Creating the Core Components of an Astronomy Lab in AR

THESIS

Presented to the Faculty of the Department of Physics and Astronomy
in Partial Fulfillment of the Major Requirements
for the Degree of

BACHELOR OF SCIENCE IN
PHYSICS

Isaac Shirk

May 2021

(Creating the Core Components of an Astronomy Lab in AR)

(Isaac Shirk)

Signature of Author:

____

Department of Physics and Astronomy
May 2021

Certified by:

____

Dr. John Wallin
Professor of Physics & Astronomy
Thesis Supervisor

Accepted by:

____

Dr. Ronald Henderson
Professor of Physics & Astronomy
Chair, Physics & Astronomy

ABSTRACT

Augmented Reality technology has developed tremendously in recent years, with some efforts being made to create educational content. The power of AR to visualize difficult concepts while maintaining an individual's ability to interact with their surroundings and environment has tremendous potential to be incorporated into classrooms and lab. The goals of this project were to create the software for one such lab and lay the foundation for the creation and testing of future AR labs.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Computer-powered visualization has been steadily improving over the last couple years. Improvements in rendering techniques, simulations, and display technology have all combined to allow for impressive visual displays in a variety of mediums. Out of all the technologies that have been developed, virtual and augmented reality have both had some of the largest growth. Virtual reality refers to a full field-of-view immersion into a virtual world, typically through the use of a headset that covers the eyes and blocks external visual input. Augmented reality combines virtual imagery with the real world, superimposing the two to make it look like the virtual objects are part of your actual surroundings. Examples of AR include the well known game Pokemon Go, but also include apps from furniture stores that let you see what a couch would look like in your own room. Augmented reality, or "AR", spans a broader spectrum of possibilities than virtual reality, especially when applied to education. While both technologies are powerful visualization tools for abstract or complicated 3D ideas, AR does not isolate users from each other. These features, along with its ability to make abstract concepts more intuitive through interactive visualizations, means that AR has an incredible potential for the classroom.

The use of AR for education has been developed quite a bit recently, but still has much room to grow. While some applications have been made for the purpose of education, there is still no software that allows for the creation of AR educational content by instructors, to be used teaching their own coursework. The AR development team at MTSU aimed to remedy that problem, as well as demonstrate the effectiveness of AR when used as a teaching aid for difficult concepts.

The focus of this paper is my effort and accomplishments in developing an AR lab for students to use in the classroom, and a system that allows instructors to create AR content for their courses. This involves designing systems to display information to students, creating methods for collecting student input, and making a way for those tools to be intuitively by instructors. My work consisted of creating a multiple choice question system, helping construct and integrate systems manipulating various media developed by other team members, and designing and developing the structure and flow of an astronomy moon phase lab. A secondary goal was also to make the developed components able to be expanded on in the future, so that they can be used by any instructor trying to make an AR lab for their class.

II. TECHNOLOGY

There are a variety of AR technologies available today. Perhaps the most common is the common smartphone. Using only a smartphone camera and screen, virtual content can be displayed superimposed on top of real-time footage. Perhaps the most well-known example is Pokemon Go, as well as Ikea's app that allows for furniture to be "placed" in one's home.

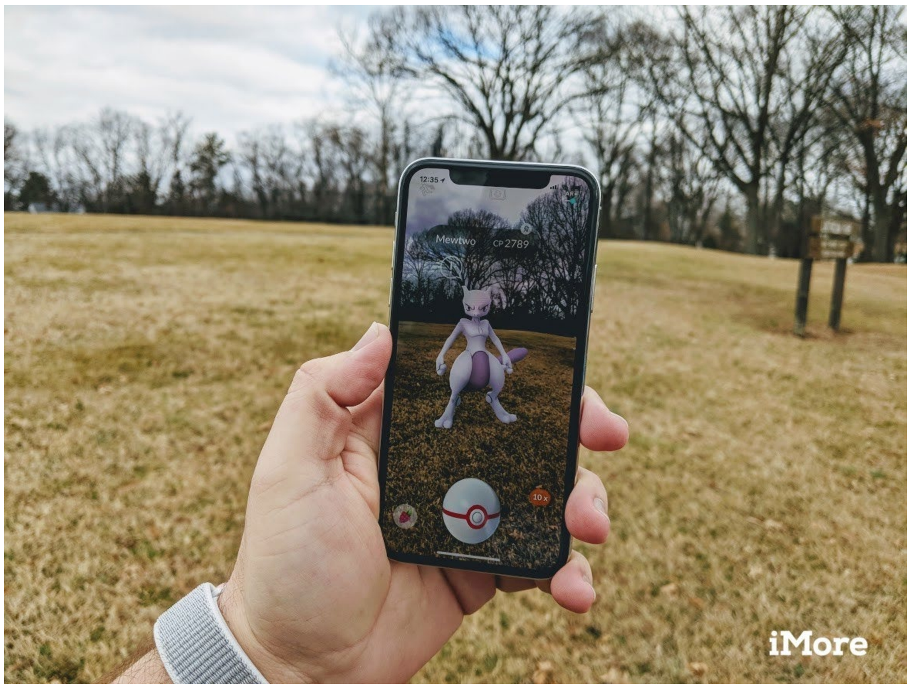


Figure 1. The most ubiquitous example of smartphone AR: Pokemon Go

While smartphone AR is certainly very accessible, and would ensure that every student would have the ability to do an AR experience using their own smartphones, it still falls short in a number of ways. First, not every student will have a compatible device, and providing compatible devices would be difficult. Furthermore, smartphone AR has a very restrictive field of view, very few controls, and is less engaging than the experience that a headset AR device would provide. The Microsoft Hololens is a popular headset AR device, but previous experiments in the library showed that it was difficult to work with, and did not perform well.

We decided that the best technology to implement our goals for an educational AR application was a headset called the Magic Leap One. This headset is made by a company called Magic Leap, and has all the tools we need to create educational AR content. The headset's display runs at a high refresh rate of 120Hz, with a 50

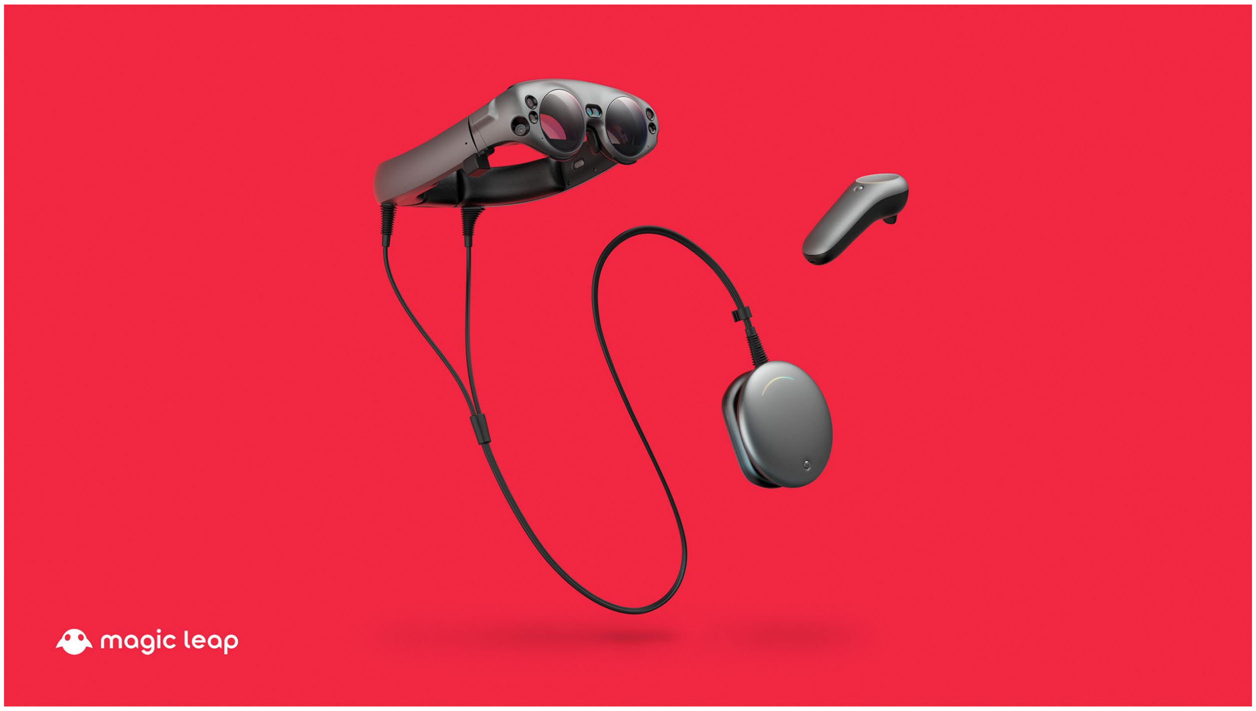


Figure 2: An image of the Magic Leap One, consisting of the headset, lightpack, and controller.

degree field of view. The high refresh rate helps the visuals feel more natural, and be less likely to cause motion sickness. The display area is an improvement over previous AR headsets like the Hololens 1, allowing for more immersive AR content. However, There are still large areas of one's peripheral vision where the AR visuals cut off because the screen simply still isn't big enough. This is a limitation that must be considered when designing applications that cover large spaces.

