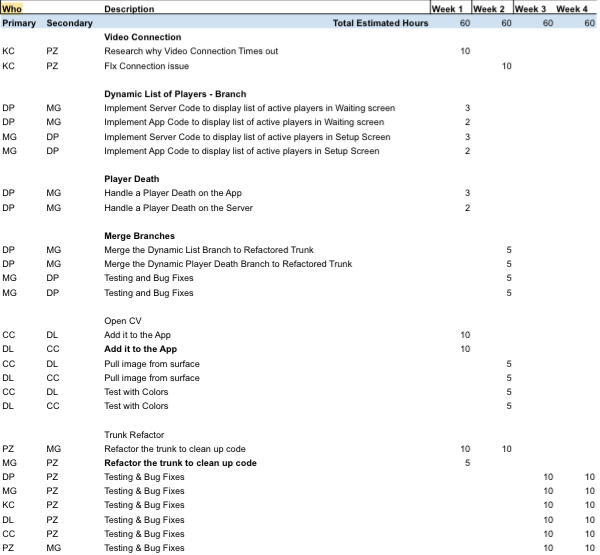
**Team A**

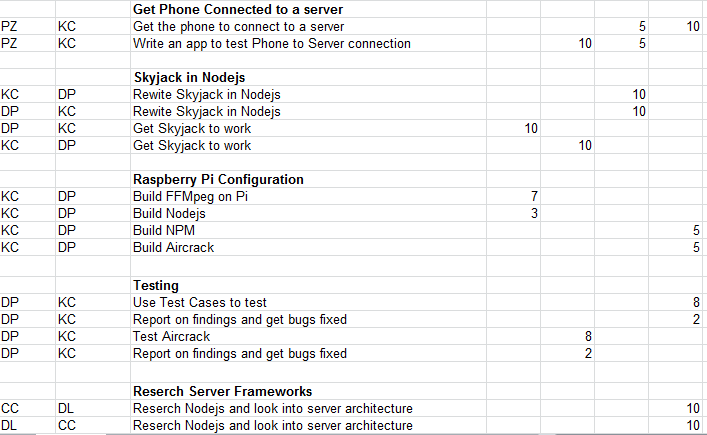
**Section A**

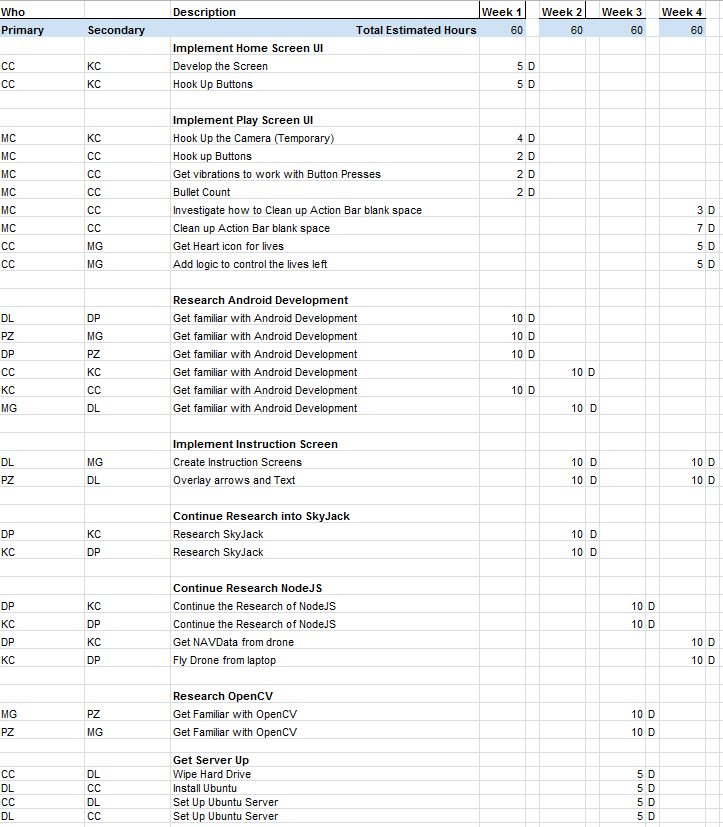
**Sprint 5 Backlog:**

****

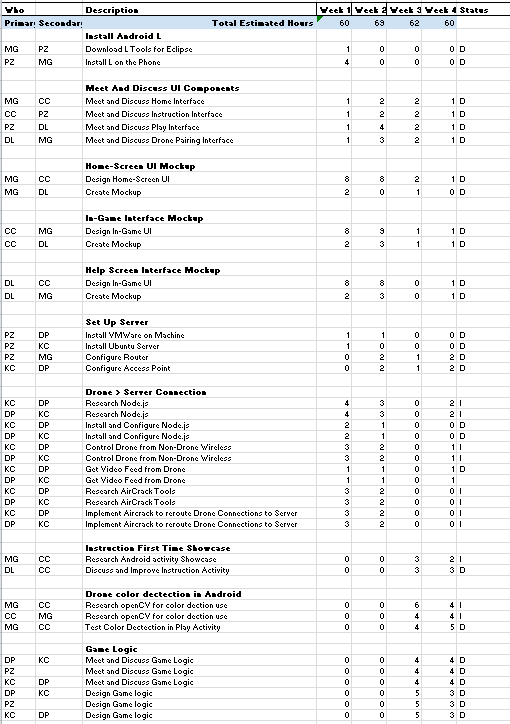
**Sprint 4 Backlog:**

**Sprint 3 Backlog:**



**Sprint 2 Backlog:**

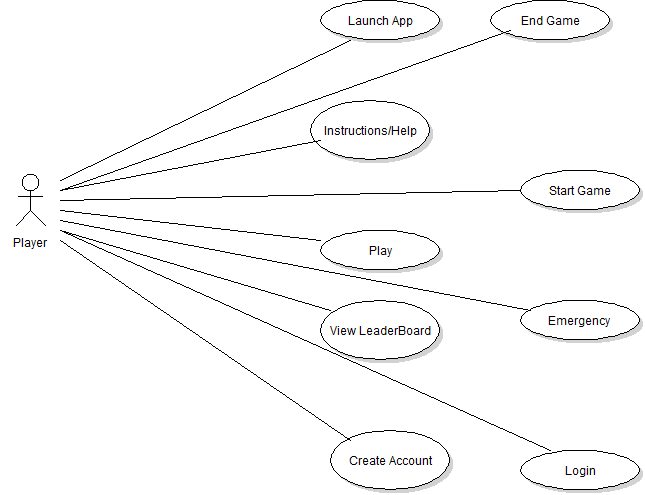
**Sprint 1 Backlog:**



**Section B:**

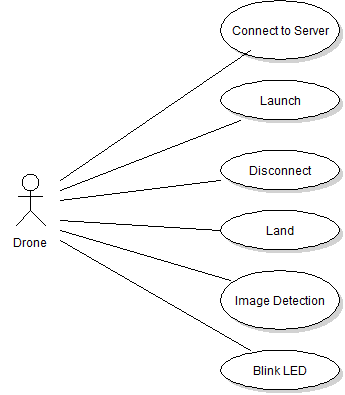
**Use Case Diagrams - MG**

The application DWA has three distinct users; these include the player, the drone itself and the leaderboard administrator. The player will be able to launch the DWA application and be able to access all activities needed to play and end the game and access/update the leaderboard. In *Figure B.1* the player actor is shown with the use-cases Launch App, Instructions/Help, Leaderboard and Play Game. These use-cases allows the user to control the **drone** actors and register their previously played game to the DWA leaderboard server by either creating an account or logging in. The **player** also has the ability to start and end their own involvement in the game.



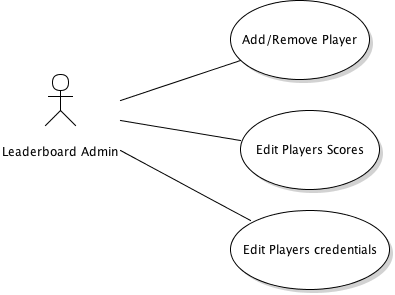
***Figure B.1*** *The Player Actor Use Case diagrams*

From *Figure B.2*, it is apparent that the **drone** actor has the ability to connect to the game server. The game server connects the **Drone**, to the game server and the **player** by IP. The **Drone** actor will be able to disconnect itself from the game server if its lives run out and it enters the auto land feature and it can disconnect if the **player** actor flies out of the designated games boundary. The **drone** actor also has the ability to blink its LED and perform image detection.



***Figure B.2*** *The Done Actor Use Case*

*Figure B.3* shows that the **leaderboard admin** actor has the ability to connect to the leaderboard server. Once connected the **leaderboard admin** has the ability to add or remove players, edit player’s scores in case of a discrepancy and edit the player’s credentials.



