Excalibur: A Heroes Story

11, 2017

SDV602 – Milestone 3

DEAN STANLEY-HUNT

2017

Contents

[Introduction 2](#_Toc498376317)

[Requirements 3](#_Toc498376318)

[Game Description 4](#_Toc498376319)

[Storyboards 5](#_Toc498376320)

[Screen 1: Login Menu 5](#_Toc498376321)

[Screen 2: Register Menu 5](#_Toc498376322)

[Screen 3: Game Menu 6](#_Toc498376323)

[Screen 4: Game Setup 6](#_Toc498376324)

[Screen 5: Game Play 7](#_Toc498376325)

[Analysis 8](#_Toc498376326)

[Use Case Diagram (Milestone 2) 8](#_Toc498376327)

[Use Case Explanations (Milestone 2) 9](#_Toc498376328)

[Login 9](#_Toc498376329)

[Register 9](#_Toc498376330)

[Logout 9](#_Toc498376331)

[Start Game 9](#_Toc498376332)

[Text Input 10](#_Toc498376333)

[Scene Commands 10](#_Toc498376334)

[Class Analysis (Milestone 1) 11](#_Toc498376335)

[Design 12](#_Toc498376336)

[CRC Cards (Milestone 2) 12](#_Toc498376337)

[Class Diagram (Milestone 2) 16](#_Toc498376338)

[Milestone 3 Enhancements 17](#_Toc498376339)

[Database Migration (JSON Drop) 17](#_Toc498376340)

[Game Interaction (Accelerometer) 17](#_Toc498376341)

[Multiplayer GamePlay (MQTT) 18](#_Toc498376342)

[References 18](#_Toc498376343)

# Introduction

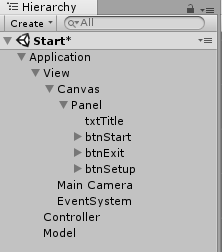
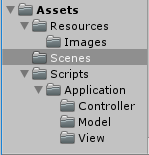
Excalibur is a Text based adventure game targeting the mobile app market to bring back a classic old school gaming feel that crosses a bit between a good old pick a path book and a quest style board game all at once. A player will experience a quest to recover a famous sword and return it to the king, with important decisions to make along the way that will affect the journey, the adventure, the outcome!

The Design Pattern used to develop this game will be AMVCC (Application Model View Controller Component) which is a Game Development specific variation of MVC and will be used in Conjunction with the Game Development Platform known as Unity.

* **Application** - “*This makes it necessary to have a single root reference object, through which all instances in the Application can be reached and recovered.*” (Unity with MVC, 2015) , This essentially means the top of the hierarchy starts with one main umbrella for most of the game components to be structured under, and therefore able to create a singleton instance to make everything visible to everything, through that one structured instance, eliminating scattered class references which can be hell to track or be lost if Unity crashes.
* **Model** – “*Holds the application’s core data and state, such as player health or gun ammo.”*  (Unity with MVC, 2015) , keeping all the games data in one place, or extending from this one instance for the model, makes accessing, updating, saving and loading the game data a lot easier to manage.
* **View** – “*Can get data from Models in order to represent the up-to-date game state to the user*.” And “*Should never mutate Models*.” (Unity with MVC, 2015) , the View can read game data stored in the model to update on-screen elements like a health bar level, and contain all the interfaces game objects along with interaction detection.
* **Controller** – “*Update and use the Model’s data.*” And “Manages Unity’s scene workflow.” (Unity with MVC, 2015) , the Controller contains all the methods and routines required by the view to process interaction, calculate things for visual elements and do things that interact with the Model, so you could say the Controller is a middleman forming a controlled bridge of interaction between Model and View without the two ever directing touching each other.
* **Component** *– “Some elements encapsulate general functionality that should be highly reusable, and which does not naturally fall into one of the three main categories of Model, View, or Controller.”*(Unity with MVC, 2015) *,* the Component part is not seen in the Hierarchy as a physical game object only exist in the game assets and are capable of being reused for multiple game objects. For example, buttons could have a reusable button add listener script that points all button clicks to one other script with an identifier where any buttons actions can be then called upon in the View.

The idea here is that one instance of the game can exist as the **Application**, whereby **Model, View** and **Controller** are branched under it and accessible through it and only feature once each. And **Component** refers to all the reusable scripts that are not uniquely for one game object.

