



# Digital Media Final Project Report

MSc Game Development (Programming)

Student Name: Gordon Johnson

K Number: 1451760

Project Title: Virtual University RPG

Supervisor: Vasileios Argyriou

The type of project is a BODY of WORK

**Kingston University** London

## **WARRANTY STATEMENT**

This is a student project. Therefore, neither the student nor Kingston University makes any warranty, express or implied, as to the accuracy of the data nor conclusion of the work performed in the project and will not be held responsible for any consequences arising out of any inaccuracies or omissions therein.

Gordon Johnson

K1451760

## Table of Contents

Project Links .....	3
Abstract .....	3
Introduction and Background .....	3
Analysis.....	4
Stakeholders .....	4
Methodology .....	5
Technologies .....	6
Game Engine .....	6
Integrated Development Environment.....	6
Version Control .....	6
Third Party Assets .....	6
Requirements.....	7
Functional Requirements .....	7
Non-Functional Requirements.....	7
Design .....	8
Wire Frames .....	8
Login UI .....	8
Administration UI.....	13
Student UI.....	20
Technical Documentation .....	28
Entity Relationship Diagram.....	28
Class Diagram .....	29
Dependency Graph.....	30
Implementation .....	31
uMMORPG .....	31
School Scene.....	32
Integration of assets .....	32
Custom Avatars .....	33
Camera Position.....	34
Mobile UI.....	34
Administrator Implementation .....	35
Quiz Implementation.....	38
Lecture Implementation .....	43

Database Implementation .....	47
Evaluation .....	54
Future Improvements .....	54
References .....	55
References .....	55
Technical References .....	55
Asset References .....	56

## Project Links

Box URL: <https://kingston.box.com/s/zqp9vq0ywyyp37t6jv5x22xziudpzfdaf>  
YouTube URL: [https://youtu.be/\\_6Pqcjq3WkY](https://youtu.be/_6Pqcjq3WkY)  
GitHub URL: <https://github.com/LordGee/VirtualUniversity>

## Abstract

The project is to create a Virtual University game, incorporating features from the Role-Playing Game genre, based on research conducted by (Argyriou et al., 2010). This will be a Massive Multiplayer Online game, allowing many students and academics to simultaneously be immersed within the world. The initial purpose of this Virtual University will be to provide online Lectures, Workshops and Quizzes, within a fun and exciting environment. The artefact will be developed to a high-level, utilising existing libraries and assets where possible. (Johnson, 2018)

## Introduction and Background

The aim is to develop a gamified University environment in the form of a Role Player Game, inspired by previous research conducted by (Argyriou et al., 2010) ‘Virtual university as a role-playing game’. It will be a Massive Multiplayer Online experience, allowing many users to log into the world simultaneously. Since this report was published, technology requirements have changed, making it necessary to build a mobile platform, thus extending the accessibility of this product.

The Virtual University would primarily target University Lecturers and their Students. With lecturers taking on the role of administrators, allowing them to add content and activities, while students participate in the activities created.

This project would provide huge benefits to both lecturers and students, including, access for lecturers to virtually contribute to quizzes, lectures and workshops and an expanded geographical reach for potential student enrolment. Students will be able to participate in a social and interactive game world, within a gamified learning experience, regardless of their personal circumstances or physical location.

Whilst various gamified learning experiences exist, very few have opted for an RPG style game world to host in. A popular learning tool is (FrogPlay, 2018) which provides quizzes and assignment allocation, in a gamified environment and motivates students through games and rewards.

A more recent development is Minecraft Education Edition, which provides a code learning tool in the Minecraft Universe. This includes the creation of labs such as the Chemistry update, which provides the building blocks of matter, combining elements into useful compounds, from which lessons can be created in a safe learning environment. (Mojang, 2018)

## Analysis

### Stakeholders

Stakeholders are the entities affected by this product.

Entity	Description
<b>University</b>	Provided with a new and exciting service to offer their members (staff and students alike). With the potential of extending the Universities Geographical reach.
<b>Academic Staff</b>	Able to create content, quizzes, lectures and workshops whilst monitoring student progress. Will provide an additional method of communication between staff and students.
<b>Students</b>	Able to receive content, partake in lectures and workshops, communicate with academics in real time and socialise with their peers, regardless of their personal situation or location.

## Methodology

A variation of an Agile Scrum methodology will be used, utilising Kanban boards to manage ongoing activities. Kanban boards are primarily a team management tool, despite this being an independent project, it will still be beneficial in the management of the project. The solution chosen to deliver this tool is (Taiga, 2018) which will aid in task-based management, the board consists of five main columns (New, Ready, In Progress, Ready for Test and Done), plus an additional column to hide older archived items.

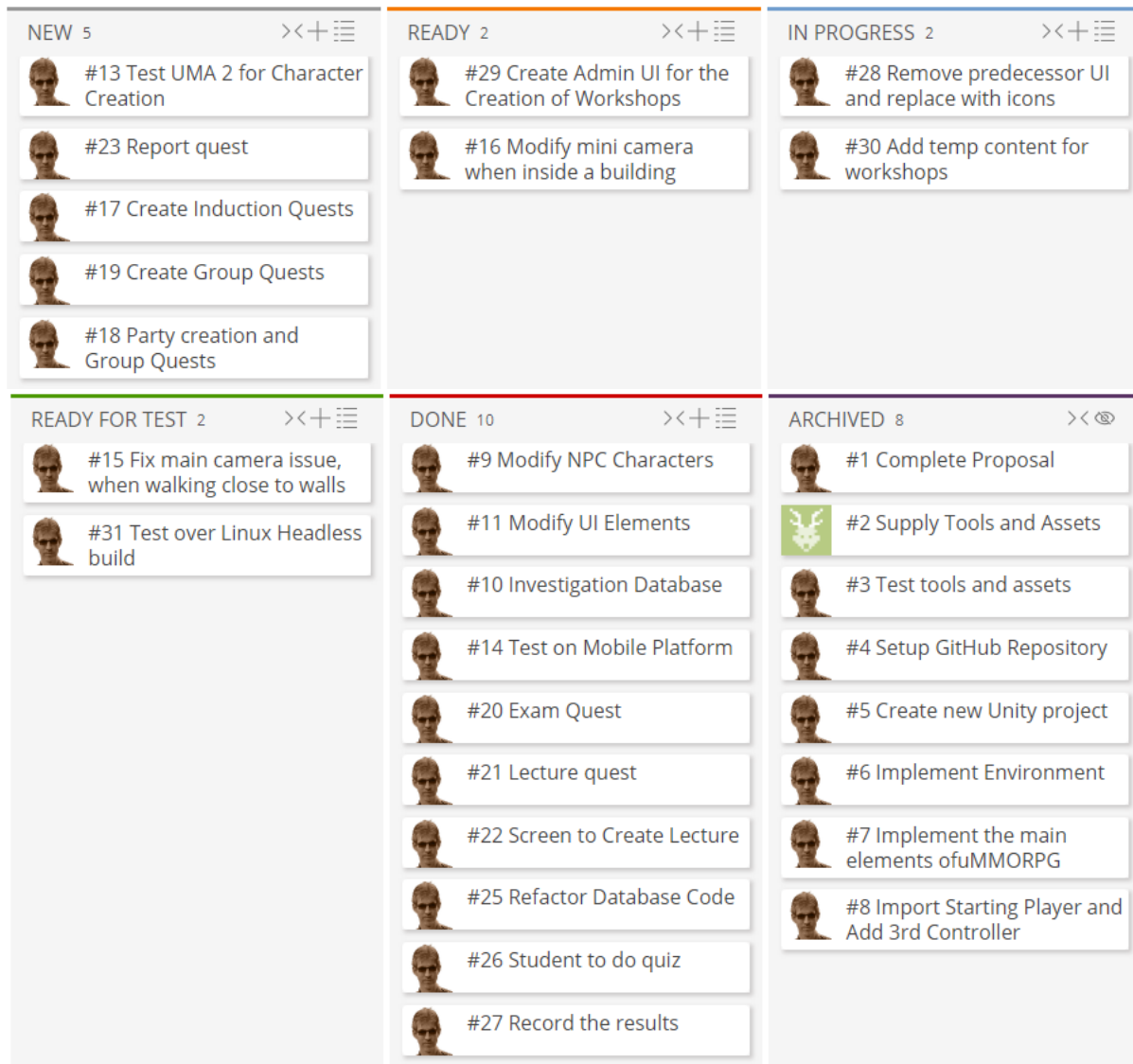


Figure 1 - Kanban Board

## Technologies

The following defines the tools and versions that will be used for this project.

### Game Engine

The main development tool will be Game Engine, the selection of this tool was based on experience and suitability. The decision was made to utilise the Unity 3D engine (Unity 3D, 2018), which the project team has the greatest exposure and experience with. The latest version at the start of this project was Unity 2018.1.0f2 and throughout the project has successfully been upgraded to Unity 2018.2.7f1. In the project proposal, it was decided not to use patch updates throughout the project so as not to lose certain features support.

### Integrated Development Environment

The IDE utilised throughout this project, is Visual Studio Enterprise 2017 IDE (Microsoft, 2018), this will be used for C# coding. The version at the start of this project was 15.6.7 regular updates throughout the lifecycle have been applied through to version 15.8.4. A plugin for Visual Studio will be utilised, JetBrains ReSharper Ultimate (JetBrains, 2018), providing useful features such as improved intellisense, auto include namespaces and quick refactoring.

### Version Control

GitHub (GitHub, 2018) will be the Version Control solution, utilised for this project. Commits will be actioned on a regular basis to the repository, all commits will be error free, allowing successful builds to be recreated at every successful commit. This Git facility is a great tool for keeping track of the development history and provides a contingency in case of any unforeseen issues.

### Third Party Assets

The two main assets that will be implemented into this project, will be supplied by the project supervisor (Vasileios Argyriou). The first asset is the uMMORPG (uMMORPG, 2018) which provides a starting structure / template for a Massive Multiplayer Online Role-Playing Game. Providing a suitable Network Manager, which handles the connection between a Server and multiple client users and a standard RPG style User Interface. The second asset is a 3D Model environment of a school (Tirgames, 2018), including offices, lecture theatre and surrounding areas, this will form the Virtual University.

## Requirements

The following requirements define the project aims and how to achieve them.

### Functional Requirements

#### *All Users:*

- Able to register a new standard account.
- Create an avatar profile during registration.
- Can navigate the Virtual University world.
- Can use the in-game chat function, to communicate with all or individual users.

#### *Admins (Academic Staff):*

- Create and edit quizzes.
- Create and edit lectures.
- Create and edit workshops.
- View and give feedback on student submissions.

#### *Super Admins (Product Owner):*

- Can upgrade standard User accounts to an Admin account.

#### *Students:*

- Partake in Quizzes relevant to their Course / Subjects.
- Partake in Lectures relevant to their Course / Subjects.
- Partake in Workshops relevant to their Course / Subjects.
- View feedback received from Admins.

### Non-Functional Requirements

- Windows 10 Compatible.
- Android SDK, compatible with version 7 and Higher.
- Linux Ubuntu Server 18.04 (for hosting a Headless Build).
- Active internet connection required.



## Design

### Wire Frames

Utilising Adobe XD CC, the following wireframes have been designed as a rough guide to illustrate the layout and sequence of events. As the target platform is mobile devices, there is a limited offering of input and selection options on each screen, this provides additional space for a larger selection area of elements and buttons.

*Note: Layout decisions may change over the lifecycle of the project.*

### Login UI

When launching the application, the user will be presented with a Login Screen.