***Figure B.3*** *The Leaderboard Admin Use Case*

**Section C:**

**User Based Specifications/Interfaces**

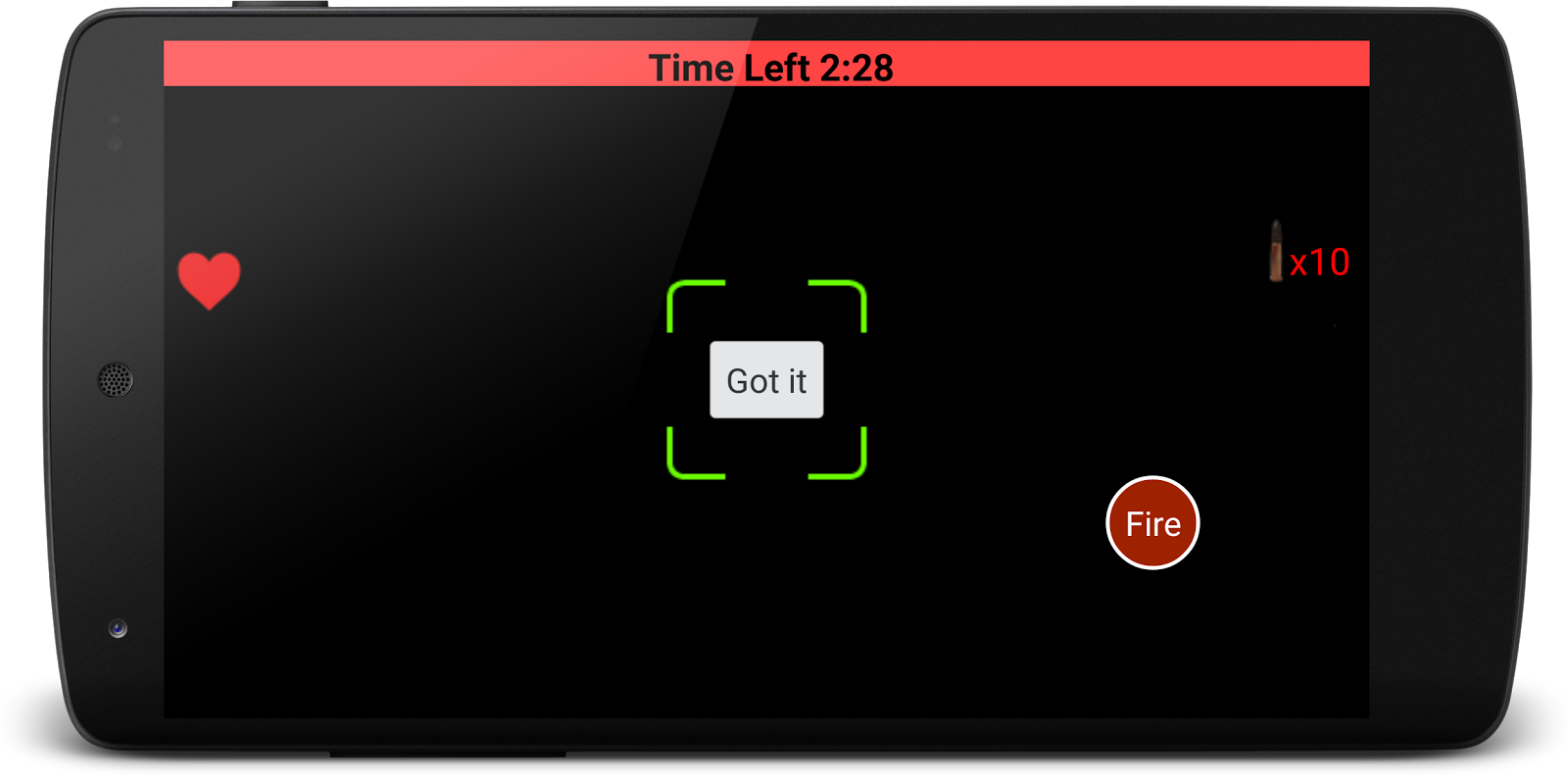
The application contains many interfaces which the user can interact with. These interfaces provide a lot of functionality from viewing the leaderboard, piloting the drone, to viewing the instructions. The interfaces are all Androidbased and communicate with the server to provide real time communication while playing the game. Furthermore, the interfaces will display information provided by the server in real time including video, game statistics, and drone status. *Figure C.1* shows the Home-screen and the three buttons available to the user. From the Home-screen the user will be able to choose from 3 options: the first is Leaderboard, the second is Instructions, and the last is Play Drone Wars.



***Figure C.1*** *The Drone Wars Home-screen*

If the user selects the Play Drone Wars button for the first time then the user will be brought a help screen where the Play Screen Instructions will be shown. The Play Screen Instructions label and explain the various parts of the Play Screen. If it is not the user’s first time selecting Play Drone Wars then the user will go immediately to the Drone Selection Screen and then from there to the Play Screen. When the user selects the Instructions button they are brought the Instructions Screen. If the user selects Leaderboard they are brought the Leaderboard Screen (this is an additional feature and will be implemented if time allows).

*Figure C.2* is a limited mockup of the Play Screen Instruction Screen. This screen will show up when the user first launches the app and goes through all the other instruction screens. This screen is going to be populated with the showcase view and all of the icons will have explanations of what they do. There will be a different view for each control. This will encompass all the action bar items, all drone control buttons as well as HUD items, and life/bullet displays.



***Figure C.2*** *The drone setup page Instruction Screen*

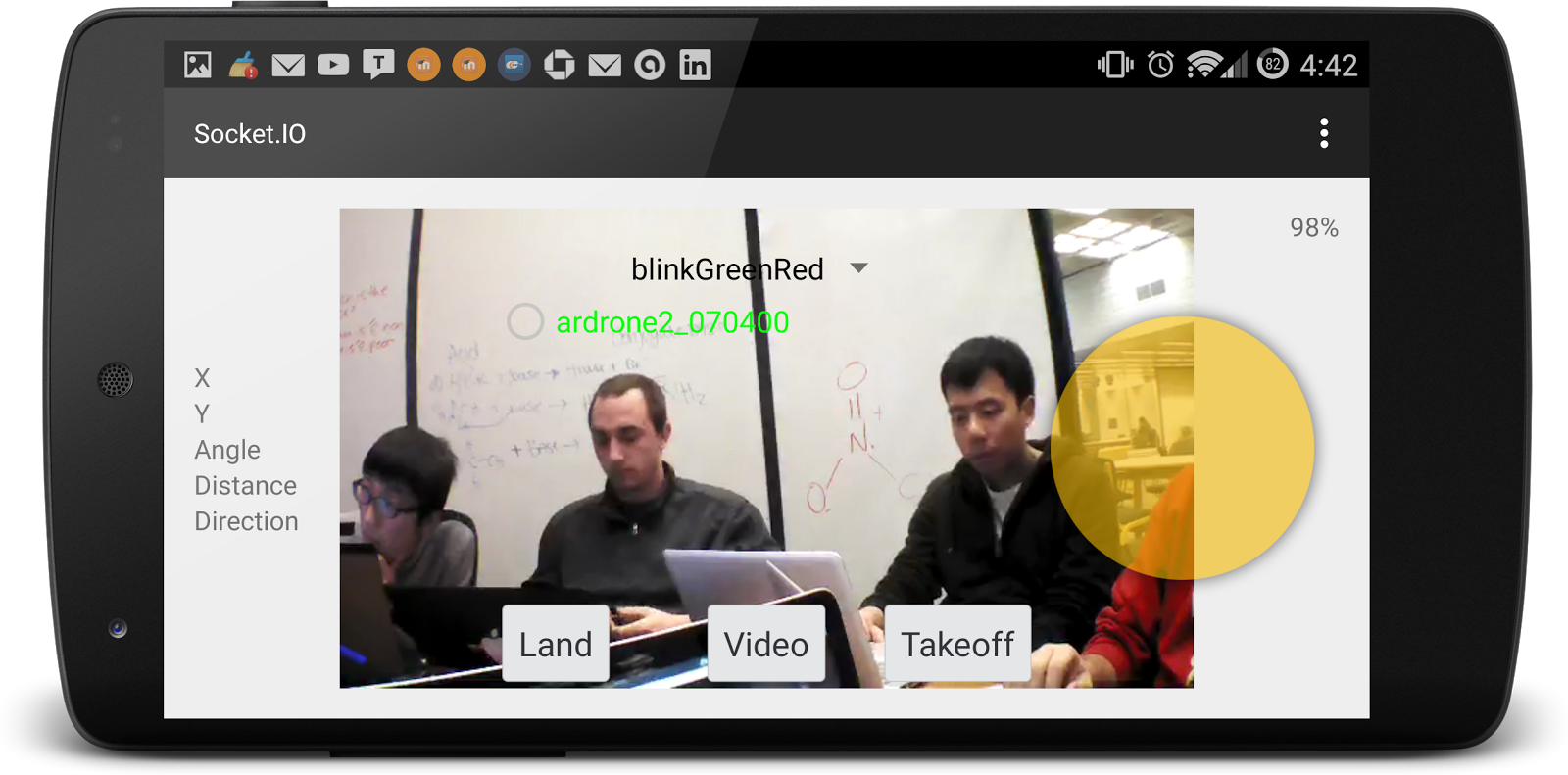
The mockup screen of the play mode is shown in *Figure C.3* There are two main areas of the screen: the status bar and the game interface. As a multi-player game, different drones will have different colors. The background of the status bar will be the color of the drone that user picked before playing. In the center of the status bar, is the game timer; it shows the time remaining of the current game. On the left of the bar, there is a wifi signal strength sign. This shows the wifi signal of the drone from the router of the server. Keeping this signal strength will allow us to detect if a player is too far away from the play area, so that the application can notify the play to go back. To the right, it is the battery of the drone. This allows the player and the application to keep track of the battery on the drone. If the drone’s battery reaches 5%, it will notify the player and go to emergency landing mode automatically. On the right side of the status bar, there is a battery life of the phone.



***Figure C.3*** *The interface for play mode*

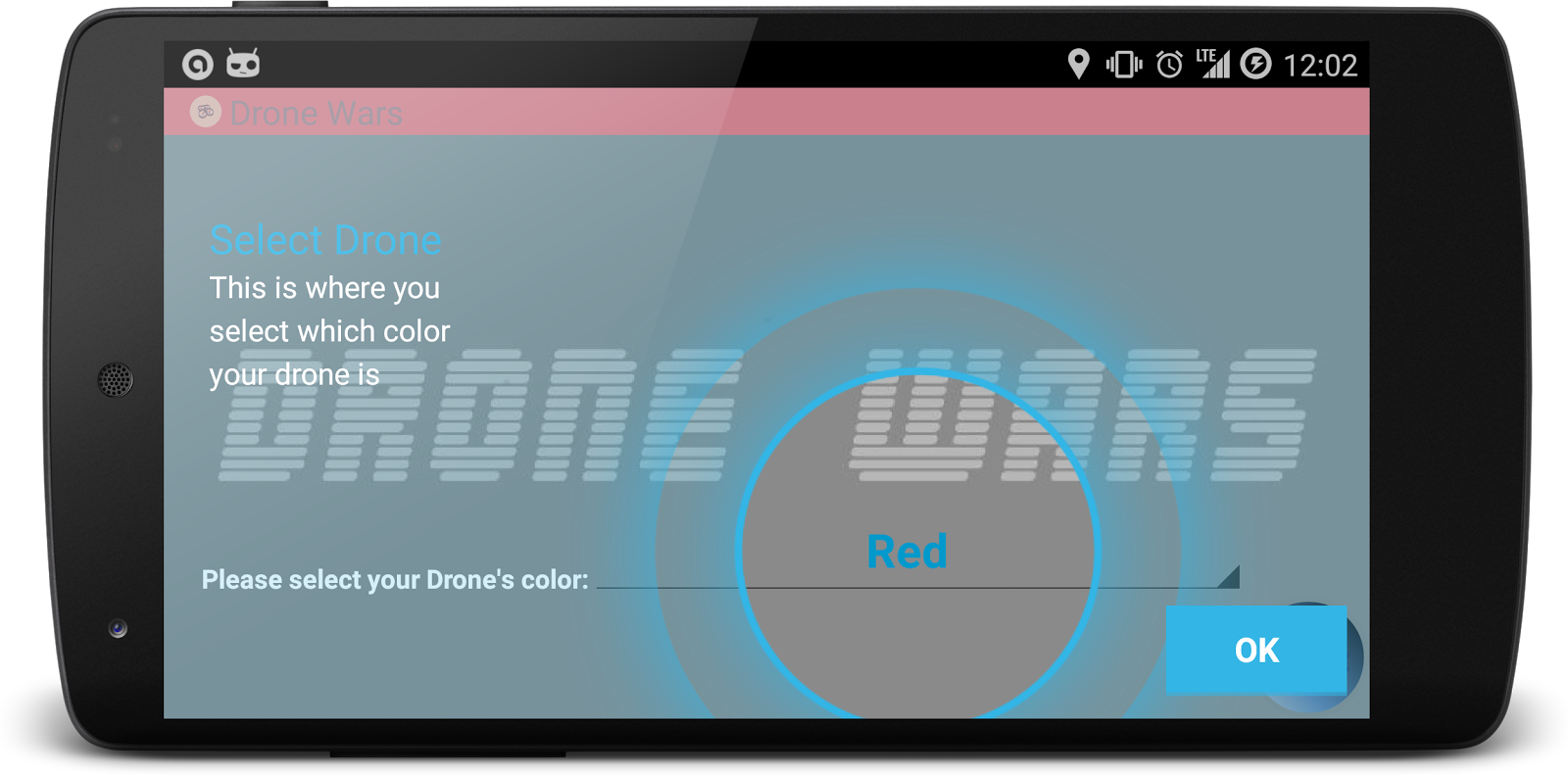
The background of the game interface will be the video feed from the drone that is being controlled. In the upper left corner of game interface, there is remaining life of the drone. In the upper right corner, there is the remaining ammunition of the player. On the left side of the screen, it shows the drones that are still in the game. There are three controls on the bottom of the screen: the accelerometer activator, the joystick/fire button and the land button. The accelerometer activator allows the player to move the drone. As the accelerometer activator is pressed, tilting the phone to control the motion. The joystick will control the drone’s facing direction. We will let the joystick be able to detect double tap, which will be the fire button. The red box on the screen is the indicator of the enemy drone, it will show whenever an enemy drone is in sight; and the red box will follow the enemy drone. In the middle of the screen, it is the target area. Whenever the player make a shot, the server will computes to see if a red box is in the target area. If so, the enemy is hit.

*Figure C.4* shows the prototype control screen which contains the joystick, takeoff, and land. These controls are going to be merged with the screen on *Figure C.3.* to form the final screen for the play screen. The video feed seen as the background of *Figure C.3* will be the background of the play screen. The yellow circle to the right is the virtual joystick which the user will use to pilot the drone. The takeoff button will send a takeoff command to the drone causing it to take off. The land button sends a land command to the drone causing it to land. The video button is on the prototype app for testing purposes only and is to be removed.



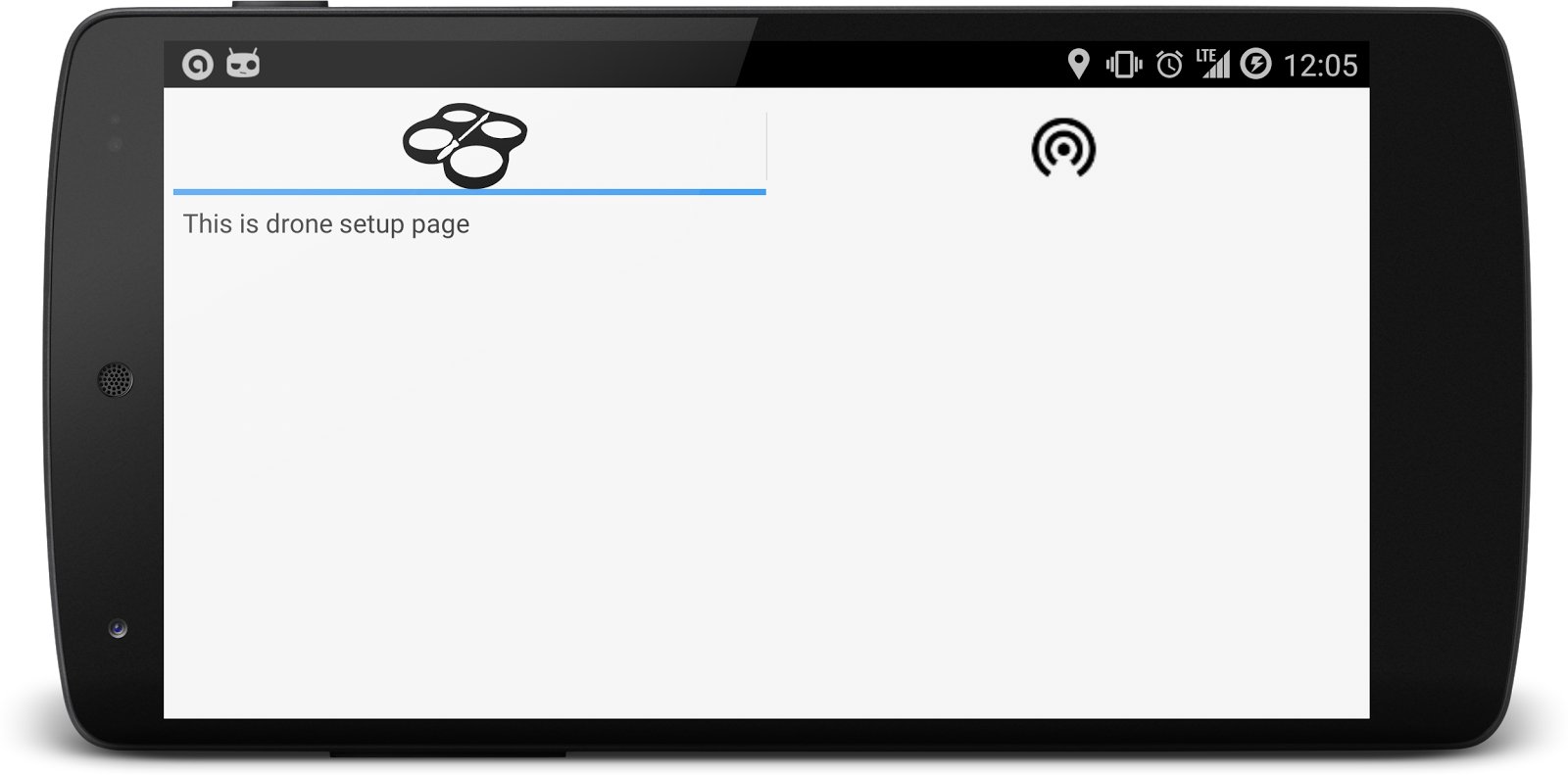
***Figure C.4*** *The Prototype buttons to be merged with the play screen*

The Instructions Screen will enable the user to read and review the instructions for different parts of the app. *Figure C.5*is the first screen that the user will see when they enter the play mode for the first time. The showcase viewer will overlay the screen to provide the user with the information needed to understand what is going on. The viewer will allow the user to participate in selecting the drone color. Once the selection has been made, the user can select the next screen.



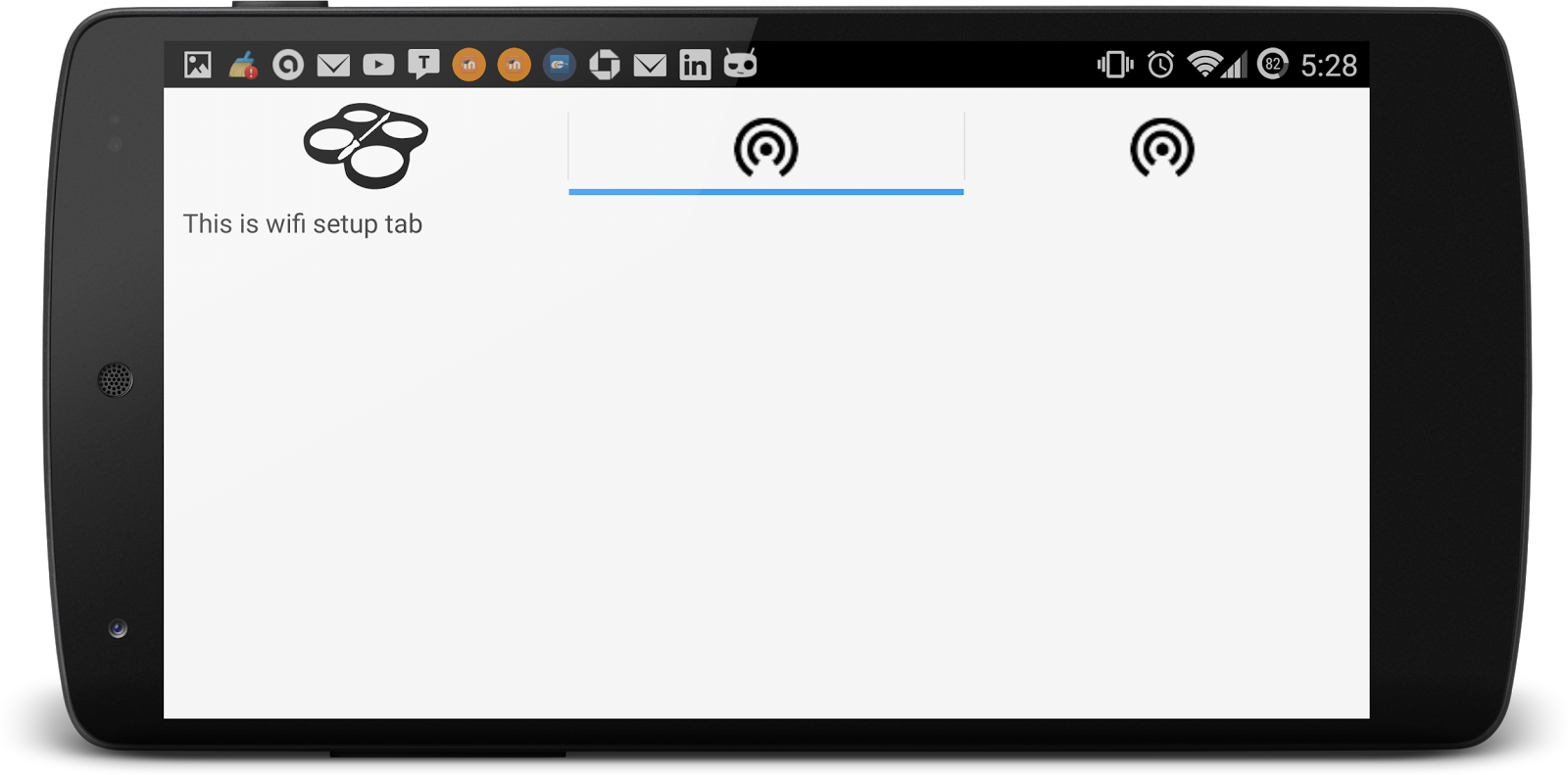
***Figure C.5:*** *Drone Selection Instruction Screen*

