ARuko and Editour:

A Platform for Augmented Reality Tour Guide Apps

A Major Qualifying Project
Submitted to the faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

By
William Campbell
Cole Granof
Joseph Petitti

Date:
October 11, 2019

Report submitted to:

Atticus Sims, Sponsor
Kyoto VR

Professor Joshua Cuneo, Adviser
Worcester Polytechnic Institute

Professor Ralph Sutter, Adviser
Worcester Polytechnic Institute

Abstract

Kinkaku-ji, the Temple of the Golden Pavilion, is one of Kyoto's most visited tourist attractions. Most people who visit the site do not have the opportunity to experience the rich art and culture that are part of its history. We worked with our sponsor, Kyoto VR, to create an app that would allow visitors of Kinkaku-ji to better experience these aspects of the site. We used Unity to create an audio tour app that provides historical insight on the location, displays galleries of art throughout the visit, and offers accessibility features through AR. We also developed a tool to aid in the design and creation of audio tours. We hope our app will enhance the experience of visiting Kinkaku-ji.

Acknowledgments

Our team would like to thank Atticus Sims, CEO and founder of Kyoto VR, for sponsoring this project and providing help and guidance throughout our time in Japan. His knowledge of the history of Kinkaku-ji and Kyoto in general has been invaluable to our experience.

We would also like to thank Inoue Hikaru for his help with translations and researching AR technology. A special thank you to the IQP team working with Kyoto VR, Lewis Cook, Nicole Escobar, Ahad Fareed, and Cameron Person, for filming and editing the app trailer, as well as providing help with testing. Further, we are grateful to the assistance and resources provided by Ritsumeikan University and Professor Noma Haruo. Finally, a thank you to our advisers, Professors Joshua Cuneo and Ralph Sutter. Without their expert advice and feedback this project probably could not have been completed.

Executive Summary

Our team consists of three Senior Worcester Polytechnic Institute (WPI) students: William Campbell, computer science and interactive media and game development double major; Cole Granof, computer science major; and Joseph Petitti, computer science major. We spent twelve weeks in Shiga prefecture, Japan, completing the requirements for our Major Qualifying Project, a WPI graduation requirement in which students solve a real-world problem through practical project-based work. Our project was sponsored by Kyoto VR, a startup founded in 2016 to "document and preserve the culture and heritage of the city and communicate the richness of Japan's Old Capital to the world" [1].

Our team worked directly under Kyoto VR founder and CEO Atticus Sims to develop and test a mobile app to deliver an audio tour, incorporating various augmented reality (AR) features to enhance the experience. Our app delivers a tour of Kinkaku-ji, or The Temple of the Golden Pavilion, a popular tourist destination in Kyoto. In this app, which we call ARuko, many of the AR features were aimed at improving accessibility. These features include translating signs and providing additional info on the physical map located at the site. Other features were purely for spectacle, such as the ability to take an AR "souvenir" home by scanning the ticket to enter Kinkaku-ji.

ARuko went through multiple iterations of user testing and bug testing at Kinkaku-ji. Sections 4.3.1 and 4.3.2 go into detail about our two days of field testing on-site. From these tests, we were able to refine the user experience, and identify important features to add to the app. Much of the app's functionality was decided by our project sponsor, who plans to monetize the app. We also worked with an Interactive Qualifying Project (IQP) team, who had been tasked with exploring funding options and creating promotional content.

Since our app had to deliver an audio tour supplemented by images based on the user's latitude and longitude, we developed an internal tool to design physical regions for each step of the tour. This tool, called Editour, provides a rich editing suite through a simple web interface. Section 3.1.3 goes into detail about the features of this web app. We also wrote a backend to manage the tour files, allowing one of us (or our sponsor, Atticus Sims) to download a zip file containing all of the media and info to reconstruct the tour. This zip can be extracted into the source files for ARuko, which rebuilds the tour from the downloaded contents. We were able to host the backend on the server provided by Kyoto VR's hosting plan; we discuss the hurdles that we overcame to deploy the Editour backend with Kyoto VR's current setup in Section 3.1.6.

In Section 5, we also discuss the alternative AR app concepts that were explored before development became completely focused on ARuko.

Contents

List of Figures

List of Tables

1 Introduction

The field of consumer-oriented augmented reality (AR) technology is just coming into full swing, and it presents many unique opportunities for a myriad of fields. This budding technological platform uses digital displays to augment real-world information with computer-generated data (see Figure 1. As the technology progresses, several disparate industries have found novel uses for it, some becoming extremely profitable [2]. With smartphone manufacturers investing billions into AR technology [3], it seems poised to be the next big digital interface.

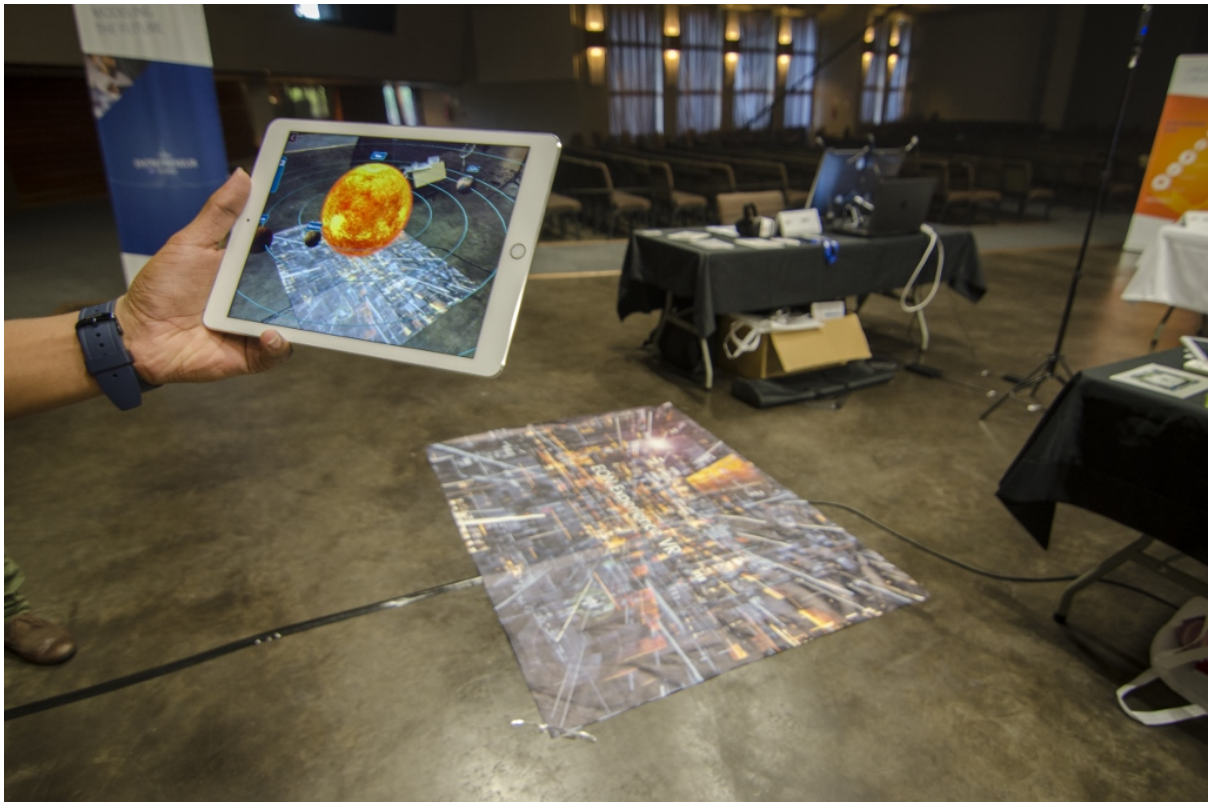

Figure 1: A virtual model of the solar system projected onto the real world using augmented reality [4]

Tourism and art are among the fields that could gain the most from AR technology [5, 6]. This technology can provide accessible information to tourists and art viewers that they would not be able to get any other way. Kinkaku-ji, the "Golden Pavilion" of Kyoto (see Figure 2, is the perfect example of a site that would benefit from AR [7].

Kinkaku-ji is a UNESCO World Heritage Site [9] and one of the most popular tourist destinations in Japan [10]. The Golden Pavilion is a magnificent sight to behold, but the gardens and grounds are the true beauty of Kinkaku-ji. It is a living masterpiece of Zen Buddhist landscape design, expanding on the natural beauty of Kyoto's forests and mountains through the practice of borrowed scenery [11].

However, much of the historic information at the site is lost on non-Japanese speaking visitors because it is often only available in Japanese. In addition to the language barrier, the information provided does not include most of the history and context necessary to fully understand Kinkaku-ji.

To solve this problem and open up the peerless beauty of the Golden Pavilion to more people, our

Figure 2: The Golden Pavilion [8]

team created ARuko and Editour. These two tools comprise a complete platform for designing and experiencing AR-enhanced tours. Editour makes it easy to design custom audio tours including audio, text, and images, while ARuko allows users to seamlessly experience historic sites in an intuitive and novel way. Many AR apps require specific markers, like the those shown in Figure 3. To avoid the need to place any physical object that would disturb the buildings or grounds of a historic site, our app does not rely on these markers. Instead, ARuko recognizes images of objects that are already at the site, and uses them to display informative AR pop-ups. See Sections 3.2.2 and 3.2.3 for more details.

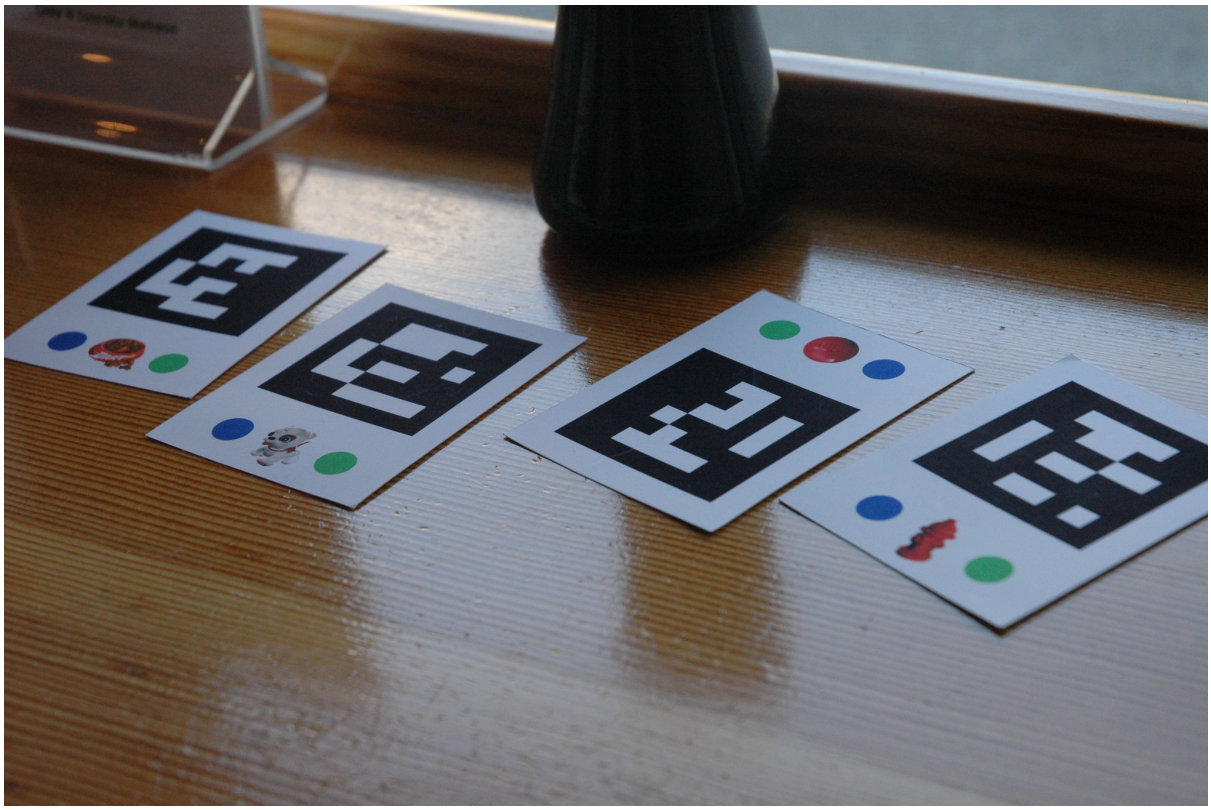

Figure 3: Augmented reality markers [12]

2 Background

As smartphones and mobile technology become more prevalent, new forms of human-computer interaction are becoming mainstream. Smartphones allow for an unprecedented degree of connectivity with the digital world, but can also serve as a tool for digitally enhancing the physical world. In this section we explain the origins and uses of this technology.

2.1 What is Augmented Reality?

Augmented reality, or AR, is a type of human-computer interface where perceptions of the real world are enhanced by computer-generated information. This differs from virtual reality (VR) in that a VR experience consists exclusively of virtual information in a virtual environment. In AR, virtual information is mixed with sensory input from the real world [13]. This can enhance the user's perception of reality by providing information that would be difficult or impossible to display through traditional means.

For example, AR can be used to display information about historical events, places, and objects overlaid onto images of the real world [5], in a way that is more intuitive or visually appealing than a purely virtual environment. This provides the user with useful information without needing to alter the physical historic site.

2.1.1 Current Augmented Reality Technology

While preparing for this project our team researched the current state of AR technology. Smartphones are the most commonly used AR hardware by far [14]. Typically smartphone AR applications make use of the phone's camera, accelerometer, gyroscope, and GPS sensors to reproduce a view of the real world with virtual information layered on top of it [15].

2.1.2 Augmented Reality Use Cases

Even though AR is still a developing technology, it has been used in many disparate disciplines, including data visualization [16], commerce [17], marketing [18], education [19], visual art [6], and even archaeology [20].

The video game industry has readily embraced augmented reality, leading to the development of many AR games for smartphones and dedicated head-mounted displays. Perhaps the most popular AR game, *Pokémon Go*, has been downloaded over a billion times [2]. This game makes use of GPS and camera data to overlay game objects (in this case fictional monsters) on top of images of the real world [21] (see Figure 4).

2.1.3 Challenges of Augmented Reality

Most existing applications using AR, in our team's opinions, do not leverage the full potential of this emerging technology. As with any new field, our understanding of AR and how to develop for it is limited. All sorts of different challenges and limitations face the development and production of AR apps. In this section we discuss some of the challenges we faced when working on our app.

Figure 4: A screenshot from the AR game *Pokémon Go* [22].

Many of the applications and use cases that exist for AR right now are, simply put, not very good. Many AR apps and features available on smartphones offer very little in the way of functionality or practicality; quite simply, they primarily revolve around deploying gimmicks [23]. For example, one of the most popular apps to use AR, *Pokémon Go*, primarily uses its AR feature to place a Pokémon on top of a camera's video feed. Disappointingly, the Pokémon does not interact with anything around it. It just floats a fixed distance away from the user.

Additionally, in many cases where AR is used, the task could be accomplished as well or better without AR [23]. Returning to the *Pokémon Go* example, when the AR feature is left on while capturing a Pokémon, the extra power needed for this feature drains the device's battery faster. Also, it makes it harder to hit the Pokémon with a Pokéball due to the model drifting in the environment. On top of all of this, the performance of the app suffers drastically, especially on older devices. All of these problems detract from the ability to play and enjoy the game.

One of the major problems facing AR development as a whole—including our project—is limited hardware. For starters, smartphone cameras present a major limitation. Many smartphone cameras

only capture images in 2D, which can make generating AR content in a 3D world difficult without the use of QR or barcode markers [24]. Additionally, GPS sensors on smartphones can also be too imprecise for good AR tracking [24].

AR apps are most commonly designed for smartphones due to their relatively low cost and how accessible they are. However, smartphones come with a host of hardware limitations that make achieving satisfying AR capabilities difficult. Additionally, AR apps for smartphones tend not to be very user-friendly, and oftentimes even complicate the task or activity they were meant to enhance [23]. The alternative then is to turn to dedicated hardware, like Microsoft's HoloLens. The issue is that dedicated hardware like the HoloLens is very inaccessible and expensive; Microsoft only offers developer editions of the HoloLens, and for the steep price of $3,500 [25].

The challenges we faced during development mirrored those that plague the industry as a whole very accurately. We struggled with how exactly we would use AR for our app in a way that benefited the experience as a whole. Despite our initial apprehension, we believe the way we have integrated AR into our app enhances the user experience. One of our largest challenges came from the lack of hardware on which to test. Only one member of our team had a phone new enough to run AR technology at a decent level, making testing of our application difficult.

2.2 izi.TRAVEL

Before our project had begun, Kyoto VR was already using an app to deliver audio tours. The app, called izi.TRAVEL, allows users to create audio tours with audio and image files tied to specific locations in the real world [26] (see Figure 5). The app is aimed at museums and city tours [27] but can be used for tours of historic sites too. Kyoto VR used the platform to make a simple audio tour of Kinkaku-ji.

Our group aimed to implement all the functionality of izi.TRAVEL and more in our own app. See Section 3.2 for more details.

2.3 Augmented Reality Platforms

Before we could start working on creating our app, we first needed to decide which AR development platform to use. Based on our preliminary research, we learned there are many powerful platforms available. There were many things we needed to consider before deciding on a platform—considerations such as what features the platform offered, which devices and operating systems the app could run on, and the cost. Detailed below are some of the different platforms and technologies we considered.

2.3.1 ARCore and ARKit

ARKit and ARCore are both frameworks that enable apps to take advantage of powerful AR related features. ARKit is developed by Apple and can only be used by iPhones [29]. Google's ARCore framework can deploy to both Android and iPhone. Many cross platform solutions will leverage the full potential of their target platform by using either ARKit or ARCore "under the hood." Both ViroReact and Vuforia use ARKit or ARCore depending on the target platform [30, 31].

Figure 5: A screenshot of the izi.TRAVEL app interface

ARKit 3 aims to support a wide array of new features and will be available in iOS 13 [32]. Notably, one of these features is "people occlusion," which will allow virtual objects to be realistically obscured when someone walks in front of the object. This kind of technology could be useful for us since our app uses AR in crowded places. As of August 2019, this feature is not on the horizon for ARCore.

Both ARKit and ARCore have well-supported APIs for Unity. ARKit 3 support is coming soon to Unity, along with all of the advanced features that it brings [33].

2.3.2 Wikitude

Wikitude is another major AR development kit that is very prevalent in the world of AR development. It has over 130,000 registered AR developers, and is powering over 30,000 AR apps across smartphones, tablets, and smart glasses [34]. It is easy to understand why Wikitude has so many registered developers considering the list of features the platform offers. Some of the features that were most appealing to us were their "Instant Tracking," allowing for augmentation using flat surfaces; "Image Tracking," allowing for augmentation using one or multiple 2D images; and "Geo AR," allowing for creation of geo-markers at specific locations to trigger AR content.

