

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

January 31, 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

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

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	ii
3	Introduction	1
4	Notation	1
5	Module Decomposition	2
6	MIS of Hardware Module	3
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
7	MIS of Game Room Module	4
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
8	MIS of Text Communication Module	7
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	7
8.4.3	Assumptions	8
8.4.4	Access Routine Semantics	8
8.4.5	Local Functions	9
8.4.6	Local Variables	9
9	MIS of Voice Communication Module	10
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	10
9.4.3	Assumptions	11
9.4.4	Access Routine Semantics	11
9.4.5	Local Functions	11
10	MIS of Game Save Module	12
10.1	Module	12
10.2	Uses	12
10.3	Syntax	12
10.3.1	Exported Constants	12
10.3.2	Exported Access Programs	12
10.4	Semantics	12
10.4.1	State Variables	12
10.4.2	Environment Variables	12
10.4.3	Assumptions	12
10.4.4	Access Routine Semantics	13
10.4.5	Local Functions	13
11	MIS of Multiplayer Puzzle Module	14
11.1	Module	14
11.2	Uses	14
11.3	Syntax	14
11.3.1	Exported Constants	14
11.3.2	Exported Access Programs	14
11.4	Semantics	14
11.4.1	State Variables	14
11.4.2	Environment Variables	14

11.4.3	Assumptions	14
11.4.4	Access Routine Semantics	15
11.4.5	Local Functions	15
11.4.6	Local Variables	15
12	MIS of Simon Says Puzzle Module	16
12.1	Module	16
12.2	Uses	16
12.3	Syntax	16
12.3.1	Exported Constants	16
12.3.2	Exported Access Programs	16
12.4	Semantics	16
12.4.1	State Variables	16
12.4.2	Environment Variables	17
12.4.3	Assumptions	17
12.4.4	Access Routine Semantics	17
12.4.5	Local Functions	18
12.4.6	Local Variables	18
13	MIS of Code Puzzle Module	19
13.1	Module	19
13.2	Uses	19
13.3	Syntax	19
13.3.1	Exported Constants	19
13.3.2	Exported Access Programs	19
13.4	Semantics	19
13.4.1	State Variables	19
13.4.2	Environment Variables	20
13.4.3	Assumptions	20
13.4.4	Access Routine Semantics	20
13.4.5	Local Functions	21
13.4.6	Local Variables	21
14	MIS of Wires Puzzle Module	22
14.1	Module	22
14.2	Uses	22
14.3	Syntax	22
14.3.1	Exported Constants	22
14.3.2	Exported Access Programs	22
14.4	Semantics	22
14.4.1	State Variables	22
14.4.2	Environment Variables	22
14.4.3	Assumptions	22

14.4.4	Access Routine Semantics	23
14.4.5	Local Functions	23
14.4.6	Local Variables	23
15	MIS of Maze Puzzle Module	24
15.1	Module	24
15.2	Uses	24
15.3	Syntax	24
15.3.1	Exported Constants	24
15.3.2	Exported Access Programs	24
15.4	Semantics	24
15.4.1	State Variables	24
15.4.2	Environment Variables	24
15.4.3	Assumptions	24
15.4.4	Access Routine Semantics	24
15.4.5	Local Functions	25
15.4.6	Local Variables	25
16	MIS of Math Puzzle Module	26
16.1	Module	26
16.2	Uses	26
16.3	Syntax	26
16.3.1	Exported Constants	26
16.3.2	Exported Access Programs	26
16.4	Semantics	26
16.4.1	State Variables	26
16.4.2	Environment Variables	26
16.4.3	Assumptions	26
16.4.4	Access Routine Semantics	27
16.4.5	Local Functions	27
16.4.6	Local Variables	27
17	MIS of Bomb Puzzle Module	28
17.1	Module	28
17.2	Uses	28
17.3	Syntax	28
17.3.1	Exported Constants	28
17.3.2	Exported Access Programs	28
17.4	Semantics	28
17.4.1	State Variables	28
17.4.2	Environment Variables	28
17.4.3	Assumptions	29
17.4.4	Access Routine Semantics	29

