guide (MG) and is based on the previously created Software Requirements Specification (SRS). The SRS contains information about the functional and non-functional requirements that will guide the creation of a correct system. The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document include new project members, maintainers of the software, and the project's designers. New project members can use this document to easily and quickly understand the overall structure of the software and well as find the modules relevant to them. Maintainers can use the hierarchical structure of the MG to guide their modifications to the system and update this document when changes are made. Lastly, designers can use the completed MG to check for consistency among modules, feasibility of the decomposition, and the design's flexibility. After this document, a Module Interface Specification (MIS) will be created to provide detailed information about each individual module.

## 1.3 Design Principles

The design principle that is being used to guide the design of the software system is information hiding. Information hiding is the principle that each module hides a design decision from the rest of the system. Through decomposition, based on this principle, the system can be designed for change. This is invaluable because during development, modifications are very frequent. The principle of information hiding is followed by separating likely changes as the secrets of their own modules. Additionally, each data structure is used in only one module and any other module that requires information stored in a module's data structure will obtain it by calling access programs belonging to that module. This principle also enables high cohesion, low coupling, and no cycles among modules. High Cohesion refers to elements of a module being closely related. This vastly increases the software's maintainability because related functionality will be localized to an area of the software. Low coupling states that modules not be reliant on other modules. Module independence ensures that changes can be made to one module without altering another. No cycles means that no two modules will use each other, thus making the code easier to understand and maintain.

### 1.4 Document Structure

The rest of the document is organized as follows. Section 2 lists the anticipated and unlikely changes of the software requirements. Section 3 summarizes the module decomposition that was constructed according to the likely changes. Section 4 specifies the connections between the software requirements and the modules. Section 5 gives a detailed description of the modules. Section 6 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 7 describes the uses relation between modules

## 2 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

### 2.1 Anticipated Changes

**AC1:** The specific hardware on which the software is running.

**AC2:** The format of the initial input data.

**AC3:** The calculation of the score

**AC4:** The design and format of the menu

**AC5:** The board customization

**AC6:** The amount of players in one multiplayer lobby

**AC7:** The Tetromino customization

**AC8:** The multiplayer gameplay

### 2.2 Unlikely Changes

**UC1:** Input/Output devices (Input: Keyboard, Output: Screen/Server).

**UC2:** There will always be a source of input data external to the software.

**UC3:** The purpose and objective of Tetris

**UC4:** The usage of the p5.js library for the implementation of the software

**UC5:** The usage of a server for multiplayer functionality

**UC6:** The usage of a socket interaction for input output functionality

**UC7:** The design pattern and architecture for the software

**AC8:** The use of a web application to interact with the software

# 3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

**M1:** Hardware Hiding Module: User Inputs Module

**M2:** Behaviour Hiding Module: Main View Module

**M3:** Behaviour Hiding Module: Leaderboard Module

**M4:** Behaviour Hiding Module: Menu Module

**M5:** Behaviour Hiding Module: Singleplayer View Module

**M6:** Behaviour Hiding Module: Multiplayer View Module

**M7:** Behaviour Hiding Module: Singleplayer Module

**M8:** Behaviour Hiding Module: Multiplayer Module

**M9:** Behaviour Hiding Module: Player Module

**M10:** Behaviour Hiding Module: Tetromino Module

**M11:** Behaviour Hiding Module: Board Module

**M12:** Software Decision Hiding Module: Server Module

| Level 1 | Level 2 |
|---|---|
| Hardware-Hiding Module | N/A |
| Behaviour-Hiding Module | User Inputs Module<br>Main View Module<br>Leaderboard Module<br>Menu Module<br>Singleplayer View Module<br>Multiplayer View Module<br>Singleplayer Module<br>Multiplayer Module<br>Player Module<br>Tetromino Module<br>Board Module |
| Software Decision Module | Server Module |

Table 2: Module Hierarchy

# 4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 3.

# 5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

## 5.1 Hardware Hiding Modules (M1)

**Secrets:** The data structure and algorithm used to implement 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. So, the system can use it to display outputs or to accept inputs.