*Figure C.6* shows the drone setup page instruction screen. In this screen the user will be able to read instructions regarding drone setup. These instructions will help the user connect to their drone to the server so that they can then control their drone in game.



***Figure C.6*** *The drone setup page Instruction Screen*

*Figure C.7* shows the wifi setup page instruction screen. In this screen the user can learn how to connect their phone to the server via wifi. These instructions will help make clear how to configure their Android device so that they can control their drone.



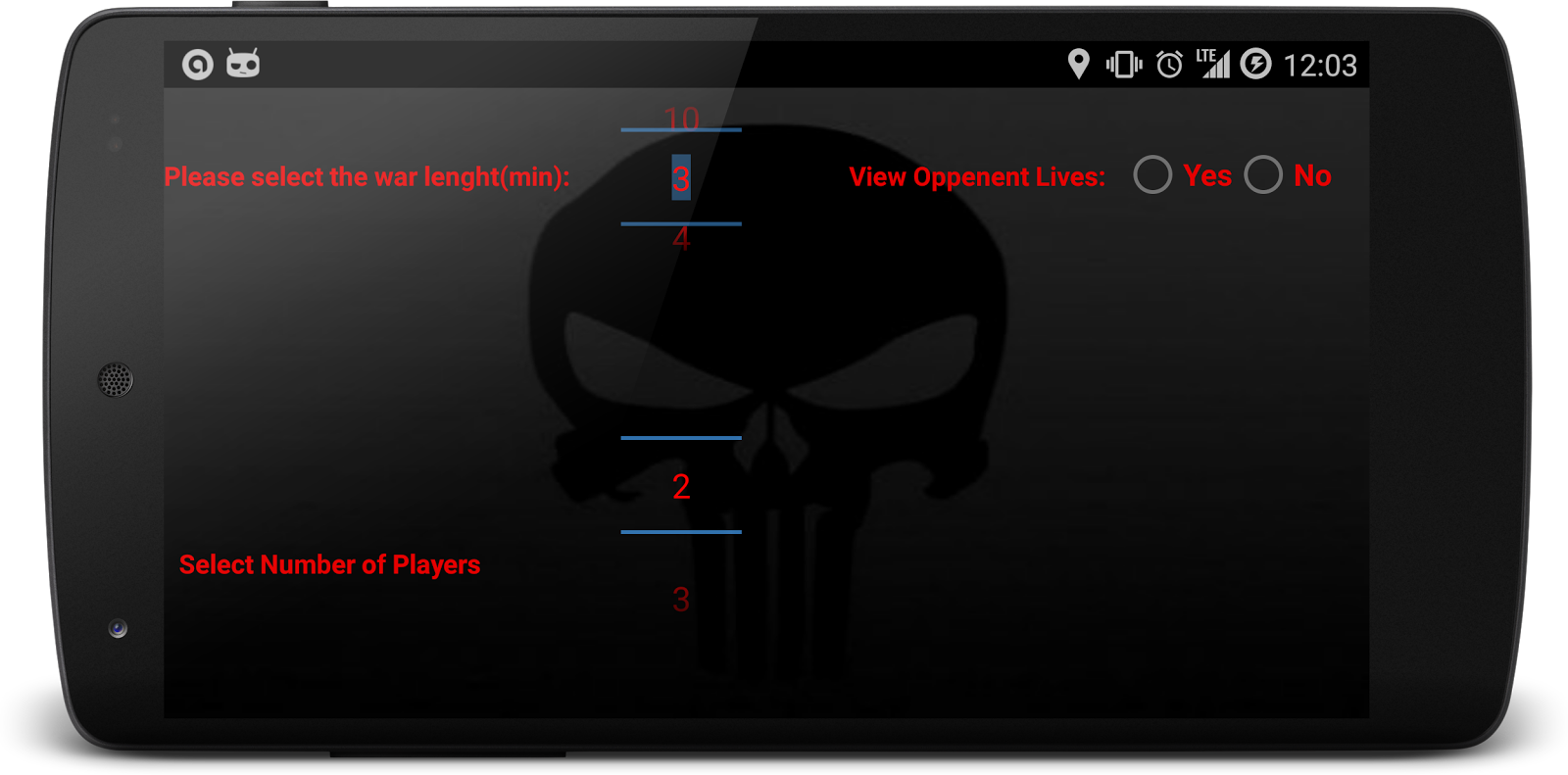
***Figure C.7*** *The WiFi Setup Instruction Screen*

The Drone Selection Screen is very minimal and will allow the user to select their respective drones. *Figure C.8* shows the Drone Selection Screen where the user will select which drone is theirs. Depending on the color the user has chosen for their drone determines which drop down they will select. The list will be dynamically populated based on the number of drones connected. The action bar will reflect the current displayed color on the page. The user can change colors before hitting the select button.



***Figure C.8*** *The Drone Selection Screen*

The Game configuration screen is where the user who selects red will set the match parameters like game time, lives, etc. *Figure C.9* shows the Game Configuration Screen. If the user elects to be the red drone, by default they will be elected into the administrative position for the match. The interface that they will see is basic parameter selection spinners. This user will be responsible for all the game properties like time of match, lives, type of flight (future) and what ever configurations pertain to the match.

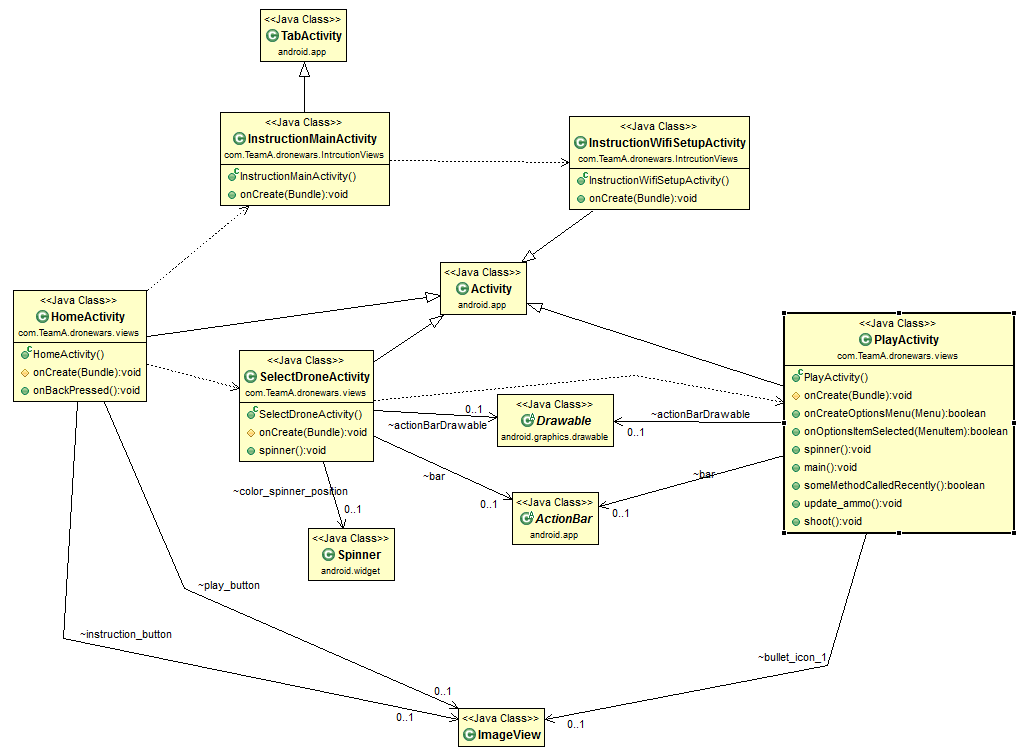


***Figure C.9:*** *Game Configuration Screen*

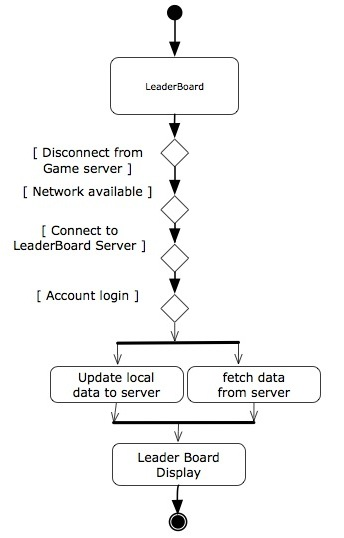
**Section D:**

**Design & Diagrams**

This section shows all of the diagrams of our application design: UML Diagram of the program, activity diagram, server sequence diagram, app sequence diagram, Android app statechart UML diagram, and server statechart UML diagram. *Figure D.1* shows the UML diagram of our application so far. We have partially implemented two parts of the app: instruction and play. As it is shown in the figure above, the instruction has a main activity, **InstructionMainActicity**, which extends **TabActicity**, and each tab is also an activity. All other classes extend Activity as well. The play part contains two activities so far: **SelectDroneActivity** and **PlayActivity**. A user will go to **SelectDroneActivity** before **PlayActivity**. **SelectDroneActivity** and **PlayActivity** have custom action bars, so they contain ActionBar class in order to be able to customize the action bar. **SelectDroneActivity** contains a spinner, so that a user can pick a color to represent his drone. The **Drawable** class is included to modify the **action bar**. Both **HomeActivity** and **PlayActivity** have ImageViews.

