SE 3XA3: Module Guide
Namcap

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

November 29, 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
2016-11-10	1.4	Completion of Module Decomposition Section
2016-11-13	1.5	Addition of Anticipated and Unlikely Changes
2016-11-13	1.6	Addition of Initial Traceability Matrices
2016-11-13	1.7	Addition of Connection Between Requirements and Design
2016-11-13	1.8	Addition of remaining Traceability Matrix mapping, initial revision
2016-11-29	1.9	Modifications for Ghost to Enemy refactoring

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, following the Software Requirements Specification (SRS). The SRS document specifies the functional and non-functional requirements for the project, where as the MG provides a modular decomposition of the system and shows the modular structure.

After MG is documented, the Module Interface Specification (MIS) is generated. The MIS is responsible for explaining the semantics and syntax of exported functions for each module. Semantics, sure but syntax no. It should be more or less a blueprint of your modules. The framework is there, implementation is not. - CM

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 mechanisms, and the user interface. Using this model will allow testing for the game mechanics 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: Traceability 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

The following are anticipated changes pertaining to the design of Namcap. The development team will approach the project with an adapted design for change methodology that will require only the one module that contains the decision elements for the change to be altered.

AC1: The graphical sprites for the Player and Enemy entities in the game.

AC2: The representation of the score and player lives on the GUI.

AC3: Tracking of the high score value to be instance-independent (high score will persist from program run to program run).

2.2 Unlikely Changes

The following are changes that are unlikely to occur as many parts of the design will potentially need to be modified based on the decisions made. The referred-to decisions primarily pertain to system architecture and core functionalities.

UC1: Input/Output devices (Input: File (high score) and Keyboard (player control), Output: FIle (high score), Screen (displays GUI)).

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 (with the exclusion of M1 - M3 representing generalized groups of modules), are the modules that will actually be implemented. The Enemy Module has been implemented such that it only contains the algorithm for enemy movement 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 View Module

M10: Board View Module

M11: Button Controller Module

M12: Key Controller Module

M13: Enemy Module

M1 - M3 should not be modules, but groups of modules - CM

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	Enemy

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements detailed in the project SRS. The connection between requirements and modules is listed in Table 3. The Traceability Matrix maps the project modules to the corresponding requirements and anticipated changes.

For these purposes only requirements that specify the code and/or implementation of the redevelopment are included. Non-functional requirements NF8 - NF17 are withheld from the traceability matrix as those requirements pertain primarily to environmental, maintainability, cultural, and legal aspects of the development. More specifically, the non-functional requirements that are included in the requirements traceability matrix have been approximated in terms of the modules that represent the respective requirement. Usabiliy (typo! - CM) and performance envelope the modules that implement the major mechanical and input functionalities of the application. For NF6, the matrix uses the non-leaf modules to generalize that the implementation as a whole implements the requirement.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984) fix reference - CM. 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.

5.1 Hardware Hiding Modules (M1)

Secrets: The data structures and algorithms 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. Furthermore, 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) document. 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 Main Menu Module (M4)

Secrets: The instantiation of a new game.

Services: Starts a new game when user clicks on Start Game button.

Implemented By: Namcap

5.2.2 Character Module (M5)

Secrets: Attributes of a character.

Services: Acts as a super-class to the Player and Enemy modules. Provides the character's location on the map/grid.

Implemented By: Namcap

5.2.3 Player Module (M6)

Secrets: The movement mechanics of the Player.

Services: Allows the user to navigate the player across the map on a valid path.

Implemented By: Namcap

5.2.4 Board Module (M7)

Secrets: The barrier detection mechanism.

Services: Provides a layout of all the barriers on the map and a layout of all the dots on the map.

Implemented By: Namcap

5.2.5 Score Module (M8)

Secrets: The scoring mechanism.

Services: Provides the user's score.

Implemented By: Namcap

5.2.6 Main Menu View Module (M9)

Secrets: The display of the main menu.

Services: Displays the main menu to the user.

Implemented By: Namcap

5.2.7 Board View Module (M10)

Secrets: The main game display.

Services: Displays the game's board/map to the user.

Implemented By: Namcap

5.2.8 Button Controller Module (M11)

Secrets: The actions performed by any buttons.

Services: Performs the action.

Implemented By: Namcap

5.2.9 Key Controller Module (M12)

Secrets: The actions performed by key presses.

