

# Module Interface Specification 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, 2024	1.0	Initial changes
January 15	1.1	Module Definitions
January 16	1.2	Module Details
January 17	1.3	Appendix
January 18	1.4	Revision 0
April 2	1.5	Adding new modules and removing old ones
April 4	1.6	Updated environment variable format Updated modules to match graph General syntax updates/ Made naming conventions consistent Updated descriptions to make it clear when functions were part of existing frameworks Added constructors to abstract modules Moved local variables to state variables where applicable Cleared up some semantics (CheckEntry, etc.) Added note justifying existence of error module Cleared up ambiguous statements Improved detail of reflection

## 2 Symbols, Abbreviations and Acronyms

See SRS Documentation [here](#)

symbol	description
AR	Augmented Reality
M	Module
MG	Module Guide
MIS	Module Interface Specification
OS	Operating System
SRS	Software Requirements Specification

# Contents

<b>1</b>	<b>Revision History</b>	<b>i</b>
<b>2</b>	<b>Symbols, Abbreviations and Acronyms</b>	<b>ii</b>
<b>3</b>	<b>Introduction</b>	<b>1</b>
<b>4</b>	<b>Notation</b>	<b>1</b>
<b>5</b>	<b>Module Decomposition</b>	<b>2</b>
<b>6</b>	<b>MIS of Hardware Module</b>	<b>3</b>
6.1	Module . . . . .	3
6.2	Uses . . . . .	3
6.3	Syntax . . . . .	3
6.3.1	Exported Constants . . . . .	3
6.3.2	Exported Access Programs . . . . .	3
6.4	Semantics . . . . .	3
6.4.1	State Variables . . . . .	3
6.4.2	Environment Variables . . . . .	3
6.4.3	Assumptions . . . . .	3
6.4.4	Access Routine Semantics . . . . .	3
6.4.5	Local Functions . . . . .	3
<b>7</b>	<b>MIS of Game Room Module</b>	<b>4</b>
7.1	Module . . . . .	4
7.2	Uses . . . . .	4
7.3	Syntax . . . . .	4
7.3.1	Exported Constants . . . . .	4
7.3.2	Exported Access Programs . . . . .	4
7.4	Semantics . . . . .	4
7.4.1	State Variables . . . . .	4
7.4.2	Environment Variables . . . . .	5
7.4.3	Assumptions . . . . .	5
7.4.4	Access Routine Semantics . . . . .	5
7.4.5	Local Functions . . . . .	6
<b>8</b>	<b>MIS of Text Communication Module</b>	<b>7</b>
8.1	Module . . . . .	7
8.2	Uses . . . . .	7
8.3	Syntax . . . . .	7
8.3.1	Exported Constants . . . . .	7
8.3.2	Exported Access Programs . . . . .	7

8.4	Semantics . . . . .	7
8.4.1	State Variables . . . . .	7
8.4.2	Environment Variables . . . . .	8
8.4.3	Assumptions . . . . .	8
8.4.4	Access Routine Semantics . . . . .	8
8.4.5	Local Functions . . . . .	9
8.4.6	Local Variables . . . . .	9
<b>9</b>	<b>MIS of Voice Communication Module</b>	<b>10</b>
9.1	Module . . . . .	10
9.2	Uses . . . . .	10
9.3	Syntax . . . . .	10
9.3.1	Exported Constants . . . . .	10
9.3.2	Exported Access Programs . . . . .	10
9.4	Semantics . . . . .	10
9.4.1	State Variables . . . . .	10
9.4.2	Environment Variables . . . . .	11
9.4.3	Assumptions . . . . .	11
9.4.4	Access Routine Semantics . . . . .	11
9.4.5	Local Functions . . . . .	12
<b>10</b>	<b>MIS of Multiplayer Puzzle Module</b>	<b>13</b>
10.1	Module . . . . .	13
10.2	Uses . . . . .	13
10.3	Syntax . . . . .	13
10.3.1	Exported Constants . . . . .	13
10.3.2	Exported Access Programs . . . . .	13
10.4	Semantics . . . . .	14
10.4.1	State Variables . . . . .	14
10.4.2	Environment Variables . . . . .	14
10.4.3	Assumptions . . . . .	14
10.4.4	Access Routine Semantics . . . . .	14
10.4.5	Local Functions . . . . .	15
10.4.6	Local Variables . . . . .	15
<b>11</b>	<b>MIS of Simon Says Puzzle Module</b>	<b>16</b>
11.1	Module . . . . .	16
11.2	Uses . . . . .	16
11.3	Syntax . . . . .	16
11.3.1	Exported Constants . . . . .	16
11.3.2	Exported Access Programs . . . . .	16
11.4	Semantics . . . . .	17
11.4.1	State Variables . . . . .	17