#### Step 01 – Login Screen

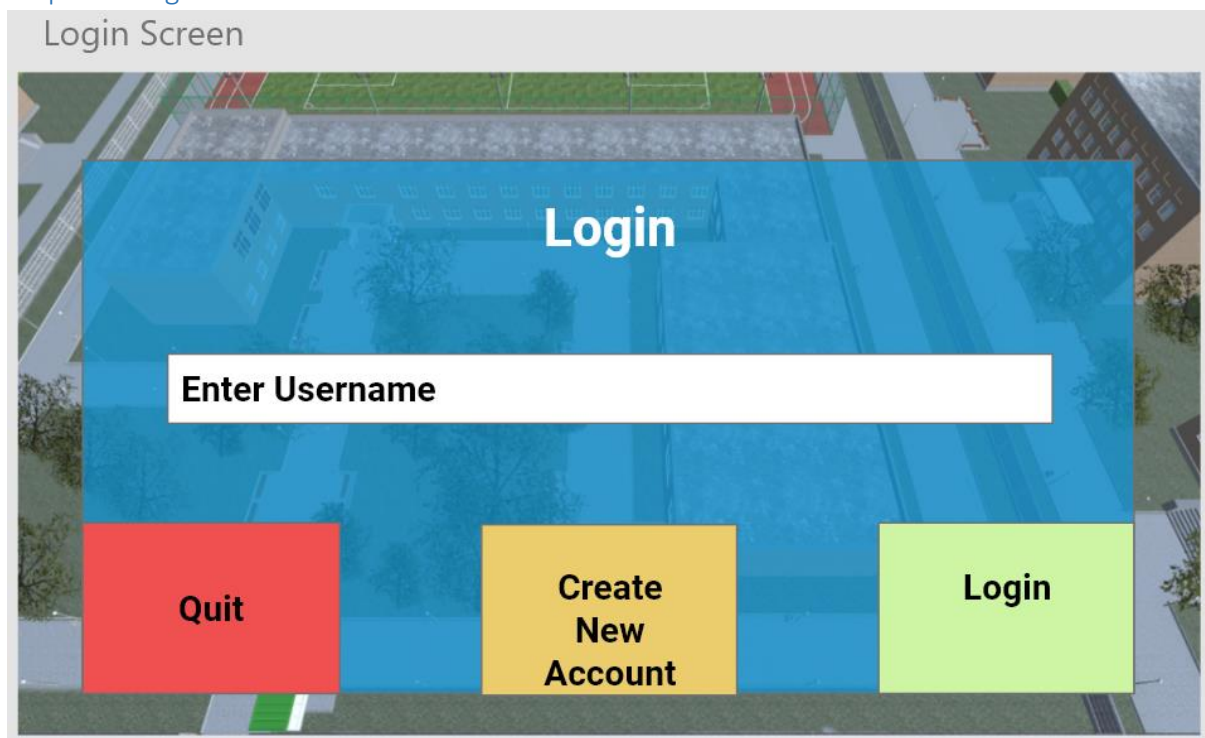
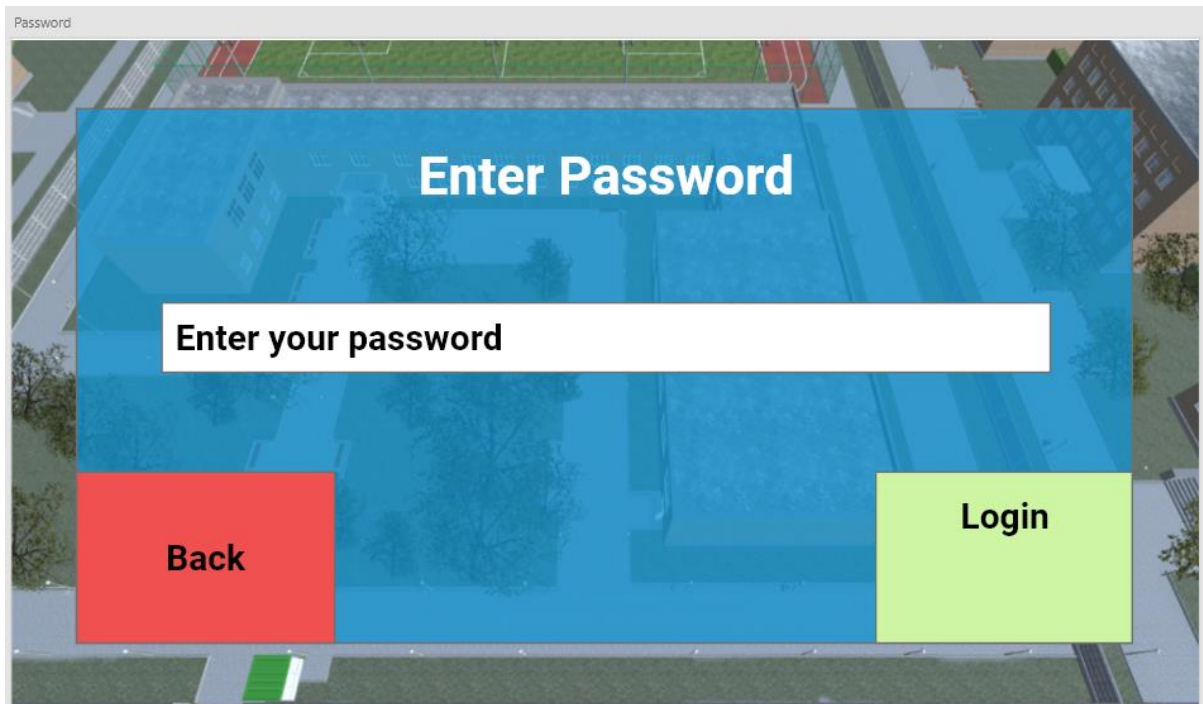


Figure 2 - Step 01 - Login Screen UI

The user is prompted to enter their username into the input box. The user can then select either to create a new account or login to an existing one. A quit option is also available to close the application.

Selecting either Login or Create New Account will take the user to the Enter Password page. Wording should change depending on the option selected.

## Step 02 – Enter Password Screen



*Figure 3 - Step 02 - Enter Password Screen UI*

The user proceeds to enter either, their previously chosen or desired password. Upon completion, the Login button (or Register button) should be selected, this will take the user to the next screen which is Server selection.

## Step 03 – Select Server Screen

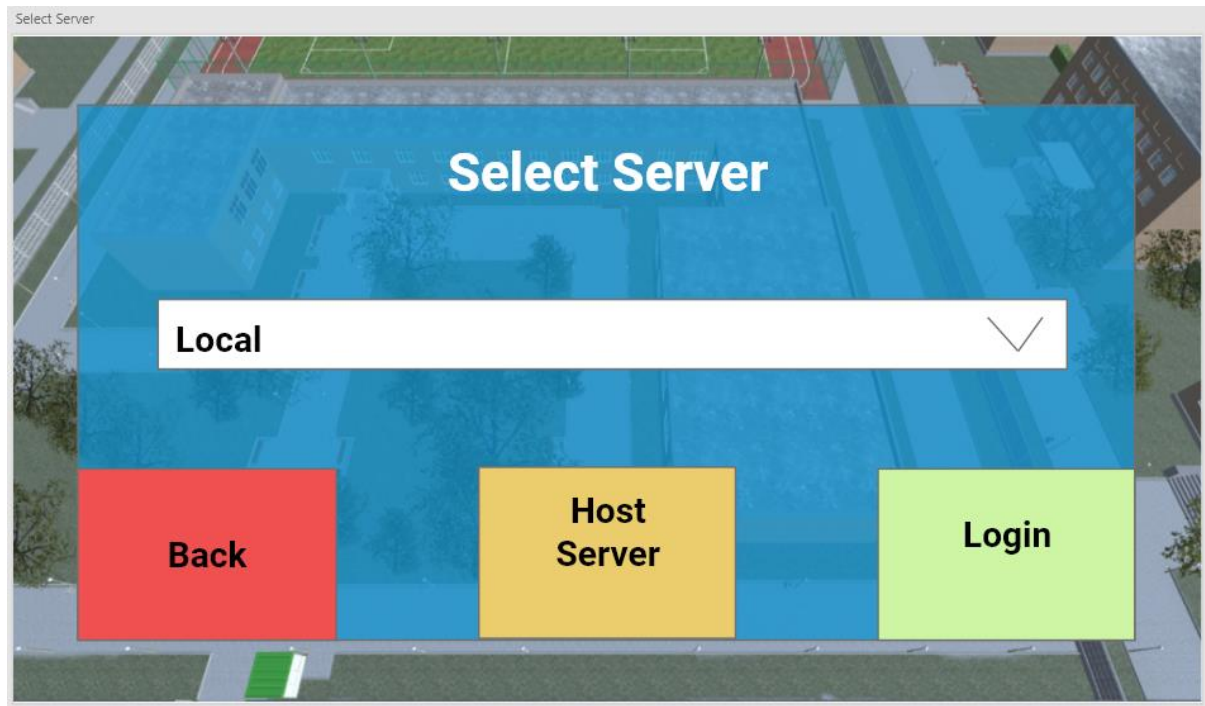


Figure 4 - Step 03 - Select Server Screen UI

The Server Select screen is a temporary placement, until a live server with a fixed IP address is available. Selecting Host Server, allows the user's computer to act as both server and client.

If the Login button is selected, a connection will be attempted on the selected server, if no connection can be established, a timeout will occur, and the connection aborted.

*Note: Once there is a dedicated server this screen will be obsolete, with connection established automatically.*

## Step 04 – Select Course Screen (Only whilst registering a new account)



Figure 5 - Step 04 – Select Course Screen UI

The Select Course screen only appears during the registering of a new account. The user selects their course from the drop-down options.

## Step 05 – Select Character Screen

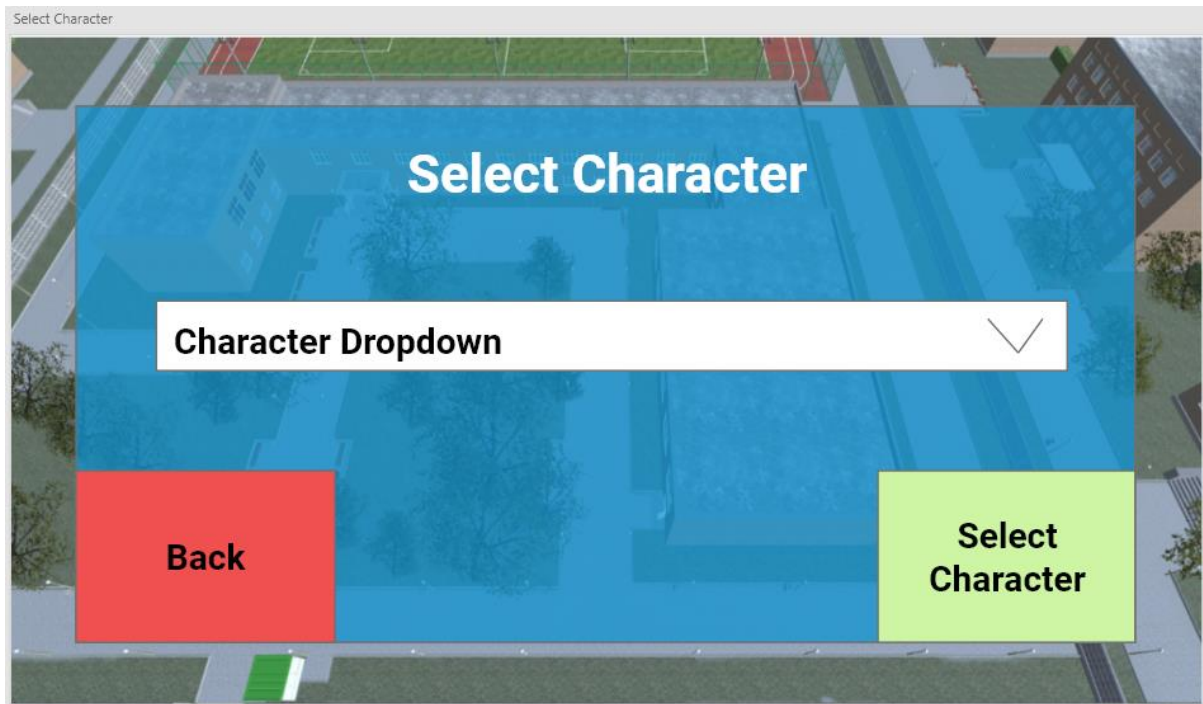


Figure 6 - Step 05 – Select Character Screen UI

The character selection allows users to select, from pre-defined avatars, which one will represent them within the game.

*Note: Long term a character creation option will be added, allowing users to name and define their avatars appearance.*



### Administration UI

The main administration is only available to users with Admin Account status. It can be reached with any of the Non-Playing Characters (NPC) within the game world.

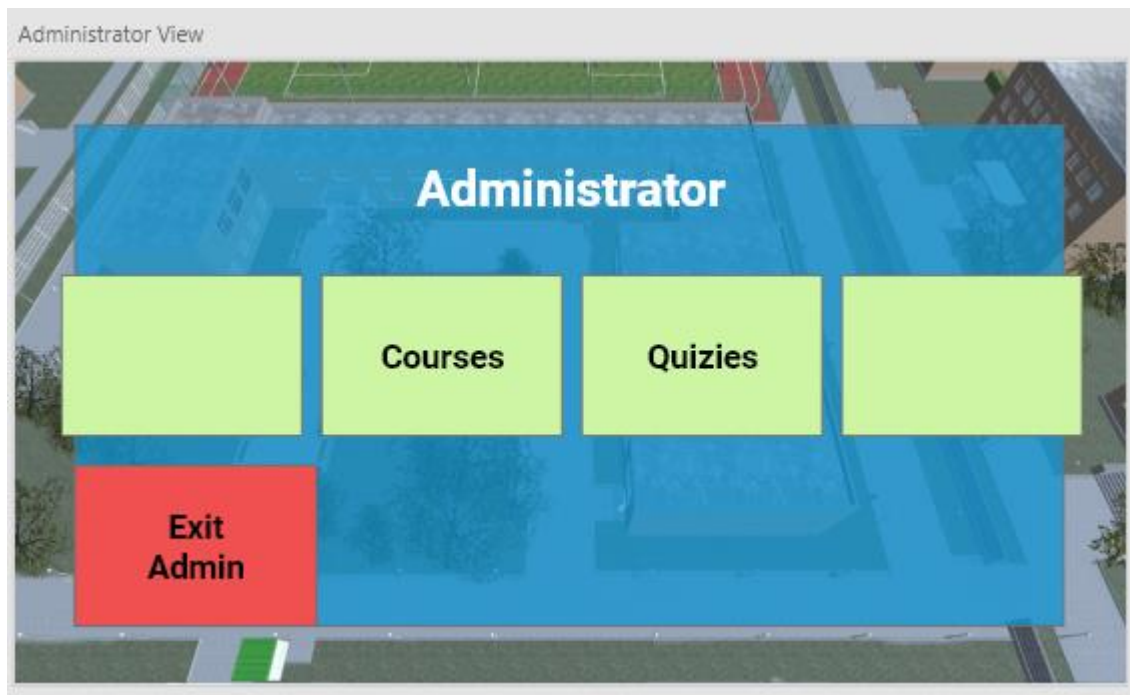


Figure 7 - Administration View UI

The Administrator view provides options that are only available to users with an administration profile. These options allow the admin to manage Courses, Quizzes, Lectures and Workshop content.

### Course Management

Due to Quizzes, Lectures and Workshops being course / subject dependent, it is important to ensure new courses can be added to the system and once allocated, allow for subjects to be assigned to individual courses.

