Picture This!

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

***Abstract*—This project aims to create a game of AR Pictionary where one player, the drawer, will have a hardware device called a pen and the guesser player will be able to view the AR drawings with their personal device cameras. We will be creating the pen out of an NodeMCU module, an IMUm and using Android phones in order to run the AR Unity App. The NodeMCU will collect the IMU data and double integrate it, via trapezoidal estimation, in order to obtain linear displacement values. This data will then be sent to participating Android devices via Wi-Fi. The Unity App running on the Android devices will take the line data and then render it in virtual space. 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, ESP8266 (WiFi Module)**

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 3 seconds 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 that is within 10% error of the distance their hand has moved.

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. Furthermore, the device should be able to last a few rounds of game play, so we propose a battery life of at least 4 hours.

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 3 seconds 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 will be holding the pen device and perform gestures in order to "draw" in physical space once game play begins. This data will be captured by the pen device and sent to all players' personal devices where the Unity App will be able to render lines that the drawer is creating in physical space to the AR space. This AR space will be visible to all players via the camera on their personal devices, where they will be able to see the drawer creating lines in AR space. 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, which will contain 4 hardware components: a NodeMCU, IMU, buttons, and an LED. When button is pressed, the NodeMCU will start recording the IMU data that corresponds to the player's movement. 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 perform gestures 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 that will be downloaded on all of the "guessers" personal devices. 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. As mentioned later in 5.1.2, this could potentially also be used for a CV assisted approach if necessary.

All of the guessers currently active in the game 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 Udunio package in order to record data from the NodeMCU itself, which will be sent over by the drawing mechanism during game play. This data will be processed, and using AR Foundation packages within Unity, will be able to process and render lines the drawer is trying to produce in the AR Space. By viewing their screens and camera, users will be able to view the rendered lines that the drawer has created. Other packages that come with AR Foundations, like Cloud Anchoring, will be used in order to make sure that the drawing is tied to a specific location in the real world.

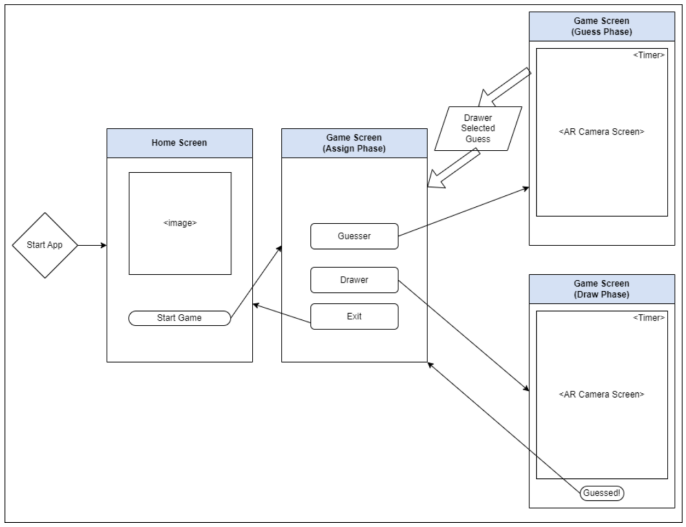

Figure 2: Game UI and how players get redirected from page to page

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.

Upon running the app, players will need to select the role that they will play during the current round of the game. The guesser players will select the "Guesser" role and be redirected to a screen with their camera view to see the AR drawing and a timer in the corner to indicate how much time is left in the game round. From this screen they will be able to see exactly what the drawer is drawing, and make guesses throughout the round.

The drawer player will instead select "Drawer" on the Game Screen. They will also have a camera view to see what they are drawing with the pen. When a guesser correctly guesses what the drawing is, the drawer can press the "Guessed" button at the bottom of the screen to end the round and return everyone to the assign role screen.

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 erasing drawn lines or making it completely remote such that 3 players on opposite sides of the world could play together. But for the current scope and timeline, this basic design will be suffice.

.