**Model** and **View** depend on each other for the game to work but communicate logic via the **Controller**.

Above you see how Excalibur implements **AMVCC**, with both the physical Hierarchy and Assets Scripts to benefit from the Game Design Pattern even further.

# Requirements

Excalibur is a Text-based 2D adventure game with a player going a quest when the player run’s this App here are functions expected to be offered:

* Login with Username and Password (Coming at next Milestone)
* Change game setup
* Play single or Multiplayer online games
* Exit Game button
* Input text to navigate the story on a quest with many possible paths, pick up items, find items, view current items, save or load and more
* Button to click and submit text entry
* Button to quit game back to the main menu

Excalibur will have 5 Screens upon completion, these are as outlined below:

* Login Screen – To be Implemented for Milestone 2.
* Menu Screen – Player can navigate to change game setup settings, exit the game or start playing a game from here.
* Setup Screen – In the setup area, the player will be able to change any settings such as turning music ON or OFF once such things are implemented at a later Milestone.
* Game Screen – Where all the fun happens with text based game play, an image for the current scene is displayed, all the possible commands at that scene and the storyline for said scene. Here the player requires the ability to input the commands and navigate the story scenes, or be capable to find and pickup items, see an inventory of items or even quit back to the main menu.
* Game Over Screen – When the game comes to an end by either a Death or Completion of the game this screen loads with the appropriate corresponding canvas and allows navigation back to the Main Menu screen

Excalibur requires suitable images that help support the context of each scenes storyline, Custom text styles and main menu artwork to help bring it to life. By the end of this Project, the game will allow players to play the game with background music, using text-based commands or controlled by the Accelerometer sensor and support online multiplayer enjoyment through database login session management and a full chat system showing who is onine in each scene players enter.

# Game Description

Excalibur

A Heroes Story…

Players will represent a knight of the round table, with the story being a quest to find the famous and powerful but currently lost sword of Excalibur belonging to their great King Arthur.

There will be multiple paths to take through caves, forests, rivers, and mountains, encountering bandits, ogre’s and goblins, a dragon and more to defeat or run from.

Once the player finds the sword they must survive the journey back to the castle of Camelot and get the sword into the hands of King Arthur where it belongs, just in time hopefully before the great siege that is coming to Camelot from enemy Vikings.

Dangers at every turn, players must decide when to fight and when to run… but fight at the right time and they might just find the sword! Run at the wrong time and a player may end up searching for the sword a lot longer.

Where multiple players are in the same game, there will be a chat text area to communicate to each other and interact by text to plan the next move at each scene for the game, where any of the players can input the next text command to proceed in the game further individually. With battles, the more players you have to do the quest with you, the more damage attacks to enemies will do!

The game will end once the player finds the sword, returns it to King Arthur and then helps defeat the enemy attacking the castle! Health objects will be able to be gathered during the quest to increase health stats and weapons that will affect damage dealt.

# Storyboards

## Screen 1: Login Menu



The Login screen is where the game first loads to and allows the player to input their username and password to login the game, or they can go off to the Register area and make a new account if not yet registered or exit the game!

* Login Button – Once clicked checks Username and Password are correct then go to Main Start Game Screen. If wrong Login details then show message Failed wrong Name or Password!
* Register Button – Once clicked loads the Register screen where players can Create an Account.
* Exit Button – Once clicked will end the application.

