Picture This!

Joseph Ayala, Anthony Meza, Sophia Zhang
Electrical and Computer Engineering, Carnegie Mellon University

Abstract— **This project involves the creation of AR Pictionary. One player will use a hardware device to create drawings in virtual space while other players view the drawings in a shared screen to guess what is being drawn. The hardware device will use an Arduino, IMU and HC-005 Bluetooth module. An Android phone will run the AR Unity Application and will be connected to a monitor to allow guessers to view the screen. Using the Arduino to collect IMU data, the hardware device will send the data through the Bluetooth module to the Andriod phone via Bluetooth. The Unity application will then collect this data, render it in virtual space, and the share the screen for the other viewers to see. While there are existing version of AR Pictionary, we aim to expand this by creating interactive and immersive 3D drawings.**

Index Terms— **Arduino, Augmented reality, Design, Bluetooth**

1 INTRODUCTION

In current times, games have began to increase in popularity as a main source of entertainment. "The video gaming industry generated 179.7 billion of revenue for the entire year, compared to 2019's 150.2 billion." [3]. Unfortunately, traditional forms of gaming tend to predominately involve a single player experience that restricts players to sitting in front of a monitor or screen. Excessive time spent sitting has been shown to have harmful effects on mental health and have effects related to insomnia in adolescents [2]. We aim to address this issue by taking a game that most people recognize and putting a spin on it to make it more appealing, engaging, and interactive for users. Our medium for this will be AR, a cutting-edge technology, which makes the digital world interactive with our physical world. For this capstone project a prototype of Pictionary, a multiplayer drawing and guessing game, will be created where AR is utilized in order to display the drawings for other players to guess.

This experience will be possible with the creation of a pen device and a phone application. The pen device will be used to draw in the AR space. By pressing down a button, the pen will record the players' movements. The pen's data will be broadcast to the phone application which will be downloaded on the personal devices of the participating players. By using the cameras on players' personal devices, they will be able to view the drawing in the AR space that is also overlaid onto the physical world. They will be able to move around the drawing with their device's cameras to see it from different angles or zoom in on certain parts of the drawing in real time by walking around in physical space.

2 USE-CASE REQUIREMENTS

The use cases of our project can be generalized into a few different categories: the game should be representative of the original Pictionary, lines should resemble the motion of the user's hand movements (line accuracy), and the hardware should be robust and easy to use.

2.1 Game Requirements

The basic game Pictionary requires a player who is assigned as the "drawer" and at least two other players that will be assigned as "guessers". The drawer will be given a random word which they will need to represent in the form of a drawing. While the drawer is creating their drawing, the other guessers will be able to see the progress that the drawer is making and attempt to guess what the random word drawer is trying to represent. The first guesser that is able to figure out what the word is wins the round. After this, players will alternate as guessers and drawers for a few rounds until satisfied.

Due to the competitive aspect of this game, we need at least a drawer and two guessers participating at the same time. Thus our game will need to be able to support at least these 3 players. Additionally to emulate the real-time viewing of the drawing while its being made, we want it to take less than 1 second for a drawer's hand motion to translate into a line render on Unity.

2.2 Line accuracy

The lines that are rendered in virtual space are loosely derived from the movements that the drawer player does. Since the drawer is gesturing in space and does not produce a visible representation, exact accuracy of the movement of their hand is not necessary.

However the line will need to have two aspects correct: line direction and length. What this means is if the drawer player is moving their hand in an upward direction, the line produced should also be vertical. If the player moves their hand side-to-side, the line produced should also be horizontal. Additionally, the player should be able to move their hand an amount and be able to produce a line length that scales with the time their hand has been moving.

2.3 Hardware efficiency

The hardware pen device that is being created will be held and moved through the air in order to create lines. As such, this device should be light weight (less than 0.5 lb or weighing around the same as an average smart phone) and can fit comfortably in someone's hand (6 x 9 x 3 in). To keep the game accessible to everyone, we want our hardware components and cost of implementing the game to be less than $50 dollars.

Previously we aimed to attach a battery case to it and power the components on our pen with a battery. We hoped that we could attain a battery life of around 4 hours. However, after integration and finalizing our pen, we have determined that using a battery pack would be difficult to power everything. More details are provided in 4.3

Additionally, the hardware pen should be robust and be able to transmit the necessary data to reduce the game overhead. This point can be reiterated from Section 2.1 where when a player moves their hand to draw a line, that line should be rendered in 1 second or less.

3 ARCHITECTURE AND PRINCIPLE OF OPERATION

On a high level, the framework for our system will consist of N players (where $N \geq 3$): 1 drawer player and $N-1$ guesser players. The drawer player will be the one responsible for drawing the image and viewing the image. They will be holding the pen device and perform gestures in order to "draw" in physical space once game play begins. The guesser players will be responsible for watching what the drawer is currently trying to make, and guessing what they are attempting to create. While the drawer is drawing, this data will be captured by the pen device and sent to the drawer's personal device where the Unity App will be able to render lines they are creating in physical space to the AR space. Originally, this AR space was planned to be visible to all players through the cameras on their own personal devices. Due to time constraints, this was updated so now the drawer's phone screen will be shared on a monitor, where the guessers will be able to see what is being drawn and make guesses about what the drawer is trying to create. Fig. 1 is a diagram overview that demonstrates data flow and general layout.

Looking at Fig. 1 with a little more detail, the *Drawing Mechanism* will be our pen device, which will contain 5 hardware components: a Arduino, IMU, HC-005 Bluetooth module, two buttons, and two LEDs. When button is pressed, the N will start recording the IMU data that corresponds to the pen device's movment, which will send the data through the Bluetooth module. The LED will light up as well, providing visual feedback for all players, that the device is currently drawing in AR space.

The drawer will have access to the pen device and rotate it in air in physical space in order to "draw". These gestures will be captured by the pen device the moment game play begins, and store it on the device.

This stored data will then be sent to the game app on the drawer's personal device. Using this data, the Unity App will be able to render lines that are overlaid onto the Real World through AR. Fig. 1 is a picture overview of this data flow and general layout.

The drawer will need to have the Unity Android Application downloaded to their devices in order to be able to view what is being drawn. The Unity software will be using the Android Bluetooth plugin package in order to record data from the Arduino itself, which will be sent over via Bluetooth by the drawing mechanism through the HC-005 during game play. This data will be processed, and using ARCore packages within Unity, will be able to process and render lines the drawer is trying to produce in the AR Space. By viewing the screen replicated from the camera, users will be able to view the rendered lines that the drawer has created. Other packages that come with AR Core will be used in order to make sure that the drawing is tied to a specific location in the real world. The main packaged that assists with this would be AR Foundation, which allows the creation of many varying types of AR GameObjects. Through this, an anchor point can be created at which AR LineRenderer Objects can be attached to (the drawings the drawer creates).

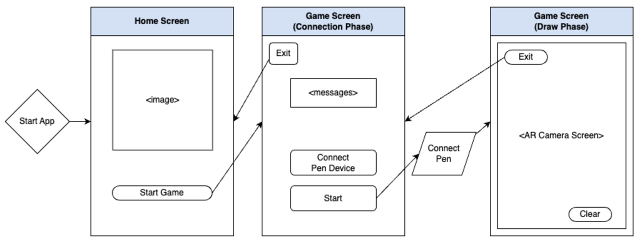


Figure 2: Game UI and how the drawer will redirected from page to page

In Fig. 2, we are able to get a better idea of how the Unity Application will function in order to be intuitive enough for player to use, but also simple enough to create for our scope of project.

The drawer will be responsible for running the application on a handheld device, which will be connected to a monitor where the guessers can view what is happening.

When they start the application, there will be the option to be able to connect to the pen device in order to use the hardware in the game. To connect the user will first connect via Bluetooth in the Android settings application (just like for any other Bluetooth mechanism). Once the application is opened, the user should be able to connect to the pen device by clicking the connect button. Once they start the game the pen will completely connect, and allow the game to start.