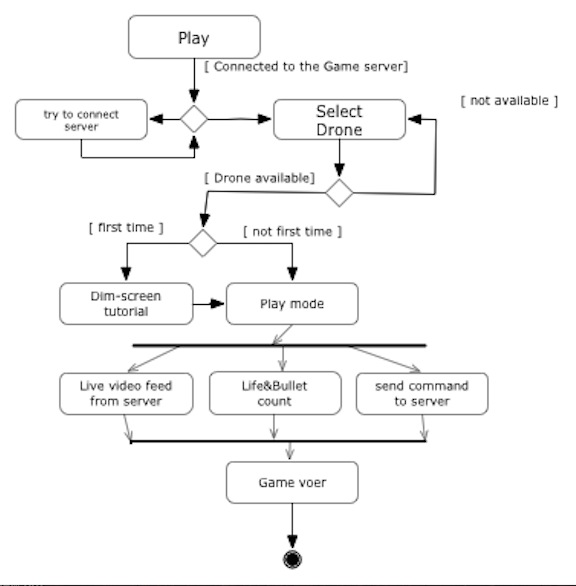
***Figure D.1*** *UML Diagram of the program. - C.C.*

The following diagrams illustrate the different states of the App that user may be in and the transition between the states. When launching the app, user will be able to select next step within 3 options. In the *Figure D.2* is **LeaderboardActivity** in which the user can check his/her rank on the Leaderboard Server. In the **LeaderboardActivity**, the app will first check if there is already an account that can be used to login into the server, if not, the user will be asked to provide a valid account and password with which the App can fetch out user data from the leaderboard server.



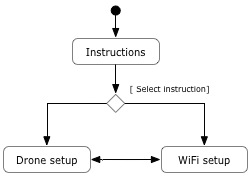
***Figure D.2*** *LeaderBoard Activity Diagram - D.L.*

In the *Figure D.3* is **PlayActivity**, which is the main activity of the App that allows user to play the drone wars. In this activity, the app will first check if the Android device is connect to the play server, if no, it will ask the user the concept to the server in order to go to next step. If successfully connect to the server, the App will allow user to select a available drone that has not been picked by other player. The server will send to information to the App that which drone is whether available or not. After picking the drone, the user will enter the play mode, which allows user to control the drone and fight against other player. If it’s the first time entering the play mode, there will be a dim-screen tutorial (The device screen will be dimmed a little bit while some of the buttons or parts on the screen will be highlighted with description in order to help user get familiar with the drone control interface). In the play mode the App will keep communicating with the server for getting live video feed and sending out the command for drone controlling. The video feed will be displayed as the background of drone control interface and through the video user can see the picture captured by the front camera of the drone. When the user press the button on the drone control interface, a command will be sent to server accordingly and then be transferred to drone. Meanwhile the app will count the game time, bullets and health points of the drone and will terminate the game when certain conditions are met (such as the depletion of health points or running out of time).



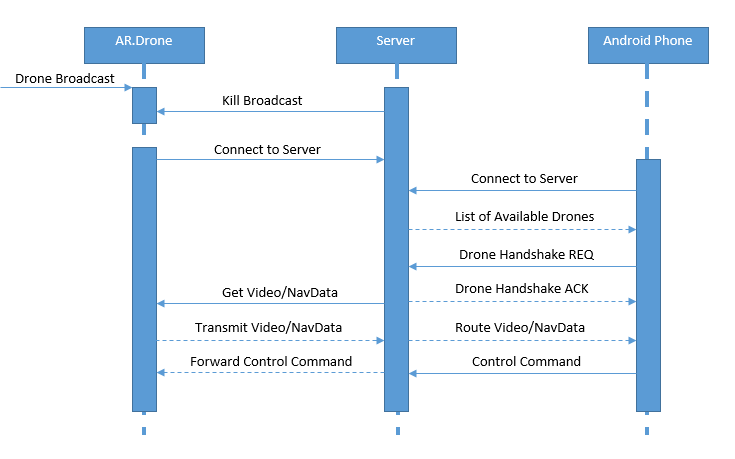
***Figure D.3 Play Activity Diagram - D.L.***

In the *Figure D.4* is the **InstructionActivity**. There will be instructions for how to setup the drone and server in order to play the game. There will also be an exhaustive tutorial about how to control the drone using the drone control interface in the play mode. All of the three activity can go back to the home screen of the App by pressing the physical return button on the device. One game session is supposed to be completed when the game ends in the play mode.



***Figure D.4*** *Instruction Activity Diagram - D.L.*

*Figure D.5* depicts the interaction between the game server and AR.Drone and Smartphone clients. By default AR.Drones wireless interface is configured to act as a host and broadcast a local wireless network to which a smartphone connects in order to control it. This default configuration is not ideal for our implementation because it severely limits the interaction between multiple AR.Drones. In order to allow communication between all connected devices we have created a server which acts as a central hub that all clients connect to. To achieve that the server has two wireless interfaces **wlan0** and **wlan1**. The **wlan1** interface is configured as an open access point for all AR.Drones and Smartphones to connect to while **wlan0 interface** is configured to force AR.Drones to connect to **wlan1 interface**.



***Figure D.5*** *Server Sequence Diagram - K.C.*

As described before when an AR.Drone is turned on it acts as a host and broadcasts a local wireless network. The server uses **wlan0 interface** to scan for wireless networks available in the area and then uses a Regular Expression to determine if the access point **MAC address** is in a specified range registered by Parrot for all AR.Drones. Once such access point is found the server initiates a connection and asks **DHCP** to provide a valid **IP address**. Once a valid **IP** is provided the server then utilizes a **TELENET** connection to login into the **Linux BusyBox** distribution running on the AR.Drone to reconfigure the **ath0 interface** from acting as a host and broadcasting to acting as a client and connect to the local wireless network being broadcasted on the server **wlan1 interface**. Smartphones on the other hand connect directly to server **wlan1 interface** and upon successful connection the server replies with a list of all AR.Drones connected to the server. The transmission contains the AR.Drone names, **MAC unique identifier** and the handshake status. Once the smartphone requests a handshake with an unpaired AR.Drone the server makes sure that the AR.Drone is still unpaired and checks if the given smartphone was previously paired with another AR.Drone and if so it performs an unpair and then finally completes the handshake with the requested AR.Drone. If the handshake is successful the server replies with the AR.Drone MAC. The smartphone then sends the MAC unique identifier along with any commands to the server which then routes the commands to the paired AR.Drone. Also all navigation data and video emitted by the AR.Drone are forwarded to the paired smartphone.

pi@raspberrypi ~/DWA $ sudo node wireless.js

[PROGRESS] Enabling wireless card...

[PROGRESS] Wireless card enabled.

[PROGRESS] Starting wireless scan...

[ANDROID][CONNECTED][192.134.3.2] Dalvik/1.6.0 (Linux; U; Android 4.4.4; A0001 Build/KTU84Q)

[AR.Drone][FOUND] Endless\_Drone [90:03:B7:EA:8E:CD] 51% 74 dBm

[AR.Drone][CONNECTED] Endless\_Drone [90:03:B7:EA:8E:CD]

[AR.Drone][90:03:B7:EA:8E:CD][IP] 192.168.1.2

[AR.Drone][TELNET][CONNECTED]

[AR.Drone][TELNET][READY]

[AR.Drone][TELNET][CLOSE]

[AR.DRONE][INITIALIZED] Endless\_Drone

[AR.Drone][FOUND] ardrone2\_070400 [90:03:B7:EA:B6:B1] 68% 95 dBm

[AR.Drone][CONNECTED] ardrone2\_070400 [90:03:B7:EA:B6:B1]

[AR.Drone][TELNET][CONNECTED]

[AR.Drone][TELNET][READY]

[AR.Drone][TELNET][CLOSE]

[AR.DRONE][INITIALIZED] ardrone2\_070400

[PAIR] 8lM8Ao13k1MC\_Nj9AAAA <---> ardrone2\_070400

[AR.Drone][ardrone2\_070400][TAKEOFF]

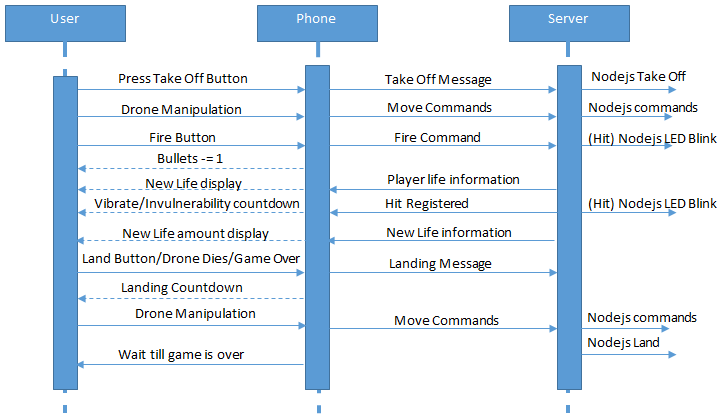
[AR.Drone][ardrone2\_070400][LAND]

[PROGRESS] Gracefully shutting down from SIGINT (Ctrl+C)

[PROGRESS] Disabling Adapter...

