

# Module Guide for Software Eng

Team #11, Mac-AR

Student 1 Matthew Collard

Student 2 Sam Gorman

Student 3 Ethan Kannampuzha

Student 4 Kieran Gara

April 5, 2024

# 1 Revision History

Date	Version	Notes
January 13	1.0	Initial changes
January 14	1.1	Module hierarchy chart and specifications
January 17	1.2	Revision 0
April 4	1.3	Timeline information moved to a table
April 4	1.4	AC3 updated to be more specific. Module types fixed to be correct (updated puzzle modules to have abstract object rather than library, game room module to abstract data type)
April 4	1.5	Template content (ie. Dr. Smith's comments) removed. Non-leaf modules have had dashes added
April 4	1.6	Updated user interface images
April 4	1.7	Updated traceability matrix

## 2 Reference Material

This section records information for easy reference.

### 2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
AR	Augmented Reality
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Eng	Explanation of program name
UC	Unlikely Change

# Contents

<b>1 Revision History</b>	i
<b>2 Reference Material</b>	ii
2.1 Abbreviations and Acronyms . . . . .	ii
<b>3 Introduction</b>	1
<b>4 Anticipated and Unlikely Changes</b>	2
4.1 Anticipated Changes . . . . .	2
4.2 Unlikely Changes . . . . .	2
<b>5 Module Hierarchy</b>	3
<b>6 Connection Between Requirements and Design</b>	4
<b>7 Module Decomposition</b>	4
7.1 Hardware Hiding Module (M1) (-) . . . . .	4
7.2 Behaviour-Hiding Module (-) . . . . .	5
7.2.1 Game Room Module (M2) . . . . .	5
7.2.2 Text Communication Module (M3) . . . . .	5
7.2.3 Voice Communication Module (M4) . . . . .	5
7.2.4 Multiplayer Puzzle Module (M5) . . . . .	6
7.2.5 Simon Says Puzzle Module (M6) . . . . .	6
7.2.6 Isometric Puzzle Module (M7) . . . . .	6
7.2.7 Wires Puzzle Module (M8) . . . . .	6
7.2.8 Maze Puzzle Module (M9) . . . . .	7
7.2.9 Combination Puzzle Module (M10) . . . . .	7
7.3 Software Decision Module (-) . . . . .	7
7.3.1 Database/Network Manager Module(M11) . . . . .	7
7.3.2 Error Manager Module(M12) . . . . .	8
7.3.3 Documentation Module(M13) (-) . . . . .	8
<b>8 Traceability Matrix</b>	8
<b>9 Use Hierarchy Between Modules</b>	13
<b>10 User Interfaces</b>	15
<b>11 Timeline</b>	22

## List of Tables

1	Module Hierarchy . . . . .	4
2	Trace Between Requirements and Modules . . . . .	9
3	Cont. Trace Between Requirements and Modules . . . . .	10
4	Cont. Trace Between Requirements and Modules . . . . .	11
5	Cont. Trace Between Requirements and Modules . . . . .	12
6	Cont. Trace Between Requirements and Modules . . . . .	12
7	Trace Between Anticipated Changes and Modules . . . . .	13

## List of Figures

1	Use hierarchy among modules . . . . .	14
2	Main Menu Screen . . . . .	15
3	Host/Join Game Screen . . . . .	16
4	Create Lobby Screen . . . . .	17
5	Lobby Screen . . . . .	18
6	Combination Puzzle Image . . . . .	19
7	Wires Puzzle Image . . . . .	20
8	Simon Says Puzzle Image . . . . .	21
9	Isometric Puzzle Image . . . . .	22

### 3 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 (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). 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 Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented 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 [System Requirement Specifications](#) (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). 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 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 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 9 describes the use relation between modules.