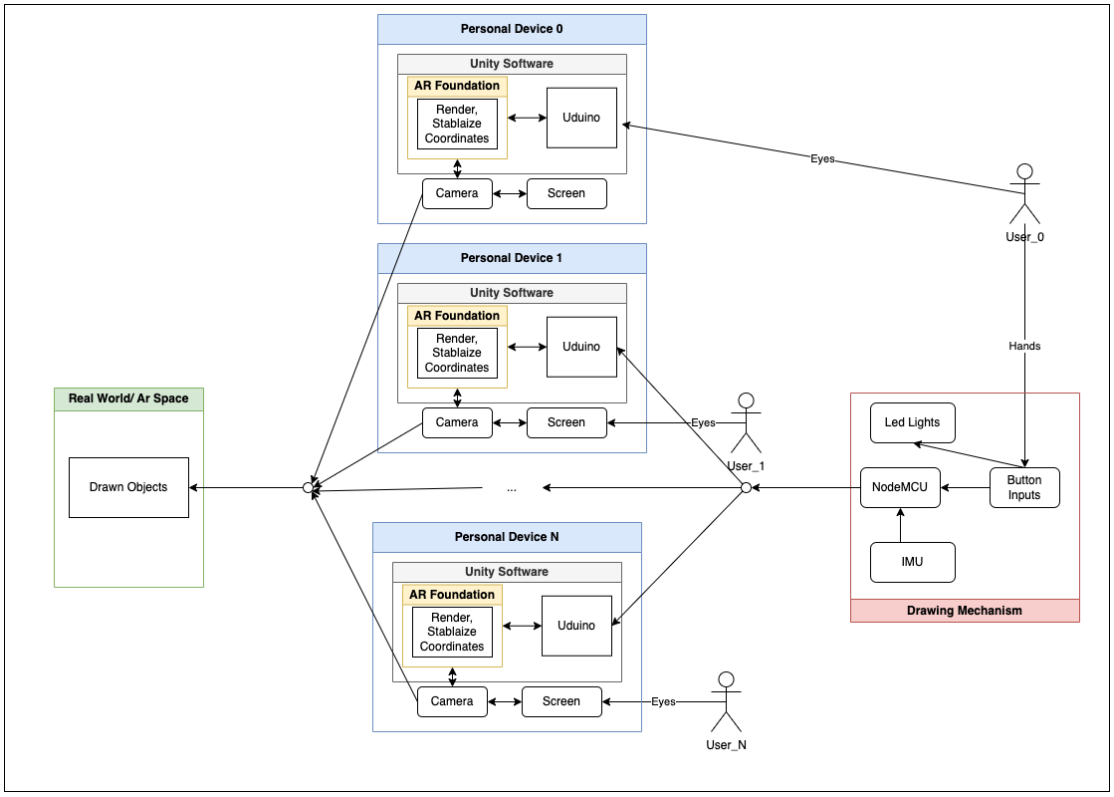

Figure 1: An overview of the game structure and the components. Red blocks represent any physical devices. Blue represents any software components. Green represents the virtual space where AR objects are overlaid onto the real world.

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

As previously stated in 2.1, the users of our game will require our App to be supported on at least 3 devices as well as have real time line drawing.

The first requirement naturally implies that our App will need to be downloadable onto multiple devices. For sake of simplicity, we will be implementing our applications for Android devices in order to streamline the development process. Since all players will need to be able to view the drawing, all devices will need to be able to interact with the pen from all the different devices. This interaction between devices will be handled mainly by AR Foundation, and the AR Core packages that are used in conjunction with the software.

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 an the AR space, which will be visible for players on the camera of their personal devices. 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 a distance test. We want the total distance deviation of a test line to only be 10% off of our reference line. More details on this metric will be discussed in Section 7.

We want the drawn line length to be correct within 10 cm of our test line and our reference line to account for the noise that wil be generated from the IMU. Once again, 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.

In order to hit our metric of a 4 hour battery life, we 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.

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

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 NodeMCU 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

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.

