SE 3XA3: Module Guide CraftMaster

Group Number: 307
Group Name: 3 Craftsmen
Members:
Hongqing Cao 400053625
Sida Wang 400072157
Weidong Yang 400065354

Contents

List of Tables

List of Figures

Date	**Editor(s)**	**Change**
Mar 9	Sida	General Content added
Mar 13	Weidong	Edited for Rev0 submission
Mar 13	Sida	Minor Modification
Apr 6	Sida	Completed for Rev1
Apr 6	Hongqing	Final Check

Table 1: **Revision History**

1 Introduction

The format of this document follows the template provided by Dr.Bokhari and Thien Trandinh.

1.1 Project Overview

CraftMaster is a re-implementation of **Michael Fogleman's Simple Minecraft-inspired Demo** (referred to as **original project** in the following content), which is developed in Python and Pyglet. The CraftMaster design team has initiated and completed multiple requirements(specified in the **Software Requirement Specifications(SRS)**) to add new features, including new block types, day and night mode shift, game saving, and game menu frames, to the original project. During the software design process, the team has also applied software architecture design patterns and software design principles to increase the quality of the software design. The specification of those design methodologies will be described in this document.

In terms of the software product characteristics, CraftMaster is a 3D Sandbox Game that allows players to control the character and build the game world based on their imagination. We believe that CraftMaster will be beneficial to teenagers and children in the way that it inspires them to unleash their creativity.

1.2 Context of Module Guide

The **SRS** document shows the features, functionalities and desired properties that the system should have. The **Module Guide(MG)** is generated based on the **SRS**, which further evaluates how requirements are achieved and also specifies the modular structure decomposition of the software system. It will be distributed to help potential readers easily identify the decomposed parts of the software. The potential readers are as follows:

- **New Project Members:** The **MG** acts as a guideline for the new project members to easily and quickly understand the modular structure of the system and its decomposition specifications. With this document, those new members can search for relevant modules more efficiently.
- **Designers:** The **MG** is used to help software system designers check for the consistency among modules, the flexibility of the design, and the feasibility of the modular decomposition.
- **Developers:** The hierarchical structure specified in the **MG** will give the developers a better understanding of the system decomposition and use relationships between different modules.

- **Maintainers:** The hierarchical structure of the system improves the maintainers' understanding of either the system as a whole or individual modular parts when they need to make changes to the system and the documents.

The **Module Interface Specifications(MIS)** is another section of the Design Specifications Documents other than the **MG**. The **MIS** shows the semantics and syntax of exported functions for each module in details. The **MG** should be an entry document for the design specifications and the readers should read the **MG** first to get an overview of the system, then browse the **MIS** for further references once they identify which module(s) they are searching for.

1.3 Design Principles

- **High Cohesion and Low Coupling**: This principle has been applied to the project in the way that the modules are designed to be strongly related and the dependency has been minimized.

- **Open-Closed Principle**: The modules are designed to be closed to modification and the system is designed to easily extended. For instance, the **screen** module is implemented as a template and all other scenes(such as **gameScene**, **mainScene** and **settingScene**) inherit it. There might be new scenes to be implemented in the future to easily extend the system.

- **Liskov Substitution Principle**: The inheritance relationships among modules follows the Liskov Substitution Principle.

- **Dependency Inversion Principle**: The inheritance design pattern is used to support the abstraction of the system. With the abstraction, the dependency inversion principle can be followed.

- **Interface Segregation Principle**: The inheritance design pattern minimizes the interfaces of subclass modules, which follows the Interface Segregation Principle.

- **Law of Demeter**: The abstraction of the system supports the principle of the Law of Demeter. The communications only depend on the interfaces as limited knowledge.

- **Information Hiding**: The information hiding principle is supported by abstraction and private methods.

1.4 Content 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.

1.5 Naming Conventions and Terminology

The naming conventions and terminology section will aid readers from different backgrounds to clearly understand the content of this document. The naming conventions and terminologies used in this document are listed below:

- **OS:** Operating System.
- **GUI:** Graphical User Interface, which allows the user to interact and visualize the program by graphics instead of text.
- **Sandbox Game:** A type of game that allows player to create, modify, and destroy the environment.
- **Pyglet:** The Python library for the design of graphical user interface and multi-media.
- **Player/Gamer:** The person who controls the PC to play the game.
- **Character:** The fictional character who is controllable by the player/gamer(not visible to the player).
- **3D Game:** A game in three dimensions.