Services: Provides a direction, based on what key is pressed, to the player's movement module.

Implemented By: Namcap

5.3 Software Decision Module(M3)

I really like that you define what these are, just in the wrong place - CM

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 Enemy Module (M13)

Secrets: The enemy's movement mechanism.

Services: Implements an (I am being picky, but fix these typos) - CM simple AI that moves the enemy on a valid path.

Implemented By: Namcap

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
F1	M4, M9, M11
F2	M5, M6, M7, M10, M12
F3	M5, M6, M10
F4	M6, M10
F5	M5, M13
F6	M5, M10, M13
F7	M6, M7, M10
F8	M6, M7, M10
F9	M6, M10
F10	M6, M7, M10
F11	M6, M12
F12	M12
F13	M6, M8
F14	M6, M7, M8
F15	M6, M10
NF1	M9, M10
NF2	M9, M10
NF3	M5, M6, M7, M10, M12
NF4	M9, M10
NF5	M5, M6, M7, M8, M10, M12
NF6	M2, M3
NF7	M5, M6, M7, M10, M12

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M10
AC2	M6, M7, M8, M10
AC3	M7, M8, M10

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, thus 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 spelling - CM in the figure.
I like the design overall. If the board does in fact, hold / contain the player and enemies, it does work. An exception to this is if a player is abstracted from the level and can function on its own. Also, the model never interfaces with the view. The controller *controls* how the model is displayed and thus should handle that - CM

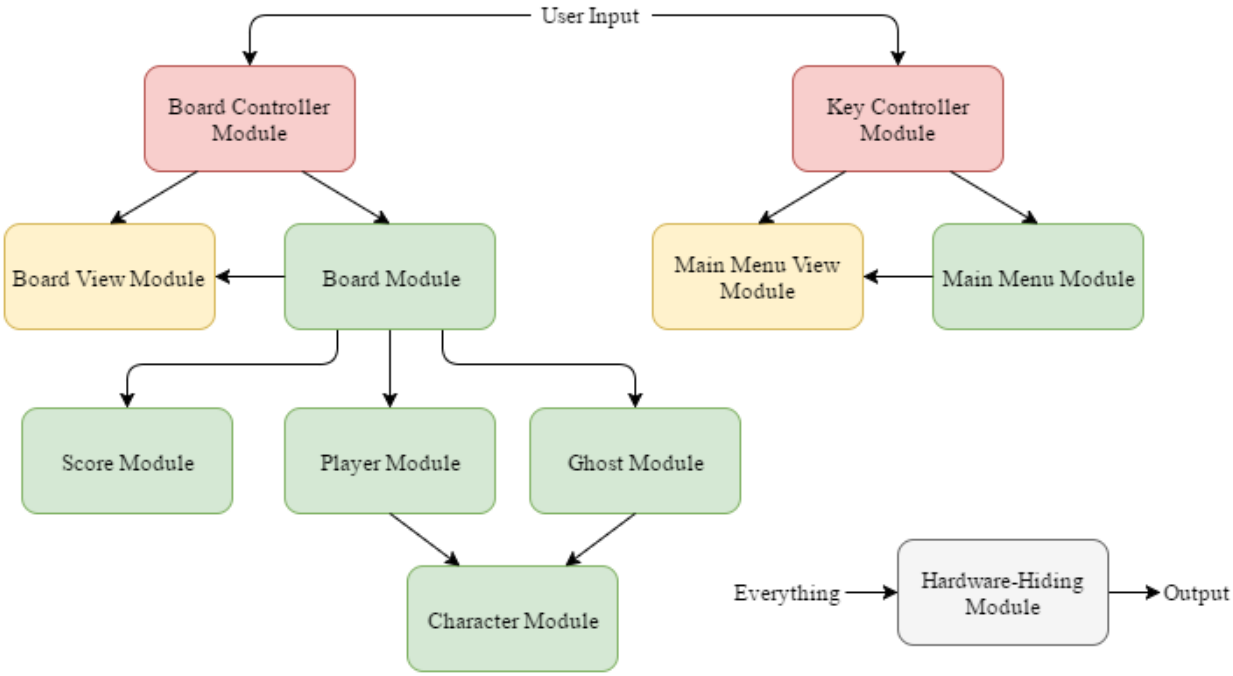

Figure 1: Use hierarchy among modules

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.