The headset also tracks your eye movement, and can even estimate at what point in 3-D space you are focusing on. The Magic Leap 1 also tracks your surroundings, having a number of outward facing cameras that it uses to constantly scan your environment. Using these scans, the Magic Leap can have digital content interact with the real world in believable ways. For example, if I throw a virtual ball, it could bounce off the physical wall, bump into a chair, and get stuck in a trashcan as if it was actually there. Unlike virtual reality devices, this means the Magic Leap can be used in a complex environment like a classroom without the fear of students running into something, because the classroom is part of the application. The cameras can also track the user's hands and what gestures they are making, allowing for intuitive application control and interaction.

While using only your hands is fun, The Magic Leap also has a controller with a touchpad, trigger, and button. The controller's position is tracked, and allows for even easier interaction with an Augmented reality application. The controller links via bluetooth to a small round computer called the lightpack. The lightpack is hung from a user's shoulder, and connects to the headset by a cord, and powers the AR experience. Overall the hardware is light, has a respectable battery life, and works well.

With the hardware chosen, the next step was to determine what to use to develop the software. Magic Leap had tools for development using a game engine called Unity. Unity comes with many, many tools built in. It allows you to manage virtual objects, show images and video, and write and attach scripts to manage how the virtual objects behave and interact. Unity is implemented in C++, but the language used to write custom scripts is C#.

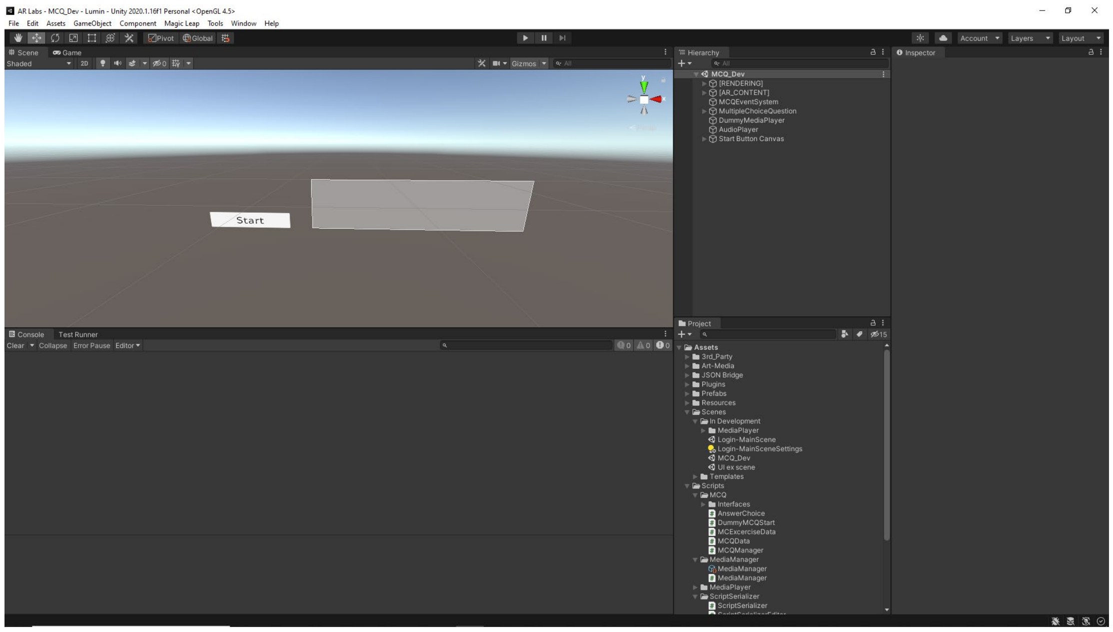


Figure 3: An example screenshot of the Unity editor

Developing for the Magic Leap using Unity had its many challenges. Unity publishes updates for the editor every 2 weeks or less, and produces a large update every couple months. This can produce conflicts with the versions of the tools used to develop for the Magic Leap. Magic Leap also has limited documentation troubleshooting common errors, and they shut down their forums. The combined effect is that there is limited access to helpful information when issues arise during the development process. Numerous bugs have come and gone with changing versions, as well as large changes to the structure of the tools provided by Magic Leap, causing large slowdowns and periods of relearning how to develop for the device.

III. Goals

With the technology chosen, the AR lab was made and tested with students as soon as possible so we could learn how it impacted their learning and how we could improve our educational content. The first lab covers moon phases, their names and appearances, and how they are caused by the moon's motion around Earth. The lab was to begin with an introductory video, explaining the content to be learned, followed by a collection of multiple choice questions. This necessitated the creation of at least three major components for the application: a media player to show videos, display images, and play audio; a multiple choice question object to show text questions; and an overarching control object that governed the order of the lab.

This lab is part of a larger NSF grant-funded project that aims to produce many AR labs to demonstrate the effectiveness of the technology, and to produce a framework for easily producing more educational AR content. Since we knew that we wanted to reuse the components we created to eventually make new labs, we needed every component to be more general and reusable. These building blocks that made the astronomy lab had to be created in a way that they could be reconfigured and used for an entirely different lab, without having to edit any code.

IV. DESIGN

The basic design of the lab is a sequence of exercises bookended by an introductory video and concluding slide. The introductory would be displayed by a media player that got reused throughout the lab to show many different images, videos, and audio. The exercises are designed to be self contained and independent, so they could be done in any order. Some common exercises that were developed were multiple choice questions, sorting, and interactable 3D objects. Finally, an overarching script would control the flow of the lab. While for the moon phase lab it would be inflexible, it had to be made in a way that would allow for flexibility to be easily added. All of these parts depend on images, videos, and audio files packaged with the lab. This design can be seen in “Phase 1” of the figure below.

