



SDV 602

Milestone 2

Achuthanand Vasudevan

Graduate Diploma in IT - 2019

Table of Contents

1. INTRODUCTION.....	1
2. MODEL-VIEW-VIEWMODEL ARCHITECTURE	2
3. GAME DESCRIPTION	3
4. STORYBOARD & DESCRIPTION	6
4.1 Login.....	7
4.2 Register	8
4.3 Game Home	9
4.4 Character Interaction.....	10
4.5 Game Room	11
4.6 Co-Player Details	12
4.7 Game Room Message	12
4.8 Chat Room Panel	13
5. USE-CASE DIAGRAM.....	14
5.1 Use-Case Explanation	16
6. CLASS DIAGRAM.....	19
6.1 Domain Classes.....	22
7. SQLite Database.....	24
7.1 Self-Contained	25
7.2 Zero-Configuration	25
7.3 Transactional.....	25
7.4 Dynamic Datatypes	26
8. FUNCTIONAL TEST	28
8.1 Use case Testing	28
8.2 Data Update Testing	32
9. REFERENCES.....	33

1. INTRODUCTION

The game, “**Jack, The Savior!**” is a text-based game where the player needs to input certain text data to continue playing. The text data will be processed and according to the input further actions will decide by the game management system. The game is basically designed and developed using Unity game development platform and using C# for programming.

The text-based environment is made possible in game by creating questioning and answering session between the player and system. The player will have certain health while starting the game and will be reduced by certain amount when the player answered wrongly. The game will start by explaining game situation and then the player can start the game.

There are various approaches exists now for creating game applications, such as:

- Model View Controller (MVC)
 - Model View Template (MVT)
 - Model View View/Model (MVVM)
- etc..

Each approach has their own strength and weakness towards the development. Here I am going to utilize Model View View/Model approach which is also known as MVVM architecture. More details regarding MVVM will be discussed in the later part.

The game is a 2-dimensional (2D) game which supports multiplayer facility. Multiple players can join the game and the winner will be those who finish the game first and with most health.

2. MODEL-VIEW-VIEWMODEL ARCHITECTURE

As the technology more emerge the need of effective and fast development process is inevitable. By making it possible it is also relevant to keep the system simple without being much complexity. To reduce the complexity of a system patterns are good and effective solution. Model-View-ViewModel or MVVM is one of the several patterns or approaches now using. Here the abstract flow is $\text{Model} \leftrightarrow \text{ViewModel} \leftrightarrow \text{View}$.

In traditional User Interface development, developer need to create a view using available window functionalities or similar process and will write all logical implementation. This makes the View large which then creates a strong dependency between UI and binding logic. This led to complex situation in working as team environment and keeping all code in one are leads to ineffective maintainability. This mainly happens because of the tight coupling between the view and logic which includes both business logic, event handling and data binding.

MVVM pattern usually takes advantages of device capabilities and thus it makes use of device memory to extend the application performance. Thus, it allows better user experience which then allows then to have the application on various devices which has varying screen size. MVVM enables the separation of graphical UI from business logic.

- **Model:** It represents the actual data or information that the application deals with. Here in the game the text which moves the story and game is stored using the game models.
- **View:** It is the most familiar part of any system. Here is where the end users really interact with. This visualizes data in a presentable way according to the nature or data and the way user needs it. The view has no knowledge or information about the model. It is fully controlled by a controller. It is where the player interacts with the game. Simple the game interface.
- **ViewModel:** It is what enables the view using the models available. It converts the data to presentable format instead of letting the model aware of user view. In game it is what which accepts the input from the user and then process it for further actions/moves.

3. GAME DESCRIPTION

“Jack, The Savior!” is a fun and entertainment game for children since it doesn’t involve any violent actions or behaviors which will affect the mind of children. The questions designed for the game is similar to brain teaser and basic navigations which helps the children to think more and thus, they can develop their problem-solving skills.

The player has 2 options to play the game. Either the player can start a new game or join some random game. Joining some random game is like joining a game where other players are playing the game thus, those players may have more advantage than the later joined player since they joined earlier.

The game story is as follows.

Once there lived a bunny, Jack with his parents Will and Pink in a village very close to a jungle. In the deep jungle there also lived a fearsome monster in a castle who preyed on innocent animals. One day, Will and Pink along with their neighbor Tinku duck went to jungle in search of food and firewood. They couldn’t find enough food in the outer jungle, so they went deep into the forest. Suddenly they found a garden full of fruits and vegetables and they ran into the garden to pick some. They didn’t know that it was the monster’s garden. By that time the monster came back from his hunting. He found these three uninvited animals in his garden and got angry. He rushed to the garden and caught Will and Pick. Tinku somehow managed to escape from the monster and ran back to the village. He explained everything to Jack and told that only the ones who are blood related can go into the castle and save them. Jack took on oath that he will save his parents by any means.

The game starts here, the player will play the role of Jack. In-order to get inside of the castle successfully the player need to complete first level. Each level will have certain questions to answer. Passing each level successfully will help the player move further inside of the castle & will get elements required to kill the monster and save parents. When all the levels are passed the player will be able to fight with the monster and to release the parents. The player needs to complete all the tasks before he loses his all health.

When the player will start the game, 10 hearts are given as health. Each wrong answer in the game session will reduce 2 hearts. Hence 5 wrong answers/tries will drain all the health and the player will lose the game.

The whole game is divided in to 5 levels. Each level will consist of 3 – 5 questions or directions to navigate. Level 1, 3 and 5 contains brain teaser questions whereas in level 2 and 4, player need to analyze the question and need to track in which direction he/she must move.

The game consists of mainly 6 scenes including game room and certain dialog boxes to show information, warning, danger messages.

- Login scene
 - This scene is loaded first when the game application is loaded. It enables the user to login into the system.
- Register scene
 - This allows new users to create profile in the system.
- Dashboard scene
 - This provides the game story outline and options to start a new game or to join a random game.
- Character Interaction scene
 - This scene will load when the player starts playing the game where Jack and Tinku is exchanging their dialogues.
- Game Room
 - The actual game will run here in game room. The game room is where the game system interacts with the player.
- Co-player detail scene
 - This scene will provide the details of co-players, like their health and time details. The Co-player details scene will show when the player joined a random game and whenever the player needs he/she can open it through Game Room.

