**Chapter 3**

**SYSTEM DESIGN**

This document specifies the detailed design of the “Mystery Doors” project. It deals with system overview of the overall project, proposed architecture designs and alternate architecture design and justification for the proposed one. High level block diagrams which illustrates the database and application program interaction. High level use case report which includes all users and interaction between them. Low level design of each module consists individual use case reports, and related sub-module sequence diagrams, activity diagrams, pseudo code, Analysis and Asymptotic Notation for algorithms.

**3.1 System Overview**

GUI contains the welcome screen or main menu screen which must consist of two buttons. PLAY and EXIT. If user clicks on EXIT button the game must terminate. When user clicks on PLAY button, he/she must be given an option to enable or disable sound. If user enables sound then background audio need to be played using background audio player agent.

When user clicks on PLAY button the game must navigate to next page showing the introduction to game using motion of characters. Characters are moved using motion of images(.gif to enable animation). During gameplay different set and type of questions need to displayed on screen and answers must be validated before proceeding to next step. Different set and type of questions and all the solutions must be stored in an isolated file storage medium i.e. mobile’s flash memory. At a different level’s of game suitable type of question must be fetched from file and displayed on screen.

**3.2 Architectural Design**

The architectural design is the early stage of the system design process which represents

the link between specification and design processes. This involves the identifying major

system components and their communications. The Figure 3.2 shows the architectural design

of our project; here the architecture style used is pipe and filter.

HERE ARCHITECHTURAL DIAG MUST COME

**3.3 Alternate Architectural Design**

This section describes the alternative approach to design our system.

ARCHITECTURAL DESIGN FOR THE ALTERNATIVE APPROACH FOR THE SYSTEM

**3.4 Justification for the proposed design**

This section deals with justification for proposed designs. It also contains the

comparison between the proposed and alternative design, and the reason behind choosing the

proposed design.

BOTH THE ABOVE DIAG COME HERE

|  |  |
| --- | --- |
| PROPOSED DESIGN | ALTERNATIVE DESIGN |
|  |  |
|  |  |
|  |  |
|  |  |

Table 3.1 - Difference between two architecture designs

Table 3.1 shows the comparison between two architectural designs. Since the proposed design has lots of advantages compared to alternative design and also it operates at lower cost compared to of alternate design so we have selected proposed design for building our project.

**3.4 Design Aspects**

The goal of this design is to ensure that the architecture is preserved, and the relationship between the component and modules is clear. This design views the modules as classes. Object oriented approaches are believed to be more natural and provide richer structures for thinking and abstraction. As we are using java programming language which is truly object oriented languages.

**3.4.1 System use case diagram**

DIAGRAM

The system has the following use cases:

Use case 1: Menu Screen

*Primary actor*: User

*Precondition*: -

*Main success scenario*:

1. User logins if username and password match

2. Rejects login if the password is incorrect

Use case 2: Select option

*Primary actor*: User

*Precondition*: User logged-in successfully

*Main success scenario*:

1. Selects any one of the option listed

Use case 3: Contact update

*Primary actor*: User

*Precondition*: User logged-in successfully

*Main success scenario*:

1. Selects contact numbers

2. Enters new number

3. Updates contact

Use case 4: Pre-timed message

*Primary actor*: User

*Precondition*: User logged-in successfully

*Main success scenario*:

1. Enters text body

2. Sets date and time

3. Sends message.

Use case 5: Remote access

*Primary actor*: User

*Precondition*: User logged-in successfully

*Main success scenario*:

1. Logins with the desired username

2. Selects option

**3.4.2 System class diagram**

The class diagram defines Classes that exists in the system, variables used in that class and the functions present in that particular class.

SYSTEM CLASS DIAG

**3.5 Module level design**

This section deals with the various modules used to build the entire system. Our system comprises 4 modules i.e. Contact Update, Pre-timed messages, Remote Access and GUI. Here we describe each module by using use case diagram, class diagram and sequence diagram for each module.

**3.5.1 Module 1:** Game Play

This module consists of the GUI and placing the images at the right place. The main feature of the Game Play is the motion of the images to make the game more interactive.

**3.5.1.1 Use Cases:**

**Figure 3.7 Use case diagram of Game play Module**

Use case 1: Select Contact update

Primary actor: Sender

Precondition: User accepted

Main success scenario:

User logins appropriately

Selects Contact Update option from the menu

Use case 2: Select Contacts

Primary actor: Sender

Precondition: User has selected Contact update option

Main success scenario:

User selects list of contacts for which updating process needs to be carried out

**3.2.1.2 Class Diagram:**

Class diagram is a type of static structure diagram that describes the structure of a system by showing the system‟s classes, their attributes and the relationship between the classes.

