Team A - Drone Wars Final Report

Drone Wars is an action-packed, dogfight video game using the combination of Android smartphones and AR Parrot v2 drones. The main goal of Drone Wars is to provide entertainment and a little competition to owners of drones. Anyone with a Parrot drone and a smartphone running at least Android 4.4 will be able to easily jump into a game of Drone Wars with very little effort.

Drone Wars consists of players and a single server administrator. The administrator’s job is to force drones to land or shutdown if there is an emergency. The administrator will be able to do this via direct commands to the server. The game itself is setup, controlled, and run by the players. Each player selects their identifying color to mark their drones. The player who selects their identifying color to be red is the game creator and is responsible for determining how long the game will be, how many lives each player will have, as well as other rules which determine how the game will be played. To set up the game network, it is as simple as connecting a wireless router to a laptop and clicking the application to start the game server. This is the entire setup, the game doesn’t even require a connection to the Internet.

The game consists of having the players use their smartphones to control their drones. While controlling their drones via the drones’ front-facing camera, they need to line up the aiming reticle, which is overlaid on top of the video input from the camera, with an enemy drone. Pressing the fire button when the opponent’s drone is correctly targeted will result in the targeting player’s virtual ammunition count decrementing and the targeted player’s life count decreasing by one and allowing them a small window of time where they are invulnerable. After a couple of seconds, a round of ammunition will be replenished up to a maximum of 10 virtual bullets. The game ends when either the game time runs out or there is only one player with at least 1 life left. As mentioned before, the amount of game time and the number of lives per player is determined at the beginning of the game by the player who selected the color red.

Assessing the project now at the approximate halfway point, we can say that thus far we have not had any major changes to the overall goal of the project. For the most part, we have stayed true to creating a game for owners of AR Parrot drones. Originally, we had the thought of implementing autonomous flying to the game but that has changed and we no longer have a goal for that implementation.

In terms of actual design choice, there has been some slight deviations from our original intent. We first planned on using REST API to implement the video transfer between the drones and the users’ phones. However, we decided to use Socket IO API in place of REST. The reason for this is that Socket IO will allow us to pass a wireless stream of video to and from clients with greater efficiency and more ease. This change will affect our project since we are also planning on developing an equivalent app for IOS. The reason that REST was chosen originally was because the networking structure that we would use to communicate to the server would stay the same between Android and IOS. Now, using Socket IO we may find that there might have to be some noticeable differences between the Android and IOS version. However, since we have not yet created the IOS version, it is not certain that these differences may exist.

Team Assessment

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

Group A this semester chose to create an augmented reality, drone warfare mobile application. In order to make the vision of our mobile application a reality, we had to first create the game logic, so that the users would have an enjoyable experience while playing with our Drone Wars project. Once the game logic was finalized, the application screens needed to be mocked up and approved by all of the group members, which included all the artwork designs as well. The application design team had to research various techniques ( instruction overlay) , creating a custom action bars and how to implement head up displays (HUD) to add image overlays to the play screen so that physical drones camera feed would have the users controls all in one view for the user. These accomplishments were important for the overall design of the Drone Wars application, by simplifying the UI without added unnecessary distractions to the player, and helped reduce the amount of screen space used so the player could focus on flying the drone. The custom action bars, change color to match the individual users drone color, as a simple yet affect reminder as to which drone they are controlling. The HUD allows for the ammunition remaining, health remaining, the drone control inputs and the “gun’s” (cameras) crosshairs to be visible above the incoming video feed from the server. The HUD is also used to create a hovering instruction set that appears the very first time the user opens the app, this is done by accessing the phone shared preferences and setting and storing a Boolean value to the phones memory. The HUD instructions were implemented to guide the first time user by explaining the applications flow and what each symbol does and how to play the game safely.