UI components used for designing the game are:

- Image
- Text

- InputField
- Scrollbar
- Buttons
- Panels
- Empty Game objects for containing a group or data

Each components and button actions are defined in the storyboard section.

The game room is an active interactive system which acts like a quiz where the player need to think wisely and answer these. All the questions are in child level and no need to think like a pro but think like a child to answer it. The player can only move further only if the answer is correct. Wrong answer reduces the player health by 2 hearts. Hence it is very important to play the game wisely.

Since the game interaction is in the form of a questioning answering session, this will be a great choice for students to increase the analytics as well as problem solving skills. The game interaction is made possible with dynamic question which retrieved from the database. Thus, just altering the questions in the database can extend the game to any level of players.

4. STORYBOARD & DESCRIPTION

Storyboard is a graphical organizer of the application which needs to be developed. These graphical representations can be visualized in the form of illustrations or images in a sequence which happens in the application. This graphical visualization helps the developers to initiate the development process with the basic design which can be maintained throughout the development. This gives a visual idea on how the game is playing and thus the developers will be able to develop the application real time without having confusion in the design part.

It is a quick way of getting a perspective into what the game will look like prior to the production even before a prototype is developed and tested. These storyboards will help to see where there are gaps in the gameplay actions which will help to give richer experience for the game. While giving the storyboard it is very important to organize it in the proper way so that it will depend on the precise goals of storyboard session.

With storyboard it is equally important to describe what is happening in the storyboard. Storyboard includes the story background of game and the elements required for the game progress. These elements are initiated with the help of user actions. Hence it is essential to describe what will happen when the user initiates the action.

Advantages of storyboard and description are:

1. Will get an overview of the application just with a look.
2. Can improvise the design ideas.
3. Provides concise information.
4. Clear idea about the system before its development

Below explains the story board of game application and the descriptions of user actions related to the game.

4.1 Login

Board 1

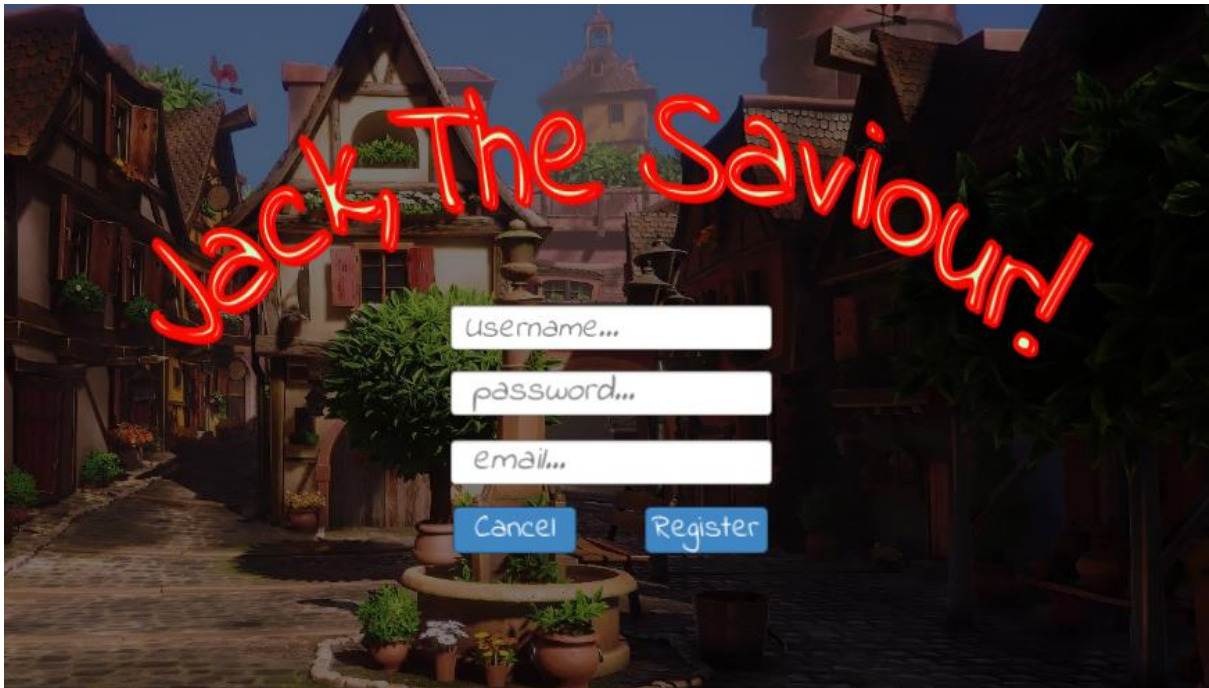


The player need login to the system before start playing the game. Only registered player can login successfully.

Action	What happens?	Where to?
Click Login	1. Check if user exists or not?	
	1.1. If not, throw an error saying so.	Go to board 2
	1.2. If exists, then validate username & password	
	1.2.1. If validation success	Go to board 3
	1.2.2. If validation fails, throw error message and stay in the same login page	
Click Register	1. Opens registration page	Go to board 2
Click native back button	Closes the application	

