**Team A - Drone Wars Final Report**

Drone Wars is a fast-paced quadcopter dogfighting game played with AR Parrot v2 drones.  Players take to the skies using their drones in combination with Android smartphones.  The object of the game is to be the last pilot remaining before time runs out.

The rules of the game are straightforward. Each player flies their drone and fires virtual bullets at other drones they spot in the air using the drone’s front-facing camera.  Firing a bullet reduces the current bullet count by one. Bullets regenerate every 10 seconds up to a total of 10 bullets.  If an enemy is hit with a bullet, they lose a life.  If time runs out or if there is only one player left with at least one life then the game is over.  Players can set specific game rules. They can set how many lives each player gets, how long the game lasts, whether or not God-Mode is enabled and if it is enabled how long it lasts. God-Mode grants a hit player a period of invulnerability to allow time to reposition and possibly remove themselves from the line of fire.

Drone Wars consists of three core components: the server, the Android application, and the drones.The server is the glue that holds the game together. The server holds all the game logic for the game and makes sure that all players are synchronized.  The server, once started, produces a wifi that all clients will connect to. When the drones’ batteries are connected the server automatically detects the signal and connects to them. The players select the server wifi and after that the only thing that remains is setting the game rules.

The Android application is the only user interface for the players. It has two key parts: instructions which explain setup for all pieces of the game and the Play screen where the game is played. In the Play screen the users select their corresponding drone and the color that their drone is. Once that is complete, the first player to have paired with their drone is chosen as host. Then, players who are not the host wait until the host selects the rules for the game about to be played. When those are selected, everyone moves into the actual Play screen where they will have a heads up display overlaid on top of their drone’s video feed. The heads up display allows for precise movement control of the drone. It also shows the user’s ammunition count, the user’s life count, the enemies’ life count, game time, drone battery life, and allows the user to fire.

A major change that occurred while development cycle progressed is the abandoning of the leaderboard implementation. We found that, while implementing the leaderboard would have been possible, our attention needed to be focused elsewhere.  We decided as a group to focus our time making sure that the actual gameplay was extremely smooth for the user. We chose to work on the video connection, making sure that there was minimal lag and almost no disconnects. We also chose to fine tune our image detection. Proper image detection with high reliability allows for consistent hit detection. We felt that sacrificing the leaderboard for optimized gameplay and user experience was a balanced trade that was in the interest of the players and would greatly increase the playability of Drone Wars.

**Team A – Drone Wars Team Assessment**

**What did you/your team accomplish in this course this semester?**

Our team accomplished much throughout the semester. Essentially, we completed the task at hand. We set out this semester to have a working product. That is what we have here as the semester comes to a close. At the beginning of the semester, we had half an app that contained all the very basic functionality that we completed in the first semester. Most of the functionality was primitive and still under heavy development. The server and the application were still very independent. There was no functional video feed from the drone. The game logic was unimplemented and the instruction screens were in shambles. The user interface was crude as we were waiting for images and finalized designs for the user interface. There was no way to detect another drone. To say the least, if we wanted to pass this course, this semester would have to be extremely productive. And that is exactly what this semester was – productive.

        To start with the most glaring issue that the team faced going into the second half of the project was the video feed. The last half of the first semester the team spent an inordinate amount of time researching multiple ways to try and access video from the drone. The solution was with Android Media Codec. Once implemented, the media codec was rendered on a TextureView.

        With video implemented, the next logical step was to look to image detection. It was a design choice to incorporate the open-source OpenCV. Open CV was implemented using pixel color detection base on the HSV value. The algorithm looked at pixels at certain locations and averaged them out. The pattern was chosen to have the highest density nearest the center of the target area. This would mean that an indirect hit would still be a hit but if the drone is closer to the center, the hit probability is greatly increased. We also set a threshold value of Saturation and Value in HSV for the target color such that we can reduce the disturbance and noise from the background color.

        There were also a ton of changes on the app and server. The app was updated with new user interfaces and workflows. This included the removal of the step by step instruction screen and instead we opted for a readme style instructions where the user can browse step by step instructions and information on all 4 aspects of the project – the Raspberry Pi, the drone, the app, and the Server. The instructions include images to help with understanding. The instruction framework is built into the app and does not require a connection to the internet or server.

