Lego Battle Bots

John Chambers, Kyle Graham, Kenneth Hunter

***Abstract*—Despite the enjoyment the modern world has experienced from demolition derbies and battle bot contests, a common issue for these events is the inability for all competitors to participate in subsequent matches. Instead of simple destruction, this project provides Lego battle bots capable of multiple rounds of combat. The battle bots are comprised of a Lego shell that has parts capable of coming off of it, and is armed with either a sword, axe, or lifter to dismantle other Lego shells in a combat game.**

I. Introduction

Modern demolition derbies, especially those involving robots, have seen a resurgence within the past few years. Although these events are entertaining, they only leave a victor with a semi-functional system. In an attempt to create entertaining devices for home use, there is still a need for systems which can both be demolished and reused. Combining software and hardware, there is a potential solution to provide both the entertainment and the destruction that these demolition events provide to observers and participants. This project involves the creation of "battle bots" which are able to be reconstructed after each individual event. In this way, devices are still able to tear each other apart and be used multiple times without the need for new components.

Instead of simple destruction, this project provides the same level of entertainment as the previously mentioned events while being able to reconstruct devices after a match has been won or lost. In this way, participants are still able to experience the destruction common to related events, and reuse their creations continuously. Each component within the battle bot device is able to be removed, thus disabling it. A bot is not declared dead, however, until all of its armor panels have been removed. Each battle bot has a total of three armor panels that can be further connected to other Lego bricks, thus allowing for a creative approach to armor usage.

This project provides user configurable battle bots. Each bot is be powered by a Raspberry Pi and controlled through a smart phone app over Bluetooth. Bots are interconnected through a local network, thus allowing them to share data. Bot assembly includes the form of locomotion, some armor panels, and weapons. Once a bot is started, the provided Android app allows a user to select the appropriate weapon and begin combat. Each bot is then controlled through an individual's smart device, and continues to remotely update until it is defeated.

The game begins when users assemble their battle bot out of a selection of premade armor panels, weapons, and form of motion. Each type of accessory, armor, weapon, locomotion, will all fit only into slots designed for them, to prevent incorrect information being sent to the app. This is enforced through the project's custom quick connect mechanism. Once the battle bot is assembled, it will be paired with the app, and the user will select what weapon and locomotion is attached in drop down menus. Once these steps are done, a bot is entered into battle.

Within the battle bots Android app, users will move their bots around while trying to use their weapons to knock the armor panels off their opponent. How the user knocks off armor depends on the weapon. A spinning blade or flail will remove them from striking the opponent, while a bulldozer blade will ram directly into their opponent. Armor panels make up the health of the battle bot. When a bot loses all of its armor panels, it stops and the user is alerted that their game is over through the Android app. The last player alive wins, and is able to exercise their bragging rights.

II. Background

Each individual battle bot consist of a Raspberry Pi with a motor driver system, and an accompanying Android app. The finalized project provides both a functional battle bot, and an intuitive Android application. Both communicate through several custom APIs to allow for quick and seamless use. In short, each battle bot is capable of: movement through Bluetooth control, detection of various parts and modification of Android application info, degradation of functionality correlating with damage sustained, and participating in a fully functional game allowing players to compete using their own custom creations.

Creating the aforementioned battle bots required a system that is small enough to drive the logic needed for proper destruction and part detection. It is fortunate, then, that pocket size computers have made many advances within the last few years. In particular, a Raspberry Pi not only contains the necessary interfaces to communicate with custom components, but it is also able to provide two steady interfaces for communication between itself and other devices: Bluetooth and WiFi. Each battle bot is controlled through a mobile phone app over Bluetooth 4.0. This controller not only supports movement of an individual bot, it will also handle activation of any custom parts applied to an individual vehicle. Custom parts are simply components that provide a certain method of destruction to another participating bot. These components directly communicate with the Raspberry Pi to indicate their presence on a device, and their current functionality level. These functionality levels will affect the performance of each component, and the bot itself. Once a device has lost all of its armor, it will cease to function and the final functioning device is declared the winner.

Raspberry Pi devices have become commonplace within the last five years. Their simple interface for General Purpose Input and Output (GPIO) makes interacting with other components simple, and allows for various functionality to be added to the small computer. This supported the project perfectly, as each individual battle bot requires a networking interface, Bluetooth capabilities, and enough power to drive a bot's four wheels. Utilizing various information from the following projects, each battle bot is individually controllable, and self contained.