4.2 Register

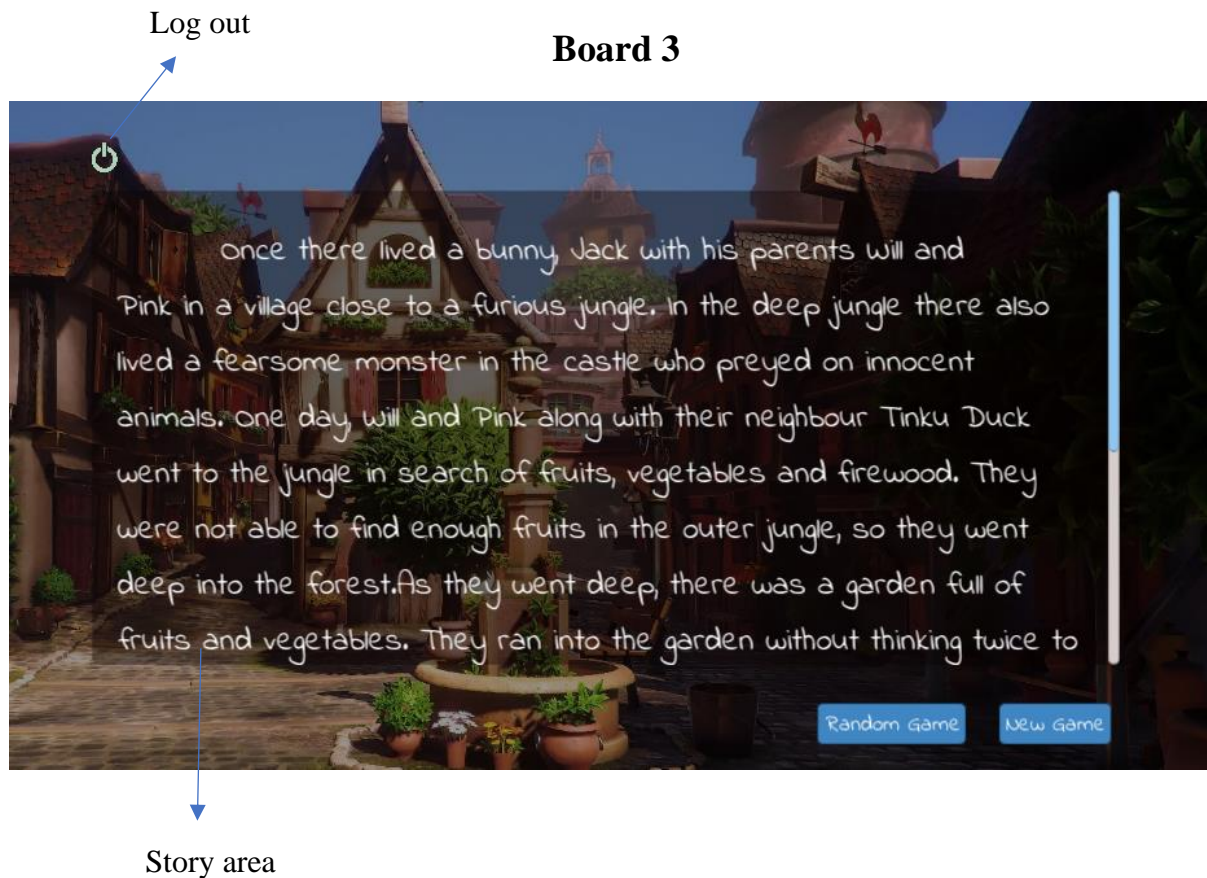
Board 2



Only registered player can login and play the game. The username should not be an already taken one. The email should be a valid email.

Action	What happens?		Where to?
Click Register	1. Check if user exists or not?		
		1.1. If exists, throw an error saying so.	
	1.2. If not check whether all fields are filled or not		
		1.2.1. If not, throw an error saying so	
		1.2.2. If all fields are filled then register the new user details	Go to board 1
Click Cancel	1. Navigate back to login page		Go to board 1
Click native back button			

4.3 Game Home



Once the player is successfully login, the game home will load. Game home will describe the basic story outline and actions to start game, join some random game and log out.

Action	What happens?	Where to?
Click New Game	1. New game will create	Board 4
Click Random Game	1. The player will join in some random game	Board 4
Click Logout	2. The player will be logged out	Board 1

4.4 Character Interaction

Board 4

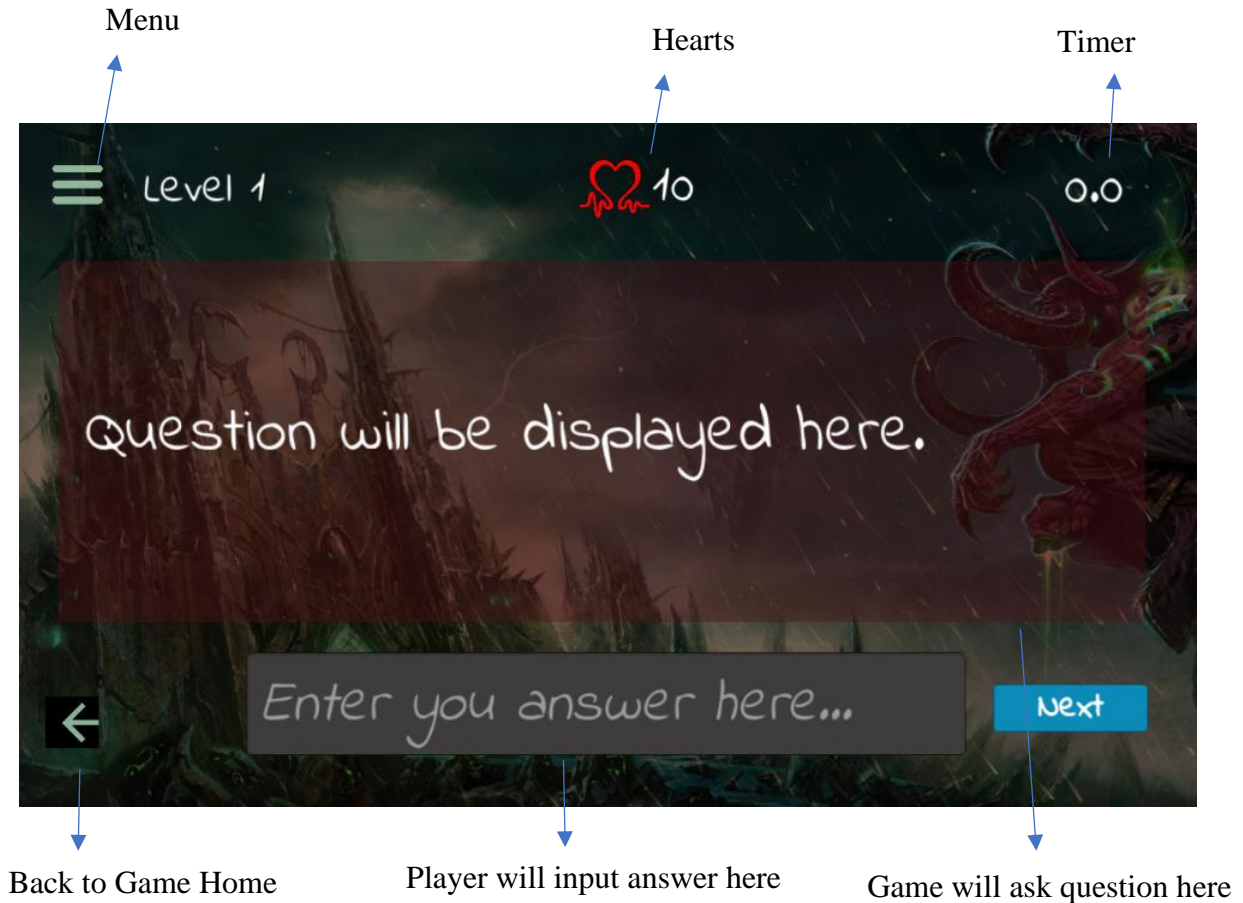


The character interaction screen is where the characters of the game will interact each other. The main dialogue delivery is that the Tinku duck explains the situation happened in the jungle to Jack so that the Jack will get a clear about it.