Wikitude supports Android, Apple, and Windows phones and tablets, as well as smart glasses like the Epson Moverio, Microsoft HoloLens, and Vuzix smart glasses. Additionally, many different development

Figure 6: The Golden Pavilion of Kinkaku-ji, Kyoto, Japan [28]

frameworks are capable of running Wikitude's software development kit (SDK). According to Wikitude's website, their SDK works with Windows, Android, and iOS development frameworks, as well as ARCore and ARKit, Flutter, Cordova, Xamarin, and Unity.

Based on this information, Wikitude was initially appealing to us. However, the cost was a major issue. Wikitude is free for eligible startups, but given the criteria they list on their site, we were not confident that we would qualify. Aside from this option, Wikitude offers a 30-day demo license for €499, which was far beyond the resources available to us. They also charge €1990 for their SDK Pro, and €2490 for their SDK Pro 3D. Given these prices, and our very limited budget, we decided to consider other options.

2.3.3 motive.io

The company motive.io previously was developing a product that would allow users to create an engaging AR experience in the style of *Pokémon Go* [35]. Their service would have purportedly handled technical details such as hosting, storage, and user accounts. The promotional video shows the creation of a location-based AR game where the user pretends to hack into ATMs using the real-world locations of ATMs in a city.

Unfortunately, it appears that the company has completely pivoted off of this idea, and has since moved to developing AR training software [36]. We reached out to motive.io via email for any advice they might be willing to give us, but we received no response.

2.3.4 ViroAR

Viro AR comes in two versions: ViroReact and ViroCore [37]. ViroCore allows developers to build an AR Application with Java. The disadvantage of this is that ViroCore only allows developers to target Android, which unfortunately made ViroCore not an option for our project; one of the requirements from our sponsor was that the app had to be cross-platform. ViroReact is the cross-platform option for ViroAR. ViroReact leverages the cross-platform capabilities of React, which is Facebook's JavaScript Library for developing user interfaces [38].

Since our project is not a game, we initially wanted to avoid using a fully-featured game engine such as Unity (see Section 2.3.5. Therefore, we decided to take the time to explore this framework since it appeared to be more tailored to our specific use case. Viro Media also recognizes that all developers who wish to create an app with 3D capabilities are not necessarily game developers. Viro Media pitches ViroAR as "The perfect alternative to specialized game engines, ViroAR is a platform for rapidly building ARKit and ARCore apps. Our platform allows developers to focus on what they do best by leveraging familiar tools and frameworks used in mobile application development" [37]. While the majority of our team has decent experience with writing JavaScript, we have have very little experience writing React apps.

In most app development scenarios, to test the app on actual hardware, it must be compiled using either Xcode or Android Studio depending on the target platform. Once you have built the app, it must be installed on the device. This process can be very time consuming, especially when needing to run frequent tests on the target platform.

An alternative to building an app for a physical phone is to run it through an emulator on a computer. According to Techopedia, "Emulation is the process of imitating a hardware/software program/platform on another program or platform. This makes it possible to run programs on systems not designed for them" [39]. Emulators are often very useful for speeding up development. Loading programs onto the Android emulator is usually much faster than installing an Android package file (APK) onto the actual device. Additionally, emulators can mock almost all of the important features of an actual Android device, including calls, text, GPS position, device rotation and more [40].

Since an emulator uses software to emulate hardware, this poses two technical issues that prevent emulators from being a proper replacement for testing on real hardware. Emulation is often imperfect, so glitches sometimes emerge in emulation that do not manifest on the actual hardware [41]. Conversely, errors that exist in the hardware version may not appear in the emulation. Secondly, and most importantly for AR, it is common for emulation to be much slower compared to real hardware.[1] Considering that AR already pushes modern phone hardware to the limit, emulation is not something we wanted to struggle with. On top of this, our testing required us to move the camera through the world at various

[1] In a question submitted to *Compute!* magazine asking whether it is possible for a Commodore 64 to emulate MS-DOS, the editor responded "Yes, it's possible for a 64 to emulate an IBM PC, in the same sense that it's possible to bail out Lake Michigan with a teaspoon." The editor goes on to explain "Emulation is a complex business, but here's one rule of thumb: The only way to successfully emulate a machine is with a much more powerful machine" [42]. In our case, emulating Android on a desktop operating system is closer to emulating a C64 on an IBM PC running MS-DOS (rather than the other way around, thankfully). The takeaway is that the speed problem facing emulators is just as real as it was in 1988.

speeds and angles, which is simply cumbersome with a laptop webcam.

As an alternative to emulation or compiling a native binary, ViroReact offers a convenient "test bed" for rapidly testing ViroReact apps on native hardware without the need for an emulator. This allowed us to launch a working AR app on our devices much more easily than any of the other frameworks or engines we explored. Unfortunately, the test bed app was somewhat unreliable based on our experience; the test bed would frequently crash or would not be able to download the files from the server. The most reliable way to test our app was to compile a binary and manually install it onto our devices, which completely defeated the purpose of the test bed app.

2.3.5 Unity

Unity is a game engine that was originally introduced in 2005 [43]. Because of the low barrier to entry—experience-wise and price-wise—the engine quickly became incredibly popular. Ironically, because the engine is so accessible, Unity has faced a degree of critism. Because Unity appeals to so many beginners, there are many unpolished games that bear the Unity logo. In response to this sentiment, Marcos Sanchez, head of global communications at Unity, said "Democratizing development—in our minds, we don't think is the wrong thing, we think it's the right thing. You want more people understanding, and everyone's gotta start somewhere" [43]. Unity CEO John Riccitiello made the claim at TechCrunch Disrupt San Francisco that half of all games are made with Unity [44]. We consider Unity's ubiquity to be an advantage; there is a glut of information about how to use augmented reality within Unity.

Even though Unity is beginner-friendly for making games, the augmented reality app we planned to create was not a game. Therefore, we were somewhat concerned that Unity's ease-of-use would be undermined simply because we were not making a traditional game experience. Instead we would be using Unity to create a piece of software that both looked and felt like a familiar mobile app. Unity has very limited support for this compared to technologies that are specifically geared toward creating modern mobile user interfaces such as React.

All of Unity's scripting is done with C#, which is a general purpose, object-oriented language [45]. Even though none of us had extensive experience with C# prior to this project, this was not of great concern since C# is similar to other object-oriented languages such as Java and C++. The design goals of the language were also appealing to us; C# includes features important for programmer productivity such as strong type checking (unlike JavaScript), detection of using uninitialized variables (also unlike normal JavaScript,) range checking for array bounds (unlike C) and garbage collection. At the same time, our lack of familiarity with the language and Unity as a whole is another reason we first looked into cross-platform AR solutions that did not use Unity.

At the same time, Unity has excellent support for augmented reality through various extensions. Because we initially understood AR to be the core of the app experience, we chose to use Unity with the Vuforia extension for all future development on the mobile app.

Unity is distinct from the technologies previously discussed in this section because it is not an AR platform unto itself. Instead, Unity can take advantage of AR platforms such as ARKit, ARCore, Vuforia and Wikitude through the use of plugins.

2.4 Kinkaku-ji

Kinkaku-ji, the Temple of the Golden Pavilion, is undoubtedly one of Japan's greatest attractions. As of September 2019, it was the second-most visited site in Kyoto—second only to Kyoto Station [10]. Kinkaku-ji's rich history and stunning beauty bring tourists from all over the world to see its shining gold walls and magnificent gardens. Our project intends to enhance users' enjoyment and understanding of Kinkaku-ji through informative interactive media, teaching about the history and significance of the site.

The site that is now Kinkaku-ji was originally a villa owned by poet and nobleman Saionji Kintsune, until the land was bought by shōgun Ashikaga Yoshimitsu in 1397 [46]. As Zen Buddhism was in vogue among the warrior class in Japan at the time, the site was converted into a Zen temple according to Yoshimitsu's wishes upon his death [7].

The main pavilion is a three-story tall building on the edge of a landscaped pond. The outer walls of the top two floors are completely covered in gold leaf, leading to the name "Golden Pavilion" (金閣 *kinkaku*). The grounds surrounding the Golden Pavilion are in the style of a Japanese strolling garden, with carefully manicured plants and paths and designated viewing points. The buildings and landscaping make use of a Zen practice called "borrowed scenery" (借景 *shakkei*), in which the background landscape is incorporated into the design of a garden [46].

The pavilion sits on the edge of a carefully designed pond (see Figure 7), with small islands designed to replicate the shape of Japan and several other famous locations from Japanese and Chinese literature [47]. All of these landscape features have been immaculately maintained and preserved through the Golden Pavilion's six hundred year history.

Figure 7: Expertly landscaped islands in the Kinkaku-ji mirror pond [48]

In 1950 the pavilion was burned down by a paranoid schizophrenic monk who had been living at the

temple [49]. The structure was rebuilt in 1955, close to the original design but with more extensive gold leaf [7]. The interior was also restored later to match the original.

None of this history is readily apparent to a typical non-Japanese speaking tourist at Kinkaku-ji. There are a few signs describing different areas, but they are only available in Japanese and contain very little information about the history and design of the site. Because of this, an intuitive audio tour app with AR features would be very helpful to tourists at Kinkaku-ji.

3 Implementation and Technology

In this section we discuss the implementation of ARuko and Editour, which are the names for the mobile AR mobile app and web-based tour editor respectively. We also discuss the technologies and techniques involved in creating both of them, along with the the design decisions we made and challenges we faced in creating these applications.

3.1 Editour: The Tour Editor

The goal of Editour is to allow a non-technical user to design audio tours through a web app. At its core, Editour is a tool to draw arbitrarily-shaped geographic regions onto a map, edit those regions and assign media to each of the drawn regions.

An audio file and multiple image files can be uploaded for each region the user defines. The user can also provide a text transcript of the audio file as well. After naming and uploading the tour with the "Upload" button, our backend builds a zip containing files that can be placed directly into the Unity project.

Our backend allows users to save their tours to a server and load those same tours back into the editor to resume work. The geographic area of each region can be adjusted by clicking and dragging. Vertices can also be added and deleted if the needs of the tour change over time.

3.1.1 The Need for an Editour

One of the goals for our project is to create a working prototype that others can build off of in the future. Similar to the app izi.TRAVEL discussed in Section 2.2, we needed some way to easily define arbitrary polygons (which we call "regions") and associate media with each one, such as audio and image files.

For testing purposes, we initially defined these regions directly in the C# scripts, hard-coding the coordinates. Below is an example of a quadrilateral surrounding the Creation Core building at the Ritsumeikan Biwako-Kusatsu campus.

```
Regions.add(new GPSPolygon(new List<GPSPoint>{
    new GPSPoint(34.979222, 135.963628),
    new GPSPoint(34.979187, 135.965130),
    new GPSPoint(34.979794, 135.965053),
    new GPSPoint(34.979754, 135.963669)
}, "Creation Core"));
```

Even with knowledge of C# and Unity, inputting tour data this way is obviously cumbersome. Atticus is not a software developer, and does not want to work with C# scripts. In order to make our app prototype at all useful for the future, we needed to provide an easy way to "design" a tour using a simple graphical interface.

3.1.2 A Complete Tour Definition

In order to create a fully-functional editor for designing and exporting "tours," we had to consider how to fully define a tour purely in terms of text and a directory of associated media.

Here is an example of a tour metadata file in JavaScript Object Notation (JSON) format:

```
{
   "regions": [
      {
         "name": "Beautiful Place",
         "points": [ {"lat": 34.979222, "lng": 135.963628}, ... ],
         "audio": [ "beautiful-audio.mp3" ],
         "images": [ "mountain.jpg", "stream.jpg" ],
         "transcript": "To your left, you can see a beautiful place."
      },
      {
         "name": "Gorgeous Place",
         "points": [ {"lat": 34.979754, "lng": 135.964889}, ... ],
         "audio": [ "gorgeous-audio.mp3" ],
         "images": [ "hill.jpg", "valley.jpg", "gorge.jpg" ],
         "transcript": "To your right, you can see a gorgeous place."
      }
   ]
}
```

Within each object in the region list, we can see that the region has five properties: a name, a list of coordinates, a list of audio files,[2] a list of multiple image files, and a transcript of the audio file. One disadvantage is that all media files have to have unique names. This is because the zip that is created by the Editour backend contains all of the media files in a flat file structure. The above JSON is used to associate each media file with a region, meaning that the filename needs to act as a unique identifier. The backend will respond with a message if it finds duplicate filenames, prompting the user to use unique names for all files.

3.1.3 Editing a Tour

The user can easily create a new region by shift-clicking the map, and then continuing to click to define more points. Figure 9 shows this in action. To terminate the polygon, the user must shift-click again. The "Welcome to Editour" card (seen in Figure 8) provides these instructions, which should be enough to get started. A user can only terminate the polygon after two points have been placed to prevent the user from drawing a region with an area of zero. Once a region is created, a region card appears in the side bar. From here, the user can edit anything about the region.

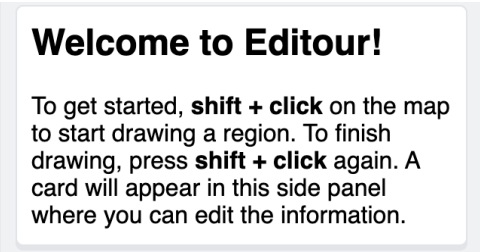

Figure 8: The "Welcome to Editour" card providing basic instructions to get started

The top of the region card displays the region name in bold text. By clicking on the region name, the map will pan and zoom to frame the region in the center of the screen. This is useful if there are many regions. To the right of the region name, there are two arrow buttons, one pointed up and another

[2] In the current design of the app, it only makes sense to associate one audio file per region, but the format makes it easy to accommodate for multiple audio files in the future.

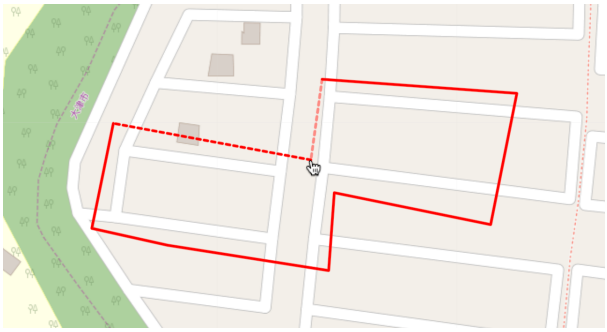

Figure 9: Editour when a region is being drawn, including the two dotted preview lines

pointed down. This allows the user to reorder the regions in the column. This was one of the features that was added late into development in order to adapt to an important change in the tour app, ARuko. This is discussed in Section 3.2.6.

In the region card, there are four multicolored expandable "subcards" that allow the user to edit different information about each region. Since there can be many region cards in the column, it was important to be able to collapse each section. Collapsing the subcards gives the user enough room to see multiple region cards at a time.

The first subcard seen in Figure 10 is the "Rename" subcard. This section expands to reveal a textbox, allowing the user to rename the region after clicking "Okay" or hitting the enter key.[3]

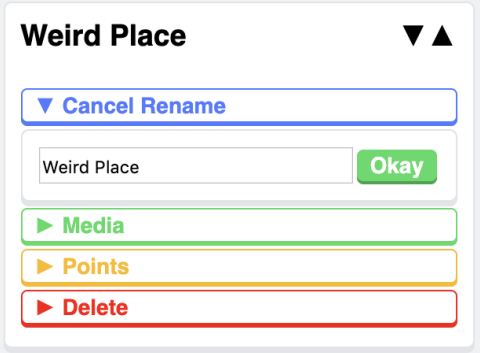

Figure 10: The "Rename" subcard

The second subcard seen in Figure 11 is more involved. Expanding the "Media" subcard by clicking on the green bar will reveal a bevy of options for augmenting the audio, images, and text to be associated with that region of the tour. From here, the user can upload an audio file and multiple image files from the user's hard drive with the two file-select fields. If the user is editing an existing tour that has already been uploaded, the names of the files already present on the server will appear as cards with 'X' buttons. This allows the user to remove images or audio from that region even after the tour has been uploaded.

The third subcard is the "Points" subcard. This subcard can be revealed in one of two ways. The first way is by clicking on the yellow bar labeled "Points", which is the same for all subcards. Alternatively, the user can click directly on the region on the map. The region will flash yellow and be scrolled into

[3]There are multiple places in the UI where a textbox is immediately adjacent to a "Confirm" button. We made sure all of these can be triggered by simply hitting enter, since this is common behavior when filling out forms on the web.

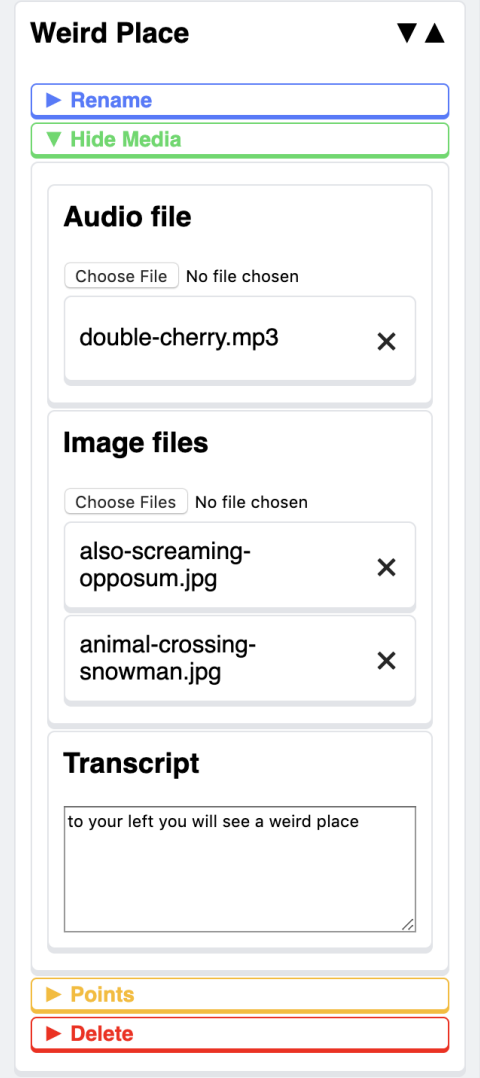

Figure 11: The "Media" subcard

view to let the user know which region was clicked on. Both of these methods will enable editing mode, whereby the user can directly manipulate the vertices that make up the region. Blue circles appear on top of each vertex, and red circles appear at the midpoint of each edge. Clicking on a red midpoint will add an additional vertex at that midpoint. Dragging one of the blue vertices will move the vertex around. The latitude and longitude are updated in the "Points" subcard as the user moves the vertex. Clicking on the vertex directly will create a blue pin-shaped marker that the user can also drag. Clicking on the coordinate box in the subcard will also create this blue pin-shaped marker; the map will also pan to this coordinate. The user can also delete vertices in the same way that files can be deleted from the "Media" subcard by clicking on the 'X' button on that vertex's coordinate card. If there are exactly three points in a region, the 'X' buttons will be "grayed-out," preventing the user from deleting any more points, since a polygon must consist of at least three points.