The Drone Selection screen, Color selection screen and play screen are all revamped. The drone selection screen displays a dynamic list of drones that are connected to the server and available for selection. If a drone gets selected, it remains on the screen but is no longer selectable. The color selection screen has an all new layout with new drone buttons. The buttons are also dynamic in that once a color is selected, the choice is reflected on all phones connected to the server. The button transitions from an image of the drone with a colored hull to text reading that the color has been selected. Again, this is dynamic in the same manner which the drone selection is, the buttons change as they are selected and unselected.

        The setup screen has changed from dials to sliders. It has been updated to include all game properties including game time, number of lives, and god mode timer. There is also a dynamically populated list of all the players whom are ready with their respective colors displayed on both the waiting and setup screen.

        The play screen has both joysticks implemented. Both relay the correct movement commands to the respective drone. The bullet display has been moved up into the action bar along with the battery and the user’s life. The life of other users are displayed dynamically on the left hand side of the screen. These are real time stats of the users in the game. When a user it hit or dies, the display is updated. The fire mechanism has been implemented and calls correctly to the OpenCV functions to detect a hit in real time. There are messages that are displayed on the screen displaying a shot result, god mode, landing, winner, etc. When the player dies, they can no longer take off during the game. If they do not land within the landing timer, they are forcefully landed, for safety reasons.

        The Server has been augmented to support all the functionality required for cross phone support. All of the game logic is located on the server. This includes the lives and colors of all players, whether the player is dead, flying, etc. The server tracks all selected buttons and ready players. It knows when the game timer is up, when a user has won, disconnected, etc. It also has the ability to land drones after they have died or the game is over. The server also turns the drones back to master mode upon exiting where the user can then use the stock Parrot app to pilot them.

**What have you/your team learned through your accomplishments listed in response to the previous question?**

There is a lot that the team has learned through our accomplishments. One is that we will not get it right the first time. On the contrary, we usually didn’t get anything to work as designed until the third or fourth iteration. Take everything related to the video feed for example. The team approached the video feed in many different ways and invested an extremely large amount of time to getting the video feed to render in the app. The first iteration was to try and use a UDP stream. The second was to use FFMPEG to decode the h.264 frames being sent from the server. The third was to use the media codec objects. Fortunately, this worked well. However, like the fruitless attempts before, the SurfaceView proved to be troublesome with the OpenCV implementation because it did not share its frames. Using the SurfaceView, it was impossible to get a frame from the video feed to run the color detection algorithm on it. Thus, the fourth iteration of the video feed, the one that appeared to work the best, the media codecs with a TextureView.

        More to the point, we also learned that any product as large as we decided to go after will have repeated roadblocks which we will have to work through. The roadblocks for us in this project were both hardware, software, and people related. The first of many road blocks came with unfamiliarity with languages. As only a handful of team members knew Node.js by the start of the semester and development was heavy throughout, sometimes team members found themselves stalled in one part of the app because they needed something on the server implemented and they were not familiar with Node.js. This was a two-way Street where server development was stalled due to something on the app and they were not skilled enough in Android to get it done at that time.

        One major thing that we were able to accomplish as a team was to familiarize ourselves with the frameworks that we were using throughout the semester. There were members of the team who had a previous course working with Android and there were some members of the team who had no previous Android development experience. There were even fewer on the team with JavaScript experience. There was certainly a learning curve for each of the languages. This coincides with another thing that the team learned throughout the process, resource allocation. We realized that different team members had different strengths. This helped in many different ways. The first way was that instead of someone who was not familiar with either Android or Node.js did not have to waste time trying to implement something. All that they had to do was seek advice from the team member who was practiced in the field. This also kept the team members familiar with the code base. If there was an issue in the app, it was easily fixable because each team member knew their code.

        One of the most applicable lessons from this semester was learning how to break down a very large project into small manageable pieces. This was due in part to the length of time for which we had to complete the project and the Agile Development strategy. Every sprint we were able to know how much work we would get complete based on the hours of work we had. Then we could prioritize work and allow for collaborations amongst team members. The agile practice took an overbearing project and broke it down into manageable pieces. This practice is a skill that is highly regarded in the workforce.

**If you/your team had to do the whole project again, what would you do differently or change? Include both the successes (what has gone right so far) and failure (problems).**

If we had to do the whole project again, knowing what we know now, there is much that we would change. There is also a lot that we would keep the same. One thing that we would probably change is the prioritization of the project. There was a very good chunk of time spent on things that are not even in the project. The leaderboard is one of the big wastes of time for the team as a whole. There was a point where we had half of the team members working on the leaderboard server. This was during the end of the first semester where the two big pieces of the project being worked on were the video feed and the leaderboard functionality. Knowing how troublesome and large of a piece of the project the video feed is and how minuet and unimportant the leaderboard was to the core project, the team would have been better off dropping all the attention that the leaderboard was getting to maybe one team member and having the others either assist in the video feed or working on other core aspects of the application. The video feed is built upon an external library that appears to have some resiliency issues. There is a video feed timing out issue that we have had to deal with in what appeared to be high signal traffic areas. This is one thing that if we had a second chance, we would elect to devote more time to.

        The video feed and the leaderboard are not the only things that we as a team would have changed. If we could, we would also save ourselves the trouble of having to redo the instruction screens. It was decided from the beginning that the instructions screens would be a walkthrough of mock up screens through the app. This was a great idea except for the fact that the screens were constantly changing. It was not uncommon to have many user interface changes a week. This was a nightmare for the instruction screens that were supposed to be mirrors of the user interfaces. They were not updated along with the screens and the whole project got so far behind that it was scrapped. The decision was made to have a standard readme style instructions that were broken up by project component.

        Another thing that seemed to really throw off the momentum of the group was people missing deadlines. All but one team member are seniors and are in higher level classes. This meant that during the brunt of the semester, most did not have the time to devote to the project. This meant that missed deadlines were a common occurrence. Because the team was so dependent on each other to complete things whether it be on the server needing something in the app or vice versa, this posed an annoyance for team members. This ended up in a few tense meetings. In the end it worked out, we were able to complete the task at hand but some more accountability for missing deadlines would have helped both keep the development going smoothly and ease tensions at the meetings.

        The framework of the app is something that the team unanimously agrees would be redone if the project could be done over. This is because the framework was one of the first things written in the beginning. This was before anyone had a real idea of what we wanted and how the connection to the server would behave. The issue is that the entire app minus the OpenCV libraries and the instructions is in one monstrous class. This goes against every aspect of software design that we learned here during our tenure at UConn. At the time it was the only way that we knew to pass a connection from activity to activity. However, this turned into an issue because all of the code was crammed into this one class and was very sloppy. All of the issues that arose were fixed with band aids and the real problem was never addressed. There was an ill-fated attempt towards the end of the semester to refactor the problem but it proved too involved for such a late stage of development. If given a chance to redo it that is one of the sweeping changes to the app that would need to be done to ensure that the application would remain maintainable and reusable. The server is also an area of refactoring given the chance. The server framework follows suit with the application framework. It was written at a very early stage of development and falls victim to the same issues. Knowing what we know now, there are components of the server would could use a rewrite to conform to maintainability and reusability.

Given a second chance, the team would not jump into development. There would be a larger team investment in both getting to know each other’s’ strengths and weaknesses as well as further familiarizing ourselves with the Android to Server connection and how they are handled inside the application. As pertaining to the server, there would be a similar investment by the team members to maybe look at open-sourced game servers and realize that there was a lot more planning and design to be done before any line of code should have been written. It would have made adding functionality to both components much easier and would have allowed the project as a whole to be more stable.

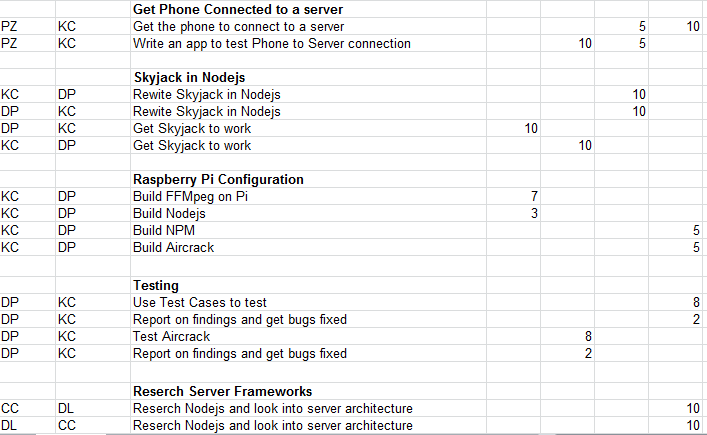
        Of all the things that we would redo if we were given the chance, there is much that we would not redo. There were a lot of successes in the project. Firstly, the overall completeness and quality of the application is something that the team is extremely happy with. The application is complete and given the proper space and hardware, the project is a fully functioning drone dogfighting platform.  Of course with the close proximity of UConn and all the team members having to meet on campus with their drones and hardware, there was never a full battle between four drones. However, there was numerous one vs one battles that performed very well for a pre alpha product.