During gameplay, a drawer can touch the screen for where to start drawing (creating an anchor point and new

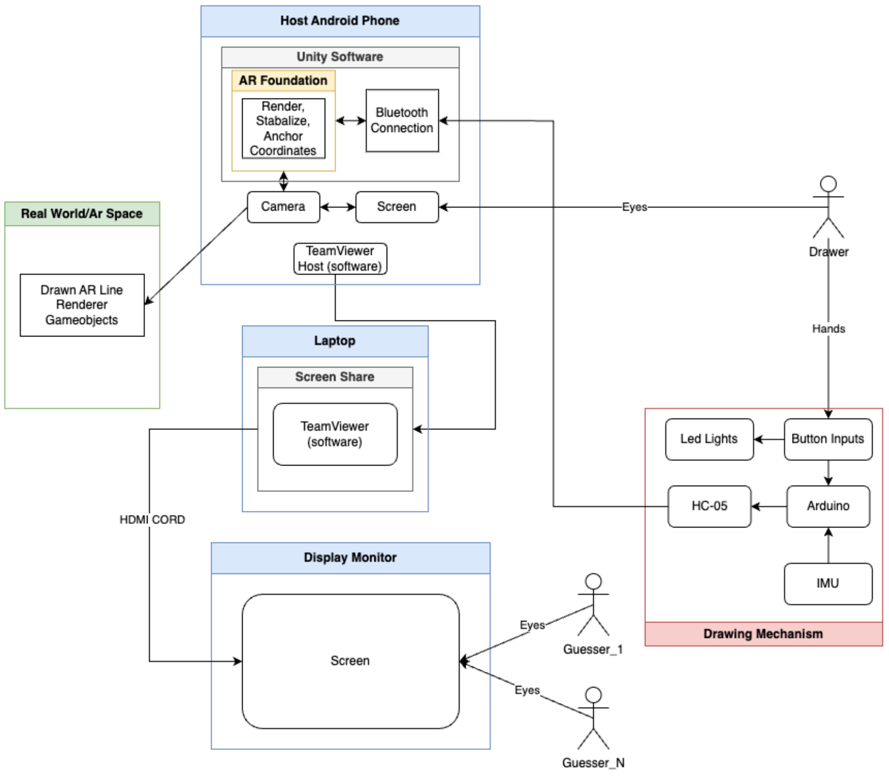


Figure 1: An overview of the game structure and the components. Red blocks represent any physical devices. Blue represents any software components. Green represents the virtual space where AR objects are overlaid onto the real world.

LineRenderer), and then move the pen device in order to draw in 3D space. If they want to restart what they are drawing, they can click the clear button to delete what they have drawn.

After playing a round, the players will be able to exit the game screen at any time and go to the connection phase in order to switch drawers, swapping the phone and the pen device in the process. If any error connections occur while in the draw phase, players will be redirected to the Connection phase to try and reconnect.

This system is simple enough for most players to be able to play the game. If time permits, more complexity could be added by adding more features like adding interactivity with guessers of some sort. For the current scope and timeline, this basic design will suffice.

.

4 DESIGN REQUIREMENTS

The Design Requirements for our game of AR Pictionary are built off of our Use-Case requirements and can once again be split into the same overarching categories: the game should be representative of the original Pictionary, lines should resemble the motion of the user's hand movements (line accuracy), and the hardware should be robust and easy to use.

4.1 Game Requirements

Before, we aimed to have 3 devices that were capable of seeing the same virtual space. However, due to time constraints from unprecedented problems in our hardware and software integration, we decided that we did not have enough time for the networking and multiple device Cloud Anchoring required for this. Therefore we decided to scrap the networking and transition to screen sharing, where our game would be running on a host Android device which is then screen shared to a separate laptop, which is then displayed on a bigger monitor through HDMI. Further details on this are elaborated on in the software subsection of our design tradeoffs. This tradeoff still satisfies our first requirement of our App needing to be viewable by multiple users.

We also aim to have real time line drawing. The second requirement implies that the overall end-to-end latency of communication needs to be kept to a minimum. The defined metric of at most 5 seconds from hand movement to drawn line will need to consider the combined latency of a number of different components. This includes time taken for: 1) hardware data collection on the drawing mechanism, 2) transmission of the data from the drawing device to the Unity Software, 3) calculation of linear displacement based on the data received, and 4) rendering that the software will perform in order to create the line all in the AR space, which will be visible for all players. Calculation and rendering of the line latency will be described in more detail in 4.2. Hardware data collection and transmission latency will be described in more detail in 4.3.

4.2 Line accuracy

As stated in 2.2, the line's direction needs to correlate with the movement of the drawer's hand, as well as the length of time the user is moving their hand.

To assess the correct direction of the line, we will base this metric off of an orientation test. More details on this metric will be discussed in Section 7.

We were originally using positional data in order to determine where the points of the line were. We originally were aiming for a drawn line length to be correct within 10 cm of our test line and our reference line to account for the noise from the IMU. However, this method was in practice too noisy for our purposes. A discussion of these details can be found in Section 5.1. Because of this, we decided to focus more on the orientation/angle that the device is being held at to determine the angle. Once again, more details of how this will be assessed will be discussed in Section 7.

The latency of line calculation and render should be less than 3 seconds in order to hit our metric of 5 seconds between hand movement to drawn line to leave sufficient overhead for data collection on the IMU and transmission of this data.

4.3 Hardware efficiency

Our pen should be light weight (less than a 1 lb) and can fit comfortably in someone's hand (6 x 9 x 3 in) as stated in 2.3.

We were originally aiming for a 4 hour battery life. We would need to find a battery that will be able to supply 4 hours of power to our microprocessor and IMU, which are the components that that will be drawing the most current. Equation (1) describes how our battery life t, current draw A, and battery capacity C are related to each other. N is the total number of components drawing current and A_i is the current component i is drawing.

$$t = \frac{C}{A} = \frac{C}{\sum_{i=0}^{N} A_i} \tag{1}$$

However, after integrating and testing everything, we would need 2 3.7 V batteries to have sufficient voltage for our system. However, since everything on our board runs uses either 3 V or 5 V, we would need a buck converter to step down this voltage to ensure our components are not destroyed in the process. This system would have taken up too much space on our board and ultimately would have been extremely power inefficient. Because of this, we are instead using an long USB 2.0 cable to power our Arduino and components. This effectively nullifies this previous use case since there will be a ready supply of power.

Additionally, the hardware pen should be robust and be able to transmit the data with low latency. This point can be reiterated from 2.1 where when a player moves their hand to draw a line, that line should be rendered in 5 seconds or less. Thus we are aiming for a communication latency between our pen and the Unity App to be less than 200 ms, as this is the typical latency for Wi-Fi and bluetooth communication. We will additionally consider an extra 100 ms overhead needed for collecting the data from the IMU.

5 DESIGN TRADE STUDIES

5.1 Hardware Calibration Methods

Calculating relative position with solely an IMU is difficult to do so in an accurate method. An IMU will only be able to sample data at a constant rate and if there are changes in frequency that happen faster than this sampling rate, they will be missed and therefore not incorporated into the output of the IMU. This makes it difficult to obtain an IMU with a high enough sampling rate and low enough error to get accurate enough metrics for line displacement. These errors can accumulate over time and extended duration of IMU usage will result in data with high errors.

There are a few methods that can be used to address this problem that will be further described in the following sections.

5.1.1 Multiple IMUs

One method of correcting for IMU drift and accumulated error over time is to use multiple IMUs that can correct each other through a process called multi-sensor data fusion. Sensor fusion combines sensor data or data derived from different sources that will produce data that has less uncertainty than would be possible when these sources were used individually. The goal is that combining these sensors will result in data that is more accurate.

One of our concerns with this approach is that it will likely add non-trivial overhead to our line calculations. We will have to implement our own data combination algorithm (like Brooks–Iyengar algorithm) on either our Arduino Nano or as a C# script in Unity. Due to us having to implement our own version of this algorithm, there may not be enough time to fully optimize this to meet our latency metric

Another small concern is that we want to keep our pen as small and as low-cost as possible. Having to add more IMUs will increase the size of it as well as increase the cost of our device as well.