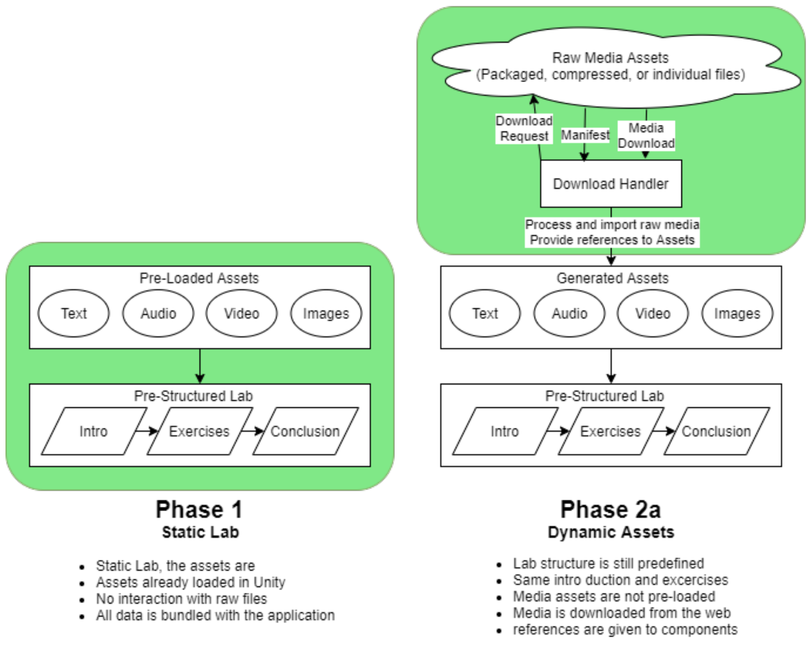
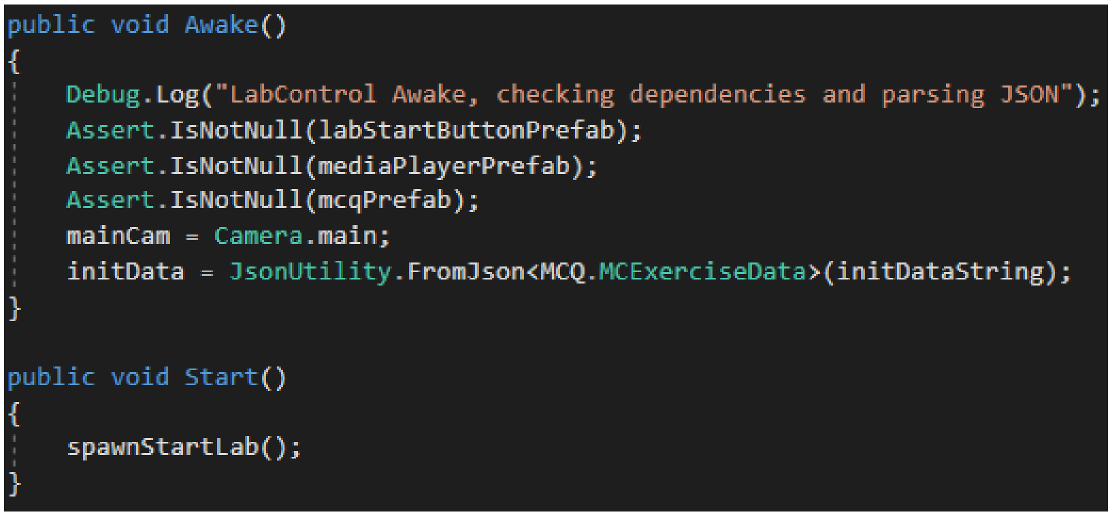

Raw Media Assets
(Packaged, compressed, or individual files)
Download Request
Manifest
Media Download
Download Handler
Process and import raw media
Provide references to Assets
Pre-Loaded Assets
Text
Audio
Video
Images
Pre-Structured Lab
Intro
Exercises
Conclusion
Generated Assets
Text
Audio
Video
Images
Pre-Structured Lab
Intro
Exercises
Conclusion
Phase 1
Static Lab
Static Lab, the assets are
Assets already loaded in Unity
No interaction with raw files
All data is bundled with the application
Phase 2a
Dynamic Assets
Lab structure is still predefined
Same intro duction and excercises
Media assets are not pre-loaded
Media is downloaded from the web
references are given to components

Figure 4: The AR project development plan as an infographic

V. IMPLEMENTATION

The moon phase AR lab has many parts that act individually, so in order to have it behave in an overall cohesive manner something must coordinate all of the individual pieces. The script in charge of this is called the lab manager, or LabControl as it appears in the code. The lab as a whole can be split into smaller modules, each consisting of a self contained task. The module that I created is a multiple choice exercise, and the script that controls it is called MCQManager. The lab also displays various forms of media, from images, to video and audio. The object that performs this task is called the media player, or AudioPlayer as it appears in the code (it's misleading name resulted from a simpler audio player being expanded to handle all types of media). I did not create the original implementation of the AudioPlayer, but contributed to it heavily.

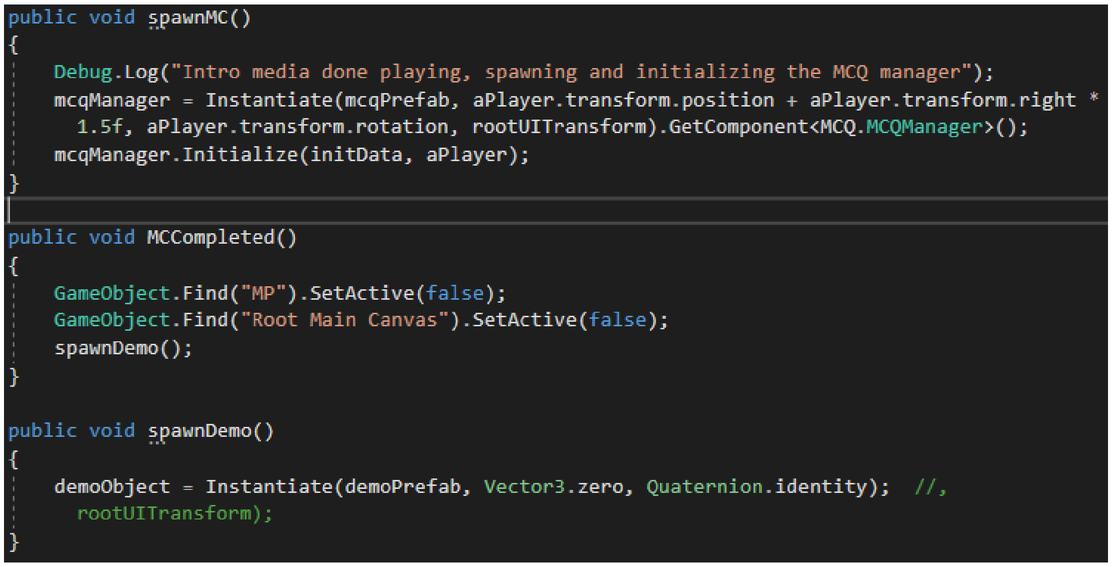
LabControl is responsible for how the application behaves at the start of the lab, in between each section, and at the end of the lab. LabControl stores references to copies of all the parts the lab application needs in order to run smoothly, and at the highest level the LabControl simply creates copies of these objects and scripts, tells them to run, and waits for them to complete before creating a copy of the next object.



```
public void Awake()
{
    Debug.Log("LabControl Awake, checking dependencies and parsing JSON");
    Assert.IsNotNull(labStartButtonPrefab);
    Assert.IsNotNull(mediaPlayerPrefab);
    Assert.IsNotNull(mcqPrefab);
    mainCam = Camera.main;
    initData = JsonUtility.FromJson<MCQ.MCExerciseData>(initDataString);
}

public void Start()
{
    spawnStartLab();
}
```

Figure 5: The variable check, and initial function call

When the lab application is first opened, LabControl checks its variables to ensure that none of them contain invalid values. It then generates a data object that will be covered in more detail later on. Then, it calls a function called spawnStartLab(). While I did not write that function, it follows the same general pattern as the other functions in the script. First a spawn function is called telling LabControl to create a new object, set up and start the script controlling the object, and wait for the newly created object and script to tell LabControl it is done. Then the process repeats until the lab is done.

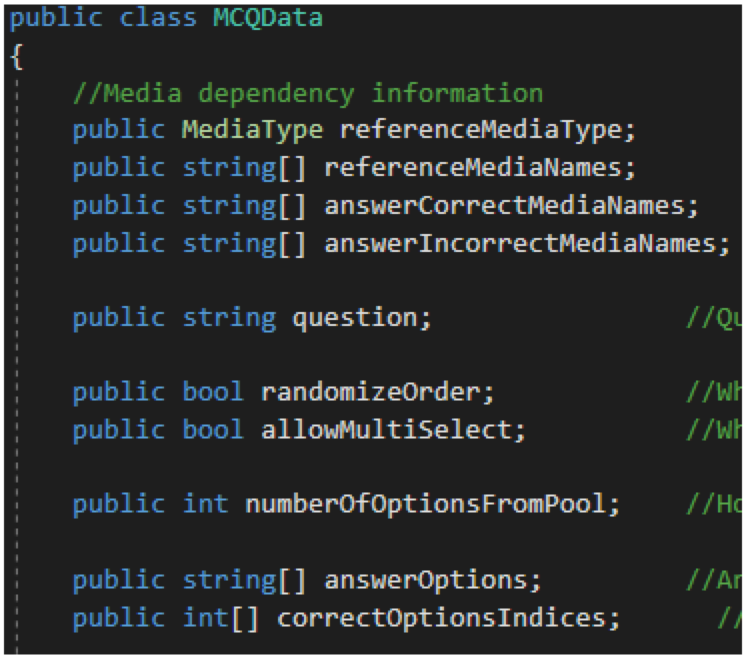


```
public void spawnMC()
{
    Debug.Log("Intro media done playing, spawning and initializing the MCQ manager");
    mcqManager = Instantiate(mcqPrefab, aPlayer.transform.position + aPlayer.transform.right *
      1.5f, aPlayer.transform.rotation, rootUITransform).GetComponent<MCQ.MCQManager>();
    mcqManager.Initialize(initData, aPlayer);
}

public void MCCompleted()
{
    GameObject.Find("MP").SetActive(false);
    GameObject.Find("Root Main Canvas").SetActive(false);
    spawnDemo();
}

public void spawnDemo()
{
    demoObject = Instantiate(demoPrefab, Vector3.zero, Quaternion.identity);  //,
      rootUITransform);
}
```

Figure 6: An example of LabControl's design pattern. A spawn function creates an object, a "completed" function indicates the new object is done running, and the next "spawn" function is called.