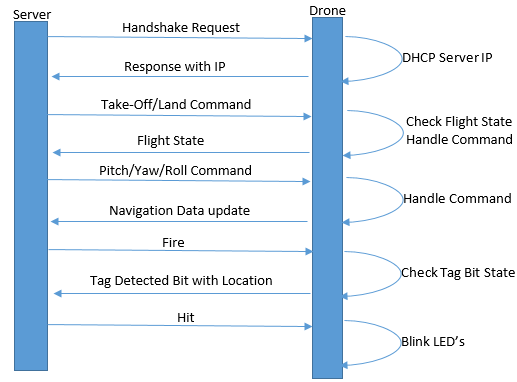
[PROGRESS] Double SIGINT, Killing without cleanup!

*Figure D.6* shows how the app will handle certain commands and actions coming from the server and user. The first thing that the user will do is issue the **takeoff command**. The game will not start until all **drones** are flying. The user will select the **takeoff** command and the **phone** will send the command to the server which will relay a **node.js** command to the **drone**. Once in the air, the user can use the **drone** manipulation buttons like left, right, up down, etc. to fly the **drone**. These commands will be again sent to the **server** and then relayed to the **drone** as **node.js** commands. If the user hits the fire button, the phone will see if they indeed have a shot remaining. If they do, the command is sent to the server which will see if the enemy **drone** is in the target area. At the same time, the phone will update the bullet display to reflect the shot fires. If a hit is registered, the **server** sends a **node.js** LED animation command to visually show that the drone has been hit and entered an invulnerability state. At the same time, the **server** will not allow the affected **drone** to take any damage while in this state. The **server** will send the new life data to the phone and the phones display will update to reflect the data. When the user hits the land button, loses all their lives, or the game ends, the phone will send a land command to the **server**. The **server** will start a timer and show it to the user. The user will have the specified amount of time to get to a safe location to land. During this time, the player can only move the **drone**; all other game functions are disabled. Once the **drone** has landed safely, all commands are disabled until all drones are landed and the playing field is safe and the game is over. The player must wait until the game is over.



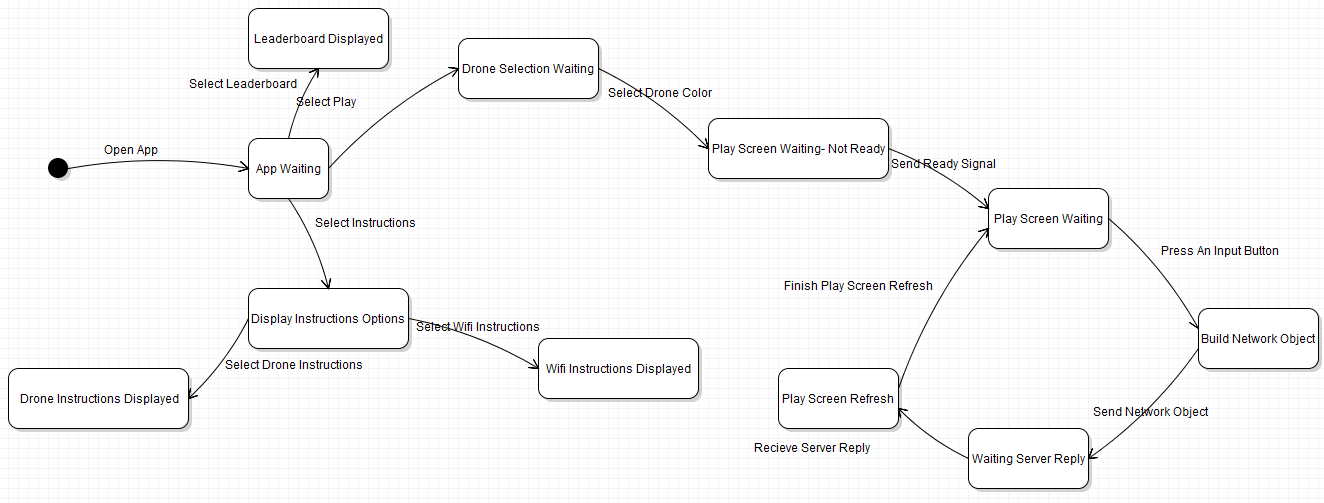
***Figure D.6*** *App Sequence Diagram - D.P.*

Figure D.7 shows a server to drone sequence diagram. The diagram shows how the drone will handle commands sent to it from the Node server. The commands are predefined by parrot and as we are using an unchanged Drone, the responses are unchanged. The only thing that is changed is the **handshake** protocol. By default, the **drones** are configure to be **wireless hosts**. For this project we need them to become clients. We initiate a **handshake protocol** and have the **unix** system issue an **IP address** to the **drone**. The **drone** returns the **IP address** to the **server** so the **server** knows how to communicate with it. The other custom command is a fire command. This will be handled on the server side with a query of the **NavData** to acquire the state of the **Tag Detection bit**. If the **bit** is 1, the tag is within the frame. We then must check the location of the tag, also provided through the **navdata**. This data is interpreted by the server. If a hit is registered, the **command** is sent to the **drone** to blink its **LEDs**. The other stock commands take-off, land, emergency, tag detection, etc., are all handled locally on the **drone**. There is nothing that has been changed in that regards.



***Figure D.7*** *Drone Sequence Diagram - D.P.*

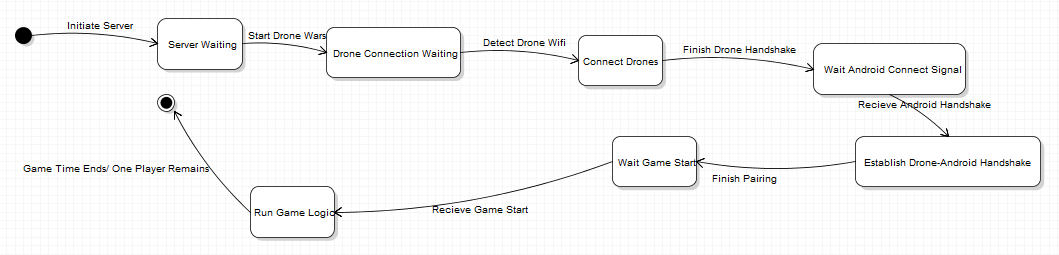
*Figure D.8* shows the statechart diagram UML for the Android application. The machine starts when the user opens the app. From there the application will be in a waiting state where it is expecting input from the user. If the user selects the **Leaderboard** section of the app the machine moves into the **Leaderboard Displayed** state. As the name suggests, this state is where the user will view the leaderboard. The user can also leave the **App Waiting** state by selecting the **Instructions**. This will bring the user into the **Display Instructions Options** state where the user will leave by choosing either the **Drone Instructions** or the **Wifi Instructions**. Depending on what they select, the user will move into the **Display Drone Instructions** or **Display Wifi Instructions**, respectively.



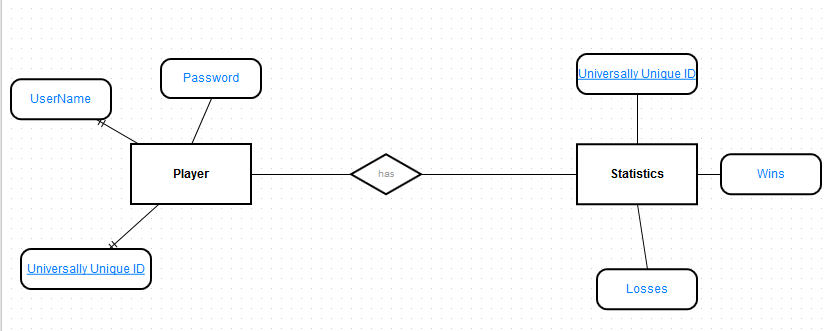
***Figure D.8*** *Android App Statechart UML Diagram - P.Z.*

The last way that the **App Waiting** state can be left is if the user select the **Play** option. Once the user selects the **Play** option, the machine moves into the **Drone Selection Screen Waiting** state. This state can only be left once the user selects which **drone** they wish to pair with. Upon leaving, the **Play Screen Waiting** **- Not Ready** state will be entered. This can only be left when all players select that they are ready to begin the game. From there the **Play Screen Waiting** state is entered. This state is constantly looped to. The state is left when the user presses an input button on the screen to control the **drone**. The machine will then enter the **Build Network Object** state. This state will build the network object to send to the server. Once the object is sent, this state is left and the **Waiting Server Reply** state is entered. This state state will be left when the server sends back a response. At this point the **Play Screen Refresh** is entered and the visuals on the screen are updated. This state is left and the **Play Screen Waiting** is re-entered when the **Play Screen** is finished refreshing. Every state is able to the **App Waiting** state by transitioning on the back button of the device being pressed. Likewise, every state is able to end the state machine by transitioning on the event of the **home button** being pressed. In order to simplify the diagram these were left out.

*Figure D.9* show the statechart UML diagram for the server. The state machine begins when the server is booted up. At this point it has entered the **Server Waiting** state where it waits for the **Drone Wars** application to start. When the **Drone Wars** application starts the machine enters the **Drone Connection Waiting** state. When the server detects **Drone Wifis** then the machine will enter the **Connect Drones** state. This state is left when the **drones** finish the handshake with the server. At this point the machine will be in the **Wait Android Connect Signal** state. This state will be left when the server receives the Android handshakes. The machine will then enter the **Establish Drone-Android Handshake** state. This state can only be left when the pairing between the Android devices and the **drones** is complete. At this point the machine will be in the **Wait Game Start** state. This state is left when the server receives the game start signal. The server will then be in the **Run Game Logic** state. Finally, the machine will finish when either the game time ends or only one player remains.