Clicking Next button will trigger the dialogue delivery once all the dialogues are finished clicking Next button will open the Game Room.

4.5 Game Room

Board 5

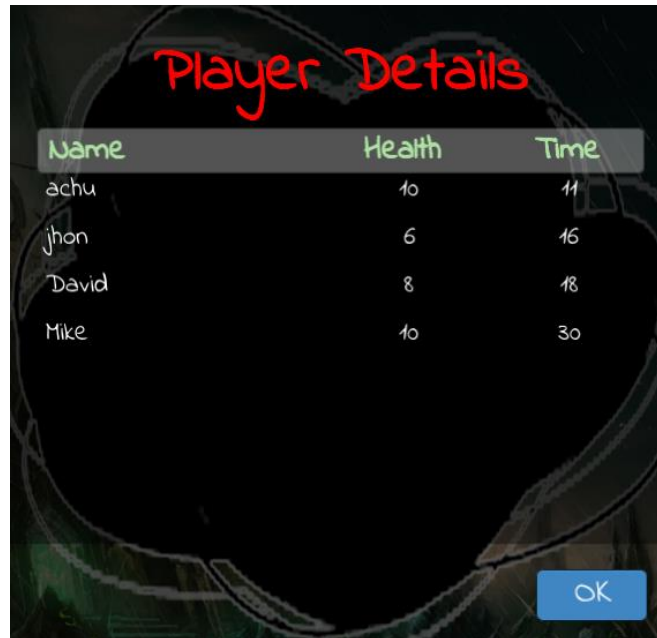


Once the game starts the interaction between player and game will happen here. The game will ask questions and player will answer.

Action	What happens?	Where to?
Click Menu	1. Co-Player details will show	Board 6
Click Back	1. The player will navigate back to the Game Home	Board 3
Click Next	1. The entered answer will check	
	1.1 If correct next question will appear	
	1.2 If not correct reduce health and stay in same	

4.6 Co-Player Details

Board 6



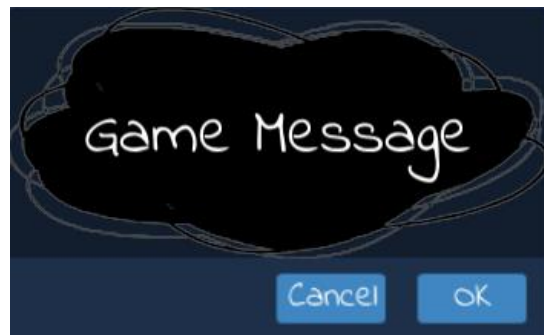
The screenshot shows a 'Player Details' screen with a dark background and a chalkboard-like texture. At the top, the title 'Player Details' is written in red. Below it is a table with three columns: 'Name', 'Health', and 'Time', all written in green. The table lists four players: 'achu', 'jhon', 'David', and 'Mike'. At the bottom right, there is a blue 'OK' button.

Name	Health	Time
achu	10	11
jhon	6	16
David	8	18
Mike	10	30

The co-players details will be shown here when clicking menu button in Game Room. Clicking OK button will close the detail page.

4.7 Game Room Message

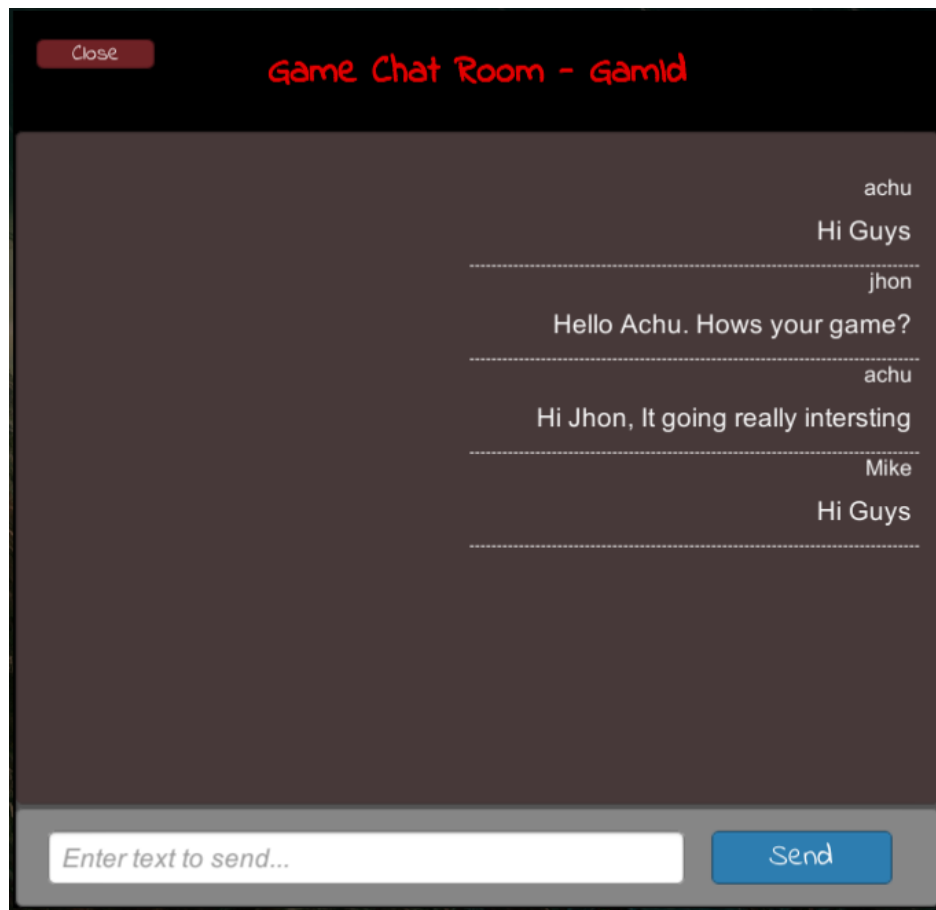
Board 7



All the game related messages will show in the above message scene.