17.4.5	Local Functions	30
17.4.6	Local Variables	30
18	MIS of Database/Network Manager Module	31
18.1	Module	31
18.2	Uses	31
18.3	Syntax	31
18.3.1	Exported Constants	31
18.3.2	Exported Access Programs	31
18.4	Semantics	31
18.4.1	State Variables	31
18.4.2	Environment Variables	31
18.4.3	Assumptions	31
18.4.4	Access Routine Semantics	31
18.4.5	Local Functions	32
19	MIS of Error Manager Module	33
19.1	Module	33
19.2	Uses	33
19.3	Syntax	33
19.3.1	Exported Constants	33
19.3.2	Exported Access Programs	33
19.4	Semantics	33
19.4.1	State Variables	33
19.4.2	Environment Variables	33
19.4.3	Assumptions	33
19.4.4	Access Routine Semantics	33
19.4.5	Local Functions	34
20	MIS of Documentation Module	35
20.1	Module	35
20.2	Uses	35
20.3	Syntax	35
20.3.1	Exported Constants	35
20.3.2	Exported Access Programs	35
20.4	Semantics	35
20.4.1	State Variables	35
20.4.2	Environment Variables	35
20.4.3	Assumptions	35
20.4.4	Access Routine Semantics	35
20.4.5	Local Functions	35
21	Appendix	37

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 of ...	Array of ...	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
	Game Save Module
	Multiplayer Puzzle Module
	Simon Says Puzzle Module
	Code Puzzle Module
	Wires Puzzle Module
	Maze Puzzle Module
	Math Puzzle Module
	Bomb Puzzle Module
Software Decision 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

7.3 Syntax

7.3.1 Exported Constants

Min Room Capacity = 2

Max Room Capacity = 10

Minimum Password Length = 8

Maximum Password length = 64

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
StartHost	\mathbb{N} , string, string	-	-
Initialize	LobbiesList, Lobby	-	-
readyPress	-	\mathbb{B}	-
RefreshLobbiesList	-	LobbiesList	-
JoinAsync	Lobby, string	\mathbb{B}	-
StartGame	-	-	-
setLobby	Lobby	-	-
setConnections	\mathbb{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

This module has external interaction with the phone screen the user is using, there will be buttons to be pressed to call some of these functions.

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

readyPress():

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

JoinAsync(lobby,password):

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

RefreshLobbiesList():

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

StartGame():

- transition: $GameScene := Puzzle$
- exception: None

StartHost(connections, lobbyName, password):

- transition: $GameScene := Lobby, Lobby.MaxConnections := connections, Lobby.LobbyName := lobbyName, Lobby.Password := password$
- 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

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

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

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

Device screen

Device keyboard

There are UI components such as a chat box and buttons that will be present on the user screen when some of these access routines are called.

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

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: $user.input == return \Rightarrow SubmitTextToVivox()$
- 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

8.4.6 Local Variables

```
_vivoxVoiceManager : VivoxVoiceManager instance  
_textChannel : ChannelId  
_textChatScrollRect : ScrollRect
```

9 MIS of Voice Communication Module

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

9.1 Module

voiceCommunication

9.2 Uses

Vivox (Unity Library)

Database/Network Manager Module

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

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

9.4 Semantics

9.4.1 State Variables

VoiceToggleIsOn : B

9.4.2 Environment Variables

Device microphone

Device audio

Device screen

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

Start():

- transition: $_vvm := \text{Vivox voice manager instance}$
- exception: None

isMicPermissionGranted():

- transition: $isGranted = \text{Permission.HasUserAuthorizedPermission}(\text{Permission.Microphone})$
- output: $out := isGranted$
- exception: None

AskForPermissions():

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

OnUserLoggedIn():

- transition: $_vvm.LoginState == \text{VivoxUnity.LoginState.LoggedIn} \Rightarrow \text{Join voice channel}$
- exception: $_vvm.LoginState! = \text{VivoxUnity.LoginState.LoggedIn} \Rightarrow \text{SignInException}$

OnUserLoggedOut():

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

VivoxToggle():

- transition: $\text{VoiceToggleIsOn} \Rightarrow \text{AudioInputDevice.Muted} = \text{false}$
 $!\text{VoiceToggleIsOn} \Rightarrow \text{AudioInputDevice.Muted} = \text{true}$
- exception: None

9.4.5 Local Functions

None

10 MIS of Game Save Module

The Game Save Module implements functionality for saving the current puzzle state of the game, load the puzzle state of an existing game or delete a save entry.

10.1 Module

GameSaveModule

10.2 Uses

Database/Network Manager Module

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
SaveGame	\mathbb{Z} , set of \mathbb{B}	-	-
LoadGame	\mathbb{Z}	set of \mathbb{B}	-
DeleteSave	\mathbb{Z}	-	-

10.4 Semantics

10.4.1 State Variables

saveTable: Dictionary of arrays of \mathbb{B} saveId \rightarrow puzzleStates

10.4.2 Environment Variables

This module has external interaction with the phone screen the user is using, there will be buttons to be pressed to call some of these functions.

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

SaveGame(saveId, puzzleStates):

- transition: saveTable.Add(saveId, puzzleStates)
- exception: None

LoadGame(saveId):

- output: out := puzzleStates
- exception: None

DeleteSave(saveId):

- transition: saveTable.Delete(saveId)
- exception: None

10.4.5 Local Functions

None

11 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.

11.1 Module

MultiplayerPuzzleModule

11.2 Uses

Database/Network Manager Module
Error Manager Module

11.3 Syntax

11.3.1 Exported Constants

ALLOWED_SKIPS = 3
ALLOWED_HINTS = 3

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
GeneratePuzzle	-	-	-
SkipPuzzle	\mathbb{Z}	\mathbb{B}	-
GenerateHint	\mathbb{Z}	string	-
CompletePuzzle	\mathbb{Z}	-	-

11.4 Semantics

11.4.1 State Variables

PuzzlePlayers : $\text{Tuple}\langle\mathbb{Z}, \text{Array}\langle\mathbb{Z}\rangle\rangle$

11.4.2 Environment Variables

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

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

GeneratePuzzle():

- transition: Assign all available players to a puzzle in the PuzzlePlayers tuple

SkipPuzzle(playerId):

- transition: $\forall \text{ player} \in \text{PuzzlePlayers}[\text{getCurrentPuzzle}(\text{playerId})] : \text{PuzzlePlayers}[\text{newPuzzle}].\text{append}(\text{player}) \wedge \text{numSkips}++$
- output: $\text{numSkips} < \text{ALLOWED SKIPS}$

GenerateHint(playerId):

- transition: $\forall \text{ player} \in \text{PuzzlePlayers}[\text{getCurrentPuzzle}(\text{playerId})] : \text{PuzzlePlayers}[\text{newPuzzle}].\text{appnd}(\text{player}) \wedge \text{numHints}++$
- output: $\text{numHints} < \text{ALLOWED HINTS} \Rightarrow \text{hintString}$

CompletePuzzle(playerId):

- transition: $\forall \text{ player} \in \text{PuzzlePlayers}[\text{getCurrentPuzzle}(\text{playerId})] : \text{PuzzlePlayers}[\text{newPuzzle}].\text{append}(\text{player})$

11.4.5 Local Functions

None

11.4.6 Local Variables

None

12 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 area 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.

12.1 Module

SimonSaysPuzzle

12.2 Uses

Multiplayer Puzzle Module

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
GenerateColourSequence	-	Array of \mathbb{N}	-
DisableInteractionWithButtons		-	-
EnableInteractionWithButtons		-	-
ButtonClickOrder	-	-	-
CheckIfSuccess	Array of \mathbb{N}	B	-
ColourBlink	-	-	-
IncreaseLevel	-	-	-
Reset	-	-	-

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

Device Screen

This module has external interaction with the phone screen, as there will be buttons in the game environment that the user presses which will be mapped to functions

12.4.3 Assumptions

None

12.4.4 Access Routine Semantics

GenerateColourSequence():

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

DisableInteractionWithButtons():

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

EnableInteractionWithButtons():

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

ButtonClickOrder():

- 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

ColourBlink():

- transition: $success == True \Rightarrow \text{screen flash green} \mid else \Rightarrow \text{screen flash red}$
- exception: None

IncreaseLevel():

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

ResetLevel():

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

12.4.5 Local Functions

None

12.4.6 Local Variables

level := 1 {1 ≤ N < 10}

success : \mathbb{B}

colourSequenceArray : Array of \mathbb{N}

buttonsArray := {0, 1, 2, 3}

colourDictionary := {Red → 0, Blue → 1, Green → 2, Yellow → 3}

13 MIS of Code Puzzle Module

The Decipher Code Puzzle Module implements functionality for the Decipher Code puzzle that is present in the application. This puzzle involves two users. User 1 has a group of symbols located next to each other in a row. User 2 has a cipher that shows the correlation between English letters and the symbols. User 1 must describe to User 2 the symbols they are seeing in order to crack the cipher. Once the symbols have been mapped to English letters, User 1 must enter the code to solve the puzzle.