11.4.2	Environment Variables . . . . .	17
11.4.3	Assumptions . . . . .	17
11.4.4	Access Routine Semantics . . . . .	17
11.4.5	Local Functions . . . . .	18
11.4.6	Local Variables . . . . .	18
<b>12</b>	<b>MIS of Isometric Puzzle Module</b>	<b>19</b>
12.1	Module . . . . .	19
12.2	Uses . . . . .	19
12.3	Syntax . . . . .	19
12.3.1	Exported Constants . . . . .	19
12.3.2	Exported Access Programs . . . . .	19
12.4	Semantics . . . . .	19
12.4.1	State Variables . . . . .	19
12.4.2	Environment Variables . . . . .	20
12.4.3	Assumptions . . . . .	20
12.4.4	Access Routine Semantics . . . . .	20
12.4.5	Local Functions . . . . .	21
12.4.6	Local Variables . . . . .	21
<b>13</b>	<b>MIS of Wires Puzzle Module</b>	<b>22</b>
13.1	Module . . . . .	22
13.2	Uses . . . . .	22
13.3	Syntax . . . . .	22
13.3.1	Exported Constants . . . . .	22
13.3.2	Exported Access Programs . . . . .	22
13.4	Semantics . . . . .	22
13.4.1	State Variables . . . . .	22
13.4.2	Environment Variables . . . . .	22
13.4.3	Assumptions . . . . .	22
13.4.4	Access Routine Semantics . . . . .	23
13.4.5	Local Functions . . . . .	23
13.4.6	Local Variables . . . . .	23
<b>14</b>	<b>MIS of Maze Puzzle Module</b>	<b>24</b>
14.1	Module . . . . .	24
14.2	Uses . . . . .	24
14.3	Syntax . . . . .	24
14.3.1	Exported Constants . . . . .	24
14.3.2	Exported Access Programs . . . . .	24
14.4	Semantics . . . . .	24
14.4.1	State Variables . . . . .	24
14.4.2	Environment Variables . . . . .	24

14.4.3	Assumptions . . . . .	25
14.4.4	Access Routine Semantics . . . . .	25
14.4.5	Local Functions . . . . .	26
14.4.6	Local Variables . . . . .	26
<b>15</b>	<b>MIS of Combination Puzzle Module</b>	<b>27</b>
15.1	Module . . . . .	27
15.2	Uses . . . . .	27
15.3	Syntax . . . . .	27
15.3.1	Exported Constants . . . . .	27
15.3.2	Exported Access Programs . . . . .	27
15.4	Semantics . . . . .	27
15.4.1	State Variables . . . . .	27
15.4.2	Environment Variables . . . . .	27
15.4.3	Assumptions . . . . .	28
15.4.4	Access Routine Semantics . . . . .	28
15.4.5	Local Functions . . . . .	29
15.4.6	Local Variables . . . . .	29
<b>16</b>	<b>MIS of Database/Network Manager Module</b>	<b>30</b>
16.1	Module . . . . .	30
16.2	Uses . . . . .	30
16.3	Syntax . . . . .	30
16.3.1	Exported Constants . . . . .	30
16.3.2	Exported Access Programs . . . . .	30
16.4	Semantics . . . . .	30
16.4.1	State Variables . . . . .	30
16.4.2	Environment Variables . . . . .	30
16.4.3	Assumptions . . . . .	30
16.4.4	Access Routine Semantics . . . . .	31
16.4.5	Local Functions . . . . .	31
<b>17</b>	<b>MIS of Error Manager Module</b>	<b>32</b>
17.1	Module . . . . .	32
17.2	Uses . . . . .	32
17.3	Syntax . . . . .	32
17.3.1	Exported Constants . . . . .	32
17.3.2	Exported Access Programs . . . . .	32
17.4	Semantics . . . . .	32
17.4.1	State Variables . . . . .	32
17.4.2	Environment Variables . . . . .	32
17.4.3	Assumptions . . . . .	32
17.4.4	Access Routine Semantics . . . . .	32

17.4.5	Local Functions . . . . .	33
<b>18</b>	<b>MIS of Documentation Module</b>	<b>34</b>
18.1	Module . . . . .	34
18.2	Uses . . . . .	34
18.3	Syntax . . . . .	34
18.3.1	Exported Constants . . . . .	34
18.3.2	Exported Access Programs . . . . .	34
18.4	Semantics . . . . .	34
18.4.1	State Variables . . . . .	34
18.4.2	Environment Variables . . . . .	34
18.4.3	Assumptions . . . . .	34
18.4.4	Access Routine Semantics . . . . .	34
18.4.5	Local Functions . . . . .	34
<b>19</b>	<b>Appendix</b>	<b>36</b>



### 3 Introduction

The following document details the Module Interface Specifications for the Mac-AR Augmented Reality escape room style game.

Complementary documents include the [System Requirement Specifications](#) and [Module Guide](#). The full documentation and implementation can be found at <https://github.com/SammyG7/Mac-AR>.

### 4 Notation

The structure of the MIS for modules comes from [Hoffman and Strooper \(1995\)](#), with the addition that template modules have been adapted from [Ghezzi et al. \(2003\)](#). The mathematical notation comes from Chapter 3 of [Hoffman and Strooper \(1995\)](#). For instance, the symbol  $:=$  is used for a multiple assignment statement and conditional rules follow the form  $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$ .

The following table summarizes the primitive data types used by Software Eng.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in $(-\infty, \infty)$
natural number	$\mathbb{N}$	a number without a fractional component in $[1, \infty)$
real	$\mathbb{R}$	any number in $(-\infty, \infty)$
string	string	a group of characters
Lobby	Lobby	represents a lobby object
LobbiesList	LobbiesList	represents a collection of lobbies
Array	Array.	represents an array containing elements of a particular data type
ChannelID	ChannelID	represents a voice/text channel object
IParticipant	IParticipant	represents a voice/text chat participant object
TextMessage	TextMessage	represents the text message data type present in Vivox library
UnityObject	UnityObject	represents unity objects
PlayerData	PlayerData	tuple representing a player by an ID and ready status

The specification of Mac-AR uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of

characters. Tuples contain a list of values, potentially of different types. In addition, Mac-AR uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

## 5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

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
Software Decision Module	Combination Puzzle Module
	Database/Network Manager Module
	Error Manager Module
	Documentation Module

Table 1: Module Hierarchy

## **6 MIS of Hardware Module**

### **6.1 Module**

This module is dependant on the user's phone, and the hardware interfaces are entirely dependant on the device used.

### **6.2 Uses**

None

### **6.3 Syntax**

#### **6.3.1 Exported Constants**

N/A

#### **6.3.2 Exported Access Programs**

N/A

### **6.4 Semantics**

#### **6.4.1 State Variables**

None

#### **6.4.2 Environment Variables**

None

#### **6.4.3 Assumptions**

Assuming the device the user is using has a working camera and gyroscope, and the required firmware to run the application

#### **6.4.4 Access Routine Semantics**

Implemented by operating system

#### **6.4.5 Local Functions**

Implemented by operating system

## 7 MIS of Game Room Module

### 7.1 Module

GameRoomModule

### 7.2 Uses

Database/Network Manager Module

Error Manager Module

### 7.3 Syntax

#### 7.3.1 Exported Constants

MIN\_ROOM\_CAPACITY = 2

MAX\_ROOM\_CAPACITY = 10

MINIMUM\_PASSWORD\_LENGTH = 8

MAXIMUM\_PASSWORD\_LENGT = 64

#### 7.3.2 Exported Access Programs

Name	In	Out	Exceptions
GameRoom	-	GameRoom	-
StartHost	N, string, string	-	-
ReadyPress	-	$\mathbb{B}$	-
RefreshLobbiesList	-	LobbiesList	-
JoinAsync	Lobby, string	$\mathbb{B}$	-
StartGame	-	-	-
SetLobby	Lobby	-	-
SetConnections	N	-	-
SetPassword	string	-	-

### 7.4 Semantics

#### 7.4.1 State Variables

inLobby :  $\mathbb{B}$

isReady :  $\mathbb{B}$

isJoining :  $\mathbb{B}$

lobbies : LobbiesList

### 7.4.2 Environment Variables

Device Screen : 2D Array of Pixels GameScene : Unity Scene

### 7.4.3 Assumptions

None

### 7.4.4 Access Routine Semantics

GameRoom():

- output: out := self
- exception: None

StartHost(connections, lobbyName, password):

- transition: GameScene := Lobby, Lobby.MaxConnections := connections, Lobby.LobbyName := lobbyName, Lobby.Password := password
- exception: None

ReadyPress():

- output: out := isReady
- exception: None

RefreshLobbiesList():

- transition:  $\forall lobby \in lobbies \implies Initialize(lobby)$
- output: out:=lobbies
- exception: None

JoinAsync(lobby,password):

- transition:  $join := (lobby.password = password \wedge lobby.availableSlots > 0)$
- output: out := isJoining
- exception: None

StartGame():

- transition: GameScene := Puzzle
- exception: None

SetLobby(lobby):

- transition: Lobby := lobby
- exception: None

SetConnections(connections):

- transition: Lobby.MaxConnections := connections
- exception: None

SetPassword(password):

- transition: Lobby.Password := password
- exception: None

#### **7.4.5 Local Functions**

None

## 8 MIS of Text Communication Module

This module uses the built in Unity library called Vivox to implement text chat.

### 8.1 Module

textCommunication

### 8.2 Uses

Vivox (Unity Library)

Database/Network Manager Module

Error Manager Module

### 8.3 Syntax

#### 8.3.1 Exported Constants

None

#### 8.3.2 Exported Access Programs

Name	In	Out	Exceptions
TextCommunication	-	TextCommunication	-
ClearMessageObjectPool	-	-	-
ClearOutTextField	-	-	-
SubmitTextToVivox	-	-	-
EnterKeyOnTextField	-	-	-
SendScrollRectToBottom	-	-	-
OnParticipantAdded	string, ChannelId, IParticipant	-	-
OnTextMessageLogReceivedEvent	string, IChannel- TextMessage	-	-

### 8.4 Semantics

#### 8.4.1 State Variables

\_vivoxVoiceManager : VivoxVoiceManager instance

\_textChannel : ChannelId

\_textChatScrollRect : ScrollRect

### 8.4.2 Environment Variables

Device Screen : 2D Array of Pixels

Device Keyboard : 2D Array of Characters

### 8.4.3 Assumptions

None

### 8.4.4 Access Routine Semantics

TextCommunication():

- output: `out := self`
- exception: None

ClearMessageObjectPool():

- transition: Clear chatbox
- exception: None

ClearOutTextField():

- transition: Clear user chat box input field
- exception: None

SubmitTextToVivox():

- transition: Send text message present in input field to `_textChannel`
- exception: None

EnterKeyOnTextField():

- transition: *`user.input == return => SubmitTextToVivox()`*
- exception: None

SendScrollRectToBottom():

- transition: Enable scrolling of `_textChatScrollRect`
- exception: None

OnParticipantAdded(username, channel, participant):



- transition:  $\_vivoxVoiceManager.channel.users = \_vivoxVoiceManager.channel.users + 1$
- exception: None

OnTextMessageLogReceivedEvent(sender, channelTextMessage):

- transition:  $channelTextMessage.FromSelf == 1 \Rightarrow sender.color = green \wedge SendScrollRectToBottom$   
 $channelTextMessage.FromSelf == 0 \Rightarrow sender.color = white \wedge SendScrollRectToBottom$
- exception: None

#### 8.4.5 Local Functions

None

#### 8.4.6 Local Variables

None

## 9 MIS of Voice Communication Module

This module uses the built in Unity library called Vivox to implement voice chat. Notably, certain core functions utilize Vivox and Unity frameworks, and as such their names, inputs, and outputs can't be changed.

### 9.1 Module

VoiceCommunication

### 9.2 Uses

Vivox (Unity Library)

Database/Network Manager Module

Error Manager Module

### 9.3 Syntax

#### 9.3.1 Exported Constants

None

#### 9.3.2 Exported Access Programs

Name	In	Out	Exceptions
VoiceCommunication	-	VoiceCommunication	-
Start	-	-	-
IsMicPermissionGranted	-	B	-
AskForPermissions	-		-
OnUserLoggedIn	-	-	SignInException
OnUserLoggedOut	-	-	-
VivoxToggle	-	-	-

### 9.4 Semantics

#### 9.4.1 State Variables

VoiceToggleIsOn : B

\_vvm : Vivox voice manager instance

### 9.4.2 Environment Variables

Device Microphone : Sound Input

Device Audio : Sound Output

Device Screen : 2D Array of Pixels

Permission : OS Permissions Object

### 9.4.3 Assumptions

None

### 9.4.4 Access Routine Semantics

VoiceCommunication():

- output: `out := self`
- exception: `None`

IsMicPermissionGranted():

- output: *Permission.HasUserAuthorizedPermission(Permission.Microphone)*
- exception: `None`

AskForPermissions():

- transition: Request access to user microphone
- exception: `None`

OnUserLoggedIn():

- transition: *\_vvm.LoginState == VivoxUnity.LoginState.LoggedIn => Join voice channel*
- exception: *\_vvm.LoginState! = VivoxUnity.LoginState.LoggedIn => SignInException*

OnUserLoggedOut():

- transition: Disconnect from Vivox voice manager
- exception: `None`

VivoxToggle():

- transition: *VoiceToggleIsOn => AudioInputDevice.Muted = false*  
*!VoiceToggleIsOn => AudioInputDevice.Muted = true*
- exception: `None`

#### 9.4.5 Local Functions

None

## 10 MIS of Multiplayer Puzzle Module

The base module that all other puzzle modules will inherit from. The module will handle all of the common behaviour shared between the puzzle implementations. This consists of the puzzle hint system, the puzzle skip system, and the coordination between users working on the same puzzle instance.

### 10.1 Module

MultiplayerPuzzleModule

### 10.2 Uses

Database/Network Manager Module  
Error Manager Module

### 10.3 Syntax

#### 10.3.1 Exported Constants

ISO\_ID = 0  
COMBINATION\_ID = 1  
WIRE\_ID = 2  
SIMON\_ID = 3  
MAZE\_ID = 4  
BATCH1 =  $\langle$ COMBINATION\_ID, WIRE\_ID, SIMON\_ID $\rangle$   
BATCH2 =  $\langle$ ISO\_ID $\rangle$   
BATCH3 =  $\langle$ MAZE\_ID $\rangle$   
PUZZLE\_BATCHES =  $\langle$ BATCH1, BATCH2, BATCH3 $\rangle$

#### 10.3.2 Exported Access Programs

Name	In	Out	Exceptions
SpawnPuzzleBatch	-	-	-
SkipPuzzle	-	-	-
GenerateHint	$\mathbb{Z}$	-	-
CompletePuzzle	$\mathbb{Z}$	-	-
CompletePuzzleBatch	-	-	-
InitializePuzzle	-	-	-
SetActive	$\mathbb{B}$	-	-

## 10.4 Semantics

### 10.4.1 State Variables

connectedClients : Array $\langle \mathbb{N} \rangle$   
puzzleInstances : Array $\langle \text{NetworkObject} \rangle$   
activePuzzleBatchIndex :  $\mathbb{Z}$   
activePuzzleIndex :  $\mathbb{Z}$   
seed :  $\mathbb{Z}$   
active :  $\mathbb{B}$   
puzzleId :  $\mathbb{Z}$   
hintList : Array $\langle \text{string} \rangle$

### 10.4.2 Environment Variables

Device Screen : 2D Array of Pixels

### 10.4.3 Assumptions

None

### 10.4.4 Access Routine Semantics

SpawnPuzzleBatch():

- transition:  $(\forall \text{puzzle} \in \text{PUZZLE\_BATCHES}[\text{activePuzzleBatchIndex}] : \text{SpawnPuzzle}(\text{puzzle}))$
- transition: activePuzzleIndex = 0
- exception: None

SkipPuzzle():

- transition: CompletePuzzle(activePuzzleIndex)
- exception: None

GenerateHint(hintId):

- transition: Display hintList[hintId] on device screen
- exception: None

CompletePuzzle(puzzleId):

- transition:  $(\text{puzzleId} == \text{activePuzzleId}) \implies \text{puzzleInstances}[\text{puzzleId}].\text{SetActive}(\text{false})$

- transition:  $(\text{puzzleId} == \text{activePuzzleId} \wedge \text{activePuzzleId} + 1 < \text{PUZZLE\_BATCHES}[\text{activePuzzleBatchIndex}].\text{length}) \implies (\text{puzzleInstances}[\text{puzzleId} + 1].\text{SetActive}(\text{true}) \wedge \text{activePuzzleId} += 1)$
- transition:  $(\text{puzzleId} == \text{activePuzzleId} \wedge \text{activePuzzleId} + 1 \geq \text{PUZZLE\_BATCHES}[\text{activePuzzleBatchIndex}].\text{length}) \implies \text{CompletePuzzleBatch}()$
- exception: None

CompletePuzzleBatch():

- transition:  $\forall \text{puzzle} \in \text{puzzleInstances} : \text{puzzle.Despawn}()$
- transition:  $\text{activePuzzleBatchIndex} += 1$
- transition:  $\text{SpawnPuzzleBatch}()$
- exception: None

InitializePuzzle():

- transition: N/A (This function is overridden by each puzzle instance and the exact implementation varies)
- exception: None

SetActive(status):

- transition:  $\text{active} = \text{status}$
- exception: None

#### 10.4.5 Local Functions

SpawnPuzzle(puzzleIndex):

- transition:  $\text{puzzles}[\text{puzzleIndex}].\text{Instantiate}()$
- exception: None

#### 10.4.6 Local Variables

puzzles :  $\text{Array}\langle \text{NetworkObject} \rangle$

## 11 MIS of Simon Says Puzzle Module

The Simon Says Puzzle Module implements functionality for the Simon Says puzzle that is present in the application. This puzzle involves two users. User 1 has a 4 buttons of different colours (red, blue, green, yellow) in their game environment and User 2 has a cube in their environment that flashes with different colours. User 2 must remember the pattern of colours that was shown and communicate with User 1 to let them know the order to press the coloured buttons in.

### 11.1 Module

SimonSaysPuzzle

### 11.2 Uses

Multiplayer Puzzle Module

### 11.3 Syntax

#### 11.3.1 Exported Constants

None

#### 11.3.2 Exported Access Programs

Name	In	Out	Exceptions
SimonSaysPuzzle	-	SimonSaysPuzzle	-
GenerateColourSequence	-	Array of N	-
DisableInteractionWithButtons		-	-
EnableInteractionWithButtons		-	-
TrackUserInput	-	-	-
CheckIfSuccess	Array of N	B	-
IncrementLevel	-	-	-
ResetLevel	-	-	-
UpdateCubeServerRpc	Colour	-	-
UpdateTextServerRpc	string	-	-



## 11.4 Semantics

### 11.4.1 State Variables

level := 1 {1 ≤ N < 4}  
success : B  
colourSequenceArray : Array of N  
buttonsArray := {0, 1, 2, 3}

### 11.4.2 Environment Variables

Device Screen : 2D Array of Pixels

### 11.4.3 Assumptions

None

### 11.4.4 Access Routine Semantics

SimonSaysPuzzle():

- output: out := self
- exception: None

GenerateColourSequence():

- transition:  $\forall i : level \leq i < level + 1 :$   
 $colourSequenceArray.append(randInt(0, 4))$
- output: out := colourSequenceArray
- exception: None

DisableInteractionWithButtons():

- transition:  $\forall i : 0 \leq i < 4 :$   
 $buttonsArray[i].interactable = False$
- exception: None

EnableInteractionWithButtons():

- transition:  $\forall i : 0 \leq i < 4 :$   
 $buttonsArray[i].interactable = True$
- exception: None

TrackUserInput():

- transition: Detect and store user input order of buttons in game environment
- exception: None

CheckIfSuccess(sequence):

- transition:  $success == False \Rightarrow ResetLevel()$
- output:  $success := \forall i : 0 \leq i < length(sequence) : sequence[i] == userInput[i] \Rightarrow True \mid else \Rightarrow False$
- exception: None

UpdateCubeServerRpc(colour):

- transition: Cube flashes colour present in input for all users besides player 1
- exception: None

UpdateTextServerRpc(text):

- transition: Simon Says level text updated amongst all users when correct or incorrect sequence inputted
- exception: None

IncreaseLevel():

- transition:  $level := level + 1$
- exception: None

ResetLevel():

- transition:  $level := 1$
- exception: None

#### 11.4.5 Local Functions

None

#### 11.4.6 Local Variables

None

## 12 MIS of Isometric Puzzle Module

The Isometric Puzzle Module implements functionality for the Isometric puzzle that is present in the application. This puzzle involves 2-8 players. Each player has an equally distributed amount of letters. e.g with 2 players, each player will receive 4 letters. The players will look at the arrangement of cubes from different angles, and it will appear to be a letter from one angle, and a number from the other angle. The players must put together all their letters at their respective index indicated by the number they see to complete the final word.

### 12.1 Module

IsometricPuzzle

### 12.2 Uses

Multiplayer Puzzle Module

### 12.3 Syntax

#### 12.3.1 Exported Constants

SOLUTION = "TWILIGHT"

CUBE\_WIDTH=5

CUBE\_LENGTH = 5

CUBE\_HEIGHT = 5

#### 12.3.2 Exported Access Programs

Name	In	Out	Exceptions
IsometricPuzzle	-	IsometricPuzzle	-
SetCubes	string	{B,B,B}	-
SendPuzzleDataServerRpc	ℕ	ℕ	-
UpdateTextServerRpc	string	string	-
SetIsometricPuzzlesServerRpc	string	string	-

### 12.4 Semantics

#### 12.4.1 State Variables

puzzleData.connectedClients : Tuple⟨ℕ, Array⟨ℕ⟩⟩

### 12.4.2 Environment Variables

Device screen : 2D Array of Pixels

This module has external interaction with an input field called solutionField, which has a property called "text" which allows characters to be pulled from it, as well as printed to the screen. There are also two buttons that call PrevIsometric, and NextIsometric.

### 12.4.3 Assumptions

None

### 12.4.4 Access Routine Semantics

IsometricPuzzle():

- output: out := self
- exception: None

SetCubes(key):

- output:  $\forall x, y, z \in key \Rightarrow activeGrid[x, y, z] = true | activeGrid[x, y, z] = false$
- exception: None

SendPuzzleDataServerRpc(p[]):

- output: puzzleData.connectedClients := p
- exception: None

UpdateTextServerRpc(text):

- transition: If text=solution, transition to next puzzle
- output: solutionField.text=text to all users in the lobby
- exception: None

SetIsometricPuzzlesServerRpc(word):

- output: every player's *\_cubeNames* is updated to have a random assortment of letter-number pairs from word, with each player having unique letter-number pairs.
- exception: None

### 12.4.5 Local Functions

NextIsometric(): Change the viewable letter-number pair to the next one in the list

- output:  $\text{cubeIndex} = \text{cubeIndex} + 1$   
SetCubes(*\_cubeNames*[*cubeIndex*])

- exception: None

PrevIsometric(): Change the viewable letter-number pair to the previous one in the list

- output:  $\text{cubeIndex} = \text{cubeIndex} - 1$   
SetCubes(*\_cubeNames*[*cubeIndex*])

- exception: None

solutionFieldChanged(): Called when user hits enter on the inputfield on the screen, updates the inputfield on each user's screen

- output: UpdateTextServerRpc(solutionField.text)
- exception: None

### 12.4.6 Local Variables

activeGrid : Array⟨B⟩

\_cubeNames : Array⟨string⟩

cubeIndex :  $\mathbb{I}$

## 13 MIS of Wires Puzzle Module

Puzzle module to be interacted with by two or more users. The module will generate a set of interactable wires for one user, and provide information on the correct order of wires to other users.

### 13.1 Module

WirePuzzle Module

### 13.2 Uses

Multiplayer Puzzle Module

### 13.3 Syntax

#### 13.3.1 Exported Constants

None

#### 13.3.2 Exported Access Programs

Name	In	Out	Exceptions
WiresPuzzle	-	WiresPuzzle	-
RandomList	$\mathbb{Z}$	$\text{Array}\langle\mathbb{N}\rangle$	-
SetActive	$\mathbb{B}$	-	-
UpdateSequence	$\mathbb{Z}, \mathbb{Z}$	$\mathbb{B}$	-

### 13.4 Semantics

#### 13.4.1 State Variables

correctSequence :  $\text{Array}\langle\text{Array}\langle\mathbb{Z}\rangle\rangle$

currentSequence :  $\text{Array}\langle\mathbb{Z}\rangle$

lights :  $\text{Array}\langle\text{GameObject}\rangle$

#### 13.4.2 Environment Variables

Device Screen : 2D Array of Pixels

#### 13.4.3 Assumptions

None

#### 13.4.4 Access Routine Semantics

WiresPuzzle():

- output: out := self
- exception: None

RandomList(seed):

- output: List containing the elements 0, 1, 2, 3 in a random order based on the seed
- exception: None

SetActive(status):

- transition: active = status
- transition:  $(\text{status} == \text{true} \wedge \text{currentSequence} \neq \text{correctSequence}) \implies (\forall \text{light} \in \text{lights} : \text{light.colour} = \text{red})$
- exception: None

UpdateSequence(wire, anchor):

- transition:  $\text{currentSequence}[\text{wire}] = \text{anchor}$
- transition:  $(\text{currentSequence} == \text{correctSequence}) \implies (\forall \text{light} \in \text{lights} : \text{light.colour} = \text{green})$
- output:  $\text{currentSequence} == \text{correctSequence}$
- exception: None

#### 13.4.5 Local Functions

None

#### 13.4.6 Local Variables

None

## 14 MIS of Maze Puzzle Module

### 14.1 Module

The maze puzzle module is a module dedicated to the maze puzzle, where one user will be in control of rotating a maze to get a ball to roll from the start to the end, while the other users will guide the user in control to move the ball through the maze.

### 14.2 Uses

Multiplayer Puzzle Module

### 14.3 Syntax

#### 14.3.1 Exported Constants

MAZE\_WIDTH = 10

MAZE\_LENGTH = 10

#### 14.3.2 Exported Access Programs

Name	In	Out	Exceptions
MazePuzzle	-	MazePuzzle	-
GenerateMaze	$\mathbb{N}$	$\mathbb{Z}$	-
RotateMaze	$\mathbb{R}, \mathbb{R}, \mathbb{R}$	-	-
BallHitsGoal	-	-	-
SendPuzzleDataServerRpc	$\mathbb{I}$	$\mathbb{I}$	-
GenerateMazeServerRpc	$\text{Array}\langle\mathbb{I}\rangle$	$\text{Array}\langle\mathbb{I}\rangle$	-
ResetRotationPress	-	-	-

### 14.4 Semantics

#### 14.4.1 State Variables

BallPosition :  $\mathbb{R}, \mathbb{R}, \mathbb{R}$

MazeRotation :  $\mathbb{R}, \mathbb{R}, \mathbb{R}$

mazeLayout :  $\text{Array}\langle\mathbb{I}, \mathbb{I}\rangle$

puzzleData.connectedClients :  $\text{Tuple}\langle\mathbb{Z}, \text{Array}\langle\mathbb{Z}\rangle\rangle$

#### 14.4.2 Environment Variables

Gyroscope : Rotation Detection



A button is on the screen that calls ResetRotationPress

### 14.4.3 Assumptions

#### 14.4.4 Access Routine Semantics

MazePuzzle():

- output: out := self
- exception: None

GenerateMaze(NumberOfPlayer):

- transition: SpawnMaze(), BallPosition := (0,0,0)
- output: ControllingPlayer := randInt(0,NumberOfPlayers-1)
- exception: None

RotateMaze(Pitch, Yaw, Roll):

- transition: MazeRotation := Pitch,Yaw,Roll
- exception: None

BallHitsGoal():

- transition: CompletePuzzle()
- exception: None

SendPuzzleDataServerRpc(p[]):

- output: puzzleData.connectedClients := p
- exception: None

GenerateMazeServerRpc(Array(I,I)mazeLayouts):

- output: mazeLayouts of all connected clients becomes mazeLayouts
- exception: None

ResetRotationPress():

- output: MazeRotation = 0,0,0
- exception: None

### 14.4.5 Local Functions

BallMovement: BallPosition  $\times$  MazeRotation  $\implies$  BallPosition  $\equiv$  Based on current Ball-Position and Maze Rotation, the current Position of the Ball changes to reflect the effect of gravity on the ball as it rolls downwards.

