SE 3XA3: Module Guide
Mini-Arcade

Andrew Hum
huma3
400138826

William Lei
leim5
400125240

Arshan Khan
khana172
400145605

Jame Tran
tranj52
400144141

3/13/2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
3/9/2020	1.0	Arshan and Andrew created document
3/11/2020	1.1	Arshan added modules to section 3
3/11/2020	1.2	Andrew completed section 1, and sections 2, 3, 5, 6 where Maze modules and changes were relevant.

1 Introduction

1.1 Overview

Mini-Arcade is the re-implementation of three simple open-source Python games: Maze, Pong and Flappy. These games can be accessed through a simple easy-to-use launcher. These games are to be completely re-designed with improved functionality, graphics and performance.

1.2 Context

This document is the Module Guide (MG). This document is developed based upon the Software Requirements Specification (SRS) developed earlier. The SRS highlighted the functional and non-functional requirements of our system to ensure a thorough and correct solution. The MG focuses on the concept of modular decomposition to develop and display the modular structure of our project. After this document, the Module Interface Specification (MIS) will be created. The purpose of the MIS is to provide in-depth explanations of each modules individual syntax (exported access programs) and semantics (state variables, environment variables, assumptions and access program semantics).

1.3 Design Principles

The design principle guiding the module decomposition is Information Hiding. The use of this principle is important in module decomposition as it supports design for change, which is the concept of allowing each module to be modified often. This principle also supports the low-coupling, independence of modules, and high-cohesion, strong relation of elements, between the modules.

1.4 Document Structure

The document's structure is as follows.

- Section 2 lists the anticipated and unlikely changes of the software requirements
- Section 3 summarizes the decomposition of modules by Hardware-Hiding, Behaviour-Hiding and Software Decision
- Section 4 identifies the connections between the requirements of our project and the modules
- Section 5 gives a detailed description of the module decomposition
- Section 6 includes three traceability matrices. The first matrix checks the completeness of the design in regards to the Functional Requirements provided in the SRS. The second matrix checks the completeness of the design in regards to the Non-Functional

Requirements provided in the SRS. The last matrix highlights the relations between the anticipated changes and the modules

- Section 7 describes the use relation between modules

2 Anticipated and Unlikely Changes

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 of the program

AC2: The format of the input data.

AC3: The format and calculation of the score.

AC4: The details/style of the in-game graphics and the graphical user interface.

AC5: Character customization.

AC6: Theme customization.

AC7: The efficiency and variability of the maze generation algorithm

AC8: Create more options to configure the game (adjust sound, adjust brightness)

2.2 Unlikely Changes

UC1: Input/Output devices (Input: File, Keyboard, Mouse, Output: File, Memory, Screen).

UC2: There will always be a source of input data external to the software.

UC3: The purpose of the program to provide an easy and accessible way to launch mini-games.

3 Module Hierarchy

M1: Hardware-Hiding Module

M2: *Launcher Modules*

M3: Maze Generator (Maze)

M4: Game State (Maze)

M5: Score Tracking (Maze)

M6: Draw Game (Maze)

M7: Player Movement (Maze)

M8: Menu and Settings (Maze)

M9: Ball Trajectory (Pong)

M10: Score Tracking (Pong)

M11: Draw Game (Pong)

M12: Player Movement (Pong)

M13: Menu and Settings (Pong)

M14: *Flappy Modules*

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	? *Launcher Modules* Draw Game (Maze) Player Movement (Maze) Menu and Settings (Maze) Draw Game (Pong) Player Movement (Pong) Menu and Settings (Pong) *Flappy Modules*
Software Decision Module	? *Launcher Modules* Maze Generator (Maze) Game State (Maze) Score Tracking (Maze) Ball Trajectory (Pong) Score Tracking (Pong) *Flappy Modules*

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 and 4.

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 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, Python

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.

Implemented By: N/A

5.2.1 Draw Game (Maze) Module (M6)

Secrets: Graphics

Services: Draws the Maze (generated by the Maze Generation Module), Player, and Elapsed Time (defined by the Score Tracking Module)

Implemented By: Mini-Arcade, Python Libraries, pygame

5.2.2 Player Movement (Maze) Module (M7)

Secrets: Inputs

Services: Moves the player based upon the user's keyboard inputs

Implemented By: Mini-Arcade, Python Libraries, pygame

5.2.3 Menu and Settings (Maze) Module (M8)

Secrets: Graphics

Services: Allows the user to navigate the game features (How-to-Play, Settings, etc.) or return to the launcher.

Implemented By: Mini-Arcade, Python Libraries, pygame

5.2.4 Draw Game (Pong) Module (M11)

Secrets:

Services:

Implemented By: Mini-Arcade, Python Libraries, pygame

5.2.5 Player Movement (Pong) Module (M12)

Secrets:

Services:

Implemented By: Mini-Arcade, Python Libraries, pygame

5.2.6 Menu and Settings (Pong) Module (M13)

Secrets:

Services:

Implemented By: Mini-Arcade, Python Libraries, pygame

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 Maze Generator (Maze) Module (M3)

Secrets: Algorithm

Services: Generates a random maze based upon the desired difficulty

Implemented By: Mini-Arcade, Python Libraries

5.3.2 Maze Generator (Maze) Module (M4)

Secrets: Inputs

Services: Determines the state of the game to decide which action to take

Implemented By: Mini-Arcade, Python Libraries

5.3.3 Score Tracking (Maze) Module (M5)

Secrets: Points

Services: Records the elapsed time to complete the maze and records the data for the scoreboard.

Implemented By: Mini-Arcade, Python Libraries

5.3.4 Ball Trajectory (Pong) Module (M9)

Secrets:

Services:

Implemented By: Mini-Arcade, Python Libraries

5.3.5 Score Tracking (Pong) Module (M10)

Secrets:

Services:

Implemented By: Mini-Arcade, Python Libraries

6 Traceability Matrices

6.1 Functional Requirements Traceability Matrix

Req.	Modules
FR1	
FR2	
FR3	
FR4	
FR5	M5
FR6	M5
FR7	M8
FR8	M5
FR9	M3, M6
FR10	M8
FR11	M5, M6, M4
FR12	M7, M6
FR13	M8, M3, M6, M4
FR14	M8
FR15	
FR16	
FR17	
FR18	
FR19	
FR12	
FR21	
FR22	
FR23	
FR24	
FR25	
FR26	
FR27	
FR28	
FR29	
FR30	

Table 3: Trace Between Functional Requirements and Modules

6.2 Non-Functional Requirements Traceability Matrix

Req.	**Modules**
NFR1	M6, M7
NFR2	M6
NFR3	M8
NFR4	M7, M8
NFR5	M3, M6
NFR6	M7, M8, M6
NFR7	
NFR8	M3, M5, M6, M7, M8, M4
NFR9	M7, M8
NFR10	M6, M8
NFR11	M3, M5, M6, M7, M8, M4
NFR12	M3, M5, M6, M7, M8, M4

Table 4: Trace Between Non Functional Requirements and Modules

6.3 Anticipated Changes Traceability Matrix

AC	**Modules**
AC1	M1
AC2	M7
AC3	M5
AC4	M6, M8
AC5	M6
AC6	M6, M8
AC7	M3
AC8	M8

Table 5: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas 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

8 Gantt Schedule

See Gantt Chart at the following url: