ECE 445
Senior Design Laboratory
Final Report

BAGS:

Bags Automated Game System

Team No. 23
Annabelle Epplin
(aepplin2)
Sania Huq
(saniah2)
Owen Schaufelberger
(ods2)

TA: Zicheng Ma
Professor: Olga Mironenko
May 3, 2023

Abstract

This project for ECE 445 Senior Design was dubbed BAGS: Bags Automated Game System. The objective of this project was to remedy score-tracking in a game of bags, also known as cornhole, as it can be an issue to keep track of the score, especially considering that bags is often played during parties, where people are more prone to becoming intoxicated. We approached this problem by finding hardware, such as an ultrasonic sensor and a radiofrequency (RF) antenna and receiver to establish location of the bags that land on the board in the hole to manage score to then display onto an app. An Sparkfun receiver was used as an Arduino shield as a means to send information via firmware since there were numerous problems with interference with the ultrasonic sensor and the RF antenna. Despite some technical difficulties with both hardware and software, we considered our project successful primarily due to the fact it met our high-level requirements to be considered a functional product.

Table of Contents

1. Introduction

1.1 Problem

Due to its fairly simplistic nature, cornhole is a staple at events such as barbecues, tailgates, and other outdoor get togethers. Some other staples at these types of events are adults drinking, energetic children, and engaging conversations. While these are great and part of what makes these events fun, they are distractions that can affect one's ability to keep track of the score. This can lead to heated arguments that are heightened due to alcohol consumption or take away the competitive edge as it can devolve into just throwing bags back and forth with no clear objective. Once losing track of the score of the game, it can be difficult and frustrating to realize the players either need to start over or have no idea who has actually won the game.

1.2 Solution

To combat participants losing track of score, we will be removing the need for them to score the game entirely. This will be accomplished by creating a set of bags, a board with a sensor array, and an app that will automatically score the game for them.

The Bags Automated Game System is meant to appear as a normal cornhole game that can be played fully as normal. Underneath the board, there is an array of sensors mounted to the back that are used to determine which bags are hitting the board and then when. The primary sensor to be used to do this on the board is the RFID detection system. RFID (Radio-frequency identification) uses electromagnetic fields to wirelessly communicate between objects for identification and location purposes [1]. To detect a bag falling into the hole, we used ultrasonic sensors, which determined if a bag passed through the hole. These sensors deliver the board information to the microcontroller unit which communicates wirelessly via Bluetooth via an app to a phone. This entire system is fueled by the power subsystem, which consists of a voltage regulator on the PCB, battery packs, and a wall adapter for the RFID reader.

1.3 High Level Requirements

To consider our project successful, we wanted to accomplish several objectives:

1. Successfully keep track of bag locations with regards to the board and the hole
2. App is able to keep track of score and time played with the game
3. System is able to distinguish between bags of different teams

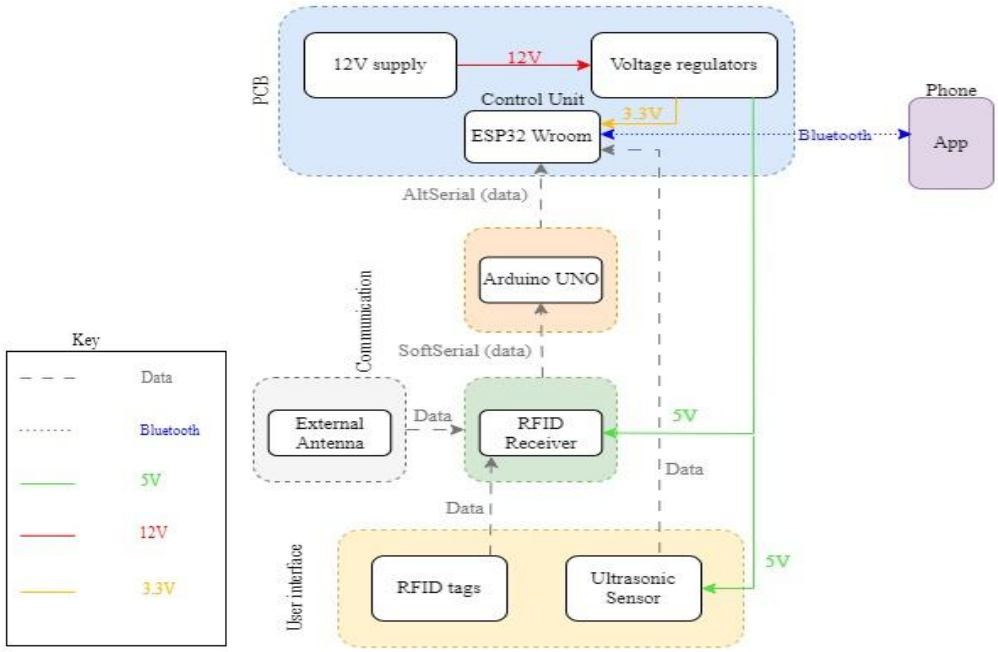


Figure 1: Block Diagram

1.4 Visual Aid

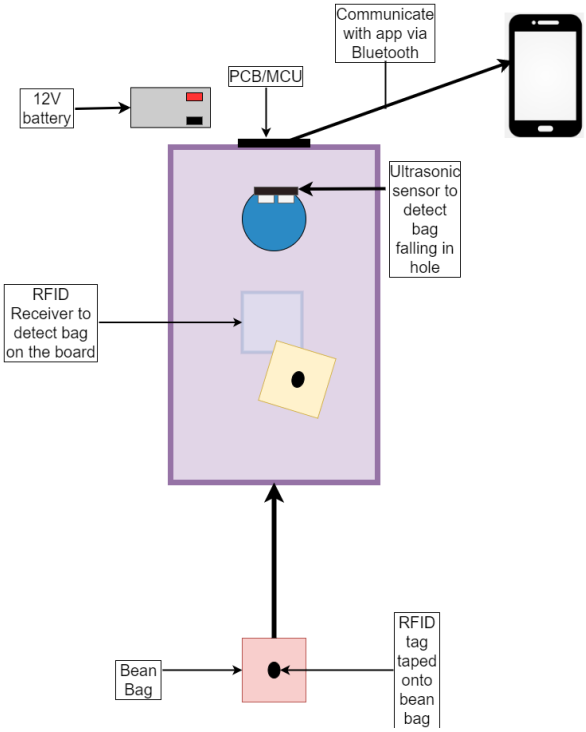


Figure 2: Visual Aid

1.5 Subsystems

Refer to Appendix for full tables of verification and requirements for each subsystem. Further discussion of the design of each subsystem is shown in Section 2.

1.5.1 Power Subsystem

The power subsystem locates itself on the PCB which provides adequate power to all our components including the ESP32, ultrasonic sensor, and the RFID receiver. It consists of a 12V battery, two voltage regulators and the ESP32 which was mounted underneath the cornhole board. In order to power all the other components that aren't directly attached, the appropriate wiring was attached to sufficiently power the IR emitter and receiver. The main purpose of our PCB is to supply voltage to all the necessary components and to ensure that enough voltage and current is delivered to ensure all the components are properly working.

1.5.2 User Interface Subsystem

The user interface subsystem consists of two main components: the RFID tags located in each game bag and the ultrasonic sensor that will detect game bags that go in the hole. The RFID tags will interface with the RF antenna whose data will be read by the RFID reader whereas the ultrasonic sensor was placed across the hole underneath the cornhole board. When a bag falls in the hole, the ultrasonic sensor detects an incoming bag that then communicates to the ESP32. These components are vital to the scoring of the game as they are the sensors that will actively detect the bag hits which will eventually become the game score.

1.5.3 Control Subsystem

The control subsystem is the brains of the operation that will function as a central location where the sensor data from the user interface subsystem will be received and stored as well as be able to communicate with the other subsystems. The control subsystem is divided into two main groups: the communications and the microcontroller. We went with the ESP32 primarily for its ability to use Bluetooth which seamlessly communicates with our sensors and the app. Bluetooth capabilities are needed for sensor data storage and processing in the app to collect information from the game board electronics. The ESP32 was programmed with the Arduino IDE, enabling the transmission of data via Bluetooth. The firmware is uploaded to the ESP32 via a micro USB type B connector located on the PCB. The other primary group within the control subsystem is the communications system, which consists of the RFID Reader.

1.5.4 App Subsystem

The app subsystem is responsible for keeping track of game statistics, which information will be received through the ESP32 Bluetooth module.

The app itself doesn't have any direct impacts on the actual gameplay of bags, rather it is a medium to manage score, game statistics and methods for improvement. An important aspect of the app is to effectively gather data from the ESP32 module, create game statistics, and keep track of the game. We tested a lot of software with an Android phone as there were more resources available at our disposal compared to iOS.

2. Design and Verification

2.1 PCB Design and Power

2.1.1 Schematic

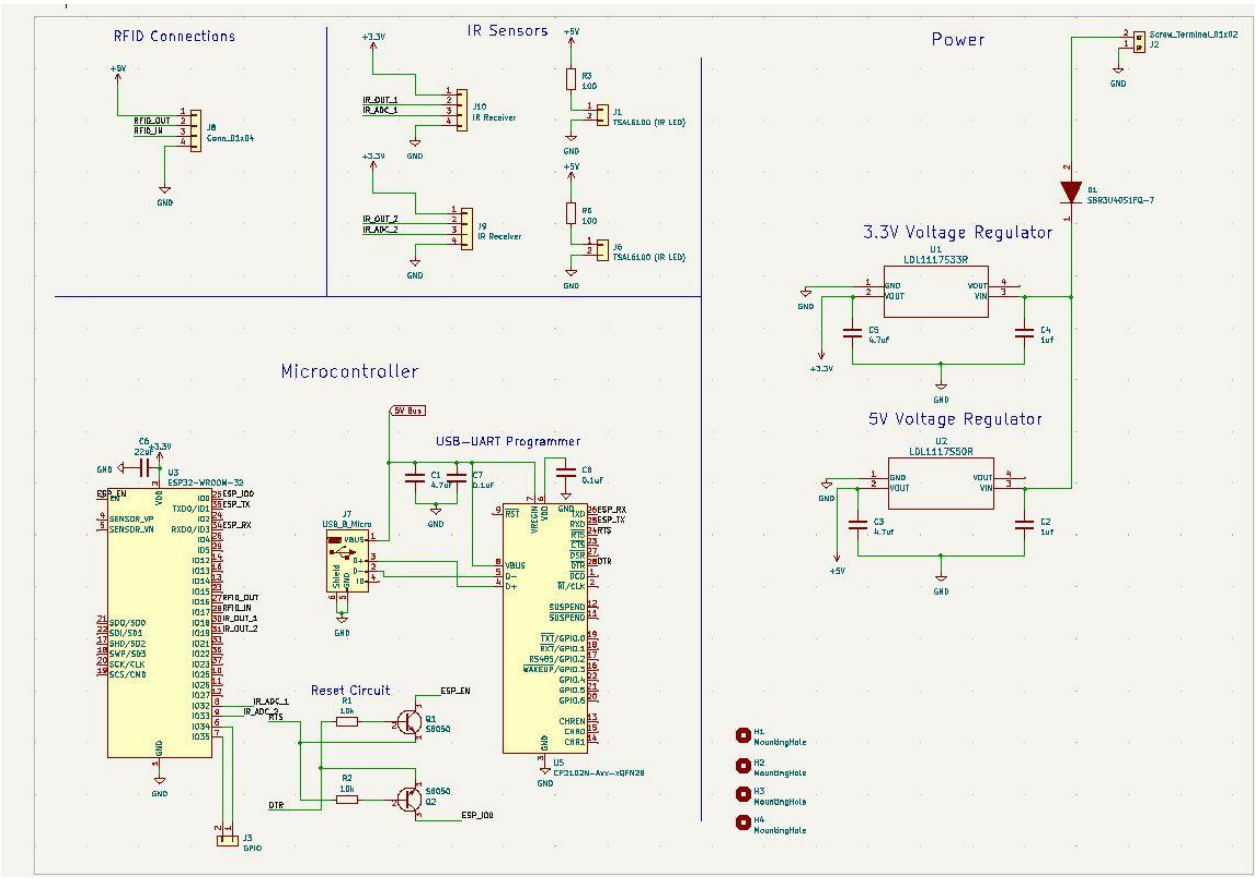


Figure 3: PCB Schematic

The PCB houses the power subsystem, the ESP32 microcontroller, and connections to our peripheral hardware such as the RFID reader and ultrasonic sensor.

The power subsystem consisted of our 12V battery, a LDL1117S33R 3.3V linear regulator, and a LDL1117S50R 5V linear regulator. We needed to be able to supply appropriate voltage and current to all components simultaneously, so we selected a 12V battery so we had enough potential power to do this. We opted for a battery rather than a wall plug due to the game of cornhole generally being played outside or in an area where wall outlets are not readily available. We specifically chose the linear regulators we did because they were both rated for an output current of 1.2A. We felt confident that there would be a sufficient amount of current based on the max current draw of each component as seen in table 1 below.

	HC-SR04 Ultrasonic Sensor	**Arduino UNO**	**ESP32 WiFi-BT-BLE MCU Module**	**Simultaneous RFID Reader**
Current Draw	15 mA	50 mA	Min (not transmitting Bluetooth): 500 mA Max (transmitting Bluetooth): 630 mA	170 mA
Voltage Draw	5 V	5 V	3.3 V	5 V
Power Draw (P= V * I)	0.075 W	0.25 W	Min: 1.65 W Max: 2.08 W	0.85 W

Table 1: Power Subsystem Voltage/Current Draws

Between our 12V battery input and the 3.3V and 5V linear regulators, we placed a Schottky diode. This serves as our reverse polarity protection in the instance the 12V battery is connected incorrectly. Both linear regulators are equipped with the recommended supplementary circuits provided in their datasheets [2] as shown in fig. 4. Each has decoupling capacitors at their inputs and outputs to ensure that there is alway stable supply signal for our components.

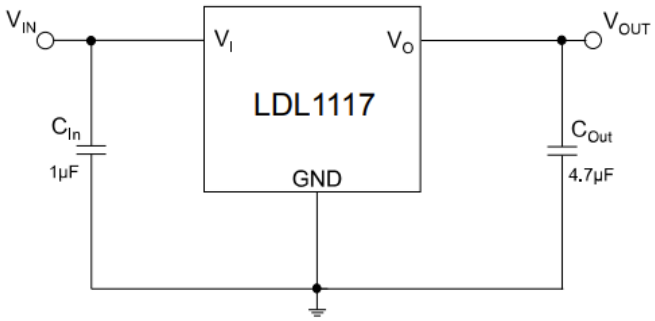


Figure 4: Linear Regulator Schematic

In our schematic above, we have circuits for IR emitters and receivers. These are remnants of our original design, however we still made use of one of the IR emitter connections. This particular

connection is connected to the 5V linear regulator and GPIO pins of the ESP32, which we were able to make use of to integrate the ultrasonic sensor.

The microcontroller section contains the ESP32 Wroom module, a CP2102N USB to UART IC, and supplementary auto programming circuits. The intention was to use a micro USB type B cable to upload our firmware to the ESP32 via Arduino IDE. The USB to UART IC generates a collection of signals based on the incoming data from the cable. Based on resources available online [3], we designed a circuit to utilize these signals consisting of two BJTs and resistors. Labeled as the reset circuit in the figure above, it was designed to either reset the ESP32 or put it into programming mode. It did this by sending its outputs directly to the EN and GPIO0 pins on the ESP32, which when pulled low places the microcontroller into programming mode to receive the uploaded firmware.