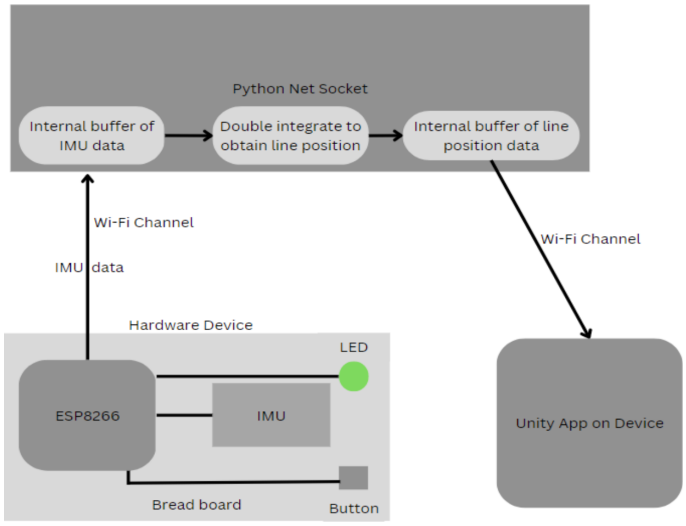

Figure 3: Schematic of an alternate hardware design that involves using a Python Net socket to assist in data transfer between ESP8266 module and our Unity App

Additionally, 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 communication protocol is necessary to ensure both ends are ready to send and receive data. Without this protocol, the line data would have been 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.

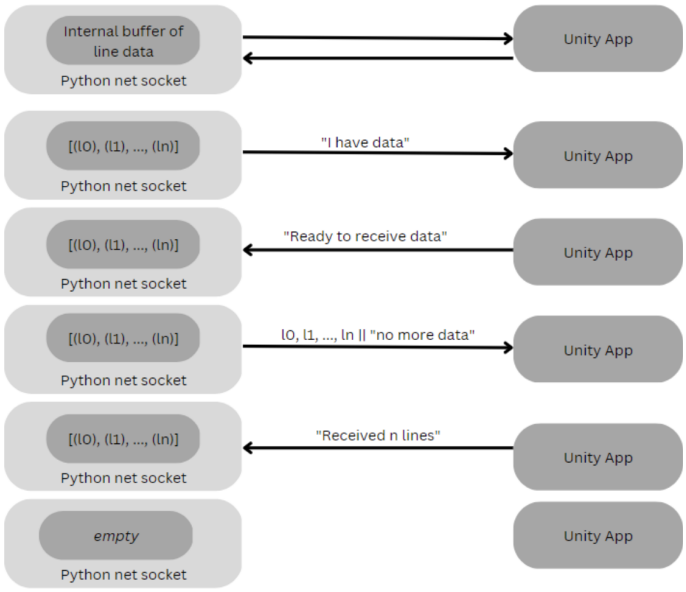

Figure 4: Communication protocol between the Python Net Socket and Unity on how to transfer data

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 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.

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 ESP8266 NodeMCU, green LED, IMU, and a button. The NodeMCU will be powered via two 3.7 V batteries that will be able to supply 2 Ah. This ESP8266 will also be able to power the IMU through its 3.3 V pin and will be connected to a button and LED to receive inputs and provide visual feedback to the user.

The drawer will press and hold on the button to indicate that they are in the process of drawing a line. The LED will also light up to provide visual feedback for the drawer. The IMU data will be collected while the button is pressed in order to determine relative position of the player's hand as they move it through the air. This data will be sent to the the Unity App on all players' phone through Wi-Fi.