4.8 Chat Room Panel

Board 8



The chat rooms allows the players in a game to interact. In the current phase only chat room component is made, no backend implementation.

Action	What happens?	Where to?
Click Close	1. Chat room will close	Board 5
Click Send	1. The message will update in the chat room	

5. USE-CASE DIAGRAM

Use-Case diagram is a graphical interface to depict the interaction between actors and system in a most simple and efficient way. The main aim to identify, clarify and organize various requirements required for the system to design. In-order to deploy use case, UML is used which is a standard notation for the mapping of real-world objects and systems. It helps to analyze various scenarios as well as scope of the system.

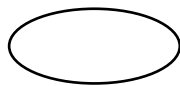
If the requirement to analyze the system in a deep nature, use-case is not a good choice. It depicts only a high-level overview of the relation between actors and system.

In-order to draw a use-case diagram, it is essential to have a understanding of basic building blocks.

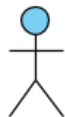
- **Actors:** Actors are those who interact with the system. It can be a person, an organization or even can be an external system who tries to interact.
- **System:** System can be considered as a scenario happening between actors and the system.
- **Goals:** It is what the final aim of use-case, it should describe the while activities to reach the goal.

Here, we mentioned below symbols and notations to represent use-case for the required scenario.

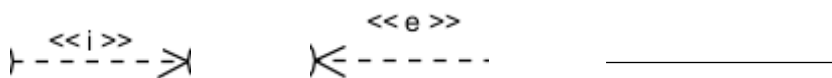
- Use cases are represented using horizontally shaped ovals.

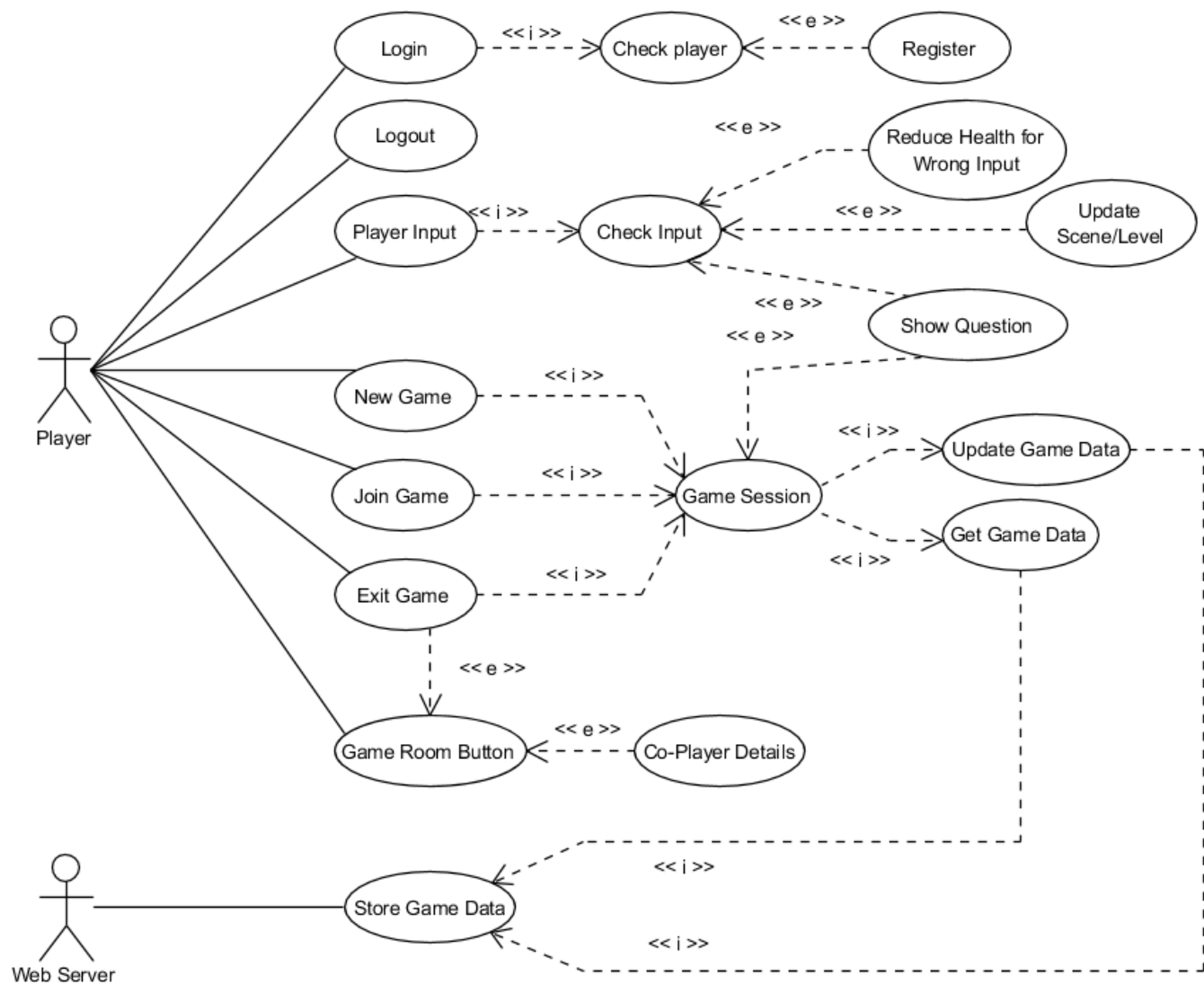


- Actors are represented by stick figures



- Dependencies are represented by dashed line & Association with a straight line





Use-Case Diagram

The above shown is the Use-Case diagram of game “**Jack, The Savior!**”. There are basically 2 actors exists for the game, Player and Web server. Player plays the game and Web Server provides the data to serve the game and store data for future use. The use cases are user initiated and game initiated, means user-initiated use cases are those which initiated by users and remaining happens either in the background or by means of game action. Below describes each use cases in detail.

5.1 Use-Case Explanation

1. Login

It defines the process of user logging into the system. The user should have valid username and password to login into the system.

- If username and password combination is correct the user will successfully enter the game and user status is updated as active
- If the username does not exist, will throw error
- If the username and password combination does not exist, will throw error

2. Logout

It defines the process of user exiting from the game. When user logouts the game the status of user is updated to inactive and will update the last login time. This is processed when player clicks the logout button inside Game Home scene.