To1DArray(input): Takes in a 2D int array and converts it to a 1D array

- output: result := input to 1d array
- exception: None

convertLayoutToGrid(mazeLayouts): Takes in an array of integers between 0 and 3 and converts them to a maze grid.

- transition: `_mazeGrid := Make2DArray(mazeLayouts)`
- exception: None

Make2DArray(input,height,width): Takes in a 1D array and outputs a 2D array.

- output: result := input to 2d array with height and width
- exception: None

GetNextUnvisitedCell(MazeCell currentcell): Helper function for generating the maze, gets an unvisited cell based off adjacent cells

- output: cell := random(currentcell.adjacent)
- exception: None

ClearWalls(MazeCell previousCell, MazeCell currentCell): Removes two walls of the maze, based on relative location of the two cells.

- transition: `if(previousCell.x < currentCell.x) then previousCell.ClearRightWall() currentCell.ClearLeftWall()`  
`if (previousCell.transform.position.x > currentCell.transform.position.x) then previousCell.ClearLeftWall() currentCell.ClearRightWall()`  
`if (previousCell.transform.position.z < currentCell.transform.position.z) then previousCell.ClearFrontWall() currentCell.ClearRearWall()`  
`if (previousCell.transform.position.z > currentCell.transform.position.z) then previousCell.ClearRearWall() currentCell.ClearFrontWall()`
- exception: None

### 14.4.6 Local Variables

Goal Location :  $\mathbb{R}, \mathbb{R}, \mathbb{R}$

ControllingPlayerIndex :  $0 < \mathbb{N} < 10$

## 15 MIS of Combination Puzzle Module

The Combination Puzzle Module implements functionality for the combination discovery puzzle that is present in the application. This puzzle involves two to 4 users. An instruction card is generated for each user that only they can see, giving a subset of the instructions to solve a number combination. All users must communicate to combine their instructions to solve the combination.

### 15.1 Module

CombinationPuzzle

### 15.2 Uses

Multiplayer Puzzle Module

### 15.3 Syntax

#### 15.3.1 Exported Constants

COMBO\_LENGTH=4

#### 15.3.2 Exported Access Programs

Name	In	Out	Exceptions
CombinationPuzzle	-	CombinationPuzzle	-
GenerateCombo	-	Array of strings	-
ConvertButtonPress	-	char	-
CheckEntry	char, Array of strings	B	-
ColourBlink	-	-	-
Restart	-	-	-

### 15.4 Semantics

#### 15.4.1 State Variables

comboSet : Array of strings

#### 15.4.2 Environment Variables

Device Screen : 2D Array of Pixels

### 15.4.3 Assumptions

None

### 15.4.4 Access Routine Semantics

CombinationPuzzle():

- output:  $out := self$
- exception: None

GenerateCombo():

- transition:  $comboSet := randCombo(comboArray)$
- output:  $out := (\forall i : 0 \leq i \leq numPlayers : player[i].instructions = comboSet[i + 1])$ , if  $numPlayers < 4$ , loop back through remaining players assigning remaining instructions
- exception: None

ConvertButtonPress():

- transition: None
- output:  $userInput :=$  When button on keypad pressed, output character of equivalent number from 0-9
- exception: None

CheckEntry(userInput, comboSet):

- transition:  $(userInput == comboSet[0][currentDigit] \wedge currentDigit < 4) \implies currentDigit++ = 1$
- output:  $(userInput == comboSet[0][currentDigit]) \wedge (currentDigit == 4)$
- exception: None

ColourBlink():

- transition:  $success == True \implies \text{screen flash green} \mid else \implies \text{screen flash red}, Restart()$
- output: None
- exception: None

Restart():

- transition:  $currentDigit = 0, GenerateCombo()$
- output: None
- exception: None

### 15.4.5 Local Functions

randCombo(comboArray): Select random combination-instruction set from comboArray

- output: combo := newCombo
- exception: None

### 15.4.6 Local Variables

currentDigit := 0  $\{0 \leq \mathbb{N} < 4\}$

success :  $\mathbb{B}$

comboArray := Array of arrays of strings of length 5. The first element of these subarrays is the combination of length COMBO\_LENGTH. The next 4 elements are the 4 instructions associated with this combo.

## 16 MIS of Database/Network Manager Module

### 16.1 Module

This module manages the bulk of communication between the server and the clients. Notably, its core functions are derived from built in Unity frameworks, and as such names, inputs, and outputs are not able to be modified.