The Drone Wars application requires a game server to control the games logic and to forward commands from the each player’s phone to the correct corresponding drone. The group had to research various types of servers to establish which would be the best for our application. The server design team then determined that Ubuntu server was the best server for our applications. Once the server was determined, we researched various dual band routers to determine would be the best to use with our Parrot A.R drones and server. The server team researched air cracking and NodeJS. The first major accomplishment of the server team was using NodeJS scripts to connect their Parrot AR drone to their computers and using scripts to access the drone’s video feed to recognize enemy tags. The team was also able to successfully use NodeJS scripts to connect and pilot the drones from their computers. The server team also researched a tool called aircrack, which searches for predetermined drone mac addresses, disconnects them from their current Wi-Fi session and forces them to connect to our server’s router. The server is running on a raspberry pi, and all the video and control goes through it and acts as a self-contained unit. There is no outside resource needed to get the connections. It is has a very small footprint and is capable of streaming HD video. At the same time the team was setting up a Linux based server and configuring our dual band router to accept the incoming phone and drone communications. Once connected to the server the drones will handshake with the corresponding user’s phone.

What we have accomplished so far has two independent parts, the server part and the application part. For the server side, we are able to connect the drone using a laptop and get the video feed from the drone via its Wi-Fi connection. We can also send commands to fly the drone using a laptop. The two parts are independent for now because the server is not up and running so the application is not able to connect to it and send commands like the laptop can do. After the game concept was finalized, three of us started developing the application and interface mockups; three of us started development on the server. We decided that the application has three parts: leaderboard, instructions, and play mode. So far we have the mockups for the instructions and the play mode. Developing the leaderboard involves using a database, and it is not the most important part of the application, so we decided to develop the leader board last (after the application is functional). Instructions were relatively easy. However, we needed to do the mockup for the play mode first because the instruction needed the screenshots from the play mode. One of us did the mockup for the instruction and two of us were working on the play mode. Based on the concept of the game, users have to select a drone to use before they go into the play mode. Before the real play screen we implemented a “drone selection screen”. It lets a user select an available drone and gives time for the phone to communicate with the server which communicates with the drone also known as the handshake process. This will be the main task for us next semester. All the screenshots of the application will be provided later in the documentation.

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

Our team has learned valuable lessons through our research and errors. Once our initial designs were finalized, we realized there were several tasks that needed to be addressed and researched. One of those tasks, addressed by Peter Zaffetti, was displaying a showcase view that would display over the current activity which the user is in. This view would step through what each symbol and button means and what they do before they can play for safety reasons. The showcase view instructions are able to keep track of the instructions that were seen before by saving a boolean key in the users android devices shared preferences. Another task that needed to be addressed was how to modify the Android action bar to suit our application’s needs. Chunze Chen and Dake Lu researched the best way to change the stock Android action to address our needs. They found that although it was possible to modify the stock Android action bar’s background color, it was not possible to add the various symbols and game time (countdown clock) to the action bar. Michael Grillo created a custom action bar by nesting layouts within the activity’s xml file and hid both the stock action and navigation bars. The custom action bar is essential to the user experience because it displays all of the game’s connections, game time and what color drone the user is flying in a subtle way. In order to ensure the app’s activities’ designs are seamless Dake began mocking up various screens layouts, buttons and symbols with corresponding designs and color schemes. His mock ups were then sent out to a graphic designer to be completed since no one in the group had graphic design experience.

The application’s back end had various challenges that were addressed this semester. One of those challenges was to determine what language would be best suited to use for the drone to server and server to phone connections. Krystian, Dustin and Peter researched and decided that Node JS and Socket IO were the best to use because there is framework to communicate between the drone. Additionally, it is easy to configure for server connections. Peter Zaffetti configured the dual band wireless router for network communications so that it would have optimal bandwidth speeds to ensure live video feed is received by the server and passed to the user’s phone without lag. Dustin and Peter also setup and configured two Ubuntu servers, one of which will be the game server. The game server will host the game logic and image process components. Once Dustin and Krystian gained an understanding of NodeJS, they wrote scripts to connect their drones to their computers through Wi-Fi. Through trial and error, they discovered that the drones could only be programed to recognize one “enemy” tag at a time. Since the game was designed to be a multiplayer, we decided to use Open CV image processing software on the server to process each drone’s video feed. Open CV will be used to bypass the drones limitation of only being able to scan for one enemy tag at a time This will allow Drone Wars to be truly multiplayer. Connecting the drones to their laptops was a great start, but Drone Wars requires a connection to the server’s router instead. Through his research, Krystian developed a technique to force the drones to connect to our wireless router using air cracking software, a few NodeJS scripts, and a Raspberry Pie. Krystian’s technique connects the Raspberry Pie to a wireless receiver that scans for all the predetermined drone mac addresses. Once the addresses are found, the existing Wi-Fi connections are interrupted and the drone is forced to connect to our game’s router instead.