The third subcard pictured in Figure 13 is the "Delete" subcard. When revealed, another button will appear labeled "Really Delete." The action can be canceled by collapsing the subcard by hitting the bar now labeled "Don't Delete!" This prevents the user from accidentally deleting a region unintentionally.

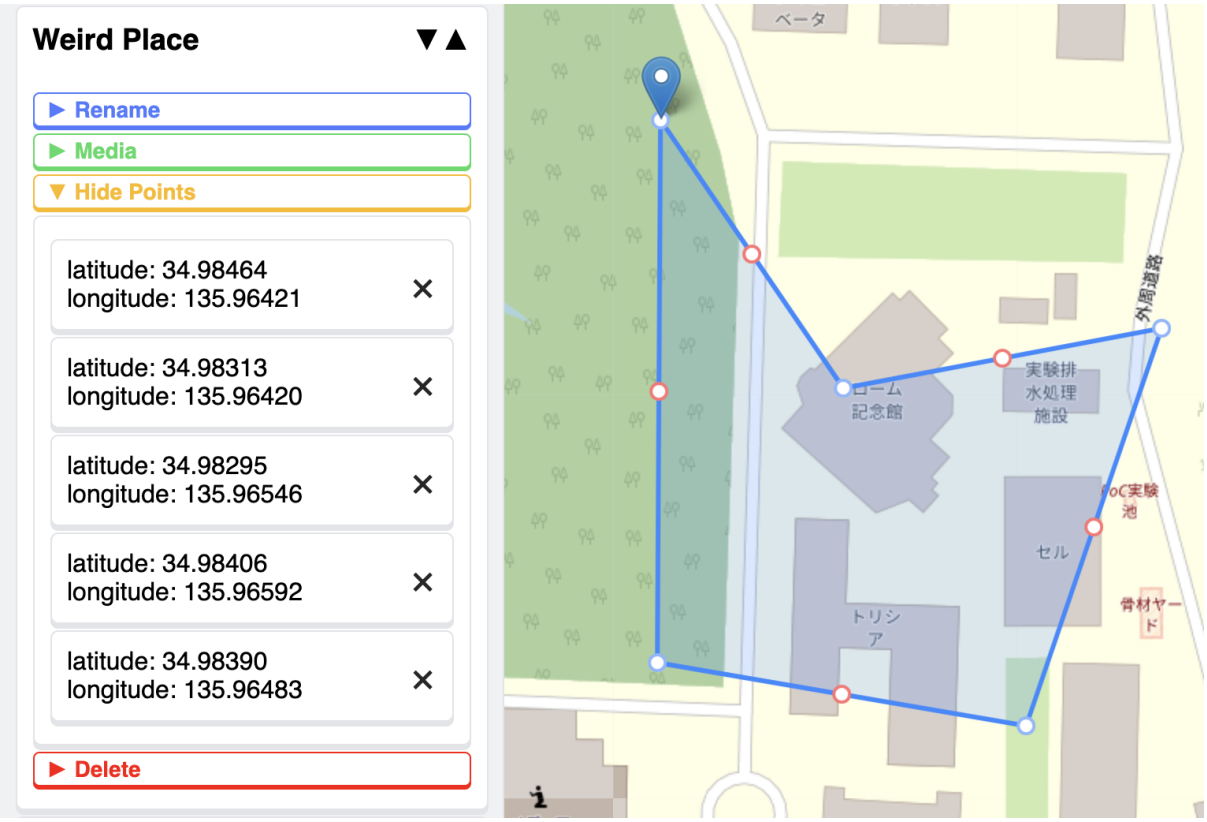

Figure 12: The "Points" subcard with the map showing vertices and edges that can be edited

Above all of the region cards, there are various options for uploading, downloading, renaming and deleting tour files on the server. Within the upload, download and delete cards there are message boxes to indicate the status of that operation. When uploading and downloading, a percentage will be displayed indicating the upload/download process, which can be seen in Figure 14. If an HTTP 200 (OK) or 201 (Created) code [50] is returned from the server, a "Success" message will appear. If there was some problem, the error message returned by the server is displayed in this box instead. Since uploading, downloading, and deleting are all network-related operations, it makes sense to include this message box for each card.

When the page is loaded, the frontend will request the names of the tour files stored on the server from the backend. This data is used to create a box of buttons, which includes one button for each tour file on the server. Clicking on one of these blue buttons will fill the textbox in with the correct name. Before this feature was implemented, the user had to correctly type the name of the saved tour into the textbox in order to retrieve it. The user is still free to do this, but the buttons make this easier and less error-prone. This feature can be seen in figure Figure 15. Clicking "Download" does not download the tour to the user's computer. Instead, it requests only the tour metadata from the server so the tour's data can be displayed in the web app. The user can then edit the tour and upload a new version.

We did not include a way to resolve addresses or place names, like the service provided by Google Maps. Instead of exploring plugins/libraries that might provide this functionality, we offered a simpler solution due to time constraints. You can type in a latitude and longitude directly and hit the "Jump"

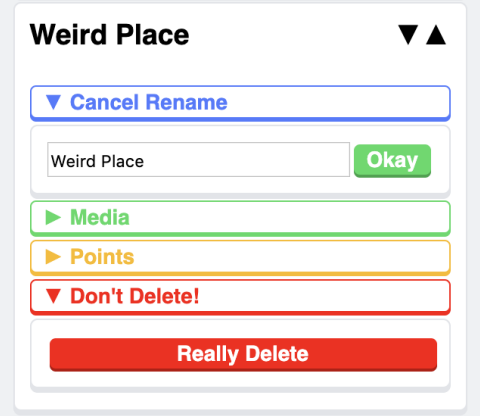

Figure 13: The "Delete" subcard

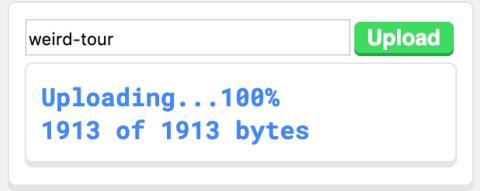

Figure 14: The upload subcard with an upload status

button to pan to the specified location. We also hard-coded a few shortcuts for places that Atticus might want to design a tour for in the future, which can be accessed by clicking on the yellow buttons with place names.

Once a tour is either uploaded or downloaded, a link to download the zip file of that tour will appear. The zip file contains all of the uploaded media, and a metadata file. The metadata is in the form of a JSON file; it contains all of the latitudes, longitudes, and enough information to associate all of the media with the correct regions. The downloaded zip file is what we decompress and place into the Unity project before it is compiled. This feature is also useful if the user wants to check the audio and image files that were uploaded.

3.1.4 Editour Frontend Implementation

The frontend was implemented as a web application, using web technologies such as HTML, CSS and JavaScript, with JavaScript being the vast majority of the code. Type-checking provided by Visual Studio Code and JSDoc made the growing code complexity more manageable.[4] Excluding Node.js modules, the only external library we used for Editour was Leaflet, which is "an open-source JavaScript library for mobile-friendly interactive maps" according to the official Leaflet website [52]. We did not use any framework for developing frontends like Facebook's React or Google's Angular. Instead, all of the complex UI interactions were handled by modifying the DOM directly with JavaScript's built-in

[4] One helpful specification of the ECMAScript standard in the way of programmer sanity is "strict mode." By placing the string `"use strict"` at the top of a JavaScript source file, errors will be thrown in many cases where ordinary JavaScript would not complain. The simplest example of this is the line `a = 1;` will not throw an error ordinarily. In strict mode, you must at first explicitly declare the variable with the `let`, `var`, or `const` keyword before assigning it, similar to Java or C++. We included the string at the beginning of every source file except for class declarations, which are in strict mode by default [51, p. 386].

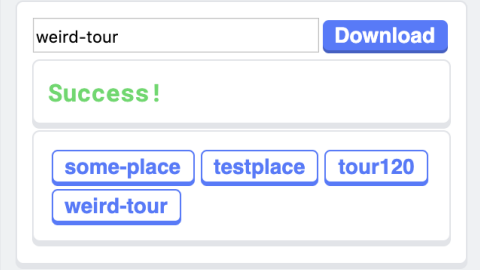

Figure 15: The download card

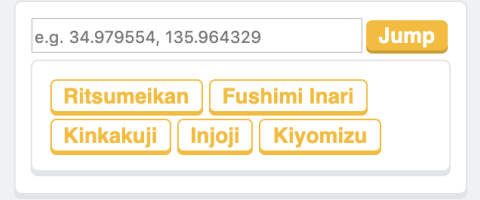

Figure 16: The jump card

functionality. Furthermore, we did not use any advanced frontend build tools such as Rollup, webpack or Parcel. For a JavaScript project of this scope, this was the correct decision. Considering we had limited knowledge of React or Angular, it was also faster to develop the UI in the way we were used to from previous experience, but if we were to continue to develop Editour, UI libraries and frontend build tools would be something to consider.

3.1.5 Editour Backend

The server-side backend of the Editour application is written in JavaScript for Node.js. This backend both serves the static frontend content and runs server-side scripts that process incoming application programming interface (API) requests. We chose Node.js as the server-side framework because of our prior experience with JavaScript, its high-performing asynchronous architecture [53], and its ease of development.

When a user submits new tour from the web application, the files and metadata are sent to the server running the Node.js scripts. The server zips up the files and saves them to the disk with a timestamp. Then any application can request a particular tour and the backend will serve the most recent version.

As previously mentioned in Section 3.1.3, our web application allows users to edit existing tours. When the user requests to edit a tour on the frontend it sends a request to the backend server for that tour's metadata file. This file contains information about all regions and files in the tour. The user can then edit the regions, change names, upload new files, or delete existing ones without ever having to download the other tour files from the server, which could take a long time depending on their size. When the user is done editing they can upload the new metadata and any newly added files to the server, which intelligently collects new and old files required by the tour and zips them into a new tour file on the disk.

The backend implements a representational state transfer (REST) API. In a "RESTful" service, web

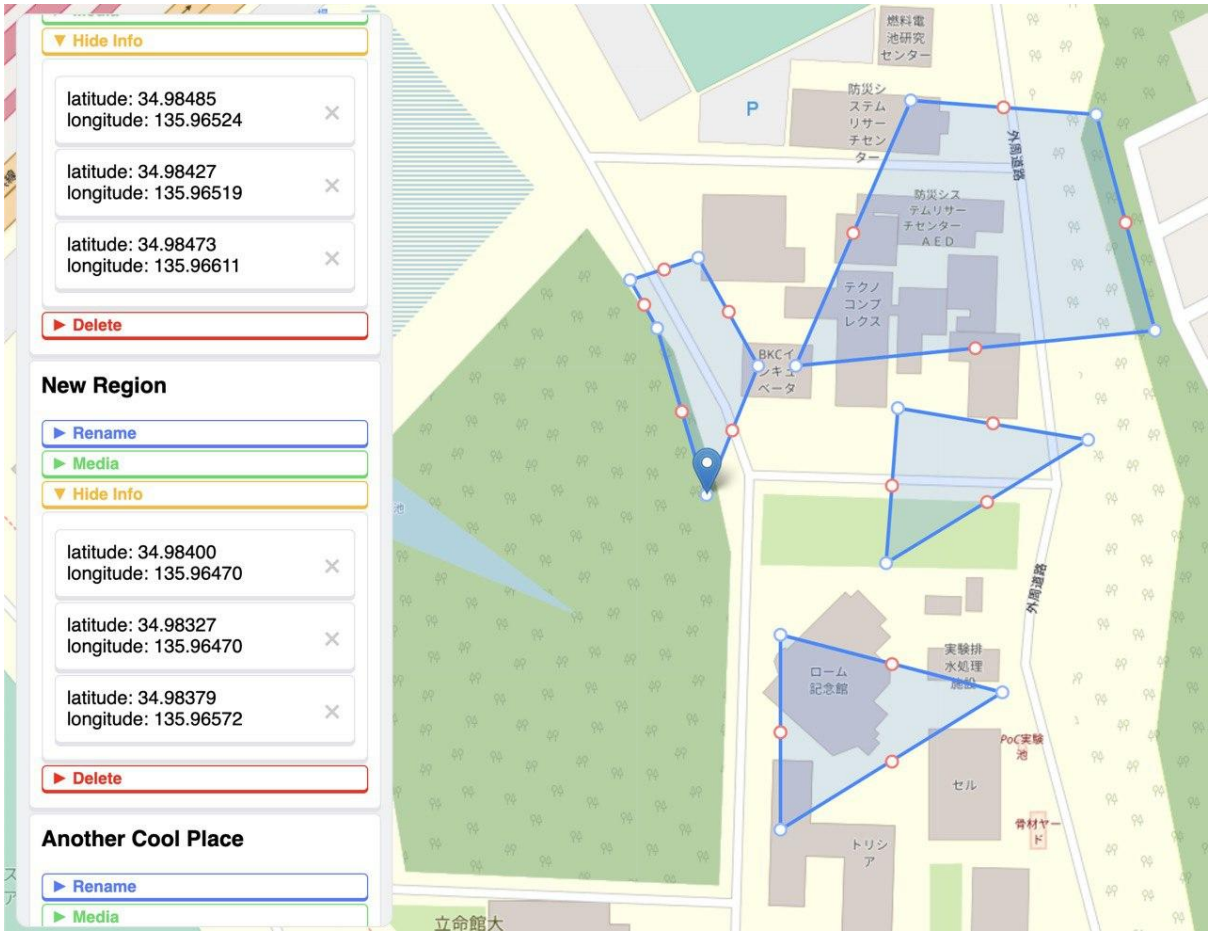

Figure 17: A screenshot of the Editour web app frontend

resources are manipulated with a predefined set of stateless operations [50]. Because the REST API is operated over the Hypertext Transfer Protocol (HTTP), the available operations are just the standard HTTP methods, such as GET, POST, and DELETE. For full documentation of the backend's API, see Appendix B.

3.1.6 Deploying the Editour Backend

For testing and development purposes the Editour backend was originally hosted on Joseph Petitti's personal server. This was a CentOS virtual machine running on an old Dell rack server in Worcester, Massachusetts. We wrote a custom systemd service to run Editour as a daemon and keep its log files organized. The Node.js backend was served by an Nginx reverse proxy running on the same virtual machine.

While this worked well enough for testing, it was not a permanent solution. We asked Atticus to provide a more powerful—and geographically closer—hosting solution for running Editour in the future. He obtained a domain name and a shared hosting environment on a Red Hat Enterprise Linux (RHEL) server for us to use on September 23$^{\text{rd}}$. Unfortunately, this shared hosting environment was designed for running simple PHP applications like WordPress, so it took a bit of work to coerce it into running a Node.js application.

We were able to get shell access to the shared hosting environment, but without superuser privileges. This meant we could not install packages, change system settings, or access `systemctl`, the command that manages systemd services. In order to install Node.js and npm in this environment we had to use Node Version Manager, a series of shell scripts that handles downloading and compiling various versions of Node [54]. The only difficulty with this was finding a version of Node.js that could compile with the older version of GCC that the RHEL server had. After resolving this challenge, we were able to simply clone the backend from GitHub and run it with npm.

However, the server already had Apache listening on port 80, and without superuser privileges we could not edit the Apache configuration files to change this. Instead we decided to configure Apache as a reverse proxy, using only the rules that can be put in a `.htaccess` file,[5] since this is the only configuration file we could access. Using Apache's `mod_rewrite` module we were able to rewrite incoming requests on port 80 to instead go to `localhost:3000`, the port where Node.js was listening [55].

With no access to `systemctl` we could not use the custom systemd service we wrote to manage the server daemon. Instead, we used a custom shell script and an npm package called "forever" to handle starting and stopping it. The forever package runs Node.js scripts continuously, automatically restarting them if they crash [56].

Tours are stored on the server as compressed zip files, but when uploading or editing a tour these zips are decompressed to a temporary directory. To prevent the server's limited disk space from filling up with old temporary files we wrote a simple bash script that removes directories that have not been modified in the past five days. We set up a cron job[6] to run this script automatically once per day to clean out old temporary files that are not being used any more.

Unfortunately, the hosting solution Kyoto VR provided us with severely throttles disk write speeds. Because the backend has to write potentially large amounts of data to the disk every time a tour is uploaded or edited this becomes a major bottleneck for the web app's performance. For large tours (bigger than 50 MiB) this can cause an issue where the Node server takes so long to respond that Apache thinks it has timed out, and sends an erroneous 504 Gateway Timeout message [50]. Without access to superuser permissions or a faster hosting provider there is nothing we can do about this bug. The Node server does eventually complete the request and respond correctly, so until Kyoto VR can get a better hosting provider we consider it a minor issue.

3.1.7 Future of Editour

In the current implementation, the tour file generated by the Editour backend still has to be moved manually to the Unity project folder. In the future, we would like the app to be able to contact a server running a version of the Editour backend and update the tour info by itself. This would allow Kyoto VR to update the tours without updating the entire app.

Furthermore, we foresee Editour being a useful tool outside of the app we designed for Kyoto VR. The design of Editour is agnostic about how the media will be used. For an app like izi.TRAVEL, the

[5] This is a directory-level configuration file with limited options compared to Apache's global configuration files.
[6] The program cron allows for periodic scheduling of repeated commands

images are displayed when the associated audio tour is playing.

The files produced by Editour could be used to power an app that does not incorporate AR, such as an audio guide mobile web app that runs entirely in the browser, using the geolocation API to access the user's location (if the user grants permission). This kind of infrastructure would allow users to access an audio tour—supplemented by images and audio transcripts—without the need to download an app at all. By simply visiting a link, a user could experience a guided tour that changes as the user walks through the route. For example, colleges could distribute tour links via QR codes, which would instantly launch an interactive guided tour of the campus through the browser.

Currently, anyone can clone the GitHub repository of Editour and deploy a working version on a personal server. The source code for Editour is under a free license, namely the GNU General Public License. This allows anyone to expand the frontend and backend to suit their needs. With additional hardware and by extending the current implementation of both the frontend and backend, Editour could be used as the groundwork to create an entire crowd-sourced walking tour platform, allowing users to design tours for others to take.

In summary, we want to explore the idea of creating a community-driven audio tour database that allows mobile users to take tours without the need to install an app. The free code we wrote for Editour empowers us (or anyone else) to develop and launch this platform in the future.

3.2 ARuko

ARuko, named after the Japanese word "歩こ" (*aruko*), meaning "let's walk," is the app we created to guide users through an AR-enhanced tour. The app was built with Unity, which enabled to us to use a single codebase for the iOS and Android versions of the app.