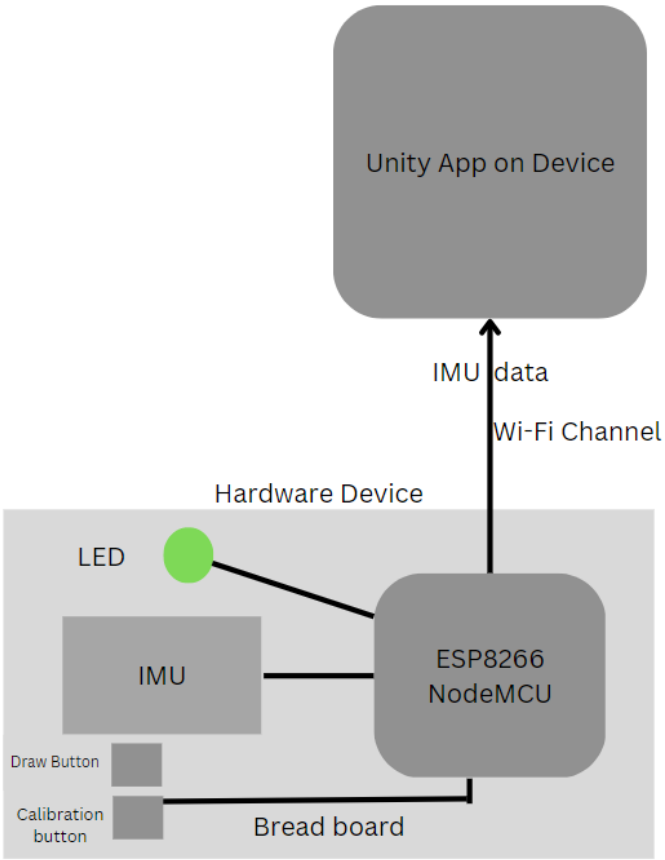

Figure 5: Schematic of the pen

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 exactly 2 additional lines. Once these 2 lines are drawn, the player must return to this calibration phase.

The ESP8266 module will collect IMU data once every 0.01 seconds to obtain an acceleration vector at this point of time. From this data, we will double integrate using trapezoidal estimation in order to obtain linear displacement between the starting design. This will be used in order to determine the location and shape of the line. This displacement data will be sent to the Unity App.

Displacement and the acceleration data will be collected for the purposes of testing and validation as discussed in 7.

6.2 Unity App

Figure 6 describes the general flow of data within our Unity App. Once ported to each players' phone, the Unity app will directly interact with the ESP8266 module through a Wi-Fi channel. Positional data from the pen will through be received through a C# script. This script that directly updates and renders the AR drawing will, at the start of a round of the game when the player first holds down the button to draw, instantiates a AR line renderer gameobject, through the use of the AR Foundation Package. Then it will tie that gameobject's position to a specific location in the real world using Cloud Anchoring from the ARCore Package. This anchor is created through taking the locational data from the NodeMCU and forming it into a Vector3 position, and then using that and the identity rotation to create a Pose struct. This will anchor the overall drawing's absolute position, relative to the real world, so that the drawing shows up in the exact same location on every users' phone, regardless of the point-of-view.

After this initial instantiation and tying to a location, the script will then always be in busy polling state, waiting for any new positional data from the NodeMCU to come through. Once it receives new data, the script will take the locational data and form it into a new Vector3 struct, which it will use to update the current position of the new point in space, while saving the previous position in space. The lines will be rendered, using the line renderer gameobject, between these two Vector3 positions in space, which a new pair of is created every set distance of 0.001 meters.

Everytime the user releases and then holds down the draw button on the pen, the instantiation of a new cloud anchor will occur. 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