## 4 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 4.1, and unlikely changes are listed in Section 4.2.

### 4.1 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:** Additional puzzle modules

**AC2:** Modification of puzzles to involve more users

**AC3:** Update versions of third-party frameworks used in the creation of the application (such as updating from Vivox version 15.1.2)

**AC4:** The user interface layout of the application

**AC5:** The number of users that can use the application at one time (database/network size)

**AC6:** The number of users that can play cooperatively in the same game room

### 4.2 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:** The engine that the application is made from (ie. Unity engine)

**UC2:** The addition of an account system for users

**UC3:** The ability to play non-cooperatively (single player)

**UC4:** The environment that the user will access the game environment from

**UC5:** User input for gameplay controls

## 5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

**M1:** Hardware Module

**M2:** Game Room Module

**M3:** Text Communication Module

**M4:** Voice Communication Module

**M5:** Multiplayer Puzzle Module

**M6:** Simon Says Puzzle Module

**M7:** Isometric Puzzle Module

**M8:** Wires Puzzle Module

**M9:** Maze Puzzle Module

**M10:** Combination Puzzle Module

**M11:** Database/Network Manager Module

**M12:** Error Manager Module

**M13:** Documentation Module

Level 1	Level 2
Hardware-Hiding Module	Hardware Module
Behaviour-Hiding Module	Game Room Module Text Communication Module Voice Communication Module Multiplayer Puzzle Module Simon Says Puzzle Module Isometric Puzzle Module Wires Puzzle Module Maze Puzzle Module Combination Puzzle Module
Software Decision Module	Database/Network Manager Module Error Manager Module Documentation Module

Table 1: Module Hierarchy

## 6 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 2.

## 7 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.

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.

### 7.1 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

## 7.2 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:** Mac-AR

### 7.2.1 Game Room Module (M2)

**Secrets:** The format and structure of the game room objects.

**Services:** Allows users to create and maintain a game room, as well as join and leave game rooms.

**Implemented By:** Mac-AR

**Type of Module:** Abstract Data Type

### 7.2.2 Text Communication Module (M3)

**Secrets:** The format and structure of text chat data objects

**Services:** Provide functionality allowing users to communicate with other users through text and voice

**Implemented By:** Mac-AR (Vivox framework)

**Type of Module:** Abstract Data Type

### 7.2.3 Voice Communication Module (M4)

**Secrets:** The format and structure of voice chat data objects

**Services:** Provide functionality allowing users to communicate with other users through text and voice

**Implemented By:** Mac-AR (Vivox framework)

**Type of Module:** Abstract Data Type

#### 7.2.4 Multiplayer Puzzle Module (M5)

**Secrets:** The format and structure of puzzle objects

**Services:** Allows users to interact with puzzle objects and their auxiliary components of hints and skips

**Implemented By:** Mac-AR

**Type of Module:** Abstract data type

#### 7.2.5 Simon Says Puzzle Module (M6)

**Secrets:** The format and structure of Simon Says puzzle object

**Services:** Allows users to interact with an instance of the puzzle module, which is a memory based Simon Says puzzle

**Implemented By:** Mac-AR

**Type of Module:** Abstract Object

#### 7.2.6 Isometric Puzzle Module (M7)

**Secrets:** The format and structure of Isometric puzzle object

**Services:** Allows users to interact with an instance of the Isometric puzzle, which is a puzzle you have to inspect the arrangement of blocks in 3d space to see letter and number pairs to decipher a word.

**Implemented By:** Mac-AR

**Type of Module:** Abstract Object

#### 7.2.7 Wires Puzzle Module (M8)

**Secrets:** The format and structure of Wires puzzle object

**Services:** Allows users to interact with an instance of the puzzle module, which is a wire connection puzzle

**Implemented By:** Mac-AR

**Type of Module:** Abstract Object

### **7.2.8 Maze Puzzle Module (M9)**

**Secrets:** The format and structure of Maze puzzle object

**Services:** Allows users to interact with an instance of the puzzle module, which is a maze puzzle which can be tilted and rotated

**Implemented By:** Mac-AR

**Type of Module:** Abstract Object

### **7.2.9 Combination Puzzle Module (M10)**

**Secrets:** The format and structure of Combination puzzle object

**Services:** Allows users to interact with an instance of the puzzle module, which is a keypad with associated instructions, from which the users must determine the four digit combination

**Implemented By:** Mac-AR

**Type of Module:** Abstract Object

## **7.3 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:** Mac-AR

### **7.3.1 Database/Network Manager Module(M11)**

**Secrets:** The format and structure of the establishment of the connection and storage of data between multiple players.