3.2.1 GPS Functionality

The core user experience of ARuko is the walking tour. By walking the given route, the user enters and exits specific geographic regions that invoke changes in the app, such as playing audio and displaying different images. Originally, we planned to mark regions by singular points, and trigger that region if the user was within a certain radius of that point. Implementation-wise, this would have been much simpler, but obviously limits possible tour designs. Consequently, we decided at an early stage to accommodate regions with arbitrary shape and size. In the implementation, we provided support for both "GPS Bubbles" and "GPS Polygons." In the end however, Editour made it trivial to use the more versatile "GPS Polygons," so we eventually dropped support for regions triggered by distance.

3.2.2 Translation Feature

ARuko uses augmented reality to translate text at the site of the tour as well. While AR translation apps exist, the experience can be frustrating, and produce poor results. Google Translate's text recognition and machine-learning powered translation technology is certainly impressive. At the same time, the translation service works better in "scan" mode instead of "instant" mode. In "instant" mode, the app

attempts to translate text as quickly as possible, superimposing the translated text onto the real world using the camera. In "scan" mode, the user first takes a photo. Then, the user is prompted to highlight where text appears in the photo. After the translation is complete, the translated text appears in a separate UI element. In most cases, it is more convenient and readable to have the translation appear in a separate scrollable text box. This motivated the current design of the "info buttons" in ARuko. Instead of placing text directly onto signage, we place a blue button with a stylized "i" in the center. By tapping on this button, a separate UI element will appear containing the translated text. Since this button is cast into the world using the AR camera, the button appears to be placed directly on top of the content.

3.2.3 Map Overlay Feature

Near the entrance of Kinkaku-ji before the ticket gate, there is a large illustrated map. Much like the signage, there is no language on the map other than Japanese. Using the AR camera within the app, information can be projected onto the map. Figure 18 shows the blue highlighted path of the walking tour, including a few translated labels. The blue path and English text are not part of the physical map. These are superimposed on the image by ARuko.

3.2.4 Virtual Souvenir Feature

Tickets to enter Kinkaku-ji are long sheets of paper printed with calligraphy. The tickets are large and visually appealing; they are souvenirs unto themselves that one might feel inclined to keep around. Over the course of our three month stay we visited Kinkaku-ji thrice. For each visit, we received identical tickets. Seemingly, the design on the tickets do not change on a regular basis, if at all. Figure 19 shows a ticket we kept from one of our visits to Kinkaku-ji.

Because the tickets are unlikely to change frequently, we are able to provide the user with a "virtual souvenir" using the AR camera. This is the only AR feature that is not tied directly to an image target that can only be accessed at Kinkaku-ji. While the image gallery and segments of the audio tour can be accessed at any time using the "Chapter Select" menu, most of the AR features can only be experienced during the walking tour, pointing the camera at objects physically located at the site. Augmenting the ticket with 3D content allows visitors to take something home, giving them a reason to launch the app later.

We found a free untextured model of the Golden Pavilion to use for this feature on Thingiverse, a site for sharing user-created 3D models [57]. Using Unity's built-in capability to design materials, we created a shiny golden material and applied it to the model (see Figure 20).

Originally, the AR model of the Golden Pavilion was flush against the plane of the ticket, sitting on the paper as if it were ground. Atticus suggested that we re-orient the model such that the ticket served as a back plane. This would allow users to hold the ticket vertically, and view the Golden Pavilion floating in front of the ticket. This allows users to place their hands under the base of the Golden Pavilion model as if they were holding it; Vuforia's image tracking is flexible enough such that the image target can be partially obscured by the hand.

Figure 18: Map overlay displaying the path and translated labels

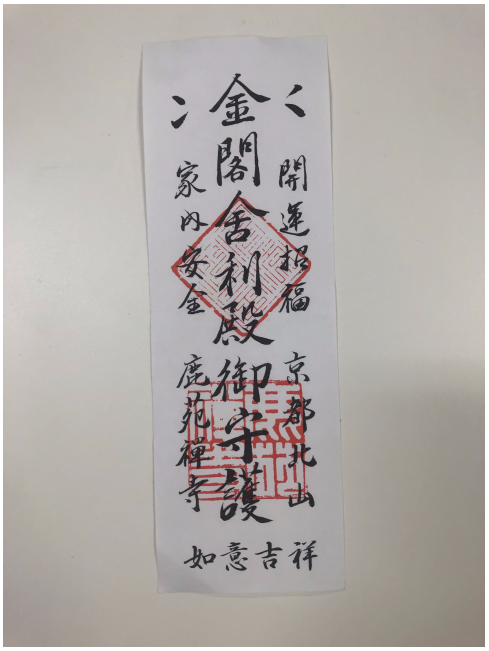

Figure 19: Kinkaku-ji ticket

3.2.5 Early User Interface

As ideas changed and ARuko became what it is today, the user interface (UI) also changed greatly from its early conception to the final version. Still, the early UI was a valuable learning experience, providing us with skills that were useful when building the finished product.

The early UI was purely functional with no bells or whistles; many of its elements were created on the fly as new features were added. As the app grew, it became clear that the UI could not consist simply of buttons and other basic features provided by Unity.

Pictured in Figure 21 is our early UI. It was very minimal, but still covered all of the uses for our app at the time. The top of the screen had two drop-down menus. The one on the left (which is shown open in Figure 21) shows AR image targets users could look for and scan. The right drop-down would be dynamically populated in each region with buttons bearing different images. These images were ones that Atticus wanted to be viewable in AR using ground plane detection. When tapped, each button would create a cube, with the given image on each face, and place it on the ground where the user was pointing. Figure 22 shows this feature in action.

Atticus initially wanted us to explore the efficacy of using Vuforia's 2D image target capabilities for scanning objects outside of the supported use case, such as 3D landscapes or the side of a building. We had serious doubts about how well this would work, and our suspicions were confirmed after some field testing. This was the reasoning behind focusing on ground-plane detection instead of image targets; ground-plane detection can be used anywhere, and is much more reliable than treating 3D objects as 2D image targets.

Learning to create these drop-down menus proved to be a very important lesson later on, since we would put that exact knowledge to use multiple times in the final UI. We were able to create the dynamically populated drop-down menus by creating an empty vertical layout group. Then, a script

Figure 20: ARuko's virtual souvenir feature

would create buttons for each image to display, and assign those images as children of the layout group. To get the smooth open/close animation, we "lerp" (linearly interpolate) the menu to the corresponding position each frame until it is fully opened or closed. In between these two drop-down menus is a panel that would display the name for the region the user is in at the time. The empty space in the middle of the screen is where the AR camera display would be shown. In this implementation, the AR functionality was the focus of the app, so it made sense to display the AR camera feed at all times. Lastly, on the bottom is a slider that controls the playback position of the audio file played in each region. We learned about how audio sliders work in Unity with this feature, and would expand on that knowledge for the final version of the app in order to optimize the slider's functionality.

Although most elements of the initial UI did not survive into the final product, this original phase was still quite valuable as a learning tool. While crafting this prototype we discovered many of the intricacies of UI design in Unity. For example, all UI elements must be attached to a Unity canvas element, which covers the entire phone screen. Elements should be anchored to edges or corners of this canvas so that they stay in a constant location on screens of different sizes.

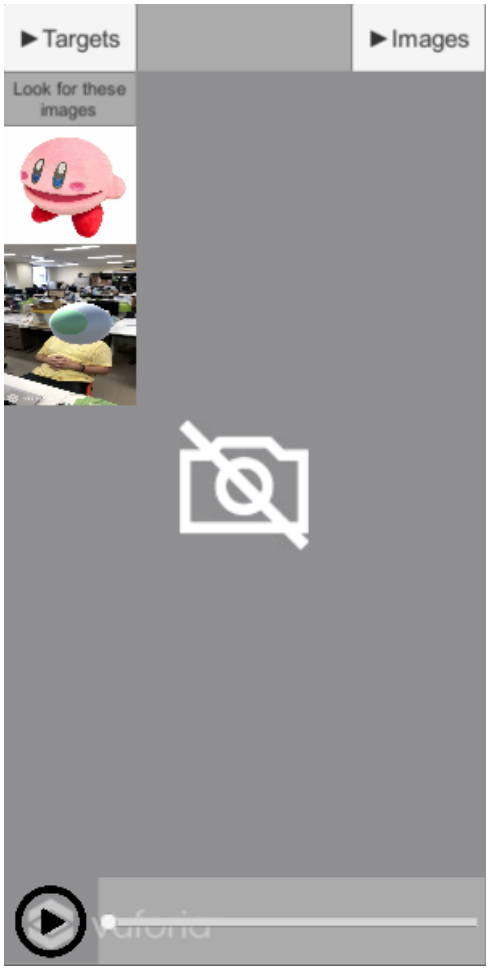

Figure 21: A screenshot of our early UI

3.2.6 Adobe XD Design

By the end of the project, we were able to turn ARuko into an app rich with features and polish. However, many of these features were requested relatively late into the project. This section discusses some of our initial concerns when we were first given the task of bringing Atticus's ideas to life.

On September 10th Atticus Sims presented us with a storyboard for the app experience. We would not be able to resume work until September 17th due to prior travel plans. The design he put forth had ramifications that rippled throughout the entire software stack we had constructed up to this point. Figure 23 shows the experience design document created in Adobe XD.

One of the notable changes is the chapter select screen, which is the third screen in Figure 23. Before, we worked under the assumption that each geolocation-triggered region would not have to be ordered in any way. This assumption made its way into Editour, which at the time did not have a way to reorder regions once they were placed. Because of this chapter selection screen, we needed some way to specify and rearrange the order of regions within Editour.

The inclusion of a "chapter selection" screen had other notable design implications as well. Initially different regions were triggered solely by the GPS location. Walking into a region would cause the available images to change on screen. Also, a new audio track would begin to play. This forced us to

Figure 22: An "art cube" floating over the water containing a placeholder programming meme

answer certain basic questions about the chapter select screen for ourselves. For example, consider a user that steps into one region, triggering the audio and images. Then, that user selects a chapter from the chapter selection menu, overriding the "true" region the user is physically standing inside of. When does the GPS "take over" again? Should the region change back to the "true" region once the audio of the selected region finishes? This might be intuitive and preferred for most users, but what if the user is more interested in the image gallery than the audio? Then, the persistence of the image gallery is strangely tied to the duration of the audio track playing underneath. What if a user selects a chapter, and then physically enters that region moments later? Should the audio restart to indicate that the user has arrived at the location, or will this appear as though the audio track skipped to the beginning again for no reason? This seemingly minor inclusion created a significant amount of discussion.

Other concerns about the chapter selection feature were technical. For a long time, we were working under the assumption that regions would only be triggered by GPS location. Because of this long-standing assumption, we were uncertain about what bugs implementing a chapter selection feature would introduce.

Another notable change can be seen in the first screen on Figure 23. Atticus was not enthusiastic about being able to place images onto the ground-plane using the AR camera. Instead, he suggested that we have the images be viewable in a more traditional fashion—in 2D as part of the UI.

At this point in the project, the codebase for Editour was already around three thousand lines of vanilla JavaScript. With the changing requirements, Editour was becoming onerous to maintain at this

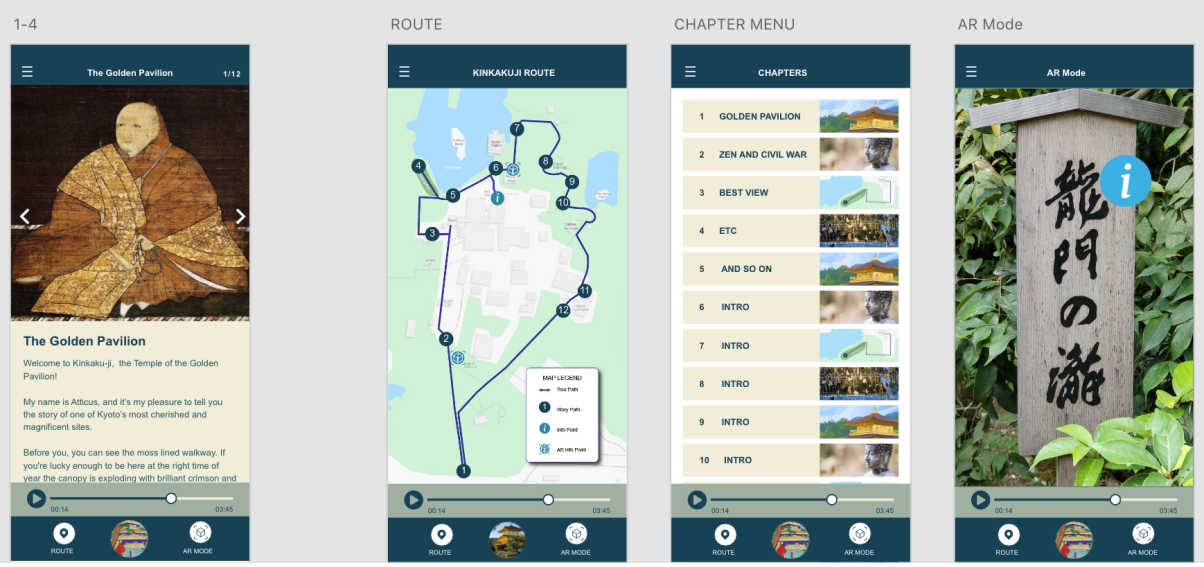

Figure 23: A screenshot of the Adobe XD project designed by Atticus Sims

stage of the project. Luckily, these changes were still manageable to implement.

With more time, we might have been able to get the ground-plane image placement feature to a state where Atticus might have liked it. We wanted to explore ways to theme the 2D images in the 3D world, such as by framing the images in a traditional Japanese pagoda. We had already spent two weeks on this feature, so dropping it was a setback we had to plan around.

On the same screen, there is a scrollable window of text for a transcript of the audio. Because of this, the Editour UI and tour metadata JSON file needed to be updated to accommodate one transcript per region. On top of this, the C# code that interprets the tour file needed to be updated to expect an audio transcript field. Figure 24 shows the UI elements that changed in order to add these new features.

With the ground-plane detection features stripped away from the main app, the AR focus shifted to translating the maps and signs using the camera. The fourth screen on Figure 23 shows a mock-up of this feature. Atticus took photographs of various signs throughout Kinkaku-ji for us to use as image targets. Hikaru Inoue helped us by translating all of the text in these photos from Japanese to English. In this design, the AR camera superimposes an "info" button on top of the sign. Tapping on this button brings up a separate UI element containing the translation, which can be seen in Figure 25. The details of this feature is discussed more thoroughly in Section 3.2.2.

Since Adobe XD is largely a prototyping and design tool, a considerable amount of legwork would still need to be done to rebuild the proposed design using Unity's UI components. Adobe XD allowed us to export a few important UI elements as PNGs, such as the buttons to enter AR mode and view the map. We briefly explored a few solutions to automating the task of getting the Adobe XD project into Unity. We found one paid extension (for $18) on the Unity Asset store called "Experience Importer - Adobe Xd files importer." The download page for this extension claims "Transfer AdobeXd project directly to Unity. Forget about hours of positioning objects. They're already there! Just drag .xd file to project and... voila!" [58]. We suspect this sounds too good to be true because it is. As of September 20th, 2019, there are 6 user reviews. All of the five-star reviews were have the caveat "Reviewer was

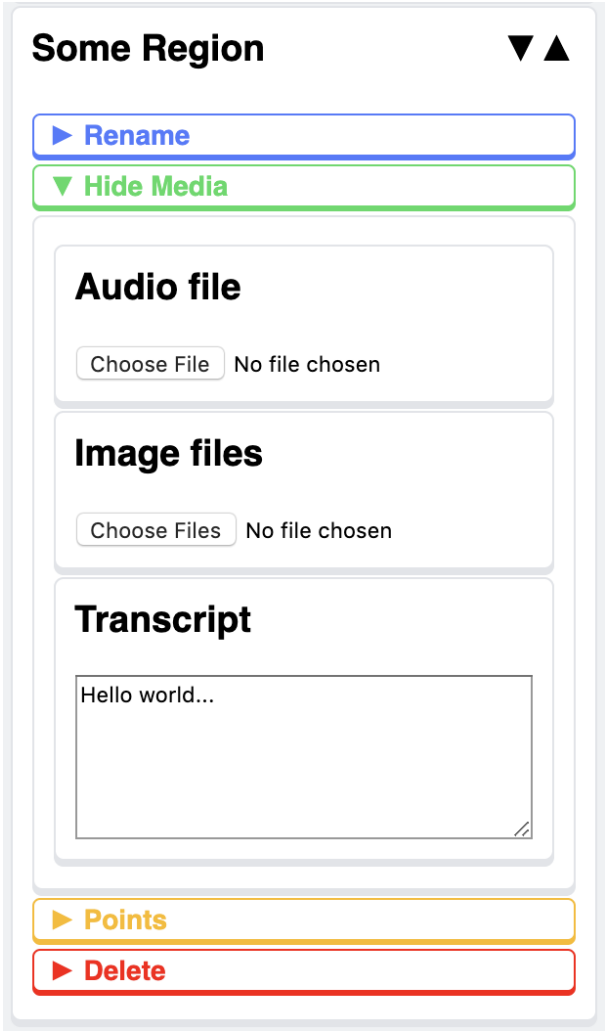

Figure 24: Additions to Editour to accommodate the new design

gifted package by publisher" which which we interpreted as a warning sign. Because of this, we decided to rebuild the UI largely from the ground up, using the Adobe XD file to guide us.

3.2.7 Final UI Implementation

The lessons we learned from the initial UI design (see Section 3.2.5) helped us create the final design in such a short period of time. Although the new UI is significantly more complicated than the original prototype, we were able to implement all of Atticus's desired features within the time limit.

One of the first hurdles we had to overcome was keeping certain UI elements constant while the user switches between the home screen, AR camera, and map view. The mode buttons, audio controls, and top bar need to remain on the screen at all times. To achieve this, we divided the screen into panels. Only one of the three main panels are set to "active" at a given time. Previously, the app left the AR camera running at all times, since there was only one main screen. In the new design, the AR camera is only set to "active" only when the AR panel is being shown. This prevents us from incurring unnecessary computational cost when the AR camera was not in use.

Another group of challenges were created by the content of the main screen, which showed the images

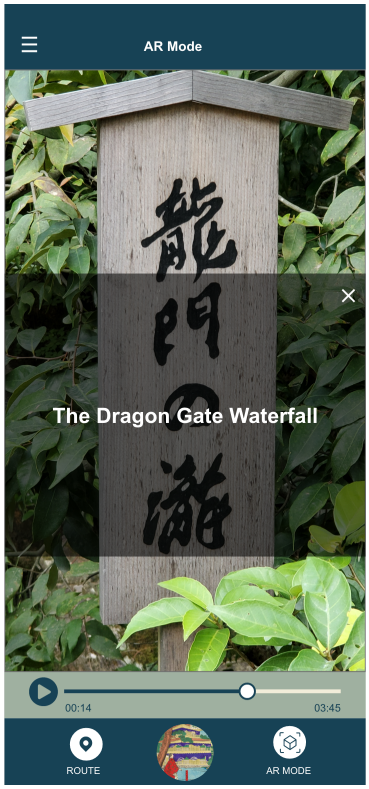

Figure 25: Translated sign using AR camera

and audio transcript associated with each region. To display the images, we wanted the user to be able to click through them like an image gallery.

