SE 3XA3: Module Guild

Snake Game Remake

Team #, 302, Team Name: 404
Student: Shunbo Cui cuis13
Student: Xiangxin Kong kongx9
Student: Shuo Zhang zhans18

March 13, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
March 13, 2020	Shuo Zhang, Xiangxin Kong, Shunbo Cui	Create the MG
April 6, 2020	Shuo Zhang, Xiangxin Kong, Shunbo Cui	Edit Part 5,6,7

1 Introduction

1.1 Overview

Snake-Game-remake is a Java game developed for providing an enjoyable and challenging gaming experience for the user. It is a game inspired by Snake-Java-2D-game which focuses on the growth of a snake character by consuming the food items generated in the game map. The snake keeps moving and the player can adjust the direction to the left or right. The player has to navigate the snake to avoid collision with the boarder of the map and other barriers, as well as its own body. The team's main purpose of the project is to modularize the structure so that it will be easier to add more functions and features to the system. The Module Interface Specification is created, which explains the breakdown of the project for each module.

1.2 Scope

The scope of this project is to implement the 2D Java game board and additional functions. The team is going to add more functions such as leader board, restart option and more in-game items. The user interface is also going to be reconstructed. All the additional function will be implemented in individual Java modules with high cohesion.

1.3 Document structure

The rest of the document is organized as follows.

- Section2 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 our team considering to make. 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 format of the input data.

AC2: Default player settings for input.

AC3: The hardware on which the software is ran.

AC4: The in-game textures of snake, map and items.

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: Keyboard, Output: Speaker, Screen). Keyboard is essential to enable the function of multiplayer in the same game. The speaker is used to play effect sound in the game and screen for displaying.

UC2: The method of defining a map. A lot of other components in the system requires to be changed if this part of design is modified.

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: Main module

M2: Game panel

M3: Snake

M4: Food and Item

M5: Map

M6: Sound effect

M7: Main

M8: Color

M9: Input Manager

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Map Sound Effect Game Panel Main Color InputManager
Software Decision Module	Snake Food and Item

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 input and output hardware such as keyboard and the screen.

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 Java Virtual Machine

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.

5.2.1 Game panel module (M2)

Secrets: The game panel generation and interaction.

Services: Construct the basic panel as the game board.

Implemented By: Java Libraries

5.2.2 Game map construction module (M5)

Secrets: The game map constructions.

Services: Construct the game map from the input data from the map files.

Implemented By: Java Libraries

5.2.3 Sound effect control module (M6)

Secrets: The sound effect controls.

Services: Details the sound effect used in the game.

Implemented By: Java Libraries

5.2.4 Main module (M7)

Secrets: The choices from Users.

Services: Construct panel for user to make game choices and setting before game start.

Implemented By: Java Libraries

5.2.5 Color module (M8)

Secrets: The color controls.

Services: Details the color used in the game.

Implemented By: Java Libraries

5.2.6 Input manager module (M9)

Secrets: The Input key controls.

Services: Details the keyboard keys used in the game.

Implemented By: Java Libraries

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: –

5.3.1 Snake module (M3)

Secrets: Algorithms implemented for generation and interaction of snakes in the game.

Services: Provides the data structure and algorithms for generation and interaction of snakes in the game.

Implemented By: Java Libraries

5.3.2 Food and Item module (M4)

Secrets: Algorithms implemented for generation and interaction of all in-game items.

Services: Provides the data structure and algorithms for generation and interaction of in-game items.

Implemented By: Java Libraries

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, M4, M5, M6, M7, M8, M9
FR2	M2, M7
FR3	M2, M7
FR4	M2, M5, M7
FR5	M2, M5, M8
FR6	M2, M3, M5
FR7	M2, M3
FR8	M2, M4, M5
FR9	M2
FR10	M2
FR11	M2
FR12	M2, M3, M4, M6
FR13	M2, M5, M4
FR14	M2, M3
FR15	M2, M4, M5
FR16	M2, M4, M5, M3, M6
FR17	M2, M3, M5, M6
FR18	M2
FR19	M2, M8, M7
FR20	M2, M9

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M2
AC2	M2
AC3	M1
AC4	M2

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.

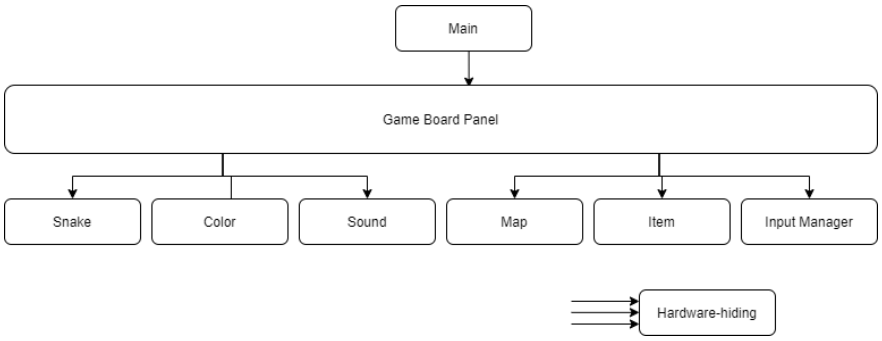

Figure 1: Use hierarchy among modules