Convention for remote control uses radio signals, but for short distances between user and vehicle a smart devices Bluetooth adequately covers a large room. When working indoors, devices can take advantage of WiFi networks instead of either radio signals or Bluetooth. The WiFi RC Car project demonstrates the ability for any WiFi compatible device to control an RC Car [1]. As the Raspberry Pi is both WiFi compatible and able to act as an RC device with proper components, this makes it the core of many projects. This same project also highlighted the necessary information to provide a functional server module for bots to communicate over. Due to the provided information, the battle bots project also uses a Raspberry Pi as the core component running the battle bot. This provides numerous options for how the user will connect the app to the battle bot and receive updates. While the main method of controlling a battle bot is through Bluetooth, a remote server is still used to handle game states. Both of these are fully customizeable in an effort to provide users with flexibility based on the location of use.

The Raspberry Pi contains GPIO pins for connection of various modules, including multiple motor controllers and other sensors. These pins are able to provide different pulse-width modulation signals to the attached motors, but motor controllers provide power to the motor itself. In a project by K. Dumbre et al, a Raspberry Pi controls multiple motors and sensors powering a remote vehicle. This remote vehicle had motors for movement as well as an arm to interact with objects in the environment and the user observed all of this through a camera mounted on the robot [2]. In a similar fashion, we have created a small bot controlled through a Raspberry Pi that uses traditional wheels for movement. It is capable of moving forward, backwards, left, and right. These four degrees of movement are handled with a singular motor and servo, leaving enough pins for all other components. Although an arm is not used for interaction, similar principles have allowed the project to read armor state, and control weapons.

In a Raspberry Pi controlled arm project K. Premkumar et al, created a robotic hand meant to replicate movement of a human hand. In this project an Android app was created and it would send a set of predefined signals to the robot hand. The user could select from several options and the hand would preform said actions [3]. In this project we created predefined commands for several of the weapons. The accompanying app provides a specific interface to manipulate bot movements. For example, weapons all function from a primary button within the Android application. For the user, they would tap the weapon button in the app, and the app would send signals for the motor to raise up to its set maximum pulse-width. As long as the weapon button is held it remains at max position, when released the app handles the signal to lower to minimum height. The accompanying Android application features options to connect to a battle bot, store game session data, and features intuitive touch controls and buttons. Each individual button allows a user to control the battle bot without needing to look at the app.

III. Project Implementation and Performance

A. Hardware

Each bot has four main hardware components. The first is the Raspberry Pi, the Raspberry Pi functions as the control for every part and the relay between the server, app, and hardware. The second is the power module, this is a battery and a breakout module to allow multiple devices to pull from the same current source. The battle bot must be mobile and trailing power wires severely limits that ability. Modern battery packs are readily available and easily provide the require outputs in both voltage and current. These usually come in the form of mobile chargers for phones and tablets. The easiest to use are those with multiple USB ports and support multiple device charging at once. This allows the Raspberry Pi its own direct supply and the second goes to breakout module. The breakout module is useful to connect the different motors to. The supply provides enough current to run all three motors from one USB port but not enough to run the motors and the Raspberry Pi. The third module is the motors, these can be grouped easily because they are all pooling current and voltage from the same power module. The base battle bot require three motors, two servo-motors and one DC motor. The two servo-motors are split one for the weapon and one for the steering. The DC motor is the main drive and needs to be connected to a motor controller and a voltage stepper. The motor controller allows the motor to run in both directions controlled by the Raspberry Pi, and the voltage stepper bring the 5 volts breakout module to run at 12 volts for the line to the motor controller, see Fig. 1. The fourth and final part of the battle bot is the shell. The shell is mostly just a hard container to protect the more delicate electronics but it does have the armor circuit attached directly to it.

The power module and the Raspberry pi fit up into the shell and the shell is closed up. The core of the shell is a 3d printed frame to cradle the Raspberry Pi with Lego's attached to it that the walls of the shell attach to. The motors all have Lego's attached to them so they can be attached to the shell. The wires connect to a quick connect system that connects allows power and signals to pass through the shell of the battle bot. The quick connects are wires attached to magnetic connectors and the motors or the Raspberry Pi. The quick connects on the shell are organized into patterns that prevent incorrect attachments of motors. Each motor has its own corresponding magnetic pattern. These magnets are glued into Lego blocks to preserve those patterns. The quick connect for the servo-motors have three connections, +5v, ground, and signal. The DC motor has just two attachments positive and return. The armor that attaches to the shell is Lego's glued together to create larger

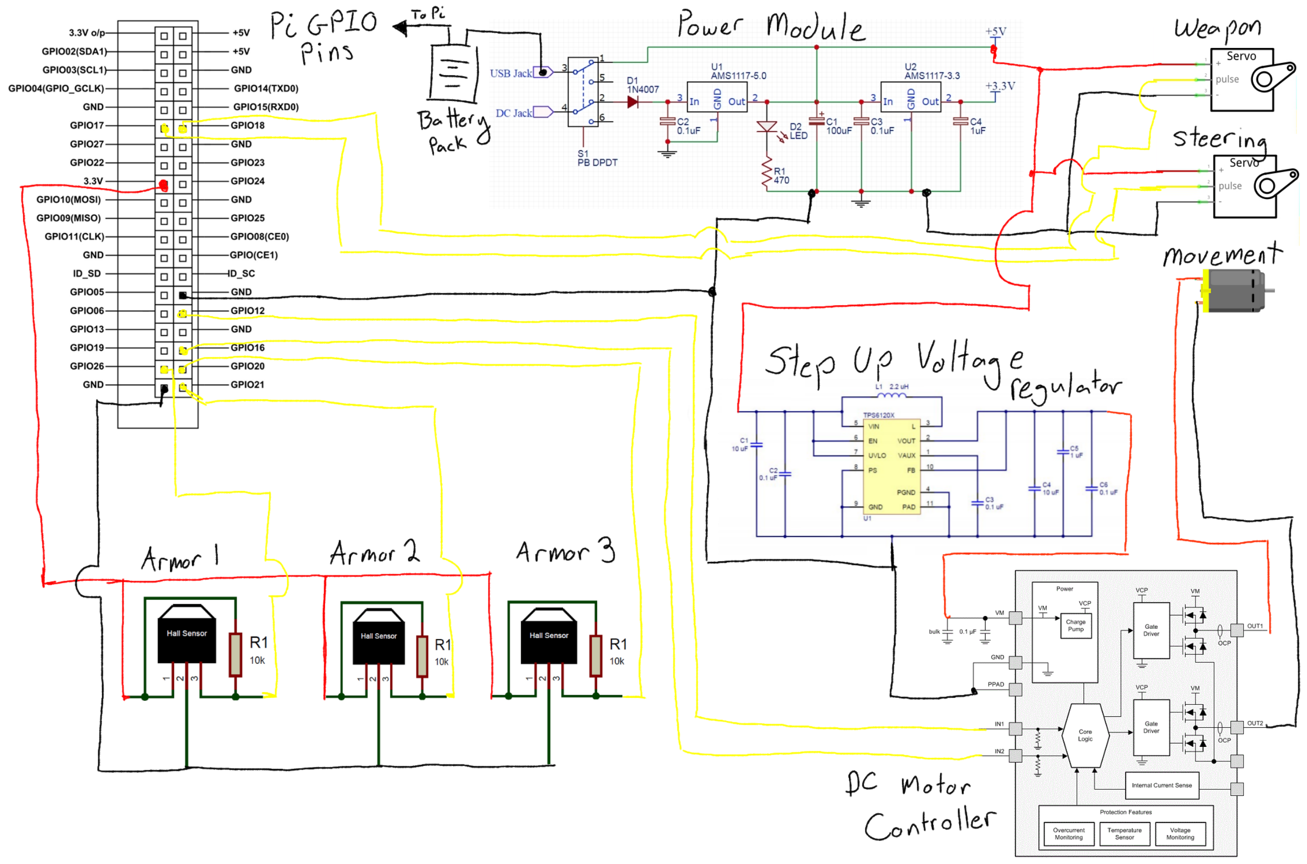


Fig. 1. Schematic

blocks that are easier to remove during combat. The armor has a magnets embedded into it to activate the armor circuits inside the shell. The armor circuit is a hall effect sensor with a 10k ohm resister between power and out pin to pull up the voltage, this causes the pin to go low when the armor is attached to the shell. Additional passive elements can be attached to the shell, such as additional wheels for stability or mini figures for aesthetics.

Development of the battle bot was different iterations of the shell. The first version protected only the Raspberry Pi leaving all the other parts out. The second version contained all parts but the limited space made packing all the electronic inside difficult. The third and final form gave room for all the parts to easily fit in the shell and to have an insulator between parts to prevent any cross currents damaging parts. The quick connects were developed along the way as originally a way to make testing easier and kept for user convenience in the final version.

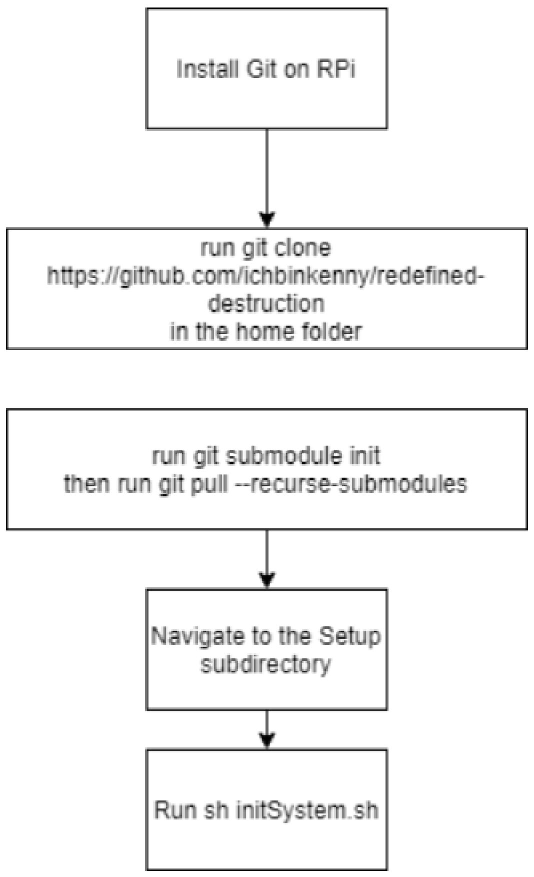
The overall cost of a bot came to $108.58. This is about where we planned to still be considered acceptable. The price could be dropped quickly if any of the parts can be bought in bulk, such as if someone were to assemble several at once. The majority of the expense is in the Raspberry Pi and the battery. The breakdown is for the basic unit which is a shell, one weapon, steering, three armor panels, drive motor, and all connections needed.

TABLE I
AVERAGE BATTLE BOT COST BREAKDOWN

Price List		
Part	Quantity	Price
Raspberry Pi	1	$35.00
Lego Sheet	1	$10.00
Lego Wheels	1	$3.50
Lego Weapons	1	$0.15
3D Core	1	$0.80
SG90 Servo Motor	2	$1.80
DC Motor	1	$1.50
Motor Driver	1	$10.37
Power Breakout Module	1	$7.50
Voltage Step Up	1	$7.74
Magnet	22	$0.11
Battery	1	$20.00
USB Cable	2	$3.00
Total Price		$108.58

B. Software

In order to reduce the complexity of creating a new battle bot, a Setup module was created to provide easy implementation of both a bot client and a bot server.


Install Git on RPi
run git clone https://github.com/ichbinkenny/redefined-destruction in the home folder
run git submodule init then run git pull --recurse-submodules
Navigate to the Setup subdirectory
Run sh initSystem.sh

Flow.PNG

Fig. 2. Necessary Setup Flow for Implementing a Battle Bot System

The process of creating a new bot or server thus follows the flow outlined in Fig. 2, and results in a fully functional system provided both a client and server are present. Throughout the process, a user is first asked to select the type of system they are trying to implement: client or server. Once this is done, several calls are made to apt-get to install the necessary packages for the specified system. After ensuring these necessary libraries are installed, the appropriate service files are moved into their correct spots and a simple reboot will launch the client or server. Alternatively, one can move into the Bluetooth folder and run "sudo python3 BluetoothControl.py" to launch their client immediately. This setup is done entirely through a bash script executing the requisite commands on the Raspberry Pi.

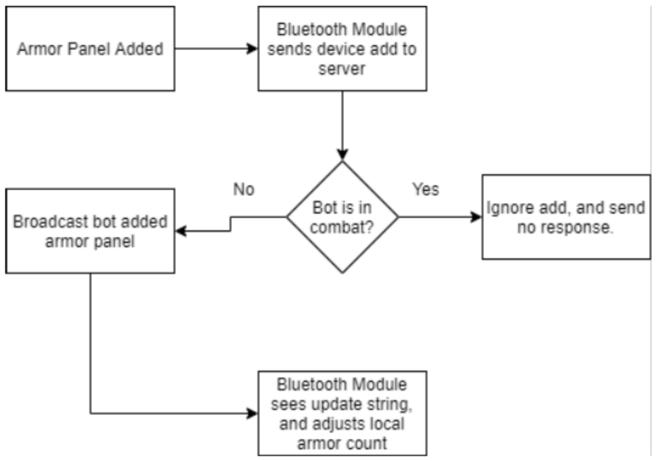
Each individual battle bot performs communication with an Android application and a remote server. In order to achieve proper communication between the Raspberry Pi and the Android application, this project used Bluetooth and conformed to the serial port protocol (SPP). To send data to other participants, however, this project made use of a remote server that could provide updates to bots directly. Without this remote server, it would have been much more tedious to properly update individual battle bots. Instead, the remote server allowed for all code to logically fit within four distinct modules: Bluetooth, Movement, Networking, and Setup. As such, the following descriptions involve each module's logic and functionality.

