SE 3XA3: Module Guide
DNA Says

Team #10, Team Name: DNA
Kareem Abdel Mesih - abdelk2
John-Paul Dakran - dakranj
Shady Nessim - nessimss

December 6, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
3/11/2016	1.0	Addition of section 2
7/11/2016	1.1	Addition of section 3
8/11/2016	1.2	Addition of section 1
11/11/2016	1.3	Addition of section 4
12/11/2016	1.4	Addition of section 7
13/11/2016	1.5	Addition of section 5, 6
13/11/2016	2.0	First Revision
13/11/2016	2.1	Second Revision

1 Introduction

1.1 Overview

This project is a redevelopment of the famous game Simon Says, with a slight modification that makes DNA Says unique while keeping the integrity of the game consistent with the original version. The game consists of three distinct modes - Kareem Says, JP Says and Shady Says.

1.2 Context

This document consists of the Module Guide (MG) for the project DNA Says. This is the second portion of the design documentation along with the Module Interface Specification (MIS) - which explains the semantics of the code in natural language.

The Module Guide is a decomposition of the software system into modules. A module is an independent self-contained unit that makes up a complex software system. Decomposing a problem into modules is an extremely important aspect of software design as it promotes the principle of information hiding. Each module is completed concurrently by a programmer and houses a secret of the modules functionality.

The Module Guide (MG) reveals how the software system will carry out the functionality that is described in the Software Requirements Specification (SRS) document. The potential readers of this document are listed below:

- Designers/Developers: This document is extremely important for the designers of the software system. It provides a means for the designers to easily identify different parts of the software and relate the implementation to the requirements.
- New Project Members: This document allows new project members to easily identify the components of the software system. It is a simple way to understand and locate information that relates to specific parts of the software.
- 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.

1.3 Design Principles

The first major design principle utilized by the developers of DNA Says is modularity. This design principle was achieved by dividing the software into independent self-contained units. After the concurrent development of the individual modules, they are integrated together to fulfill the functional and non-functional requirements in the SRS.

The second major design principle used in this software system is abstraction. After division of the software system into modules, each module was viewed at an abstract level. In other words, the design team neglected the implementation of the module and focused on the most essential information which is relevant to that specific module.

The third major design principle utilized by the developers of DNA Says is information hiding. This design principle was employed during the development of the individual modules. The theory behind this principle is that the modules were designed in such a way that the information and secret of a module cannot be exposed to other modules that do not need to know that information.

1.4 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.

1.5 Naming Conventions & Terminology

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
MG	Module Guide
SRS	Software Requirements Specification
MVC	Model View Controller
OS	Operating System

Table 3: **Table of Definitions**

Term	**Definition**
Mode	Different subsections of the game
Gantt Chart	Chart outlining the timeline of the project
Python	A programming language
Pygame	Cross-platform set of Python modules designed for writing video games

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 routine that the user follows to start the program. There will be an attempt to create a standalone application rather than require the user to install Python and Pygame.

AC2: The main menu.

AC3: The flow of the program. There will be a button within each mode to go back to the main menu.

AC4: The colours of labels and buttons of the main menu along with all the modes.

AC5: The size of the text in each mode.

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 decisions 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: The MVC structure will be transformed to accommodate for the separation of concerns, along with information hiding.

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 4. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module*

M2: Main Module

M3: Menu Module

M4: JP Module

M5: Kareem Module

M6: Shady Module

M7: Setup Module

M8: Update Module

M9: ShowInst Module

M10: ShowScore Module

M11: ShowGoBack Module

M12: DrawKeys Module

M13: FlashKeyAnimation Module

M14: ChangeBackgroundAnimation Module

M15: GameOverAnimation Module

M16: CheckForQuit Module

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	M3 M4 M5 M6 M7 M8 M9 M10 M11
Software Decision Module	M2 M12 M13 M14 M15 M16

Table 4: Module Hierarchy

Note*: The Hardware-Hiding Module is not implemented in the hierarchy as there is no hardware involved in this software system.