The design pattern is shown above. First, LabControl calls spawnMC() which creates a new object with a MCQManager script controlling it. Then it sets up and starts the newly created script by calling a function called "Initialize". How the MCQManager runs will be discussed later. Once it completes running, MCQManager calls the function called "MCCompleted", which tells LabControl that the multiple choice module of the lab is complete. It then disables the objects related to the multiple choice module and calls the next spawn function, in this case "spawnDemo". Once the last module is complete, in this case a sorting task made by Dr. Wallin, LabControl closes the application, concluding the lab.

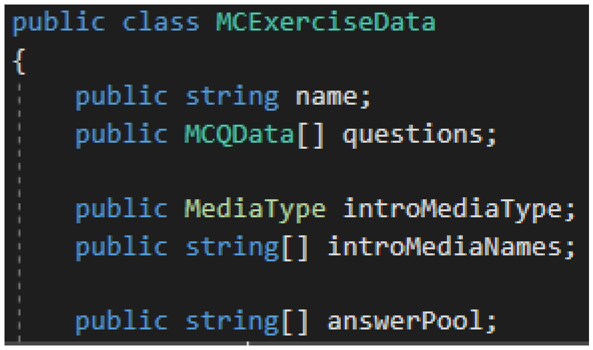
The first module that runs after playing an introductory video is my multiple choice question module, controlled by the MCQManager script. MCQManager depends on a few other small scripts in order to run, and so they must be understood before delving into how MCQManager runs.



```
public class MCQData
{
    //Media dependency information
    public MediaType referenceMediaType;
    public string[] referenceMediaNames;
    public string[] answerCorrectMediaNames;
    public string[] answerIncorrectMediaNames;

    public string question;              //Qu

    public bool randomizeOrder;          //Wh
    public bool allowMultiSelect;        //Wh

    public int numberOfOptionsFromPool;  //Ho

    public string[] answerOptions;       //Ar
    public int[] correctOptionsIndices;    //
```

Figure 7: All the MCQData script is is a collection of variables

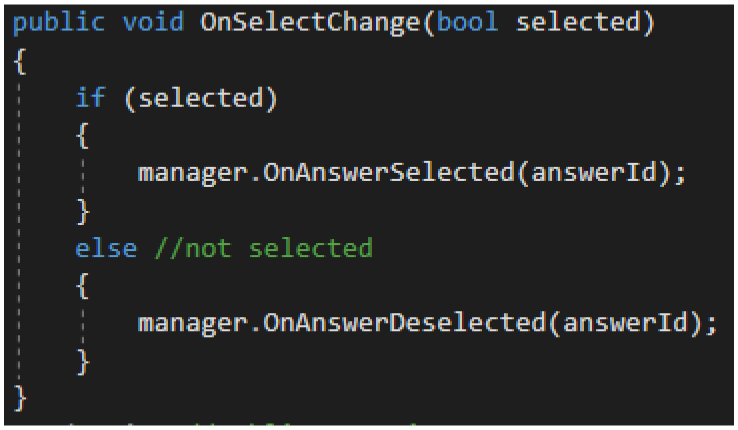
Perhaps the most important of these small subscripts is MCQData. The script is simply a container of variables describing a single multiple choice question. All of the variables containing "MediaNames" in their identifier contain filenames of videos or audio to play at certain points during the question. The "question" variable simply contains the question that will be displayed and that the student must answer. The following two variables either be true or false, and identify if the order of the answers should be scrambled every time the question is displayed, and if multiple answers can be selected at once. “answerOptions” is a list containing the answers options that the student can pick from, while the “correctOptionsIndices” is a list identifying all of the right answers in the “answerOptions” list. The “numberOfOptionsfromPool” variable identifies how many extra options to add from a predefined set of options, which is expanded on more in the description of the next script. All of these variables combined comprehensively define all the data needed to show a single multiple choice question. By defining all the characteristics in code, we can then store a description of each question as a string of characters that can be easily modified and sent over the internet. This allows for someone to customize a question simply by editing a small text file, rather than having to download and modify the source code for the entire lab application.



```
public class MCExerciseData
{
    public string name;
    public MCQData[] questions;

    public MediaType introMediaType;
    public string[] introMediaNames;

    public string[] answerPool;
```

Figure 8: Another collection of variables, this time with a list of MCQData objects

The MCExerciseData script is another collection of variables similar in function to MCQData, but instead of defining a single multiple choice question, it defines the entire exercise. It does this mainly by having a list of MCQData objects that define each question in the exercise. Another important variable is “answerPool”, which stores a list of predefined answers that can be pulled from at random. This is a useful, but not required component for any multiple choice exercise where all the answer options are from the same small group of possible answers, in this case eight moon phases. Then, a question can be completely defined by identifying the right answer, say “Full Moon”, and then saying “use three other moon phases as the wrong answers”. In order to fill in those “three other phases”, MCQManager uses the “answerPool” variable. Finally, there are also variables that name the exercise and what media should be played at the start of the multiple choice module.

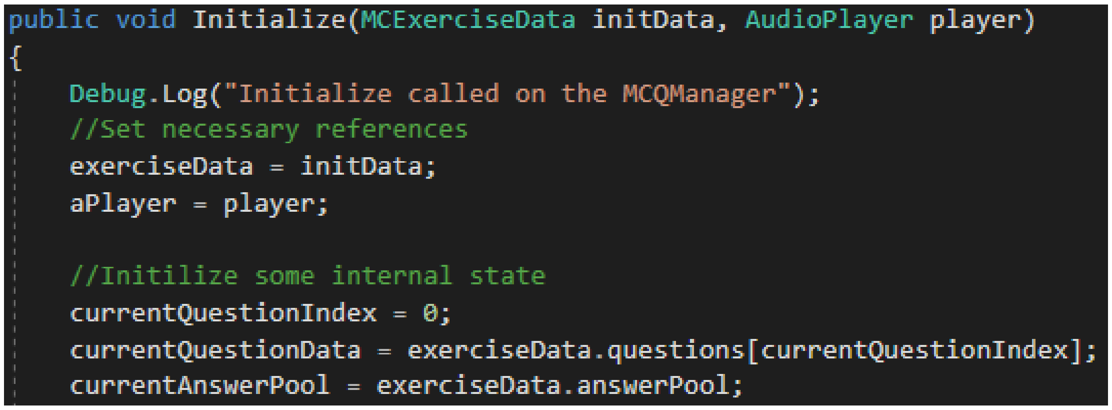


```
public void OnSelectChange(bool selected)
{
    if (selected)
    {
        manager.OnAnswerSelected(answerId);
    }
    else //not selected
    {
        manager.OnAnswerDeselected(answerId);
    }
}
```

Figure 9: This method responds to a toggle being pressed, and contact LabControl

The simplest script MCQManager depends on is called “AnswerChoice”. It has some other small functions, but the most important one is displayed in the figure above. The function “OnSelectChange” is called every time a student clicks on an answer choice, and simply relays which answer choice has been selected or deselected to MCQManager. MCQManager uses this information to grade the student.

With all its dependencies discussed, MCQManager can now be delved into. The script is verging on 500 lines, and while some of that is due to many comments and spacing to maintain clarity, it contains a lot of functionality. As such, only some of its most critical points will be covered here.