**Services:** Provide functionality for establishing the connection of users to the network and associated database that allows for data to be transferred and stored in between different users devices.

**Implemented By:** Mac-AR

**Type of Module:** Abstract Object

### **7.3.2 Error Manager Module(M12)**

**Secrets:** The error handling methods.

**Services:** Provide functionality for handling all types of error affecting the game functionality including network/connectivity issues and environment issues.

**Implemented By:** Mac-AR

**Type of Module:** Abstract Object

### **7.3.3 Documentation Module(M13) (-)**

**Secrets:** N/A

**Services:** Provide mapping of requirements to user documentation

**Implemented By:** Mac-AR

**Type of Module:** N/A

## **8 Traceability Matrix**

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
CG1	M2, M11
CG2	M2, M11
CG3	M2, M11
CG4	M2, M11
CG5	M2, M11
JG1	M2, M11
JG2	M2, M11
JG3	M2, M11
JG4	M2, M11
JG5	M2, M11
RS1	M2, M11
RS2	M2, M11
RS3	M2, M11
RS4	M2, M11
ER1	M2, M11
ER2	M2, M11
ER3	M2, M11
ST1	M2, M11
ST2	M2, M11
ST3	M2, M11
ST4	M2, M11

Table 2: Trace Between Requirements and Modules

Req.	Modules
PI1	M <sub>5</sub> , M <sub>6</sub> , M <sub>7</sub> , M <sub>8</sub> , M <sub>9</sub> , M <sub>10</sub>
PI2	M <sub>5</sub> , M <sub>6</sub> , M <sub>7</sub> , M <sub>8</sub> , M <sub>9</sub> , M <sub>10</sub>
PI3	M <sub>5</sub> , M <sub>6</sub> , M <sub>7</sub> , M <sub>8</sub> , M <sub>9</sub> , M <sub>10</sub>
PI4	M <sub>5</sub> , M <sub>6</sub> , M <sub>7</sub> , M <sub>8</sub> , M <sub>9</sub> , M <sub>10</sub>
HR1	M <sub>5</sub>
HR2	M <sub>5</sub>
HR3	M <sub>5</sub>
SP1	M <sub>5</sub>
SP2	M <sub>5</sub>
SM1	M <sub>3</sub> , M <sub>11</sub>
SM2	M <sub>3</sub> , M <sub>11</sub>
SM3	M <sub>3</sub> , M <sub>11</sub>
SM4	M <sub>3</sub> , M <sub>11</sub>
RM1	M <sub>3</sub> , M <sub>11</sub>
RM2	M <sub>3</sub> , M <sub>11</sub>
RM3	M <sub>3</sub> , M <sub>11</sub>
VM1	M <sub>4</sub> , M <sub>11</sub>
VM2	M <sub>4</sub> , M <sub>11</sub>

Table 3: Cont. Trace Between Requirements and Modules

Req.	Modules
SS1	M6, M5
SS2	M6, M5
SS3	M6, M5
SS4	M6, M5
SS5	M6, M5
SS6	M6, M5
SS7	M6, M5
MP1	M9, M5
MP2	M9, M5
MP3	M9, M5
MP4	M9, M5
MP5	M9, M5
IP1	M7, M5
IP2	M7, M5
IP3	M7, M5
IP4	M7, M5
IP5	M7, M5
CP1	M10, M5
CP2	M10, M5
CP3	M10, M5
CP4	M10, M5
CP5	M10, M5
CP6	M10, M5
WP1	M8, M5
WP2	M8, M5
WP3	M8, M5
WP4	M8, M5
WP5	M8, M5
WP6	M8, M5
WP7	M8, M5
WP8	M8, M5
WP9	M8, M5
WP10	M8, M5