        Of course, the product functions because the component functions. We were able to find drones that served our purpose extremely well that were relatively affordable for college seniors. The drone SDK was well documented and there was a huge drone community that was available for resource. The Raspberry Pi was an excellent choice of server hardware as it is extremely cheap, proved to be a functional server. It was easy to implement the Node framework and was just a natural choice over a full size laptop. Not only was the hardware required for the project relatively cheap, but all of the team members pitched in to acquire five drones and enough raspberry pis for everyone involved. The team members proved we were all willing to invest in the project to see it through.

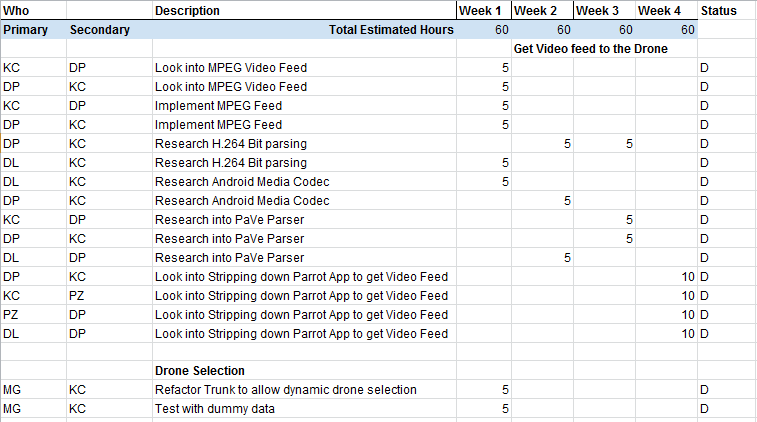
        The last thing that the team would look to improve on given a second go would be inter team communication. There was a lot of times where team members did not speak to the person whom they needed to. It left a lot of issues between team members unsolved and, again, did not make for a functional team unit. There was a lot of weekends and long nights working on the project and tensions mounted, deadlines passed, and bugs continued to show up. Communication is always one of the hardest aspects of group work that has a direct relation to quality.