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: More static block types such as **Dirt** and **Sand**, shown in Figure 1.

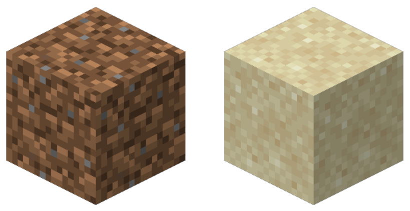
Figure 1: Dirt and Sand Block

AC3: The dynamic block types including **Lava** and **Water**, shown in Figure 2, which provide explicit interactions with the game character.

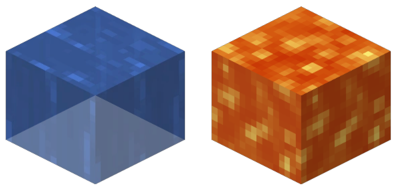
Figure 2: Lava and Water Block

AC4: Objects that can interact with the game character, such as **Chicken** and **Cow**, shown in Figure 3.

Figure 3: Chicken and Cow Object

AC5: Minimizing coupling: There could be some possibilities for minimizing the coupling of the system. The current system is considered to be over decomposed and some modules are not necessary to be an individual one. Therefore, to reduce the coupling of the system, some modules can be merged into others. For example, the button module is now used by different scene modules, and those scene modules inherit the screen module. As a consequence, the button module can be merged into the screen module as part of its implementation so that the those scene modules only need to inherit the new screen module with buttons implemented within and the button module will not be needed.

2.2 Unlikely Changes

This section specifies the changes that are planned out but difficult to implement due to the limitation of the utilities of Pyglet.

UC1: Input Devices such as game console controller.

UC2: Saving more game scenes, as the maximum game saving is two for now.

UC3: Inventory System.

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 Hardware-hiding section hides the hardware machine or the “virtual machine” provided by the Pyglet. The Software decision section hides internal data structures and algorithms. The Behaviour hiding section hides input formats, screen formats, and text messages. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: block

M2: button

M3: creature

M4: devTools

M5: game

M6: loadSource

M7: main

M8: gameScene

M9: mainScene

M10: settingScene

M11: player

M12: processQueue

M13: screen

M14: shape

M15: world

Level 1	**Level 2**
Hardware-Hiding Module	**Pyglet**
Behaviour-Hiding Module	game gameScene mainScene settingScene player world loadSource
Software Decision Module	creature block screen processQueue shape devTools button

*Note that the main module only acts as a trigger to the program and does not support information hiding and therefore it is mentioned in either this table or Section 5.

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 consistency between requirements and modules is listed in Table 3. The game module will focus on the requirements related to macro control of the GUI, menu directions, and player's input/output collection. The gameScene module will focus on the requirements related to controlling the game scene perspective captured by the player. The mainScene and settingScene modules will focus on the requirements related to menu interactions. The player module will focus on the requirements related to the character's operations and status. The world module will focus on the requirements related to loading and controlling the game world and block operations. The loadSource module will focus on the requirements related to media loading such as textures and background music. The creature module is a template module and the player module inherits it. The block module defines game blocks used by the world module. The screen module is a template module and all the scene modules inherit it. The processQueue module defines a queue to for game processes. The shape module specifies 2D and 3D coordinates locating methods. The devTools module is a development tool used by the developer to generate standard texture images. The button module will focus on the requirements related to menu buttons.

5 Module Decomposition

5.1 Hardware Hiding Modules (M16)

Secrets: The algorithm used to implement the virtual hardware that controls the mouse, keyboard, monitor and audio player.

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** and **Pyglet**

5.2 Behaviour-Hiding Module

5.2.1 game Module (M5)

Secrets: The interactive game frame(GUI).

Services: Provide the game frame(GUI) that controls the scenes switches and game world initialization.

Implemented By: game.py

5.2.2 gameScene Module (M8)

Secrets: The interactive content of the game scene that will be rendered on the GUI.

Services: Render the interactive content of the game Scene on the GUI.