We created the gallery out of a horizontal layout group, populated it with images dynamically in each region, and then had the buttons "lerp" the gallery to different positions in order to switch between each image in a smooth and pleasing way. As for the audio transcript, we knew we needed a scrollable text field that could accommodate varying amounts of text, since the audio files varied so greatly in length. We created a vertical layout group to create a text field that expanded vertically within predefined bounds to fit an arbitrary amount of text.

The last major UI implementation obstacle we had to overcome was a way to keep audio playing with the phone in a pocket. Atticus desired this sort of feature so that users could listen to the audio tour without using any of the other functionality the app provides, such as the image gallery or the AR features. Despite how common it is for apps to play audio in the background or while the phone is locked, Unity does not provide a way to accomplish this. Our workaround was to create a "pocket mode" that prevents UI elements from accidentally being triggered. The user exits pocket mode with a deliberate swipe to the right. If the slider is released partway through the action, the slider springs back to the its original position.

4 Testing

Both pieces of our project—Editour and ARuko—are designed to be used by people who are not programmers. Editour will be used by Atticus in the future to design more tours, and ARuko will be used by the general public. Because of this, we needed to do extensive testing to ensure not only that the products work as intended without bugs, but also that they are pleasant and easy to use for their intended users.

4.1 Testing Editour

Because Editour is simply an internal tool, our user testing was less rigid. Even so, the frontend and backend we built for Editour ended up being reliable. The few reliability issues stemmed from the hosting provider, which was out of our control.

4.1.1 Unit Testing the Editour Backend

To test the API and backend of the Editour web app we used the Mocha testing framework. Mocha is the most depended-upon package in the Node Package Manager [60], and is widely used by professional software development teams. It has a myriad of useful features that allow developers to quickly write unit tests for complex Node.js projects [61]. Most importantly for us, it has good support for testing asynchronous JavaScript functions, since most of the backend is written asynchronously. Along with Mocha we also used SuperTest, a library for writing high-level end-to-end HTTP tests [62].

For each of the six API endpoints that the backend responds to, we wrote an extensive series of end-to-end tests to ensure that each step along the pipeline from receiving a request to sending a response worked properly. We also wrote many fuzz tests to make sure the backend would respond correctly to invalid requests, for example, by responding with an HTTP 400 error message [50].

These tests were very useful in tracking down bugs and unexpected behavior in the backend, and also allowed us to make sure no functionality changed when adding new features or making changes to underlying logic later on.

4.1.2 Editour Frontend Testing

We suspect that Editour might have a life outside of this particular use case depending on how it is developed later on. This is why Editour is perhaps more polished than typical internal tools.

For the duration of the project however, Editour only had to be used our team and Atticus. Therefore, the testing for the frontend of Editour was less systematic than other parts of the software stack. This decision also helped us focus our efforts on testing the reliability and user experience of ARuko. Still, we were able to add many features and get the Editour experience to a polished state through our own testing, using GitHub's tools to keep us organized.

In order to test for bugs, all three of us periodically interacted with successive builds of the frontend. When bugs were found, we would log these as "Issues" on the GitHub repository.

Since we had to use Editour frequently to create test tours for ARuko, we were able to identify features that would make our own work more efficient. This worked as a kind of informal user testing. We kept track of feature requests using the same system as bug reports.

Issues and feature requests alike got resolved in future pull requests. We resolved over thirty issues and feature requests using this workflow. The GitHub issues tab (which can be see in Figure 26) was very useful for ensuring we made use of our user testing and bug testing.

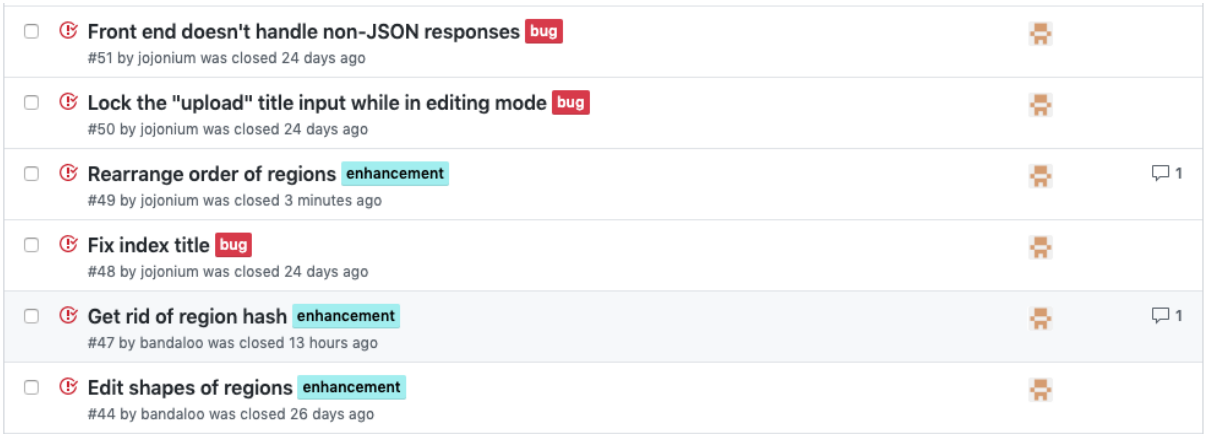

Figure 26: The GitHub issues tab for Editour

4.2 Unit Testing ARuko

The UI and AR features of the app are very important to the user experience. At the same time, ARuko is not as purely graphical as it may seem. There is a significant amount of "business logic" that interprets the tour from the Editour file. On top of this, the geometric tests to check whether the user is inside a particular arbitrarily-defined geographic region are very important to get right. Using Unity's built-in NUnit unit testing framework, we wrote tests to verify whether the tour data was being parsed correctly, and whether the tour would be run correctly.

Regions are a set of three or more points that define a closed polygon. Using the user's GPS location, the app needs to periodically run a check to see if the user is located within one of those polygons.

One decent algorithm for accomplishing this is by drawing a ray from a given point to infinity. Barring edge cases (which will be discussed shortly) the point is inside the polygon if the ray passes through an even number of line segments. The point is outside of the polygon if the ray intersects an odd number of times. Figure 27 does not serve as a proof, but should at least make it clear why this works. This algorithm's time complexity is $O(n)$, where n is the sum total of the number of vertices of all of the polygons being tested [63].

For this, we implemented a few mathematical functions for individual parts of the algorithm, such as one that tests to see if two lines intersect. We wrote unit tests to verify each part of the algorithm. This gave us confidence that the GPS component of the app would place users into the correct regions.

There is one edge case where this algorithm has difficulties. If the ray that is drawn for testing intersections passes directly through a vertex, the algorithm will count two intersections instead of just

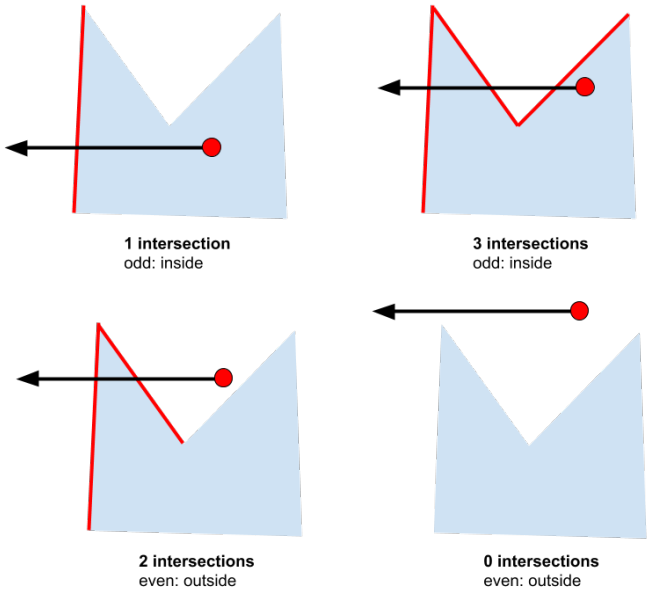

Figure 27: A visualization of the algorithm

one. Even though this is highly unlikely considering our use case, we decided that our algorithm should be robust enough to handle this on a matter of principle. We solved this by running a check on all vertices in the polygon to see if any were exactly colinear with the ray. If this is the case, one intersection count is subtracted from the total. Unit testing helped us catch and deal with this edge case.

4.3 Testing Taking a Tour with ARuko

Because it was not feasible to travel to Kinkaku-ji every time we needed to test the GPS functionality of the app, we used Editour to design an example tour around the Ritsumeikan Biwako-Kusatsu campus where we worked. This allowed us to catch various bugs before we created a finalized prototype for field-testing at Kinkaku-ji. The tour represented in Figure 28 is what we used for testing the app. The "C-Shaped Region" helped us test concave regions. This uniquely shaped region also prompted us to think about what UI behavior made the most sense if a user were to walk into a region, walk out and then in again. We decided that it was awkward if the audio restarted upon re-entering a region, since the audio would still be playing once a user left the region. As we began to implement more complex UI that changed based on GPS position in various ways, directly taking the tour by walking around was an invaluable way to test. Section 3.2.6 details how the UI grew in complexity in the late stages of the project. Having a way to regularly run field tests was integral to polishing the app in time before we revisited Kinkaku-ji. Because Editour was flexible enough to easily swap out media and add regions to the tour, Editour proved itself to be as helpful for testing as it was for crafting the final tour that could be taken at Kinkaku-ji.

To maximize testing efforts and negate having to physically walk around we also simulated the GPS coordinates in software. This allowed us to simulate walking in a single direction without actually having to build to a mobile device or move.

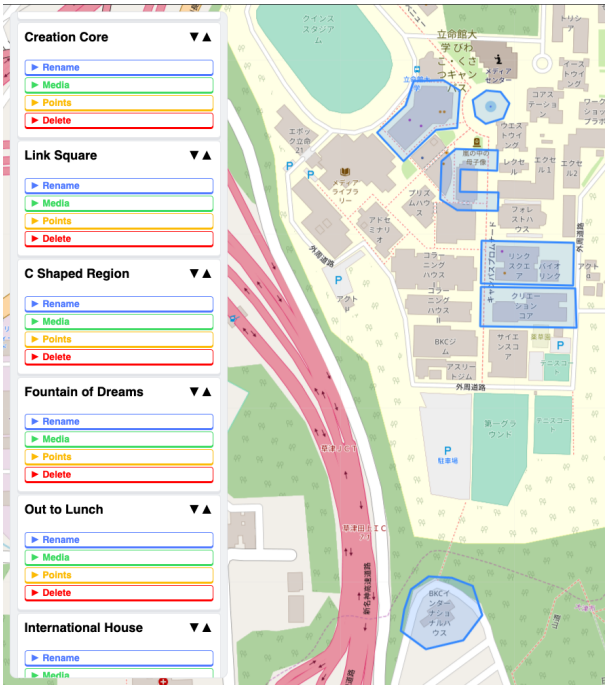

Figure 28: The Ritsumeikan example tour used for testing

4.3.1 Initial Field Testing

Our first day of on-site testing with the app took place on September 6$^{\text{th}}$. Using Editour, we rebuilt the izi.TRAVEL audio tour that Atticus had created previously. The first version of the reimplemented tour had regions that were relatively small; some regions were only a few meters across. This version of the tour can be seen in Figure 29. On this day of testing, we still had the limited UI described in Section 3.2.5.

Due to concerns about GPS accuracy, we adapted the tour-file `kinkakuji` to create a second, "chunkier" tour. In the tour-file `kinkakuji-chunkier`, we expanded all of the regions to accommodate for potential GPS inaccuracies that could place the user outside the range of the region. Figure 30 depicts the tour that we used for field testing at Kinkaku-ji.

We tested using two phones at the same time: an iPhone 8 and a Samsung Galaxy S7. The iPhone 8 is newer, more powerful and therefore more suited for use with AR. We primarily used this phone to test the AR features of the app. Even though the Galaxy S7 had difficulty with basic AR features such as ground-plane detection, it was still capable of triggering the audio track based on GPS position.

As discussed in Section 4.3, our previous testing involved creating a "mock tour" on the Ritsumeikan campus. While this let us know generally which features of the app were working, it was very difficult to know how ahead of time how well our app would work on-site.

We were surprised by how intuitively each region triggered on the first try. With a few exceptions, the audio started when the subject of the audio track was in view. During our first walkthrough of Kinkaku-ji, we triggered every region and audio track but one. The region that we missed on the first pass, "Final Viewpoint," requires the user to take a right at a fork in the road instead of continuing along the main path. This is not necessarily a problem with the design of the regions; instruction provided by

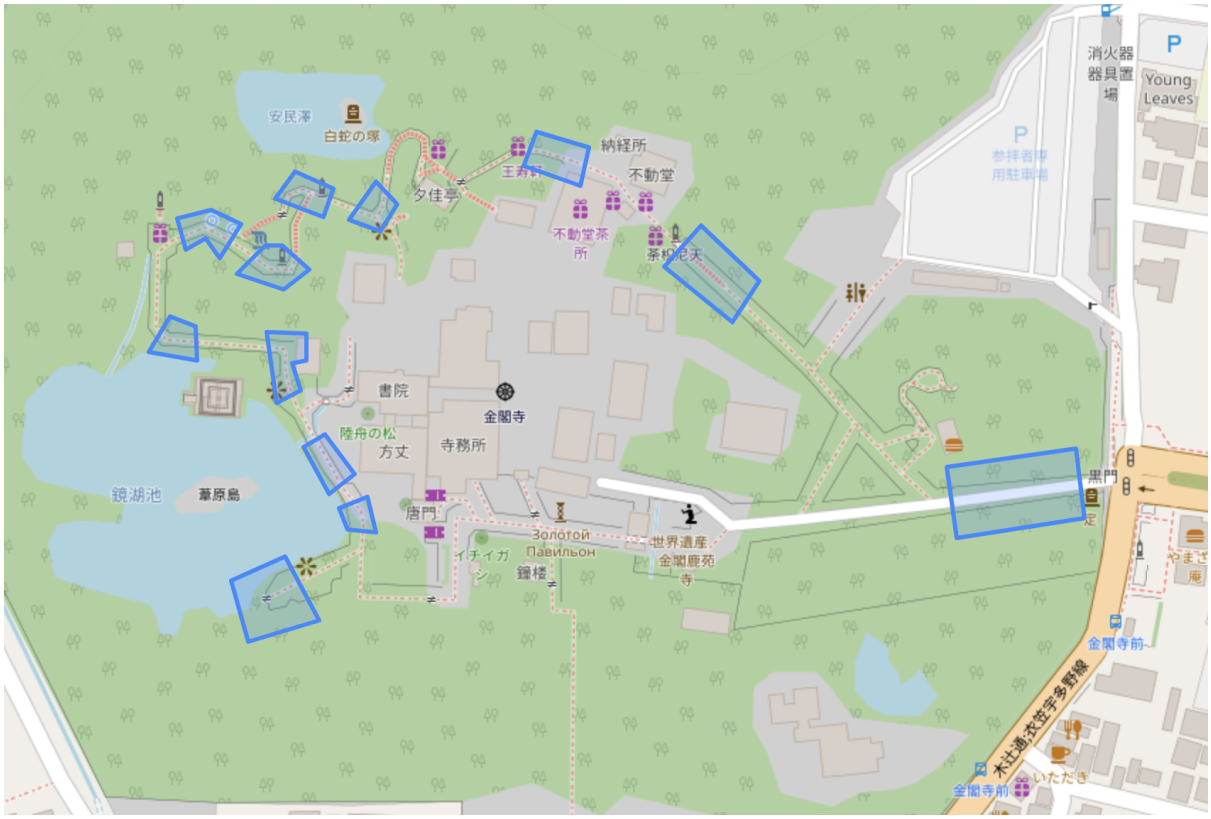

Figure 29: The original version of the Kinkaku-ji tour with small regions

the audio tour could remedy this, directing the user toward the viewpoint that is off of the main path.

Initially, we loaded the app onto the Galaxy S7 as a backup, since the AR features were not fully functional on a phone that old. At the same time, it was informative to take both phones through the tour. Initially, we made the design decision to update the GPS location every 60 frames. On a phone running the app at full speed, this should happen once every second. Because the Galaxy S7 was having trouble running the app at full speed, the GPS updated far less frequently. This was an oversight that would have been difficult to catch if we had not tested on hardware below the recommended specifications.

Other bugs resulted from oversights that were difficult to catch with our on-campus testing. For example, we noticed that we forgot to reset the audio slider when entering a new region. Because of this, when we entered a new region when the audio slider was past the endpoint of the new audio track, the audio would not play. For example, if the previous audio track were a minute long, and if the second track were only thirty seconds long, the app would try to play a thirty second audio track from the one minute mark. Since we were running the app on phones, we could not easily see when errors were thrown. Similarly, if the user were to enter a new region while thirty seconds into the previous audio track, the new audio track would start at the thirty second mark. During our on-campus testing, we were using songs rather than narration. Since the songs were relatively long and not always obvious when they started partway through, we did not catch this until we got out to Kinkaku-ji. Luckily, this was trivial to fix.