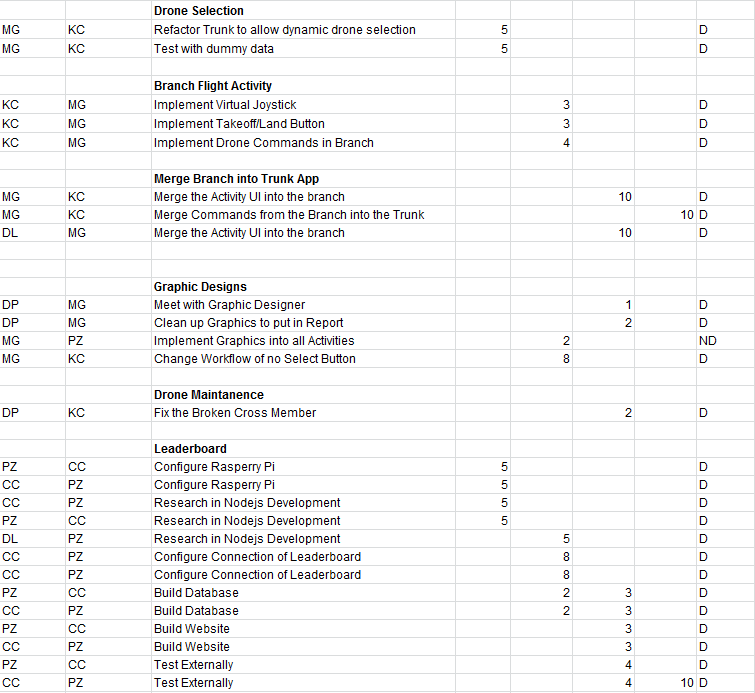
**Team contribution**

**Sprint 3**

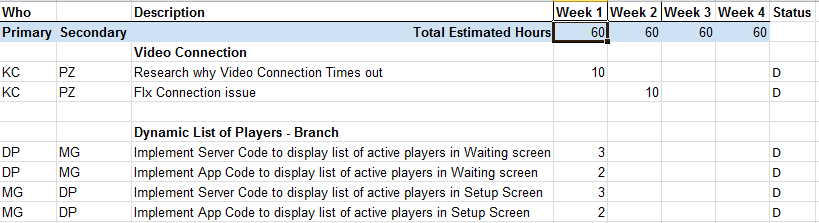
****

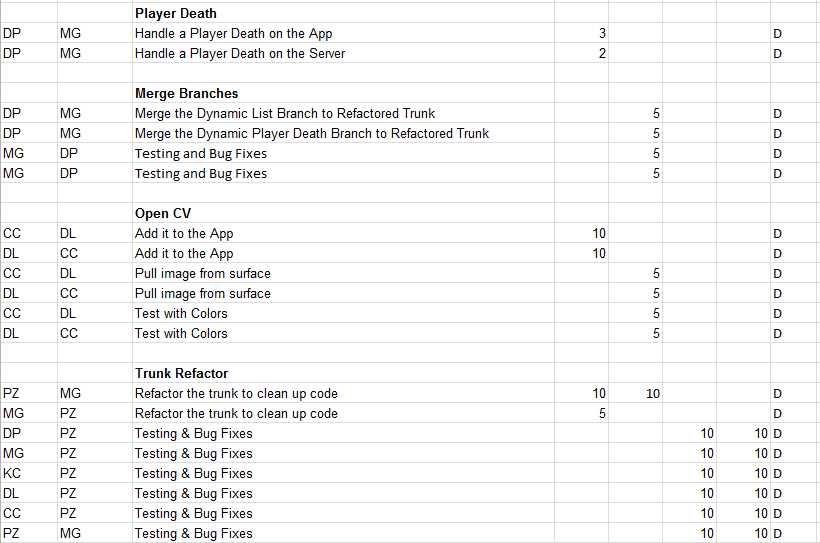
**Sprint 4**





**Sprint 5**





**Developer Documentation**

Required Technologies:

IDE:

* Android Studio: <https://developer.android.com/sdk/index.html>

SDK:

* Android: <https://developer.android.com/sdk/index.html#Other>
* NodeJS: <https://nodejs.org/>
* SocketIO: <http://socket.io/>

OS:

* Server OS: Raspian
* Android Development OS: Windows, Mac OSX, or Linux

Github:

* Repository of Android Source: <https://github.uconn.edu/DroneWars/DWA2.5>
* Repository of Server Source: <https://github.uconn.edu/DroneWars/DWA_Server>

**Development Documentation - Server**

**Operating System**

First the Raspbian operating system needs to be installed on an SD card which will allow the Raspberry Pi to boot. The directions at the following link guide the user through the installation process <https://www.raspberrypi.org/help/noobs-setup/>.

**Wireless Adapter Software**

After Raspbian has been installed and the Raspberry Pi is booten we need to insert the two Edimax EW-7811U wireless adapters into the available USB ports.

Now we will need to setup one of the wireless adapters to act as a host that other devices can connect to. In order to do that we will need to configure hostapd to broadcast on the specified wireless interface. Since Edimax EW-7811U requires a Realtek driver we will need to compile the Realtek version of hostapd to get broadcast working. The following set of commands needs to be executed to uninstall current version of hostapd and then download the Realtek, compile it and install it.

sudo apt-get autoremove hostapd

wget https://github.com/jenssegers/RTL8188-hostapd/archive/v2.0.tar.gz  
tar -zxvf v2.0.tar.gz

cd RTL8188-hostapd-2.0/hostapd  
sudo make

sudo make install

Next we need to install a local dhcp server that will allow the Raspberry Pi to lease IP addresses to devices connecting to host network. To achieve that we will need to install isc-dhcp-server using the following command.

sudo apt-get install hostapd isc-dhcp-server

**DHCP Server Setup**

Next we will need to edit the dhcpd.conf file in order to correctly setup the DHCP Server to provide IP addresses to connected devices. Run the following command to begin editing the configuration file.

sudo nano /etc/dhcp/dhcpd.conf

and make the configuration file look like the one shown below

ddns-update-style none;

default-lease-time 84600;

max-lease-time 84600;

authoritative;

log-facility local7;

subnet 192.134.3.0 netmask 255.255.255.0 {

range 192.134.3.2 192.134.3.30;

option domain-name-servers 192.168.1.1;

option domain-name "home";

option routers 192.134.3.1;

