# SFWRENG 3XA3: Module Guide
# Rummy For Dummies

Lab 2 Group 7, Rummy For Dummies
Joy Xiao, xiaoz18
Benson Hall, hallb8
Smita Singh, sings59

March 18, 2021

# Contents

# List of Tables

## List of Figures

Table 1: **Revision History**

| **Date** | **Version** | **Notes** |
|---|---|---|
| March 16, 2021 | 1.0 | Started on the MG |
| March 18, 2021 | 1.1 | Finished the MG |

# 1 Introduction

## 1.1 Overview

Rummy For Dummies is a Java re-implementation of the card game Gin Rummy. The intent is to recreate a single-player card game normally played with several people. This is to provide users the experience of playing a multi-player card game.

Rummy For Dummies will be an improved version of the original open-source project played on the command line. This is done through being more user-friendly compared to the open-source implementation and more accessible to the general public, even if they do not have a technical background. This will be achieved in both the documentation and the game itself.

## 1.2 Context

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (**?**). We advocate a decomposition based on the principle of information hiding (**?**). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by **?**, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (**?**). 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 are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.

- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

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 use 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

Anticipated changes are the sources of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

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

**AC2:** The presentation of the player's hand on the console.

**AC3:** The presentation of the discard pile on the console.

**AC4:** The addition of exception handling to all the classes.

### 2.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

**UC1:** The computer opponent playing algorithm.

**UC2:** The meld algorithm to find melds in a hand.

**UC3:** The data structure for the stock pile.

**UC4:** The data structure for the discard pile.

**UC5:** The data structure for the hand.

**UC6:** The data structure for the card.

**UC7:** There will always be a source of input data external to the software.

**UC8:** The data structure for the player.

**UC9:** The algorithms for the game's basic functions.

## 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

**M2:** Game Operations Module

**M3:** Input Module

**M4:** Computer Module

**M5:** Stock Pile Data Structure Module

**M6:** Discard Pile Data Structure Module

**M7:** Card Data Structure Module

**M8:** Hand Data Structure Module

**M9:** Melds Module

**M10:** Player Data Structure Module

| **Level 1** | **Level 2** |
|---|---|
| Hardware-Hiding Module | |
| Behaviour-Hiding Module | Game Operations Module<br>Input Module<br>Melds Module |
| Software Decision Module | Computer Module<br>Stock Pile Data Structure Module<br>Discard Pile Data Structure Module<br>Card Data Structure Module<br>Hand Data Structure Module<br>Player Data Structure 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 ?. 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, Java

## 5.2 Behaviour-Hiding Module

### 5.2.1 Game Operations Module (M2)

**Secrets:** The algorithms representing the game's functions

**Services:** Uses the algorithms to perform the game's basic functions

**Implemented By:** GameOps.java

### 5.2.2 Input Module (M3)

**Secrets:** The methods for taking user data and formatting it to be usable for the data structures and algorithms of the game.

**Services:** Accepts and converts input data into data that will be used by the game operations module.

**Implemented By:** UserInputOps.java

### 5.2.3 Melds Module (M9)

**Secrets:** The algorithm used to determine optimal melds

**Services:** Determines optimal melds from a hand of cards, and returns a 2D ArrayList of sequence and group melds.

**Implemented By:** Meld.java, SortByRank.java, SortBySR.java

## 5.3 Software Decision Module

### 5.3.1 Computer Module (M4)

**Secrets:** The algorithm of the computer module.

**Services:** Performs the computer opponent move with a specific algorithm used.

**Implemented By:** Computer.java

### 5.3.2 Stock Pile Data Structure Module (M5)

**Secrets:** The data structure of the stock pile for the game.

**Services:** Stores the information about the stock pile such as the order of cards, as well as functionalities such as adding and removing cards.

**Implemented By:** StockPile.java

### 5.3.3 Discard Pile Data Structure Module (M6)

**Secrets:** The data structure of the discard pile for the game.

**Services:** Stores the information about the discard pile such as the order of cards, as well as functionalities such as adding and removing cards. It also stores the information for displaying the discard pile on the console.

**Implemented By:** DiscardPile.java

### 5.3.4 Card Data Structure (M7)

**Secrets:** The data structure for the cards in the game

**Services:** Stores information of card objects as well as provides other functionalities

**Implemented by:** Card.java

### 5.3.5 Hand Data Structure Module (M8)

**Secrets:** The data structure of a hand in the game.

**Services:** Stores the information about a hand of cards as well as functionalities such as removing a card from the hand and adding a card to the hand.