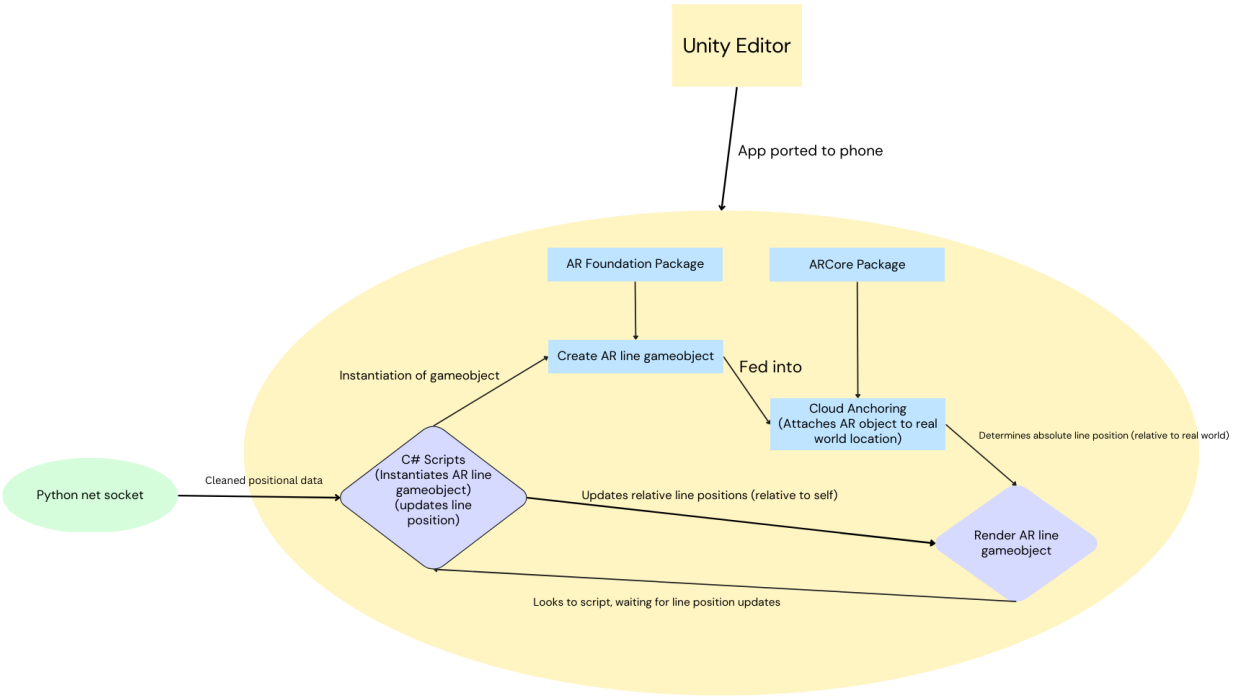

Figure 6: Flow and transfer of data within Unity

drawn in space, instead of following the drawer's phone around as they move around. At the same time, the cloud aspect allows other users to see the drawing in the same absolute location in the real world, as all the anchors are stored in the cloud.

7 TEST & VALIDATION

7.1 Line Accuracy Tests

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 7 gives an example of such a line with arbitrary points in space.

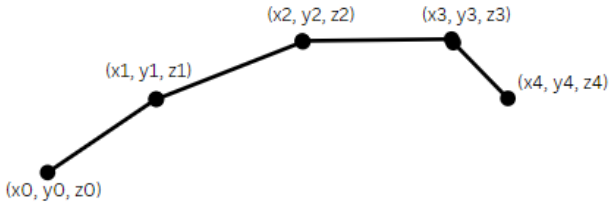

Figure 7: An example of a reference line with defined points in 3D space

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 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 7, 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.

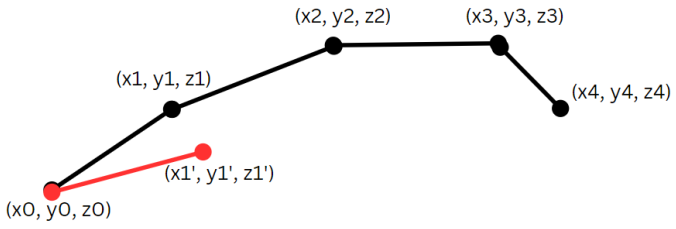

Figure 8: An example of a reference line with defined points in 3D space as well as an test line drawn by a user

Using the distance formula as described in Equation 2, the distance between each consecutive point in the reference line and the test line will be calculated.

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2} \quad (2)$$

We will calculate the percent error between the reference line distances and the test line distances as demonstrated in Equation 3. d_e is the expected distance or the distance from our reference line. d_a is the actual distance we determined from our test line.

$$\delta = |\frac{d_e - d_a}{d_a}| * 100\% \quad (3)$$

If the percent error between our reference line and test line is 10% of less, our AR lines are accurate enough.

7.2 Pen Size

We will construct our device onto a bread board and try to compact it as much as possible. We will measure the width, length, and height of this compacted area to obtain a metric of how large our device is.

7.3 Battery Life

We will run a test script that mimics automation of drawing the line by constantly setting the button input signal to high and low. We will run this script on loop until the battery dies and see how long it takes for the battery.

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. In case we do not hit this metric, we can further split this into two different categories.

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.

8 PROJECT MANAGEMENT

8.1 Schedule

The schedule is shown in Fig. 10.

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 on 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.