Implemented By: gameScene.py

5.2.3 mainScene Module (M9)

Secrets: The main menu scene that will be rendered on the GUI.

Services: Render the main menu on the GUI.

Implemented By: mainScene.py

5.2.4 settingScene Module (M10)

Secrets: The setting menu scene that will be rendered on the GUI.

Services: Render the setting menu on the GUI.

Implemented By: settingScene.py

5.2.5 player Module (M11)

Secrets: The game character.

Services: Provide a game character with its operations in the game world.

Implemented By: player.py

5.2.6 world Module (M15)

Secrets: The game world.

Services: Provide a game world and its operations.

Implemented By: world.py

5.2.7 loadSource Module (M6)

Secrets: The algorithm of loading media.

Services: Provide algorithms to load texture images, background music and on-click effect sound.

Implemented By: loadSource.py

5.3 Software Decision Module

5.3.1 screen Module (M13)

Secrets: The interactive scene template.

Services: Provide a basic template for other scenes.

Implemented By: screen.py

5.3.2 creature Module (M3)

Secrets: The moving object in the game world.

Services: Provide a basic template for the moving objects in the game world.

Implemented By: creature.py

5.3.3 block Module (M1)

Secrets: The game blocks.

Services: Provide a basic template for the game blocks in the game world.

Implemented By: block.py

5.3.4 processQueue Module (M12)

Secrets: The processQueue data structure.

Services: Provide a data structure to store processes.

Implemented By: processQueue.py

5.3.5 shape Module (M14)

Secrets: The shape drawing algorithm.

Services: Provide an algorithm to draw 2-dimensional and 3-dimensional shapes.

Implemented By: shape.py

5.3.6 devTools Module (M4)

Secrets: The image process and formating algorithm.

Services: Provide an algorithm to process and produce texture images in a certain format.

Implemented By: devTools.py

5.3.7 button Module (M1)

Secrets: Buttons.

Services: Provide two types of buttons(switches by OnOffButton and Toggles by Button).

Implemented By: button.py

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	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15, M16
FR2	M16, M5, M7, M9, M2, M14, M13
FR2.1	M5, M9, M2, M16, M14
FR2.1.1	M5, M9, M2, M14, M16
FR2.1.2	M10, M2, M14, M15, M16
FR2.2	M16, M9, M5, M2, M14
FR3	M5, M11, M3
FR4	M16, M8, M13, M5
FR4.1	M16, M8, M13, M5
FR5	M5, M8, M11, M3
FR5.1	M5, M8, M11, M3
FR5.2	M5, M8, M11, M3
FR5.3	M5, M8, M11, M3
FR5.4	M5, M8, M11, M3
FR6	M5, M8, M11, M3
FR7	M5, M8, M11, M3
FR8	M5, M8, M11, M3
FR8.1	M5, M8, M11, M3
FR8.2	M5, M8, M11, M3
FR8.3	M5, M8, M11, M3
FR9	M16, M5, M8, M15, M11, M13, M14
FR10	M5, M8, M15, M11, M12
FR11	M5, M8, M15, M12, M11, M6
FR11.1	M5, M8, M6
FR11.2	M5, M8, M6
FR11.3	M5, M8, M6
FR12	M5, M16
FR12.1	M5, M16, M10
FR12.2	M5, M15, M14, M2, M10
FR12.3	M5, M10, M2, M14
FR13	M5, M10, M2, M14
FR14	M7, M6, M16, M5
FR15	M5, M6, M8, M16, M15, M11, M12
FR16	M5, M8, M15, M11
FR17	M5, M8, M6, M1
FR18	M5, M8, M10, M9
NFR1	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15

NFR2	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15
NFR3	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15
NFR4	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15, M16
NFR5	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15, M16
NFR6	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15, M16
NFR7	M16
NFR8	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15
NFR9	M16
NFR10	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15, M16
NFR11	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15
NFR12	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15
NFR13	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15
NFR14	M1, M2, M3, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC1	M16
AC2	M4, M6, M8, M15
AC3	M4, M6, M8, M15
AC4	M3, M4, M6, M8, M15

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Arrow represent USES relationship between modules.The **Hardware Hiding Module** refers to **OS** and **Pyglet** modules, which belong to external program. The figure contains two software components, **Software Game** and **Texture Image Processor**. The **Software Game** component demonstrates the use hierarchy between modules in the implementation of the game program itself. The **Texture Image Processor** is a software tool implemented by the team to produce texture images in a certain format that will be implicitly used by the block module, which is shown with a dashed arrow line.