3. Register

The player should have valid credentials to login in-order to play the game. Register use case is used to register new player.

- Username should be unique, else throw error.
- Email should be in valid format, else throw error.

4. Check Player

Check Player use case is used to check whether the user already exists or not. This will be used by Login & Register use case to check whether the user is valid user or not and does username already exists or not consecutively.

5. New Game

New Game use case is used to create a new game when the player did so. The player can start a new game from Game Home scene by clicking New Game button. This will create a new game with 10 hearts as health level.

6. Join Game

Join Game use case is used to join a random game. The player can join random game by clicking Random Game button in the Game Home scene. Here also the player will get 10 hearts.

7. Exit Game

Exit Game use case is used to exit from the game session. The player information will be saved and can continue when comes again later. The exit game button will be available in the left bottom corner of the game room.

8. Game Room Button

The game room button is mainly used to check the details of co-players in the game. This will show the health details of co-players in the game. It will help to know the health condition and time taken of other players and thus it gives an idea of how far they are. The game room button is available in the left top corner of the game room.

9. Co-player details

Co-player details use case will be used to get the details of player in the same game when game room button is clicked.

10. Player Input

Player Input use case is used to manage the player input in the game. The actual input is the answer to question which is asked.

11. Check Input

This use case is used to check the input is correct or not according to the question asked. If the input is wrong the game will throw an information message and reduce the health level.

12. Reduce Health for Wrong Input

This use case is triggered when the check input use case throws wrong answer message, thus, the health level of the player will be reduced.

13. Update Scene/Level

This use case is used to update the questions/level when the player inputs right answer.

14. Show Question

Show Question use case is used to update the question when player answer the previous question successfully. If the answer is wrong old question will hold else new question will be updated. This use case will be triggered according to the Check Input use case.

15. Game Session

Game session use case is used to manage single game instance. It stores all the information related to the game and will provide necessary information required for the game. It acts as a temporary storage for the game.

16. Update Game Data

Update Game Date use case is triggered by the game session use case when the game data needs to be updated. It happens periodically and when player exists the game.

17. Get Game Data

This use case is used to get the game related information such as questions and answers as well as other data required for the game to run.

18. Store Game Data

This use case is used to store all the game related data where data stored in the game session is transferred to the web server so that it can be retrieved and updated as game needs.

6. CLASS DIAGRAM

Class diagram is a Unified Modelling Language. It describes the architecture/structure of a system's classes by associating attributes and methods/operations associated with those classes and the relationships among them.

Class diagram mainly built with :

- Classes
- Relationship between classes

In the diagram a class consists of

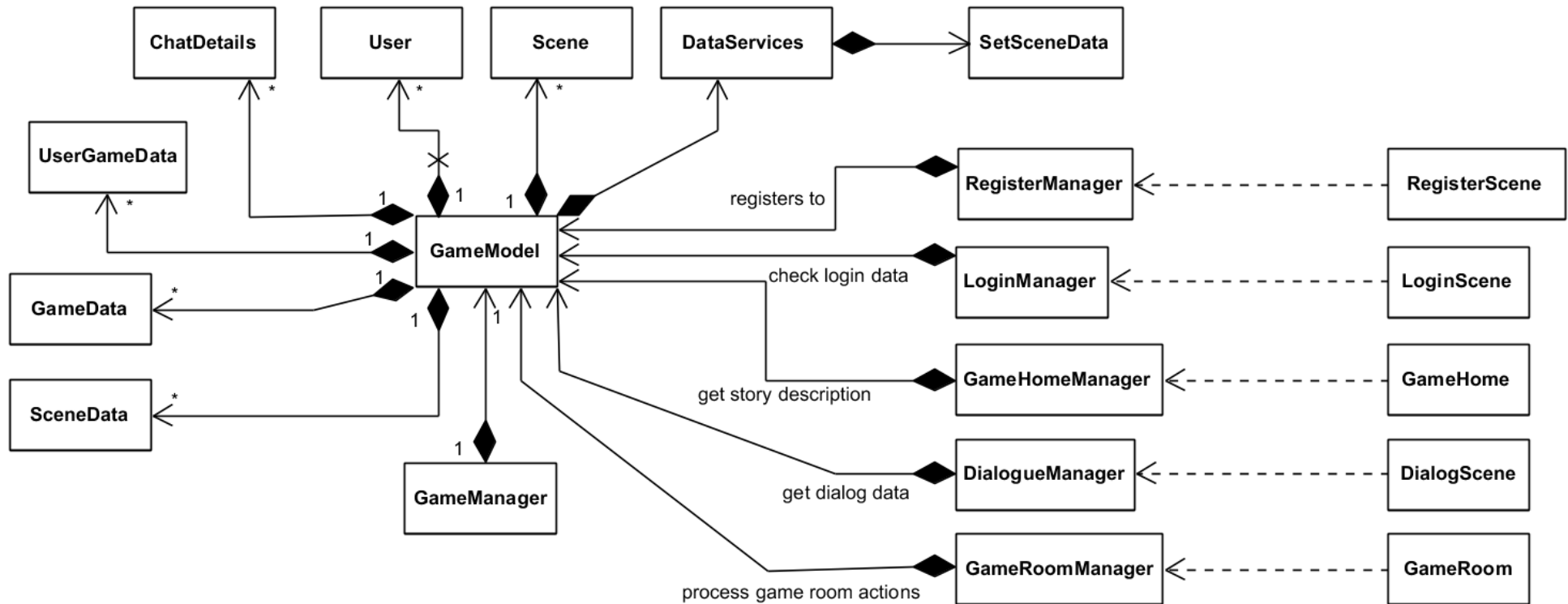
- Class name
- Attributes
- Operations/Methods

Relationship between classes can be one of the following types

- Inheritance
- Simple Association
- Aggregation
- Composition
- Dependency

The relationship between classes made clearer and descriptive by using following features.

- Relationship roles
- Navigability
- Visibility of class attributes and operations
- Multiplicity



Class Diagram

The above shown is the class diagram for the game application. The shown class diagram is a high level representation. It describes only what classes are using and the relationship between them. This is still the game is in developemnet mode and once it completes all the attributes and operations can be included in the class diagram. Hence by progressing development process these will be more clearer and the class diagram will adapt into its descriptive form and may more classes will be added to make the application scalable and maintainable.