Figure 3.8 Class diagram of game play module

The Figure 3.8 shows the class diagram of Contact Update module which shows

the different classes and their data members, methods and interaction between the classes.

The different classes in this module are as follows

1. Current Number: this class fetches the current sim number. It has two private

variables ie Phone\_no and Flag. It has three functions

a. setphone\_no(): fetches the sim number and store it in variable

b. getCurrno(): return current number which is of long type

c. validateCurrNo(): validates whether the stored number is valid or not. It

returns Boolean type

2. ReceiveMessage: This class receives the incoming message and processit. It

had two variables ie senderPhno and formatted Text. It has two functions

a. receiveMsg(): which receives the formatted message and stores the

message information into class variables.

b. ValidateMsg(): This function validates the received formatted message.

3. ProcessContacts: this class has two list variables ie seleceted list and selected

count. This class fetches all the contacts from the mobile phone database and

display it for selection. Once the user selects the contacts the selected contacts

area stored in these variables. It has 4 functions ie

a. readContactDB(): reads all the contacts from the mobile phone database.

b. displayContacts(): display the read contacts on the user interface.

c. selectContacts(): provides a mechanism to allow user to select the

displayed contacts

d. getSelectedContacts(): gets the selected contacts.

4. SendMessage: this class is used to send formatted message to receipent. It has

two private variables ie ReceipentNumber and FormattedText. It has two

member functions ie

a. sendMessage() : sends the formatted message to receipent

b. ValidateMessage(): validates formatted message before sending

5. ProcessText: this class formats or unformat the text. It has three functions ie

a. validateText(): validates the formatted Text

b. formatText(): Coverts unformatted text to formatted Text

c. unFormatText(): converts format text to unformatted text

6. DatabaseUpdate: this class updates the old number with new number in the

database. It has three variables OldNumber, NewNumber, Command. It has

three functions

a. ValidateNumbers(): Validates both new number and old number.

b. ValidateCommand(): validates the update command.

c. updateDatabase(): Performs updating process in mobile database.

**3.2.1.3 Sequence Diagram:**

This section shows how processes operate with one another and in what order

with the help of sequence diagram

SEQUENCE DIAGRAM OF THE INDIVIDUAL MODULE

**3.5.2 Module 2:** Accelerometer Puzzle

To provide the orientation of the device during the game.

Use cases

**3.5.3 Module 3:** Accessing the Database and Editing the Images

This module requests the background file transfer. It also helps in enabling the management of the background file transfer.

USE CASES

**3.5.4 Module 4:** Playing the Background music

Since this application is designed to play audio in the background, it is likely that the user will return to this application with audio already playing. The user interface should show this by updating appropriately to reflect the current state of playback, as well as the currently playing track.

USE CASES

**Chapter 4**

**TESTING**

This chapter gives a brief description about the testing phase of the system which is to be followed after the implementation phase of the project. This consists of the unit testing of each module and integration testing of the whole system.

**4.1 Unit Testing**

Unit testing refers to tests that verify the functionality of a specific section of code,

usually at the function level. In an object-oriented environment, this is usually at the class

level. This chapter specifies the testcases for four modules of the system.

**4.1.1 Module 1:** Game Play

This module consists of the GUI and placing the images at the right place. The main feature of the Game Play is the motion of the images to make the game more interactive.

|  |  |
| --- | --- |
| Test Case Id: | 1.1 |
| Unit to test |  |
| Assumptions |  |
| Test Data: |  |
| Steps to be executed |  |
| Expected result |  |

Table 4.1 Test case for game play

DESCRIPTION OF THE ABOVE TABLE

**4.1.2 Module 2:** Accelerometer Puzzle

To provide the orientation of the device during the game.

|  |  |
| --- | --- |
| Test Case Id: | 2.1 |
| Unit to test |  |
| Assumptions |  |
| Test Data: |  |
| Steps to be executed |  |
| Expected result |  |

Table 4.1 Test case for Accelerometer Puzzle

DESCRIPTION OF THE ABOVE TABLE

**4.1.3 Module 3:** Accessing the Database and Editing the Images

This module requests the background file transfer. It also helps in enabling the management of the background file transfer.

|  |  |
| --- | --- |
| Test Case Id: | 3.1 |
| Unit to test |  |
| Assumptions |  |
| Test Data: |  |
| Steps to be executed |  |
| Expected result |  |