2.1.2 Layout

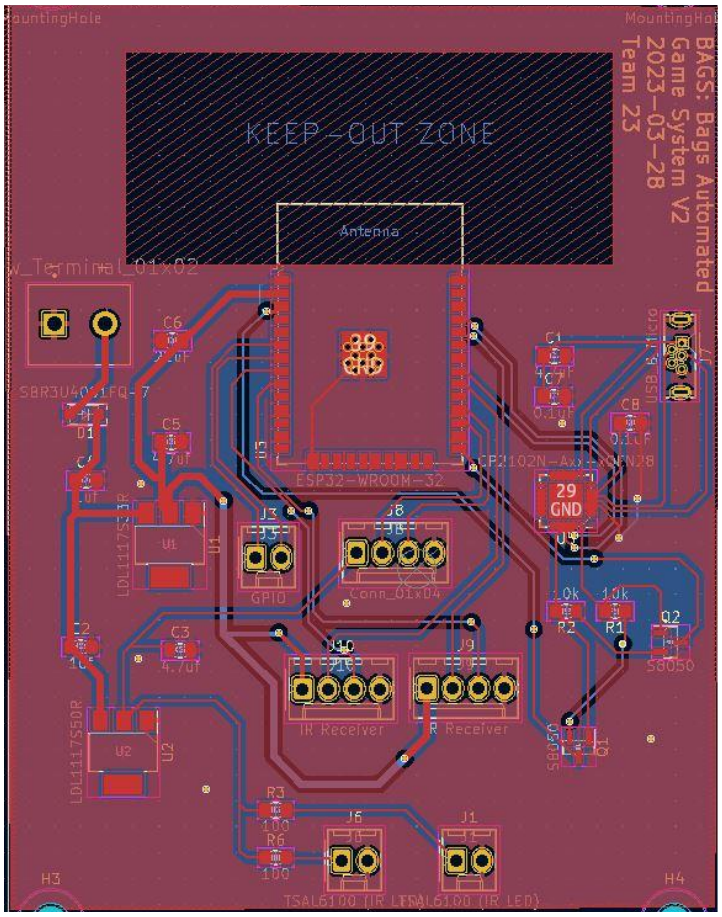


Figure 5: PCB Layout

We had many things to carefully consider when designing the layout of the PCB. The most significant design consideration is the width of the connections. We prioritized having wider traces for power delivery, which can be seen in fig. 4. We have the widest traces round the 12V battery terminal, inputs to both the 5V and 3.3V linear regulators, and into the voltage inputs for all our hardware peripherals because these are where we will have the highest voltage and current draws. For less

strenuous applications such as data transfers from the RFID read to the ESP32, we opted to use thinner traces which provided us more flexibility. We also placed 22uF ceramic capacitors as close as possible to our most important and vulnerable components. These served as decoupling capacitors to provide a more stable input voltage to the ESP32 and RFID reader.

The ESP32 Wroom's Wi-Fi and Bluetooth antennas are located at its top. Since Bluetooth was crucial for this project, we had to ensure that no other components were located within the designated keep-out zone, designated in the ESP32's footprint. This space is reserved to prevent as much electromagnetic noise as possible so that we could have the strongest Bluetooth connection possible.

2.1.3 Soldering and Testing

Some of the difficulties encountered while soldering were component sizing. Many of the capacitors and resistors were 0402's, so around .01 inches. The soldering pads were significantly larger than the actual components themselves and so many times, components would become damaged through soldering and lead to shorts. We also ended up having to create jump connections to make the regulators work, but since our components experienced damage through the soldering process, the continuity tests and voltage outputs were not passing our requirements. Some considerations we had for the second round of PCBs was utilizing the rest of the ESP32 pins and LED receiver ports for testing and debugging purposes.

Once everything was soldered, testing the components straightforward. We connected a 12 voltage source to see that our 5V voltage regulator and the 3.3V voltage regulator were functioning properly. On top of that, we were also able to get the ESP32 pins connected properly with support of the continuity tests.

When it came time to test the ESP32, we realized that we neglected pull up resistors. This was critical because it would enable the ESP32 to jump into boot mode. What we tried instead was physically adding a resistor to GPIO0 and connecting it to 3.3V in hopes of having the ESP32 get into boot mode. That failed and so our last resort was taking apart a development board that already had all the connections and solder connections there. This method didn't work either, and we believed that the ESP32 shorted, and after some multimeter verifications, it was shown to be true. While the ESP32 Wroom was meant to be surface mounted on our PCB, due to these issues we were unable to use it directly on our PCB. Instead, we ended up having to use a Sparkfun Thing Plus ESP32 Wroom dev board as our microcontroller. This was powered using the 5V linear regulator located on our PCB. While the ESP32 uses 3.3V for its power, the Sparkfun board contains a pin named VUSB equipped to take a 5V source and then use a 3.3V linear regulator to power the actual ESP32 module [4]. To avoid reaching the current limit of the 5V linear regulator, this also forced us to use a wall adapter to power the Arduino UNO and the RFID reader in the final demo.

2.2 Hardware Design and Verification

The hardware design is the aspect of this project that went through the most changes throughout the semester. At the conception of this project, we envisioned a cornhole board covered in force sensitive resistors, an ultrasonic sensor and IR receiver in the hole, and an RFID reader underneath the board. The idea was that the force sensitive resistors would be used to register where and when a bag had landed on the surface of the board. The RFID reader would simply be used to identify which team a bag belonged

to. The idea of having both the ultrasonic sensor and the IR receiver in the hole, was that the ultrasonic sensor would detect that a bag is incoming and the broken IR connection would sense that a bag has actually made it into the hole. We also experimented with the thought of using a camera and image processing to identify and score bags on the board and in the hole based on their colors.

The final hardware design we settled on was a simpler version of our first initial concept. It uses only an RFID reader to detect when a bag lands on the board and an ultrasonic sensor to sense when a bag lands into the hole. These peripherals were connected to our ESP32 Wroom microcontroller, which collected data, generated a score, and sent the results to the mobile app via Bluetooth, We opted not to use force sensitive resistors at all as we could not locate any with a large enough sensing area to cover the surface of our board. We also felt that using force sensitive resistors was redundant as we could detect and identify a bag landing on the board with just the RFID reader. We also abandoned the idea of using both an ultrasonic sensor and an IR receiver. We wanted to just use the IR receiver to detect bags falling into the hole, however through testing we found that electromagnetic interference made this approach unreliable. In the end we used only an ultrasonic sensor to detect bags falling into the hole.

2.2.1 ESP32 Wroom

At the heart of our hardware design lies the ESP32 Wroom microcontroller. This serves as the major junction that connects all of the peripheral hardware necessary for the complete product. We needed a microcontroller that was capable of interfacing with several peripheral hardware components at the same time. The ESP32 fits this criteria as it has 40 different GPIO pins that can be programmed in a multitude of ways such as serial inputs and outputs and digital-analog converters. While there are many microcontrollers on the market that have these features, our biggest draw to the ESP32 Wroom was its Bluetooth capabilities. One of the high level requirements we established for this project was to be able to keep track of the score with a smartphone app. We found that the most elegant way to do this was to send the score data to the app via Bluetooth.