As previously mentioned, Bluetooth would be the only means of communication between a battle bot and a phone. In order to achieve communication between the Raspberry Pi and the Android application, the Bluez library would be utilized with a reliance on Bluetooth's SPP. As SPP requires a server and a client, the battle bot would function as a server to receive commands from its android clients. Once a client was found, the Raspberry Pi would immediately begin to send it's armor status back to the Android application. This allowed for sub-100ms updates on the Android device for a bot's health. Before the first armor update would be sent, however, a separate process would be spawned to read any available data sent from the Android client. Thus, messages would be send to and from the battle bot as fast as possible and prevent unnecessary lag between the Android application and the bot. Although this was not necessary for armor updates, which are currently delayed, it was imperative to allow for proper movement of a bot itself. Any introduced delay would have made controlling a bot sluggish, and unenjoyable.

Communication delay within the Lego battle bots project was the number one challenge to handle. Although Bluetooth's delay was managed, we also needed a way to ensure movement and GPIO manipulation would not interfere with local or remote communication. As such, each individual movement component was created within its own python module. Although each module controlled a specific component on the battle bot, their baseline implementation followed a common flow: first read the input piped from the main Bluetooth control file, and then parse the command to produce a result. For example, the front wheels module will first await input on the input pipe that is shared between it and the Bluetooth module. Once numeric data is received, the front wheels module then adjusts the pulse-width modulation of the servo controlling the front wheels, providing rotation in the correct direction. The remaining movement modules also follow a similar pattern to provide the correct outputs on their assigned GPIO pins.

Perhaps the most critical part of the battle bots code was implementing armor state changes. Although most movement components require input to describe output, the armor panels required precisely the opposite: any new GPIO change had to be sent as input to the main Bluetooth module. In order to handle this, armor status is run immediately after the Bluetooth module starts. Unlike the movement modules, the input between the Bluetooth module and the armor module is not shared. Instead, the armor module's output is sent to the Bluetooth module's input to allow for reading armor updates. Fortunately, armor updates are quite small and consist only of the string "False:False:False" when no armor is connected. Connecting one piece of armor provides the Bluetooth module with "True:False:False", where the True value appears in the armor slot that is occupied. These simple strings are then easily parsed, and the resulting state is then sent to a connected Android client to update a bot's health.

Although individual bots can handle their armor state, they lack the ability to handle a global game state. As such, a remote server became necessary to allow for game state to be synced between each individual battle bot. The server is another python module that can register clients and update game state. Currently, the server's biggest function is to ensure that users do not try to reattach armor panels once their bot has entered combat. To handle this, however, much of the flow between a bot and its armor panels needed to be handled on the server side.



Armor Panel Added
Bluetooth Module sends device add to server
Bot is in combat?
No
Yes
Broadcast bot added armor panel
Ignore add, and send no response.
Bluetooth Module sees update string, and adjusts local armor count

Fig. 3. Armor Connection Flow

As seen in Fig. 3, armor updates must first travel to the remote server to ensure they have not been added after a bot has begun combat. If and only if a bot is not in combat, the server then broadcasts a message indicating an add and the client id where the add occurred. Each message takes the format of "statusCode: addedItem clientId". These status codes can be expanded on, and allow for a consistent way for clients to observe each other. This not only preserves game state, but it allows for future modification if a skillful player wishes to change when items are allowed to be added and removed. Finally, the server also handles abrupt client disconnects. If a bot suddenly loses power, or is temporarily unable to communicate, its current socket is closed but its IP is saved with its client id. Thus, when a bot disconnects and then reconnects its previous state is still used to handle game interaction. This is yet another method to ensure no armor manipulation occurs, but it also functions as a way to make connection more robust.

Overall, the software performance on the Raspberry Pi has proven to be reliable and quite robust. Setup is quite trivial, and allows access even to those who may be uncomfortable with Linux-style systems. Network performance is reliable, and perhaps the only real issue with communication comes from the discoverability of a bot when using the accompanying Android application. This process can take upwards of ten seconds, but is largely due to the Android application itself. If a different Bluetooth scanner is used, the device shows up almost instantly.

C. Android App Functionality

One of the goals of this project was to have a seamless controller experience. We wanted the Lego Battle Bots to be controlled with a very simple joystick and button functionality that fits into the palm of your hand and has fast and responsive communication. We also wanted it to be easily accessible and not add to the cost of the project, so we opted to build a simple and stylish Android app that connects to the bots via Bluetooth and controls movement and attack functions. Our goals for this app were to connect to a battle bot, control motor speed, directional movement, and attack commands, save game data to a player's account, and keep track of armor so that when the battle bot loses all of its armor, the game ends. We used Android Studio to design this app, with Kotlin and Java as our programming languages of choice. In Android Studio, the display of each screen is represented by an xml file, which is connected to a code file called a Fragment. We designed our app to have a Fragment for each screen of the game.