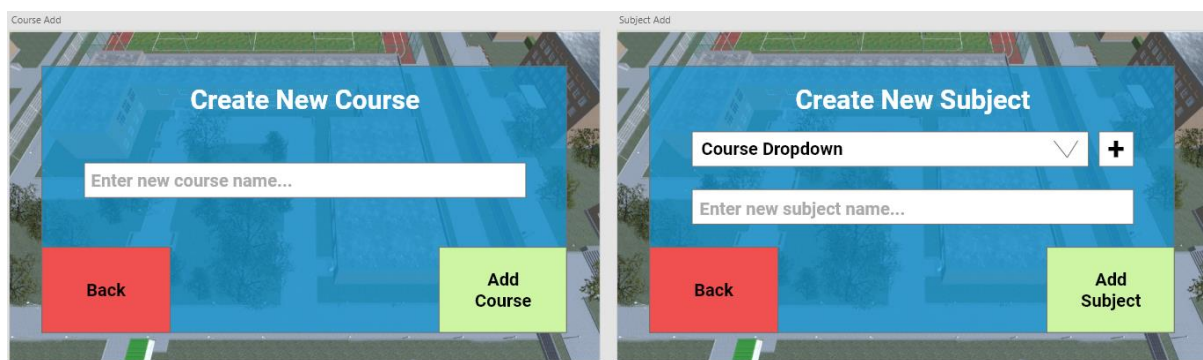
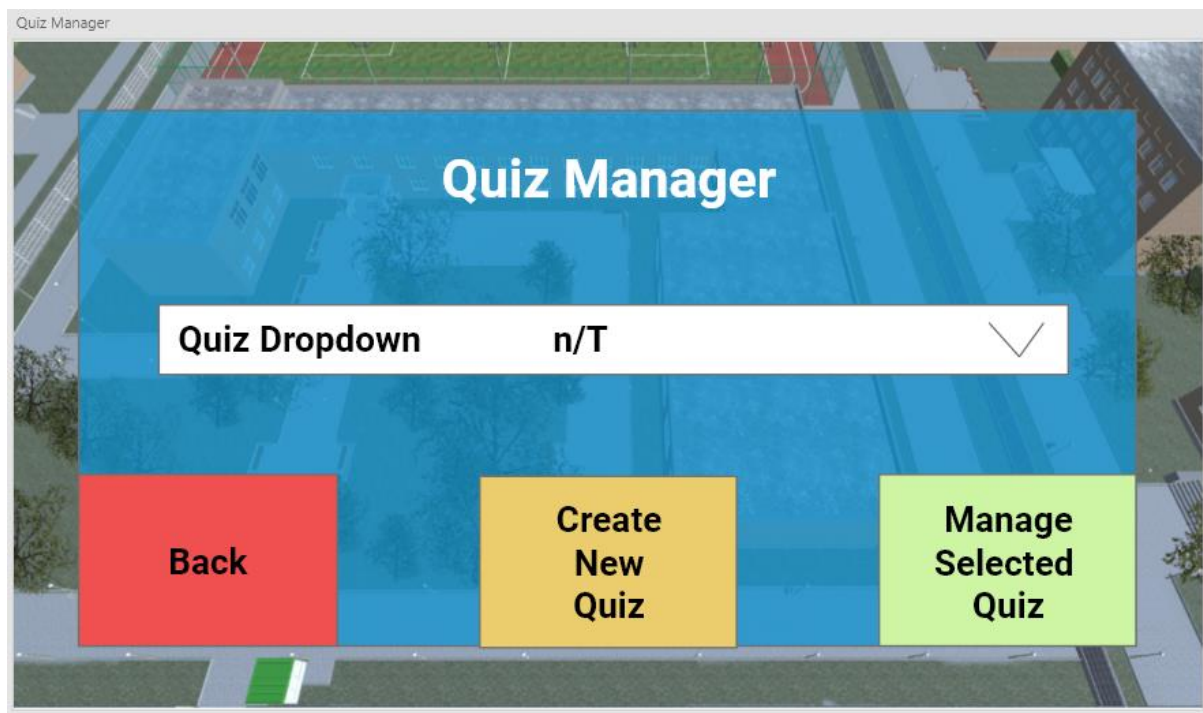


Figure 8 - Course Management UI

The admin can add a new course to the database, by entering the course name into the input box, after which an option to add subjects can be applied to individual courses.

### *Quiz Management*

When entering the Quiz Manager section, the following management view will be presented.



*Figure 9 - Quiz Management UI*

The Admin will be presented with a drop-down option box, for any quizzes they have created. Allowing them to manage/amend the selected quiz. An option to create new quizzes is available, the following user interface steps describe this process.

## Step 01 / 02 – Select Course and Subject

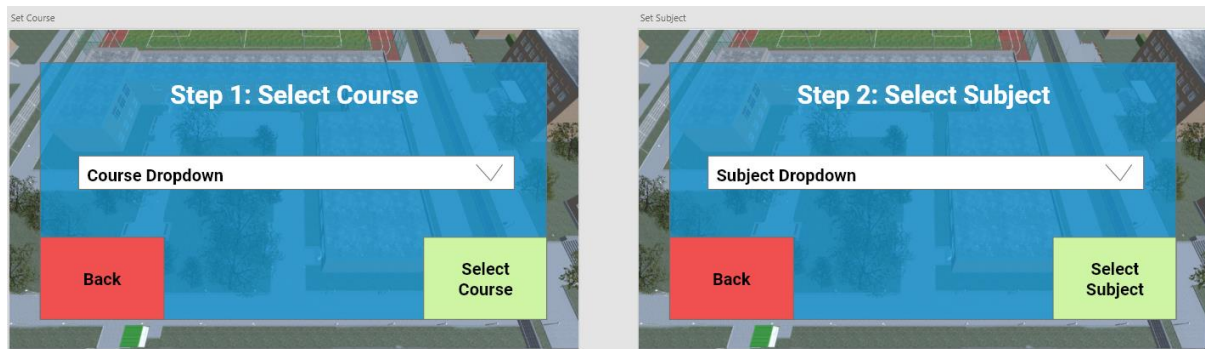


Figure 10 - Step 01 / 02 – Select Course and Subject UI

The first steps when creating a new quiz, will be to define the course and subject that it relates to, by selecting the relevant information in the drop-down options.

## Step 03 – Set a Quiz Name



Figure 11 - Step 03 – Set a Quiz Name UI

The next step is to provide a unique name for the quiz. If the name is already in use, an error message should be returned asking the user to choose a new name.



#### Step 04 – Set the Question

The following screens will iterate through, until the user exits the process. This allows multiple questions to be applied one after another.



Figure 12 - Step 04 – Set the Question UI

This step allows the user to define a question for the desired quiz, once implemented each question can be modified via the initial Quiz Manager screen.

#### Step 05 / 06 – Add correct and Incorrect Answers

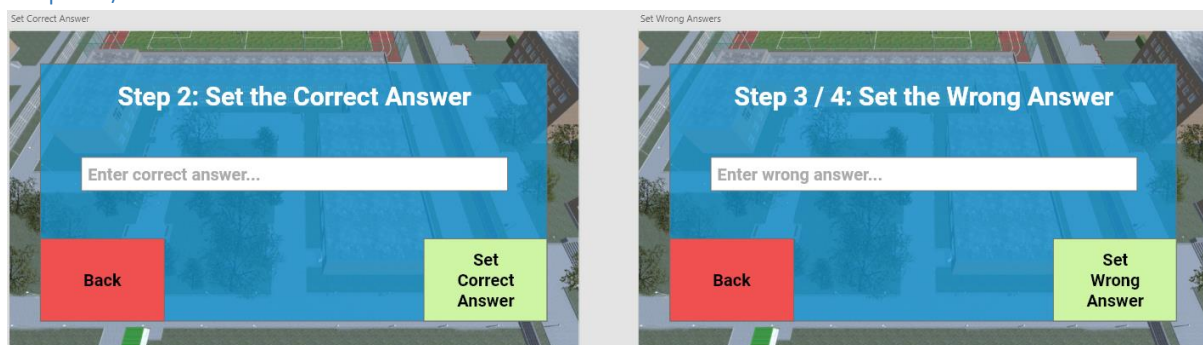


Figure 13 - Step 05 / 06 – Add correct and Incorrect Answers UI

Initially the tool prompts the user to input the correct answer first, it then asks for three wrong answers. As with the questions, these answers can be amended later. Once set, the process loops back to Step 04.

### *Lecture Management*

Currently a pre-requisite for implementing a new lecture, is to ensure a video is already uploaded to a suitable web hosting, enabling the user to supply a direct link URL to the MP4 or AVI file.

#### Step 01 - Enter Lecture Title


The image shows a mobile app interface for adding a new lecture. At the top, there is a header bar with the text "Add New Lecture". Below this, a large blue semi-transparent overlay covers the screen. Inside the overlay, the text "Enter Lecture Title" is displayed in white. Below the text is a white input field with the placeholder text "Enter Lecture Title...". At the bottom of the overlay, there are two buttons: a red button on the left labeled "Back" and a green button on the right labeled "Add Lecture". The background of the app is a 3D architectural rendering of a city street with buildings and a park.

Figure 14 - Step 01 - Enter Lecture Title UI

The first step will be to provide the new lecture with a unique lecture title. If the title already exists, an error message should be returned requesting a unique title.

#### Step 02 / 03 – Select Course and Subject

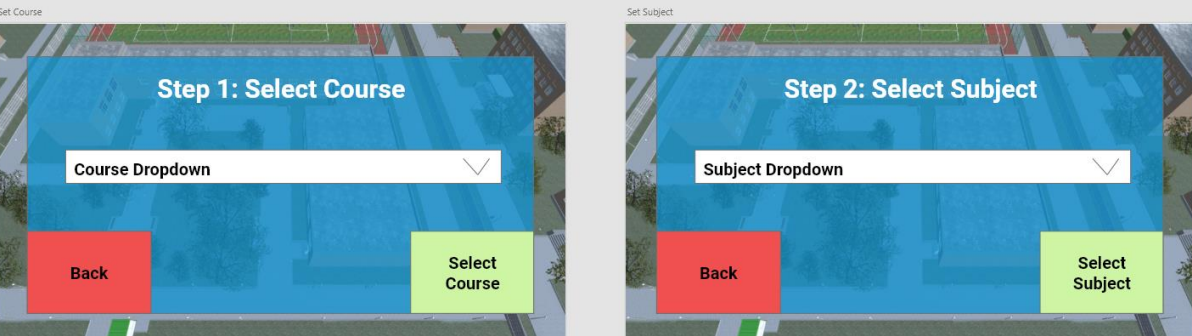
The image shows two side-by-side mobile app screens. The left screen is titled "Set Course" and displays "Step 1: Select Course". It features a white dropdown menu labeled "Course Dropdown" with a downward arrow. Below the dropdown are two buttons: a red "Back" button and a green "Select Course" button. The right screen is titled "Set Subject" and displays "Step 2: Select Subject". It features a white dropdown menu labeled "Subject Dropdown" with a downward arrow. Below the dropdown are two buttons: a red "Back" button and a green "Select Subject" button. Both screens have a blue semi-transparent overlay and a 3D architectural background.

Figure 15 - Step 02 / 03 – Select Course and Subject UI

The next step is to define the course and subject that it relates to, this can be done by selecting the relevant information in the drop-down options.

## Step 04 – Provide URL to Video

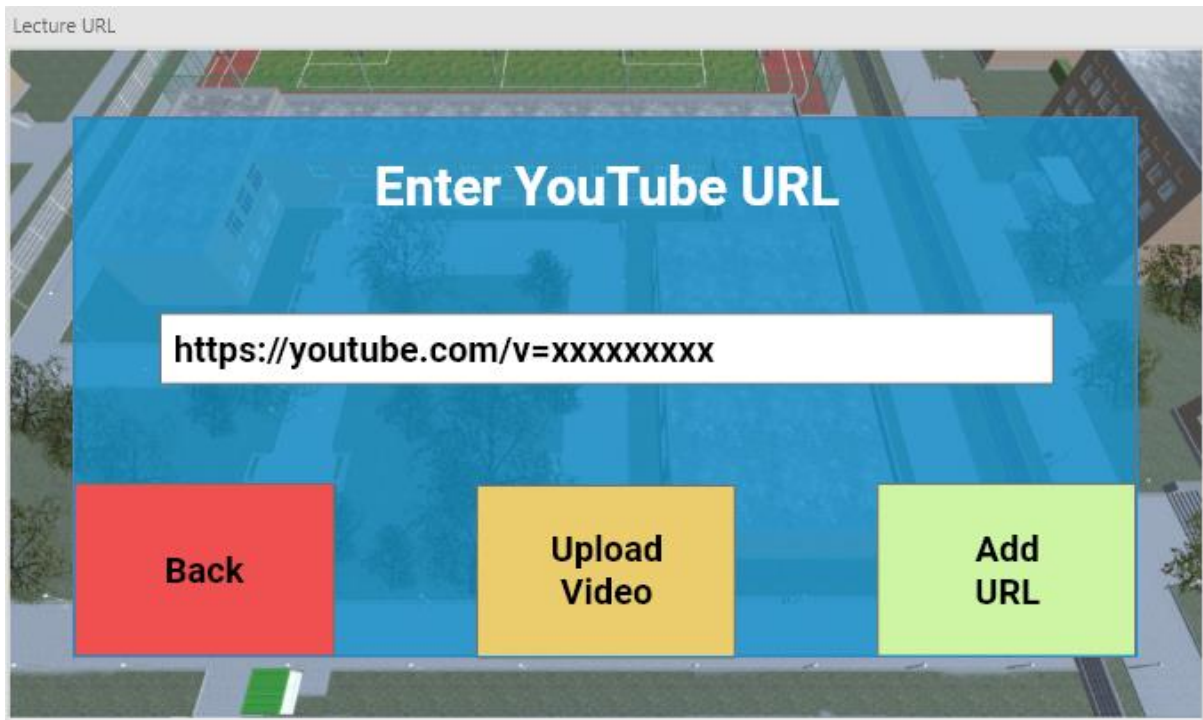


Figure 16 - Step 04 – Provide URL to Video UI

A video needs to be uploaded to web hosting, in order to provide a URL to the video location. Currently YouTube URLs will not work with Unity's video player.

## Step 05 / 06 – Set a Break Point and Define a Question

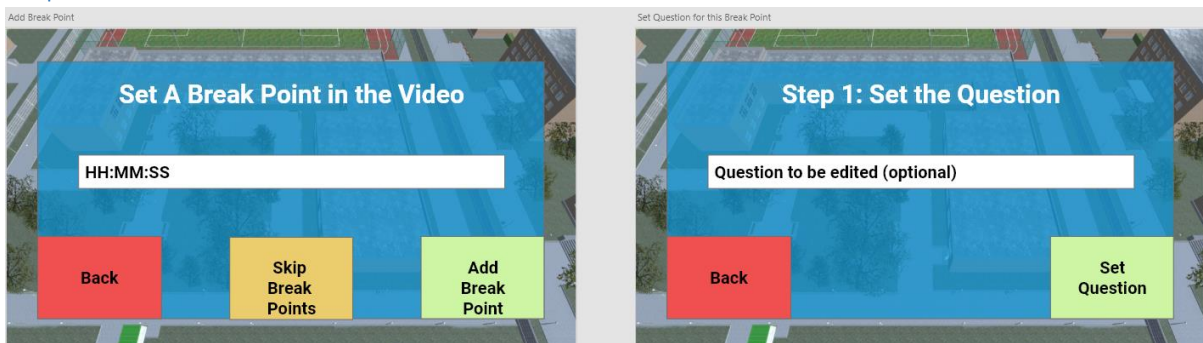


Figure 17 - Step 05 / 06 – Set a Break Point and Define a Question UI

Setting break points throughout a video lecture, breaks up the watching of a video and provides a more involved viewing experience. This is optional and can be skipped, however, step 05 requires a specified time within the video to break. At this given time a question can then be defined, similar to the quiz management process, although only one question can be presented per break point.