To connect the ESP32 Wroom to the RFID reader and ultrasonic sensor, we took advantage of the many available GPIO pins on the ESP32. We connected the ultrasonic sensor to GPIO pins 18 and 19. These are general GPIO pins that were tuned in our ESP32 firmware to output a trigger signal to the ultrasonic sensor and receive the analog output signal of the sensor. The RFID reader sends its data serially, and thus needs to utilize one of the three hardware serial ports available on the ESP32. The first serial port UART0 (GPIO1 and GPIO3) are by default designed to send and receive data upon boot, meaning this serial port is set aside for uploading the desired firmware to the microcontroller. The second serial port UART1 (GPIO9 and GPIO10) is connected to the integrated SPI flash by default. This left us with having to use the third serial port UART2 (GPIO16 and GPIO17), which by default are free for any use [5]. We had incredible difficulty getting the ESP32 and RFID reader to communicate directly, and thus had to make use of an Arduino UNO to act as an intermediary. We struggled with this due to poor documentation around the RFID reader as it is designed as an Arduino shield and thus, nearly all documentation for it revolves around an Arduino.

Most testing of the ESP32 Wroom's functionality before integrating all the components revolved around making an LED light up. To test the ability to simple program the ESP32, we uploaded a code in which one of the pins periodically outputted a high signal to light up an LED. To verify functionality and gain a basic understanding of the ESP32 Wroom's Bluetooth capabilities, we experimented with a simple serial Bluetooth example provided in the ESP32's library in Arduino IDE. With this example code, we were able to type messages in the serial monitor and send them to be displayed on a serial Bluetooth

smartphone app and vice versa. To verify we could make use of this, we modified the code such that a specific message sent to the ESP32 would make an LED light up. With these verifications in place, we were able to integrate

2.2.2 Ultrasonic Sensor

In the final design, we elected to employ an ultrasonic sensor to detect if a bag has fallen into the hole. The ultrasonic sensor is supplied 5V from our 5V linear regulator and connected to the ESP32. Originally, we had planned to use a Vishay TSOP34838-IR receiver in conjunction with an IR emitting LED. The idea was that the IR receiver outputs a low logic signal whenever it is detecting the IR light from the LED. When this connection is interrupted the receiver outputs a high logic signal. The emitter and receiver would be placed across the diameter of the hole. When a bag was detected by the RFID reader and the IR receiver outputs a high signal, the bag was detected as falling into the hole. The receiver required a pulsing light that matched its rated frequency of 38kHz in order to produce meaningful outputs. We had great difficulty producing this signal as the ESP32 does not have an elegant way of producing the required squarewave. Since we were already using the Arduino UNO, we used the tone() function (unavailable on the ESP32 [6]) to produce the desired square wave. When testing the whole system however, we found that when in close proximity to two or more RFID tags, the desired square wave had large amounts of interference and resembled a DC signal. This caused the IR receiver to constantly read at a high logic level and thus, bags that landed on the board would be detected as falling into the hole. This violated our high level requirement of being able to distinguish bag location with regards to the board and the hole.

The solution we settled on utilizes the ultrasonic sensor from our original design. In our firmware, we are repeatedly sending a pulse to the trigger pin of the ultrasonic sensor, which generates an 8 cycle ultrasonic burst from the transmitter side. This signal is then reflected back by any object within the rated range of 2 cm-400 cm and detected by the receiver side. When the reflected wave is detected, the echo pin outputs a high logic signal and using the amount of time it outputs a high signal, we can determine how far away the object is. To test the sensor's functionality, we use a very basic distance program. As shown in fig. 5, we placed a hand across the diameter of the hole across from the sensor. The program then outputs that the distance to the hand is 13 cm as shown in fig. 6. Using this distance, we were able to tune our firmware to score a bag as in the hole, if the bag is both detected by the RFID reader and within 13 cm of the ultrasonic sensor. We did however, have some difficulties with the ultrasonic sensor during our final demo. We had it working shortly before the demo, however we needed to transport the board into the lab from another location. During transportation several wires were knocked loose and due to this, the ultrasonic sensor did not function as expected. We have since fixed this issue but for our demo we did not have the time to troubleshoot and locate the issue.

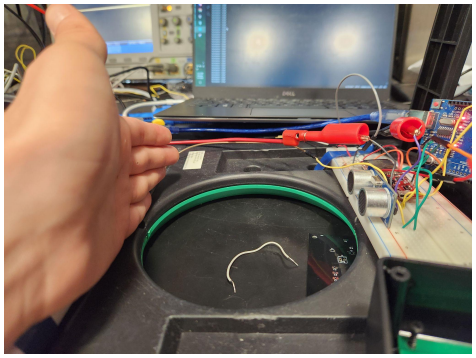


Figure 6: Hand Blocking Sensor

```
Distance: 13cm
Distance: 13cm
Distance: 13cm
Distance: 13cm
Distance: 13cm
Distance: 13cm
Distance: 13cm
Distance: 13cm
Distance: 13cm
```

Figure 7: Ultrasonic Sensor Output

2.2.3 RFID System

One of the largest hardware pieces of this project was the RFID system, which was responsible for detecting if a bag was thrown somewhere on the board or in the hole. The RFID reader used in this project was a Sparkfun Simultaneous RFID Reader with an M6ENano chip coupled with an external UHF RFID antenna. The schematic for this system is shown in Figure 8. Originally, we had planned on using the internal Sparkfun antenna, but it did not perform to the necessary read range to reliably detect multiple bags on the board. The RFID reader could be operated at a power anywhere from 5 dBm to 27 dBm [7]. The read range needed for this antenna was 2 feet, so after testing a range of power outputs, it was determined that the RFID reader should run at 18 dBm to best cover the entirety of the cornhole board without risking a possible shutoff due to a voltage brownout. This was tested by placing bags on each corner of the board at varying power outputs to see which power level could fastest identify each bag without the RFID receiver overheating.

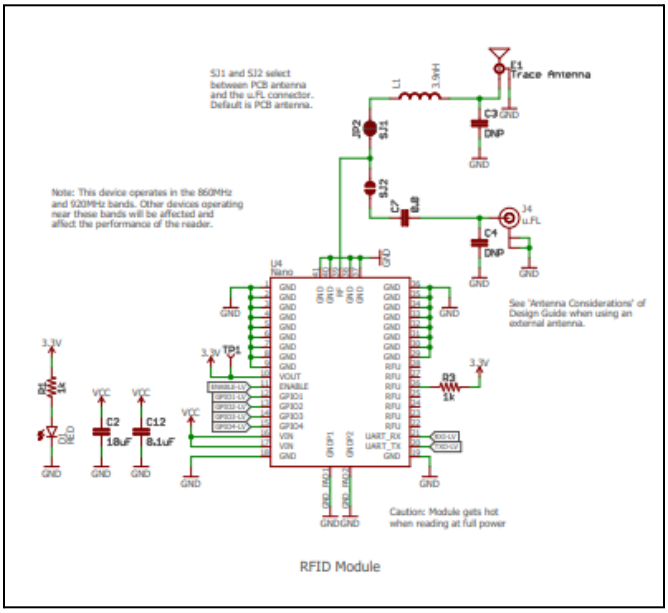


Figure 8: Sparkfun RFID Reader Schematic [7]