Table 4: Cont. Trace Between Requirements and Modules

Req.	Modules
LF1	M <sub>2</sub> , M <sub>1</sub> , M <sub>5</sub>
LF2	M <sub>2</sub> , M <sub>5</sub>
UH1	M <sub>1</sub> , M <sub>5</sub>
UH2	M <sub>12</sub> , M <sub>11</sub>
UH3	M <sub>2</sub> , M <sub>5</sub>
UH4	M <sub>12</sub> , M <sub>11</sub>
UH5	M <sub>12</sub> , M <sub>11</sub>
UH6	M <sub>12</sub> , M <sub>11</sub> , M <sub>2</sub>
UH7	M <sub>12</sub> , M <sub>11</sub>

Table 5: Cont. Trace Between Requirements and Modules

Req.	Modules
PR1	M <sub>2</sub> , M <sub>3</sub> , M <sub>4</sub> , M <sub>11</sub> , M <sub>5</sub> , M <sub>6</sub> , M <sub>7</sub> , M <sub>8</sub> , M <sub>8</sub> , M <sub>9</sub>
PR2	M <sub>2</sub> , M <sub>3</sub> , M <sub>4</sub> , M <sub>11</sub> , M <sub>5</sub> , M <sub>6</sub> , M <sub>7</sub> , M <sub>8</sub> , M <sub>8</sub> , M <sub>9</sub>
OE1	M <sub>1</sub>
MS1	M <sub>13</sub>
MS2	M <sub>13</sub>
SR1	M <sub>2</sub> , M <sub>11</sub> , M <sub>5</sub>
SR2	M <sub>2</sub> , M <sub>11</sub> , M <sub>5</sub>
CR1	M <sub>2</sub> , M <sub>11</sub> , M <sub>5</sub> , M <sub>13</sub>
CR2	M <sub>2</sub> , M <sub>11</sub> , M <sub>5</sub> , M <sub>13</sub>
LR1	M <sub>1</sub> , M <sub>2</sub> , M <sub>3</sub> , M <sub>4</sub> , M <sub>5</sub> , M <sub>6</sub> , M <sub>7</sub> , M <sub>8</sub> , M <sub>9</sub> , M <sub>10</sub> , M <sub>11</sub> , M <sub>12</sub> , M <sub>13</sub>
HS1	M <sub>12</sub>

Table 6: Cont. Trace Between Requirements and Modules

<b>AC</b>	<b>Modules</b>
AC1	M5
AC2	M5
AC3	M3, M4, M2
AC4	M2, M5
AC5	M2, M11
AC6	M2, M5, M11