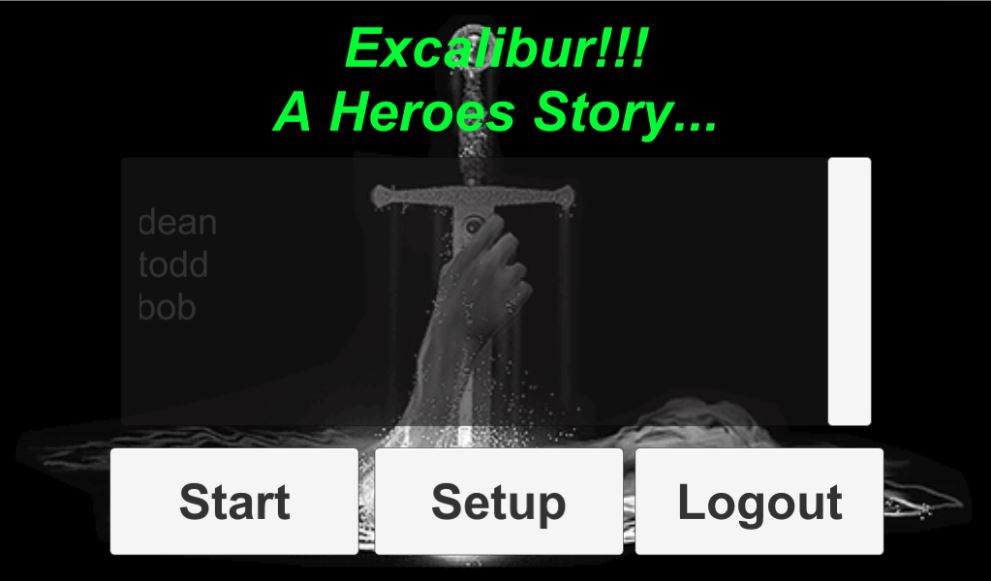
## Screen 2: Register Menu



The Register account screen is where the Player comes to make an account when they don’t have one yet to log in with, it helps the game gather information such as email addresses about the player for possible making contact by email to advertise updates. Once created player can go back and Login!

* Create Button – Once clicked will check if the account name is available and if so register it for the player to be able to then go back and login to the game and start playing!
* Back Button – Once clicked will load back to the Login screen.

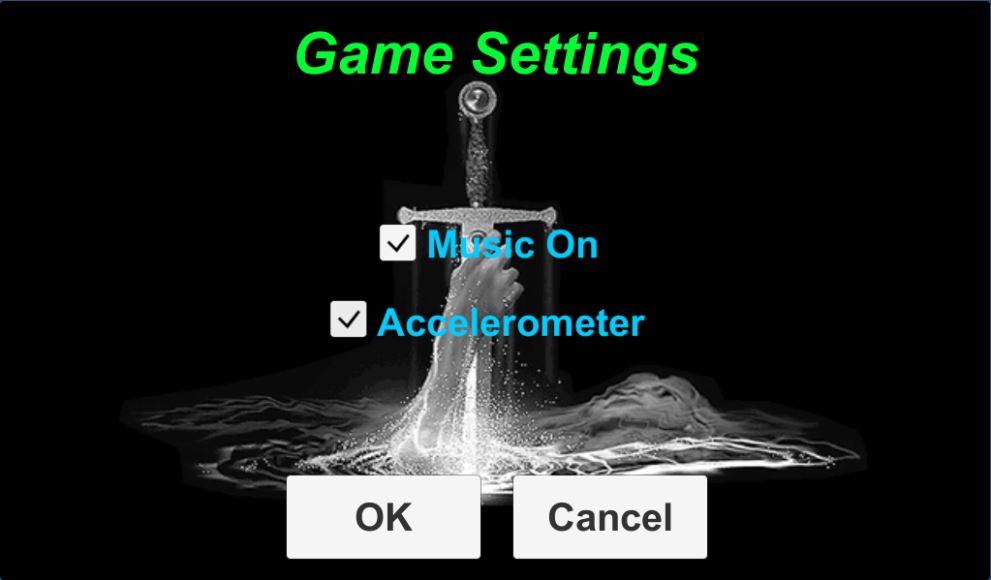
## Screen 3: Game Menu



The Main Menu game screen is where the game arrives after a successful login, this Main Menu allows the user to start playing, set up the game or log out back to the Login Menu where exiting the game is possible. Also visible to players is a List of players online right now.

* Players Online List – A scrollable list of players online in the database
* Start Button – Once clicked will start the game by loading the Game Play Screen.
* Setup Button – Once clicked loads the Game Setup screen where players can change settings.
* Logout Button – Will update player to Offline in the database and change to Login Screen.

## Screen 4: Game Setup



The Game Setup screen allows the player to set personal preferences game music ON or OFF and turning the Accelerometer sensor for Game Interaction ON or OFF! These are both stored in player preferences and loaded each time the setup screen loads so settings are remembered.

* OK Button – Will save the player's setup settings to player preferences and go to Main Menu.
* Cancel Button – Does not save settings changes and goes back to the Main Menu.

## Screen 5: Game Play



The center of all the action, we have the Game Play screen! Here players get to enjoy playing the storyline with images to support the context being set, available commands are shown to the right of the image and players must make the decision about what they want to do at any given story scene or when navigating away from a story scene, most story scenes include the common commands like, BACK, FIND, PICKUP, ITEMS, CHAT while providing unique keywords at each scene which help the game know where to load next. Navigating scenes increases score while going back decreases it!