For the verification process, the RFID reader was tested with various amounts of tags to ensure it could detect up to 8 unique RFID tag IDs. These unique RFID tags IDs were programmed using the RFID reader and a Write EPC program written in Arduino IDE and were written to be a 4 bit hex value that corresponded with each team and bag number [8]. An example of the RFID functionality is shown below in Figure 9. When a bag was thrown on the board, the RFID reader would read the input shown in the figure, consisting of: received signal strength indicator (RSSI), frequency, timestamp, and electronic product code name (EPC). The EPC name is the most vital element as it shows the unique tag identifier assigned to that tag which is eventually used to match up the right score to the team. As shown in the figure, multiple bags can be detected on the board at once and identified separately, which serves to satisfy our main requirements for the RFID system. For a further look into the specific requirements and verifications for the RFID system, see Appendix. This process was coded using Constant Read function code on Arduino IDE and programmed as an Arduino Uno shield. One issue encountered with the RFID system was the delay for it to sometimes read bags that were found on the outer edges of the board. This was mostly fixed by experimenting with the power output as described earlier, although the potential for such delay still existed. We did not encounter any significant issues with the RFID system detecting bags that missed the board and hole. The RFID system could detect all bags on the board within an absolute maximum of 90 seconds, although it was much faster with fewer bags on the board. Due to issues with the voltage regulator on the PCB, a wall adapter was needed to fully power the RFID reader during the final demonstration.

```
 rssi[-47] freq[921400] time[10] epc[22 2F ]
 rssi[-46] freq[923400] time[43] epc[22 2F ]
Scanning
Scanning
 rssi[-45] freq[924200] time[502] epc[11 1F ]
 rssi[-45] freq[918000] time[536] epc[11 1F ]
 rssi[-42] freq[927200] time[570] epc[11 1F ]
 rssi[-45] freq[920000] time[602] epc[11 1F ]
```

Figure 9: Arduino Uno Serial Monitor RFID Output

2.3 Firmware Design and Verification

The firmware of this project unites the hardware and software and allows us to collect data received from the hardware sensors and transmit this to the microcontroller to be received eventually by the app via Bluetooth. The firmware process is shown below in Figure 11.The ESP32 code is responsible for continuously pinging for new RFID or ultrasonic sensor data. When a new tag is identified, it goes through simple logic to determine if the bag is on the board or in the hole. If the ultrasonic sensor has transmitted data at the same time as the RFID receiver, the ESP32 determines that the bag is in the hole and adds +3 to the correct score. If the ultrasonic sensor does not have any new data, the ESP32 decides that the bag was on the board and adds +1 to the correct score. After this logic occurs, the tag name is printed to the ESP32 serial monitor along with a message saying which bag was found and both teams' scores. This is shown in Figure 10 and serves as a verification that the tag data was transmitted and identified properly. Once the 'bag found' flag went high for a specific bag, it would not add any more points for that bag until the round was reset, eliminating the potential for the same bag being counted multiple times in one round.

```
11 1F
bag 1 front found
1
1
11
```

Figure 10: ESP32 Serial Monitor Output

This new score is then written to ESP_BT. This is sent to the app to be presented as the score of the game. To verify these connections, we tested it by sending a given message from the ESP32 Serial Monitor to a Serial Bluetooth smartphone app, as is discussed further in the verification discussion of Section 2.2.1.. One of our beginning difficulties with the firmware was finding enough Serial communication channels to use. To communicate between the Arduino Uno and the ESP32, we used an AltSerial port as we had no more available Software or Hardware Serial ports [9]. Our other difficulties with the firmware were delays that were caused by electromagnetic noise being received by the system, which is discussed further in Section 4.1.

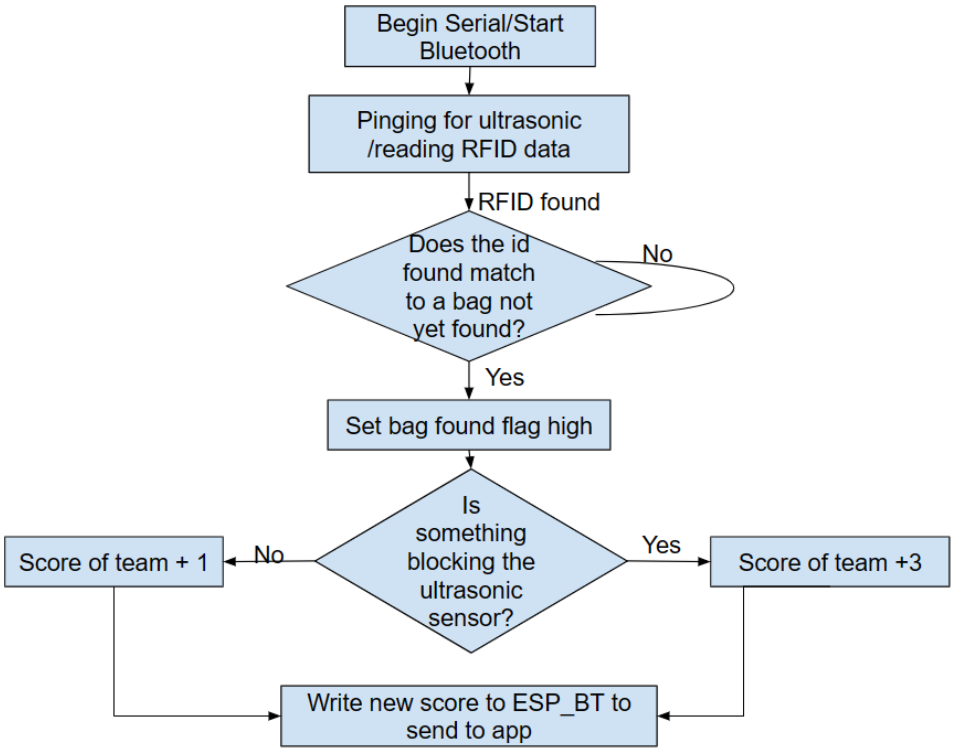


Figure 11: Firmware Process Flowchart

2.4 Software Design and Verification

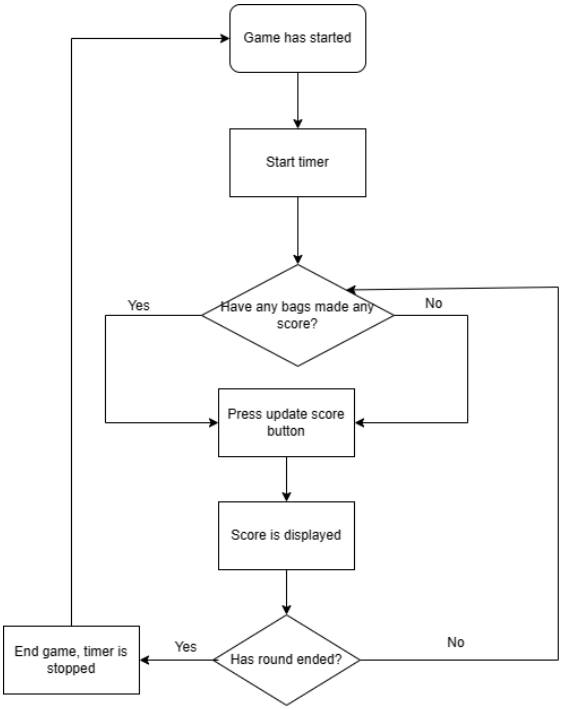


Figure 12: Software Process Flowchart

To meet one of our high level requirements, we created an app to function as the primary means of displaying the score and keeping track of the score. The operations of the app were straightforward. A timer was created alongside the score management. The decisions that had to be made within the app weren't very demanding. All it would do would check for any bags on the board once the update score button was clicked, revealing the current score. Even if no score was made, the app would display "Team A: 0, Team B: 0" for points. Once a score was made and the button was pressed, the appropriate score would be shown. This continued until all the bags were thrown. Once everything is scored, the game ends, the timer stops and a new game can be started.

Some of the issues that we came across were polling for data from the ESP32. We first encountered this when trying to integrate software and firmware together. At first, the app was only able to receive one packet and show one score since all the ESP32 serial pins were in use for the RFID system. We were able to remedy this problem by sending both scores at once and having software parse through the scores with a flag to show the two different team's scores.

Another issue that we came across was the timer on the app. The app would start the timer as soon as the application was loaded, but we felt it was more appropriate to have a timer start as soon as the game started. It was an issue of having an if-else statement, which we added and fixed the problem immediately.

One last task we hoped to accomplish was having past scores as a feature. Due to lack of time, we were unable to implement this feature, but it did not prevent us from completing our high level requirements as this was a verification method of the app.