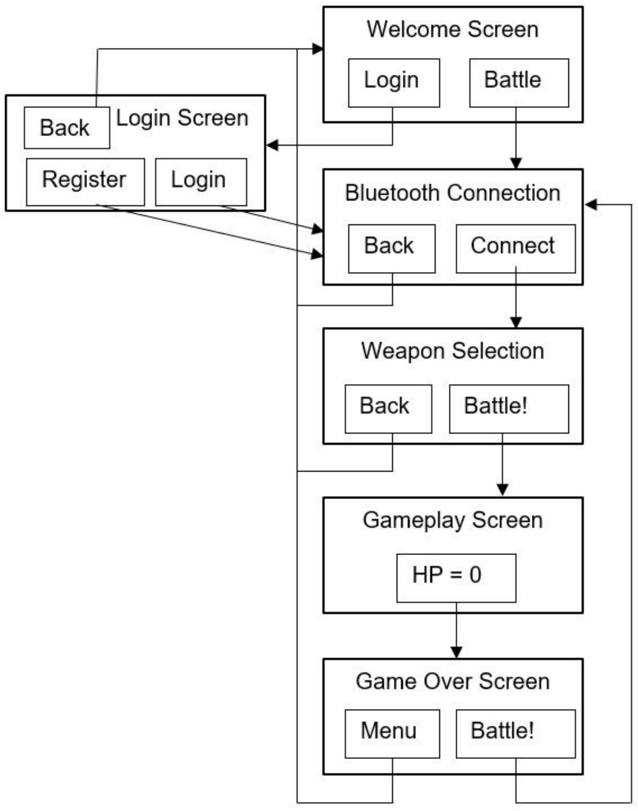


Fig. 4. How the user navigates to each screen of the Android app.

A simple flowchart of the Android app's navigation is shown in Fig. 4. When a user opens up the app, they are greeted with a cleanly designed welcome screen, shown in the Appendix, Fig. 6. They have the option to start playing or login to a new or existing account. If they choose to login, they are taken to a user account screen, shown in Appendix, Fig. 7, that gives them the option to register or login to whatever account corresponds to the text in the email and password fields. There is some simple error checking on those fields, for example, the user is required to make their password at least 6 characters, and also use an email address format for their email. If a valid email does not exist in the database that is tied to our Android app, they cannot login and should instead register that account.

For the user database, we opted to use Google's web-based Firebase platform. It provided us with an easy and free online location to store the simple data metrics that we needed for our user account system. Because Android Studio is partnered with Google, Firebase integration into the app was very easy. We simply set up a new database on Google's Firebase console, then connected the app by entering an app name and the Debug signing certificate SHA-1, which is unique to every Android Studio app. Google provided the Firebase configuration files. We only had to add commands to import those files into the build.gradle file in our app, along with the latest Firebase sdk file. In our current version of the Lego Battle Bots database, we store user login and password information, with native password encryption by Google, so no one is able to see user passwords. In future versions, we will be able to store more info such as gameplay sessions, user stats, and a leaderboard to make the competitive aspect of Lego Battle Bots more dynamic.

Once the user is logged into their account, they are taken back to the welcome screen. The user will notice their account name is displayed on all the rest of the game screens. When the user presses 'Battle!' they are taken to the Bluetooth connection screen where they can connect to a nearby Lego Battle Bot, shown in Appendix, Fig. 8. The code behind the connection screen scans for Bluetooth signals and when it finds one, it begins initializing a new battle bot object and begins the loop of sending and receiving data to the bot. Once the user selects 'Connect', they are taken to the next screen where they can select their weapon, as seen in Appendix, Fig. 9. Each weapon corresponds to different movement of the battle bot's weapon motor, so in the app, the user should select the weapon that corresponds to whatever weapon was physically placed on the bot. Once a weapon is selected, the app will include that weapon's designated weapon code in the game loop to be sent back to the Raspberry Pi and translated into the proper motor movement that makes the weapon attack.

Once the weapon is selected, the user can click the "Battle!" button and finally start to control their bot in the gameplay screen, as seen in Appendix, Fig. 10. This screen features a joystick for movement controls, buttons to send attack commands to the bot, and a progress bar that displays the bot's current armor level, labeled as "HP". The maximum armor level is 3 as there are 3 armor connection plates on the current version of the lego battle bot. The attack button increases an attack integer every time it is pressed, this attack integer is sent to the bot as a command to perform whatever attack the user has selected for the amount of times the button is pressed. This attack integer was implemented just in case any button presses were not sent to the bot, but because of the reliability and speed of our bluetooth connection to the Raspberry Pi, this issue doesn't come up during normal gameplay sessions. There was a second attack button added as a stretch goal to introduce more variety and dynamics to a battle. This second attack is not implemented in the current version of Lego Battle Bots.