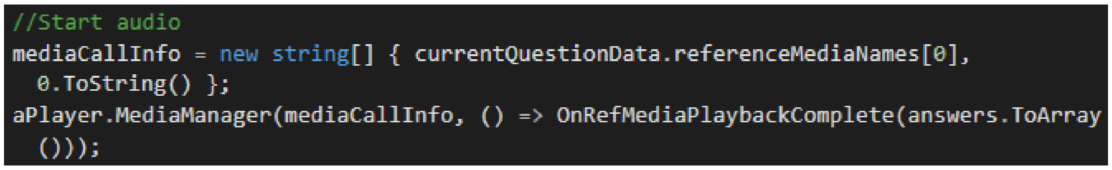


```
public void Initialize(MCExerciseData initData, AudioPlayer player)
{
    Debug.Log("Initialize called on the MCQManager");
    //Set necessary references
    exerciseData = initData;
    aPlayer = player;

    //Initilize some internal state
    currentQuestionIndex = 0;
    currentQuestionData = exerciseData.questions[currentQuestionIndex];
    currentAnswerPool = exerciseData.answerPool;
```

Figure 10: This function is what starts the MCQManager, and requires a data object

The first function that is called on MCQManager is called “Initialize”. It requires a MCExerciseData object, discussed earlier, and a reference to the AudioPlayer so that MCQManager can tell it to play media. The function reads the data object and sets up a number of variables using the values it contains. The variable “currentQuestionData” is used by other functions to display the current question to the user, as well as to grade the students response. Once all of the variables have been properly initialized, MCQManager actually begins the processing of the first question by calling a function named “SetupNextQuestion”.

This function is rather long, and can’t be pictured here legibly. Its first task is to generate the actual list of answers using the bits of information in the current MCQData and MCExercise. It checks what the correct answer is, whether it should randomly add more answers to the list from the answer pool, and whether it should shuffle the list of answers. It then notes the location of the correct answer in the final list of answer choices. The last thing it does is check if the question has some reference media to play to the user to help with the question, as displayed in the figure below. MCQManager then calls a function named “DisplayNextQuestion”.

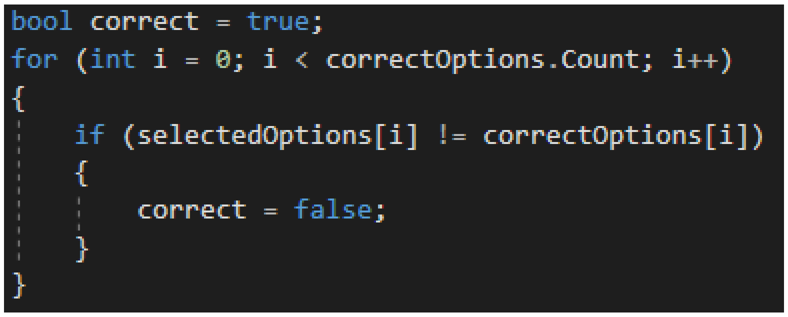


```
//Start audio
mediaCallInfo = new string[] { currentQuestionData.referenceMediaNames[0],
  0.ToString() };
aPlayer.MediaManager(mediaCallInfo, () => OnRefMediaPlaybackComplete(answers.ToArray
  ()));
```

Figure 11: All the information the media player needs is put in a package named “mediaCallInfo” and then given to the media player for playback.

The function “DisplayNextQuestion” puts the question text on the user interface for the user to read, then adds the correct number of answer options as specified by the MCQData object. Finally, with the function fully displayed. MCQManager is ready to react to the user selecting and deselecting options, and finally submitting their answer.

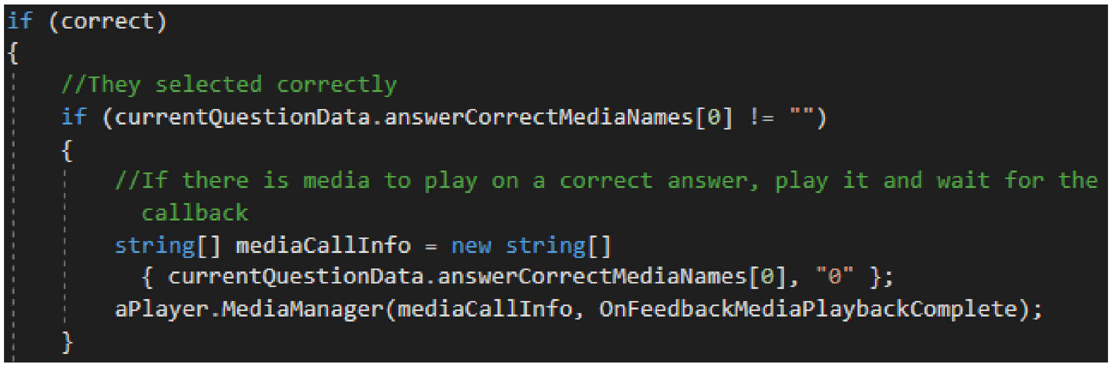
When the submit button is pressed, a function called “OnSubmitPressed” is called. First, it looks at what answers were selected, and compares them to the correct answer it has stored. It then sets a boolean variable named “correct” to true or false depending on the accuracy of the student’s answer.



```
bool correct = true;
for (int i = 0; i < correctOptions.Count; i++)
{
    if (selectedOptions[i] != correctOptions[i])
    {
        correct = false;
    }
}
```

Figure 12: The function starts assuming the student is correct, but switches to incorrect as soon as one of the selected options is incorrect.

No matter if the student got the answer right or wrong, MCQManager checks if there is an audio cue that it should play as feedback. If the student was right then it checks for positive feedback, and if the student was wrong it checks for corrective feedback. If it finds the name of a file to play then it passes it on to the media player in order to display to the user. Once the media player finishes, then the user can continue on to the next question.

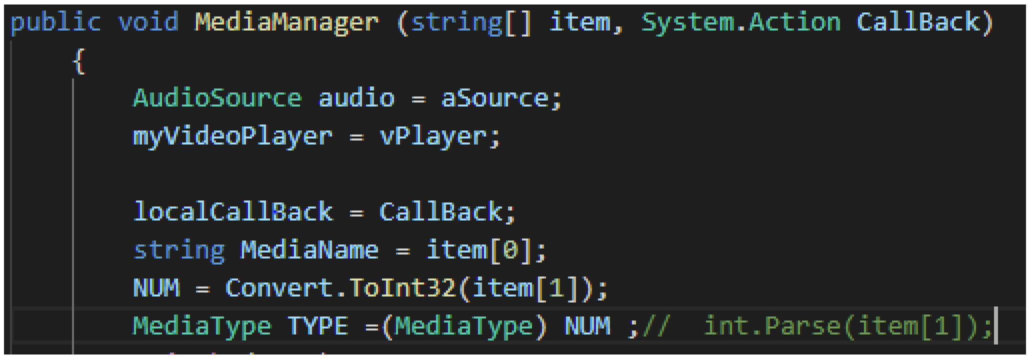


```
if (correct)
{
    //They selected correctly
    if (currentQuestionData.answerCorrectMediaNames[0] != "")
    {
        //If there is media to play on a correct answer, play it and wait for the
          callback
        string[] mediaCallInfo = new string[]
          { currentQuestionData.answerCorrectMediaNames[0], "0" };
        aPlayer.MediaManager(mediaCallInfo, OnFeedbackMediaPlaybackComplete);
    }
```

Figure 13: A check is done to see if there is a media file to play if the student was correct, and passes it to the media player to show to the user.

Once the student continues to the next question, then the functions discussed previously are called again, but with new question data. This repeats until MCQManager runs out of MCQData objects to read from, at which point it presents an end-of-module screen to the user indicating completion of the multiple choice section. Finally, the MCQManager reports that it has completed running to LabControl, which then removes MCQManager and moves onto the next portion of the lab. In the case of the moon lab, it moves onto the 3D demo created by Dr. Wallin.