Table 7: Trace Between Anticipated Changes and Modules

## 9 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.

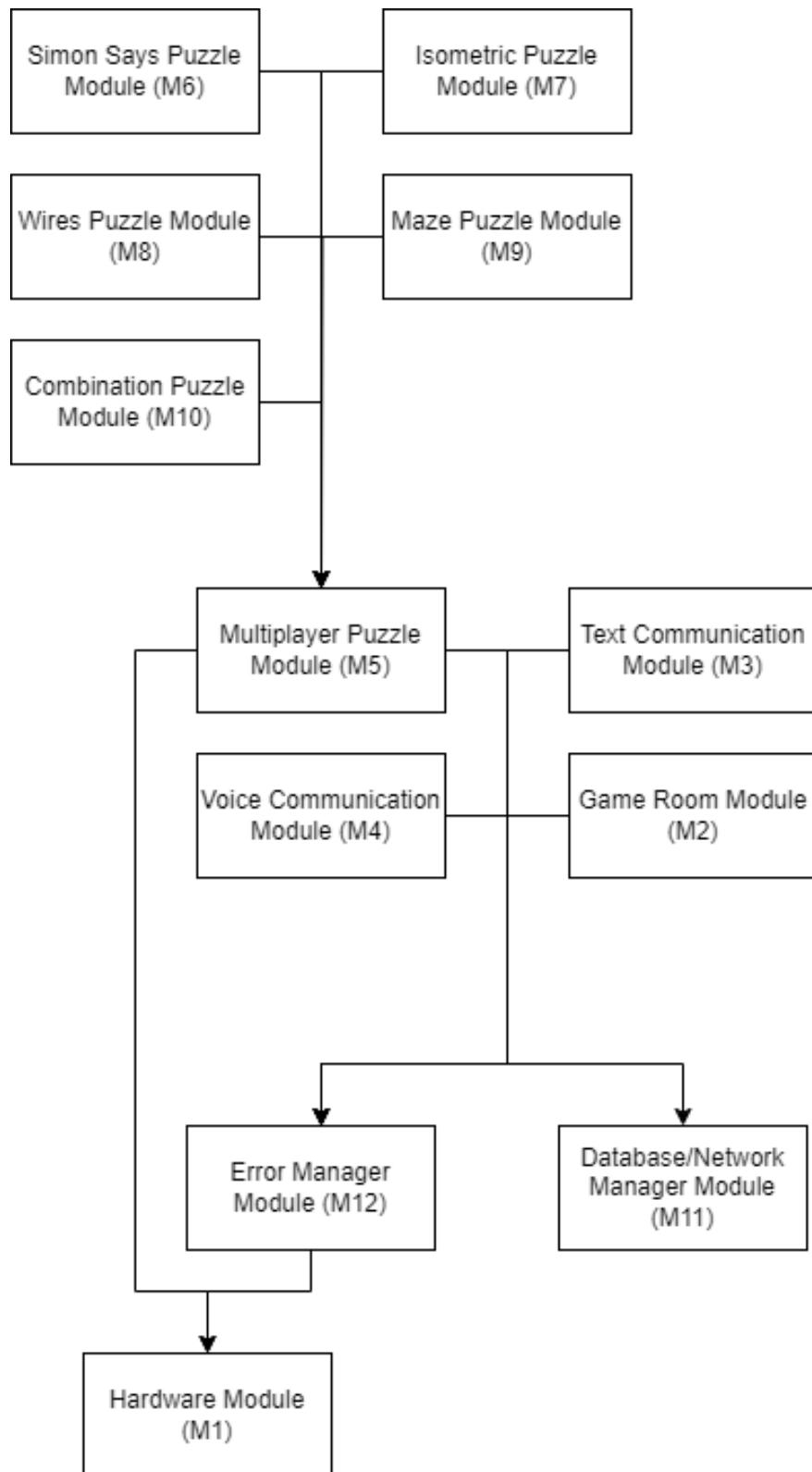


Figure 1: Use hierarchy among modules

## 10 User Interfaces

Here are some photos showing the user interface design of the application:

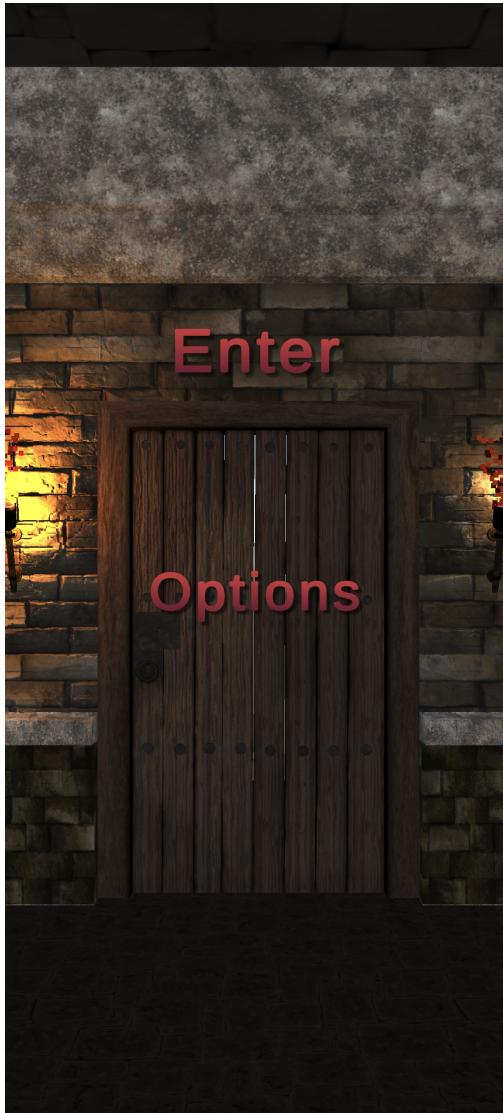


Figure 2: Main Menu Screen

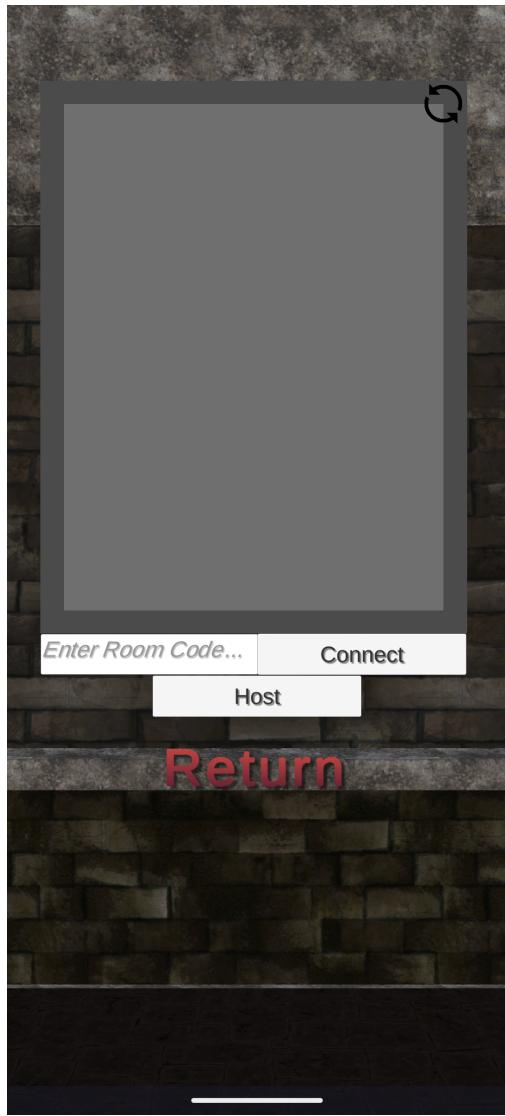


Figure 3: Host/Join Game Screen

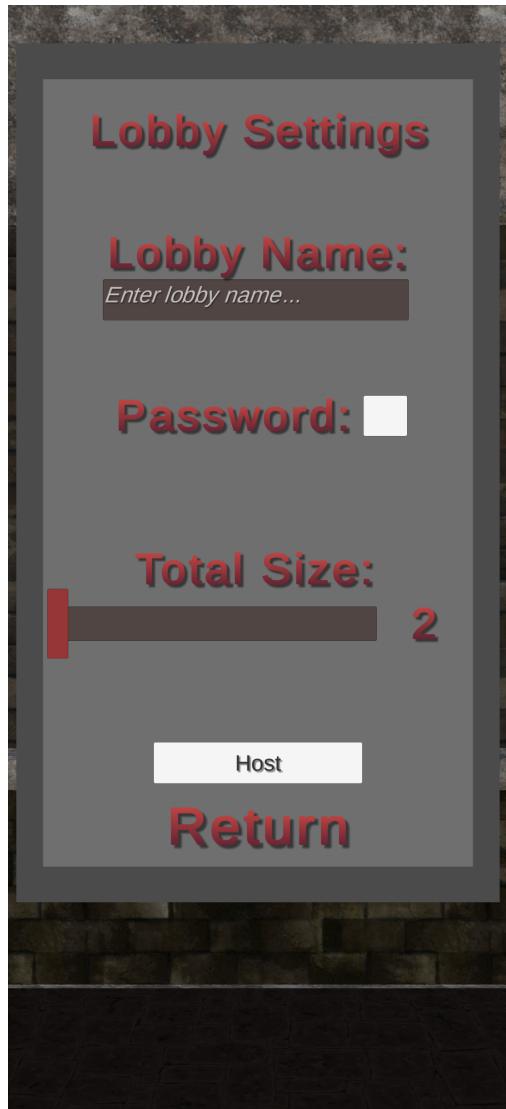


Figure 4: Create Lobby Screen



Figure 5: Lobby Screen

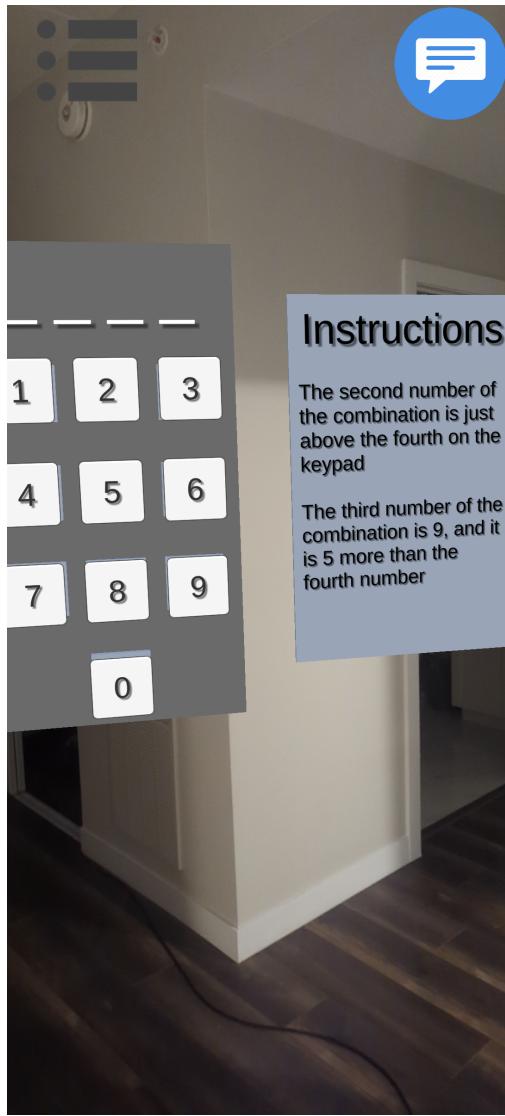


Figure 6: Combination Puzzle Image



Figure 7: Wires Puzzle Image

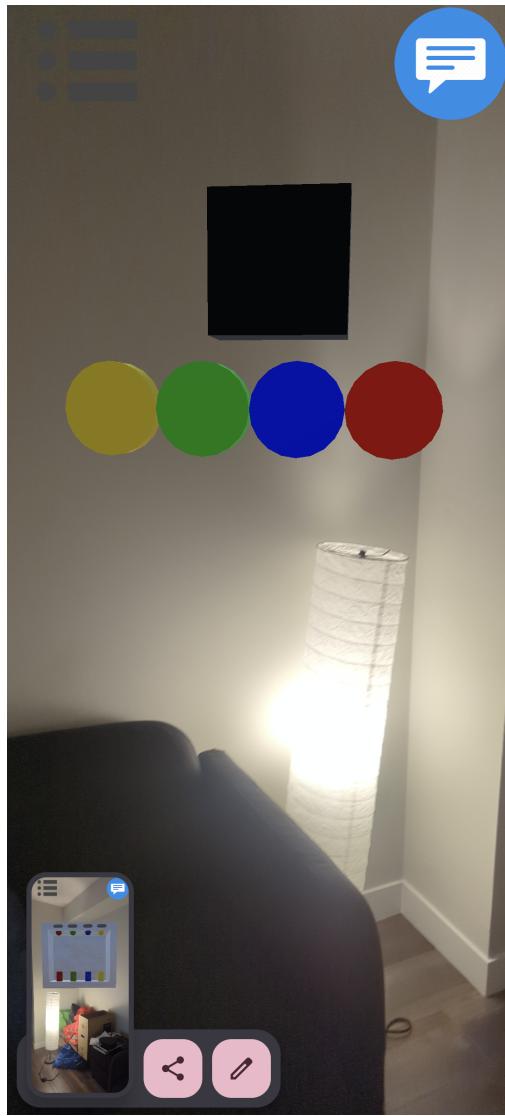


Figure 8: Simon Says Puzzle Image