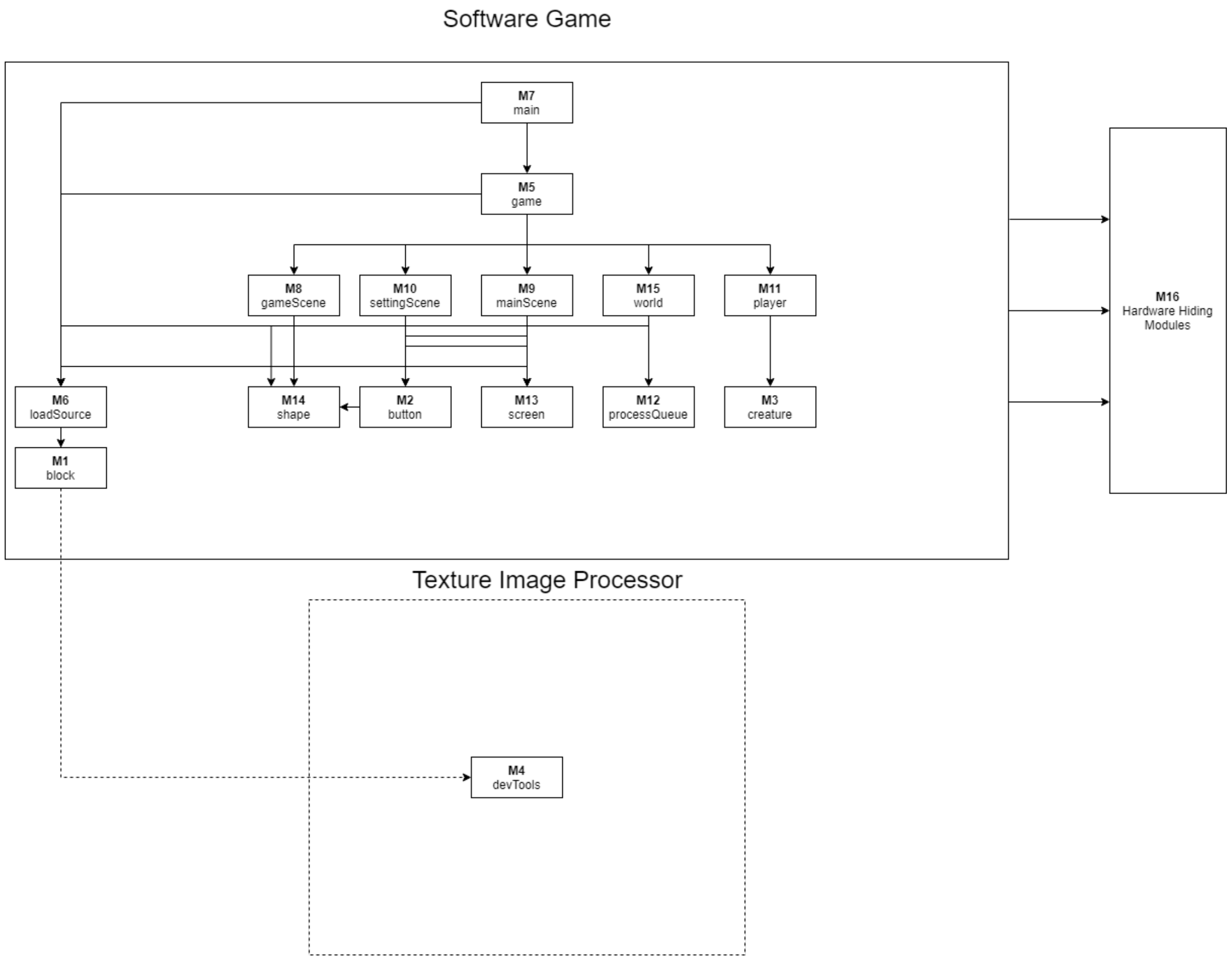

Figure 4: Use hierarchy among modules