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 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 purhcased phone for the purpose of testing, the cost of our pen itself is $32.96

Table 1: Bill of materials

Description	Manufacturer	Quantity	Cost @	Total
ESP8266 NodeMCU	HiLetgo	3	$16.39	$49.17
LSM6DSO32 6-DoF IMU	Adafruit	3	$12.50	$37.50
IMU Fusion Breakout - BNO055	Adafruit	3	$34.95	$104.85
Button	NA	2	$0	$0
LED	NA	1	$0	$0
Breadboard	NA	1	$0	$0
Uduino	Unity	1	$15	$15
Samsung A10e	Samsung	1	$150	$150.00
Android Phone	NA	1	$0	$0
				$356.52

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

Description	Manufacturer	Quantity	Cost @	Total
ESP8266 NodeMCU	HiLetgo	1	$5.46	$5.46
LSM6DSO32 6-DoF IMU	Adafruit	1	$12.50	$12.50
Button	NA	2	$0	$0
LED	NA	1	$0	$0
Breadboard	NA	1	$0	$0
Uduino	Unity	1	$15	$15
Samsung A10e	Samsung	1	$150	$150.00
Android Phone	NA	1	$0	$0
				$182.96

8.4 Risk Mitigation Plans

8.4.1 Hardware Reliability

The software portion of our project depends on the hardware, so in order to mitigate the risk of the hardware taking longer than expected to debug, we focused on the hardware right from the very start. We initially only had one person working on the hardware, but we realized that this task seemed likely infeasible so we now 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 concern for hardware is whether or not the IMU signal would be too noisy for the purposes of our project. To mitigate this risk, we did detailed research by examining previous projects to assess which IMUs had high enough quality. We a couple of simple IMU test scripts to assess the quality of them.

Both ordered IMUs were sufficient enough for our purposes, so we settled on using the cheaper one. If an IMU breaks, we have backups just in case. If the accumulated error grows to be too much, we can switch to using the other more expensive, but accurate IMU. If that does not prove to be enough, we can also integrate sensor fusion as described in 5.1.1 to help improve the accuracy of our data.

8.4.3 Hardware-Software Communication

One of the biggest unknowns is the transfer of the data recorded by the hardware to the software run by the Unity application. After some research, we have determined that there are two possible methods: use a built in Unity package Uduino or write our own Python net socket.

Uduino allows for both serial, Wi-Fi, and Bluetooth communication between an Arduino device an Unity. If this package were to work as intended, this would allow us to interface between our NodeMCU module, which functions similarly to an Arduino, and Unity without any issues. The transfer protocol will transmit data in a similar manner to what we describe in 5.2, but will instead use Uduino's builtin functionality rather than requiring us to write our own.

If there are interfacing issues, a alternate method would be to write our own Python net socket to receive data from the NodeMCU and send it to the Unity App. This is described in Section 5.2.

8.4.4 Integration

Our biggest risk is the integration of the hardware, software, and multiple devices running concurrently. In order

to mitigate that risk, we leave ample time in the latter half to work through any issues that come up.

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 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.

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.