## Step 07 / 08 – Add correct and Incorrect Answers

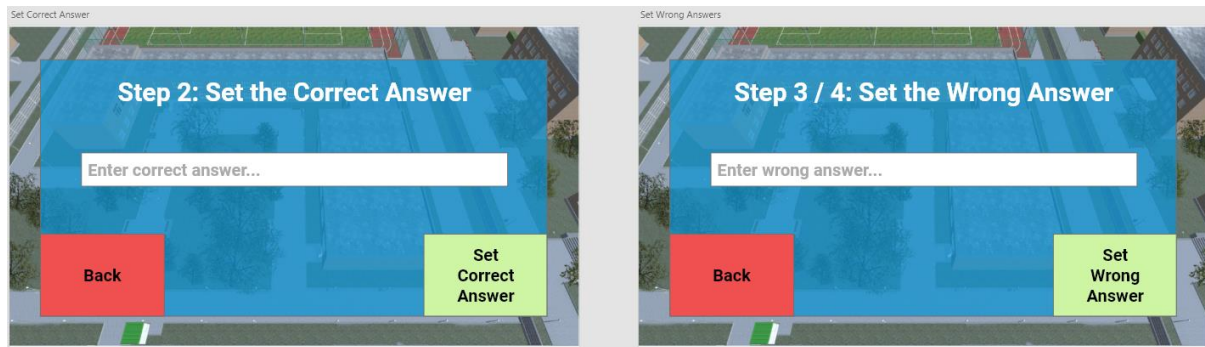


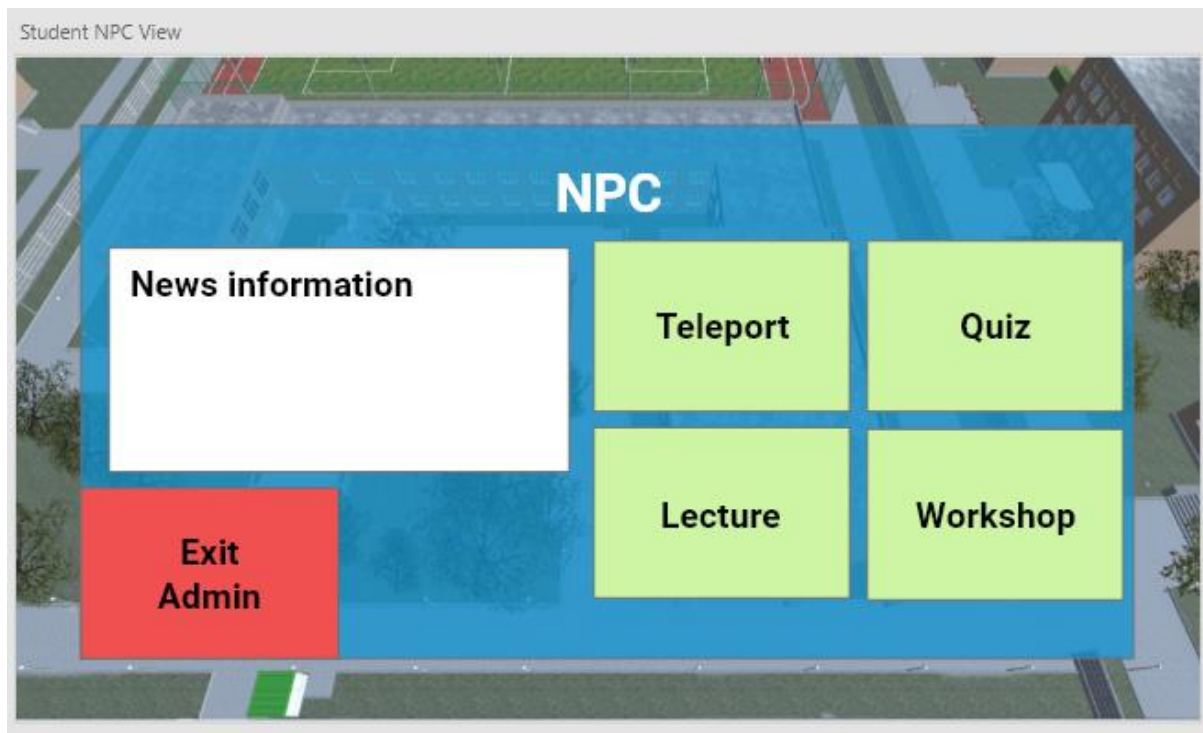
Figure 18 - Step 05 / 06 – Add correct and Incorrect Answers UI

Initially these steps ask for the Admin to input the correct answer first, followed by three wrong answers. As with the questions, these answers can be amended later if required. The process then loops back to Step 05.



### Student UI

The student user interface is only available to users with Student Account status and can be reached by visiting any of the Non-Playing Characters (NPC) within the game world.

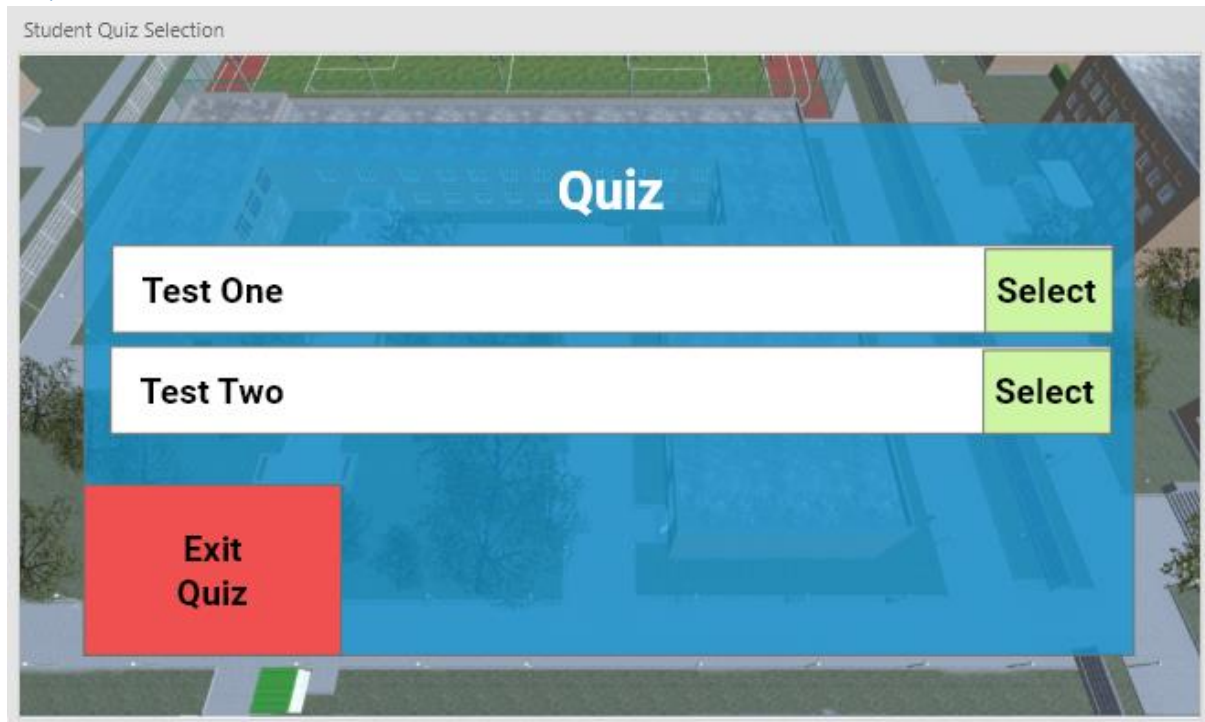


*Figure 19 - Student View UI*

The student is presented with a unique UI view, offering News information, teleportation to various areas within the game world, quizzes, lectures and workshops.

*Participate in a Quiz*

For a student to participate in a quiz, the user needs to select the Quiz option from the main student view user interface.

*Step 01 – Quiz Selection*

*Figure 20 - Step 01 – Quiz Selection UI*

The quizzes available for selection will be course based, for example a student studying Art will not see a quiz for students studying Mathematics. The student can select a quiz by clicking the Select button next to the desired quiz name.

## Step 02 – Participate in Quiz



Figure 21 - Step 02 – Participate in Quiz UI

Once a quiz has been selected, the main quiz UI will begin. This UI contains a countdown timer at the top in minutes and seconds. A large portion of the UI is dedicated to the question, ensuring the question is legible, with four answer buttons containing the multiple-choice answers for the given question.

Selecting an answer will provide immediate feedback to the student, of either correct or wrong.

## Step 03 – Results for the Quiz

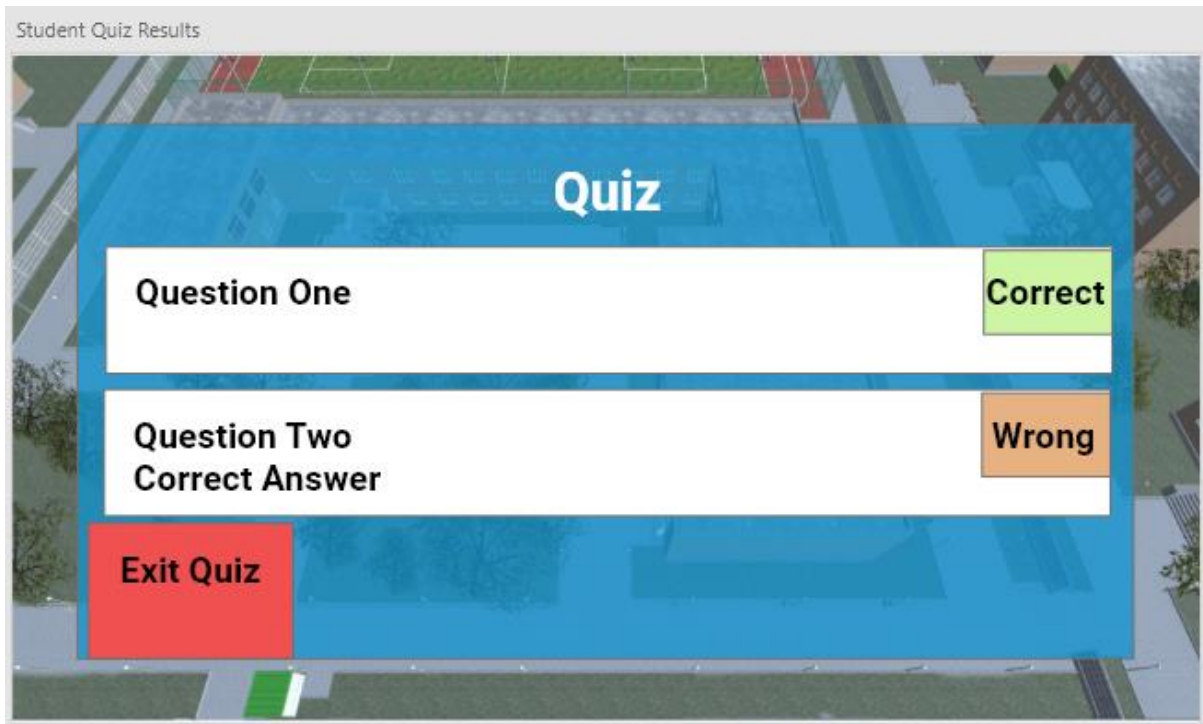


Figure 22 - Step 03 – Results for the Quiz UI

Once all questions have been answered, the student will be taken to the Results UI. The details of all questions asked and if answered correctly or not. If the question was answered incorrectly, the correct answer will be displayed.



*Participate in a Lecture*

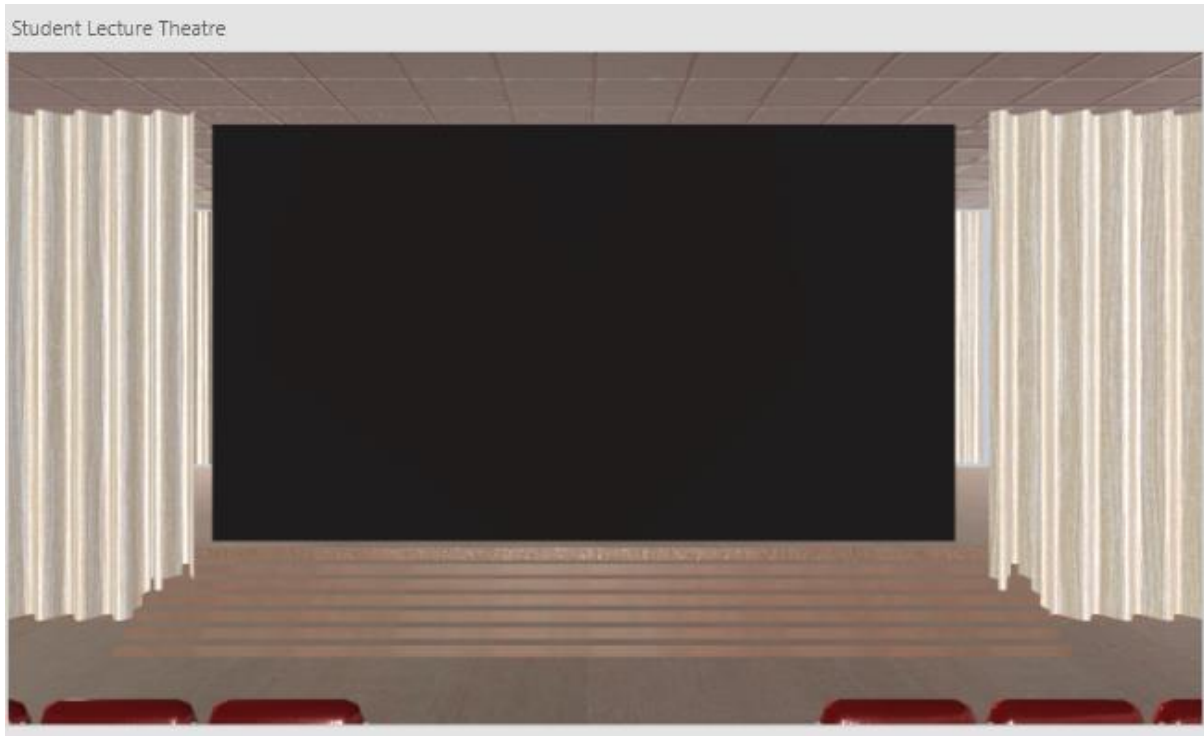
For a student to watch a lecture, the user needs to select the Lecture option from the main student view UI.