13.1 Module

CodePuzzle

13.2 Uses

Multiplayer Puzzle Module

13.3 Syntax

13.3.1 Exported Constants

CodeLength = 6

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
GenerateCode	-	string	-
MapLettersToSymbols	string	string	-
StoreUserInput	-	-	-
CheckIfSuccess	string	B	-
ColourBlink	-	-	-
ResetInput	-	-	-

13.4 Semantics

13.4.1 State Variables

SymbolsArray := {*symbol0*, *symbol1*, *symbol2*, ..., *symbol25*}

13.4.2 Environment Variables

Device screen

This module has external interaction with the phone screen, as there will be buttons in the game environment that the user presses which will be mapped to functions

13.4.3 Assumptions

None

13.4.4 Access Routine Semantics

GenerateCode():

- output: $code := \text{randString}(\text{CodeLength})$
- exception: None

MapLettersToSymbols(code):

- output: $\text{symbolCode} := \{0 \leq i < \text{length}(\text{code}) : \text{code}[i] \rightarrow \text{SymbolsArray}[\text{lettersToNumberDictionary}[\text{code}[i]]]\}$
- exception: None

StoreUserInput():

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

CheckIfSuccess(code):

- output: $\text{success} := \forall i : 0 \leq i < \text{length}(\text{code}) : \text{code}[i] = \text{userInput}[i] \Rightarrow \text{True} \mid \text{else} \Rightarrow \text{False}$
- exception: None

ColourBlink():

- transition: $\text{success} == \text{True} \Rightarrow \text{screen flash green} \mid \text{else} \Rightarrow \text{screen flash red}$
- exception: None

ResetInput():

- transition: Clear user input entry
- exception: None

13.4.5 Local Functions

randString(codeLength): Generate random string of length CodeLength

- output: code := newCode
- exception: None

13.4.6 Local Variables

success : \mathbb{B}

code : string

letterToNumberDictionary := $\{A \rightarrow 0, B \rightarrow 1, C \rightarrow 2, \dots, Z \rightarrow 25\}$

14 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.

14.1 Module

WirePuzzle Module

14.2 Uses

Multiplayer Puzzle Module

14.3 Syntax

14.3.1 Exported Constants

NUM_WIRES = 5

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
GenerateWires	-	-	-
GenerateSequence	-	-	-
ConnectWire	\mathbb{Z}, \mathbb{Z}	-	-
DisconnectWire	\mathbb{Z}	-	-

14.4 Semantics

14.4.1 State Variables

ConnectedWires : Array $\langle \mathbb{B} \rangle$

Sequence : Array $\langle \mathbb{Z} \rangle$

14.4.2 Environment Variables

Device Screen

This module has external interaction with the phone screen, as there will be buttons in the game environment that the user presses which will be mapped to functions

14.4.3 Assumptions

None

14.4.4 Access Routine Semantics

GenerateWires():

- transition: Create the unity wire objects
- exception: None

GenerateSequence():

- transition: $\forall i : 0 < i < \text{NUM_WIRES} : \text{wire}[i].\text{position} = \text{rand}()$
- exception: None

ConnectWire(wireId, connectionId):

- transition: $\text{wire}[\text{wireId}].\text{connection} = \text{connectionId}$
- exception: None

DisconnectWire(wireId):

- transition: $\text{wire}[\text{wireId}].\text{connection} = -1$
- exception: None

14.4.5 Local Functions

None

14.4.6 Local Variables

None

15 MIS of Maze Puzzle Module

15.1 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.

15.2 Uses

Multiplayer Puzzle Module

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
GenerateMaze	\mathbb{N}	\mathbb{Z}	-
RotateMaze	$\mathbb{R}, \mathbb{R}, \mathbb{R}$	-	-
BallHitsHole	-	-	-
BallHitsGoal	-	-	-

15.4 Semantics

15.4.1 State Variables

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

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

15.4.2 Environment Variables

The phone's gyroscope rotation triggers the RotateMaze function.