4 Connection Between Requirements and Design

When designing the software system DNA Says, one of the main priorities was satisfying all functional and non-function requirements outlined in the SRS. These functionalities are truly the essence of the game and the design should support each and every one of them. Each module outlined in Section 3 serves an underlying purpose of satisfying the functions of the software. For example, updated functional requirement #11 states:

There will be a score icon in the top right corner.

Module #10: ShowScore, is intended to serve the function of displaying the score in the top right corner. This module was designed in a specific way to support the functionality of requirement #11. The connection between the full list of requirements and modules is listed in Table 5.

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.

5.1 Hardware Hiding Module (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

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: N/A

5.2.1 Menu Module (M3)

Secrets: Structure that displays the UI elements that need to appear on the main menu.

Services: Displays the main menu.

Implemented By: DNA Says

5.2.2 Setup Module (M7)

Secrets: The format and implementation of the displaying surface.

Services: Sets up the displaying surface. Creates a window of a set width and height, and labels it. All graphics can then be displayed on that window.

Implemented By: DNA Says

5.2.3 Update Module (M8)

Secrets: Updates the display with respect to any changes made to the interface via the user.

Services: Updates the display.

Implemented By: DNA Says

5.2.4 ShowInst Module (M9)

Secrets: Houses the way the instructions will be displayed to the user.

Services: Displays the instructions.

Implemented By: DNA Says

5.2.5 ShowScore Module (M10)

Secrets: The data structure that holds the score.

Services: Displays the score.

Implemented By: DNA Says

5.2.6 ShowGoBack Module (M11)

Secrets: The structure behind the main menu button.

Services: Displays the main menu button text.

Implemented By: DNA Says

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: N/A

5.3.1 Main Module (M2)

Secrets: The algorithm used to initialize clock, sounds, fonts, and buttons.

Services: Initializes Pygame along with Pygame's clock, all required font sizes, the required sounds, and the main menu button in its set position. This function then proceeds to call the main menu function.

Implemented By: DNA Says

5.3.2 JP Module (M4)

Secrets: The size of keys and their positions, the implementation of the display. The algorithm and implementation of the song pattern, both visual and auditory.

Services: Starts the mode JP Says with nine buttons in square form. It plays the elements that are contained in the pattern, then checks for the user's input. If the user clicked the correct button then its sound is played and its key is highlighted temporarily, and the score is incremented. In the meantime, the program appends a random element to the pattern to be played, and the pattern is played, and the cycle goes on. However, if the user clicks on the wrong key, the game will be over. The game will also end if the user clicks on one correct key, and waits for more than four seconds, assuming that there is more than one element in the last pattern played. If the user clicks on the main menu button, the program will redirect them to the main menu.

Implemented By: DNA Says

5.3.3 Kareem Module (M5)

Secrets: The size of keys and their positions, the implementation of the display. The algorithm and implementation of the song pattern, both visual and auditory.

Services: Starts the mode Kareem Says with a piano keyboard. It plays the elements that are contained in the pattern, then checks for the user's input. If the user clicked the correct button then its sound is played and its key is highlighted temporarily, and the score is incremented. In the meantime, the program appends a random element to the pattern to be played, and the pattern is played, and the cycle goes on. However, if the user clicks on the wrong key, the game will be over. The game will also end if the user clicks on one correct key, and waits for more than four seconds, assuming that there is more than one element in the last pattern played. If the user clicks on the main menu button, the program will redirect them to the main menu.

Implemented By: DNA Says

5.3.4 Shady Module (M6)

Secrets: The size of keys and their positions, the implementation of the display. The algorithm and implementation of the song pattern, both visual and auditory.

Services: Starts the mode Shady Says with four buttons in square shape. It plays the elements that are contained in the pattern, then checks for the user's input. If the user clicked the correct button then its sound is played and its key is highlighted temporarily, and the score is incremented. In the meantime, the program appends a random element to the pattern to be played, and the pattern is played, and the cycle goes on. However, if the user clicks on the wrong key, the game will be over. The game will also end if the user clicks on one correct key, and waits for more than four seconds, assuming that there is more than one element in the last pattern played. If the user clicks on the main menu button, the program will redirect them to the main menu.