Overall sensor fusion is a feasible method of providing more reliability of our data. However, as discussed in Section 8.4.2, our diagnostic tests have showed promising signs that a single IMU falls within our error range. If more issues emerge with extended use of the IMU, sensor fusion is something we can include to improve our data reliability.

5.1.2 Software Assistance through CV

Another potential method is to use CV in order to detect the location of the pen on camera. This could be accomplished by tracking the location of a lit LED. CV would be able to more easily determine the location of the pen in the air and would have to rely less on noisy hardware for this information.

However, CV will only be able to provide 2D coordinates on a screen. This goes against our original proposal idea of using AR to draw 3D lines that players can rotate around and interact with. Additionally, there were concerns that adding CV would also greatly increase our latency for line calculation and computation.

The time and energy it would also take to implement CV falls outside the scope of the project. There is likely not enough time to implement CV and reach the desired minimum viable product.

5.1.3 Stationary Device as Reference Point

Another general method that falls in line with sensor fusion as described in Section 5.1.1 is using a stationary device as a reference point. This device would be placed on the ground to the side and would serve as a fixed reference point between the pen and the IMU.

An obvious concern is the method in which this device would be able to determine distance. One way is to use Wi-Fi in order to determine the distance between our bluetooth pen and Wi-Fi device. Once again, however, the issues with this solution approach are similar to the ones defined in Section 5.1.1. This external Wi-Fi device will likely be even less accurate than the IMU and would do little to correct the actual distances. Another large concern is that this Wi-Fi device will be extremely expensive which will break our proposed requirement of an affordable cost.

5.2 Hardware Communication Protocol

Previously, we had planned to use Wi-Fi in order to send our IMU data from our microprocessor to the Unity App. This would have been accomplished with support from an Unity library called Uduino. The method was partially successful as we were able to send our line data from our Arduino to Unity's desktop monitor. However, once we tried to port our app to our Android phones, we were unable to establish a connection. After trying our best to debug this connection, we were unable to come up with a solution so we had scrapped this idea in favor of a different Unity package and method of communication.

Figure 3 demonstrates an alternate method that was previously considered for transferring data from our ESP8266 module to our Unity App. Due to an inability for Unity to directly communicate wirelessly to our hardware, a Python net socket could have functioned as an intermediate in order to send data over.

This alternate approach would allow us to have more fine-grained control over how data is transferred between our pen and our Unity app. It would also allow us to potentially reduce the latency of line calculation by allowing us to use built-in Python libraries to perform our double integration.

However, as demonstrated in Fig. 4, there would have been additional latency added due to a communication protocol. First, data would be need to be sent over a Wi-Fi channel twice. This communication protocol would also have required our Unity app to constantly be polling the Wi-Fi channel in order to detect an incoming packet reception signal. Only once the hardware and software performed a handshake and agreed to exchange data, then they would have been able to send/receive data. This is necessary to ensure both ends are ready to send and receive data as without it, the line data could be incorrectly dropped, which would have drastically decreased the quality of the game. However, this method would have incurred additional latency as well as draw more power from the devices since they would have been constantly busy polling in the background.

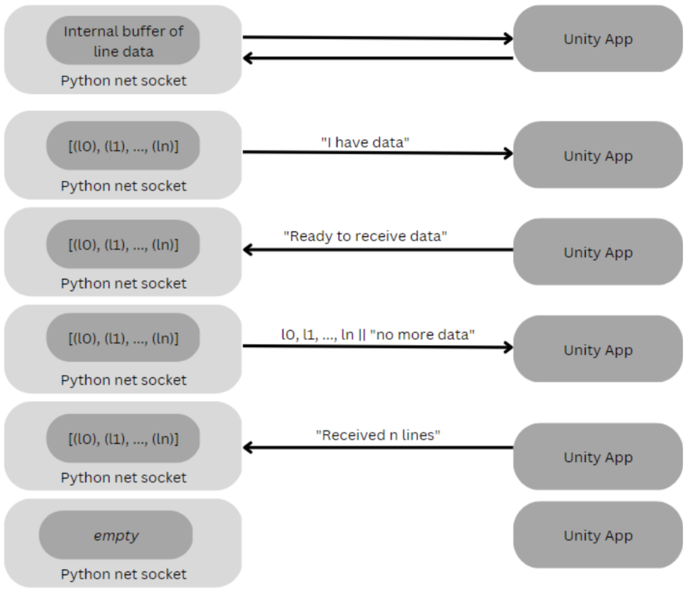


Figure 4: Communication protocol between the Python Net Socket and Unity on how to transfer data

5.3 Software Selection

Unity was an obvious first choice in terms of software development for us. Not only do we have previous experience using it, Unity is also already well integrated with several other packages such as AR Foundation in general, AR Core for Android devices or Apple AR Kit. Additionally, Unity has a very large and an active community, so there are many built-ins, guides, and documentation available to streamline this process.

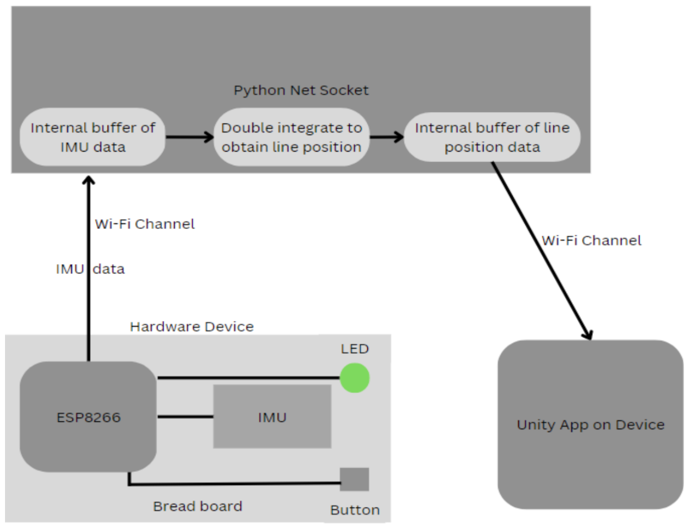


Figure 3: Schematic of an alternate hardware design that involves using a Python Net socket to assist in data transfer between ESP8266 module and our Unity App

5.3.1 Networking + Cloud Anchoring vs Screen Sharing

In terms of software trade-offs, we originally planned to use Unity's AR Cloud Anchoring package in order to store the location of AR anchors in the cloud and share them to multiple Android devices through Unity's networking package. We were able to get the AR Cloud Anchoring to work for one anchor point on one device, but found it overly complex trying to make it work for multiple different anchor points, as well as on multiple different devices through anchor points. At this point, given the short amount of time left before the final demo, we decided it would be unfeasible to not only get all of this working separately but also integrated into the rest of the pipeline (hardware communication + software line algorithms) in time.

Therefore we decided to scrap the cloud anchoring and transition to screen sharing, where our game would be running on a host Android device which is then screen shared to a separate laptop, which is then screen shared to a bigger monitor through HDMI. This is done through an application called TeamViewer, which allows one device to be used as a host and other devices to hook into this device and view what is being displayed on it. TeamViewer Host is installed on the host Android phone that has our game, while TeamViewer is installed on the laptop that will be screen shared to.

This more simple implementation still allows all users to effectively be able to see what is being drawn in the real world and guess in real-time while being feasible to integrate in the short amount of time available to us. The only drawback would be that this method is less interactive for users, as the users would only see the drawer's point of view and the drawer would simply be changed to whoever holds the host device and pen.