**Implemented By:** Hand.java

### 5.3.6 Player Data Structure Module (M10)

**Secrets:** The data structure of a player in the game

**Services:** Stores information about a player, as well as interfacing methods for performing operations on its internal state objects

**Implemented By:** Player.java

## 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** |
|---|---|
| FR1 | M2, M3 |
| FR2 | M2, M7, M6, M8, M10 |
| FR3 | M2, M3 |
| FR4 | M2, M5, M7, M8, M10 |
| FR5 | M2 |
| FR6 | M3 |
| FR7 | M2, M6, M7, M8, M10 |
| FR8 | M2 |
| FR9 | M3 |
| FR10 | M2, M7, M8, M9, M10 |
| FR11 | M2, M10 |
| FR12 | M2, M3, M7, M8, M10 |
| FR13 | M2, M6, M7, M8, M10 |
| FR14 | M2, M6, M7 |
| FR15 | M2, M10, M4, M7, M5, M6 |
| FR16 | M2, M10 |
| FR17 | M2 |
| FR18 | M2, M10 |
| FR19 | M2, M10 |
| FR20 | M2, M3 |
| FR21 | M2, M3 |
| LF1 | M2 |
| UH1 | M2 |
| UH2 | M2 |
| P1 | M2 |
| P2 | M2 |
| OE1 | M2 |
| MS1 | M2, M3, M4, M5, M6,M7, M8, M9, M10 |
| MS2 | M2, M3, M4, M5, M6, M7, M8, M9, M10 |
| C1 | M2 |
| C2 | M2 |
| L1 | M2, M3, M4, M5, M6, M7, M8, M9, M10 |

Table 3: Trace Between Requirements and Modules

| **AC** | **Modules** |
|---|---|
| AC1 | M1 |
| AC2 | M8 |
| AC3 | M6 |
| AC4 | M2, M3, M4, M5, M6, M7, M8, M9, M10 |

Table 4: Trace Between Anticipated Changes and Modules

## 7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** 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.

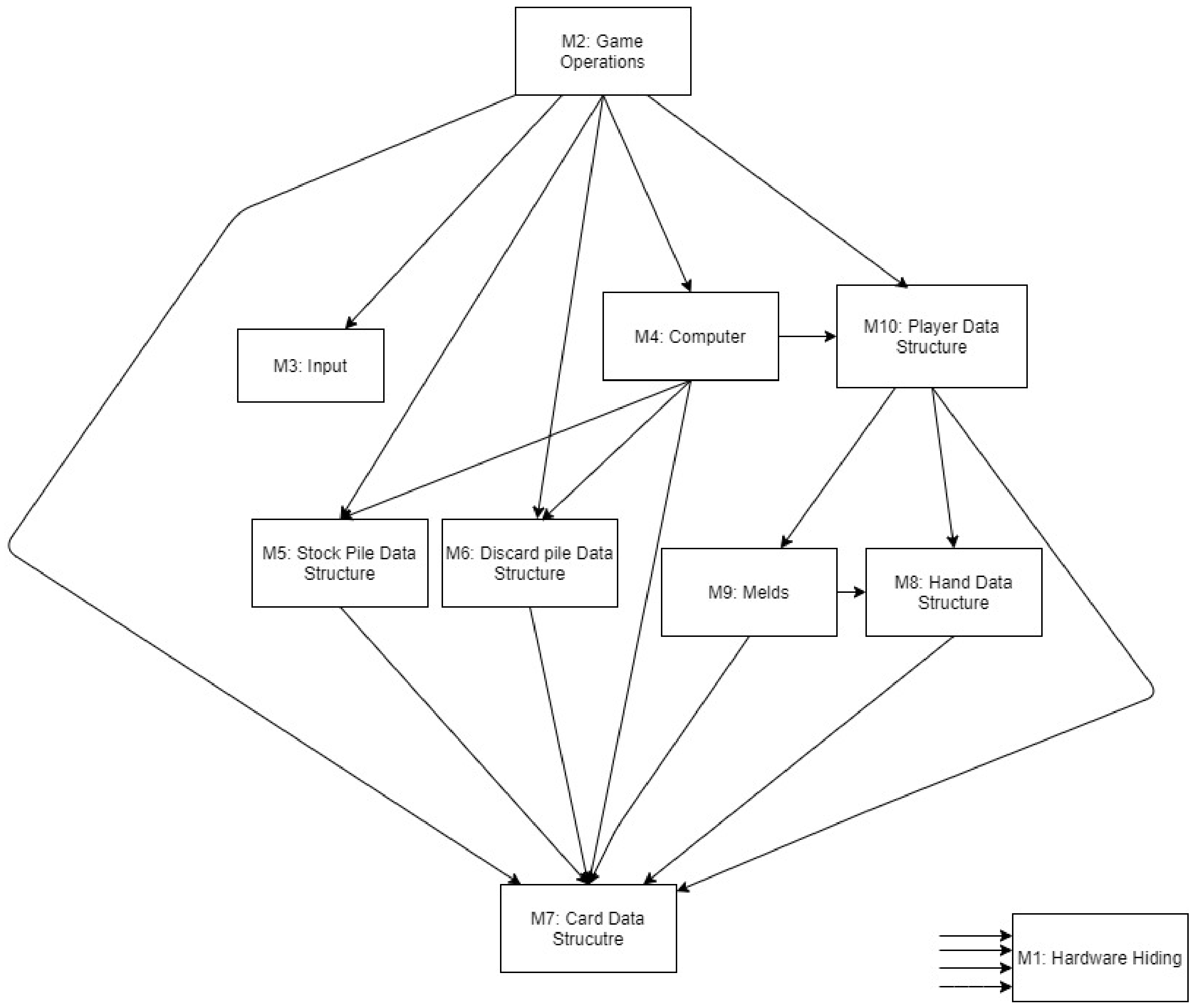

Figure 1: Use hierarchy among modules

## 8 Project Schedule

The Gantt Chart for the project schedule is located under the *ProjectSchedule* directory, named **3XA3-project.gan**.