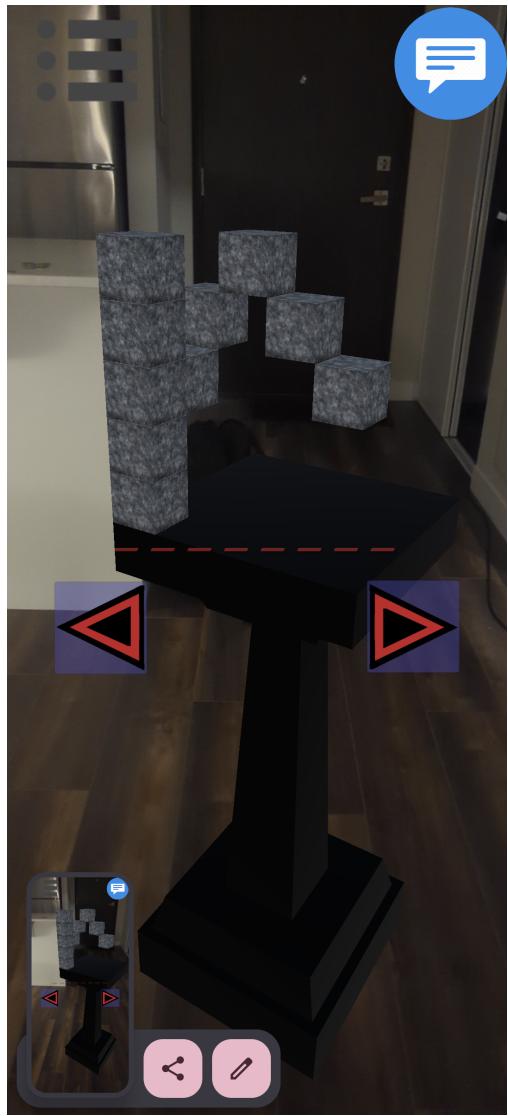


Figure 9: Isometric Puzzle Image

## 11 Timeline

The following timeline describes the work that needs to be completed in order for the implementation of Rev 0 to be complete

<b>Due Date</b>	<b>Work to be Completed</b>	<b>Assigned Member</b>
Jan 15 - Jan 25	Game Room Module	Matthew
	Text Communication Module	Kieran
	Voice Communication Module	Ethan
Jan 20 - Feb 6	Multiplayer Puzzle Module	Sam
	Simon Says Puzzle Module	Matthew
	Isometric Puzzle Module	Matthew
	Wires Puzzle Module	Sam
	Maze Puzzle Module	Ethan
	Combination Puzzle Module	Kieran
Jan 28 - Feb 10	Error Manager Module	Kieran and Matthew
	Database/Network Manager Module	Ethan and Sam
	Testing and Verification	Entire Team

## References

- David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.
- 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.