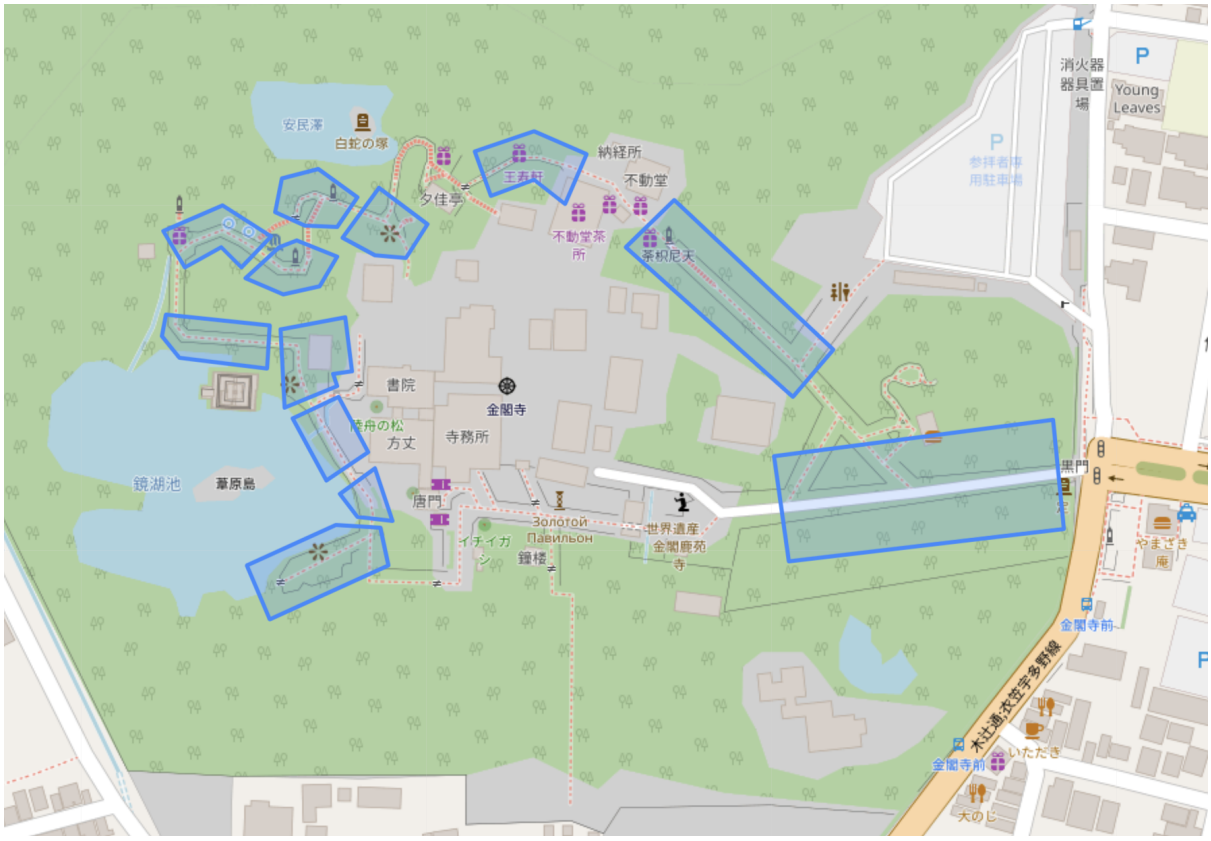

Figure 30: The "chunkier" version of the Kinkaku-ji tour with larger regions

4.3.2 Final Field Testing

On September 26th we returned to Kinkaku-ji with a nearly complete version of ARuko in order to conduct some final testing. We met up with Atticus, the IQP group, and our friend Hikaru and installed the beta version of ARuko on each of their phones. With little instruction about how the app works we had each of them try going through Kinkaku-ji while listening to the tour and using the app's features. Each member of our team also installed the beta version so we could test it.

This was our first time trying the AR image recognition on the actual signs at Kinkaku-ji, rather than photographs of the signs, and it worked surprisingly well. Vuforia was able to recognize almost all of the signs, even in different lighting than when the reference images were taken. For image targets that were small, far away, or obscured by shadows the image recognition was slower. We discovered that the targets that worked best were large, had a lot of high-contrast text, and were uniformly lit.

While testing we discovered some issues with the design of the tour. One region was completely misplaced, so the audio played at the wrong time while walking through the area. Several other regions were too big, so the audio started playing long before the user was actually near the feature it was discussing. Using the Editour web app, we were able to quickly correct these issues with the tour design once we returned to Atticus's office.

Additionally, one of the image targets had the wrong text in its information pop-up. These were all easy bugs to fix, but if we might not have noticed them if we had not tested the app on-site.

We also noticed that we were only able to augment a maximum of two image targets at a time. In

almost all cases, only two translatable signs were in view at once. The only scenario in which there were more than two image targets in view at a given time was with the four framed photos of the interior, which were arranged in a two by two grid. When standing at a distance from the grid, only two images would be augmented with the blue information button. By default, Vuforia limits the amount of image targets that the engine should accept simultaneously. In the Vuforia engine settings, we changed this limit to accommodate for four simultaneous images.

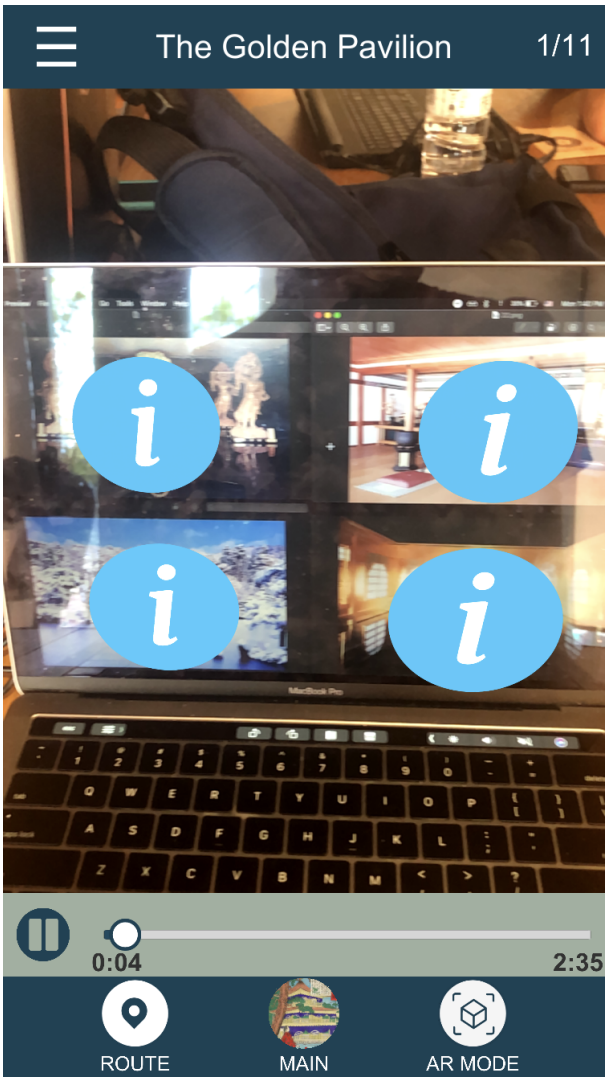

Figure 31: Tracking four image targets at once with improved info buttons.

Another major bug we discovered was that the GPS stopped updating when the main screen is not open. This means that in AR camera mode the app would not trigger new audio when you walked into a new region. This was caused by the GPS script being attached to a UI element on the main screen, which gets deactivated when switching to AR camera mode. This was easily resolved simply by moving this script.

After having Atticus go through the tour on his phone he also discovered several bugs and tweaks he wanted to make to the UI. The clickable information buttons on image targets were made larger so they are easier to see and tap. The UI buttons in the bottom menu bar were also made larger and easier

to press. The text “MAIN” was added under the central button that opens the screen with the images and transcript, so it was more clear what that button does. The header text at the top was too big, and sometimes overflowed onto other UI elements, so its size was reduced and split into two lines when the text was too long.

Testing with Atticus revealed another user interface issue we had not considered. Many modern smartphones, such as Atticus’s Samsung Galaxy S10+, have holes or notches cut out of the screen for cameras, speakers, and other hardware (see Figures 32 and 33). We had been designing our UI for rectangular phone screens, so the Galaxy S10+’s “hole punch” covered some UI elements in the top right corner. To account for this issue we used Unity’s built-in function to return a safe screen space to render the rectangular canvas in.

Figure 32: A Samsung Galaxy S10+ phone with distinctive “hole punch” cameras [64]

After going through Kinkaku-ji with the app twice we had each test participant fill out a simple survey giving their thoughts on the app. The full results of these surveys can be found in Appendix D.

Figure 33: An iPhone X with a large notch [65]

5 Alternative AR App Concepts

Throughout the duration of the project, the goalposts shifted multiple times. This section details some of the initial plans that we explored before Atticus shifted our focus to a walking tour app with AR features.

5.1 3D Spatialized Audio Tour

One idea that was proposed initially was an AR audio experience using 3D spatialized audio. Instead of traditional AR where the user is constantly looking through the camera, this experience would allow users to keep their phones in their pockets while taking the tour. Sound sources would change in relation to the real world as the user moved around.

We spent the second and third week of the project creating demonstrations to explore the possibility of a 3D spatialized audio tour. We found that this would be very difficult to do without the use of the camera. All of the technology for mobile AR we explored in Section 2.3 depended heavily on the camera. If we were to build a 3D spatialized audio tour that ran entirely without use of the camera, we would not be able to take advantage of any of the useful features of the AR libraries we explored. Furthermore, Unity would be relegated to the simple role of exporting our app to multiple devices; we would not have any need to take advantage of its 3D capabilities.

Therefore, we decided to see what was possible using the camera instead. We were able to get sound sources to fade in and out based on the distance to an image target. Naturally, if the image target were to go out of view of the camera, Vuforia stopped trying to estimate the camera's distance to the last known location of the image target. In order to make this work in the way Atticus imagined, we would need a way to shift to our own method of estimating distance to the sound source once the image target had been lost. It was difficult to estimate the amount of effort this would take (and whether it was even possible), since we would essentially be implementing a feature seemingly not supported by current AR libraries.

We explained the technical hurdles of this concept to Atticus. Ultimately, our focus shifted away from this idea for other logistical reasons. Initially, Atticus introduced the possibility of working closely with artists Kishi Bashi or Yannick Paget to produce the soundscape for the 3D spatialized audio tour. Yannick Paget is a French musician that became established in Japan. He is both conductor and composer, originally trained as a professional pianist and percussionist. He has written music for theater, film and orchestras around the world. Many of his original compositions were inspired by his experience living in Japan, such as Tears of Sakura (サクラの涙) [66]. Kishi Bashi was born in Seattle and grew up in Virginia. Kishi Bashi has appeared at many major music festivals, toured the US extensively, and has even released his own line of coffee. His latest album, Omoiyari, was released on May 31st, 2019 [67].

5.2 AR Art Gallery

Atticus Sims proposed the idea of working with Michael Whittle, a British artist received his PhD at the Kyoto City University of Arts in 2014. His art has appeared in dozens of solo and group exhibitions

across the world, including Kyoto, Beijing, London and New York [68]. Kyoto VR has worked with Michael Whittle in the past to create a 3D scan of a solo exhibition, "Portraits of Thought." This exhibition displayed diagrams inspired by real science and mathematics. Atticus provided us with some diagrams to test in the Vuforia developer portal, since Michael Whittle wanted to know how well certain pieces of art would work as AR image targets. All of images that were provided to us were ranked as "five stars" by Vuforia, indicating that those images would serve well as AR image targets.

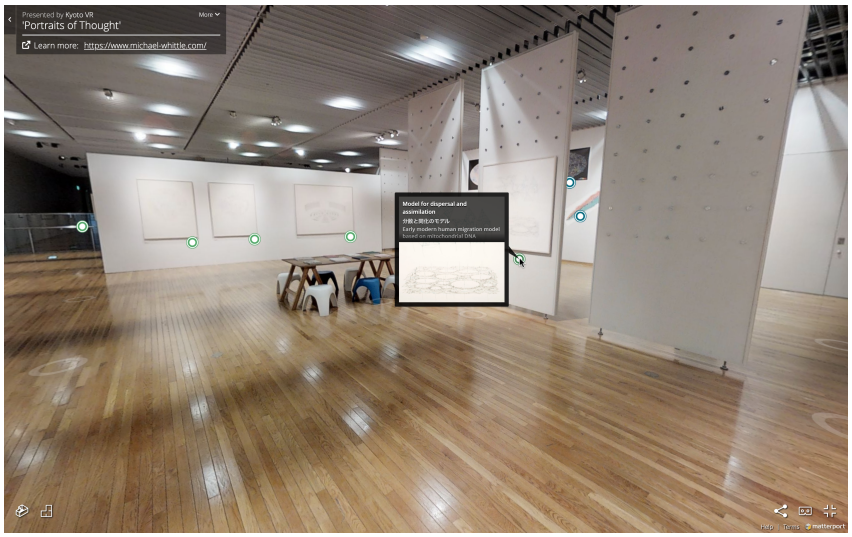

Figure 34: A screenshot of the 3D scan of "Portraits of Thought," accessible from the web

5.3 Alternative Uses for AR in ARuko

Even after the core functionality of ARuko was finalized as an audio tour with additional AR components, the exact nature of those AR components was still undecided until more details about the project crystallized. Atticus knew he wanted to use AR on the tour, but was not sure what to do with it. At first, we made recommendations to him based on the capabilities of Vuforia. We knew that image recognition and ground plane recognition were two of its strengths, so we suggested that displaying 3D models in certain locations using either of these features would be an interesting way to expand on the tour. With these recommendations in mind, Atticus expressed interest in using building facades as image targets. The intention then would be to tie content to a physical location in Kinkaku-ji, and use images of the location as targets. As explained in Section 3.2.5, this ultimately did not work out due to the limitations of Vuforia's image recognition.

After being initially deterred from image recognition, we thought that ground and mid-air plane recognition might the best way to implement AR in the app. It did not have to rely on images that meet Vuforia's criteria, and since it could be triggered anywhere, it appeared to be a good fit for an app focusing on walking around a site. Atticus explained that he primarily wanted to use AR to display art related to the site. With this in mind, we focused our efforts with ground and mid-air plane AR on ways to display 2D images. What we ended up creating first was a feature that allowed users to view certain images in AR, using ground plane detection, based on what region you were in. When users entered a

region of the tour, the image menu would be dynamically populated with buttons that, when tapped, would create a Plane primitive in Unity with the given image displayed on it and place it on the ground plane.

We encountered a few notable problems with this. Using a Plane primitive had deleterious effects on the way the object was lit in the Unity scene, as well as how it appeared when not viewed head-on. If viewed from an angle, the picture could become very dark and difficult to see. If viewed from behind, back-face culling would take effect and the plane would become completely invisible. Because of these issues, we decided to place the images on a Cube primitive instead. After showing this feature to Atticus during early testing at Kinkaku-ji, he decided that he wanted to move away from displaying images in AR. Looking back, the feature may have worked better with mid-air plane recognition so that the user could place the image directly in front of them.

In addition to the previously mentioned use cases for AR in ARuko, Atticus also expressed interest in using AR to trigger video content. He was particularly interested in triggering 3D videos through AR. As the scope of the project grew, and time ran short, the task of including 3D videos triggered by AR seemed more and more complicated, causing that idea to be phased out of the final project. Additionally, we never received any assets from Atticus to use for this feature, preventing us from spending time developing it.

5.4 Monetization for ARuko

Partway through our project, Atticus Sims let us know that he was looking to monetize the app we were developing for him. He also informed us that the IQP team would be looking for funding options in order to "bootstrap" the project. One monetization method he proposed was marketing the app to hotels, allowing hotels to sell tours for download through in-app purchases.

Getting an app approved to launch on the App Store or Google Play has many specific requirements and can be a very lengthy process. This might have been possible if our project had been more prescriptive from the start and we had more time.

We did perform some preliminary research about the monetization models Atticus proposed and brought forth some potential roadblocks regarding each one. We let Atticus know that we had logistical concerns about circumventing Apple's or Google's in-app purchases by locking content behind a different payment process as he suggested. Additionally, we informed Atticus that he would have to purchase a commercial Vuforia license in order to publish the app on any storefront [69], which Atticus was not willing to do at the time. The free version of Vuforia comes branded with a watermark in the bottom left corner.[7]

Our concerns about focusing on monetization for Kyoto VR were twofold. Firstly, we were worried that this kind of work strayed too far away from the academic intent of the project—to explore various ways in which AR and geolocation could be used in tandem to create an improved experience for tourists. For our purposes, the app was meant to demonstrate our discoveries in the form of a polished prototype.

[7]The final design of our app incidentally covers up the watermark with our own UI elements. This does not change the fact that we would not legally be able to deploy the app without a proper commercial license for Vuforia.

While we feel as though we achieved this goal, we restricted our experimentation in order to leave Kyoto VR with a potentially marketable product. We made it clear to Atticus that deploying our app on Google Play or the App Store would not be within the scope of our academic project as per our advisers' guidance.

6 Conclusion and Future Work

Our team is satisfied with the final result of this project, as we feel we have built a working platform that Kyoto VR will easily be able to build off of in the future. This section contains our final thoughts on the project and future work that could be done.

6.1 What Went Wrong

Although the finished product works as intended, we encountered several issues along the way. The changing scope and goal of the project meant we spent several weeks researching and working on features that did not make it into the final product (see Section 5), such as displaying images in 3D AR space and playing spatialized audio. If we had focused only on the features that made it into the final app from the start we could have had more time to polish and refine those features.

On a similar note, the requirements and scope of the project changed greatly from week to week, and sometimes even within one day. We spent a significant amount of time exploring concepts and features that were completely unrelated to the final product. We also did not receive official specifications, such as the design of the graphical user interface (GUI) Atticus wanted for the app, until very late in the development process.

Another challenge we faced was the difficulty in actually testing the app. The Ritsumeikan University Biwako-Kusatsu campus we stayed at is about ninety minutes away from Kinkaku-ji by expensive public transport, which meant that we were only able to go to the site to test ARuko twice. We could test some basic features from Ritsumeikan, but the image recognition targets existed only at Kinkaku-ji. More extensive user testing at the actual site could have helped us catch bugs earlier and lead to a more polished final product.

We had some initial trouble getting started with Unity. Different versions of Unity do not play well with each other. As a result, we often ran into bugs on older hardware, such as the laptops we had to work with. Furthermore, a stable version of the Unity editor is not available on GNU/Linux [70], so one of our team members had to install macOS in order to use it. We were able to resolve these issues in the first weeks of the project, but it consumed some of our valuable time in July.

6.2 What Went Well

Despite these issues, in the end we were able to create a useful working prototype. Editour is fully featured with everything needed to build an audio tour anywhere in the world. ARuko allows users to experience useful AR-enhanced tours in a pleasant and intuitive way. Together they provide a complete platform for building augmented reality tour guide apps.

Our team had broad range of skills. This made the division of labor easier, and throughout the project we never had issues splitting up work. For example, William Campbell was the only Interactive Media and Game Design (IMGD) major on our team, so he was the most familiar with game engines including Unity. Because of this, he primarily worked on ARuko in Unity.

Editour was not initially a requirement that Atticus had in mind, but it turned out to be a very useful tool for designing tours, allowing the app to be extensible. Two members of our team, Cole Granof and Joseph Petitti, had experience building web applications with HTML, CSS, and JavaScript, and Editour allowed them to use these skills to help the project.

The research time we put in at the beginning of the project also proved to be very useful. Our decision to use Unity and Vuforia based on this research made implementing many of the complex AR features easier later on. If we had chosen a less-ideal platform, we might have had to switch to a different one after wasting time on that poor decision.

6.3 Future Work

Although selling ARuko on app stores was not within the scope of our academic project, we were able polish the app enough that it would be feasible for Atticus to eventually sell it. The UI could use more testing with real users to make sure it is intuitive and easy to use. We also intended to add a feature where the app could automatically download tour files from the Editour server and import them without having to rebuild. We were not able to complete this feature due to time constraints, but it could easily be added because all of ARuko's business logic is abstracted and generalized to support multiple tours. Editour also already supports requesting and downloading the most recent versions of tours.