5.4 Build Device Selection

For our application, since we have the intention making our game easily accessible for the average person, we needed to make considerations about what devices it should be able to build on. It was originally considered that we could develop an application that could be run on any device, including phones and laptops. The idea behind this approach would mean that the application would be accessible to anyone anywhere. This idea was eventually scrapped, instead deciding to strictly build for phones, the deciding factor being that generating builds for accessibility on both laptop and phone screens would add an unnecessary amount of additional complexity that would divert focus from the more challenging components of the project as a whole. Additionally, phones would be much more practical and allow for ease of interaction during a game play session, since it would make it easier for guesser players to move around and examine the drawings created by drawers at different angles.

We also had to consider which OS device we would want to try developing our project for. IOS devices were considered due to the greater number of IOS devices that we accessible, but IOS tends to be more complicated to build on since it would require everyone in our group to own Apple products. We instead made the decision to code for Android devices, mainly because they are much more developer friendly. This did mean we would need to purchase more Android devices for testing purposes, but Android devices are usually cheaper than Apple devices as well.

6 SYSTEM IMPLEMENTATION

6.1 Pen

Fig. 5 describes the layout of the pen device. The pen will consist of an Arduino Nano, 2 green LEDs, 2 buttons.

a HC-005 bluetooth module, and an IMU. The Arduino Nano will be powered via a USB cable that will be able to supply 2 Ah. The Arduino Nano will also be able to power the IMU through its 3.3 V pin and power the HC-005 through its 5V pin. The Arduino will be connected to the buttons and LEDs to receive inputs and provide visual feedback to the user.

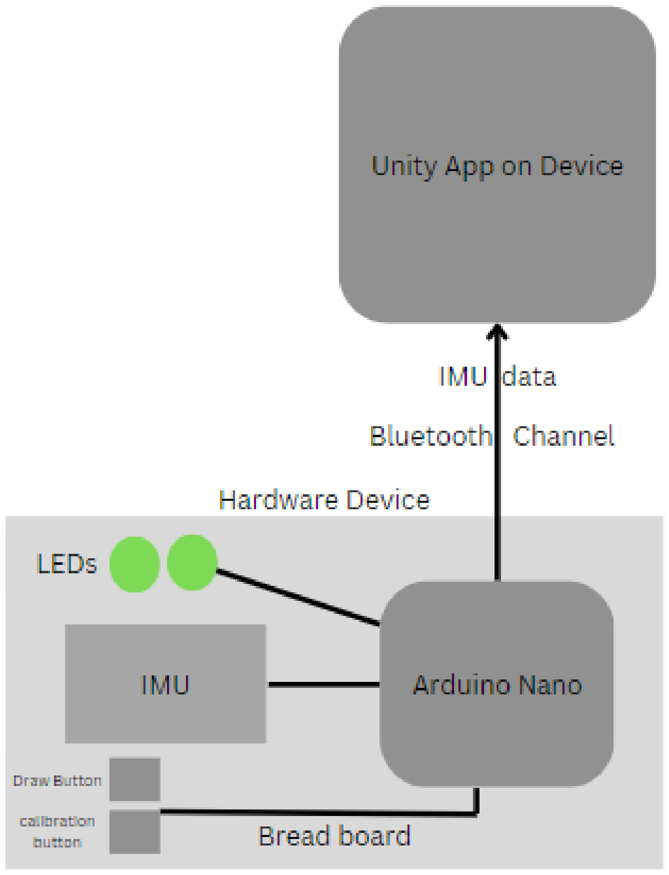


Figure 5: Schematic of the pen

The drawer will press and hold one of the buttons (as referenced in Fig. 6) to indicate that they are in the process of drawing a line. One of the LEDs will also light up to provide visual feedback for the drawer. The IMU will determine the user's hand orientation and draw a line in the corresponding direction. While the user's hand is in that orientation, a line in that direction will be drawn. The steeper the angle the user's hand is at, the greater the distance of the line will be. This data will be sent to the the Unity App on all players' phone through Wi-Fi.

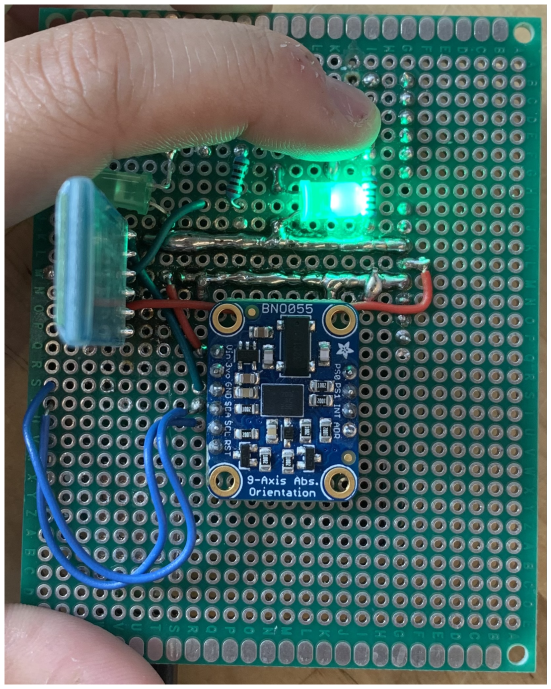


Figure 6: The pen device in drawing mode

When the drawer presses the other button, the Arduino will enter calibration mode as shown in Fig. 7. In this mode, the user will ideally hold the pen in a constant position while the Arduino keeps track of all the accelerations that are impacting it while stationary. These statistics will be used by the Arduino when calculating line position to get rid of errors due to gravity or the way the user is holding the pen. It will also reset the previously collected distance data.

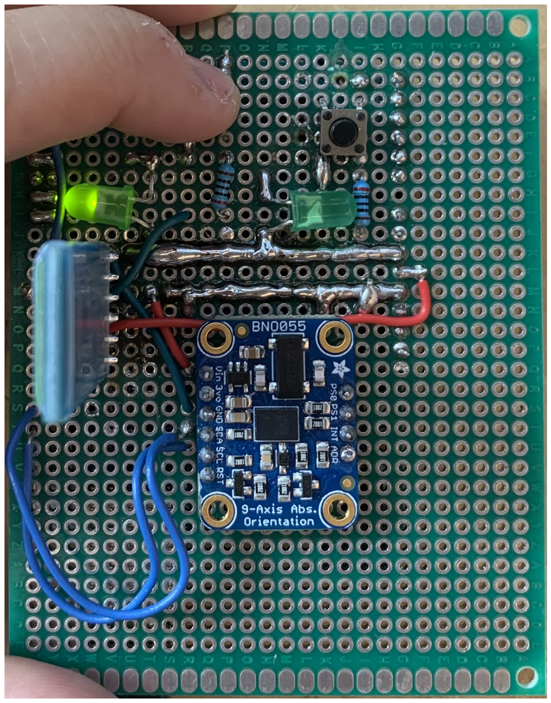


Figure 7: The pen device in calibration mode

In order to reduce the amount of IMU drift accumulated over time that would greatly reduce the accuracy of the lines and hinder the game experience, the player must start in a calibration phase. In this phase, they will will hold the pen in their dominant hand straight down, with the IMU pointed at the ground. Once they assume this position, they must press the calibration button and hold this this position for 2 seconds. After they finish calibrating, the player is free to draw 2 additional lines. Once these 2 lines are drawn, the player must return to this calibration phase. While the pen will still work after these two lines have been drawn, the quality of the drawn lines will be diminished.

The Arduino will collect IMU data once every 0.01 seconds to obtain an the accelerometer, gyroscope, and magnetometer data at this point of time. From this data, we perform sensor fusion to calculate the pitch and roll of the pen. Using this data, we then double integrate the pitch and roll data by multiplying it by the change in time twice. This data will be used in order to determine the location and shape of the line. This displacement data will be sent to the Unity App.

6.2 Unity App

The Unity app will consist of three scenes as already seen in Figure 2. Each scene will have some sort of scene manager GameObject that has functions responsible for loading different game scenes. These functions in each game scenes will be associated with certain buttons, that once pressed, will load the appropriate game scenes. The game loading screen will simply have the title screen, and a button that leads the Connection Scene. In the connection scene, the drawer will be able to connect the hardware device to the Unity App. The user will click the connect button to establish a connection with the hardware device, and then they are able to click the start button to load the Drawing Scene once they are ready. In the game scene the drawer is able to play the game as described. If players want to exit, an exit button will be present to load each previous game scene they have opened.

