SE 3XA3: Design Document
Mari0

Team 9, Ninetendo
David Hobson hobsondd
Jose Miguel Ballesteros ballesjm
Jeff Pineda pinedaj

December 8, 2016

Contents

1 Introduction **1**

2 Anticipated and Unlikely Changes **2**
- 2.1 Anticipated Changes 2
- 2.2 Unlikely Changes 2

3 Module Hierarchy **3**

4 Connection Between Requirements and Design **3**

5 Module Decomposition **4**
- 5.1 Hardware Hiding Modules (M1) 4
  - 5.1.1 Mouse Inputs (M2) 4
  - 5.1.2 Keyboard Inputs (M3) 5
- 5.2 Behaviour-Hiding Module 5
  - 5.2.1 Input Format Module (M4) 5
  - 5.2.2 Portal Physics Module (M5) 5
  - 5.2.3 Portal Gun Module (M6) 5
  - 5.2.4 Game Object Module (M7) 5
  - 5.2.5 Player Object Module (M8) 6
- 5.3 Software Decision Module 6
  - 5.3.1 Sprite/Model Module (M9) 6
  - 5.3.2 Interface Module (M10) 6
  - 5.3.3 Collision Detection Module (M11) 6

6 Traceability Matrix **6**

7 Use Hierarchy Between Modules **7**

List of Tables

1 **Revision History** 1
2 Module Hierarchy 4
3 Trace Between Requirements and Modules 7
4 Trace Between Anticipated Changes and Modules 7

List of Figures

1 Use hierarchy among modules 8

Table 1: **Revision History**

Date	**Version**	**Notes**
2016-11-13	1.0	Creation of Design Document. Added Introduction. Added Anticipated and Unlikely Changes. Added Module Hierarchy. Added Connection Between Requirements and Design. Added Module Decomposition. Added Traceability Matrix. Added Use Hierarchy Between Modules.
2016-11-29	1.1	Updated Module Decomposition. Fixed Spelling Errors. Updated Traceability Matrix for Anticipated Changes and Modules. Updated Traceability Matrix for Requirements and Modules.

1 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (**?**). We advocate a decomposition based on the principle of information hiding (**?**). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by **?**, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (**?**). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- 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.

- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

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.

2 Anticipated and Unlikely Changes

This section shows some of the upcoming changes to Mari0 listed into two categories. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

2.1 Anticipated Changes

Some of the changes below are what we believe will be implemented as development continues. Due to the modularization of our software, many of the changes that will happen will be simple and allow the system to function properly.

AC1: Level Environments. All of the levels that will be created will be unique and making sure that game objects interact properly is crucial.

AC2: Types of Enemies. Adding new enemies in the game is expected and is considered in the design.

AC3: Operating System. All operating systems will be supported and allowing the software to be dynamic and run on these systems is important.

AC4: Music and Sound Effects. These will be changed based off of different environmental factors of the game.

2.2 Unlikely Changes

The changes listed below are changes that will not be considered as important for the final product and demonstration.

UC1: Platform. For this game, the only platform that it will run on will be a personal computer, although operating systems may change, this game will not run on mobile or game console devices.

UC2: Sprites/Models. Although in the original game there?s customization available for sprites, this will not be considered in our design.

UC3: Game Mechanics. The basic idea of the game will stay as a platformer with the player walking to the right and interacting with different obstacles.

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: Hardware-Hiding Module

M2: Mouse Input Module

M3: Keyboard Input Module

M4: Input Format Module

M5: Portal Physics Module

M6: Portal Gun Module

M7: Game Object Module

M8: Player Object Module

M9: Sprite/Model Module

M10: Interface Module

M11: Collision Detection Module

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.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Collision Detection Module Event Input Module Interface Module Sprite/Model Module Portal Physics Module
Software Decision Module	Game Object Module Player Object Module