Milestone 3 introduces Accelerometer interaction ability to scroll left or right through commands and tilt forward to process the command along with a list of players in scene and full chat system support!

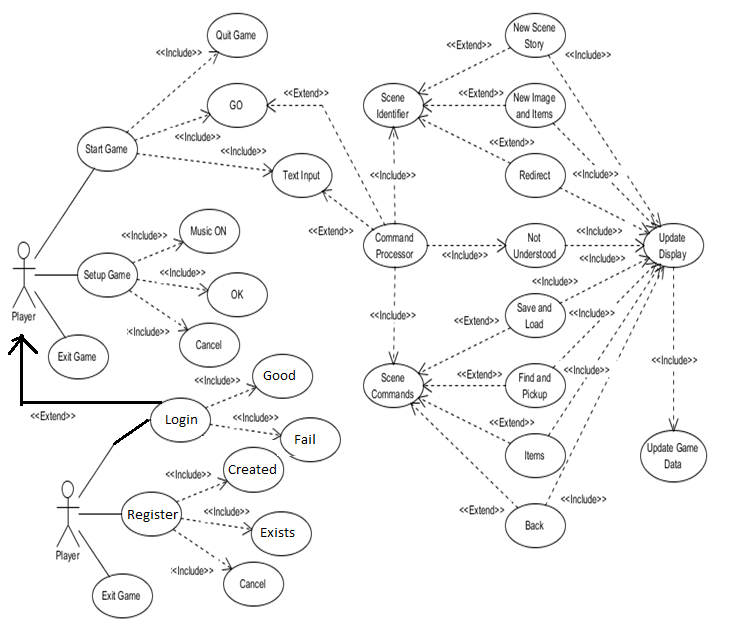
* Image – Shows the corresponding image for the current scene dynamically & automatically.
* Green Text – Always shows the storyline for the current scene, and other data can be displayed with the storyline such as items found, picked up, players items, the game saved or loaded.
* Green Score Text - Shows players current score in the game.
* White Text With Numbers – Shows the available Unique and Common commands for the current scene. Chat is sent using the command “chat/” + the chat text after it!
* Left scroll box shows Players in current scene, Right scroll box shows chat for current scene.
* White Input Text – This is where a player inputs their command decision.
* Quit Button – Player can press at any time to quit back to the Main Menu Start screen.

# Analysis

To perform Analysis others were asked to test the game at various stages of developing milestone 1 and provide feedback that helped it become what it is now at this point, and come up with the games Use Cases to help form the below Diagram and then Explain them all. Lastly finishing up my Analysis by showing a Class Diagram depicting all the classes involved in the game and how they all link with each other.

## Use Case Diagram (Milestone 2)

Use Case Diagram for the Game as is at Milestone 2.



## Use Case Explanations (Milestone 2)

### Login

A Button in the Login Screen that when clicked will verify players login details and if successful goto the Start game screen, otherwise show failed login message.

### Register

A Button in Register Screen that when clicked trys to create an account for the player and if successful shows message to the player that account was created! Otherwise, shows error/name exists message!

### Logout

A Button in Main Menu screen that allows the player to Logout their account back to the Login screen.

#### Setup Game

A Button in the Main Menu screen that when touched will bring the player to the Game Setup screen.

#### Music On

Found in the Game Setup screen a player can toggle the Music On option ON or OFF, Music will be properly implemented at the completion of the project, currently only for show.

#### Ok & Cancel

Found in the Game Setup screen either of the two buttons when touched go back to Main Menu screen.

#### Exit Game

A Button in the Login screen that when touched will shut down the application.

### Start Game

A Button in the Main Menu screen that when touch will bring the player to the Game Play screen.

#### Quit Game

Found in the Game Play screen when touched by a player will leave game returning to the Main Menu.

#### Not Understood

The Use Case active when players submit incorrectly input commands. Do Not Understand is returned.

### Text Input

Found in the Game Play screen when touched by a player will display an input keyboard on screen for mobile which once the player has to input a command, can be submitted for command processing.

#### Command Processor

The primary Controller method that handles the logic to the Game deciding and actioning what happens from any input command such as Not Understood, Scene Identified and other Scene commands. The default response is that the command was not understood. This Use Case is activated once any Input text is submitted.