During the semester our team gained a better understanding of the Android operating system and how it works since not all of us had experience with it. From our research we found the correct ways to implement multiple HUD displays for the user, so that the drones’ video feed could be show in the background with minimal distractions. Throughout the semester the research done with NodeJS, Socket IO and air cracking, allowed us to gain a better understanding of how to accomplish the drone to server connections. Those server connections helped us better understand what we needed to get accomplished in order to make our vision of Drone Wars a reality.

**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 our team had to do the whole project over again, we would have done a couple things differently. First, we would have researched the actual drones to use for this project a little more thoroughly. We would have also dedicated more time on the server instead of the android application. This is because once we were able to connect to the drone we realized the server was going to be more complex than we originally thought. The drones were found to have limitations in their software that only allows one tag to be recognized at a time. That limitation was a major setback to us because in order to create the multiplayer game we envisioned, the server would not only have to host the game logic, it would have to do image processing on each of the drones’ video feed to check for multiple enemy tags. The need for Open CV image processing to be run on the server changed our design completely. The server was switched from a Raspberry Pi to a dedicated laptop running as our Ubuntu server. Another aspect of the project that we would change would be not splitting up the team in half to work on the application and the other half on the server. Since the basic design of the server changed and had to be reconfigured on a laptop it set back the projects progress a bit. Without the server being up the application design team had could not write the server connection classes, so if we had to do it over we would have definitely shared the load of the server a bit more than we did this time around.

Although the Android application team was relatively successful this semester, if given the chance to redo the project we would have created our own action bar first instead of wasting a few days trying to modify the stock Android action bar to suit our needs. If we had just researched how to create our own action bar sooner we would not have wasted our time with the trial and error attempts. The team, given the chance, would have spent more time helping the group members that were not familiar with Android to understand the basics. This would have helped them become acquainted with it faster which would have sped up development. In addition to spending more time with those unfamiliar with android, our group should have spent more time discussing the applications flow. Discussing the application’s flow would have saved our team time by not having to modify the application’s activity after the mockups were replaced with functioning activities. Once the application was run for the first time, we found that the flow we imagined in the mock ups was actually quite confusing for the users and seemed fragmented.

Although we have a lot that we would have changed, there is much that we would not. We would keep our overall app software design the same. We found that our original design allows us to easily change the flow of the Android application. We had to modify the flow of the application to add in additional instruction screens. However, due to the nature of the design we chose, it was very simple to insert the activities needed. We did not need to do a major overhaul to the code or the flow of the activities. This allows us to greatly decrease complexity while making our application extremely user friendly. Along those lines, we would not change our user interface or experience. While surveying our application to various classmates, we found that they thought it was quite easy to navigate and understand. We understand that creating simple user interfaces and an easy experience for the users is very important. Having gotten this feedback from our peers we would not change this aspect of our app.

Furthermore, we would also keep our high level design of the game. Our client-server architecture is extremely modular and is very powerful. It allows us to separate our concerns. By using this architecture we are able to work on multiple things at once without having to worry about messing up another team member’s code. For example, while Mike, Chunze, and Dake worked on the Android application, Krystian and Dustin could work on the server, all while Peter could work on the router. These three separate parts could be developed individually by separate members of the team and now they just need to be connected. Our design allowed us the freedom to work unhindered by different parts of the game. The final product will simply be connecting up the distinct pieces of the project.

Note Spreadsheet below embedded xlsx – right click, Worksheet Object, Open or Edit

Team Contributions



Team A - Drone Wars Application Developer Documentation

**Written by Chunze -**

Required Technologies:

Integrated Development Environment

* Eclipse with Android SDK, available at:<https://developer.android.com/sdk/index.html?hl=i>

Software Development Kit

* Android 4.4.2 (API 19) or higher version, available in the SDK manager in the eclipse with Android SDK.

API and API documentation

* API for android application development, provided by Google at:<https://developer.android.com/index.html>

Testing Environment

* Android virtual device with Android 4.4.2 or higher version, available in the Android virtual device manager in the eclipse with Android SDK.
* Or any real Android device with Android Version 4.4.2 or higher.

Github

* Repository of source code: [https://github.uconn.edu/DroneWars](https://github.uconn.edu/pdz10001/DWA-Android)

Instruction on running the application in Eclipse:

1. Download Eclipse from the link provided above. Then extract the file to a safe location.
2. First time open Eclipse, it will ask you to assign a folder to be your workplace. Once you select a folder, all of your work will be saved in that folder.

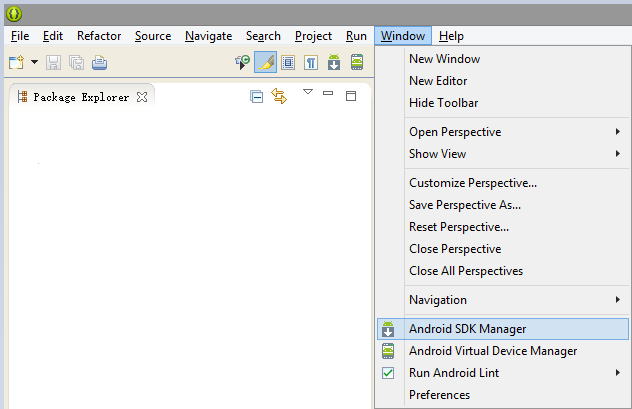


Figure 1 select “window” -> “Android SDK Manager”

1. Downloading the necessary APIs: click “Window”, then “Android SDK Manager”, see figure 1.

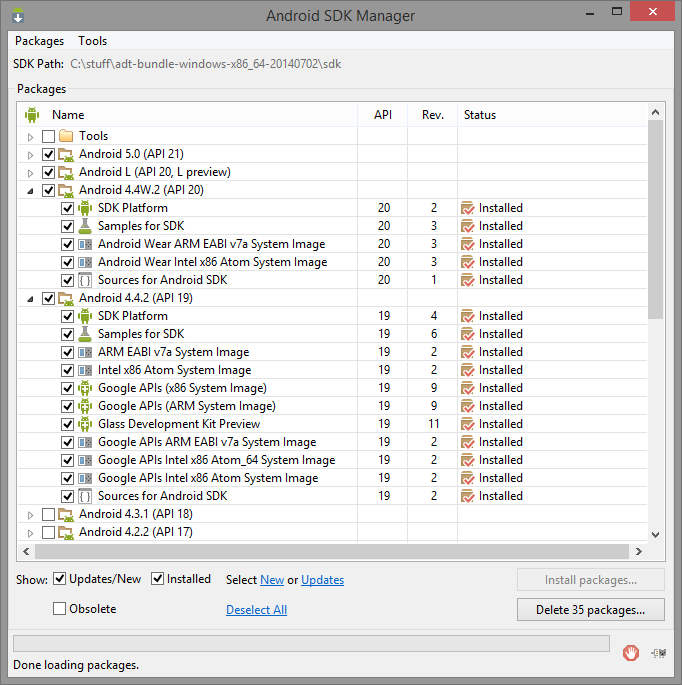


Figure 2 Android SDK Manager

1. The Android SDK Manager will open after selecting this, shown in Figure 2. Then selecting all the APIs that is higher than 19, including 19. Then click “Install packages” at the lower right of the window.
2. If an Android device if not available, an Android virtual device is needed to run the application. Setting up an Android virtual device: below “Android SDK Manager” option in Figure 1, click on the “Android Virtual Device Manager” option.

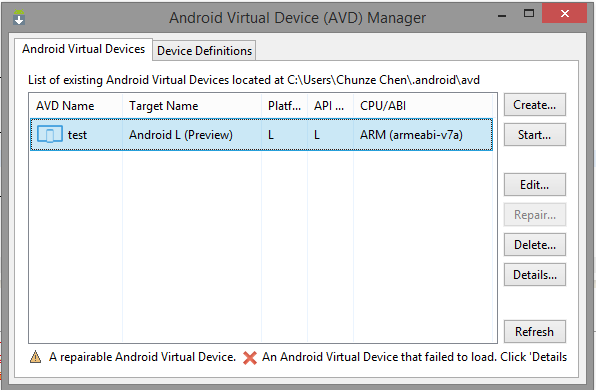


Figure 3 Android Virtual Device Manager

1. The window shown in Figure 3 will open. Click “Create” on the top right of the window. This is where the configuration of a virtual device needs to be done.