Figure 8 describes the general flow of data within our Unity App for the game Drawing Scene. Once ported to each players' phone, the Unity app, ported to Android through the use of the ARCore package, will directly interact with our Arduino through a Bluetooth channel.

Positional data from the pen will be put into an data-holding Unity GameObject within the game scene which will then be read through a C# script. This script that directly updates and renders the AR drawing will, at the start of a round of the game, scan the environment and create AR plane GameObjects on the surrounding surfaces (takes a second or two). Then when the player taps on a location on the screen, the script will create an AR anchor at that point, using the touch position and identity rotation to create a Pose Struct, and instantiate an initial AR LineRenderer GameObject at that anchor point, through the use of the AR Foundation Package. This anchor point determines the absolute location, relative to the real world, of the drawing that the drawer will draw.

After this, the script will always be in a busy polling state, waiting for the drawer to hold down the button to draw, at which the script then pulls the locational data from the Arduino, placed in the data-holding GameObject within the scene, and forms it into a Vector3 position. The script then adds a new point to the LineRenderer whenever the distance between the previous point and the current pen location is more than 0.001m, calculated by adding the Vector3 pen location to the anchor point's Vector3 location. This continues until the drawer releases the button. The drawer at this point can then either decide to hold down the button to continue drawing, tap the screen to create a new anchor point at which to draw, or clear all drawn lines by tapping the clear lines button.

If the drawer decides to touch the screen, a new anchor point will then be created at this location in space, and corresponding LineRenderer. This signifies a new drawing, or a separate continuous line, which will be drawn and anchored in the new space where the user started drawing. These anchors will be stored in a list, allowing each separate line to be rendered and held in the exact location where they were drawn in space, instead of following the drawer's phone around as they move around. All of this is able to be seen by the other players in the game through the TeamViewer app screen sharing the drawer's point of view onto a bigger monitor.

7 TEST & VALIDATION

7.1 Line Accuracy Tests

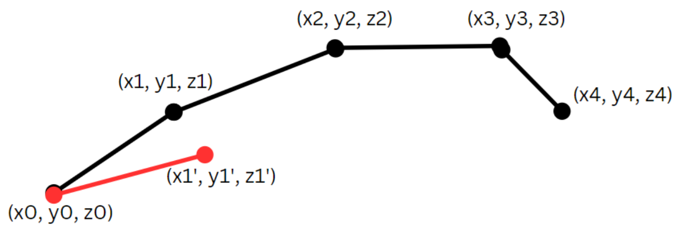


Figure 9: An example of a reference line (in black) with defined points in 3D space as well as an test line drawn by a user(in red)

We will assess line accuracy is what we define as a distance test. In this test, we generate a few **reference lines** that will consist of a a few points connected by a straight line. Figure 9 gives an example of such a line with arbitrary points in space.

Depending on exact metric we are trying to test, we can create this 3D line in two ways. If testing on a 2D plane, we can print out this reference line onto a sheet of paper. Otherwise, we can use string to represent a line in

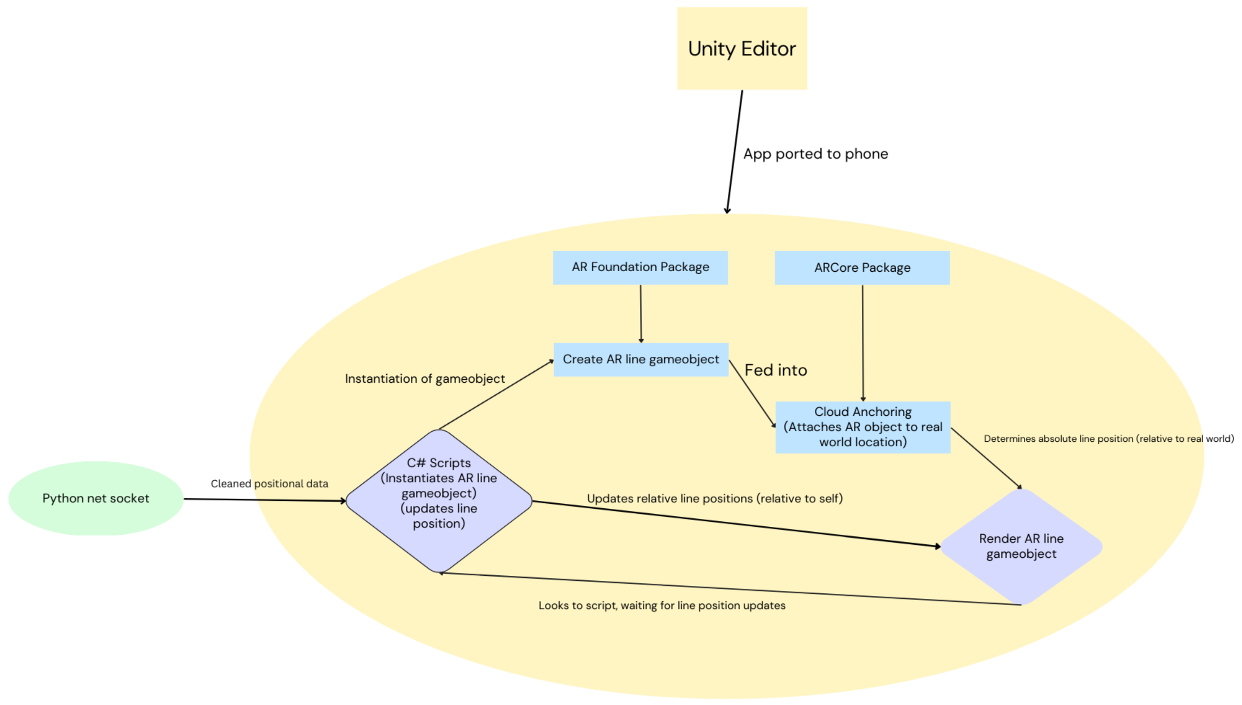

Unity Editor
App ported to phone
AR Foundation Package
ARCore Package
Create AR line gameobject
Fed into
Instantiation of gameobject
Cloud Anchoring (Attaches AR object to real world location)
Determines absolute line position (relative to real world)
Python net socket
Cleaned positional data
C# Scripts (Instantiates AR line gameobject) (updates line position)
Updates relative line positions (relative to self)
Render AR line gameobject
Looks to script, waiting for line position updates

Figure 8: Flow and transfer of data within Unity

3D space. We will then use our pen to trace over this reference line. They will draw a line from one point to another point. This drawn line will also be called a **test line**.

In reference to Figure 9, a user for example, will draw a line from point $(x0, y0, z0)$ to point $(x1, y1, z1)$. Afterwards they will draw a line from point$(x1, y1, z1)$ to point $(x2, y2, z2)$ until the reference line is finished being traced over.

We will calculate the angle between the reference line and the test as demonstrated in Equation 2. $v_r = (x_1 - x_0, y_1 - y_0, z_1 - z_0)$ is the expected vector for the reference line while $v_t = (x_1' - x_0, y_1' - y_0, z_1' - z_0)$ is the vector of our drawn line. We can determine the angle between these two vectors by solving for θ in Equation 2.

$$\cos(\theta) = \frac{v_r \cdot v_t}{|v_r||v_t|} \tag{2}$$

If the angle θ between our two lines is $\leq 10°$, our AR lines are accurate enough. With this new metric, we were able to achieve an angle deviance of 8°. With more time, we could have refined our testing a bit more and tried to fine tune the parameters and filtering more.

Fig. 10 shows an example of a line that was drawn.

7.2 Pen Size

We will first construct our device onto a bread board and try to compact it as much as possible. After development has been finished, we will solder everything to a PCB.

Our entire system fits on a 7 cm by 9 cm PCB and weights roughly 6 oz. Thus we have fulfilled our metric.