Editour is essentially feature complete. The most recent version, 1.3.0, can be found in the GitHub repository (`https://github.com/bandaloo/editour/releases/tag/v1.3.0`), and it contains all the features necessary to create and edit tours. In the future, Kyoto VR should move it to a more powerful hosting provider (see Section 3.1.6), but the current solution works for now.

Given more time and resources, the app could be augmented to include a tour browser that automatically downloads an unpacks tours from the Editour server. More historic locations around Kyoto and Japan in general should be considered for other tours. With the work our team has done, Kyoto VR now has a complete platform for creating and delivering unique tours, which can easily be expanded upon in the future.

References

[1] Kyoto VR. *About — Kyoto VR*. 2018. URL: http://kyoto-vr.com/en/about/ (visited on 10/01/2019).

[2] Andrew Webster. *Pokémon Go spurred an amazing era that continues with Sword and Shield*. Feb. 2018. URL: https://www.theverge.com/2019/2/28/18243332/pokemon-go-sword-shield-franchise-history-niantic-nintendo-switch (visited on 08/29/2019).

[3] Will Mason. *8 of the Top 10 Tech Companies in the World are Invested in VR/AR*. Mar. 2016. URL: https://uploadvr.com/8-of-the-top-10-tech-companies-invested-in-vr-ar/ (visited on 09/22/2019).

[4] TEDxPlainesWilhems. *Augmented Reality, Technology Alleyway*. Licensed under CC BY-NC-ND 2.0. May 2016. URL: https://flickr.com/photos/tedxpw/27223866876/ (visited on 10/01/2019).

[5] Aaron Saenz. *Augmented Reality does time travel tourism*. Nov. 2009. URL: https://singularityhub.com/2009/11/19/augmented-reality-does-time-travel-tourism (visited on 08/19/2019).

[6] Miranda Katz. *Augmented Reality Is Transforming Museums*. Apr. 2018. URL: https://www.wired.com/story/augmented-reality-art-museums/ (visited on 08/23/2019).

[7] Nicholas Bornoff. *The National Geographic Traveler: Japan*. National Geographic Society, 2000. ISBN: 0-7922-7563-2.

[8] Zeyi Fan. 金閣寺 *Kinkaku-ji*. Licensed under CC BY-NC 2.0. Jan. 2018. URL: https://flickr.com/photos/fanzeyi/39527667122/ (visited on 10/01/2019).

[9] United Nations Educational, Scientific and Cultural Organization. *Historic Monuments of Ancient Kyoto (Kyoto, Uji and Otsu Cities)*. URL: http://whc.unesco.org/en/list/688 (visited on 08/30/2019).

[10] Japan Guide. *Kinkakuji (Golden Pavilion)*. 2019. URL: https://www.japan-guide.com/e/e3908.html (visited on 09/23/2019).

[11] Wybe Kuitert. *Themes in the History of Japanese Garden Art*. Honolulu: University of Hawai'i Press, 2002, pp. 30–52. ISBN: 0-8248-2312-5.

[12] jeriaska. *Augmented Reality Game*. Licensed under CC BY-NC 2.0. Mar. 2010. URL: https://flickr.com/photos/jeriaska/4424958623/ (visited on 10/01/2019).

[13] Julie Carmigniani et al. "Augmented reality technologies, systems and applications." In: *Multimedia Tools and Applications* 51.1 (Jan. 2011), pp. 341–377. DOI: 10.1007/s11042-010-0660-6.

[14] Mike Boland. *762 Million AR-Compatible Smartphones in the Wild*. July 2018. URL: https://arinsider.co/2018/07/25/762-million-ar-compatible-smartphones-in-the-wild/ (visited on 09/27/2019).

[15] Kevin Bonsor and Nathan Chandler. *How Augmented Reality Works*. Nov. 2018. URL: https://computer.howstuffworks.com/augmented-reality.htm (visited on 08/19/2019).

[16] Benjamin Resnick. *Visualizing High Dimensional Data In Augmented Reality*. July 2017. URL: https://medium.com/inside-machine-learning/visualizing-high-dimensional-data-in-augmented-reality-2150a7e62d5b (visited on 08/23/2019).

[17] Lucas Matney. *Shopify is bringing Apple's latest AR tech to their platform*. Sept. 2018. URL: https://techcrunch.com/2018/09/17/shopify-is-bringing-apples-latest-ar-tech-to-their-platform/ (visited on 08/23/2019).

[18] Meghna Sharma. "Augmented reality could be advertising world's best bet." In: *The Financial Express* (Apr. 2015).

[19] Hana Stewart-Smith. *Education with Augmented Reality: AR textbooks released in Japan*. Apr. 2012. URL: https://www.zdnet.com/article/education-with-augmented-reality-ar-textbooks-released-in-japan-video/ (visited on 08/23/2019).

[20] Stuart Eve. "Augmenting Phenomenology: Using Augmented Reality to Aid Archaeological Phenomenology in the Landscape." In: *Journal of Archaeological Method and Theory* 19.4 (Dec. 2012). DOI: 10.1007/s10816-012-9142-7.

[21] Miguel Concepcion. *Pokemon Go review*. July 2016. URL: https://www.gamespot.com/reviews/pokemon-go-review/1900-6416477/ (visited on 08/29/2019).

[22] Virginia State Parks. *Pokemon Go located by the James River at Powhatan State Park*. Licensed under CC BY 2.0. July 2016. URL: https://www.flickr.com/photos/vastateparksstaff/27972275293/ (visited on 08/29/2019).

[23] Volodymyr Bilyk. *Augmented Reality Issues - What You Need to Know*. Sept. 2018. URL: https://theappsolutions.com/blog/development/augmented-reality-challenges/ (visited on 09/03/2019).

[24] Adriana Blum. *Challenges in augmented reality Mobile App Development*. July 2018. URL: https://www.geospatialworld.net/blogs/challenges-in-ar-mobile-app-development/ (visited on 09/03/2019).

[25] Microsoft. *HoloLens 2 pricing and options*. May 2019. URL: https://www.geospatialworld.net/blogs/challenges-in-ar-mobile-app-development/ (visited on 09/03/2019).

[26] izi.TRAVEL. *Platform overview for the museums (EN)*. Feb. 2015. URL: https://www.youtube.com/watch?v=GHLvz2kyYPA (visited on 08/30/2019).

[27] izi.TRAVEL. *FAQ — izi.TRAVEL*. URL: https://izi.travel/en/faq (visited on 08/30/2019).

[28] Paul Davidson. *Kinkakuji, the Golden Pavilion*. Licensed under CC BY 2.0. Nov. 2005. URL: https://www.flickr.com/photos/pauldavidson/68303779/ (visited on 08/30/2019).

[29] Cameron Summerson. *What Are the ARCore and ARKit Augmented Reality Frameworks?* Apr. 2018. URL: https://www.howtogeek.com/348445/what-are-the-arcore-and-arkit-augmented-reality-frameworks/ (visited on 08/29/2019).

[30] Vuforia. *Vuforia Fusion*. URL: https://library.vuforia.com/articles/Training/vuforia-fusion-article.html (visited on 08/29/2019).

[31] Danny Moon. *AR For Everyone: Building cross platform AR (ARCore/ARKit) Apps with React Native and ViroReact*. Mar. 2018. (Visited on 08/29/2019).

[32] Apple. *Get Ready for ARKit 3*. 2019. URL: https://developer.apple.com/augmented-reality/arkit/ (visited on 08/29/2019).

[33] William Todd Stinson. *AR Foundation support for ARKit 3*. June 2019. URL: https://blogs.unity3d.com/2019/06/06/ar-foundation-support-for-arkit-3/ (visited on 08/29/2019).

[34] Wikitude. *Wikitude Augmented Reality: the World's Leading Cross-Platform AR SDK*. Aug. 2018. URL: https://www.wikitude.com/ (visited on 09/04/2019).

[35] Jason Odom. *Easily Make Your Own Pokémon GO with Motive.io*. Mar. 2017. URL: https://mobile-ar.reality.news/news/easily-make-your-own-pokemon-go-with-motive-io-0176468/ (visited on 09/20/2019).

[36] Motive.io. *VR & AR Training Solutions for Enterprise - Motive.io*. 2019. URL: https://www.motive.io (visited on 09/20/2019).

[37] Viro Media. *Viro AR*. URL: https://viromedia.com/viroar (visited on 08/19/2019).

[38] Facebook. *React. A Javascript Library for Building User Interfaces*. 2019. URL: https://reactjs.org (visited on 08/19/2019).

[39] Techopedia. *Definition - What does Emulation mean?* 2019. URL: https://www.techopedia.com/definition/4787/emulation (visited on 09/04/2019).

[40] *Run apps on the Android Emulator*. URL: https://developer.android.com/studio/run/emulator (visited on 09/09/2019).

[41] Mohammed K Alzaylaee, Suleiman Y Yerima, and Sakir Sezer. "Emulator vs real phone: Android malware detection using machine learning." In: *Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics*. ACM. 2017, pp. 65–72.

[42] Mike Warick. "MS-DOS Emulation for the 64." In: *Compute!* (Apr. 1988), p. 43. URL: https://archive.org/stream/1988-04-compute-magazine/Compute_Issue_095_1988_Apr#page/n43/mode/2up (visited on 09/04/2019).

[43] Samuel Axon. *Unity at 10: For better—or worse—game development has never been easier*. Sept. 2016. URL: https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/ (visited on 09/30/2019).

[44] Romain Dillet. *Unity CEO says half of all games are built on Unity*. Sept. 2018. URL: https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/ (visited on 09/30/2019).

[45] ECMA International. *Standard ECMA-334 - C# Language Specification*. 2nd ed. 2011. URL: https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-334.pdf.

[46] Asano Noboru. *Kinkaku-ji in Kyoto*. 2013. URL: http://kyoto.asanoxn.com/places/kinkaku/kinkakuji.htm (visited on 09/27/2019).

[47] David Yound and Michiko Yound. *The Art of Japanese Architecture*. North Claredon, VT: Turtle Publishing, 2007.

[48] Joe deSousa. *Japanese Garden, Kyoto*. Licensed under CC0 1.0. July 2014. URL: https://flickr.com/photos/mustangjoe/16477259292/ (visited on 09/27/2019).

[49] Albert Borowitz. *Terrorism for Self-Glorification: the Herostratos Syndrome*. Kent State university Press, 2005, pp. 49–62. ISBN: 978-0-87338-818-4.

[50] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC 7231. RFC Editor, June 2014. URL: https://www.rfc-editor.org/rfc/rfc7231.txt.

[51] ECMA International. *Standard ECMA-262 - ECMAScript 2019 Language Specification*. 10th ed. 2019. URL: https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf.

[52] Leaflet. *Overview*. 2019. URL: https://leafletjs.com/ (visited on 09/20/2019).

[53] Lauren Orsini. *What you need to know about Node.js*. Nov. 2013. URL: https://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs/ (visited on 08/29/2019).

[54] Tim Caswell and Jordan Harband. *Node Version Manager - POSIX-compliant bash script to manage multiple active node.js versions*. Apr. 2019. URL: https://github.com/nvm-sh/nvm (visited on 09/25/2019).

[55] The Apache Software Foundation. *Using mod_rewrite for Proxying*. 2019. URL: https://httpd.apache.org/docs/2.4/rewrite/proxy.html (visited on 09/25/2019).

[56] Charlie Robbins. *forever - npm*. Jan. 2019. URL: https://www.npmjs.com/package/forever (visited on 09/25/2019).

[57] John Baichtal. *Thingiverse.com Launches A Library of Printable Objects*. Nov. 2008. URL: https://www.wired.com/2008/11/thingiversecom/ (visited on 10/04/2019).

[58] Glass Eye Studio. *Experience Importer - Adobe Xd files importer*. 2019. URL: https://assetstore.unity.com/packages/tools/integration/experience-importer-adobe-xd-files-importer-149207 (visited on 09/20/2019).

[59] J.T. Harvey et al. "An analysis of the forces required to drag sheep over various surfaces." In: *Applied Ergonomics* 33 (2002), pp. 523–531. DOI: 10.1016/S0003-6870(02)00071-6.

[60] Tidelift. *Popular NPM Projects - Libraries.io*. 2019. URL: https://libraries.io/search?order=desc&platforms=NPM&sort=dependents_count (visited on 09/22/2019).

[61] Mochajs. *mocha - npm*. 2019. URL: https://www.npmjs.com/package/mocha (visited on 09/22/2019).

[62] SuperTest. *supertest - npm*. 2019. URL: https://www.npmjs.com/package/supertest (visited on 09/22/2019).

[63] GeeksForGeeks. *How to check if a given point lies inside or outside a polygon?* URL: https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/ (visited on 09/22/2019).

[64] Aaron Yoo. *Galaxy S10 5G Tempered Glass*. Licensed under CC BY-ND 2.0. Apr. 2019. URL: https://flickr.com/photos/thebetterday4u/40751810673/ (visited on 09/27/2019).

[65] Aaron Yoo. *iPhone X*. Licensed under CC BY-ND 2.0. Dec. 2017. URL: https://flickr.com/photos/thebetterday4u/23938580207/ (visited on 09/27/2019).

[66] Yannick Paget. *About*. 2016. URL: https://www.yannickpaget.com/about/ (visited on 09/28/2019).

[67] Kishi Bashi. *About*. 2019. URL: http://www.kishibashi.com/about#bio (visited on 09/28/2019).

[68] Michael Whittle. *Biography*. 2019. URL: https://www.michael-whittle.com/biography.html (visited on 09/28/2019).

[69] PTC Inc. *Pricing and Licensing Options*. 2018. URL: https://library.vuforia.com/articles/FAQ/Pricing-and-Deployment-Plans (visited on 10/03/2019).

[70] Martin Best. *Announcing the Unity Editor for Linux*. May 2019. URL: https://blogs.unity3d.com/2019/05/30/announcing-the-unity-editor-for-linux/ (visited on 10/02/2019).

[71] Jon Terry. *What Is a Kanban Board?* 2019. URL: https://www.planview.com/resources/articles/what-is-kanban-board/ (visited on 09/30/2019).

[72] Atlassian. *Git Feature Branch Workflow*. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow (visited on 10/01/2019).

[73] Tom Preston-Werner. *Semantic Versioning 2.0.0*. URL: https://semver.org/ (visited on 10/03/2019).

[74] Japan Guide. *Noh Theater*. Oct. 2018. URL: https://www.japan-guide.com/e/e2091.html (visited on 10/03/2019).

[75] Kyoto VR. *Kyoto Project: NOxAR*. Mar. 2018. URL: http://kyoto-vr.com/en/kyoto-project-noxar/ (visited on 10/03/2019).

Appendices

Appendix A Version Control

Over the course of the project, we had to develop multiple applications in different languages. In total, we created five different repositories: the blog, the LaTeX source code for our paper, the Arduino code for the heart project (see Appendix C) and of course one for Editour and another for ARuko. These five repositories span across multiple different markup and programming languages, including C#, C++, JavaScript, HTML, CSS, and the LaTeX used to typeset this paper. This gave us a welcome amount of practice using Git for version control, as well as GitHub's other features oriented around coordinating a team.

All three members of the team were contributing code and updating assets such as graphics and sound. Code complexity grew quickly as more features were added across the entire software stack. GitHub and Git enabled us to view the history of the code and revert back to working versions when absolutely necessary. We researched the proper methodology of using version control in a small team, applying this methodology as best we could. This enabled us to work together efficiently, track down bugs, undo mistakes, and review each other's code for quality when time allowed.

A.1 GitHub Tools

Some members of our team were very excited by the workflows GitHub's tools enabled. In this section, we will discuss some of the tools, some of which greatly aided in our organization and productivity.

Initially, we experimented with using a four-column Kanban board to keep track of our progress. Figure 35 shows a screenshot of this page on GitHub.

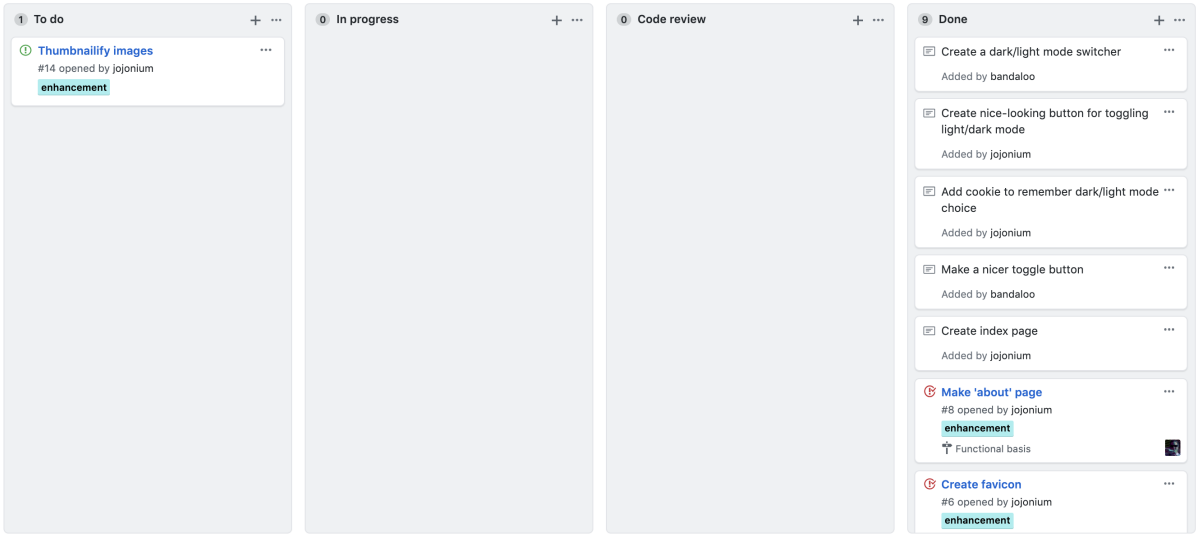

Figure 35: A screenshot of the Kanban board we used for developing our biweekly blog

The Kanban technique originates from Toyota in the 1940s. Kanban (看板) means "visual signal" or "card" in Japanese, which describes the core component of the technique [71]. At Toyota, line-workers

would move colored cards to notify other teams that parts needed for assembly work were in short supply. Now, Kanban boards are integral to Agile software development. Some teams still use physical Kanban boards, often using sticky notes and whiteboards.

The advantage of using a Kanban board is that each feature or bug that is being worked on can be assigned directly to a collaborator on the project. It is then that person's responsibility to work on those tasks and update the status of those tasks on the board. We abandoned this when moving onto the main project because we came to realize that this structure was not necessary—there were only three members on the team, and we were almost always in the same room when working on the project. For Editour, we tracked features and issues using the "Issues" page instead, which is discussed in Section 4.1.2.