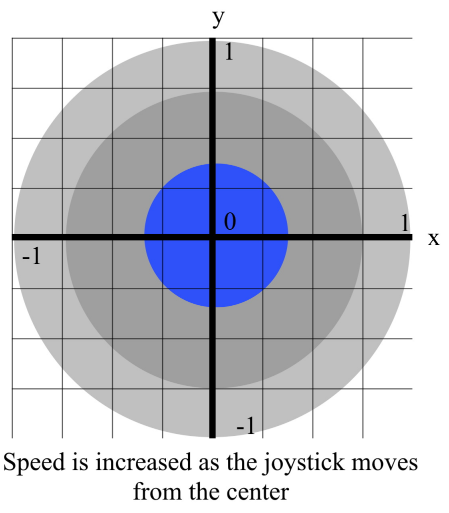


Fig. 5. Functionality of the joystick on the bot controller screen

The joystick was implemented as a SurfaceView object in Android Studio. This SurfaceView could be designed to look like a physical joystick and configured to move with user input. As seen in Fig. 5 the joystick frame is structured similar to a Cartesian plane, with an x and y-axis that go from -1 to 1, crossing at the origin. This setup contains all the information needed to control the bot's movement. The joystick computes the location on the plane as the user drags it with the Pythagorean theorem, with a maximum value of 1,1 and a minimum value of -1,-1. This x,y data is sent through bluetooth to the Raspberry Pi within the bot and converted to motor commands. When the joystick is at the origin, the bot stays still. When the joystick is dragged out, it increases motor speed proportional to the distance from the joystick origin. The x-coordinate of the joystick tells the bot to steer left and right. The bot drives right when the joystick is in the positive x quadrant and left when the joystick is in the negative x quadrant. Those configurations are reversed when the bot is driving backwards, which occurs whenever the joystick is in one of the negative y quadrants.

The joystick data, as well as the attack commands, are constantly sent to the connected battle bot with a game loop. That loop runs every 10ms, which is definitely fast enough for a seamless control experience. Also being communicated through a different loop is the armor status of the bot. Once the bot detects that an armor piece has fallen off, the armor status that is constantly being sent to the Android app is decreased and the app computes the progress value of the HP bar based on that armor status, giving near instant updates on the bot's health as it is fighting. Once an armor piece falls off, the app reflects that by decreasing the HP bar. Once all the armor has fallen off, the app will send a command to shut down connection to the bot and it will stop sending updates, disabling all movement and attack functionality. The app will show the game over screen shown in Appendix, Fig. 11, where the user has the option to start a new battle, or go to the welcome screen where they can log into a different user account, if desired.

Our Android app can be installed on any modern Android phone running Android Pie or below, allowing that person to control and play with any battle bot. Each app is tied only to one battlebot so there is no restriction on the amount of bots fighting at one time. It's a simple, inexpensive functionality that is complementary to the simple, inexpensive Lego Battle Bots. We achieved each of our goals for making the app responsive, easy to control, and easy on the eyes. In the future, we can add leaderboards, weapon usage statistics, Kill/Death ratios, and store battle sessions to provide users with more dynamic profiles on their battle strategy. For now, the functionality of using a simple, stylish app to connect, control, and battle your bot has been achieved. And once a battle bot is defeated, it isn't entirely destroyed. All a player has to do to start a new battle is place the armor back on the bot, choose their weapon, reconnect their bot to the app, and start a new battle.

IV. Conclusion

This project fills the space for both Lego enthusiast and individuals who enjoy Raspberry Pi projects. It is a designed to be a game for ages 14+ with no real cap on upper range as individuals with electrical or programming knowledge can further expand the weapons, armor, and locomotion options. With an Android app and Bluetooth connectivity, there is minimal equipment required to seamlessly control a Lego Battle Bot and store user data. With robust Lego parts they will be able to be ripped apart and built back up many times. Should some part wear out or break its modular design with make it cheap and easy to replace.