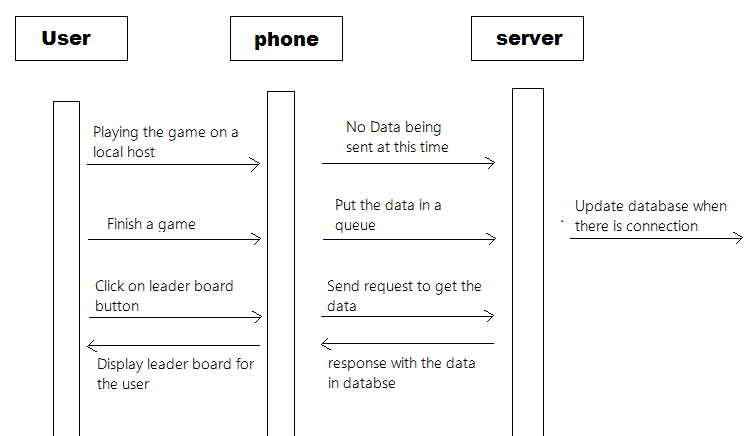
***Figure D.9*** *Server Statechart UML Diagram - P.Z.*

Our database will be very simple and only contain two tables; as seen in *Figure D.10,* the first table is the **Player** table and the second is the **Statistics** table. Included in the **Player** table is the player’s username and password as well as a generated **Universally Unique ID.** The **Universally Unique ID** is the primary key of this table and will allow the table to be searched upon. Included in the **Statistics** table is a **Universally Unique ID** and wins and losses corresponding to that **Universally Unique ID**. The **Universally Unique ID** in the **Statistics** table is both a primary key and foreign key. When querying inside of the **Statistics** table, the **Universally Unique ID** is a primary key. However, it is a foreign key corresponding to the primary key in the **Player** table. This is done so that each player can be queried uniquely but without having any attachment of the player’s chosen **UserName** or **Password**.



***Figure D.10*** *Leaderboard Entity Relationship Diagram - P.Z.*

*Figure D.11* is the sequence diagram of the **leaderboard**. Since users play the game in a local network, the phone does not have access to the server when they are playing the game, which holds the database of the leaderboard. Therefore, when the users are playing the game, there is no data being sent to the server. However, after each game, data is saved in the phone and put in a queue. The application will try to send and update the data to the **server** when the phone regains connection to the internet. When a **user** presses the leaderboard button, the application will send a get-request to the server to get all the data and then the data will be displayed to the **user**.



***Figure D.11*** *leaderboard sequence diagram - C.C*

**Section E:**

**Test Plans**

Test #1 **Ammunition limitation**

Ensuring that the Players ammunition is regenerated 15 seconds from when the clicked the fire button in sequence. The ammunition regeneration timer was taken of from the server and implemented in the PlayActivity of DroneWars. A boolean is used to keep track of when the reloading process is finished so the handler can start on the next bullet. The Ammunition limitation test will also compare the drone average flight time and compare how many shots can be fired within that time frame to ensure that enough ammunition regenerates within that time frame to make the game enjoyable and fair.

*Test Results* - PASS

Test #2 **Action Bar In Play Screen**

In our play activity, we made a custom action bar to hold all the information we need to show to the users. First, when a user select a color for his drone in select drone activity, the action bar will dynamically change to that color. After selecting the color, the action bar in the play mode will also have the same color. And the action bar in both activities will be 50% transparent. This test case will focus on testing how the action bar reacts when a different color is selected. It should dynamically change to correspond color and to be 50% transparent.

*Test Results* - Pass

Test #3 **Connect A Drones to Single Network**

Because by default the drones transmit their own wifi signal, there would be no way for them to communicate. For this project we needed the drones to connect to a single network. The team wrote code to force a drone to connect to a network of our choice. This connection should happen every time we run the code.

*Test Results* - Passed

Test #4 **Multiple Drones to a Single Network**

The game is planned to be a multi-player application so we need more than one drone to connect to the network. The code must handle up to three drones and have those all connect to the same network of our choice. This connection should happen every time we run the code.

*Test Results* - Passed

Test #5 **Drone Selection List Update**

With the drones connected to the network and the app, the list that is displayed on the screen must update when a player selects to pair with that drone. This means that the drone which was selected by the player must appear selectable to other players. This must happen dynamically and in real time.

*Test Results* - Pass

Test #6 **Android Device/Drone Handshake**

When an Android Device connects to the server it receives a list of all available AR.Drones. Once an AR.Drone is selected, the handshake sequence is initiated and a Handshake Request is sent to the server to map the current device to the requested AR.Drone. When the handshake is successful a Handshake Acknowledgment is sent to the Android device. At the same time the server sends a request to the AR.Drone to forward its video stream and navigation data through the server to the paired Android device. Test will include verifying list of available AR.Drones and success rate of pairing to AR.Drones.

*Test Results* - Pass

Test #7 **Home Screen Button Response**

The home screen will house three different buttons. Each one will do something different. The Play button will have two different behaviors depending how many times the game has been accessed. If the user is accessing the game for the first time, they are forced into the walkthrough instructions. This will walk the user through the process of selecting drones and what all the buttons and icons mean on the screen. If it is not the first time, the button will take the user straight to the drone selection screen and allow an unhindered access to the game. The Instruction Button will bring the user to the instruction screen. The leader board button will bring the user to the sign in page if they are not signed in. Once signed in, they will be brought to the leaderboard interface.

*Test Results* - Pass

Test #8 **Drone Control**

The Drone is expected to behave a predetermined way with predetermined inputs. If the user provides a takeoff command while the drone is landed, the drone is expected to take off. If a land command is used, the drone is expected to land, if flying. The drone must respond as expected with all combinations of user input.

*Test Results* - Pass

Test #9 **Tutorial**

When the user first opens selects the Play option they will be guided through a tutorial. In this tutorial they will be shown the different features of the Drone Selection Screen and the Play Screen. This tutorial will be deemed successful if all supported features are shown in the tutorial and explained regardless of current application conditions.

*Test Results* - Pass

Test #10 **Router**

The router will be tested to ensure that connections are maintained throughout the entirety of the game. The range and signal strength to the Drones as well as the range and signal strength to the phones will be tested. Furthermore, a stress test to ensure that all video feeds are smooth will be performed.

*Test Results* - Successful Connection on both 2.5 and 5 GHz bandwidth

**Test Plan Sprint 3:**

The plan for sprint 3 is to reuse the test plan for Sprint 2 when more gets implemented. Some additional test cases:

Test #11 **Router Access Point -**

The router will be tested to ensure that connections are maintained throughout the entirety of the game. The range and signal strength to the drones as well as the range and signal strength to the phones will be tested. Furthermore, a stress test to ensure that all video feeds are smooth will be performed.

*Test Results* - Pass

Test #12 **Server Latency**

The server will be a centralized location through which the video stream and navigation data from all connected AR.Drones as well as control signal from all connected Android devices will be routed through. The tests will include measuring the video latency from the drone to the Android device and the latency of control signals from the Android device to the AR.Drone. Need to make sure both latencies are as small as possible to create a smooth play experience.

*Test Results* - Passed - Minimal Latency

**Test Plan Sprint 4:**

Test #13 **Server Latency**

The server will be a centralized location through which the video stream and navigation data from all connected AR.Drones as well as control signal from all connected Android devices will be routed through. The tests will include measuring the video latency from the drone to the Android device and the latency of control signals from the Android device to the AR.Drone. Need to make sure both latencies are as small as possible to create a smooth play experience.

*Test Results* - Passed - No latency was detected when passing controls to the server.

Test #14 **OpenCV**

During gameplay if an enemy AR.Drone is seen in the video crosshairs, the enemy needs to be identified by OpenCV using its colored tag. OpenCV will either run on the server preprocessing the video streams identifying which drones are visible on the given video stream before streaming it to the Android device, alternatively OpenCV will run natively on the Android device analyzing the received video stream. OpenCV performance running on both the server and Android device will be evaluated and the one that provides best performance and least amount of implementation complexity will be used in the final product.

*Test Results* - Passed - OpenCV ran successfully on Android devices with a delay of 3 seconds. In the next sprint we plan on optimizing and testing again.

Test #15 **Video Transmission**

We will need to ensure that video is successfully passed to the phones with a stable frame rate. While playing the game the user will be controlling their drone primarily through video fed from the drone. We need to test to make sure that the video stream is stable and that there is minimal packet loss.

*Test Results* - Passed - The video stream properly transmits and is displayed successfully on the Android devices. There is slight latency with the video feed as well as some connection timeouts. These issues will be addressed in Sprint 5.

Test #16 **Game Logic**

The game will operate under strict terms and will be finished under a time/lives constraint. We need to test to make sure that when either: 1. the time runs out or 2. When there is only one player with at least one life left the game finishes. This test will ensure that the game starts and stops properly with no interruptions while running.

*Test Results* - Passed - There were no complications with the game logic.

Test #17 **Home Screen Logic**

So far we have implemented two of three buttons on the home screen: instruction and play buttons. Based on whether the application is first time open or not, the button will direct the user to different activities. For example, if the play button is selected the first time, the user will go to instruction activity automatically to ensure that the user know how to use our app before he actually play the game. And there will be a check box to indicate “Don’t show this again”. This test will focus on the flow of the app, to make sure that the design is as friendly as possible.

*Test Results* - Not Implemented - Planned for Sprint 5

Test #18 **Acquiring Target**

Acquiring target test involves checking whether the enemy drone can be recognized when they are within the target area in the mobile app. This test will include testing whether the applications crosshairs x and y coordinates will have to be mapped, or the servers image detection could be used to determine if an enemy drone enters the players crosshairs with OpenCV. Both methods will be tested to determine which the best solution is for us.

*Test Results* - Not Applicable - Test in Sprint 5

**Test Plan Sprint 5:**

Test #19 **Home Screen Logic**

(Same as above test in Sprint 4) So far we have implemented two of three buttons on the home screen: instruction and play buttons. Based on whether the application is first time open or not, the button will direct the user to different activities. For example, if the play button is selected the first time, the user will go to instruction activity automatically to ensure that the user know how to use our app before he actually play the game. And there will be a check box to indicate “Don’t show this again”. This test will focus on the flow of the app, to make sure that the design is as friendly as possible.