Table 4.1 Test case for Accessing the Database and Editing the Images

DESCRIPTION OF THE ABOVE TABLE

**4.1.4 Module 4:** Playing the Background music

Since this application is designed to play audio in the background, it is likely that the user will return to this application with audio already playing. The user interface should show this by updating appropriately to reflect the current state of playback, as well as the currently playing track.

|  |  |
| --- | --- |
| Test Case Id: | 4.1 |
| Unit to test |  |
| Assumptions |  |
| Test Data: |  |
| Steps to be executed |  |
| Expected result |  |

Table 4.1 Test case for Playing the Background music

DESCRIPTION OF THE ABOVE TABLE

**4.2 Integration Testing**

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localized more quickly and fixed. We use bottom up approach for our project, as we combine low level components into clusters and test them. These clusters are combined moving upward in the program structure. This is carried out until the entire system is tested. Here fault localization is easy, logic modules are tested thoroughly and testing can be done in parallel with implementation.

**Chapter 5**

**IMPLEMENTATION**

This chapter gives a brief description about implementation details of the system by

describing each module with its code skeleton.

**5.1 Module 1:** Game play

**Description: copy from pavan’s book**

**Code Skeleton:**

**5.2 Module 2:** Accelerometer Puzzle

**Description: copy from harish’s book**

**Code Skeleton:**

**5.3 Module 3:** Accessing the Database and Editing the Images

**Description: copy from pradeep’s book**

**Code Skeleton:**

**5.4 Module 4:** Playing the Background music

**Description:** To imply a Background agent specifically designed to play audio in the background. To handle user actions such as clicking the play, next, and previous buttons. Since this application is designed to play audio in the background, it is likely that the user will return to this application with audio already playing. The user interface should show this by updating appropriately to reflect the current state of playback, as well as the currently playing track. To update the UI when the application is loaded, override the OnNavigatedTo virtual method.

**Code Skeleton:**

**Chapter 6**

**SCREEN SHOTS AND RESULTS**

This chapter gives the screen shots of all the modules. This chapter also gives the

expected results and actual results of all the modules.

6.1 Game play module

|  |  |
| --- | --- |
| **Test Case Id:** | TC1\_1 |
| **Unit to test** | Menu Page |
| **Test Data:** | Button {PLAY} |
| **Steps to be executed** | 1. Click on the application icon.  2. You will be directed to the Menu page of the application.  3. Click PLAY button.  4.You will be directed to the second page.  . |
| **Expected result** | Successful direction to the second page of the application. |
| **Actual result** | Successful direction |
| **Pass/Fail** | Pass |
| **Comments** | ----- |

OUTPUT:



|  |  |
| --- | --- |
| **Test Case Id:** | TC2\_1 |
| **Unit to test** | Second page |
| **Test Data:** |  |
| **Steps to be executed** | 1. On successful execution of the Menu page, you are directed to this page.  2. The introduction of the story starts.  3. The animation of the characters is to be seen.  4. Click NEXT button. |
| **Expected result** | Successful direction to the next page of the application. |
| **Actual result** | Successful direction |
| **Pass/Fail** | Pass |
| **Comments** | ----- |

Output:

Screenshot

|  |  |
| --- | --- |
| **Test Case Id:** | TC3\_1 |
| **Unit to test** | Level- 1 of the game begins |
| **Test Data:** | Puzzle is asked ro the user. |
| **Steps to be executed** | 1. On successful execution of TC2\_1, you are directed to this page.  2. This page contains the 1st puzzle of the 1st level.  3. You are asked to answer correctly.  4. Click ENTER to submit the answer.  5. the answer refers the database. |
| **Expected result** | Proper display if the answer is correct/wrong |
| **Actual result** | Proper display |
| **Pass/Fail** | Pass |
| **Comments** | ----- |

Output:

Screenshot

**6.2 Accelerometer puzzle module**

**6.3 Accessing the Database and Editing the Images module**

**6.4 Playing the Background music module**

**Chapter 7**

**CONCLUSION AND FUTURE WORK**

With a large and growing user base that generates large revenues but also raises numerous technological applications, Mobile Games have recently started to attract the interest of the research community. In this work we have tried to identify the availability of the puzzle-based games in Windows phone Operating System. To address this, we have designed and implemented “Mystery Doors”, a puzzle-based gaming application for Windows phone. Our architecture focuses on puzzle game, which is one of the most important components of the generic game. Anything to add??