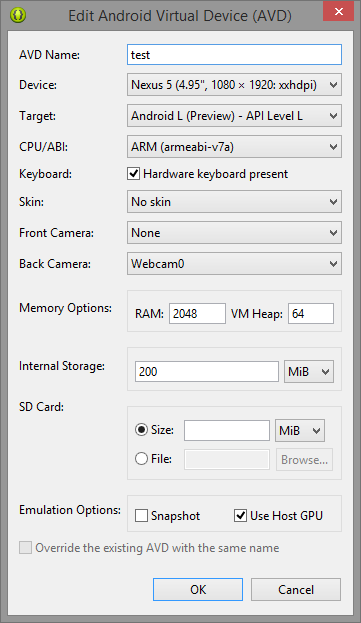


Figure 4 Android Virtual Device configuration

1. After clicking “create”, a window will open, shown in Figure 4. And select the configuration shown in Figure 4, then click OK.
2. Download the repository from the link provided in the documentation, then import it as an Android application. Right click on the project in the navigation window, then select “run as” -> “1. Android Application”.

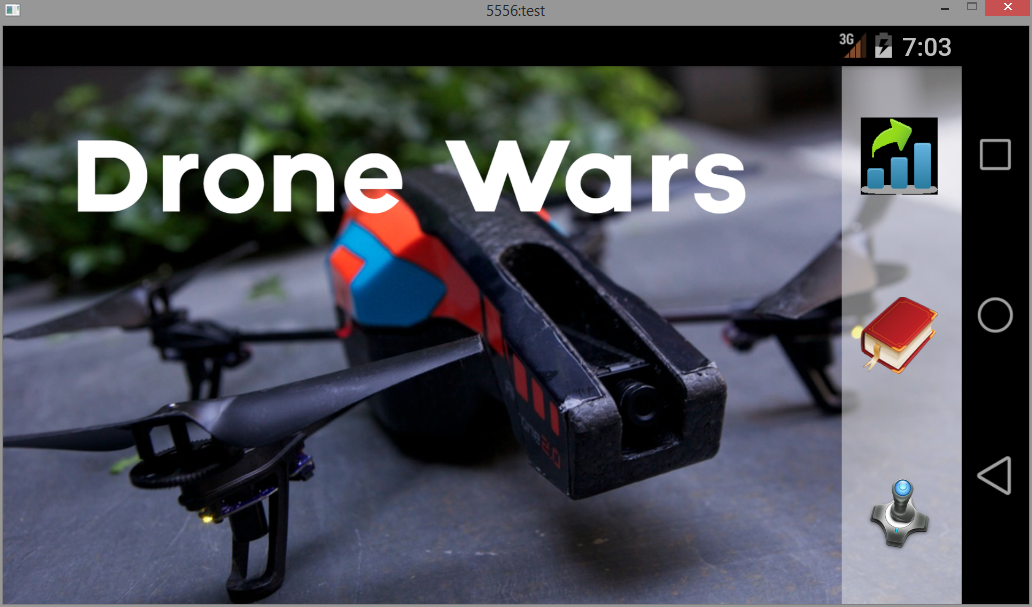


Figure 5 The home screen of our application

1. The application is defaulted to a landscape orientation, press “CTRL+F11” to change the Android virtual device in landscape mode. Figure 5 is the application running in the Android virtual device.

Team A - Drone Wars Server Developer Documentation

**Written by Krystian -**

Required Technologies:

API and API documentation

* Aircrack-ng - <http://www.aircrack-ng.org/documentation.html>
* Node.js - <http://nodejs.org/documentation/>
* node-ar-drone - <https://github.com/felixge/node-ar-drone>
* node-dronestream - <https://github.com/bkw/node-dronestream>
* keypress - <https://github.com/TooTallNate/keypress>
* shelljs - <https://github.com/arturadib/shelljs>
* Socket.io - <http://socket.io/docs/>
* AndroidAsync - <https://github.com/koush/AndroidAsync/>
* SkyJack - <https://github.com/samyk/skyjack>

