SE 3XA3: Software Requirements Specification
BigTwo

Team 06, Team Name: Aplus^3
Senni Tan, tans28
Manyi Cheng, chengm33
Jiaxin Tang, tangj63

April 12, 2021

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Mar 13, 2021	0.0	Initial draft
Apr 10, 2021	1.0	Revision 1.0

1 Introduction

1.1 Overview

The BigTwo project is a re-implementation of an open-source LAN server game that was originally found on GitHub. The original project was implemented in Java, and our re-implementation will be in JavaScript and CSS. Our project, the BigTwo game, is a web-based card game that allows the user to play alone in a web browser with mouse clicking on the computer screen to interact with the game.

1.2 Purpose

This Module Guide document will provide the information about the structure in our module design, for the stakeholders including professor Bokhari, TAs in course SE3XA3, project developers to have a document to trace along wtih the project development, and also to understand the structure of the module design for this project.

1.3 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 (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). 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.

1.4 Design Principles

Our design follows the rules layed out by Parnas et al. (1984), 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.

1.5 Design Requirements

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). 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.
- Professor and TAs in course SE3XA3: This document can be a reference for the professor and TAs in course SE3XA3 to trace our process in project development, and to understand the overall structure of our module design.

1.6 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 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 source 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 graphical user interface used to display the various elements of the game, and retrieve user input.

AC2: The format of the input data.

AC3: The format of the output data.

AC4: The actual implementation of the components.

AC5: The implementation of the classes and objects for Cards, PlayField, Players and Scene.

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: Input/Output devices (Input: Mouse clicks, Output: Screen). The game will take the input from the user mouse clicking interaction with the game, and output the result to the screen.

UC2: The purpose of the game to allow the user to play the BigTwo card game.

UC3: The amount of user player: out game only allow single-player mode so only one user player is allowed in a game.

UC4: The game rule of BigTwo: there are many versions on BigTwo game, we will take the Cantonese version of rules in our project.

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: Hardware-Hiding Module

M2: Behaviour-Hiding Module: App Module

M3: Behaviour-Hiding Module: Card Module

M4: Behaviour-Hiding Module: Player Module

M5: Behaviour-Hiding Module: PlayerBot Module

M6: Software Decision Module: Rules Module

M7: Behaviour-Hiding Module: Game Module

M8: Behaviour-Hiding Module: GameplayField Module

Level 1	**Level 2**
Hardware-Hiding Module	Hardware-Hiding Module
Behaviour-Hiding Module	App Module Card Module Player Module PlayerBot Module Game Module GameplayField Module
Software Decision Module	Rules 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 4. The App module will render the game and other objects in a browser. The Game module will connect all methods together and will be the only way the program is executed. The functional requirements will be satisfied throughout all modules. The Player Module will fulfill the Usability requirement as it is responsible for interaction between user and server. The Rules module is responsible for functional requirements related to BigTwo game rules.
The non-functional requirements will be satisfied throughout all modules. Look and feel will requirements will be implemented in the GameplayField and Game module. The game will be easy to access because the game will be deployed using the server. Performance requirements are key to the program and will be fulfilled throughout all modules, ensuring that each part of the game fulfills corresponding requirements. Cultural, Political, Health, and safety requirements will be implemented by not using any external images that may offend the user, and by reducing the use of blue color and promoting warm colors.

5 Module Decomposition

The following modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984), which are broken down into *Secret*, *Service*, and *Implemented By*. *Secret*is a brief description of the design decision hidden by the module. *Service* specifies what the module will do with details. *Implemented By* indicates how the module is implemented.

5.1 Hardware Hiding Modules

Secrets: The implementation of the virtual machine.

Services: Serves as the interface between the software and the hardware, which allows the system to communicate with the software through input and output devices.

Implemented By: Operating System

5.2 Behaviour-Hiding Module

Secrets: Required behaviours

Services: Describes the virtual behaviour of the application according to software requirements specified in the software requirements specification(SRS). This module serves as an interpreter between the Hardware Hiding Module and the Software Decision Module to ensure the application behaves as expected.

Implemented By: N/A

5.2.1 App Module

Secrets: App

Services: Stores the App of the game.

Implemented By: React.js

5.2.2 Card Module

Secrets: Card

Services: Generates and updates the cards of the game.

Implemented By: React.js

5.2.3 Player Module

Secrets: Player Model

Services: Allows the user to select cards to play in the game.