When final document for Milestone 3 submits, class diagram will describe all of the attributes and methods used. This will give a clear idea of how the game is running.

Here the Dataservices class enables the connection between game and database when the game starts and maintain this connection for further database updates. Through GameModel class the game will interact with DataService class and it then to database to update data accordingly.

The whole game implements the following each segment to run the game properly:

1. GameManager class to control game instance and to initialize the GameModel.
2. GameModel class to provide relevant data and will act as an intermediate between manager classes and DataService class
3. Supporting classes to support game flow.
 - a. SetSceneData
4. Manager classes to manage the user actions and to populate the user interface according to user actions
 - a. LoginManager
 - b. RegisterManager
 - c. GameHomeManager
 - d. DialogManager
 - e. GameRoomManager
5. User interfaces or scenes
 - a. RegisterScene
 - b. LoginScene
 - c. GameHome
 - d. DialogScene
 - e. GameRoom
6. Domain Classes which act as database tables
 - a. User
 - b. GameData
 - c. UserGameData
 - d. SceneData
 - e. ChatDetails

7. DataServices class to communicate with database

According to the MVVM architecture

- Model is GameModel
- Views are User interfaces or scenes
- View-Models are Managers

6.1 Domain Classes

- **User** – Used to store user details for login purpose. Fields in User domain are
 - Username – primary key
 - Password
 - Email
 - LoginStatus
 - GamesWon
 - BestHealth
 - BestTime
 - LastLogin
- **GameData** – Used to store game details when a game is created by a player. It has
 - GameId – primary key
 - BestHealth
 - BestTime
 - CreateDateTime
 - GameStatus
 - Winner
- **UserGameData** – Used to store game related user details. It contains
 - Id – Primary Key, Auto Increment
 - GameId
 - Username
 - Health

- TimeTaken
 - CurrentLevel
 - IsFinished
 - IsWon
 - StartDateTime
 - EndDatetime
- **SceneData** – Used to store game related data which is required for the interaction between player and game. It has
 - SceneId – Primary key
 - Question
 - Answer
 - Level
- **ChatDetails** – Used to store chat messages. It has
 - Id – Primary Key, Autoincrement
 - GameId
 - Username
 - Message
 - SendTime

7. SQLite Database

SQLite is a library that implements small, fast, self-contained, high-reliability, full-features SQL database engine (SQLite, 2019). It is one of the most popular and most used databases in the world. It is already built into mobile phones and most computers and bundled inside lot of applications that people use every day.

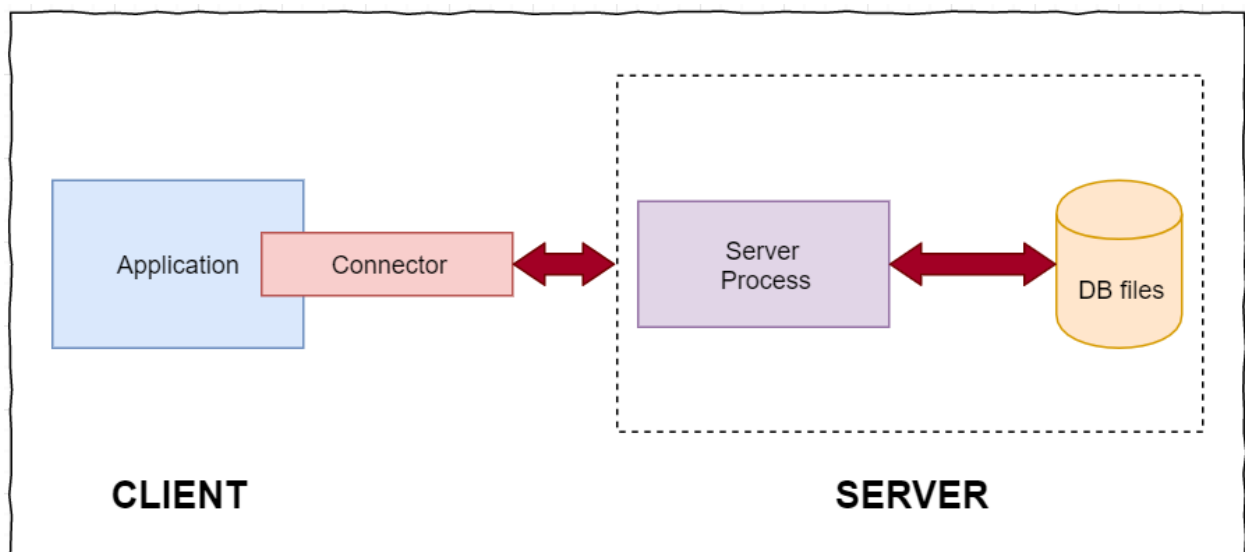
The main advantages of using SQLite are (SQLite, 2019):

- Stable
- Cross-platform
- Backward compatible
- Serverless

Before looking to other features, Serverless is the most important feature and it becomes the prominent reason of the popularity of SQLite database.

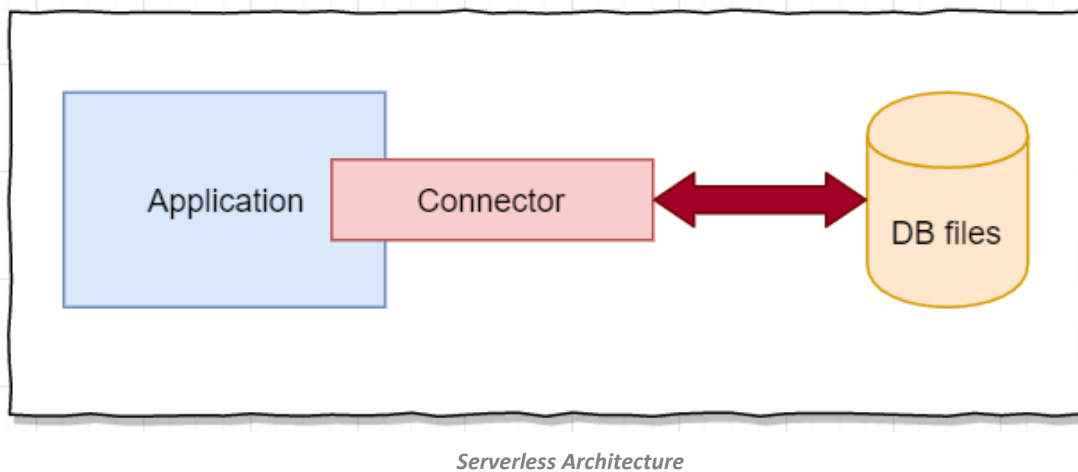
Normally, Relational Database Management System (RDBMS) such as MySQL, PostgreSQL etc. requires a dedicated server to operate. It uses TCP/IP protocol to send and receive requests through the server. This type of architecture is normally called as client/server architecture model (SQLite Tutorial, 2019).