#### Scene Identifier

One possible Use Case via the command processor is the Scene Identifier process where we identify if a command input was the command to navigate to another scene.

#### New Scene Story

If the Scene Identifier identified positively, the storyline, image, and available commands text updates.

#### Redirect

If the Scene Identifier identified positively, and the Identified next scene to load has, a redirection item and redirection scene! It will check if that redirection item exists in players inventory and if it does, change the next scene to load into that redirection scene! This Use Case allows the game to have alternative locations or scenarios for when navigating to any scene, that if the player holds a unique item they should go to this other scene story instead. After all, the game cannot be won if the player does not eventually carry the Excalibur into Camelot and to the King.

### Scene Commands

This the Use Case activated by Input Commands that are purposed for being mostly available at every scene, it identifies what common command was input and does the job, for example **BACK, FIND, PICKUP, SAVE** and more.

#### Save & Load

These Scene Command Use Cases are designed to save the Games Model Data state or load a previously saved game data state. These features provide the player with the ability to save and reopen the game continuing from where the player got up to after simply typing **SAVE** or **LOAD**.

#### Find & Pickup

These Scene Command Use Cases are designed to **FIND** and display any Items for the current scene and list the 1 or more items found in any given scene which does not already exist in the player's inventory. For any item found the player can type **PICKUP** + **Spacebar** + **Item name** to add the item to their Inventory. For example, ‘Pickup Stick’.

#### Items

This Scene Command Use Case is designed to display the player's current Inventory Items List.

#### Back

This Scene Command Use Case is designed to return a player to the previous scene they just came from and with the games unique scene history log a player can type back over and over retracing their steps through the game as far back as the player likes.

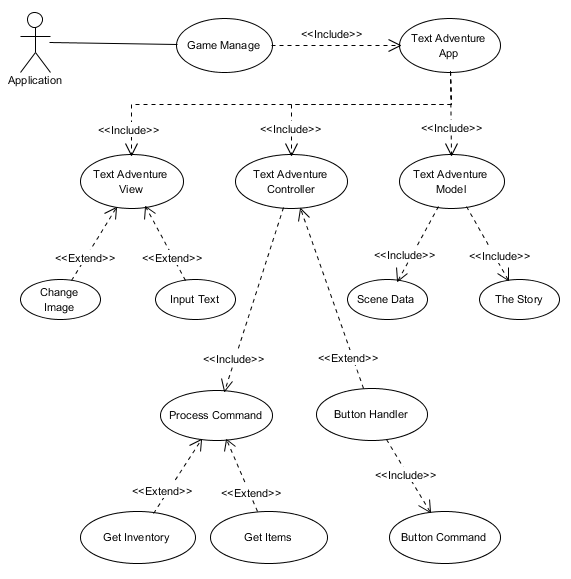
#### Update Display

The Use Case most associated with VIEW from AMVCC, the command processing that all command types derive from is the logic in the CONTROLLER, that when one is executed this Use Case is the CONTROLLER telling the VIEW to update.

#### Game Data

The last relevant Use Case that details everything about the current game state, the CONTROLLER interacts updating the game state for any changes triggered by the player commands. For Example, it is what gets saved or loaded into when SAVE or LOAD are used.

## Class Analysis (Milestone 1)



# Design

## CRC Cards (Milestone 2)

|  |  |
| --- | --- |
| **Class: GameManage** |  |
| **Responsibilities:**  Startup when gameplay screen loads and instantiate all its subclasses it handles and manages to provide structured MVC and access to other class scripts during gameplay as required.  Shutdown when gameplay screen is exited. | **Collaborators:**  TextAdventureApp |

|  |  |
| --- | --- |
| **Class: TextAdventureApp** |  |
| **Responsibilities:**  Instantiate within GameManage and Instantiate and provide access forward to the Model, View and Controller classes! | **Collaborators:**  TextAdventureModel  TextAdventureView  TextAdventureController |

|  |  |
| --- | --- |
| **Class: TextAdventureModel** |  |
| **Responsibilities:**  Instantiate and provide access for the gameplay to store data requirements locally.  Provide current room information  Provide Players Score  Provide List of Scene Identifiers  Provide List of Players Inventory | **Collaborators:**  DTOs / Scenes |

|  |  |
| --- | --- |
| **Class: TextAdventureView** |  |
| **Responsibilities:**  To provide access/interaction to GUI Elements and GameObjects, like creating submit input listener to the Input Field text box for example. | **Collaborators:**  ButtonHandler  ChangeImage |