*Step 01 – Lecture Selection*

*Figure 23 - Step 01 – Lecture Selection UI*

Only course appropriate lectures will appear. The student can select a lecture, by clicking the Select button next to the desired lecture name.

## Step 02 – Watch the Lecture



*Figure 24 - Step 02 – Lecture Theater Screenshot*

Once a lecture has been selected, the student will be teleported to the lecture theatre, where the lecture will begin.

## Step 03 – Break Point Question

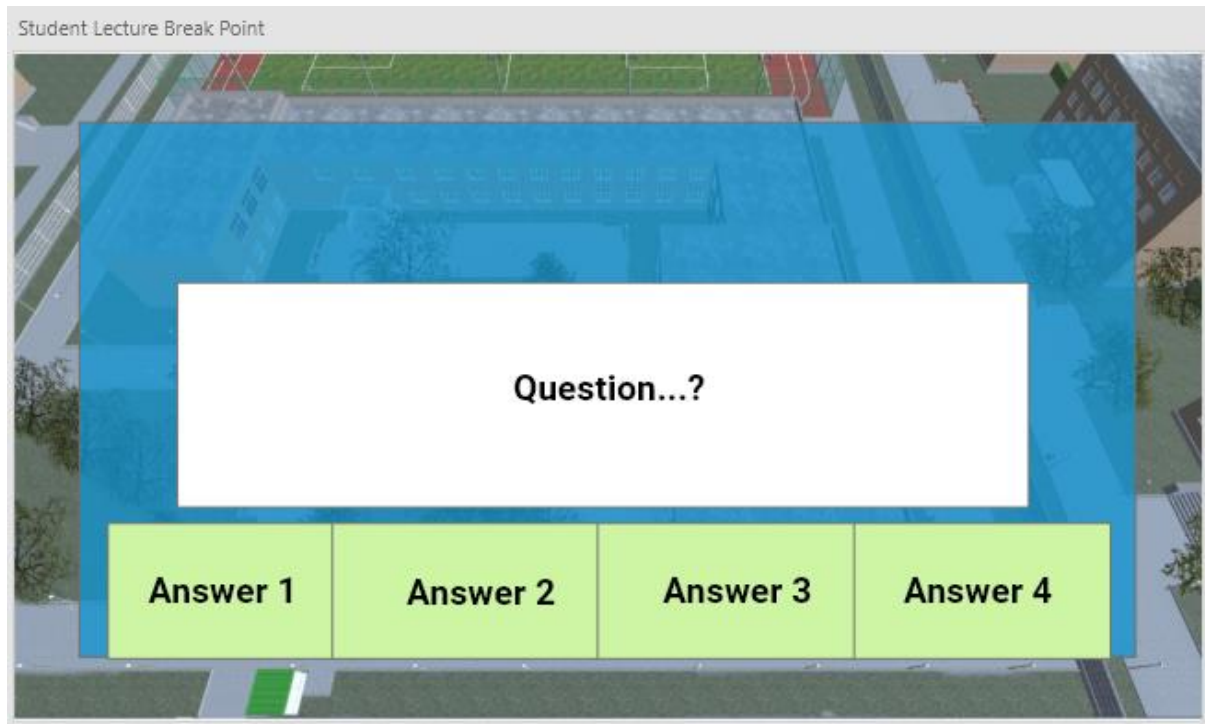


Figure 25 - Step 03 – Break Point Question UI

At previously defined points, the lecture will pause, and a question will appear. These questions have no time limit. The lecture will only resume once the question has been answered.

## Step 04 – Lecture Results



Figure 26 - Step 04 – Lecture Results UI

Once the lecture has concluded, the student will be taken to the Results UI. This will detail all questions asked and summarise if answered correctly or not. If the question was answered incorrectly, the correct answer will be displayed.

## Technical Documentation

The following diagrams are working documents, which are amended throughout the lifecycle of this project.

## Entity Relationship Diagram

The entity relationship diagram (ERD), defines the working structure of the custom attributes of the project database and its relationships.

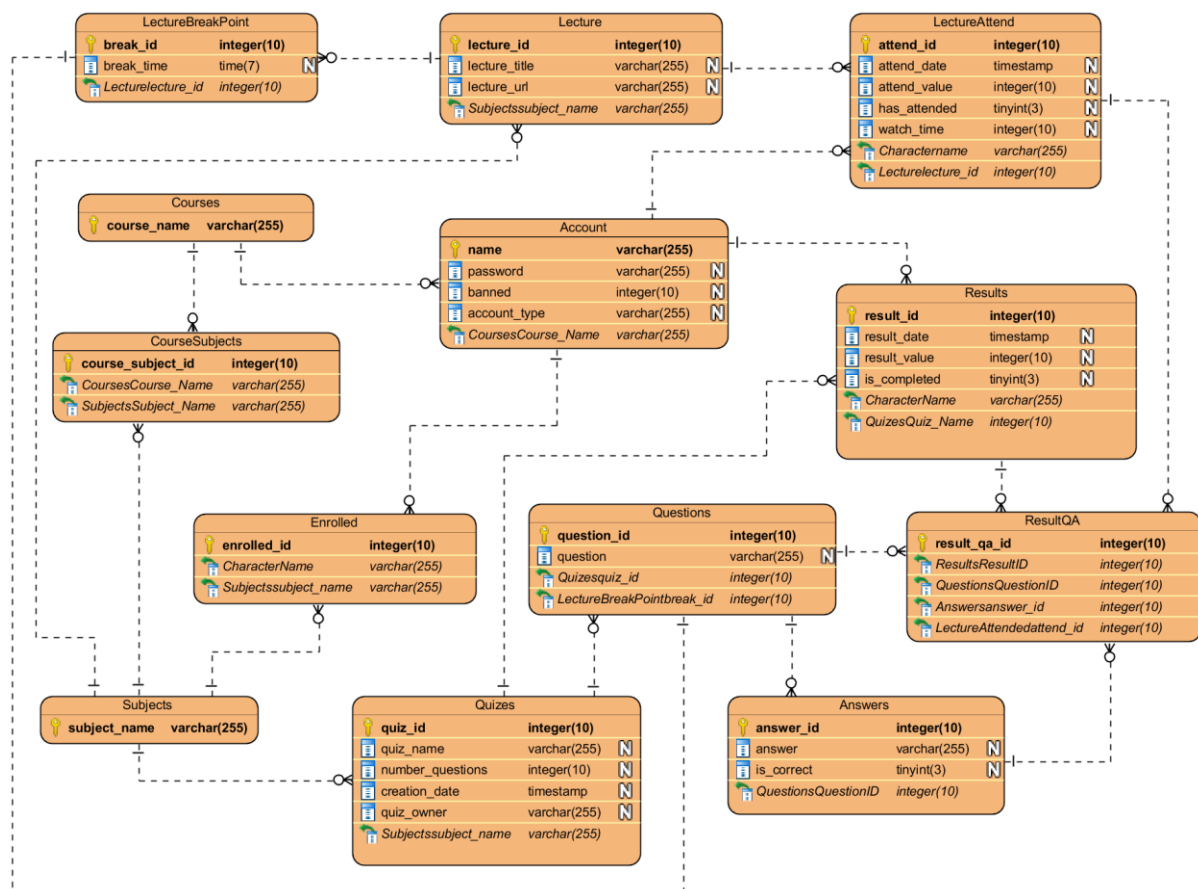


Figure 27 - Entity Relationship Diagram



## Class Diagram

The following class diagram defines the structure of the classes, attributes and operation.

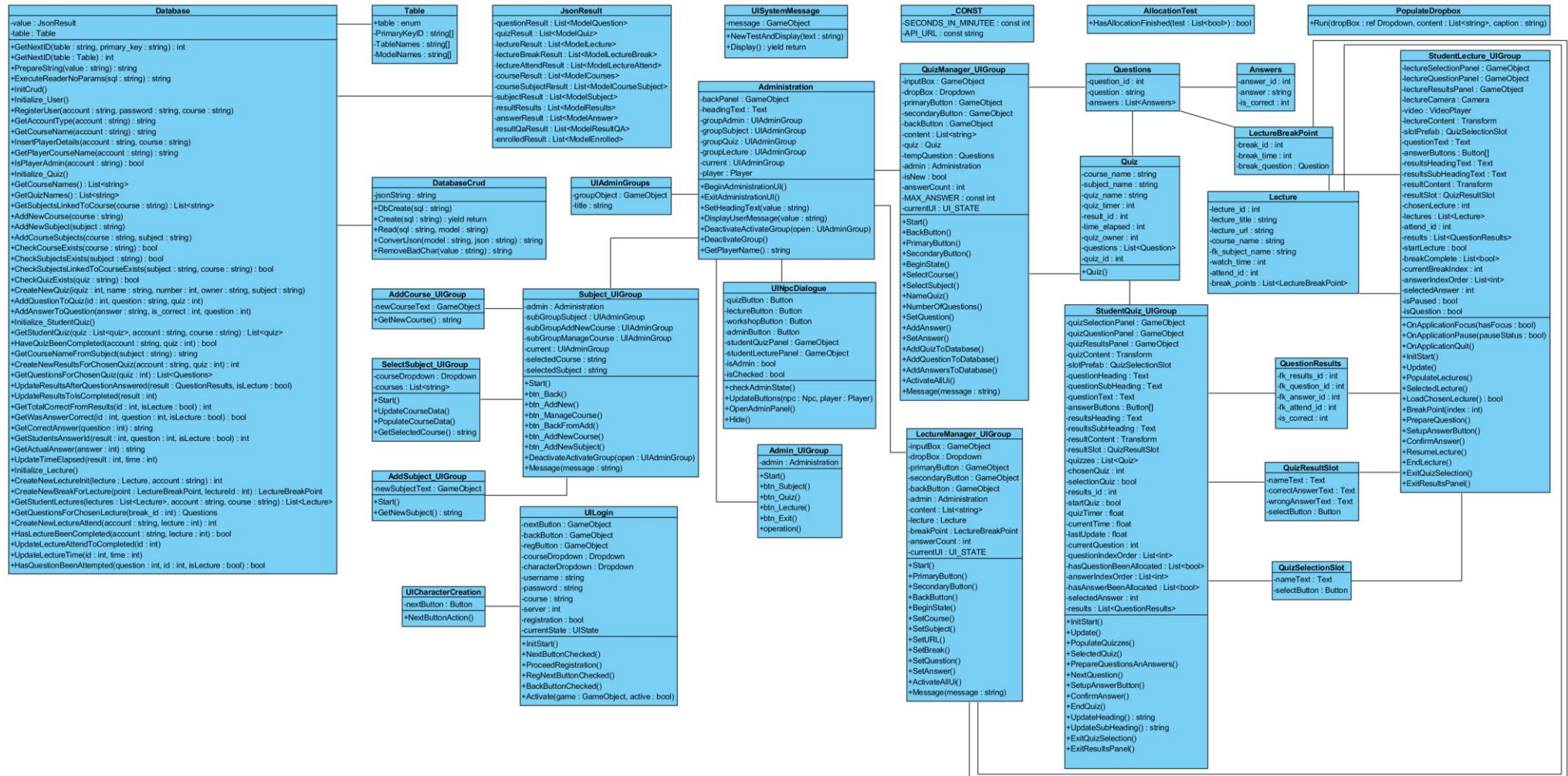
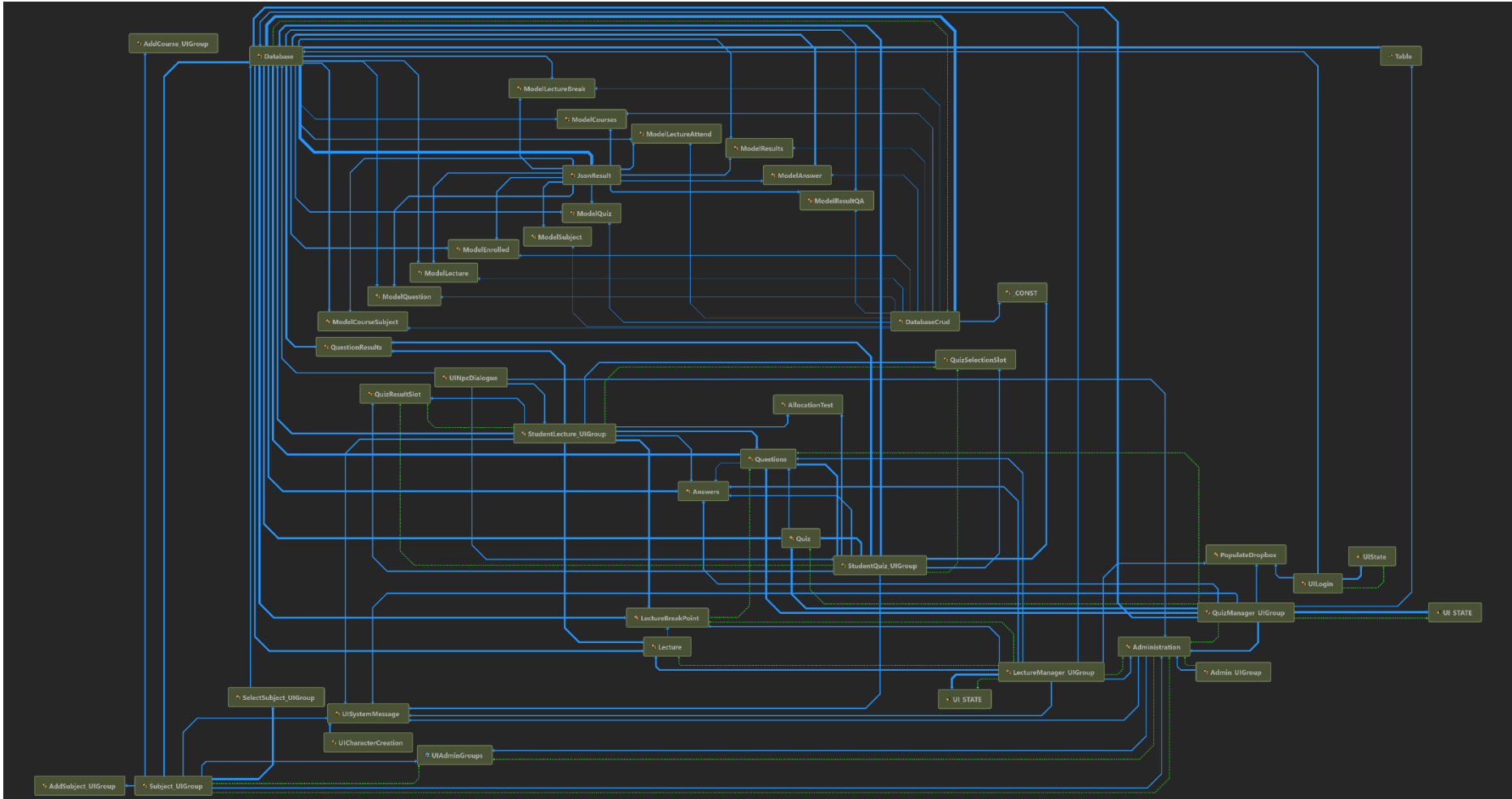