7.3 Battery Life

We originally were planning on running a test script that mimics automation of drawing the line by constantly setting the button input signal to high and low. We planned on running this script on loop until the battery dies and see how long it takes for the battery. However, due to reasons discussed in 2.3, this is no longer a considered metric.

7.4 Latency Tests

We plan on conducting a test where we draw one line. When the button is lifted, we start a timer and measure how long it take for this gestured line to be displayed in Virtual space.

One test set will will involve drawing 5 lines and recording how long it takes for those lines to be rendered completely in AR space. We aimed for an end to end latency of 1 second and a hardware latency of 150 ms. We achieved this metric with an end to end latency of 10 s and hardware latency of 8 ms.

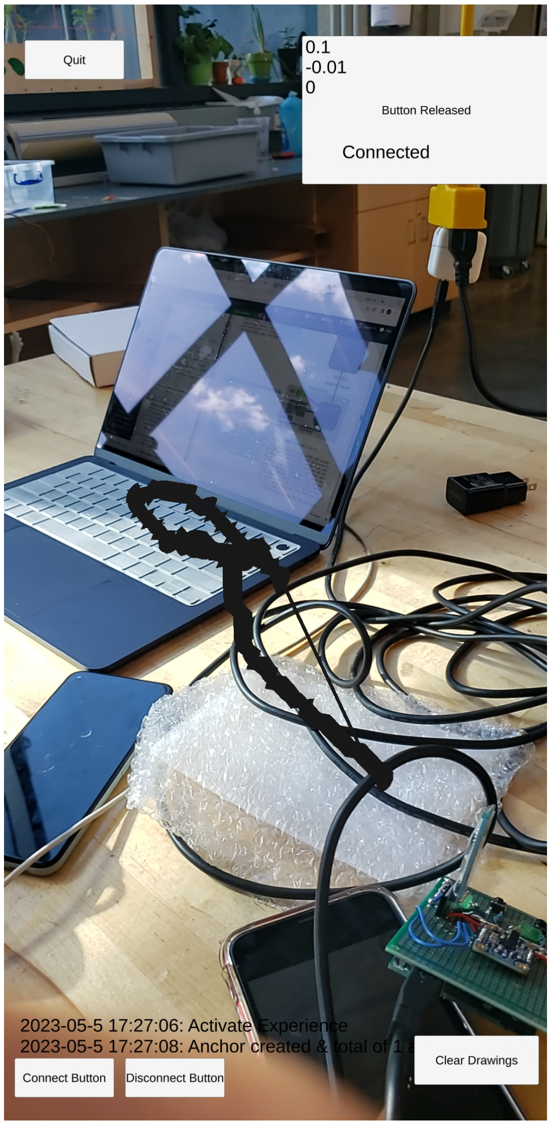

Quit
0.1
-0.01
0
Button Released
Connected
2023-05-5 17:27:06: Activate Experience
2023-05-5 17:27:08: Anchor created & total of 1
Clear Drawings
Connect Button
Disconnect Button

Figure 10: An example of a "lasso" drawn in black with the pen device.

7.5 User Satisfaction for Drawing Test

We plan on also running additional user tests in order to measure the "easy to use" metrics we have intended to establish for our game. This test will specifically focus on how easy the hardware pen device is to use. We will ask a player to attempt to create a drawing based on the word using the pen. After they complete this task and see their finished product. We will then conduct a short survey, asking for a rating on a scale of 1 to 10, on how accurate they felt the pen was at drawing their intended drawing. A goal would be to have at least 80% satisfaction for the majority of our test.

Unfortunately, with our previous scheme of trying to calculate positional data, users understandably found it frustrating to use. We achieved a metric of 2 out of 10 users saying that they were satisfied with the pen's functionality and an average rating of 1.9.

Due to this feedback, we were encourage to come up with a different method of drawing lines.

8 PROJECT MANAGEMENT

8.1 Schedule

The schedule is shown in Fig. 12.

The hardware is the main focus of our work in the first half of the course, as it is our critical path. It is imperative that it gets done in order to have usable data for the software side of things. while the hardware is in development, the initial software pieces that can be done simultaneously during the first half of the course.

The latter half of the course, after spring break, are focused on finishing up the software, finalizing the hardware, and then integrating the software with the hardware. We expect the integration to be the most troublesome area, so we leave ample time for that. The very last weeks are dedicated to testing and validation of our project.

8.2 Team Member Responsibilities

Sophia will be mainly focused on the hardware aspects of the project. This includes stuff like selecting and buying the components, designing the hardware pen, and then building, calibrating and testing the hardware.

Joseph will be mainly focused on the software aspects of the project. This includes stuff like setting up and porting the AR Unity project to the android phone, developing the main line drawing scene and line drawing functionality, and making sure all AR functionality and the line drawing works for all users as expected.

Anthony will help Sophia work on getting the hardware functional during the initial stages of the project and then shift to helping Joseph get the software functional in the latter stages of the project. This includes helping design and test the hardware, as well as designing and implementing the game application UI and functionality, and helping debug kinks that happen in the AR space and between devices.

We will all work together on integrating and testing the project as a whole.

8.3 Bill of Materials and Budget

Table 1 contains a summary of all purchased and obtained goods. Hardware pen components are at the top of the table and software and testing components are located at the bottom.

Some components like buttons and LEDs can be trivially obtained from previous projects and are not counted in the cost. Other components such as a borrowed phone also will not cost anything out of pocket.

Some of the hardware materials are for the purposes of backups and testing. Our actual device will only require one IMU. However, we wanted to purchase backups and

Table 1: Bill of materials

Description	Manufacturer	Quantity	Cost @	Total
ESP8266 NodeMCU	HiLetgo	3	$16.39	$49.17
Arduino	Arduino	1	$0	$0
LSM6DSO32 6-DoF IMU	Adafruit	3	$12.50	$37.50
IMU Fusion Breakout - BNO055	Adafruit	3	$34.95	$104.85
Button	NA	4	$0.50	$2
LED	NA	1	$0	$0
Breadboard	NA	1	$0	$0
HC-005 Bluetooth Module	NA	3	$10.39	$31.17
PCBs	NA	1	$13.97	$13.97
Uduino	Unity	1	$15	$15
Arduino Bluetooth Plugin	Unity	1	$15	$15
Samsung A10e	Samsung	1	$150	$150.00
Android Phone	NA	1	$0	$0
				$418.66

Table 2: A more accurate assessment of spending on materials for finalized product

Description	Manufacturer	Quantity	Cost @	Total
Arduino	Arduino	1	$6.64	$6.64
IMU Fusion Breakout - BNO055	Adafruit	1	$34.95	$34.95
Button	NA	2	$0.50	$1
LED	NA	2	$0.50	$0.50
PCB	NA	1	$5.0	$5.0
Arduino Bluetooth Plugin	Unity	1	$15	$15
Samsung A10e	Samsung	1	$150	$150.00
Android Phone	NA	1	$0	$0
				$213.09

different types of IMUs to test and assess which ones best suit our needs.

Table 2 contains a more accurate spending summary. If you take out the purchased phone for the purpose of testing, the cost of our pen itself is \$63.09

8.4 Risk Mitigation

8.4.1 Hardware Reliability

A large portion of our project depends on the hardware pen working, so we focused on the hardware right from the very start. Once we realized this task seemed infeasible for just one person to work on, we reorganized division of labor so more people are working on the hardware components.

At the same time, the software that can be built without hardware input is being developed simultaneously.

8.4.2 IMU Noise

A large pressing and ongoing concern for our hardware was the quality of our IMU signals and whether or not they would be too noisy for the purposes of our project. We did our best to mitigate this risk by thoroughly researching IMUs to try to find ones that were of high enough quality but cheap enough to meet our cost use cases.

Unfortunately, we quickly discovered that even if the initial acceleration data on IMUs seems to be sufficient, the small error that it has will quickly build with every operation done with it. Because of this, every quaternion rotation and double integration to get position done on the points would quickly grow more and more noisy until the positional was not extremely meaningful for our purposes.