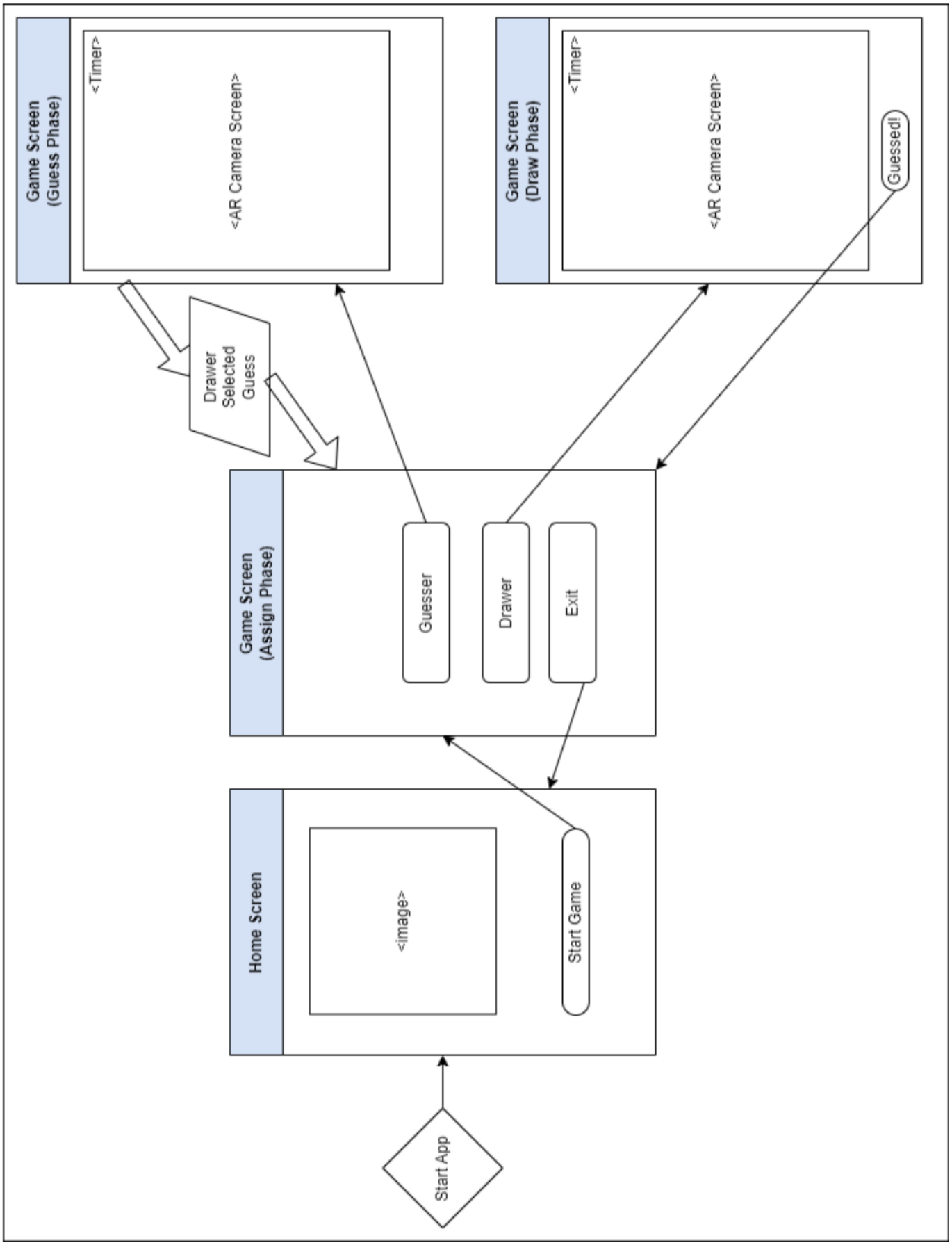

Figure 9: A full-page version of our app's UI layout and flow.

TASK TITLE	WEEK 1 2/6-2/10	WEEK 2 2/13-2/17	WEEK 3 2/20-2/24	WEEK 4 2/27-3/3	WEEK 5 SB	WEEK 6 3/13-3/17	WEEK 7 3/20-3/24	WEEK 8 3/27-3/31	WEEK 9 4/3-4/7	WEEK 10 4/10-4/14	WEEK 11 4/17-4/21	WEEK 12 4/24-4/28
Hardware												
Parts Selection	S											
Pen Design		S										
Pen Building			S	S		S	S					
Pen Callibration and Testing								S	S			
Game Interface												
Unity Github Set up	J	J										
Skeleton Game Framework		A+J										
Game Selection Screen			A									
Running Game Scene			A+J	A+J		A+J						
Enchance UI Design						A						
Line Algorithm				J		J	J					
AR Development												
Research ARcore	J+A	J+A										
Research CV package		J	J									
Model Generation in AR Space				A+J		A+J						
Reaserch User Interaction with AR Space						A+J	A+J	A+J				
Develop AR Across Multiple Devices							J	J				
Cross Intergration												
Unity and Hardware component interaction						S+A	S+A					
Integrate AR into Game Interface							A	A	A			
Portable Environment Between Devices										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 10: Gantt Chart