Figure 28 - Class Diagram

## Dependency Graph

The following Visual Studio diagram provides a visual representation of the classes and their dependences.



*Figure 29 - Dependency Graph*

## Implementation

### Third Party Assets / Tools

Various third-party tools were used in the implementation of this project, the project supervisor provided the main two. These provided a starting base for implementation to be built around. These were initially imported into a playground project for testing and manipulation.

#### uMMORPG

The uMMORPG tool (uMMORPG, 2018), provides many features of a typical Role-Playing style game, as well as a demo scene to help understand how the tool operates. A key feature is its server / client infrastructure, enabling a developer to rapidly get their game online.



Figure 30 - uMMORPG Scene

Many of the features were not required in this project. Once installed an abridged version was exported into a new package, with the relevant Scripts, UI, Player and Network Management.



### School Scene

The school scene (Tirgames Assets, 2018), provides a selection of school environment models. This asset provided the world in which the game would be played.



*Figure 31 - Day School Scene*

This asset came with components which would not be used in this project, so an abridged export was made excluding the night, abandoned and standard day implementations.

### Integration of assets

Once both assets had been exported into their minified versions, they were implemented into the project. Various testing took place to ensure they both worked as before, this included login features, database integration and standard UI. Most of the UI had lost its prefab connection within the Unity Inspector despite these prefabs being included in the export, these were manually added back into their required locations.

Due to the lack avatars, which were not taken over from the original files, actual gameplay could not be tested at this point.

### Custom Avatars

To play the game and test movement, an avatar needed to be implemented. Using the playground project and the original avatars, the first playable avatar was created called Boy (Game Asset Studio, 2018).

This proved a challenging task and was not well documented by uMMORPG. However, a video explaining the process (Gladius Studios, 2016) provided a good insight.

This involved reconfiguring the colliders, animations and body components. After completion, testing found an issue where the character would start walking backwards and would begin to fly higher and higher with every move command.

Through trial and error, it was discovered that the armature component, that initially appeared redundant, needed to remain in the prefab. This fixed the flying issue.

Using this as a template, a Girl (Asobiya, 2018) avatar was created. The implementation of this avatar was successful first time. Finally, an avatar was required for the Non-Player Characters, for which a model of a suited man (Studio New Punch, 2016) was utilised.



Figure 32 - Boy Avatar (Game Asset Studio, 2018)



Figure 34 - Girl Avatar (Asobiya, 2018)



Figure 33 - NPC Avatar (Studio New Punch, 2016)

### Camera Position

Following the design wireframe model, and after much trial and error, the camera settings were set at the following positions.

Setting Name	Setting Value
<b>Initial X Angle</b>	25
<b>Initial Distance</b>	4
<b>Min Distance</b>	4
<b>Max Distance</b>	5
<b>X Min Angle</b>	15
<b>X Max Angle</b>	50
<b>Camera Near Clipping Planes</b>	2

### Mobile UI

Initial testing on an android mobile phone highlighted some problems, including the initial UI being insufficient to allow successful selection and navigation. This required UI's to be implemented at a much larger size, ensuring input boxes and buttons were easily selectable on a small mobile screen.

To aid the new implementation, Simple UI (Unruly Games, 2017) was used for buttons, panels and icons. This provided a cosmetic element that made the UI more visually appealing, as seen in figure 35.

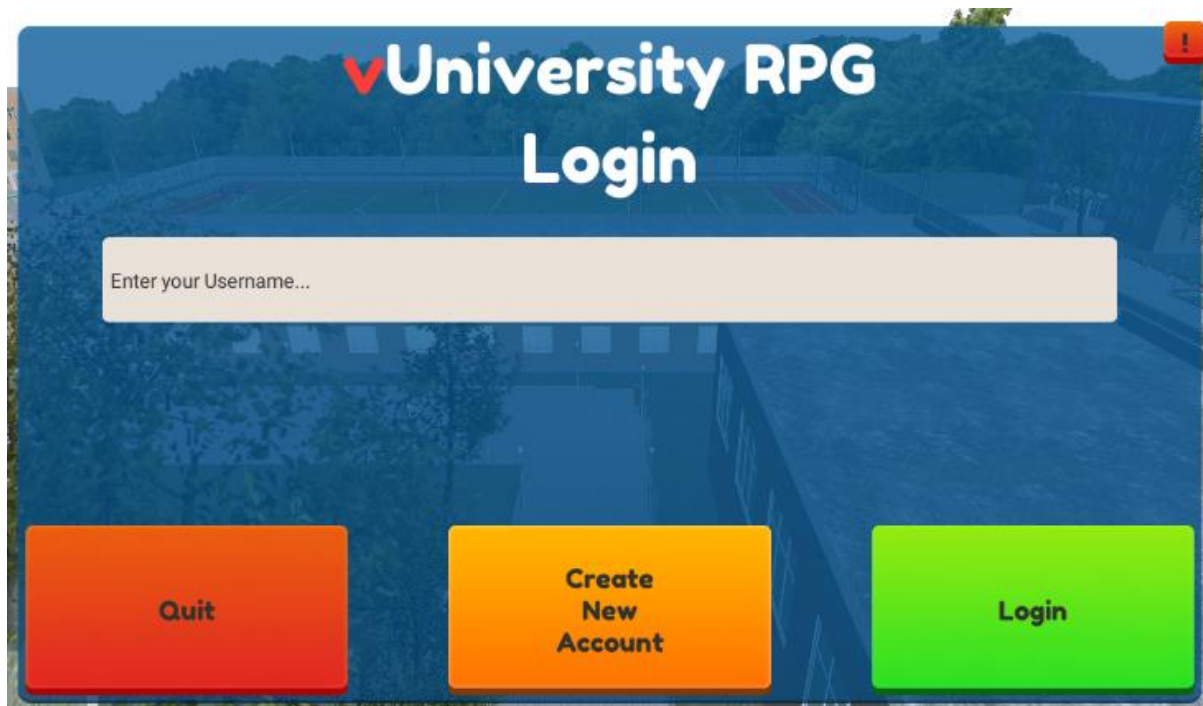


Figure 35 - Login UI



### Administrator Implementation

While implementing the administration section, two separate methods were used. Initially the process of navigating from screen to screen involved separate panels that would activate and deactivate, depending on what was required to be displayed.

To manage UI Groups from the Unity Inspector, the following structure was created so each group could have its own unique panel to load with the title defined.

```
[System.Serializable]
public struct UIAdminGroups {
    public GameObject groupObject;
    public string title;
}
```

When the administration access is selected, the following provides the UI to a starting position.

```
public void BeginAdministrationUI() {
    backPanel.SetActive(true);
    current = groupAdmin;
    current.groupObject.SetActive(true);
    SetHeadingText(current.title);
}
```

The following method provides a helper to close the previous UI and open the next, by passing in the desired group.

```
private void DeactivateActivateGroup(UIAdminGroups open) {
    current.groupObject.SetActive(false);
    current = open;
    current.groupObject.SetActive(true);
    SetHeadingText(current.title);
}
```

However, when moving onto the quiz and lecture administration, there were simply too many UI elements to manage, so a more stream lined approach was taken. This second method involved reusing elements within the UI and changing their values and requirements within the code.

To be able to do this successfully, an Enum was created to define the different states that the UI could be in.

```
public enum UI_STATE {
    Begin,
    QuestionManager,
    SelectCourse,
    SelectSubject,
    NameQuiz,
    NumberOfQuestions,
    AddQuestion,
    AddAnswer,
    COUNT
};
private UI_STATE currentUI;
```

For example, the begin state would ensure that the UI was set for this purpose.

```
private async void BeginState() {
    ActivateAllUi();
    inputBox.GetComponent<InputField>().text = "";
    inputBox.SetActive(false);
    admin.SetHeadingText("Add / Edit Quizzes");
    content = new List<string>();
    content = await Database.GetQuizNames();
    PopulateDropbox.Run(ref dropBox, content, "Select Quiz");
    primaryButton.GetComponentInChildren<Text>().text = "Manage\nSelected\nQuiz";
    secondaryButton.GetComponentInChildren<Text>().text = "Create\nNew\nQuiz";
}
```

The Activate All UI method provides a default state for all the elements and reverses any special requirements from a previous state, such as character limits, input type, etc.

```
private void ActivateAllUi() {
    inputBox.SetActive(true);
    dropBox.gameObject.SetActive(true);
    primaryButton.SetActive(true);
    secondaryButton.SetActive(true);
    backButton.SetActive(true);
    backButton.GetComponentInChildren<Text>().text = "Back";
    inputBox.GetComponent<InputField>().contentType =
        InputField.ContentType.Standard;
    inputBox.GetComponent<InputField>().characterLimit = 255;
    inputBox.GetComponentInChildren<Text>().text = "";
    inputBox.GetComponent<InputField>().text = "";
}
```

When a button is pressed, a switch case checks the current state and provides the required response, typically changing the current state to the next and executing a method to set the new UI.

```
public async void PrimaryButton() {
    switch (currentUI) {
        case UI_STATE.Begin:
            currentUI = UI_STATE.QuestionManager;
            isNew = false;
            break;
        case UI_STATE.SelectCourse:
            currentUI = UI_STATE.SelectSubject;
            SelectSubject();
            break;
    }
}
```

For example, if the state was set to Select Course the state would update to Select Subject, then initiate the Select Subject method to update the UI.

```
private async void SelectSubject() {
    quiz.CourseName = dropBox.GetComponent<Dropdown>()
        .options[dropBox.GetComponent<Dropdown>().value].text;
    Message("Course Added");
    ActivateAllUi();
    inputBox.SetActive(false);
    secondaryButton.SetActive(false);
    admin.SetHeadingText("Select a Subject");
    content = new List<string>();
    content = await Database.GetSubjectsLinkedToCourse(quiz.CourseName);
    PopulateDropbox.Run(ref dropBox, content, "Select Subject");
    primaryButton.GetComponentInChildren<Text>().text = "Select\nSubject";
    backButton.GetComponentInChildren<Text>().text = "Exit Quiz\nManagement";
}
```

Once the workflow as described in the design, is completed, a call to the database is made to insert the data.

```
private async Task AddQuizToDatabase() {
    quiz.QuizTimer = Int32.Parse(inputBox.GetComponent<InputField>().text);
    quiz.QuizId = await Database.GetNextID_Crud(Database.Table.Quizzes);
    Database.CreateNewQuiz(quiz.QuizId, quiz.QuizName, quiz.QuizTimer,
        FindObjectOfType<Player>().account, quiz.SubjectName);
}
```

## Quiz Implementation

The Quiz section begins when a student navigates there via the NPC panel, from which the method initialises all the starting UI.

```
public async void InitStart() {
    player = FindObjectOfType<NetworkManagerMMO>().loginAccount;
    chosenQuiz = -1;
    quizSelectionPanel.SetActive(true);
    quizzes = new List<Quiz>();
    hasQuestionBeenAllocated = new List<bool>();
    questionIndexOrder = new List<int>();
    quizzes = await Database.GetStudentQuizzes(quizzes, player,
        await Database.GetPlayerCourseName(player));
    PopulateQuizzes();
    selectionQuiz = true;
    startQuiz = false;
    currentTime = 0.0f;
}
```

This initialises the selection panel, setting initial variables and then populating the quiz data from the database.

```
private void PopulateQuizzes() {
    UIUtils.BalancePrefabs(slotPrefab.gameObject, quizzes.Count,
        quizContent);
    for (int i = 0; i < quizzes.Count; i++) {
        QuizSelectionSlot slot = quizContent.GetChild(i).
            GetComponent<QuizSelectionSlot>();
        slot.nameText.text = quizzes[i].QuizName;
        int count = i;
        slot.selectButton.onClick.AddListener(async () => {
            chosenQuiz = count;
            results = new List<QuestionResults>();
            if (quizzes[chosenQuiz].result_id >= 0) {
                results_id = quizzes[chosenQuiz].result_id;
            } else {
                results_id = await Database.CreateNewResultsForChosenQuiz
                    (player, quizzes[chosenQuiz].QuizId);
            }
            SelectedQuiz();
        });
    }
}
```

The populate quizzes method did not initially work, due to the count variable being initialised and set inside of the lambda function. Once this was declared outside, the method worked.