Implemented By: React.js

5.2.4 PlayerBot Module

Secrets: PlayerBot Model

Services: Serves as computer player to play with the user according to user inputs.

Implemented By: JavaScript Libraries

5.2.5 Game Module

Secrets: Game

Services: Stores and updates the state of the game according to user inputs.

Implemented By: React.js

5.2.6 GameplayField Module

Secrets: GameplayField

Services: Arranges players.

Implemented By: React.js

5.3 Software Decision Module

Secrets: Data Structures

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

Implemented By: N/A

5.4 Be Module

5.4.1 Rules Module

Secrets: Rules

Services: Stores the rules of the game.

Implemented By: JavaScript Libraries

6 Traceability Matrix

This section shows three traceability matrices: between the modules and functional requirements, between the modules and non-functional requirements, and between the modules and the anticipated changes.

Req.	**Modules**
FR1	M3, M4, M6
FR2	M2
FR3	M2, M7
FR4	M2, M3, M7
FR5	M2
FR6	M2, M4, M7
FR7	M2, M4, M7
FR8	M8
FR9	M7
FR10	M2
FR11	M2
FR12	M7
FR13	M6
FR14	M7, M8
FR15	M7, M8
FR16	M7, M8
FR17	M7, M8

Table 3: Trace Between Functional Requirements and Modules

Req.	Modules
NF-L1	M2
NF-L2	M2
NF-L3	M2
NF-L4	M2
NF-L5	M2, M3
NF-L6	M2
NF-U1	M4, M7
NF-U2	M2, M4, M7
NF-U3	M2
NF-U4	M2
NF-A1	M1
NF-P1	M1
NF-P2	M6, M8
NF-P3	M1
NF-O1	M1
NF-C1	M2
NF-Le1	M2
NF-C2	M2, M7
NF-H1	M2

Table 4: Trace Between Non-functional Requirements and Modules

AC	Modules
AC1	M1
AC2	M7
AC3	M1
AC4	M1
AC5	M1

Table 5: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

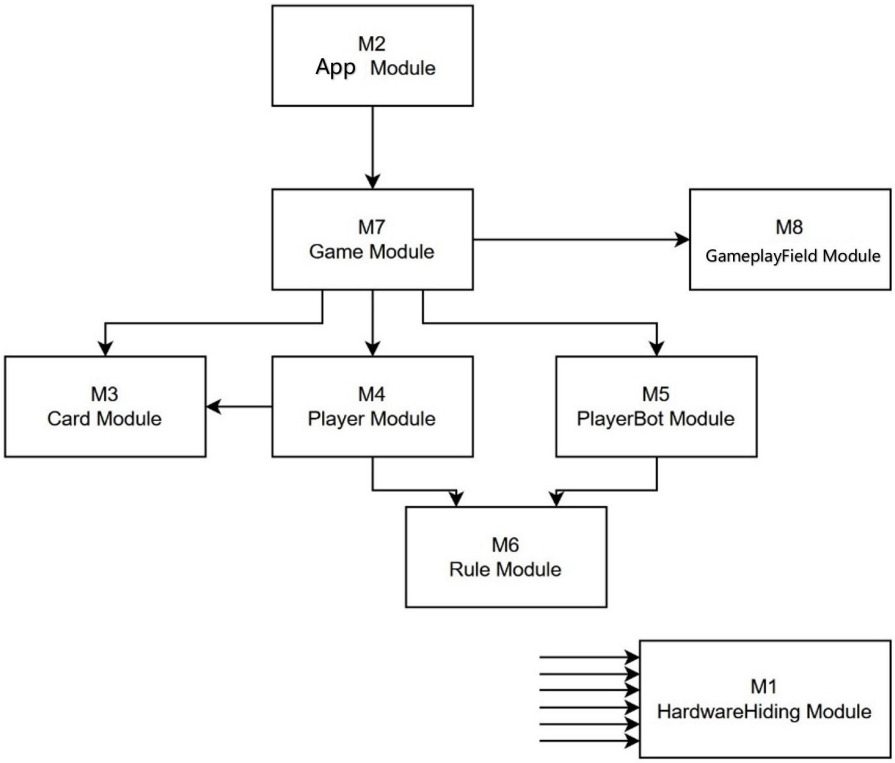

Figure 1: Use hierarchy among modules

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

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.