**Implemented By:** OS, JavaScript

## 5.2 Behaviour-Hiding Module

**Secrets:** The contents of the required behaviours.

**Services:** Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

**Implemented By:** HTML, CSS, JavaScript (p5.js, Node.js)

### 5.2.1 User Inputs Module (M1)

**Secrets:** Inputs

**Services:** Converts the input data into the data structure used by the input parameters module.

**Implemented By:** JavaScript

### 5.2.2 Main View Module (M2)

**Secrets:** Graphics

**Services:** Draws the main screen.

**Implemented By:** JavaScript (p5.js)

### 5.2.3 Leaderboard Module(M3)

**Secrets:** Score

**Services:** Displays point totals from the top scoring games.

**Implemented By:** JavaScript (p5.js)

#### 5.2.4 Menu Module (M4)

**Secrets:** Graphics

**Services:** Enables application navigation.

**Implemented By:** JavaScript (p5.js)

#### 5.2.5 Singleplayer View (M5)

**Secrets:** Graphics

**Services:** Draws the single player view representing the current state of a single player game of Tetris.

**Implemented By:** JavaScript (p5.js)

#### 5.2.6 Multiplayer View Module (M6)

**Secrets:** Graphics

**Services:** Draws the multiplayer view including a representation of a game of player versus player Tetris.

**Implemented By:** JavaScript (p5.js)

#### 5.2.7 Singleplayer Module (M7)

**Secrets:** Data

**Services:** Uses input data to store and manipulate the state of a singleplayer game of Tetris.

**Implemented By:** JavaScript (Node.js)

#### 5.2.8 Multiplayer Module(M8)

**Secrets:** Data

**Services:** Uses input data to store and manipulate the state of a multiplayer game of Tetris.

**Implemented By:** JavaScript (Node.js)

#### 5.2.9 Player Module (M9)

**Secrets:** Data

**Services:** Stores and shares information of the player state.

**Implemented By:** JavaScript (Node.js)

#### 5.2.10 Tetromino Module (M10)

**Secrets:** Data

**Services:** Stores and shares information of the Tetromino state.

**Implemented By:** JavaScript (Node.js)

#### 5.2.11 Board Module (M11)

**Secrets:** Data

**Services:** Stores and shares information of the board state.

**Implemented By:** JavaScript (Node.js)

### 5.3 Software Decision Module

**Secrets:** The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

**Services:** Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

**Implemented By:** JavaScript (SocketIO, Node.js)

#### 5.3.1 Server Module (M12)

**Secrets:** Inter-module communication

**Services:** Facilitates communication between the clients and the server.

**Implemented By:** JavaScript (SocketIO, Node.js)

## 6 Traceability Matrix

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

| Req. | Modules |
|---|---|
| BE1-FR1 | M1, M9, M10 |
| BE1-FR2 | M9, M10 |
| BE1-FR3 | M9, M10 |
| BE2-FR1 | M1,M9 |
| BE2-FR2 | M1,M9 |
| BE2-FR3 | M1,M9 |
| BE2-FR4 | M1,M9 |
| BE2-FR5 | M11 |
| BE2-FR6 | M9, M10 |
| BE2-FR7 | M11 |
| BE2-FR8 | M11 |
| BE2-FR9 | M7, M8 |
| BE2-FR10 | M7, M8 |
| BE3-FR1 | M4, M12 |
| BE3-FR2 | M4, M12 |
| BE3-FR3 | M4, M12 |
| BE3-FR4 | M2 |
| BE3-FR5 | M10 |
| BE3-FR6 | M10 |
| BE3-FR7 | M7, M8 |
| BE3-FR8 | M5, M6 |
| BE3-FR9 | M7, M8, M11 |
| BE4-FR1 | M3 |
| BE4-FR2 | M2 |
| BE4-FR3 | M3 |
| BE4-FR4 | M3 |
| BE5-FR1 | M8 |
| BE5-FR2 | M8 |
| BE5-FR3 | M8 |
| BE6-FR1 | M8 |
| BE6-FR2 | M4,M8 |
| BE6-FR3 | M8 |

Table 3: Trace Between Requirements and Modules

| AC | Modules |
|---|---|
| AC1 | M1 |
| AC2 | M1 |
| AC3 | M12 |
| AC4 | M2, M4 |
| AC5 | M11 |
| AC6 | M6, M8 |
| AC7 | M10 |
| AC8 | M12 |