|  |  |
| --- | --- |
| **Class: TextAdventureController** |  |
| **Responsibilities:**  To Instantiate and create the extension to an instantiation of the Process Command class for controlling the game.  To instantiate an extension to the Data Service class for database interactions through gameplay. | **Collaborators:**  ProcessCommand  DataService |

|  |  |
| --- | --- |
| **Class: ProcessCommand** |  |
| **Responsibilities:**  To Process the input command of a player!  To control visual and data manipulation/events  To Identify scenes and gather new scene data from database  To handle in scene events such as find items, or show players items  To provide game navigation and a scoring system logic | **Collaborators:**  DTOs / Scenes  GetItems  GetInventory |

|  |  |
| --- | --- |
| **Class: GetItems** |  |
| **Responsibilities:**  To be used by the ProcessCommand script to check if scenes have items, if items exist, and get items ready for output to display. | **Collaborators:** |

|  |  |
| --- | --- |
| **Class: GetInventory** |  |
| **Responsibilities:**  To be used by the ProcessCommand script to check if scene items exist in players inventory, and get inventory ready for output to display. And process the adding new item to players inventory. | **Collaborators:** |

|  |  |
| --- | --- |
| **Class: Player / Scenes / PlayerInventory** | **DTOs.cs** |
| **Responsibilities:**  These are Data Transfer Objects which behave as Database Table structures and local object classes for C# usable to interact with Data from the Database locally, and create tables or interact with the database.  They form a structure of data, in database terms the attributes of a single table, for c# they are the variables of a class object.  Player responsible for Player credentials and online status.  Scenes responsible for the story of the game to be used via database.  PlayerInventory not yet implemented to the database but will be in milestone 3 for handling players inventory live in online games. | **Collaborators:**  SqLite4Unity3d / SQLite |

|  |  |
| --- | --- |
| **Class: DataService** |  |
| **Responsibilities:**  Provides control between the database and the application, to create the database if it doesn’t exist for cross platforms, build the game story into database and tables if no data or tables exist, add test data for user accounts, and provide direct access to perform all aspects of CRUD with the database, Create, Read, Update, Delete. | **Collaborators:**  SqLite4Unity3d  TheStory  DTOs / All |

|  |  |
| --- | --- |
| **Class: TheStory** |  |
| **Responsibilities:**  Only ever Instantiated if creating the database fresh, it holds all the story and builds a list of scenes locally which is then used to add all scenes to the database before killing this instantiated class never to load again unless database erased/missing/corrupted. | **Collaborators:**  DTOs / Scenes |

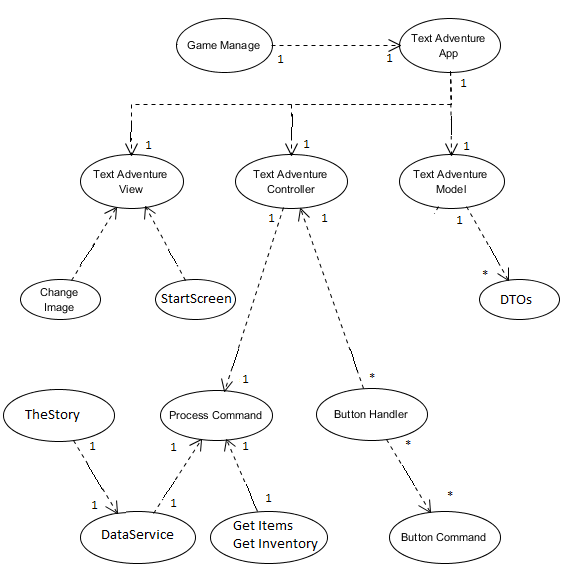
|  |  |
| --- | --- |
| **Class: ButtonHandler** |  |
| **Responsibilities:**  Reusable script for attaching to any button at any scene and handling its command type according to determined public string variable set. | **Collaborators:**  ButtonController |

|  |  |
| --- | --- |
| **Class: ButtonController** |  |
| **Responsibilities:**  Identifies what the command for the pressed button is to correctly call the relevant method associated with the command.  Execute the method for that button pressed, manipulating data and visual elements as required on a method by method situation. | **Collaborators:** |