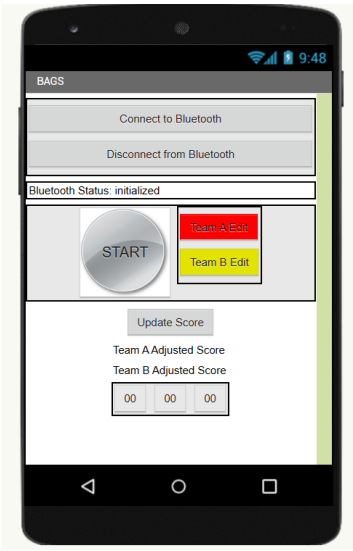
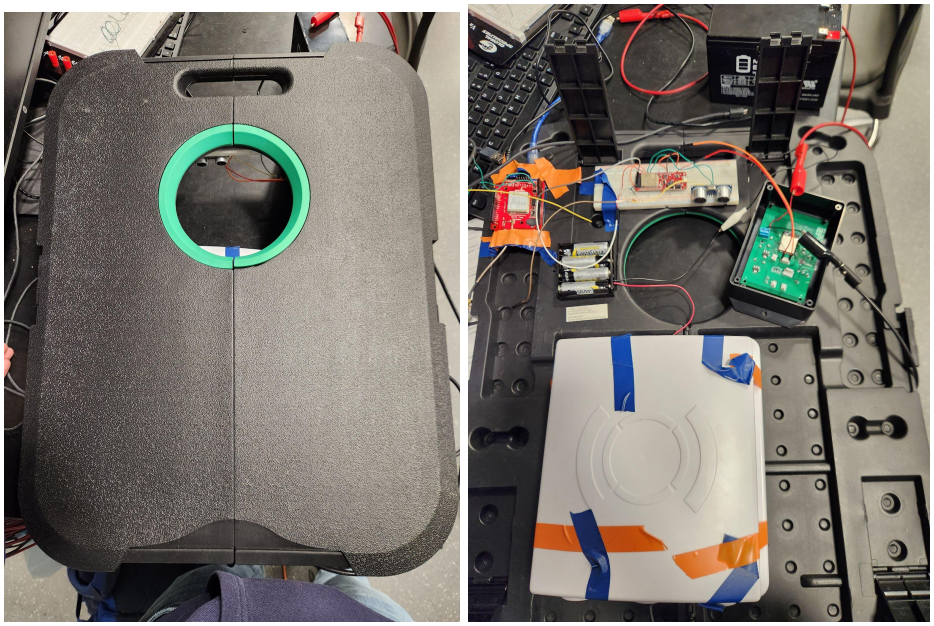


Figure 13: Mobile App Display

2.5 Integration

The integration of the entire project involved ensuring all of the hardware sensors were working, communicating with the microcontroller, and properly sending data to the software component. When assembling our total project, we housed all of the electronics underneath the board so as to not interfere with typical gameplay. Our antenna was housed in the center underneath the board to give it the best chance of successfully reading bags anywhere on the board. The ultrasonic sensor was located directly next to the hole to best determine if any bags had fallen in the hole. Our goal was to make the electronics unnoticeable from the front of the board so as to preserve the functionality of the game. As mentioned in Section 2.1.3, we ended up using a wall adapter for the RFID reader in the final demonstration, so the system had to be near a wall outlet to be successfully used. Our choice of a smaller, portable board allowed us to have a more portable system that we could transport if needed.

To verify this final integration, we tested various cases of gameplay to ensure the scoring would work no matter the number of bags that landed or their location on the board. During each test, we monitored the serial monitors of both the Arduino Uno and ESP32 to ensure data was being collected and properly transmitted. Originally, when testing the integration of the entire system, we had issues when verifying that the score data could be sent from the ESP32 to the app. To test this, we attempted to just send a “1” from the ESP32 serial monitor to the app via Bluetooth. When this succeeded, we attempted to once again send the score to the app as an integer rather than our original idea of a string, which succeeded. This final piece allowed us to fully integrate our system into a fully automated cornhole game.



Figures 14: Front and Back View of Assembled Board

3. Cost and Schedule

3.1 Cost Analysis

The first calculation needed to decide the total cost of this project is the total cost of all the components that we purchased. This includes components we used in the final design, components we did not use in the final design, and unused spare components. The price of each of these items is listed in table 7. The total sum comes out to be $442.02 without taxes. Subtracting components we already own, this becomes a total cost of $408.41. We must also calculate the total labor costs associated with this project. For this project, we can expect an hourly rate of $40 for our work. The entirety of the senior design project from project approval to the end of the course was around 12 weeks. We each spent an average of roughly 14 hours a week on this project. Given the fact that our team consists of 3 people, the total cost of labor would be $40/hr x 14 hrs/week x 12 weeks x 3 team members = $20,160. This would add up to a total project cost of $20,568.41 not including taxes and the items we already owned.

3.2 Schedule

Date	Huq	Epplin	Schaufelberger
Feb 20-27	● Software researching ● PCB drafting	● Components researching and ordering	● Components researching and ordering
27- March 5	● Software application in the works ● PCB design finalized	● PCB design is finalized ● Sensor and components all ordered, actual building begins	● Sensor and components all ordered, actual building begins ● Begin working on modularization
March 5 - 12	● PCB design finalized, order is sent out ● Teamwork evaluation filled out ● Software development	● PCB design finalized, order is sent out ● Teamwork evaluation filled out ● RF shielding prototyping	● PCB design finalized, order is sent out ● Teamwork evaluation filled out ● Firmware in progress
March 12-19	● Spring break	● Spring break	● Spring break
March 19-26	● Document updating ● Software development	● Document updating ● Unit testing of	● Document updating ● Unit testing of subsystems

		subsystems • RF shielding and testing	• Firmware development
March 26 - April 2	• Software done • Debugging	• Debugging • RF shielding + systems completed	• Firmware testing
April 2 - 9	• Debugging and testing software with firmware • Mock demo testing • Mock presentation preparation	• Debugging and testing software with RF shielding • Mock demo testing • Mock presentation preparation	• Debugging and testing software with firmware • Mock demo testing • Mock presentation preparation
April 9 - 17	• Final presentation preparation • Final paperwork	• Final presentation preparation • Final paperwork	• Final presentation preparation • Final paperwork
April 17 - 24	• Mock demos • Final demos	• Mock demos • Final demos	• Mock demos • Final demos

Table 2: Semester Schedule

4. Conclusion

4.1 Challenges/Uncertainties

The largest challenge faced by this project was electromagnetic noise generating false results within the system. Our project focused around an RFID location system coupled with multiple other serial communication channels all attempting to communicate at the same time. This had the potential for creating significant delays within data transmission. We especially noticed this to be a problem when attempting to use the 38kHz IR emitter with the 38kHz RFID reader M6ENano chip which ultimately led us to switch from the IR sensor to the ultrasonic sensor, as discussed in Section 2.2.2. This problem was especially noticeable during the final demonstration, where much longer score delays were caused due to the prevalence of significant electromagnetic noise within the senior design lab area. We know this to be the cause of our issues due to the history of electromagnetic interference when developing our project and our ability to test in an open area without the same delays.

An example of this noise affecting our system is shown in Figure 15. The figure shows the ESP32 serial monitor screen. When compared to Figure 10, it is seen that there is appearingly “junk” being sent from the Sparkfun RFID reader to the ESP32. This makes it more difficult for the ESP32 to correctly

identify the bags and attribute the points to the right team and has the potential to add a notable delay to the score calculation.