To address this, we ended up relying on the slightly more accurate orientation data. By combining the acceleration data, gyroscope, and magnetometer data provided on our IMU, we were able to perform sensor fusion similar to what is described in 5.1.1 to get more stable rotational data. We have chosen to use our pen in more of a joystick fashion, where you can rotate it and hold it in that position for an extended period of time to change the distance.

Being aware of potential problems from the beginning and coming up with backups has proven to be key in addressing the shortfalls of our IMU methods.

8.4.3 Hardware-Software Communication

One of the biggest issues that we discovered was the transfer of our IMU data to the Unity app on our phone. Some of the methods that we were debating on are discussed in 5. We initially were set on two possible methods: write our own Python net socket or use the built in Unity package Uduino.

After some feedback and comments, we decided this method would cause unnecessary latency as data would need to be sent twice: once to our netsocket and once again to our Unity App. Additionally, it would require additional equipment (i.e. a laptop), that would make our device less user friendly.

We originally planned on using Uduino, a Unity package designed for interfacing between an Arduino and Unity through either a serial, Wi-Fi or Bluetooth connection. We successfully interfaced through Wi-Fi and a serial connection with desktop Unity and our pen. Unfortunately, we discovered that our Android devices were not compatible and data could not be sent. Since there was not any documentation nor was the author being communicative, we decided to switch to a different method.

We ended up on using a Bluetooth connection with a different Bluetooth Unity package. Not only was it easier to set up, but our latency was reduced since it was a direct connection.

8.4.4 Integration

Our biggest risk is the integration of the hardware, software, and multiple devices running concurrently. We tried to start by integrating our hardware and software as early as possible. However, difficulties inevitably arose, which greatly delayed other aspects of our project such as running multiple concurrent devices and fine tuning our hardware. Because of this, we opted to screen share our device's screen instead. We also dedicated a lot of time and effort to focusing on the pen's data collection metrics to ensure we had a slightly working product.

9 RELATED WORK

AR only recently has begun to grow in popularity, but a few similar technologies that tried to accomplish similar goals that align with the product we are also trying to build.

Although it did not involve AR, when the Nintendo Wii [5], was released, it managed to integrate both physical activity and movement with video games. It was also able to add a new form of interaction between different players for select games. We hope our AR Pictionary game is able to embody a similar ideal, innovation and enforcing physical activity. We attempt to improve upon this concept by making an application that is easily accessible on any individuals personal device, instead of requiring someone to pay a lot of money for a new game system. Our game would also allow for larger groups to participate, since we are not restricted to 4 controllers that can be synced to the system at a time, instead (in concept) any number of people would be able to play our AR Pictionary game.

VRChat [4] was able to provide and innovative form of social interaction for users in a virtual-replacing-the-physical world context. Similarities exist in the form of a virtual context for interaction among people, but lack the real world interaction among individuals. There are many more activities to do within VRChat as well, but requires users to purchase a VR Headset, which can be expensive for some. Our design, since it is simple in nature and mostly requires personal devices, simplifies some complexities of requiring additional hardware, besides the pen that needs to be purchased.

There exists a current product on the market called Pictionary Air [1], which functions very similarly to our product. Drawings are made in virtual space with a pen like device, and they are viewed on a tablet where people can guess what another person is trying to draw. The difference though is that these drawings with the pen are made in a two dimensional space, and the drawings are only viewable on one single device at any time. Our game will instead be creating image in a three dimensional space, and also will be interactive among multiple devices, allowing for more participants.

10 ETHICS

Ethics were highly considered in the development of this product. AR technology is both innovative and controversial due to its immersion with real life.

One potential issues is that AR can be distracting by preventing people from being aware of their surroundings. If people are to focused on viewing the virtual world through their phone, it may cause them to be unable to see in the physical real world around them. Some of these dangers could include items on the floor that could cause players to trip, walls close to a player, and many more. One potential scenario could be a group of children playing this game in the front yard of their house. As they get more and more excited with the game, they slowly venture further and further away from it until they are playing in the street. Since their attention is dedicated to the game, they may not be aware of immediate dangers such as cars or road construction that may be happening at the same time. Some of these dangers can be mitigated by giving reminders to users to make sure they are aware of their surroundings while playing the game. This will allow them to stay immersed, but also reminded to stay aware.

There are also concerns that our app requires access to phone cameras, which can be seen as an invasion of privacy. We have addressed these concerns by first asking permission for camera access. While our app does view the surrounding environment through the camera, it does not record anything. In addition, in order to screen share, TeamViewer requires temporary access to the host device, which is also an invasion of privacy. To alleviate this, this connection is only possible locally, in close contact, so the owner of the device knows what is going on, in their device. Additionally, the connection can be aborted at any time by the owner of the device.

There is also a potential issue with conflicts amongst players that could occur if the games get to intense. This could be problematic, since instead of creating an enjoyable game like our user requirements intend, it might cause tension amongst players instead. This will hopefully be avoided since game play is simple and straight forward. The game itself will also make sure to maintain a positive atmosphere, to make sure every ones attitudes are positive during game play.

11 SUMMARY

To summarize our design; we use a hardware pen that records and sends positional IMU data through Bluetooth, a Unity software application that renders lines based on that data, as well as AR packages that allow these lines to be overlaid on the real world through a phone. The integration of all these parts allow us to make into reality our novel idea of Pictionary in AR. In order for a seamless experience, we hope to have smooth, low-latency lines that are not jarring for the user as they draw, but also manages to be seamless for use among many devices running simultaneously.

With this product an effort will be made to create a game that will promote face-to-face social connection and bonding, as well as promote a non-sedentary lifestyle by requiring active movement and interaction amongst peers. This is the perfect product for when one wants relative strangers at an event to form connections with each other through an icebreaker, or even just strengthening pre-existing bonds one has with their friends. In addition, this will be the perfect game for young children to remain active and develop socially in this current technologically isolating world.

Possible future developments are into the research of IMUs and better methods of using acceleration and orientation to determine linear displacement. While there exist IMUs that are extremely accurate in this regard, they are also extremely expensive. Research can also be done in this area to determine a way of making IMUs less expensive and more widespread as a result.

Glossary of Acronyms

- AR – Augemented Reality
- CV - Computer Vision
- DOF - Degrees of Freedom
- IMU - Inertial Measurement Unit

References

[1] Amazon. *Pictionary Air Drawing Game.* URL: https://www.amazon.com/Pictionary-Drawing-Light-up-Devices-Exlclusive/dp/B07P5PQZY7?th=1.

[2] Kaixin Liang Si-Tong Chen Liuyue Huang Tianyou Guo Can Jiao Qian Yu Nicola Veronese Fernanda Cunha Soares Igor Grabovac Albert Yeung Liye Zou Chunping Lu Xinli Chi. "Moving More and Sitting Less as Healthy Lifestyle Behaviors are Protective Factors for Insomnia, Depression, and Anxiety Among Adolescents During the COVID-19". In: *Pandemic, Psychology Research and Behavior Management* 13 (2020), pp. 1223–1233. DOI: 10.2147/PRBM.S284103.

[3] Marko Milijic. "45+ Video Games Industry Revenue Statistics: Game On!" In: (). URL: `https://spendmenot.com/blog/video-game-industry-revenue-statistics/`.

[4] VrChat. *VrChat*. URL: `https://hello.vrchat.com/`.

[5] Wikipedia. *Wii*. URL: `https://en.wikipedia.org/wiki/Wii`.