Table 2: Module Hierarchy

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

5.1.1 Mouse Inputs (M2)

Secrets: X and Y coordinates of Mouse

Services: Player mouse inputs are displayed on-screen

Implemented By: OS

5.1.2 Keyboard Inputs (M3)

Secrets: Keys Pressed

Services: Player keyboard inputs are displayed on-screen

Implemented By: OS

5.2 Behaviour-Hiding Module

5.2.1 Input Format Module (M4)

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: Mari0

5.2.2 Portal Physics Module (M5)

Secrets: How teleportation between two portals works and the game entities that can enter a portal

Services: Teleports game entities that can be teleported between two portals

Implemented By: Mari0

5.2.3 Portal Gun Module (M6)

Secrets: The coordinates of both the blue and the orange portal on a surface

Services: Allows the player to aim and place either a blue portal or an orange portal on a surface

Implemented By: Mari0

5.2.4 Game Object Module (M7)

Secrets: The size and coordinates of the game object

Services: Allows size and coordinates to be changed and be interacted with by the player

Implemented By: Mari0

5.2.5 Player Object Module (M8)

Secrets: Size, coordinates of player object, player status (alive or dead), and player attributes such as speed, jump force, and gravity.

Services: Allows size and coordinates to be changed and be interacted with by the player and keeps track of player status

Implemented By: Mari0

5.3 Software Decision Module

5.3.1 Sprite/Model Module (M9)

Secrets: How sprites and models are displayed in game

Services: Displays sprites and models in game

Implemented By: Mari0

5.3.2 Interface Module (M10)

Secrets: Information about the game, such as time left, points scored, and current level

Services: Keeps track of and displays game time, score, and current level

Implemented By: Mari0

5.3.3 Collision Detection Module (M11)

Secrets: Collision Mechanics

Services: Determines whether two entities in the game are touching or will eventually touch one another

Implemented By: Mari0

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, M3, M4, M10
R2	M3, M4, M10
R3	M3, M10
R4	M9, M10
R5	M2, M5, M6, M7, M8, M11
R6	M3, M8, M9, M11, M??
R7	M3, M8, M11
R8	M7, M8, M??, M11
R9	M7, M9, M11
R10	M7, M8, M9, M11
R11	M7, M8, M9, M11
R12	M8
R13	M7, M8, M10

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC1	M5, M7, M9
AC2	M7, M8, M9, M10, M11
AC3	M1, M2, M3, M4
AC4	M7

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.

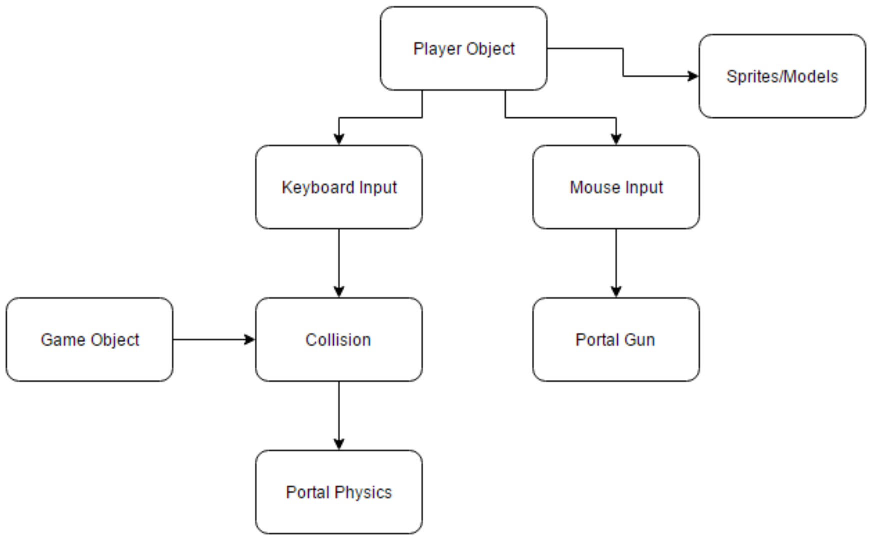

Figure 1: Use hierarchy among modules