Appendix

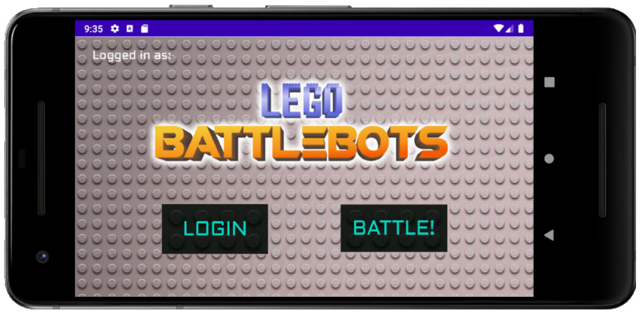


Fig. 6. The default screen when first opening the Lego Battle Bots app

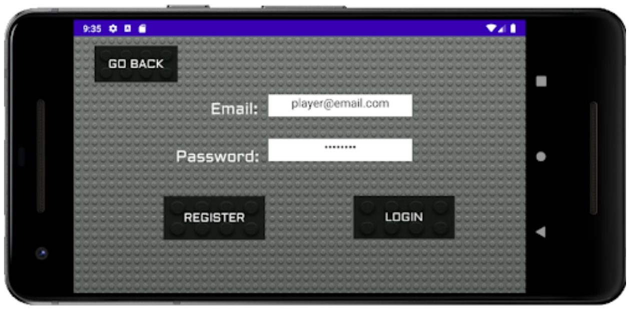


Fig. 7. The login screen on the Lego Battle Bots app

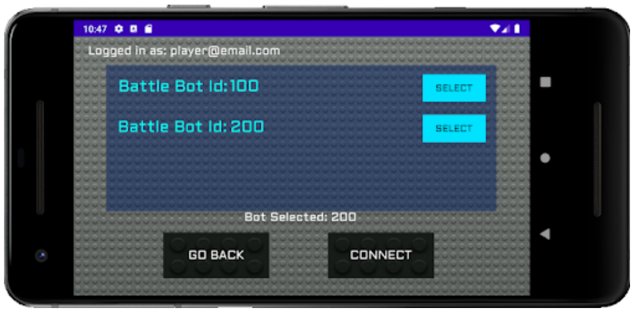


Fig. 8. The bluetooth connection screen on the Lego Battle Bots app

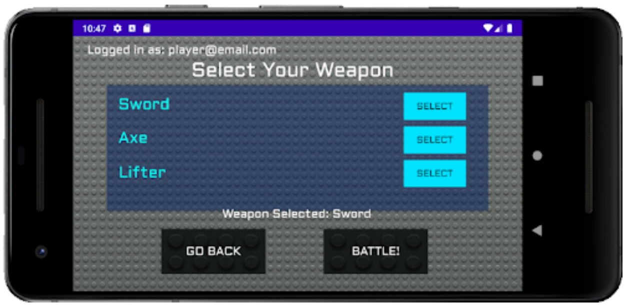


Fig. 9. The weapon selection screen on the Lego Battle Bots app

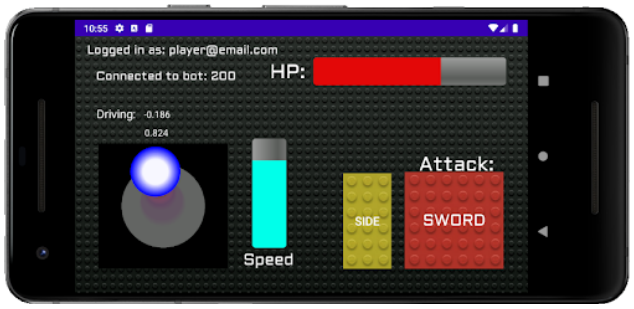


Fig. 10. The bot control screen on the Lego Battle Bots app

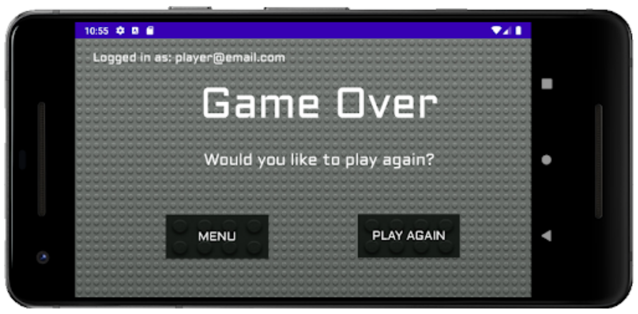


Fig. 11. The game over screen on the Lego Battle Bots app

References

[1] N. A. Zaini, N. Zaini, M. F. A. Latip, and N. Hamzah, "Remote Monitoring System Based on a Wi-Fi Controlled Car Using Raspberry Pi," 2016.

[2] K. Dumbre, S. Ganeshkar, and A. Dhekne, "Robotic Vehicle Control using Internet via Webpage and Keyboard," *International Journal of Computer Applications*, vol. 114, no. 17, p. 15–19, 2015.

[3] K. Premkumar and K. G. J. Nigel, "Smart Phone Based Robotic Arm Control Using Raspberry Pi, Android and Wi-Fi," in *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pp. 1–3, 2015.