|  |  |
| --- | --- |
| **Class: ChangeImage** |  |
| **Responsibilities:**  Starts Up when game play screen loads linked to the image game object as a mono behavior script that reads what the current scene is from model, and whenever the image name doesn’t match the currently loaded images name, it will change the background image matching current scene, doing so with the UPDATE method per screen frame. | **Collaborators:** |

|  |  |
| --- | --- |
| **Class: StartScreen** |  |
| **Responsibilities:**  When Start Game Menu screen load this script is attached to the application layer game object loading when the screen loads.  Its purpose is to hook into the list display on the screen and access the database for all currently online players and then display them to the GUI Scrollable list object.  Allows players to know whos online. | **Collaborators:**  DataService  DTOs / Player |

## Class Diagram (Milestone 2)



# Milestone 3 Enhancements

A lot has been accomplished for milestone 3 but before diving into the details I would like to re-iterate that this game since the start has been built using the AMVCC game orientated Design Pattern as illustrated at the beginning of this document in the Introduction, that is Application, Model, View, Controller, Component. Some script may look like they should be Model-View from the MVVM Design Pattern but in the AMVCC the Model-View is known as the Controller, and some scripts such as the Button Handler script are re-usable and seen under View for the game because they attach to visible Game Objects on the View such as buttons, but these should be recognized as Components!

## Database Migration (JSON Drop)

The database previously used SQLite for local login and online session management, for milestone 3 the database has been migrated to an online service JSON Drop bringing the Players accounts and story scenes to the next level with session management being handled live online, players playing the game from any location can now log in and see other players that are online!

No changes have been made to the table structures for players or scenes data, only the location being stored from locally to online making the game have a true online multiplayer feel, with all data for the game scenes being grabbed from the database whenever a player starts playing the game!

A lot was learned and gained through this migration and it took a lot of smoothing out and problem-solving in conjunction with testing to get this feature working perfectly.

## Game Interaction (Accelerometer)

For game interaction, the Accelerometer sensor has been implemented to provide full control of the game if desired other than for chatting in the new chat system. Using 3 tilt direction detection the game can be controlled scene to scene, finding or picking up items, saving or loading and more!

Tilting Left or Right cycles forward or back at a controlled pace through the available commands for any given scene placing that command into the input text field for the player and once the command a player desires to process is showing a simple forward tilt all the way slightly forward towards the players body when phone becomes completely vertical will send the command to be processed!

This makes gameplay a lot faster and more enjoyable than typing out commands but for those who don’t desire the use of this feature, there is the ability added to the games setup menu to disable the Accelerometer and enjoy the original feel of this great Text Based Adventure Game, Excalibur!

Implementing the detection of the X and Z axis from the Accelerometer was not that difficult but what certainly was is testing it out to figure out how much tilt force it needed to be collaborated for with both left and right, then again for forward tilting. To make things even more tricky was the difference for the Z-axis input between an actual Android mobile phone and the PC testing station Blue stacks!

## Multiplayer GamePlay (MQTT)

To support an Online Multiplayer game feel, the JSON Drop database could have been used but was very slow to respond and doesn’t have the great broadcast feature such as MQTT provides. MQTT is part of the Internet of Things (IoT) known in longhand as Message Queue Telemetry Transport!

MQTT was used to support showing exactly what players are inside any single given scene at any point in time, separately scene by scene and at the same time using the exact same MQTT subscription to manage and handle full chat system between players in that scene! Where players can send chat to each other using the command “chat/” + the message they desire to send appended.

To enhance and take advantage of the chat system further it has been set up to show who joins the scenes chat, what item a player picks up if any, and when play leaves… what command they used! Allowing for friends or other players to follow the same path without any need to communicate where a player intends to go or the command they are going to choose prior to selecting it and changing scene.

So it is a great way to play online as a multiplayer environment, with full co-op capabilities, players can play with a friend or few and chat first about what command they want to play next and then as a group navigate scene by scene, or simply play a game of following the leader!

It took a lot of testing to get it all working but you are sure to be happy with the results so enjoy!

# References

Bjork, R. C. (2008). *ATM Example*. Retrieved from www.math-cs.gordon.edu: http://www.math-cs.gordon.edu/local/courses/cs211/ATMExample/

Costa, E. D. (2015, March 07). *Unity with MVC*. Retrieved from toptal.com: https://www.toptal.com/unity-unity3d/unity-with-mvc-how-to-level-up-your-game-development