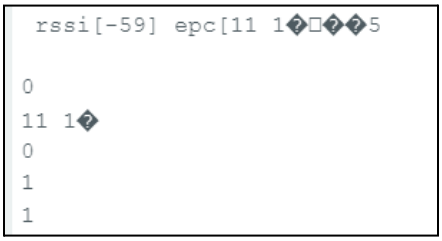


Figure 15: ESP32 Serial Monitor Electromagnetic Noise

To avoid this, we tested with multiple different baud rates for each serial communication channel in an attempt to prevent the different channels from interrupting each other and causing unintelligible data to be sent in the system. The baud rates we eventually decided on along with the frequencies certain components operate at are shown below in Table 3. The other primary challenge we faced in our project was the issue with the 3.3 V voltage regulator components on the PCB. This is discussed further in Section 2.1.3 and resulted in us being forced to use the breadboarded ESP32 rather than the PCB ESP32 for our microcontroller.

Transmitting Channel	RFID Tags	RFID Antenna	Sparkfun RFID Reader	Arduino Uno	Ultrasonic Sensor	ESP32
Receiving Channel	RFID Antenna	Sparkfun RFID Reader	Arduino Uno	ESP32	ESP32	Bluetooth
Baud Rate/Frequency	860-960 MHz	38400	115200	9600	40 kHz	2.4 GHz

Table 3: Baud Rate and Frequency Selections

4.2 Successes and Key Takeaways

In order for us to deem our project successful, we had to meet our high level requirements, which were:

1. Successfully keeping track of bag location with regards to the board and hole
2. App able to keep track of score and time played with the game
3. System is able to distinguish between two different teams playing

We were able to meet all these requirements, therefore we considered our project a success. Although we had some ultrasonic sensors during the demo (refer to section 4.1), we were still able to prove during our testing and demonstration video that all these requirements were met. As for key takeaways, all of us were able to understand the research process and understand the value of documentation while testing. Although we ran into issues, we used our resources and asked one another

for guidance. We gained experience in creating an electronic project and learning to be creative to solve unique problems. Lastly, we were able to truly understand and appreciate the value of teamwork. If one part of a team fails, it hurts the other team members and by working together, we could find solutions faster and better rather than doing it individually.

4.3 Future Work / Design Alternatives

If this project were to be continued or redesigned, there would be multiple avenues of improvement we would choose to pursue. The first item we would have changed would be our final demonstration location. Due to all of the electromagnetic noise in the senior design lab area, our project experienced significant time delays during operation, which likely would not have occurred as severely in a more open-air test area, as discussed in Section 4.1. Another potential future improvement would be the implementation of a physical LED scoreboard on the side of the cornhole board rather than a mobile application. This would allow for more simple viewing of the game score and also help to reduce delay by eliminating the Bluetooth communication delay between the ESP32 and the app. A third potential improvement that we would have changed in a redesign would be the choice to use ultrasonic sensors over IR sensors from the very beginning of the project. A final potential improvement would have been to redesign the PCB so that the 3.3 V voltage regulator would work as intended and we could have used the microcontroller on the PCB rather than the breadboard.

4.4 Ethical Considerations

For a successful senior design project, it is imperative that safety and ethics policies are followed throughout the entire process (IEEE Code of Ethics I.1) [10]. Because our project is an auto-scoring cornhole game, there are not many immediate safety risks to users. However, there is the potential of electrical shock or fire due to faulty wiring or degradation of the board in inclement weather. Although there is always going to be some risk, every care will be taken to minimize the little risk our project poses.

Ethical concerns must also be addressed to ensure that our senior design project is compliant with IEEE standards. Due to the nature of our project, the main ethical concern would most likely be plagiarism. In this project, designers will be researching several different components and methods and looking at other products in the automated game industry, such as auto-scoring dart boards. It is vital to make every effort to ensure each source is correctly cited and referenced in the senior design process.

5. References

[1]“What is RFID - how does RFID work,” AB&R, 15-Apr-2022. [Online]. Available: https://www.abr.com/what-is-rfid-how-does-rfid-work/. [Accessed: 23-Feb-2023].

[2]“1.2 A high PSRR low-dropout linear voltage regulator LDL1117” Accessed: Feb. 24, 2023. [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/datasheet/group3/0e/5a/00/ca/10/1a/4f/a5/DM00366442/files/DM00366442.pdf/jcr:content/translations/en.DM00366442.pdf

[3] A. Guirao and P. (A. PCBArtist), “ESP32 CP2102 programmer schematic,” PCB Artists, 10-Dec-2022. [Online]. Available: https://pcbartists.com/design/esp32-cp2102-programmer-schematic/. [Accessed: 29-Mar-2023].

[4] “Print esp32thingplus - sparkfun electronics.” [Online]. Available: https://cdn.sparkfun.com/assets/learn_tutorials/8/5/2/ESP32ThingPlusV20.pdf. [Accessed: 03-May-2023].

[5] Espressif Systems, “ESP32-WROOM-32 Datasheet,” Version 3.4, February 13, 2023 https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[6] Thomascountz, “Arduino tone for ESP32,” *Thomas Countz*, 21-Feb-2021. [Online]. Available: https://www.thomascountz.com/2021/02/21/arduino-tone-for-esp32. [Accessed: 03-May-2023].

[7] “SparkFun Simultaneous RFID Reader - M6E Nano,” SEN-14066 - SparkFun Electronics. [Online]. Available: https://www.sparkfun.com/products/14066. [Accessed: 23-Feb-2023].

[8]“Simultaneous RFID tag reader hookup guide,” Simultaneous RFID Tag Reader Hookup Guide - SparkFun Learn. [Online]. Available: https://learn.sparkfun.com/tutorials/simultaneous-rfid-tag-reader-hookup-guide/all. [Accessed: 23-Feb-2023].

[9]“Altsoftserial Library,” *PJRC*. [Online]. Available: https://www.pjrc.com/teensy/td_libs_AltSoftSerial.html. [Accessed: 03-May-2023].

[10] “IEEE code of Ethics,” IEEE. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 12-Feb-2023]

Appendix

Requirement	Verification
LDL1117S50R must be able to safely take an input of 12V from the battery and step down to 5V +/-5% (4.75V-5.25V) and appropriate currents for each component: - IR emitter: 90-110mA - RFID receiver: 100-200mA - **IR emitter eradicated** a. Ultrasonic sensor was adopted	1. Use an oscilloscope/multimeter to probe and display the voltage output of the 12V battery. 2. Use an oscilloscope/multimeter to probe and display the voltage and current output of only the buck converter. **-Successful** 3. Observe if current and voltage outputs are within desired voltage range. **-Successful**
LDL1117S33R must be able to safely take an input of 12V from the battery and step down to 3.3V +/-5% (3.135V-3.465V) while providing 500mA-660mA to supply power to the ESP32 and 1-5mA to the IR receiver	1. Use an oscilloscope/multimeter to probe and display the voltage output of the 12V battery. 2. Use an oscilloscope/multimeter to probe and display the voltage output of only the buck converter. **-Successful** 3. Observe if current and voltage outputs are within desired voltage and current ranges. **-Successful**
A 12V battery must be able to provide power to the ESP32, IR emitting LEDs, IR receivers, and RFID receiver simultaneously.	1. Use an oscilloscope/multimeter to prove and display the voltage output of the 12V battery. 2. Use an oscilloscope/multimeter to probe and display the voltage and current inputs to a device while all others are also connected. **-Successful** 3. Observe if current and voltage outputs are within desired ranges (+/- 5% tolerance) for each device while all devices are connected. **-Successful** 4. Repeat the process for each device. 5. Play a test game, utilizing all devices to determine if all sensors and receivers are working as expected. **-Successful**