Below shows the traditional RDBMS client/server architecture:



Since SQLite is serverless it doesn't work in this way. It doesn't need a server to run as above to communicate with the database. SQLite database is integrated with the application that access the database. The application will interact with the database directly.

Below shows the serverless architecture of SQLite.



7.1 Self-Contained

SQLite requires only minimal support from the operation system as well as from any external library. This enables SQLite to be very useful when working in environments such as embedded devices like phones(any type), game consoles etc. (SQLite Tutorial, 2019). If an application is developing that uses SQLite the developer just need to drop the source code of SQLite into the project and compile it with the application code.

7.2 Zero-Configuration

Since SQLite follows serverless architecture, no need to install SQLite for using it with applications. Hence, there is no server that needs to configure or to start or to stop. Simply SQLite doesn't use config files.

7.3 Transactional

All the transactional queries in SQLite is fully ACID-complaint. Hence, all the changes within a transaction happens completely or not even in unexpected situations such as crash, power failure etc. occurs.

7.4 Dynamic Datatypes

SQLite can create any value in any column without considering the datatype. Hence the developer can save any type of data in the database.

For a situation, if the model contains a field with datatype Boolean, when running insert query SQLite automatically identifies and save the column type according to that. In this case the Boolean value will considered as 0/1 in the database system. The developer need not to worry whether it return in the same format he want.

8. FUNCTIONAL TEST

The functional test makes sure that the system works perfectly fine without having issues with the functional requirements/specification. These are tested by giving input and analyzing the output. Below shows some of the functional testing cases for milestone two.

8.1 Use case Testing

Use Case	Function being Tested	Initial System State	Input	Expected output
Login Check Player	Check user login credentials	Login Scene shown with empty fields	Enters username & password and clicks Login button	Login successful - Display Game Home Scene
				Existing username – Display error saying so
				Invalid username password – Display error saying so
				Empty fields – Display error saying so
Register Check Player	Check user register data	Register Scene shown with empty fields	Enters username, password & email and clicks register	Valid data – Successful Register message is shown, and Login scene will load

				Existing username – Display error saying so
				Invalid email – Display error saying so
				Empty fields – Display error saying so
Logout	Log out user session	User already logged in and Game home scene shown	Click logout button	Successfully end of user session and navigates to Login Scene
New Game	Create a new game default health is assigned to player	Game home scene shown, and no new game created	Click New Game Button	New game is created and Loads the Game room with default health and the timer starts running
Join Game	Multiple players can join a running game with default health assigned	Game home scene shown, and should have some games running	Click Random Game button	Player will be added to a running game with default health and timer starts running
Game session	To store data temporarily a game session is created	No data regarding the current session	Click New Game/ Random game button	Game session with default values will be created
Get Game data	To retrieve game related data when a new game is created, or new player joins existing game	No game data to run the game	Click New Game/ Random game button	Game data required for the successful game running will be provided

Update game data	To update game related data when a player joins the game or out of the game	No game data to update since no new player joined the game or no existing player out of the game hence same game data as game session	New player joins the existing game or existing player out of the game	Updates the player actions in the server
Player Input Check Input	Player input the answer of questions asked which is then processed to check whether the answer is correct or not	The text input field in the game room is empty.	Player input the answer to the question	Correct answer – Show correct answer message and load next question
				Wrong answer – Show wrong answer message and hold the same question
Reduce health for wrong input	The player health is lowered when the answer is wrong message is shown and clicks ok	Wrong answer message is displayed but not clicked the ok button	Player clicks ok button in the message window	The player health will be reduced by 2
				If the health becomes 0 game over message will display

Show question/Update Scene/level	Upgrading to higher level or updating the questions when correct answer is given	Previous level question or same level question	Player input answer to the question asked	If the answer is correct next question will display
				If all the questions in a level is correct then player will be upgraded to higher level
Game room button, Co-player details	Showing the co-player details	Player is playing the game in game room	Player clicks menu button in the game room	A window will pop-up with details of co-player in the game. If no players in the game then will show message saying so
Exit Game	Player will be out of the game	Player is playing the game in game room	Player clicks the back button in the game room	The game data will be updated, and the player will be out of the game.

8.2 Data Update Testing

With these functional testing it is also tested that whether the database is updated correctly when:

1. New user is created
 - a. User table is updated with new user details
2. A user gets login
 - a. Status column is updated
 - b. LastLogin column is updated
3. A user gets logout
 - a. Status column is updated
4. New game is created
 - a. GameData table is updated
 - b. UserGameData table is updated with the game related user details
5. Joined a random game
 - a. UserGameData table is updated with new user to the existing game
6. When the answer is correct/wrong or when a level is updated
 - a. User table is compared with existing score details and will get updated if required
 - b. UserGameData table is updated with new score details
7. When a user fails/won the game
 - a. User table is compared with existing score details and will get updated if required
 - b. UserGameData table is updated with new score details

9. REFERENCES

Functional Testing. (2019, October). Retrieved from Software Testing Fundamentals:

<http://softwaretestingfundamentals.com/functional-testing/>

Model View ViewModel Explained. (2019, October). Retrieved from Wintellect:

<https://www.wintellect.com/model-view-viewmodel-mvvm-explained/>

SQLite. (2019, October). Retrieved from SQLite Documentation: www.sqlite.org/index.html

SQLite Tutorial. (2019, October). Retrieved from What is SQLite: <https://www.sqlitetutorial.net/what-is-sqlite/>

The 5W's of MVVM. (2019, October). Retrieved from Sagitec Blog: <https://www.sagitec.com/blog/the-5ws-of-mvvm>

Understanding the basocs of MVVM design pattern. (2019, October). Retrieved from Microsoft:

<https://blogs.msdn.microsoft.com/msgulfcommunity/2013/03/13/understanding-the-basics-of-mvvm-design-pattern/>