Table 4: Trace Between Anticipated Changes and Modules

# 7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

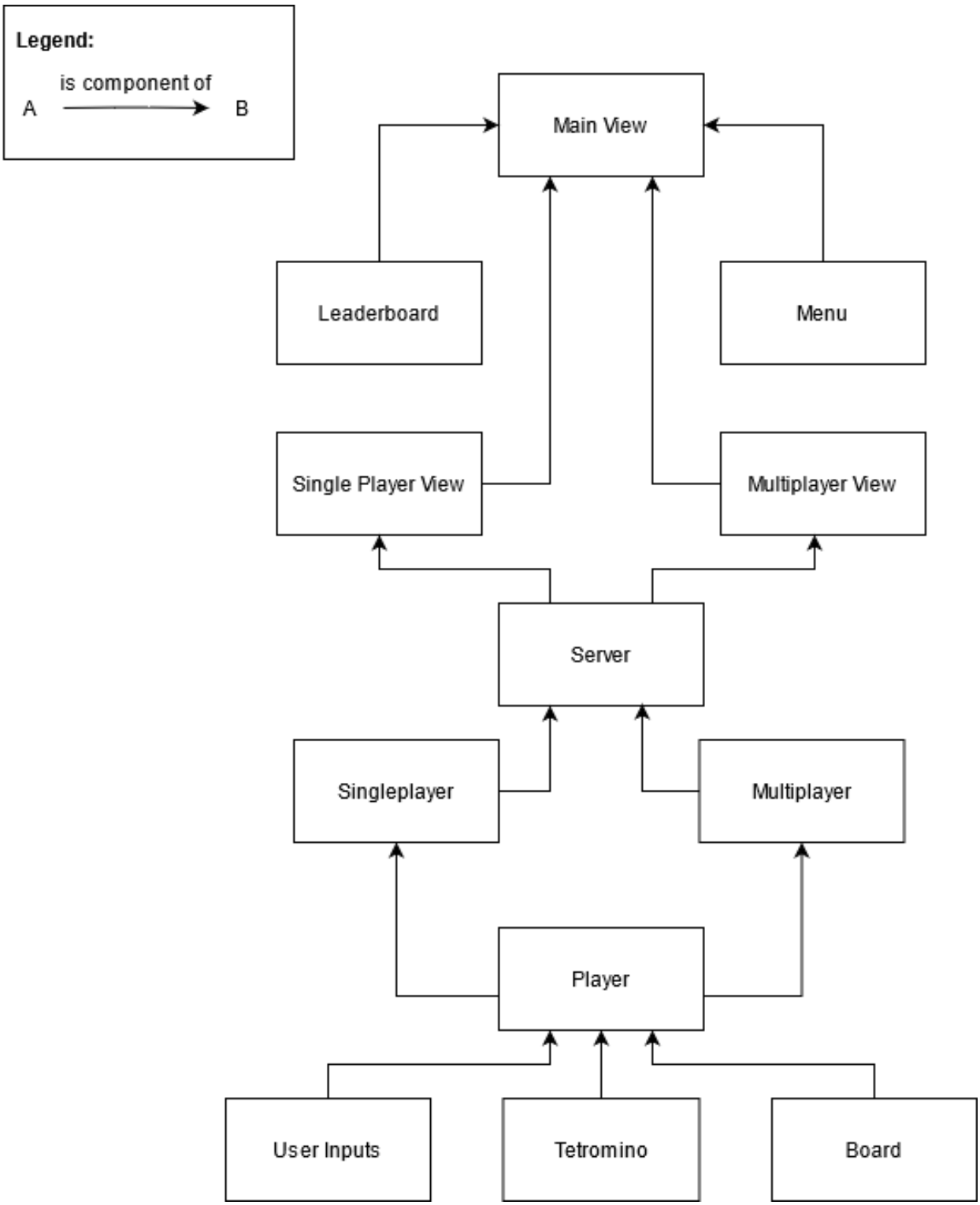


Figure 1: Use hierarchy among modules

# References

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.