Requirement	Verification

1. The IR emitting LEDs must be able to emit strong enough signals such that the IR receivers output voltage within +/- 5% of their supply voltage of 3.3V (3.135V-3.465V. **-Requirement eradicated** a. Didn't implement this due to ultrasonic sensors being adopted	1. Place the IR emitter and receiver pair 6 inches apart (same length as the diameter of the hole). 2. Supply 5V to IR emitting LED circuits and 3.3V to the IR receivers and probe to display the output pin of the receiver with an oscilloscope. 3. When an IR emitter and receiver are unblocked, connection is established and the receiver's output should be in the range of 3.135V-3.465V. 4. Cover the emitter or receiver with a fabric to observe change in output. Voltage output should now be in the range of -0.3V-+0.3V
2. Bags that have fallen into the hole should not be detected by the RFID receiver with 80% accuracy. **-Requirement eradicated** a. Didn't implement this idea to due ultrasonic sensors being adopted	1. Drop a singular bag into the metal bucket inside the hole 2. Check the Universal Reader Assistant to run the READ EPC command. 3. If a tag is detected by the READ EPC command, the hole has not been successfully shielded and needs further RF shielding. If a tag is not detected, the system is working as intended. [8]
3. The RFID tags must be able to be read through both the 1 cm fabric material of the bag and the 1 in. plastic material of the board with 90% accuracy.	1. M6E Nano board will be connected to a laptop computer for power by plugging in the serial breakout board via USB. 2. The Universal Reader Assistant program will be run, which is a Windows program designed to help test capabilities of M6E-Nano. 3. The Universal Reader Assistant will connect to the board, sending a ping to verify the module's existence. 4. The tag will be fastened in the game bag, This game bag will be placed under a sheet of plastic approximately 1" thick, and the RFID reader will be placed on top. **-Successful** 5. The READ EPC function will be run. If the tag is correctly detected, the function will return with "RESPONSE_SUCCESS" and the EPC (Electronic Product Code) will be stored in the array. **-Successful** 6. The game bag will be moved in increasing intervals (1", 2", 3" and so on) until it is no longer able to be read by the RFID reader, and this maximum read distance will be recorded. [8] **-Successful**

	*See also Verification Procedure #1 for control unit for a similar procedure setup to test RFID reader functionality.
4. At least 8 RFID tags must be able to be read simultaneously by the RFID receiver.	1. Similar process will be used as the previous verification method to set up the RFID components. 2. Another RFID tag will again be placed in a separate game bag and placed under an approx 1" sheet of plastic which will be underneath the RFID reader. **-Successful** 3. Using the Universal Reader Assistant set to constant read, the READ EPC function will be used to detect the bag. **-Successful** 4. If that bag is detected, another RFID tag will be added in a separate game bag. **-Successful** 5. The number of bags will keep increasing until the RFID reader either fails to detect a bag or until the number of bags reaches 8 total. **-Successful**

Requirement	**Verification**
1. RFID reader must be able to correctly receive signals from each of the 8 RFID tags that exist within the game bags.	1. M6E Nano board will be connected to a laptop computer by plugging in the serial breakout board via USB. 2. The Universal Reader Assistant program will be run, which is a Windows program designed to help test capabilities of M6E-Nano. 3. The Universal Reader Assistant will connect to the board, sending a ping to verify the module's existence. **-Successful** 4. A tag will be placed within a game bag and placed next to the M6E Nano. 5. The READ EPC function will be run. If the tag is correctly detected, the function will return with "RESPONSE_SUCCESS" and the EPC (Electronic Product Code) will be stored in the array. [8] **-Successful**
2. ESP32 must communicate with the RFID receiver via GPIO pins.	1. ESP32 will be connected via jumper cables before soldering. **-Success** 2. An Arduino sketch will be uploaded to the ESP32 3. The RFID sensor will echo the test to communicate whether the test string has been received or not. **-Success** 4. The output from the Arduino IDE will confirm if communication is established **-Success** 5. The Arduino sketch will then be edited to consider multiple bags. **-Success** 6. Repeat Step 3. 7. Output from Arduino IDE will confirm that communication

	is established and bags are distinguishable. **-Success**

Requirement	**Verification**
1. Receive data from the ESP32 module via Bluetooth.	1. A test signal will be created using the Arduino IDE to communicate to the ESP32 module once the ESP32 module Bluetooth connection has been verified to work. **-Successful** 2. The signal will be sent to the app via Bluetooth to ensure the signal is received and the app is able to connect to Bluetooth. **-Successful** a. Test data received is echoed onto the app.
2. After marking the end of the round, the app can produce game scores within 60 seconds.	1. Working with the procedure from above, use a stopwatch to measure the time it takes from the information to be sent from the board electronics to the mobile app. **-Successful** a. If too slow, latency must be low.
3. The app is responsive to user inputs.	1. Upon pressing the "End Round" button, the app should send a signal back to the board system to score the game. **-Successful** 2. We will code the app to send a "1" as data back to the ESP32 when the "End Round" button is pushed. **-Successful** 3. If the ESP32 receives the 1 signal, the app has shown that it can respond to user inputs and both read and write inputs. **-Successful** 4. In the final product, the user should be able to open up past game history (up to the last two or three games) and the score of the game. This will be checked against the known score of the game to ensure it is accurate. **-Didn't pass** a. Lack of time to implement

Description	**Manufacturer**	**Quantity**	**Extended Price**	**Link**
Simultaneous RFID Reader	SparkFun Electronics	1	$224.95	**Link**
UHF RFID Tags-Adhesive (5 pack)	SparkFun Electronics	2	$2.95	**Link**
LDL1117S33R (3.3V Voltage Regulator)	STMicroelectronics	1	$1.16	**Link**

LDL1117S50R (5V Voltage Regulator)	STMicroelectronics	1	$1.16	**Link**
IR Emitting Diode	RS Americas	2	$0.38	**Link**
Siliconix / Vishay TSOP34838- Infrared Receiver	RS Americas	2	$1.20	**Link**
Gater, Cornhole, Light Up and Standard Available, Easy Storage, Light Weight Perfect for Outdoor and Indoor Play	EastPoint Sports Go!	1	$29.99	**Link**
Red LED *	Rohm Semiconductor (Already Owned)	1	$0.53	**Link**
Micro USB to USB cable *	Best Buy Essentials (Already Owned)	1	$6.99	**Link**
LP-35F Plastic Enclosures for Electronics	Polycase	1	$4.31	**Link**
USB to UART Bridge	Silicon Labs	4	$19.00	**Link**
ESP32 WiFi-BT-BLE MCU Module / ESP-WROOM-32	Adafruit	4	$15.80	**Link**
RFID Blocking Fabric	SpecialtyFabric	1	$10.73	**Link**
Bluetooth LE Asset for iOS, tvOS, and Android	Unity Asset Store	1	$20.00	**Link**
12-Volt 9 Ah UPS Battery Replacement for APC BE550G	The Home Depot	1	$24.99	**Link**
Battery Screw Terminal	CUI Devices	1	$0.70	**Link**
Schottky Diode	Diodes Incorporated	2	$0.90	**Link**
Micro USB Type B connector	GCT	4	$3.24	**Link**

S8050 NPN BJT	UMW	4	$0.88	**Link**
4.7uF Capacitors	Samsung Electro-Mechanics	6	$0.60	**Link**
1 uF Capacitors	Samsung Electro-Mechanics	4	$0.40	**Link**
0.1uF Capacitors	Samsung Electro-Mechanics	4	$0.40	**Link**
22uF Capacitor	Samsung Electro-Mechanics	4	$0.52	**Link**
10k Ohm Resistor	YAGEO	6	$0.60	**Link**
100 Ohm Resistor	YAGEO	6	$0.60	**Link**
UHF RFID Antenna	Sparkfun	1	$42.95	**Link**
HC-SR04 ultrasonic sensor*	OSEPP Electronics LTD (already owned)	1	$3.59	**Link**
SparkFun Thing Plus	SparkFun (already owned)	1	$22.50	**Link**

* The indicated component is already owned and does not need to be purchased.

Table 7: Cost Analysis Table