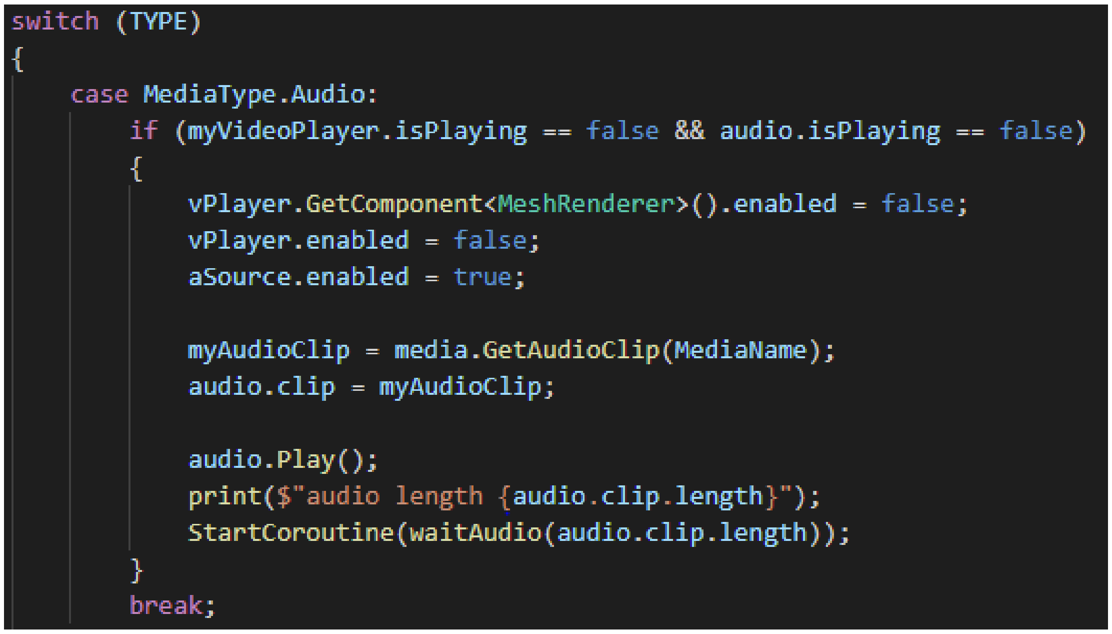
Both LabControl and MCQManager need to play media at various points during the lab. This is done by the media player, whose core functionality was written by Dr Rafet Al Tobasi, while I wrote some edits and additions. It links to video, audio, and image components provided by Unity, which it can then control or even turn on and off individually.



```
public void MediaManager (string[] item, System.Action CallBack)
    {
        AudioSource audio = aSource;
        myVideoPlayer = vPlayer;

        localCallBack = CallBack;
        string MediaName = item[0];
        NUM = Convert.ToInt32(item[1]);
        MediaType TYPE =(MediaType) NUM ;//  int.Parse(item[1]);
```

Figure 14: The function call that tells the media player to play a sound or show a video.

The media player allows other objects to tell it to play media by giving it the name of the file and what type of media it is: audio, image, or video.



```
switch (TYPE)
{
    case MediaType.Audio:
        if (myVideoPlayer.isPlaying == false && audio.isPlaying == false)
        {
            vPlayer.GetComponent<MeshRenderer>().enabled = false;
            vPlayer.enabled = false;
            aSource.enabled = true;

            myAudioClip = media.GetAudioClip(MediaName);
            audio.clip = myAudioClip;

            audio.Play();
            print($"audio length {audio.clip.length}");
            StartCoroutine(waitAudio(audio.clip.length));
        }
        break;
```

Figure 15: This is where the script determines what type of media to play, finds the correct file, and starts the playback.

The media player then looks for the file using another class called the media manager. After finding the file, the media player gives it to the correct component, such as giving the video player the “Introduction.mp4” file to play. The media player also links to UI buttons to give the student more control over playback of the media. The play button starts any paused component, and the pause button does the inverse. A special handler function is responsible for making a slider bar behave like a scrubbing bar for a video or song, jumping playback to the point you drag it to.

Once the media player detects that it has finished playing the media it was assigned, it calls a function that notifies the sender of the task’s completion. For the beginning of the lab this means that once the introductory video has completed, the media player notifies LabControl, which then knows to spawn the multiple choice question module.

VI. EXAMPLES

The following images are snapshots of what students would see and interact with during the course of the lab. Emphasis will be on the components I had the most direct input in creating. Note, the image will be of the Magic Leap simulator, not through an actual headset. This tool allows me to quickly test new code I have written without having to repeatedly don a physical headset. This is why the room in the background is a digital generation.

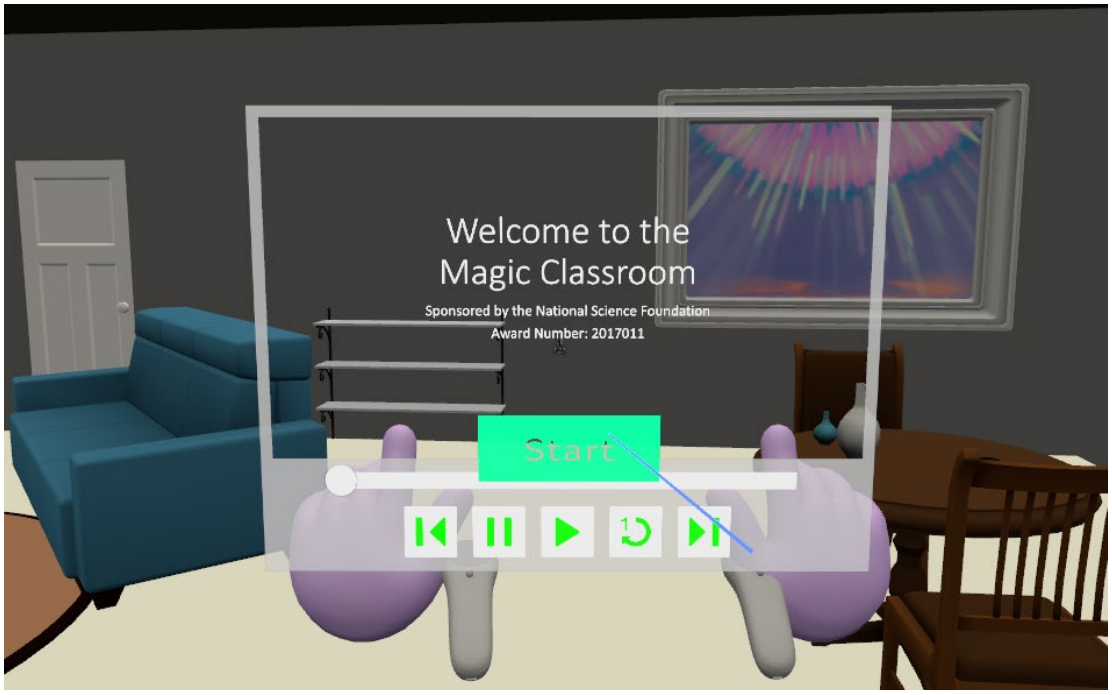


Figure 16: The introduction slide follows the user until the start button is selected.

The first image is of the first thing the student sees when they launch the lab application. The menu will smoothly follow their head's direction and location, allowing the student to find a comfortable spot before selecting the start button. At that point, the media player will lock into place and start playing the introductory video, introducing most of the concepts from the lab. After this video completes, the MCQ module is created.

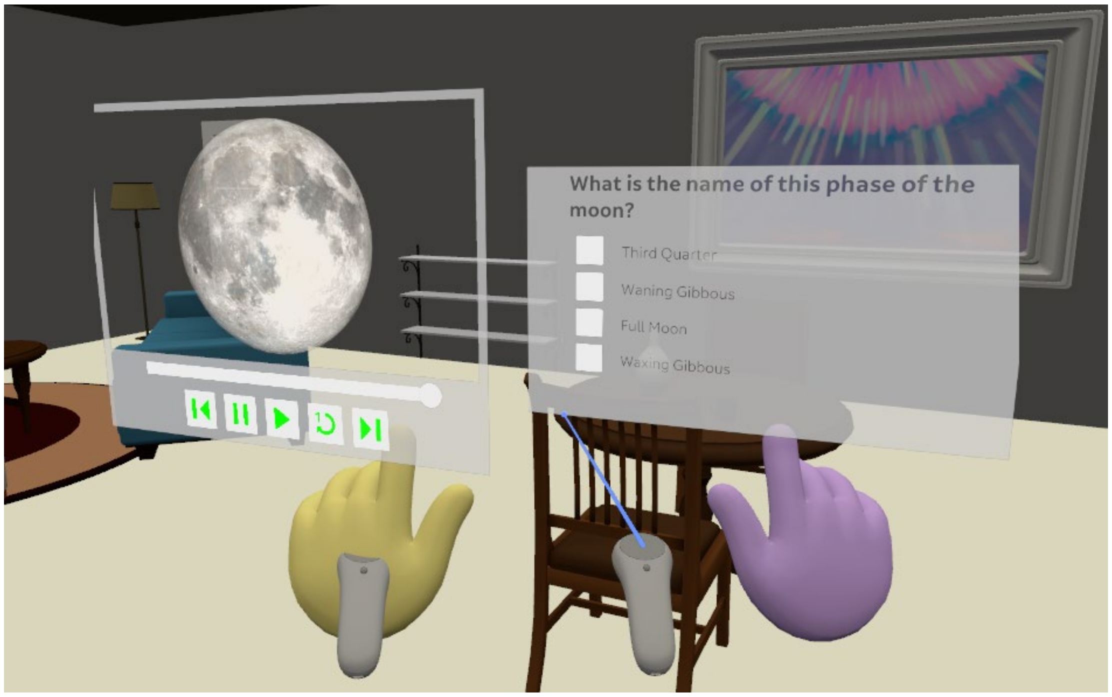


Figure 17: In the first MCQ, the student is asked to identify a picture of a full moon.