Implemented By: DNA Says

5.3.5 DrawKeys Module (M12)

Secrets: The algorithm used to draw the buttons depending on the mode.

Services: Draws the buttons on the screen.

Implemented By: DNA Says

5.3.6 FlashKeyAnimation Module (M13)

Secrets: The logic for playing patterns and flashing keys depending on mode.

Services: Plays the appropriate sound and flashes the correct key. For Kareem Says, it loads the correct note, and all highlights in this mode are bright red. The color of the keys is either white or black. For JP Says, it again loads the correct note and also each button is colored differently. For Shady Says, the sound is randomized every time and is stored in that sound variable.

Implemented By: DNA Says

5.3.7 ChangeBackgroundAnimation Module (M14)

Secrets: The random RGB creation, the implementation of temporary display surface, the fading technique of colors.

Services: Changes the background color smoothly.

Implemented By: DNA Says

5.3.8 GameOverAnimation Module (M15)

Secrets: The structure of changing background color and implementation of chord played.

Services: Plays the animation associated with incorrect user input. It plays a C chord while flashing the screen three times to indicate that the game is over.

Implemented By: DNA Says

5.3.9 CheckForQuit Module (M16)

Secrets: The logic to detect quit request.

Services: Checks for quit requests. Quits the program at anytime upon the user's request.

Implemented By: DNA Says

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
R2	M2, M3
R3	M3, M4, M5, M6, M12
R4	M2, M3, M7, M12
R5	M4, M5, M6, M7, M8, M12
R6	M4, M5, M6, M8, M12, M13
R7	M13
R8	M16
R9	M10
R10	M10, M15
R11	M10
R12	M4, M5, M6, M13
R13	M4, M5, M6, M12
R14	M8, M12, M13
R15	M4, M5, M6
R16	M4, M5, M6, M14, M15

Table 5: Trace Between Requirements and Modules

AC	Modules
AC1	M1, M2
AC2	M2, M3, M7, M9, M10
AC3	M16
AC4	M2, M3, M12, M13
AC5	M3, M4, M5, M6, M9

Table 6: 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. 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.

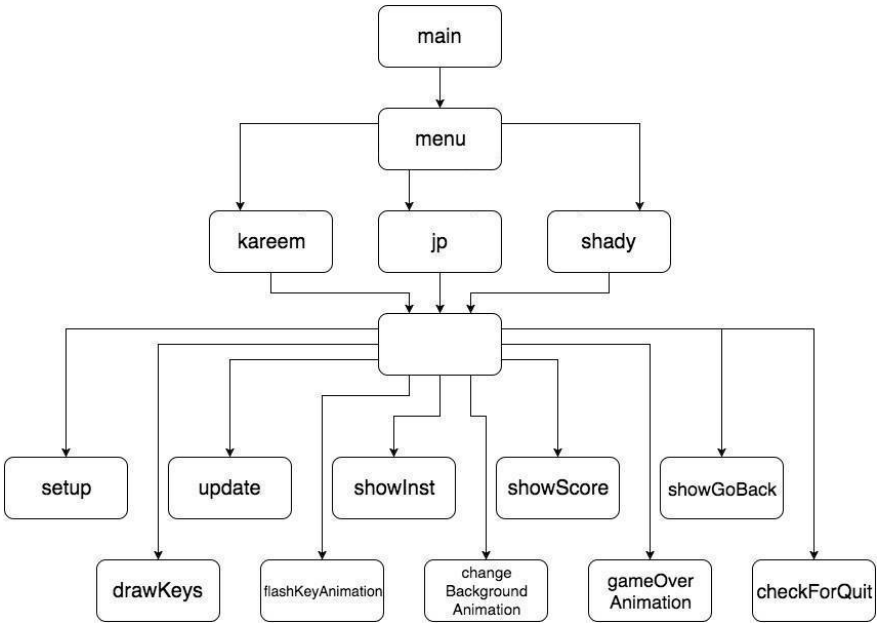

Figure 1: Use hierarchy among modules

*The middle unnamed box in the hierarchy module is not a module - it is a connection point for easier portrayal of the hierarchy.

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.