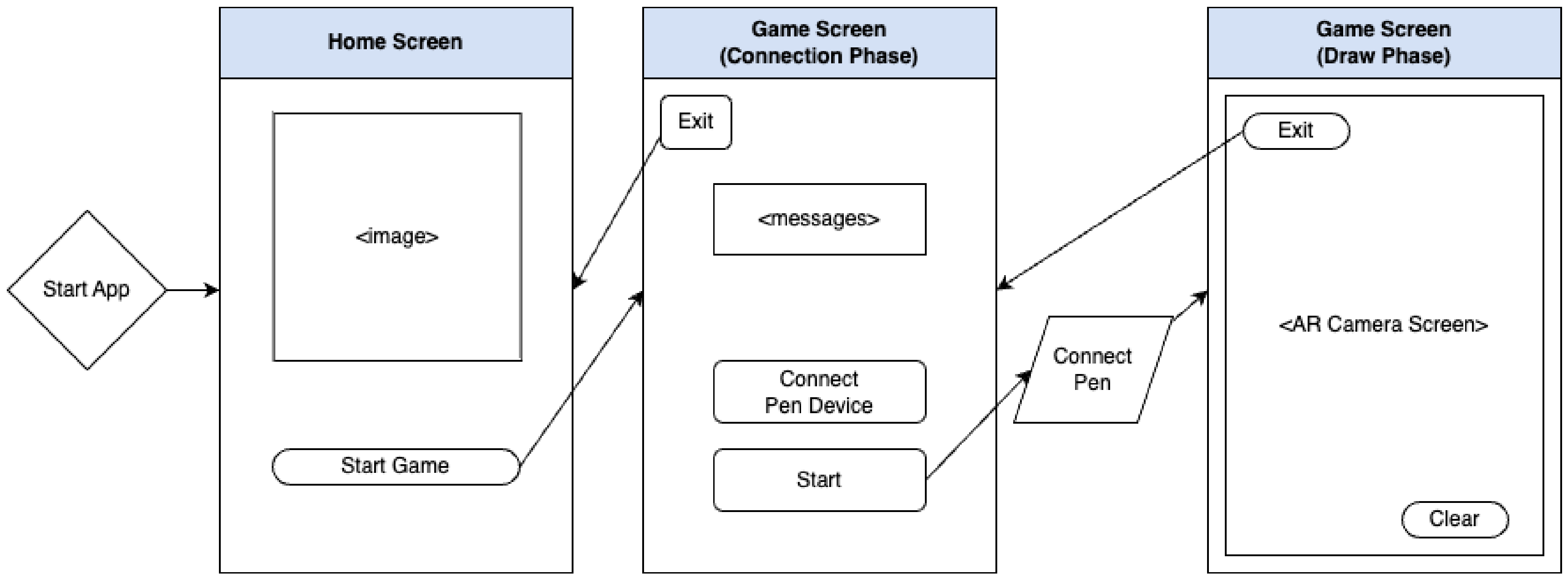
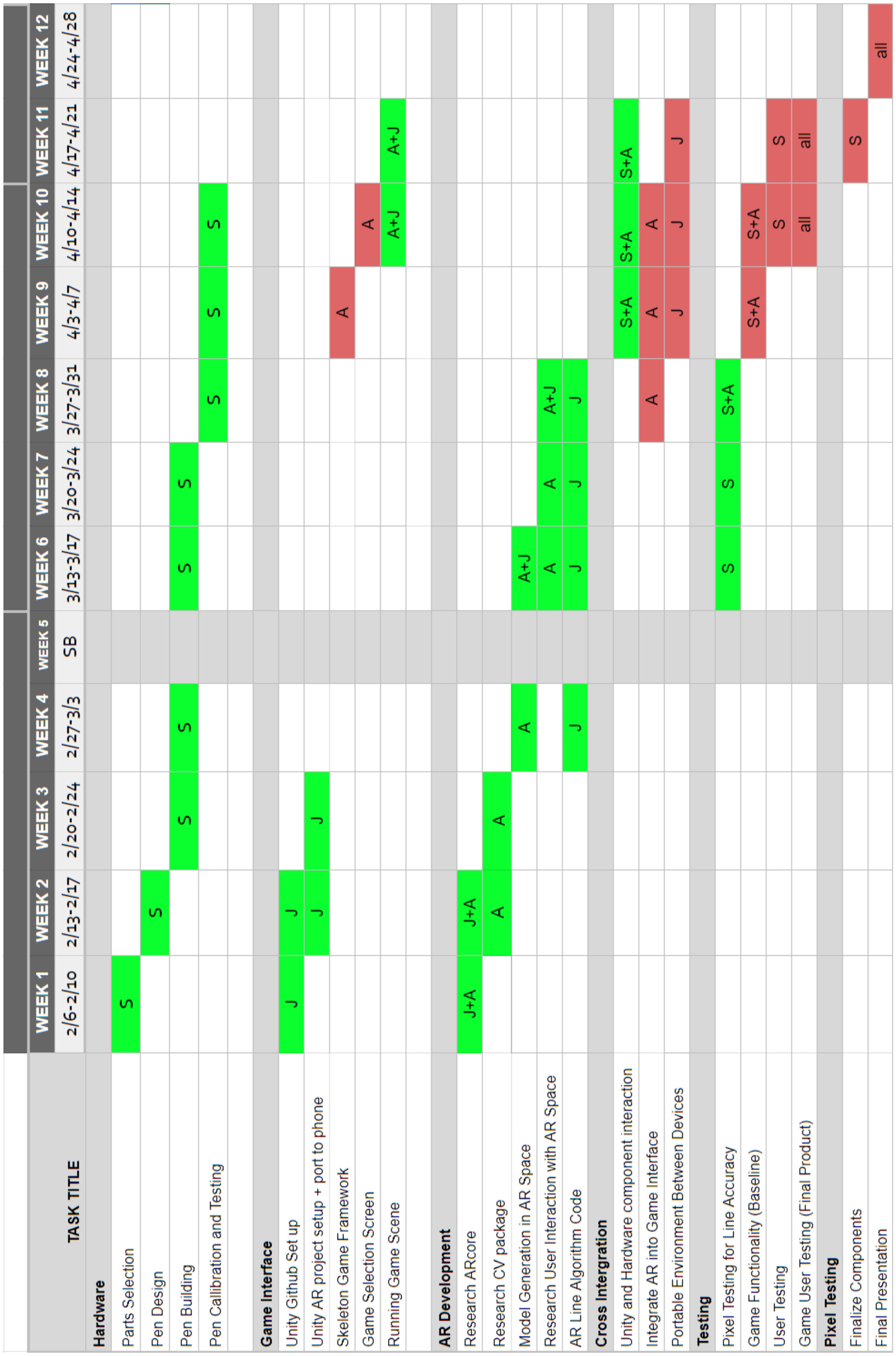


Figure 11: A full-page version of our app's UI layout and flow.



TASK TITLE	WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 5	WEEK 6	WEEK 7	WEEK 8	WEEK 9	WEEK 10	WEEK 11	WEEK 12
	2/6-2/10	2/13-2/17	2/20-2/24	2/27-3/3	SB	3/13-3/17	3/20-3/24	3/27-3/31	4/3-4/7	4/10-4/14	4/17-4/21	4/24-4/28
Hardware												
Parts Selection	S											
Pen Design		S										
Pen Building			S	S		S	S					
Pen Callibration and Testing								S	S	S		
Game Interface												
Unity Github Set up	J	J										
Unity AR project setup + port to phone		J	J									
Skeleton Game Framework									A			
Game Selection Screen										A		
Running Game Scene										A+J	A+J	
AR Development												
Research ARcore	J+A	J+A										
Research CV package		A	A									
Model Generation in AR Space				A		A+J						
Research User Interaction with AR Space						A	A	A+J				
AR Line Algorithm Code				J		J	J	J				
Cross Intergration												
Unity and Hardware component interaction									S+A	S+A	S+A	
Integrate AR into Game Interface								A	A	A		
Portable Environment Between Devices									J	J	J	
Testing												
Pixel Testing for Line Accuracy						S	S	S+A				
Game Functionality (Baseline)									S+A	S+A		
User Testing										S	S	
Game User Testing (Final Product)										all	all	
Pixel Testing												
Finalize Components											S	
Final Presentation												all

Figure 12: Gantt Chart