Once the MCQ module is created, the media player shows an unlabeled picture of a moon phase. The media player also plays a short audio clip of Dr. John Wallin identifying some qualities of the image and asking the student to identify the image.

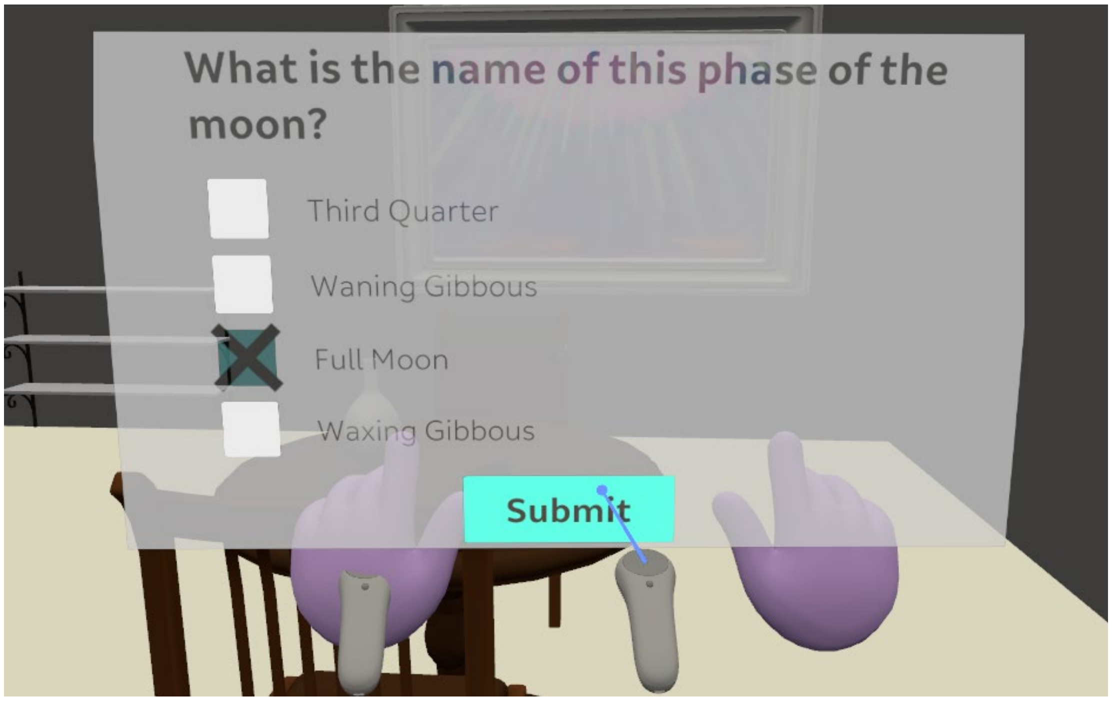


Figure 18: Once the student selects an answer, they can then submit and get feedback.

After viewing the image and listening to the audio prompt, the student can then select an answer and choose to submit it. Upon submitting their answer, the student will hear an audio cue commending their correct answer, or an explanation of why what they chose was incorrect. Students can then proceed to the next question and repeat the process until they complete the MCQ module. The following two exercises are a 3D visualization of the Earth-Sun-Moon system and a moon phase sorting task. This work was completed by another team member, and ill not be included here.

VII. FUTURE WORK

Future work can be broken down into two categories. The first being immediate revisions to the current moon phase lab, and the second being long term development plans. The moon phase lab has just finished going through preliminary testing, so that feedback will be discussed first. While the students using the app were generally very excited about it, and were very engaged with the material, there were many common points brought up, the first being the placement of the lab itself. When users start the moon phase lab, a menu and start button follow the user until the start button is pressed. After that point, the video player and multiple choice question section are all displayed in the location where the user pressed “start”. However, the task involving sorting moon phases is placed relative to the position of the device when it turns on.

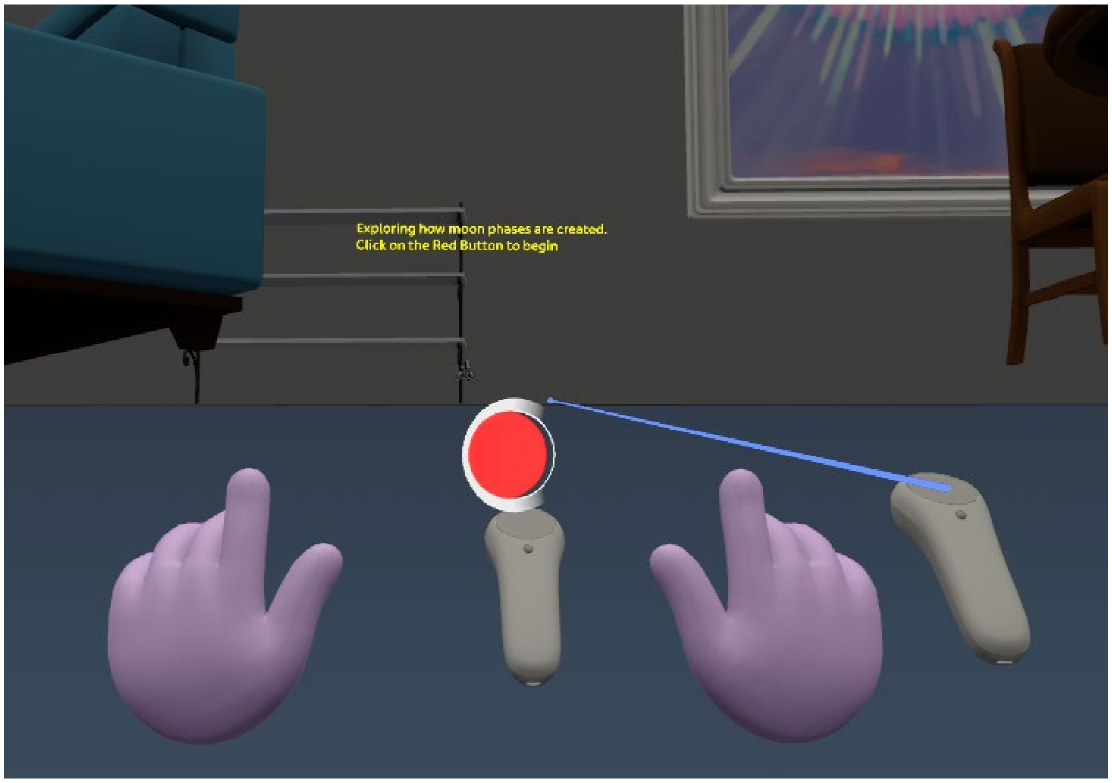

Exploring how moon phases are created.
Click on the Red Button to begin

Figure 19: The next module actually popped up "under ground" in the simulator.

This leads to the sorting task to be placed incorrectly, often being invisible to participants. As such, plans for a system to allow students to explicitly define a location in 3D space where all of the lab will take place will eventually be implemented. Another issue with the lab experience was that some parts of the lab flow irreversibly forward, meaning that once a student pushes a button, there is no going back to read the previous information, which some students wanted to do. A goal in the future will be to add the ability to navigate between sections of one module, say different questions in the MCQ module, and between entire modules.