Creating "pull requests" through the GitHub web interface has a few advantages over directly merging code using Git. First, the pull request feature integrates with the "Issues" page very well. By typing "resolves" along with the number associated with an issue into the description of a pull request, GitHub will automatically close the issue when the pull request is approved. This is not only remarkably convenient, but also self-documenting since the descriptions of pull requests would contain the issues they were resolving. Secondly, creating a pull request allows code to be reviewed by other members of the team very easily. Reviewers can view the entire diff file[8] and add comments line-by-line. Other collaborators can easily respond to those comments. Reviewers can also request certain features be changed before the branch is merged, which allowed us to enforce that issues with the code were actually addressed.

The way we incorporated branches into our workflow helped us collaborate in a way that kept us organized. In general, if one of us was working on a specific feature, we would create a branch with a name descriptive of that feature. This methodology minimized difficult-to-resolve merge conflicts. We modeled this workflow off of the "Git Feature Branch Workflow" Atlassian tutorial [72].

In order to clearly indicate new versions of the software, we use "Semantic Versioning," which is a scheme for version numbers created by Tom Preston-Werner, cofounder of GitHub [73]. In this scheme, the full version number consists of three integers separated by periods in the form `major.minor.patch`. The major version number is incremented when an incompatible change is made. In our case, an incompatible change for Editour would entail changing the format of the metadata file used to construct a tour such that the frontend could not interpret older tour files. The exception to this rule is major version 0, where the product is not considered stable and incompatible changes can be made at any time. When Editour was in this phase of development, we changed the format of the metadata; this would have broken previous versions but this was allowed because it was still in an unstable beta version. The minor version number is incremented when functionality is added but remains backwards compatible with previous versions. An example of this is the "transcript" field we added to the metadata. The frontend was still able to load tours made before this additional information was added to the metadata JSON. We added many other features throughout the development process without breaking existing functionality. The patch version number is for simple bug-fixes that also do not break compatibility. This number gets incremented the most frequently in practice.

Once we deployed Editour to a server so that Atticus could begin to use it, we changed the version

[8] A "diff" is a file generated by Git that shows the differences between two files.

to 1.0.0 to indicate the first release version. At this point, we were committed to not making any backwards-incompatible change. Since Atticus would be interacting with Editour after the release of 1.0.0, it was important that the master branch remain as stable as possible. Because of this, we created a "dev" branch where we could test experimental features to improve the experience and functionality of Editour. Maintaining a stable branch along with a separate experimental branch reduced the risks involved with continuing to iterate on the software's features post-release.

In a similar vein, with ARuko we used branches to ensure we had a stable working version. The night before field testing at Kinkaku-ji, we would cease work on the most recent branch, and build the app to our phones. This way, we would always have an exact snapshot of the code we used for that day of testing.

Appendix B Editour API Documentation

Here is a list of API functions the Editour backend handles. The response is usually a JSON like this with fields called `status` and `message` The status is an HTTP status code like 200 (OK) or 404 (Not Found) [50], and the message is a string or stringified JSON that contains the data of the response.

GET `/edit/:name`

Returns the metadata file of the most recent tour with the given name. If successful, it returns a status of 200 and the stringified metadata as the message. If no tour with the given name is found it returns 404, and if the server encounters an internal error while processing the request it returns 500.

GET `/tour/:name`

This is the only API endpoint for which the response is not a JSON. Instead, the response is the zip file of the most recent version of the tour with the given name. Returns 404 if a tour with the given name isn't found or a 500 if there is an internal server error while processing the request.

GET `/tours`

Gets a list of unique tours on the server, returning a list of their names as the message of its response. Returns 200 if successful or 500 if there is an internal server error while processing the request.

POST `/edit`

Used for editing an existing tour. This request should contain the fields `tourName`, `oldName`, and `metadata`, along with any number of files. The server will use the new metadata as well as the old and new files to create a new version of an existing tour. Returns 201 if the tour was edited successfully, 400 if the request was invalid, 404 if the requested tour could not be found, or 500 if a server error occurred while processing the request.

POST `/upload`

Used for uploading a new tour. This request should include a `tourName` field and a `metadata` field, along with any number of files. It verifies the uploaded tour and creates a zip for it on the server. Sends a 201 if the tour was created successfully, a 400 if the request is invalid, or a 500 if a server error is encountered.

DELETE `/tour/:name`

Deletes all versions of the tour with the given name from the server. Returns 200 if successful, along with a message saying how many versions were deleted, 404 if no tour with that name could be found, or 500 if a server error was encountered.

Appendix C Ritsumeikan Heart Project

Throughout our stay at Ritsumeikan University, we had the opportunity to work closely with other students to build a project that involved the integration of hardware and software. With the help of students and professors alike, we were able to produce an educational physical model of a heart controlled by an Arduino microcomputer. Professor Noma Haruo gave us tremendous assistance with hardware. He taught us how to properly use the tools around the lab, and purchased mechanical and electrical components we would need to complete our task. Professor Amano was always available to answer questions related to biology, and helped us understand the mathematical model we would need to implement to drive the physical model with software. We worked on this project every Wednesday, completing the final model during the last week of our stay.

C.1 Project Goal

The goals and requirements of this project were set out by Professor Noma, a professor at Ritsumeikan University. The goal of the project was to create an interactive model of a heart to teach heart functions to non-specialists, such as students in middle school. This meant that the model had to be highly visual and simple to understand. Each visual component would have to be designed in an intuitive fashion so that students could begin interacting with the model without advanced knowledge of biology.

C.2 The Heart Simulation

We were presented with a set of equations that defined the mathematical model of the heart we would be using to drive our physical model. Along with the differential equations, we were also given an analogous circuit model that could also be described by the same equations. The circuit and the equations can be seen in Figure 36. We were also provided with source code for an implementation of this model in C. We would still need to modify this code to calculate the output variables we would need to properly operate the physical model, and integrate the model into the main loop of the Arduino source code with the appropriate timing.

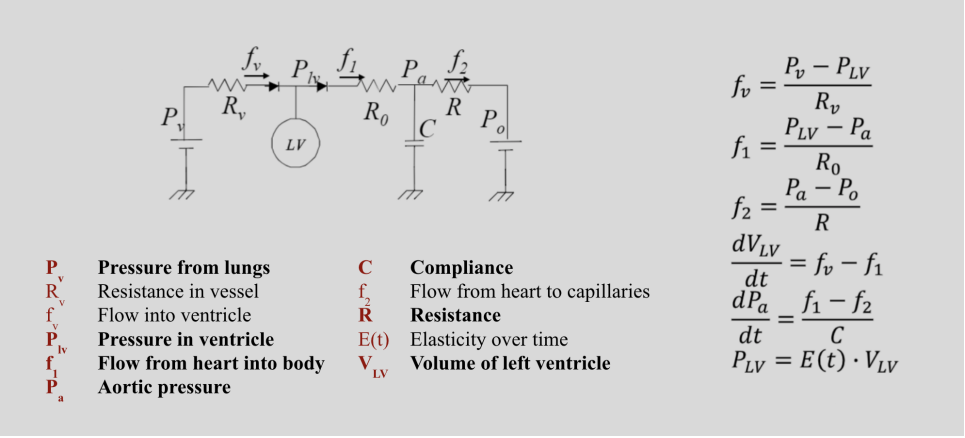

Figure 36: The reference model we used to implement the simulation

C.3 Supplies

With Professor Noma as our guide, we visited "Den-Den Town," or Nipponbashi, in Osaka in order to gather supplies we would need in order to construct the heart. One of the most important purchases of this trip was a two-meter long strip of Adafruit NeoPixel light emitting diodes (LEDs). The design of this reel of LEDs enabled us to cut the strip at specific points to create shorter strips. Each LED on the strip was individually addressable, allowing us to set the color of any LED along the strip. This perfectly suited our needs for representing blood flow.

Professor Noma had various microcomputers for us to experiment with in his lab, including an obniz, an M5Stack, a Raspberry Pi, and an Arduino. The Arduino was the only computer that had a "motor shield" component which we believed would be necessary for controlling the pumps.[9] Because of this, we chose the Arduino to be the core of the physical model. This microcomputer ran the simulation and output data to the various hardware components, such as the digital display, LED strips and switch to control the air valves.

C.4 Interactive Heart Model Design

The design for the interactive model is pictured in Figure 37. The most prominent feature of the model is the plastic bottle containing the balloon. The volume of this balloon represents the volume of the left ventricle. Instead of inflating the balloon with the air compressor directly (pictured on the left of Figure 37), we opted to pressurize a sealed two-liter soda bottle and suspend a balloon inside. By pumping more air into the bottle, we could reduce the volume of the balloon since the external pressure would be made greater than the internal pressure; the balloon would shrink as a result. To accomplish redirecting the flow of air, we designed a setup involving two valves.

Like the balloon, which represents the volume of the left ventricle, the LED strips represent both pressure and blood flow in different parts of the body. Pressure is represented by color. The lower the pressure, the more blue the lights will appear. Red represents high pressure. The LEDs can display any color in between blue and red, so the light strips are often a pinkish hue.

We were able to program custom animations by updating the color and intensity of the LEDs in a rapid fashion. We represented blood flow by changing the phase of a sinusoidal wave over time.[10] Our design included two strips of twenty LEDs. The LED strips are represented by the red arrows on Figure 37. The rate of the animation on the right strip of LEDs represents blood flow out of the left ventricle (whose value is f_1 in the model). The color corresponds to P_a in the model, which is the pressure in the aorta. The animation speed of the LED strip leading back into the heart is controlled by f_v, the flow back into the ventricles. The pressure corresponding to the color of this strip is P_v, or venous pressure. In our model, this value is not an output variable. Instead, P_v is held constant. Consequently, the color of this LED strip does not change over time.

The physical model also contains a digital number display, which can be seen in the top right corner

[9] In the end, we did not need this component, since we abandoned the idea of taking apart a pump to drive the DC motor directly.

[10] For clarity, we elongated the "troughs" of the sine waves, so the pattern of intensity was not a true sine wave.

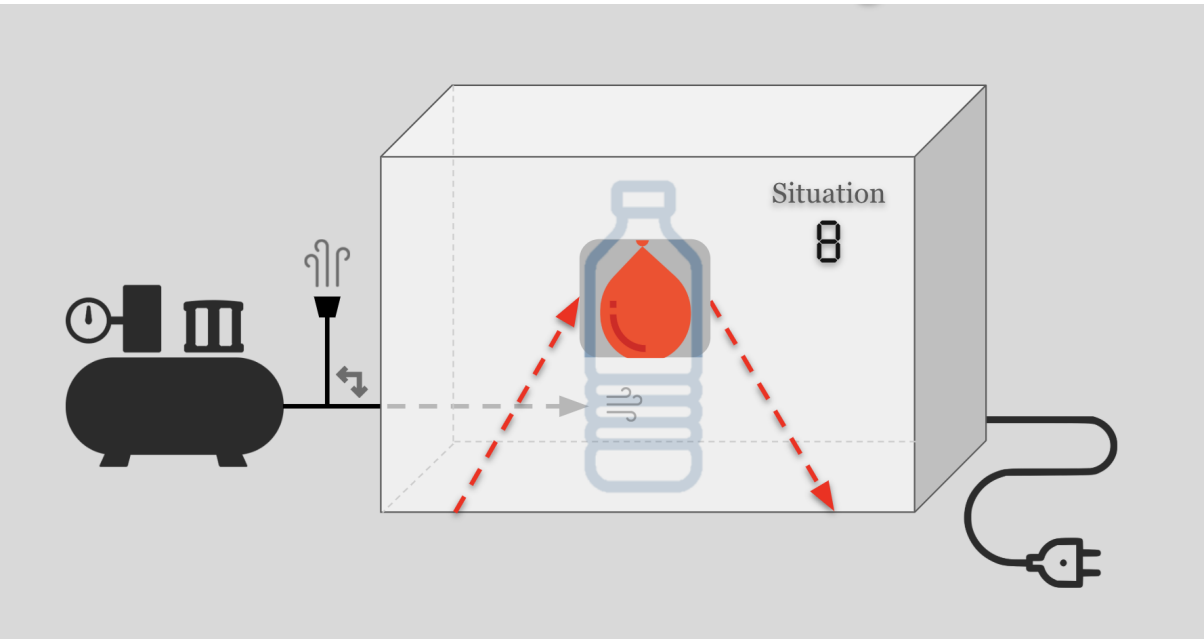

Figure 37: A diagram representing the physical model of the heart

on Figure 37. The display shows the number of the active scenario. Clicking on a scenario button in the GUI (discussed in Section C.5) will change the number being displayed.

C.5 User Interface for Heart Model

Our initial plan for the user interface was to control input and output parameters with dials. The numeric values of the input parameters would have been displayed on an array of digital number displays. Due to time constraints, we opted to support five preprogrammed scenarios that could be toggled from a GUI. Figure 38 represents the final version of the GUI we use to interact with the physical model.

The red buttons along the top of the screen allow the user to toggle between different scenarios. Each scenario also includes a helpful graphic below each button.

Not only can the GUI communicate with the Arduino; the Arduino can also relay information about the state of the model back to the GUI. The length of blue bar on the left of Figure 38 corresponds to f_v in the model, while the blue bar on the right corresponds to f_1. On the physical model, these values correspond to the rate of animation of both LED strips. The size of the circle between the blue bars represents V_{LV}, the volume of the left ventricle. Due to limitations about the accuracy with which we could control the pump, the volume of the balloon does not perfectly reflect the true value of V_{LV} in the model. By contrast, the radius of the graphic on-screen can be set to be exactly proportional to the value of V_{LV}.

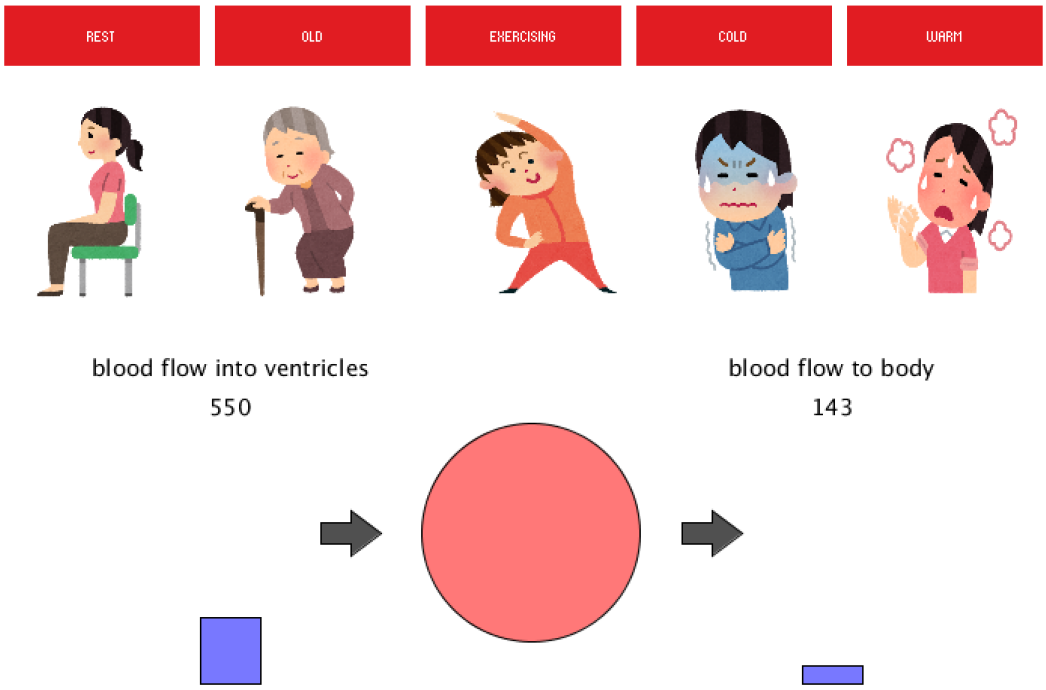

Figure 38: The GUI to enable five preprogrammed scenarios

Appendix D Testing Survey Results

After testing a near-complete prototype of ARuko at Kinkaku-ji on September 26$^{\text{th}}$, we had each test subject fill out a simple survey with their thoughts on the app. The full results are listed below.

I enjoyed the audio tour	The image gallery added to the audio tour	I found the AR translation feature helpful	I found the AR map overlay helpful	The AR features were easy to use	Do you have any other thoughts on how to improve the experience?
Strongly agree	Agree	Strongly agree	Strongly agree	Agree	I thought the idea about a sound playing before the audio starts is a really good idea!
Agree	Agree	Agree	Neutral	Neutral	Add a audio queue when a new track starts. Make the AR buttons bigger/easier to click.
Agree	Agree	Strongly agree	Neutral	Agree	Increasing GPS accuracy of course. Adding a tracking "you are here" marker to the route.
Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Typo in the text on 4/11 towards the end
Agree	Agree	Strongly agree	Disagree	Neutral	Zoom in ar mode

Table 1: Results of final field test survey

Appendix E NOxAR Assignment

On Monday, September 9th, Atticus approached us about an additional project he wanted us to tackle. He wanted us to adapt Kyoto VR's NOxAR app to use ground plane detection. Currently, NOxAR is an AR app for Microsoft HoloLens that uses image targets with Vuforia. When viewing the correct image target, the app projects a 3D model of a noh theater[11] performer dancing to background music [75]. Atticus wanted to have the app use ground plane detection in order to display the noh performer so that it could be displayed anywhere. Additionally, he wanted us to scale up the model so that when displayed in AR, the performer would appear life-sized.

For this project, Atticus sent us the original Unity project files for NOxAR. We then had to figure out how to turn the original project into a ground plane AR app. At first we tried importing the assets for the noh performer into a new Unity project. We did this two reasons. First, we did not want all of the extra components of the original NOxAR project to get in the way of what we were trying to do. We also did this because the original project was made with a 2017 version of Unity, so we wanted to try to import it into the most recent version so we would not risk running into any deprecated features or functionalities. We did eventually get this method to work, but the model of the noh performer was lacking a lot of the details that the original model had. This was attributed to errors while importing the original assets, so we decided to try making changes to the original NOxAR project instead.

Per Atticus's advice, we deleted any and all unnecessary elements from the scene before proceeding. We also updated the Unity version of the project to the 2019 version we were using, and updated the version of Vuforia to the most recent version, neither of which created any problems. The process of getting the model to appear on ground plane detection instead of image target recognition was relatively easy. We simply moved the model from an image target to a simulated ground plane, added a plane detector component to the scene, and made sure the Vuforia settings were in place for ground plane detection to work well. From there, we just had to test the app a few times so we could properly scale the model to life size. On Tuesday, September 17th, we completed the NOxAR ground plane adaptation to Atticus's satisfaction and sent him the final APK file for the app.

[11] A form of Japanese theater originating in the 14th century structured around song and slow dancing [74].