SE 3XA3: Module Guide
Namcap

Team 2, VPB Game Studio
Vatsal Shukla (shuklv2)
Prajvin Jalan (jalanp)
Baltej Toor (toorbs)

November 10, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
2016-11-09	1.0	Addition of Introduction Section
2016-11-09	1.1	Completion of Module Hierarchy Section
2016-11-10	1.2	Completion of Introduction Section
2016-11-10	1.3	Completion of Uses Hierarchy Section

1 Introduction

1.1 Overview

Namcap is a re-implementation of an open-source project for the classic arcade game, Pac-man.

1.2 Context

This is the Module Guide (MG) document, which is made after the Software Requirements Specification (SRS). The SRS document specifies the functional and non-functional requirements for the project, where the MG provides a modular decomposition of the system and shows the modular structure.

After MG is made, the Module Interface Specification (MIS) is created. The MIS is responsible for explaining the semantics and syntax of exported functions for each module.

1.3 Design Principles

The main design principle used for this project is the Model-View-Controller architectural pattern. Therefore, the game will be separated in terms of user controls, game mechanism and the user interface. Using this model will allow testing for the game mechanism apart from the input and output mechanisms.

1.4 Document Structure

The document is organised as such:

- Section 2: Anticipated and Unlikely Changes to the system's implementation.
- Section 3: Module Hierarchy, lists all modules and their hierarchy by secrets.
- Section 4: Explains Connection Between Requirements and Design.
- Section 5: Module Decomposition. Details for each module.
- Section 6: Tracability Matrix

- Section 7: Uses Hierarchy 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 specific hardware on which the software is running.

AC2: The format of the initial input data.

...

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: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: There will always be a source of input data external to 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. The Ghost Module has been implemented such that it only contains only the algorithm for movement for enemies in the game, so it is a generic module that can be used for the movement of any AI character that may be implemented in Namcap.

M1: Hardware-Hiding Module

M2: Behaviour-Hiding Module

M3: Software Decision Module

M4: Main Menu Module

M5: Character Module

M6: Player Module

M7: Board Module

M8: Score Module

M9: Main Menu Viw Module

M10: Board View Module

M11: Button Controller Module

M12: Key Controller Module

M13: Ghost Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Main Menu Character Player Board Score Main Menu View Board View Button Controller Key Controller
Software Decision Module	Ghost

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

5.2 Behaviour-Hiding Module (M2)

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: –

5.2.1 Input Format Module (M??)

Secrets: The format and structure of the input data.

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

Implemented By: [Your Program Name Here]

5.2.2 Etc.

5.3 Software Decision Module(M3)

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 Etc.

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**
R1	M1, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

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.

It should be noted that the design of the modules follows the model-view-controller architectural pattern so the Uses Hierarchy between modules has been colour-coded to differentiate the components. Red entities are controller modules (dealing with user input), yellow entities are view modules (dealing with display), and green entities are model modules (dealing with game mechanisms). To ensure clarity, since all modules are connected to the Hardware-Hiding module - which directly interacts with system hardware for input and output - it has been shown seperately in the figure.

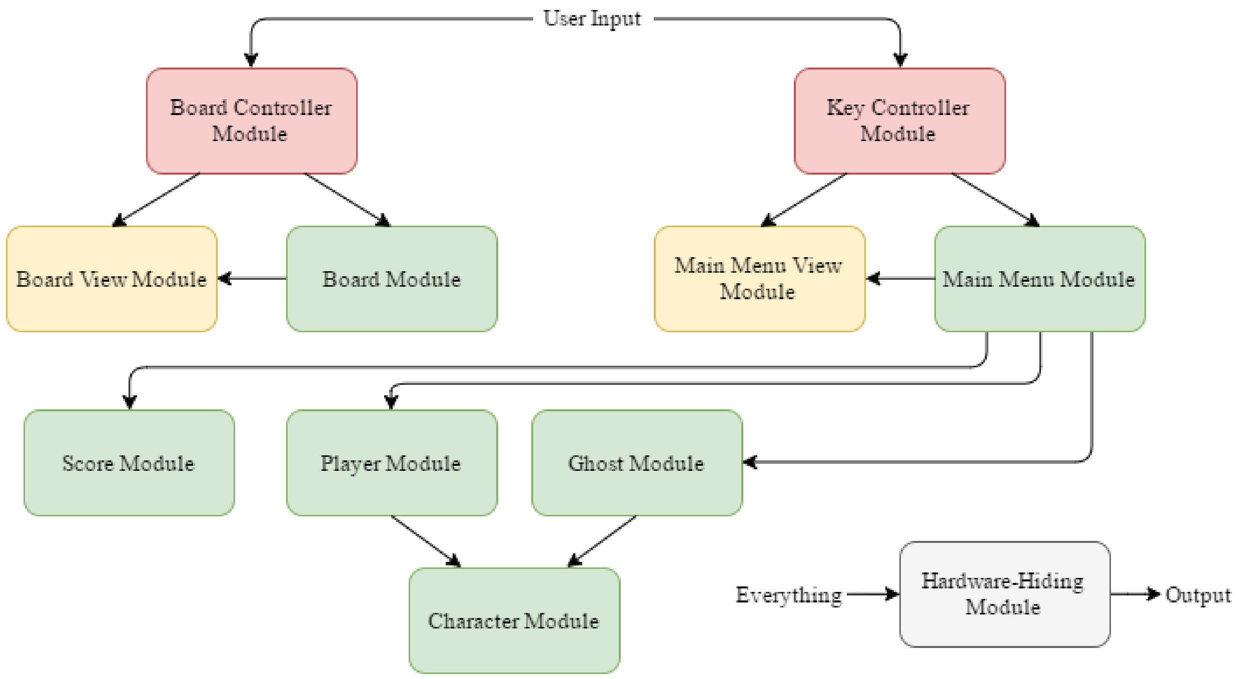

Figure 1: Use hierarchy among modules

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.