### 16.2 Uses

None

### 16.3 Syntax

#### 16.3.1 Exported Constants

None

#### 16.3.2 Exported Access Programs

Name	In	Out	Exceptions
OnNetworkSpawn	-	-	-
OnNetworkDespawn	-	-	-
HandleClientConnected	$\mathbb{R}$	-	-
HandleClientDisconnected	$\mathbb{R}$	-	-

### 16.4 Semantics

#### 16.4.1 State Variables

players : NetworkList< *PlayerData* >

#### 16.4.2 Environment Variables

None

#### 16.4.3 Assumptions

None

#### 16.4.4 Access Routine Semantics

OnNetworkSpawn():

- transition: On initial connection to the network, update *players* with the host information and all the clients who have joined the network.
- exception: None

OnNetworkDespawn():

- transition: When the network is closed disconnect all users from the network and remove all users from *players*.
- exception: None

HandleClientConnected(clientId):

- transition: `players.Add(new PlayerData(clientId))`
- exception: None

HandleClientDisconnected(clientId):

- transition: if  $\exists i : 0 \leq i < \text{length}(\text{players}) :$   
 $(\text{players}[i].\text{clientId} == \text{clientId}) \Rightarrow \text{players.RemoveAt}(i)$
- exception: None

#### 16.4.5 Local Functions

None

## 17 MIS of Error Manager Module

### 17.1 Module

This module catches and manages all the error that pop up to give the user a more descriptive and readable version of the error that they encountered. Although the number of errors handled is few, this module is critical to ensure that the networking aspects of the application can continue to work in real time.

### 17.2 Uses

None

### 17.3 Syntax

#### 17.3.1 Exported Constants

#### 17.3.2 Exported Access Programs

Name	In	Out	Exceptions
Get	-	LogHandler	-
LogFormat	N, UnityObject, string, object	LogHandler	-
LogException	Exception, object	-	-
SpawnErrorPopup	string	-	-

### 17.4 Semantics

#### 17.4.1 State Variables

ShowErrorMessage :  $\mathbb{B}$

#### 17.4.2 Environment Variables

None

#### 17.4.3 Assumptions

#### 17.4.4 Access Routine Semantics

Get():

- output: out:= LogHandlerInstance
- exception: None



LogFormat(logtype,context,format,args[]):

- output: out := m\_DefaultLogHandler.LogFormat(logtype, context, format, args)
- exception: None

LogException(exception, context):

- output: out:= m\_DefaultLogHandler.LogException(exception, context)
- exception: None

SpawnErrorPopup(errorMessage):

- transition: ShowErrorMessage := true , gameObject.SetActive(true)
- exception: None

ClearPopup():

- transition: gameObject.SetActive(false)
- exception: None

#### **17.4.5 Local Functions**

None

## 18 MIS of Documentation Module

Module used to map requirements that are related to user documentation.

### 18.1 Module

### 18.2 Uses

None

### 18.3 Syntax

#### 18.3.1 Exported Constants

N/A

#### 18.3.2 Exported Access Programs

Name	In	Out	Exceptions
-	-	-	-

### 18.4 Semantics

#### 18.4.1 State Variables

None

#### 18.4.2 Environment Variables

None

#### 18.4.3 Assumptions

None

#### 18.4.4 Access Routine Semantics

None

#### 18.4.5 Local Functions

None

## References

- Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.
- Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL <http://citeseer.ist.psu.edu/428727.html>.

## 19 Appendix

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design. Please answer the following questions:

1. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO\_ProbSolutions)

One of the main limitations of the project is the time constraint, as well as balancing the project with other courses members of the team are taking. As a result, certain features such as UI elements may be less aesthetic/nice looking due to having less time to work on them. Given unlimited resources, the following updates to the project could be made.

- Improved UI elements: Given unlimited time and resources we could implement custom assets that better fit our theme and would allow us to further personalize the game. We could also add more advanced rendering and animation effects such as are seen in the main menu screen. This would allow us to make the experience more immersive and really take advantage of the augmented reality space.
- Creation of additional puzzle modules/types: Given more time we could create more technically challenging puzzles that take better advantage of the AR space. One such puzzle that was considered was hiding AR elements in an image of a user's real-world space and showing it to another user. They would then have to communicate to the initial user what was different in the image than in their actual space. However, seamlessly integrating AR elements into an image of real space would take significantly more time to learn the technical skills necessary than what was available.
- Modification of current puzzle modules to allow them to work with several different amounts of people (ex. 2-player, 3-player, 4-player, etc.): Current puzzles are somewhat limited in their design such that all players may not be necessary for each puzzle. Given more time and resources the design of the puzzles could've been altered so that what the puzzle showed to each person was different. Alternatively the overall structure could've been modified so that different people could work on different puzzles simultaneously.

Another limitation of the project is the lack of choice regarding frameworks that can be used to implement AR elements. This project requires an AR game environment and there are only a few frameworks that can be used to implement AR elements, such as Unity. As a result, our team chose to use Unity to implement the application.

2. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO\_Explores)

As mentioned previously, there are not many frameworks that allow for the implementation of AR elements. At the beginning of the project, our team debated about whether to use Unreal Engine or Unity. The main benefit of Unreal Engine is the use of blueprints, a tool that simplifies scripting for beginners. However, we decided on Unity due to the fact that several group members had much more experience with Unity. Additionally, there are many benefits of using unity. First of all, there are many AR libraries that work hand in hand with Unity which is extremely useful for our project. Additionally, Unity has an asset store where we can get assets to be used in our game environment which is also extremely useful. There are some negatives such as it being extremely difficult to do automated testing with Unity but Unreal Engine does not solve this issue and there are no other viable main-stream 3D engines. Moving on, another design decision that we chose was using Vivox framework for the implementation of voice and text communication. The benefits of this is that Unity supports the use of Vivox and so the implementation of voice and text communication was not too difficult. The tradeoff, however, is that there is less freedom for the actual implementation of these communication features, as Vivox has to be set up in a specific way and only has certain functionality. There are very few publicly available alternatives for implementing voice chat in Unity. Some options are the [Stream Chat](#) and [Agora.io](#) but while these tools may offer slightly different services than Vivox or slightly more flexibility they are not as well documented and Vivox provides all the essential services necessary for our game.