Once a quiz has been selected, the questions are prepared and randomised.

```
private async void PrepareQuestionsAndAnswers() {
    quizzes[chosenQuiz].Questions =
        await Database.GetQuestionsForChosenQuiz(quizzes[chosenQuiz].QuizId);
    hasQuestionBeenAllocated = new List<bool>();
    questionIndexOrder = new List<int>();
    for (int i = 0; i < quizzes[chosenQuiz].Questions.Count; i++) {
        hasQuestionBeenAllocated.Add(false);
    }
    while (!AllocationTest.HasAllocationFinished(hasQuestionBeenAllocated)) {
        int index = Random.Range(0, quizzes[chosenQuiz].Questions.Count);
        if (!hasQuestionBeenAllocated[index]) {
            if (!await Database.HasQuestionBeenAttempted(quizzes[chosenQuiz].
                Questions[index].question_id, results_id, false)) {
                questionIndexOrder.Add(index);
            }
            hasQuestionBeenAllocated[index] = true;
        }
    }
    if (quizzes[chosenQuiz].time_elapsed > 0) {
        quizTimer = (quizzes[chosenQuiz].QuizTimer * _CONST.SECONDS_IN_MINUTE)
            - quizzes[chosenQuiz].time_elapsed;
    } else {
        quizTimer = quizzes[chosenQuiz].QuizTimer * _CONST.SECONDS_IN_MINUTE;
    }
    lastUpdate = quizzes[chosenQuiz].QuizTimer * _CONST.SECONDS_IN_MINUTE;
    startQuiz = true;
    NextQuestion();
}
```

The above method only gets called at the start of a quiz selection, after which each question presented calls the Next Question method. This sets the question into the UI and then randomises the potential answers.

```
private void NextQuestion() {
    questionText.text = quizzes[chosenQuiz].
        Questions[questionIndexOrder[currentQuestion]].question;
    hasAnswerBeenAllocated = new List<bool>();
    answerIndexOrder = new List<int>();
    for (int i = 0; i < quizzes[chosenQuiz].
        Questions[questionIndexOrder[currentQuestion]].
            answers.Count; i++) {
        hasAnswerBeenAllocated.Add(false);
    }
    while (!AllocationTest.HasAllocationFinished(hasAnswerBeenAllocated)) {
        int index = Random.Range(0, quizzes[chosenQuiz].
            Questions[questionIndexOrder[currentQuestion]].answers.Count);
        if (!hasAnswerBeenAllocated[index]) {
            answerIndexOrder.Add(index);
            hasAnswerBeenAllocated[index] = true;
        }
    }
}
```



Once the answers are randomised, the answer can be setup, concluding the question setup and display.

```
private void SetupAnswerButton() {
    for (int i = 0; i < quizzes[chosenQuiz].
        Questions[questionIndexOrder[currentQuestion]].
            answers.Count; i++) {
        answerButtons[i].GetComponentInChildren<Text>().text
            = quizzes[chosenQuiz]
                .Questions[questionIndexOrder[currentQuestion]]
                    .answers[answerIndexOrder[i]].answer;
        int count = i;
        answerButtons[i].onClick.AddListener(() => {
            selectedAnswer = answerIndexOrder[count];
            ConfirmAnswer();
        });
    }
}
```

When a question is answered by the user, the following method records the data and updates the relevant database tables. Feedback is passed to the system message, to display to the user.

```
private void ConfirmAnswer() {
    QuestionResults result = new QuestionResults();
    result.fk_results_id = results_id;
    result.fk_answer_id = quizzes[chosenQuiz]
        .Questions[questionIndexOrder[currentQuestion]]
            .answers[selectedAnswer].answer_id;
    result.fk_question_id = quizzes[chosenQuiz]
        .Questions[questionIndexOrder[currentQuestion]]
            .question_id;
    result.isCorrect = quizzes[chosenQuiz]
        .Questions[questionIndexOrder[currentQuestion]]
            .answers[selectedAnswer].isCorrect;
    results.Add(result);
    Database.UpdateResultsAfterQuestionAnswered(result, false);
    Database.UpdateTimeElapsed(results_id,
        (quizzes[chosenQuiz].QuizTimer * _CONST.SECONDS_IN_MINUTE)
        - (int)quizTimer);
    if (currentQuestion < questionIndexOrder.Count - 1) {
        currentQuestion++;
        if (result.isCorrect == 1) {
            FindObjectOfType<UISystemMessage>().NewTextAndDisplay("CORRECT");
        } else {
            FindObjectOfType<UISystemMessage>().NewTextAndDisplay("Wrong");
        }
        NextQuestion();
    } else {
        EndQuiz();
    }
}
```

Once all questions have been answered, the End Quiz method, calculates the results and displays the results panel with feedback, as described in the design phase.

```
private async void EndQuiz() {
    // Update Database with result is_completed
    Database.UpdateResultsToIsCompleted(results_id);
    // Display Result Panel
    startQuiz = false;
    quizQuestionPanel.gameObject.SetActive(false);
    quizResultsPanel.gameObject.SetActive(true);
    resultsHeadingText.text = "Results for " + quizzes[chosenQuiz].QuizName;
    // Calculate result as percentage
    int totalQuestions = quizzes[chosenQuiz].Questions.Count;
    int totalCorrect = await Database.GetTotalCorrectFromResults(results_id, false);
    float percentage = 0;
    if (totalCorrect != 0 || totalQuestions != 0) {
        percentage = (float) (totalCorrect * 100) / totalQuestions;
    }
    resultsSubHeadingText.text = "Your Result is " + Math.Ceiling(percentage) + "%";
    // Show each question and define correct and wrong answers.
    UIUtils.BalancePrefabs(resultSlot.gameObject, totalQuestions, resultContent);
    for (int i = 0; i < totalQuestions; i++) {
        QuizResultSlot slot = resultContent.GetChild(i).
            GetComponent<QuizResultSlot>();
        slot.nameText.text = "Q" + (i + 1) + ". " + quizzes[chosenQuiz]
            .Questions[i].question;
        slot.correctAnswerText.text =
            "Correct Answer: " + await Database.GetCorrectAnswer(quizzes[chosenQuiz].
                Questions[i].question_id);
        if (await Database.GetWasAnswerCorrect(results_id,
            quizzes[chosenQuiz].Questions[i].question_id, false)) {
            slot.selectButton.GetComponentInChildren<Text>().text = "CORRECT";
            slot.selectButton.GetComponent<Image>().color = Color.green;
        } else {
            slot.selectButton.GetComponentInChildren<Text>().text = "Incorrect";
            slot.selectButton.GetComponent<Image>().color = Color.yellow;
            slot.wrongAnswerText.text =
                "You Answered: " + await Database.GetActualAnswer(
                    await Database.GetStudentsAnswerId(results_id,
                        quizzes[chosenQuiz].Questions[i].question_id, false));
        }
    }
}
```

Throughout the quiz, the following Update method has been executing to ensure the timers are maintained and button are being listened for.

```
void Update() {
    if (selectionQuiz) {
        PopulateQuizzes();
        if (chosenQuiz != -1) {
            selectionQuiz = false;
        }
    } else if (startQuiz) {
        if (quizTimer < 0) {
            FindObjectOfType<UISystemMessage>().
                NewTextAndDisplay("Time has run out");
            EndQuiz();
        }
        if (lastUpdate - quizTimer >= 1.0f) {
            questionSubHeading.text = UpdateSubHeading();
            lastUpdate = quizTimer;
            questionHeading.text = UpdateHeading();
        }
        quizTimer -= Time.timeSinceLevelLoad - currentTime;
        currentTime = Time.timeSinceLevelLoad;
        SetupAnswerButton();
    }
}
```

A couple of helper methods. assist in presenting the heading and timer displays.

```
private string UpdateHeading() {
    return quizzes[chosenQuiz].QuizName + " - " +
        (currentQuestion + 1) + "/" + questionIndexOrder.Count;
}

private string UpdateSubHeading() {
    float minutes = Mathf.Floor(quizTimer / _CONST.SECONDS_IN_MINUTE);
    float seconds = quizTimer % _CONST.SECONDS_IN_MINUTE;
    string zero = (seconds < 10) ? "0" : "";
    return "Timer: " + minutes + ":" + zero + (int)seconds;
}
```

## Lecture Implementation

The Lecture implementation has similarities to the Quiz Implementation.

Once a lecture has been selected, the following method initiates the lecture and determines if the break points have already been completed or not.

```
private async void SelectedLecture() {
    if (await LoadChosenLecture()) {
        lectureCamera.gameObject.SetActive(true);
        breakComplete = new List<bool>();
        for (int i = 0; i < lectures[chosenLecture].
            break_points.Count; i++) {
            if (await Database.HasQuestionBeenAttempted(
                lectures[chosenLecture].break_points[i].
                break_question.question_id, attend_id, true)) {
                breakComplete.Add(true);
            } else {
                breakComplete.Add(false);
            }
        }
        startLecture = true;
    }
}
```

If the video successfully loads, the Load Chosen Lecture method is wrapped in a try catch statement. This prevents the lecture camera being set to active with no video playing, which could present problems for the user.

```
private async Task<bool> LoadChosenLecture() {
    try {
        video.Stop();
        video.url = lectures[chosenLecture].lecture_url;
        if (lectures[chosenLecture].watch_time > 0) {
            video.time = (double)lectures[chosenLecture].watch_time;
        }
        lectureSelectionPanel.SetActive(false);
        isQuestion = false;
        video.Play();
        video.isLooping = false;
        results = new List<QuestionResults>();
        if (lectures[chosenLecture].attend_id >= 0) {
            attend_id = lectures[chosenLecture].attend_id;
        } else {
            attend_id = await Database.CreateNewLectureAttend(player,
                lectures[chosenLecture].lecture_id);
        }
        return true;
    } catch (Exception e) {
        Debug.LogError("Lecture failed to load: " + e);
        return false;
    }
}
```

If a break points time frame is detected, then the video is paused, and the question, answers and UI are prepared to be displayed.

```
private void BreakPoint(int index) {
    currentBreakIndex = index;
    video.Pause();
    lectureCamera.gameObject.SetActive(false);
    lectureQuestionPanel.gameObject.SetActive(true);
    PrepareQuestion();
    SetupAnswerButton();
    Database.UpdateLectureTime(attend_id,
        Mathf.FloorToInt((float)video.time));
}
```

Once the question has been answered, the lecture will resume.

```
private void ResumeLecture() {
    lectureCamera.gameObject.SetActive(true);
    lectureQuestionPanel.gameObject.SetActive(false);
    isQuestion = false;
    video.Play();
}
```

Once all the lecture video comes to an end, the results are recorded and displayed for the user.

```
private async void EndLecture() {
    lectureCamera.gameObject.SetActive(false);
    lectureResultsPanel.gameObject.SetActive(true);
    video.Stop();
    isQuestion = true;
    startLecture = false;
    resultsHeadingText.text =
        "Results for " + lectures[chosenLecture].lecture_title;
    // Calculate result as percentage
    int totalQuestions = lectures[chosenLecture].break_points.Count;
    int totalCorrect =
        await Database.GetTotalCorrectFromResults(attend_id, true);
    float percentage = 0;
    if (totalCorrect != 0 || totalQuestions != 0) {
        percentage = (float)(totalCorrect * 100) / totalQuestions;
    }
    resultsSubHeadingText.text =
        "Your Result is " + Math.Ceiling(percentage) + "%";
    // Show each question and define correct and wrong answers.
    UIUtils.BalancePrefabs(resultSlot.gameObject,
        totalQuestions, resultContent);
    for (int i = 0; i < totalQuestions; i++) {
        QuizResultSlot slot = resultContent.GetChild(i).
            GetComponent<QuizResultSlot>();
        slot.nameText.text =
            "Q" + (i + 1) + ". " + lectures[chosenLecture].
                break_points[i].break_question.question;
        slot.correctAnswerText.text =
            "Correct Answer: "
            + await Database.GetCorrectAnswer(lectures[chosenLecture].
                break_points[i].break_question.question_id);
        if (await Database.GetWasAnswerCorrect(attend_id, lectures[chosenLecture].
            break_points[i].break_question.question_id, true)) {
            slot.selectButton.GetComponentInChildren<Text>().text = "CORRECT";
            slot.selectButton.GetComponent<Image>().color = Color.green;
        } else {
            slot.selectButton.GetComponentInChildren<Text>().text = "Incorrect";
            slot.selectButton.GetComponent<Image>().color = Color.yellow;
            slot.wrongAnswerText.text =
                "You Answered: " + await Database.GetActualAnswer(
                    await Database.GetStudentsAnswerId(attend_id,
                        lectures[chosenLecture].break_points[i].
                            break_question.question_id, true));
        }
    }
    // update lecture attend table to be completed.
    Database.UpdateLectureTime(attend_id, Mathf.FloorToInt((float)video.time));
    Database.UpdateLectureAttendToComplete(attend_id);
}
```



Throughout the lecture, the following Update method has been executing whilst the video is playing, checking for any break points which have not been completed. This update also checks for when the video has finished in order to end the lecture.

```
void Update() {
    if (startLecture) {
        if (video.isPlaying) {
            for (int i = 0; i < lectures[chosenLecture].
                break_points.Count; i++) {
                // check for breakpoint questions
                if (video.time >= lectures[chosenLecture]
                    .break_points[i].
                    break_time && !breakComplete[i]) {
                    // pause video and display question
                    isQuestion = true;
                    BreakPoint(i);
                    breakComplete[i] = true;
                }
            }
            if (isPaused) {
                Database.UpdateLectureTime(attend_id,
                    Mathf.FloorToInt((float)video.time));
                video.Pause();
            }
        } else if (!isPaused && !isQuestion) {
            video.Play();
        }
        if (video.frame > 1 &&
            video.frame == (long)video.frameCount) {
            EndLecture();
        }
    }
}
```

## Database Implementation

As seen from the Technical Documentation class diagram, the Database class is the largest of all the classes implemented. This class is organised over many files, allowing the separation of concerns, this is done by declaring the class as a partial. These are broken down into the following categories, User, Quiz, Student Quiz, Lectures and a Variable Helper.