15.4.3 Assumptions

15.4.4 Access Routine Semantics

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

BallHitsHole():

- transition: BallPosition := (0,0,0)
- exception: None

BallHitsGoal():

- transition: CompletePuzzle()
- exception: None

15.4.5 Local Functions

BallMovement: BallPosition x 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.

15.4.6 Local Variables

Hole Locations : $\mathbb{R}, \mathbb{R}, \mathbb{R}$

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

16 MIS of Math Puzzle Module

Puzzle module to be interacted with by two or more users. The module will split an equation into multiple sections and distribute the sections to each user.

16.1 Module

MathPuzzle Module

16.2 Uses

Multiplayer Puzzle Module

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
GenerateSolution	-	\mathbb{R}	-
GenerateSections	-	-	-
CompareSolution	\mathbb{R}	\mathbb{B}	-

16.4 Semantics

16.4.1 State Variables

Solution : \mathbb{R}

16.4.2 Environment Variables

Device Screen

This module has external interaction with the phone screen, as there will be buttons in the game environment that the user presses which will be mapped to functions

16.4.3 Assumptions

None

16.4.4 Access Routine Semantics

GenerateSolution():

- output: out := Solution
- exception: None

GenerateSections():

- transition: Split the puzzle into equal sections based on the amount of users assigned to the puzzle

CompareSolutions(attemptedSolution):

- output: out := Solution == attemptedSolution
- exception: None

16.4.5 Local Functions

None

16.4.6 Local Variables

None

17 MIS of Bomb Puzzle Module

The Bomb 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.

17.1 Module

BombPuzzle

17.2 Uses

Multiplayer Puzzle Module

17.3 Syntax

17.3.1 Exported Constants

COMBO_LENGTH=4

17.3.2 Exported Access Programs

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

17.4 Semantics

17.4.1 State Variables

comboSet : Array of strings

17.4.2 Environment Variables

Device Screen

This module has external interaction with the phone screen, as there will be buttons in the game environment that the user presses which will be mapped to functions

17.4.3 Assumptions

None

17.4.4 Access Routine Semantics

GenerateCombo():

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

ConvertButtonPress():

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

CheckEntry(userInput, comboSet):

- transition: $\text{userInput} == \text{comboSet}[0][\text{currentDigit}] \wedge \text{currentDigit} < 4 \Rightarrow \text{currentDigit} + = 1$
- output: $\text{success} := \text{userInput} == \text{comboSet}[0][\text{currentDigit}] \wedge \text{currentDigit} == 4 \Rightarrow \text{success} = \text{True} \mid \text{else success} = \text{False}$
- exception: None

ColourBlink():

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

Restart():

- transition: $\text{currentDigit} = 0, \text{GenerateCombo}()$
- output: None
- exception: None

17.4.5 Local Functions

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

- output: combo := newCombo
- exception: None

17.4.6 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.

18 MIS of Database/Network Manager Module

18.1 Module

DatabaseNetwork Module

18.2 Uses

None

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

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

18.4 Semantics

18.4.1 State Variables

players : NetworkList< *PlayerData* >

18.4.2 Environment Variables

None

18.4.3 Assumptions

None

18.4.4 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

18.4.5 Local Functions

None

19 MIS of Error Manager Module

19.1 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.

19.2 Uses

None

19.3 Syntax

19.3.1 Exported Constants

19.3.2 Exported Access Programs

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

19.4 Semantics

19.4.1 State Variables

ShowErrorMessage : B

19.4.2 Environment Variables

There are buttons that the user has access to, that will activate some of these functions directly when pressed

19.4.3 Assumptions

19.4.4 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`

19.4.5 Local Functions

`None`

20 MIS of Documentation Module

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

20.1 Module

20.2 Uses

None

20.3 Syntax

20.3.1 Exported Constants

N/A

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
-	-	-	-

20.4 Semantics

20.4.1 State Variables

None

20.4.2 Environment Variables

None

20.4.3 Assumptions

None

20.4.4 Access Routine Semantics

None

20.4.5 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>.

21 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
- Creation of additional puzzle modules/types
- Modification of current puzzle modules to allow them to work with several different amounts of people (ex. 2-player, 3-player, 4-player, etc.)

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, 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. Additionally, even though Unity does have many benefits, there are some negatives such as it being extremely difficult to do automated testing with Unity. 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.