*Test Results* - Planned for Sprint 5

Test #20 **Acquiring Target**

(Same as above test in Sprint 4) Acquiring target test involves checking whether the enemy drone can be recognized when they are within the target area in the mobile app. This test will include testing whether the applications crosshairs x and y coordinates will have to be mapped, or the servers image detection could be used to determine if an enemy drone enters the players crosshairs with OpenCV. Both methods will be tested to determine which the best solution for us is.

*Test Results* - Test in Sprint 5

Test #21 **Video Transmission Optimization**

Previously, in Sprint 4 we tested video transmission and found it to have some latency. During this sprint we plan on optimizing this video transmission and hope to reduce the latency so that it is not noticeable to the user.

*Test Results -* Test in Sprint 5

Test #21 **Button Logic**

When a user selects a color, that color is removed from the pool of available colors. There is logic on the server and in the Android application to make this work accordingly. The test will ensure that once a color has been selected no other user can select that color. Also, if a user goes back in the app their selected color is freed and available for use again.

*Test Results* - Test in Sprint 5

Test #22 **Instructions Screen**

With our Instruction screen implementation we need to make sure that it behaves properly. We need to make sure that all Instructions are displayed properly and flow without any errors.

*Test Result -* Test in Sprint 5

Test #23 **Game Setup Logic**

When Drone Wars is setup up there are a couple steps that need to be performed to make sure the game gets set up properly. This test will include testing the server and application logic when the game host disconnects. We need to make sure that when the host disconnects all players are backed out and their colors are freed. Also, we need to make sure that the host cannot start the game until all players have paired with their drones and picked a color.

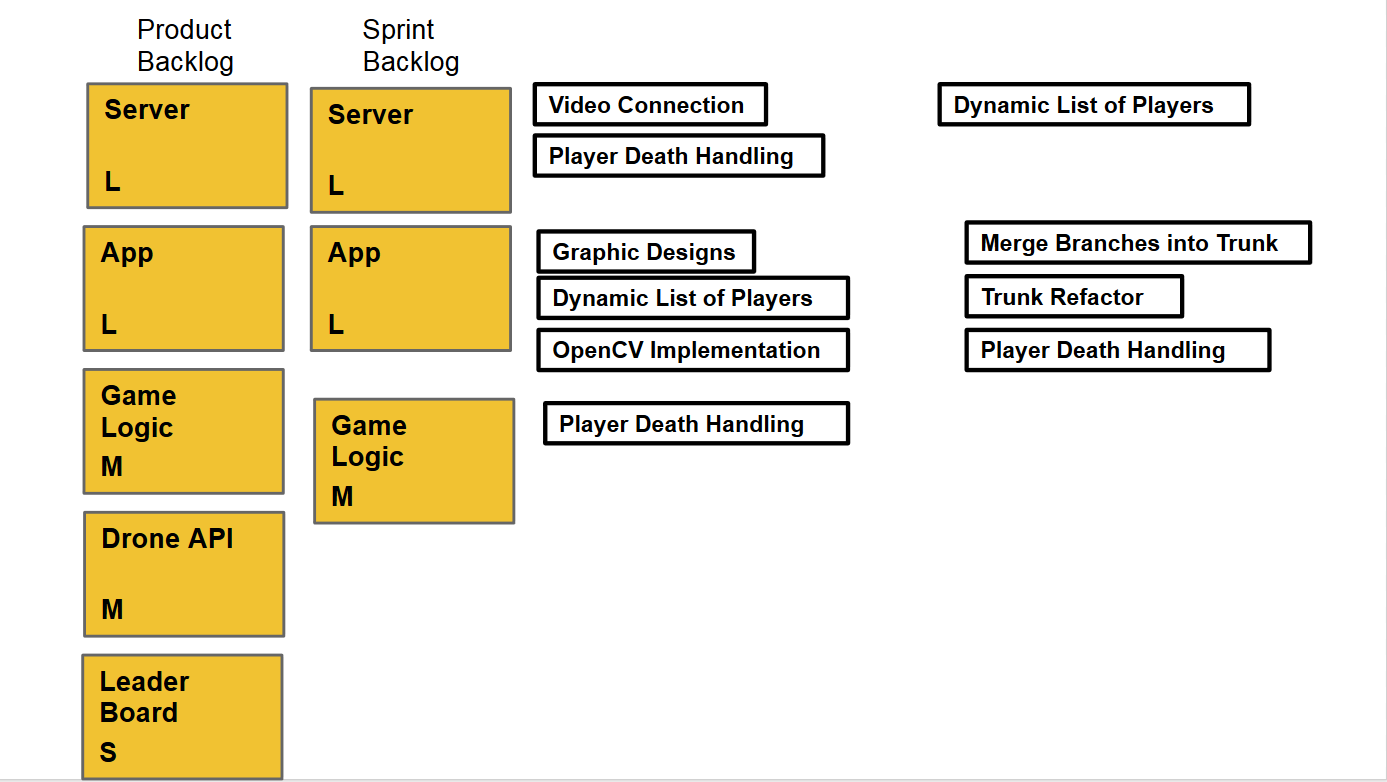
*Test Results -* Test in Sprint 5

Test #24 **OpenCV Optimizations**

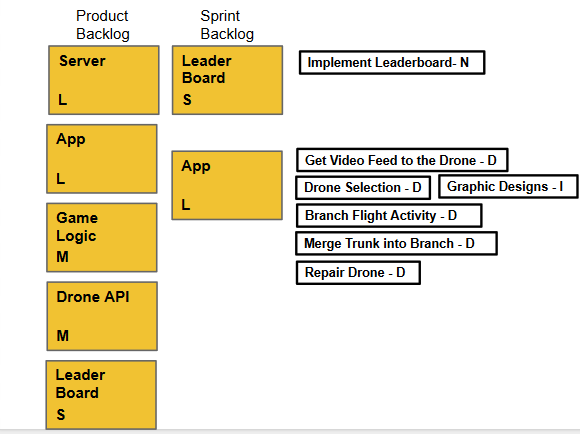
As of Sprint 4 OpenCV correctly detects a desired tag. However, there is a delay of about 3 seconds to properly detect the tag. In this sprint we need to optimize the code and test to make sure that the image detection finishes in under 500ms.

*Test Results -* Test in Sprint 5

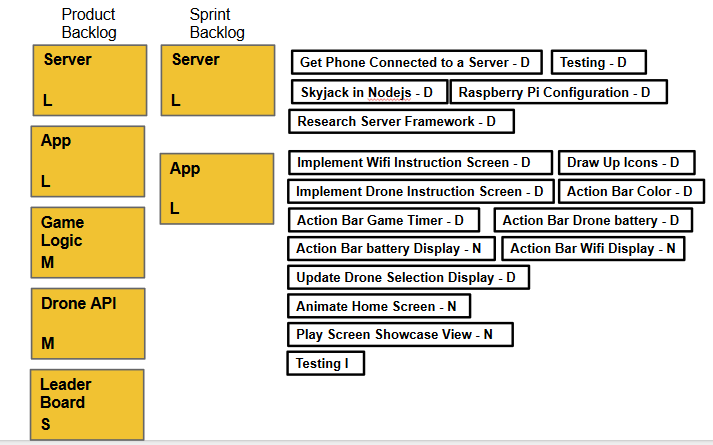
**Product Backlog**

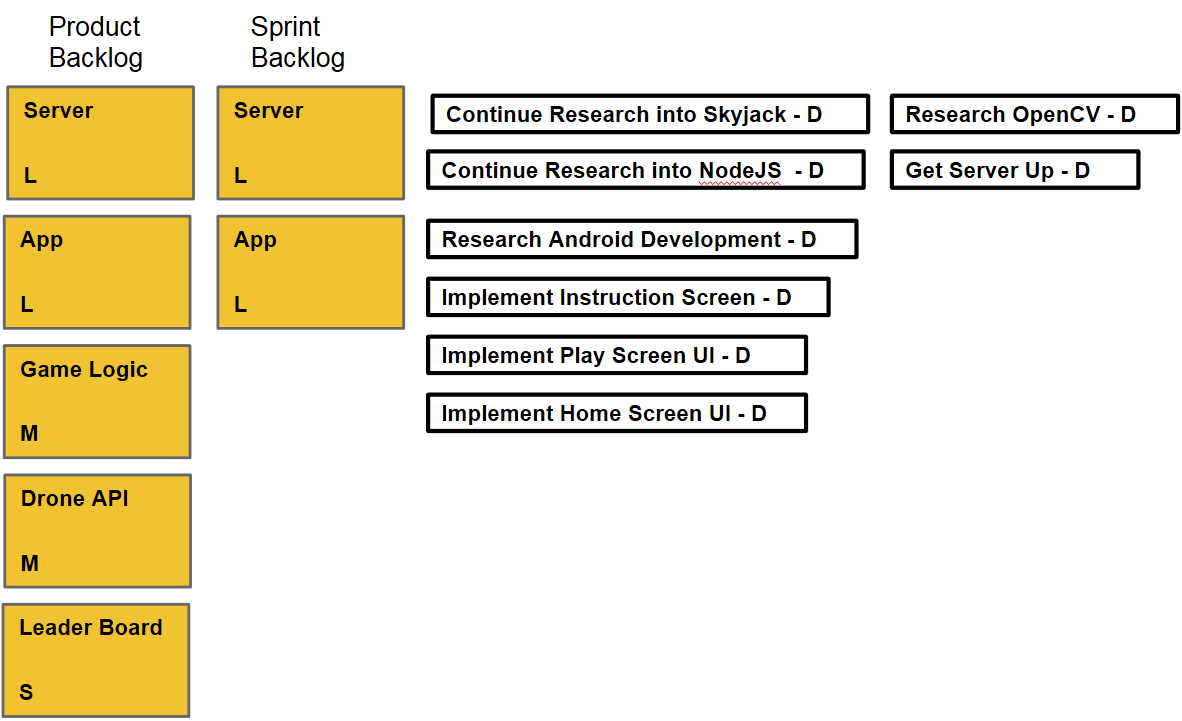
**Sprint 5:**

**Sprint 4:**



**Sprint 3:**



**Sprint 2:**

**Sprint 1:**