}

finally run the following command to specify the wireless interface on which the DHCP server should run on.

sudo nano /etc/default/isc-dhcp-server

Find the INTERFACES=’’ line and change it to INTERFACES="wlan0"

**Network Interface Setup**

Next we need to setup the wlan1 interface to be static and incoming. Run the following command to open the configuration file and make it look like the one below.

sudo nano /etc/network/interfaces

auto lo

iface lo inet loopback

iface eth0 inet dhcp

auto wlan1

allow-hotplug wlan1

iface wlan1 inet static

address 192.134.3.1

netmask 255.255.255.0

**Configure Access Point**

Now we need to configure the access point details. Mainly make it an open network that anyone can connect to. Run the following command and paste in the configuration listed below.

sudo nano /etc/hostapd/hostapd.conf

# Basic configuration

interface=wlan1

ssid=DWA

channel=2

# Hardware configuration

country\_code=US

ieee80211d=1

driver=rtl871xdrv

ieee80211n=1

wmm\_enabled=1

hw\_mode=g

device\_name=RTL8192CU

manufacturer=Realtek

Next we need to tell the Raspberry Pi where to find the configuration file. Run the following command

sudo nano /etc/default/hostapd

and change the #DAEMON\_CONF="" line to DAEMON\_CONF="/etc/hostapd/hostapd.conf"

Lastly we need to configure the Raspberry Pi to start up the hostapd and DHCP server on boot so that the host network is available when the Pi is powered on. Run the following commands to add both programs to startup script.

sudo update-rc.d hostapd enable

sudo update-rc.d isc-dhcp-server enable

**Node.js Installation**

First we need to add a package repository that contains node.js to your sources and install it using the following commands

curl -sLS https://apt.adafruit.com/add | sudo bash

sudo apt-get install node

**DWA\_Server Setup**

Lastly we can download the DWA Server package from git. Then go into the DWA folder and finally run it using the following commands.

git clone <https://github.uconn.edu/DroneWars/DWA_Server.git> DWA

cd DWA

sudo node dwa.js

**Development Documentation - Application**

**Integrated Development Environment (IDE)**

First the IDE android studio needs to be installed, which will allow the developer to open the Drone Wars source code. Instructions on installation can be found here: <https://developer.android.com/sdk/index.html>

**Software Development Kit (SDK)**

Once Android Studio is installed, access the SDK Manager and install Android 4.4.W2 (API 20) through Android 5.1.1 (API 22) are installed.

**Install Github Desktop(SDK)**

First install the github desktop application. This will allow users to clone our Drone Wars project onto their computer. The instruction for installation can be found below: <https://windows.github.com>

**Clone DWA Github Project**

Once the the Github application is installed, copy the DWA repository link found here: <https://github.uconn.edu/mfg10001/DWA2.5.git>. Open the Github application and click the add button on the top left corner ( + sign) and paste the link in and hit clone.

**OpenCV Manager**

Use google play to download the newest version of OpenCV Manager in order to support the OpenCV library in the app.

**DWA2.5 Setup**

Once the DWA application is cloned onto your machine, open Android Studio. Once Android Studio is opened  click import project on the right hand side of the startup menu, then open the cloned github project. The Drone Wars source code should be successfully imported into Android Studio and ready to be used. To install Drone Wars on an android device, plug in the android device to your computer and select file and then run from the drop down menu

**User Manual**

The following user’s manual is broken up into 3 parts: Server setup, Drone setup, and Android Application setup. The manual should be followed in numerical order (1,2, then 3) to ensure the smoothest setup possible.

**Server setup:**

1. Plug your Raspberry Pi (Raspberry Pi is a trademark of the Raspberry Pi Foundation) to a power source. If you are using a portable battery to power your Raspberry Pi please make sure you have at least 20 minutes of charge so that your server doesn’t shut down in the middle of the game.
2. Log into your Raspberry Pi with your login credentials
3. Navigate to the DWA folder within your server. The command to do so is “cd DWA”. If you are not in your home directory the previous command will not work.  Please make sure to do “cd HOME” to return home before moving into the DWA.
4. Start the Server application with the “node dwa.js” or “sudo node dwa.js” command if you don’t have root access. Note, you may need to replace “node” with “nodejs” in the commands depending on your server configuration.  Once the server application is started no more setup is needed.
5. If you wish to close the server simply press CTRL + C. To shutdown the server insert the command “sudo shutdown -h now” or to reboot, “sudo reboot”.