Each category is responsible for initialising a Create Table if one does not already exist. This ensures that when going to production, the structure of the database will self-create.

```
static void Initialize_User() {  
    ExecuteNoReturn(@"CREATE TABLE IF NOT EXISTS accounts (  
        name TEXT NOT NULL PRIMARY KEY,  
        password TEXT NOT NULL,  
        banned INTEGER NOT NULL DEFAULT 0,  
        account_type TEXT NOT NULL DEFAULT 'Student',  
        fk_course TEXT);");  
  
    crud.DbCreate(@"CREATE TABLE IF NOT EXISTS Enrolled (  
        enrolled_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,  
        fk_account VARCHAR(255) NOT NULL,  
        fk_course_name VARCHAR(255) NOT NULL);");  
}
```

As can be seen from the above code snippet, there are two different methods called. The first method, Execute No Return, is a method built into uMMORPG which is a flat SQLite Database and will execute from the Game Server (Host) only. The second method Crud DB Create, is a method that executes to a custom MySQL database hosted online, that can be executed by the Game Server or the Client.

Initially, all database functions were built to execute to the original uMMORPG database. However, when testing the network capabilities, the client machines were not able to access the SQLite database. This was a fundamental flaw in the project and hindered the ability to obtain one of its primary objectives. A second database was created to host and be available to both client and host.

The first step was to create a dirty PHP API for all SQL statements to be passed. This is not a recommended method and requires future improvements to secure this functionality but was necessary due to time constraints.

```
<?php
    ini_set('display_errors', 1);
    ini_set('display_startup_errors', 1);
    error_reporting(E_ALL);

    class DBConnection {
        private $db_host = "localhost";
        private $db_name = "mychaosc_uni_rpg";
        private $db_user = "mychaosc_unirpg";
        private $db_pass = "yvJB2z8IUZQD";
        protected $pdoConn;
        public function __construct() {
            try {
                $this->pdoConn = new PDO(
                    "mysql:host=$this->db_host;
                    dbname=$this->db_name;
                    charset=utf8mb4",
                    $this->db_user, $this->db_pass);
                $this->pdoConn->setAttribute(PDO::ATTR_ERRMODE,
                    PDO::ERRMODE_EXCEPTION);
            } catch (PDOException $ex) {
                echo 'CONNECTION ERROR: ' . $ex->getMessage();
            }
        }
        public function GetPdo() {
            return $this->pdoConn;
        }
    }
?>
```

The above class creates the connection to the database.

The following class, gets the instance of the database connection and prepares the SQL statement, encoding any return value to JSON format.

```
<?php
    require_once('db_conn.php');

    class DBExecute extends DBConnection {
        protected $pdo;

        public function __construct() {
            parent::__construct();
            $this->pdo = $this->GetPdo();
        }

        public function Query($sql) {
            $statement = $this->pdo->prepare($sql);
            $statement->execute();
            if (substr( $sql, 0, 6 ) === "SELECT") {
                $result = $statement->fetchAll(PDO::FETCH_ASSOC);
                return json_encode($result);
            }
            return 1;
        }
    }

    $DB = new DBExecute();
?>
```

The entry point for the API, accepts the SQL request and prints the results of the query.

```
<?php
    require_once('db/db_exe.php');

    if (isset($_GET['sql'])) {
        echo $DB->Query($_REQUEST['sql']);
    } else {
        echo "Welcome";
    }
?>
```

As all the functions requiring client access needed to be rebuilt, a variable helper was created to prevent any typos in the SQL command for commonly used values.

```
public enum Table {
    Quizzes, Questions, Lectures, LectureBreakPoints,
    LectureAttend, Courses, CourseSubjects, Subjects,
    Results, Answers, ResultQA, Enrolled,
    COUNT
};

protected static string[] PrimaryKeyID = {
    "quiz_id", "question_id", "lecture_id", "break_id",
    "attend_id", "course_name", "course_subject_id",
    "subject_name", "result_id", "answer_id",
    "result_qa_id", "enrolled_id",
};

protected static string[] TableNames = {
    "Quizzes", "Questions", "Lectures", "LectureBreakPoints",
    "LectureAttend", "Courses", "CourseSubjects", "Subjects",
    "Results", "Answers", "ResultQA", "Enrolled",
};

protected static string[] ModelNames = {
    "quizResult", "questionResult", "lectureResult",
    "lectureBreakResult", "lectureAttendResult",
    "courseResult", "courseSubjectResult",
    "subjectResult", "resultResult", "answerResult",
    "resultQaResult", "enrolledResult",
};
```

Each Enum integer value, corresponds to the relevant primary key, table name and model name.

To be able to pass SQL queries over to the API, an IEnumerator was created for the Insert, Update and Delete statements, typically these types of statements don't return any results.

```
public void DbCreate(string sql) {
    StartCoroutine(Create(sql));
}

private IEnumerator Create(string sql) {
    string uri = _CONST.API_URL + sql;
    UnityWebRequest www = UnityWebRequest.Get(uri);
    yield return www.SendWebRequest();
    if (www.isNetworkError || www.isHttpError) {
        Debug.LogError(www.error + "\n" + sql);
    } else {
        Debug.Log("Create Result: " + www.downloadHandler.text
            + " SQL: " + sql);
    }
}
```

Select statements return results if they exist, so a special Read IEnumerator was implemented.

```
public IEnumerator Read(string sql, string model) {
    string uri = _CONST.API_URL + sql;
    UnityWebRequest www = UnityWebRequest.Get(uri);
    yield return www.SendWebRequest();
    if (www.isNetworkError || www.isHttpError) {
        Debug.LogError(www.error);
    } else {
        jsonString = ConvertJson(model, www.downloadHandler.text);
        Debug.Log("JSON: " + jsonString + "\nSQL: " + sql);
        yield return jsonString;
    }
}
```

As this method of accessing the database online presents a delay in return, the project had to be upgraded to utilise .NET 4.0 features. This allowed the creation of async functions with await commands before a database call, meaning the operation could not continue until the results had been obtained. To help with this, the Async Await Support (Modest Tree Media, 2018) tool was obtained from the Unity asset store.



When the Json string is returned, it needs to be converted to include the relevant model assigned to it.

```
private string ConvertJson(string model, string json) {
    string result = "{\"" + model + "\": " + json.Trim() + "}";
    return RemoveBadChar(result);
}
```

Previously the JSON string, used the trim method to remove any zero width characters, but after upgrading to .NET 4.0 this no longer worked. To get around this, a method was created to identify which index contained the character and rebuilt the string excluding it from the result.

```
private string RemoveBadChar(string value) {
    char[] test = value.ToCharArray();
    int badCharIndex = -1;
    for (int i = 0; i < test.Length; i++) {
        if (test[i] == '\uFEFF') {
            badCharIndex = i;
        }
    }
    StringBuilder newValue = new StringBuilder();
    if (badCharIndex != -1) {
        for (int i = 0; i < test.Length; i++) {
            if (badCharIndex != i) {
                newValue.Append(test[i]);
            }
        }
    } else {
        newValue.Append(value);
    }
    return newValue.ToString();
}
```

An example of calling the database method for a Select statement.

```
public static async Task<List<string>> GetCourseNames() {
    int selection = (int) Table.Courses;
    string sql = "SELECT " + PrimaryKeyID[selection] + " FROM "
        + TableNames[selection] + " ORDER BY "
        + PrimaryKeyID[selection] + " ASC";
    string json = (string) await crud.Read(sql, ModelNames[selection]);
    DatabaseCrud.JsonResult value =
        JsonUtility.FromJson<DatabaseCrud.JsonResult>(json);
    List<string> result = new List<string>();
    for (int i = 0; i < value.courseResult.Count; i++) {
        result.Add(value.courseResult[i].course_name);
    }
    return result;
}
```

An example of calling the database for an Insert statement.

```
public static async Task<int> CreateNewResultsForChosenQuiz
    (string account, int quiz) {
    int selection = (int)Table.Results;
    int id = await GetNextID_Crud(Table.Results);
    crud.DbCreate("INSERT INTO " + TableNames[selection] + " ("
        + PrimaryKeyID[selection] + ", fk_account, fk_quiz_id" +
        ") VALUES (" + id + ", " + PrepareString(account)
        + ", " + quiz + ")");
    return id;
}
```

More elaborate functions can utilise different types of statements.

```
public static async void UpdateResults(QuestionResults result) {
    if (result.isCorrect == 1) {
        int selection = (int) Table.Results;
        string sql =
            "SELECT result_value FROM Results WHERE result_id = "
            + result.fk_results_id;
        string json = (string) await crud.Read(sql, ModelNames[selection]);
        DatabaseCrud.JsonResult value =
            JsonUtility.FromJson<DatabaseCrud.JsonResult>(json);
        int resultValue = value.resultResult[0].result_value + 1;
        crud.DbCreate("UPDATE Results SET result_value = " + resultValue +
            " WHERE result_id = " + result.fk_results_id);
    }
    crud.DbCreate(
        "INSERT INTO ResultQA (fk_result_id, fk_question_id, fk_answer_id)" +
        " VALUES (" + result.fk_results_id + ", " + result.fk_question_id +
        ", " + result.fk_answer_id + ")");
}
```

## Evaluation

Throughout this project numerous challenges needed to be overcome, the first of which was digesting the third-party asset tools such as uMMORPG. While their documentation (uMMORPG, 2018) presented useful information to get started, there was limited guidance for building beyond the scope of implementing custom functionality. For example, trying to obtain the users account name proved difficult from the client machine perspective, which returned the account name of the user on the host machine. To work around this, a copy of the users account name was taken at login and stored, to ensure data being stored related to the correct user.

The main aspect useful from the uMMORPG tool, was its ability to create the Server / Client connection. In hindsight creating a tool like this manually would have provided much greater control and knowledge over the implementation, although the personal challenge was to utilise existing code and manipulate it to get the desired result.

The majority of time was focused on the database area, where custom data related to Subject, Quiz and Lectures needed be stored. Ideally the issues with the client accessing the game server database, would have been picked up early but due to the wait time in creating builds, most of the testing was done in the Unity Editor. This assumption lead to most of the database class being re-written, targeting a secondary database hosted online. This involved upgrading the project to .NET 4.0 and utilising new features such as Async and Await.

Two third of the original requirements, Quizzes and Lectures, were implemented successfully. Unfortunately, due to the rebuild of the database, the workshops are still under construction and will be applied to a future iteration of this project.

### Future Improvements

- Workshop implementation
- Tutorial Quests
- Group Quests
- Integrate UMA 2 for Character Creation
- Server hosting / Rebuild of the Network Manager

## References

### References

Argyriou, V., Sevaslidou, M., & Zafeiriou, S. (2010). *Virtual university as a role playing game*. Education Engineering (EDUCON), 2010 IEEE, pp. 743-747.

FrogPlay (2018) *FrogPlay - Now, you can learn and play at the same time*. Available at: <https://www.frogplay.my/> (Accessed: 20 September 2018)

Johnson, G. (2018) *DMK Final Project Proposal*. Available at: [https://gordon.johnson7.co.uk/report/FP\\_proposal.pdf](https://gordon.johnson7.co.uk/report/FP_proposal.pdf) (Accessed: 20 September 2018)

Mojang (2018) *Minecraft Education Edition*. Available at: <https://education.minecraft.net/> (Accessed: 20 September 2018)

Taiga (2018) *Taiga.io – Love your Project*. Available at: <https://taiga.io/> (Accessed: 14 September 2018).

### Technical References

GitHub (2018) *Git Hub Inc*. Available at: <https://github.com/> (Accessed: 06 May 2018)

Gladius Studios (2016) *UMMORPG Tutorial Character Swap*. Available at: <https://www.youtube.com/watch?v=SnWs1PsXnFk> (Accessed: 25 May 2018)

JetBrains (2018) *ReSharper - Visual Studio Extension for .NET Developers*. Available at: <https://www.jetbrains.com/dotnet/> (Accessed 06 May 2018)

Microsoft (2018) *Visual Studio IDE*. Available at: <https://www.visualstudio.com/vs/> (Accessed 06 May 2018)

Stack-overflow (2014) *Strip Byte Order Mark from string in C#*. Available at: <https://stackoverflow.com/questions/1317700/strip-byte-order-mark-from-string-in-c-sharp> (Accessed 09 September 2018)

Unity Answers (2001) *Making a timer (00:00) minutes and seconds*. Available at: <https://answers.unity.com/questions/45676/making-a-timer-0000-minutes-and-seconds.html> (Accessed 02 September 2018)

### Asset References

Asobiya (2018) *Asobi-chan free*. Available at:

<https://assetstore.unity.com/packages/3d/characters/humanoids/asobi-chan-free-116360>

(Accessed: 01 June 2018)

Bensound (2018) *Adventure / Royalty Free Music*. Available at:

<https://www.bensound.com/royalty-free-music/track/adventure> (Accessed: 20 September

2018)

Game Asset Studio (2018) *Taichi Character Pack*. Available at:

<https://assetstore.unity.com/packages/3d/characters/taichi-character-pack-15667> (Accessed:

24 May 2018)

Modest Tree Media (2018) *Async Await Support*. Available at:

<https://assetstore.unity.com/packages/tools/integration/async-await-support-101056>

(Accessed: 09 September 2018)

Studio New Punch (2016) *Man in a Suit*. Available at:

<https://assetstore.unity.com/packages/3d/characters/humanoids/man-in-a-suit-51662>

(Accessed: 01 June 2018)

Tirgames Assets (2018) *School Scene*. Available at:

<https://assetstore.unity.com/packages/3d/environments/urban/school-scene-66006> (Accessed:

06 May 2018)

uMMORPG (2018) *Documentation – uMMORPG*. Available at:

<https://ummorpg.net/documentation/> (Accessed: 06 May 2018)

Unruly Games (2017) *Simple UI*. Available at:

<https://assetstore.unity.com/packages/2d/gui/icons/simple-ui-103969> (Accessed: 11

September 2018)