The limited field of view of the Magic Leap headset also caused an issue. When students completed the introductory video, the MCQ module was created and placed next to the media player. However, it was often mostly or completely outside the view of the students, who could not see where the MCQ module was, and took more time than necessary trying to find it.

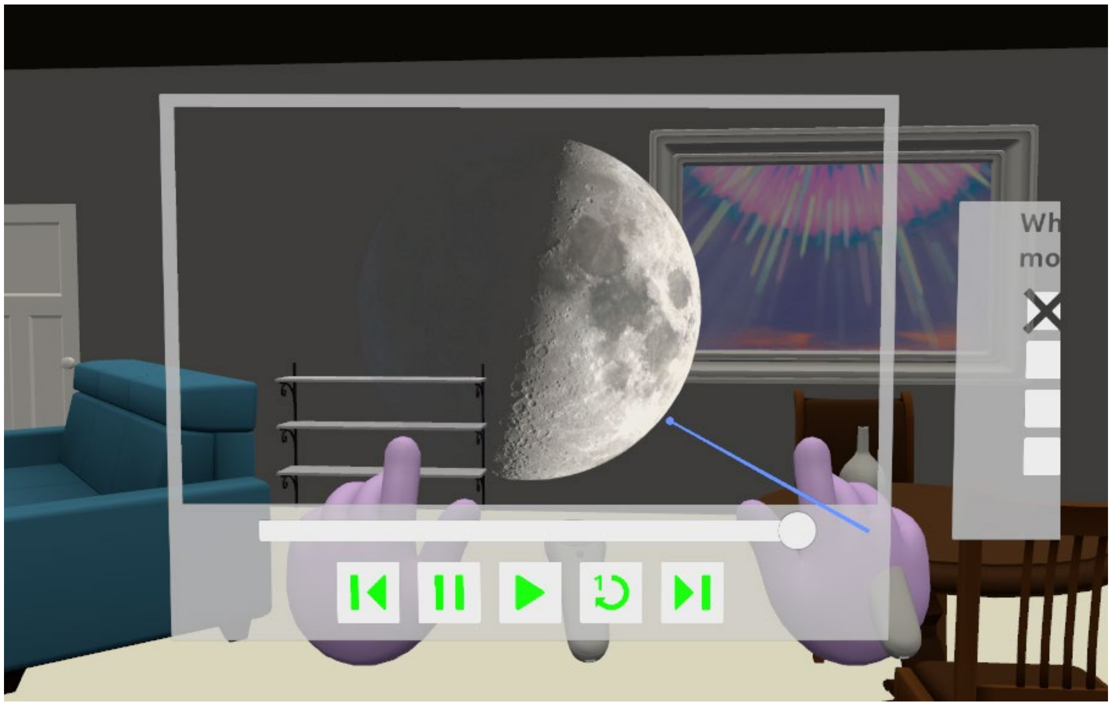
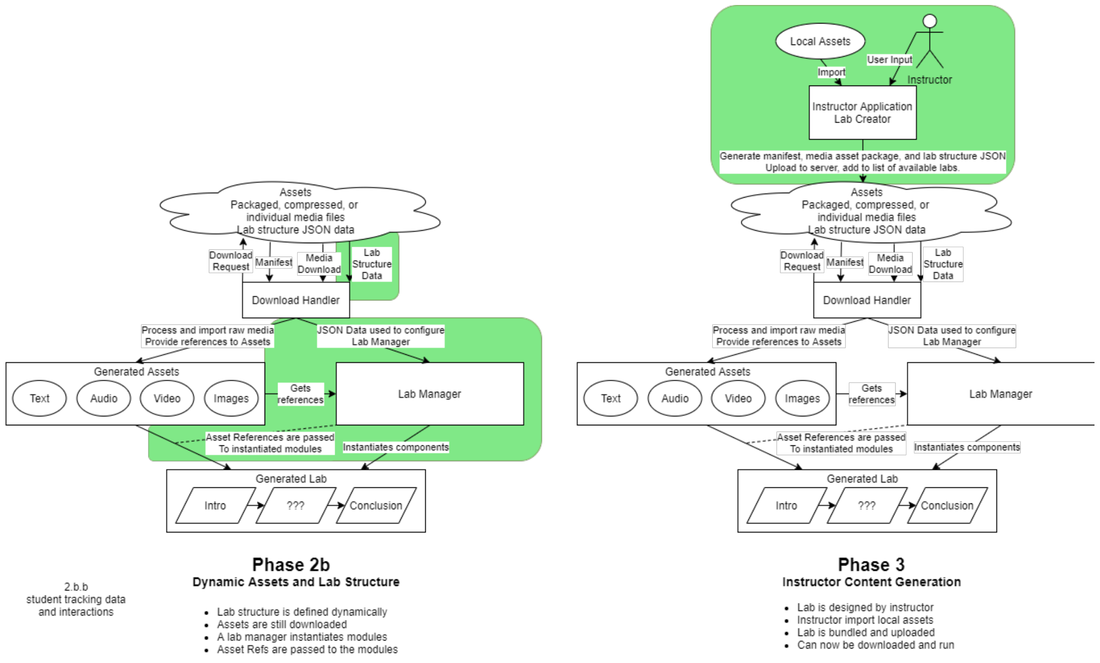


Figure 20: The multiple choice module shows up just barely to the right of the video.

Other feedback indicated a desire for students to log in to the lab experience. This would allow for individual grades, lab progress, and interactions statistics to all be tracked, benefiting lab instructors and lab developers alike. This will require investigating how to securely handle identity information, grades, and other sensitive data.



Assets
Packaged, compressed, or individual media files
Lab structure JSON data
Download Request
Manifest
Media Download
Lab Structure Data
Download Handler
Process and import raw media
Provide references to Assets
JSON Data used to configure Lab Manager
Generated Assets
Text
Audio
Video
Images
Gets references
Lab Manager
Asset References are passed To instantiated modules
Instantiates components
Generated Lab
Intro
???
Conclusion
Phase 2b
Dynamic Assets and Lab Structure
Lab structure is defined dynamically
Assets are still downloaded
A lab manager instantiates modules
Asset Refs are passed to the modules
2.b.b
student tracking data and interactions
Local Assets
Import
User Input
Instructor
Instructor Application
Lab Creator
Generate manifest, media asset package, and lab structure JSON
Upload to server, add to list of available labs.
Phase 3
Instructor Content Generation
Lab is designed by instructor
Instructor import local assets
Lab is bundled and uploaded
Can now be downloaded and run

Figure 21: The planned later stages of the AR project

The project has many, many plans for future development. The first priority is to make each module, i.e. the media player, MCQ, sorting task, etc. definable from a JSON file. That way A lab covering completely different material can be made by reusing existing building blocks, rather than taking the time to code it from the ground up. In line with that goal is the development of run-time media downloading. That is a bunch of words to simply say that the lab can change what videos, pictures, and audio it shows every time it starts up. Currently, all the media resources we are using have to be included with the app while we code it. By making a runtime media manager, it can be told what to download and use. This again allows for the construction of more labs dealing with different topics.

As all the reusable building blocks for creating AR labs become finalized, eventually the focus of the project will shift towards creating a way for end users, ie professors and lab instructors, to create their own AR labs. The goal will be for the ability to drag and drop modules to define a new lab flow, like a video presentation followed by a 3D interaction, then multiple choice questions and a conclusion video. Then the instructor can configure each module by inputting what questions should be asked, what the correct answers are, etc. Then the program they use will translate that into a data file that contains all of the information defining the lab. Then a student could select the lab from the headset, which would then download the data file and generate the lab. While still a ways off, this level of software sophistication could lead to the increased use of AR in the classroom. Any instructor could create new AR labs to engage with and teach their students, and students would benefit from the improved visualization of difficult concepts. This project is the first step towards that goal, and eventually AR could become a staple in the classroom.

VIII. REFERENCES

1. Ritchie, R. (2018, June 02). The best MULTIPLAYER Arkit 2.0 demo at WWDC 2018 would Be Pokémon Go PvP. Retrieved April 26, 2021, from https://www.imore.com/best-multiplayer-arkit-2-demo-pokemon-go
2. https://www.magicleap.com/en-us/press-resources