**Drone setup:**

1. Remove the hull of your drone
2. Insert the battery and make sure that it is strapped into the drone’s battery holder. Once the battery is secured plug the battery into the drone.
3. Wait until the LED lights under the drone’s propellers flash and the drone beeps four times. Once the LED lights stop flashing and turn green, your drone is done connecting to the server

**Android Application setup:**

1. Press the Drone Wars icon to start the Drone Wars Application

  
Home Screen

1. Select the Play icon on the home screen. You will move to the drone selection screen.

  
Drone Selection Screen

1. If your drone hasn’t connected yet, wait until it does. Once your drone connects, select it from the list. You will then move to the color selection screen.

  
Color Selection Screen

1. On the color selection screen, select the color that corresponds to your drone’s hull. If you were the first one to select your drone then you will be host and move to the game setup screen.
2. If you are not the host, skip to step 6. Otherwise, as host you need to decide with the other players what the rules of the game will be. You must decide the game length, number of lives per player, if God-Mode is enabled, and if God-Mode is enabled then how long is the period of invulnerability. Once all game settings have been set, select the continue button. Once the continue button is pressed you will go to the play screen. Please skip step 6 and go to step 7.

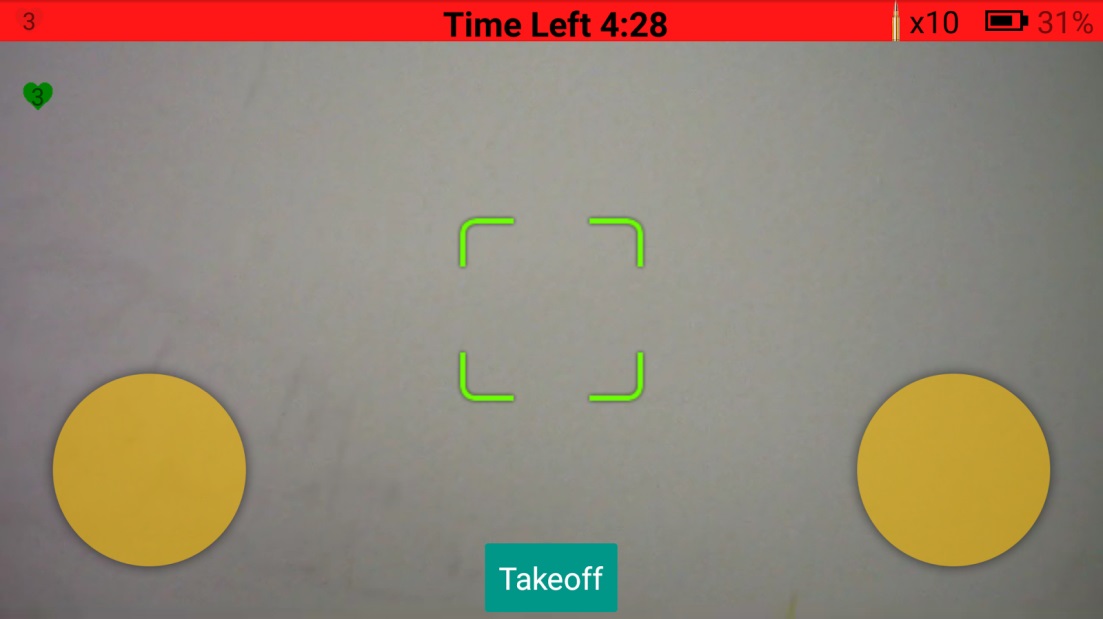


Setup Screen

1. If you were not the host, you will have seen the waiting screen. There all the drones and their colors would have been show. Once the host has finished inputting the game settings you will move to the play screen.

  
Waiting Screen

1. The play screen is where all the action occurs. At the top there is a status bar. This holds the game time, your life count, your ammo count, and the battery life of your drone. On the right hand side you will see a joystick. This right joystick controls the pitch and roll of your drone.  Double tapping anywhere in the right joystick will cause your drone to fire. On the left-hand side you will see your lives and the opponent’s lives and another joystick. This joystick controls the altitude and rotation of your drone. In between both joysticks is the Takeoff/Land button. When your drone is in the air, the button changes from takeoff to land. Finally, there is the target area marked in the center of the screen. When a drone is in the this region you want to fire.

  
Play Screen