Testing Environment

* Virtual Machine running a Linux distribution
* Raspberry Pi running Raspian

Github

* Repository of source code:<https://github.uconn.edu/DroneWars/DWA-Server>

Instruction for running application on Raspberry Pi

1. Install Rasbian by following official setup guide <http://www.raspberrypi.org/help/noobs-setup/>
2. Download Aircrack-ng source from (<http://download.aircrack-ng.org/aircrack-ng-1.2-rc1.tar.gz>) then compilw and install
3. Download Node.js / NPM source from <http://nodejs.org/dist/latest/node-v0.10.33.tar.gz> then compile and install
4. Use ‘npm install’ to install node libraries (ar-drone, dronestream, keypress, shelljs) for communicating with the AR.Drone as well as capturing user input and executing linux shell commands.
5. Plug in the [Edimax EW-7811Un](https://www.amazon.com/gp/product/B003MTTJOY/ref=oh_aui_detailpage_o04_s01?ie=UTF8&psc=1) Wi-Fi dongle into a USB port on the Raspberry Pi
6. Plug in the [Alfa AWUS036H](https://www.amazon.com/gp/product/B000WXSO76/ref=oh_aui_detailpage_o04_s01?ie=UTF8&psc=1) Wi-Fi adapter into another USB port available on the Raspberry Pi
7. Execute ‘iwconfig’ in Linux Shell to make sure that Edimax dongle is identified as wlan0 and the Alfa adapter is identified as wlan1
8. Execute ‘git clone <https://github.uconn.edu/DroneWars/DWA-Server.git> DWA’ to save source code in DWA folder located in home directory
9. Move into the DWA folder by executing ‘cd DWA’
10. Next execute the modified skyjack script by running ‘sudo perl skyjack.pl’ enter password when requested.
11. The Alfa Wi-Fi adapter will enter monitoring mode where it scans for packets being broadcasted and gets the MAC addresses of all the broadcasting and connected devices. While the process runs the output will be pi@raspberrypi ~/DWA $ sudo perl [skyjack.pl](http://skyjack.pl/)

Running: ifconfig wlan1 down

pid 3423

Running: kill 3423

Running: kill -HUP 3423

Running: kill -9 3423

Running: killall -9 aireplay-ng airodump-ng

aireplay-ng: no process found

1. Next turn on an AR.Drone and once it goes through its boot procedure it will begin broadcasting as an Access Point, skyjack will then be able to identify that the AR.Drone is broadcasting and skyjack will utilize the Edimax Wi-Fi dongle to connect to the AR.Drone using its ESSID and use DHCP to request and IP address used for communication. The shell output will be as follows.

CHANNEL 90:03:B7:EA:B6:B1 6 ardrone2\_070400

Connecting to drone ardrone2\_070400 (90:03:B7:EA:B6:B1)

Running: iwconfig wlan0 essid ardrone2\_070400

Acquiring IP from drone for hostile takeover

Running: dhclient -v wlan0

Internet Systems Consortium DHCP Client 4.2.2

Copyright 2004-2011 Internet Systems Consortium.

All rights reserved.

For info, please visit <https://www.isc.org/software/dhcp/>

Listening on LPF/wlan0/74:da:38:02:36:c8

Sending on LPF/wlan0/74:da:38:02:36:c8

Sending on Socket/fallback

DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 4

DHCPREQUEST on wlan0 to 255.255.255.255 port 67

DHCPOFFER from 192.168.1.1

DHCPACK from 192.168.1.1

bound to 192.168.1.2 -- renewal in 573 seconds.

1. After that skyjack will execute a node.js script which will be used for controlling the AR.Drone as well as streaming the live video which will be available at localhost:5555, the shell output will be as follows.

TAKING OVER DRONE

Running: node drone\_control/drone\_pwn.js

1. Finally the user can control the AR.Drone using the keyboard while the Linux shell window is active. The list of available commands is

|  |  |
| --- | --- |
| Up Arrow | Fly Forward |
| Down Arrow | Fly Backward |
| Left Arrow | Fly Left |
| Right Arrow | Fly Right |
| W | Ascend |
| S | Descend |
| A | Turn Left |
| D | Turn Right |