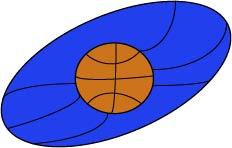
Coding Assignment

Basketball Vision - Team 3

CS 499 Senior Design Project - Fall 2016



***November 14, 2016***

***Document Authors:*** Jared Becker, Byron Giles

     Kendall Weihe, Thomas Underwood

***Document Contributors:*** Matthew Resch

***Document Editors:***  Jared Becker

Table of Contents

Implementation ………………………………………………………………………………………..…1

Source Code ………………………………………………………………………………..…...1

Quality Assurance Review ………………………………………………………….....1

User’s Manual …………………………………………………………………………………...2

Downloading the Application to Your Smartphone …………………………………2

Accessing the App for New Users ……………………………………………………2

Accessing the App for Existing Users ………………………………………………..2

Navigating the Application …………………………………………………………….2

User the User Microcontroller …………………………………………………………3

Administrator’s Manual …………………………………………………………………………3

Testing …………………………………………………………………………………………………….4

Android & iOS Testing ………………………………………………………………………….4

Test Cases ………………………………………………………………………………4

Integration Testing ……………………………………………………………………...6

Acceptance testing ……………………………………………………………………..7

Hardware Testing ……………………………………………………………………………….7

Test Plan ………………………………………………………………………………...8

Integration Testing …………………………………………………………………….10

Acceptance Testing …………………………………………………………………..10

Quality Review for Testing ……………………………………………………………………10

Metrics …………………………………………………………………………………………………...11

iOS Metrics ……………………………………………………………………………………..11

Android Metrics ………………………………………………………………………………...12

Hardware Metrics ………………………………………………………………………………13

Developer’s Notes and Website ………………………………………………………………………13

# 1 Implementation

## 1.a Source Code *(Jared)*

All source code for Android development is hosted on BitBucket and protected from public view. Thomas has given Dr. Hayes read access to the source code. If anyone needs additional access to the source code, email Thomas at [thomasunderwoodii@gmail.com](mailto:thomasunderwoodii@gmail.com).

All source code for iOS development is available for view at <https://github.com/KendallWeihe/Basketball_Vision>.

All source code for system development can be found at

<https://github.com/ProdigiousMelon/Arduino>.

### 1.a.1 Quality Assurance Review

A quality review of source code was held at our team meeting on Sunday, November 13th. The review followed the following checklist:

1. Is the code organized, commented, and easily readable?
   1. All source code was satisfactory.
2. Is the code refactored?
   1. All source code was satisfactory.
3. Does the code follow current code versions standards?
   1. All source code was satisfactory.
4. Is the code optimized for efficiency?
   1. All source code was satisfactory.
5. Are there any global variables that can be removed or replaced?
   1. There are some global variables in iOS development currently implemented as a work around for a library that wasn’t working as expected. There are plans to remove this.
6. Is the code modular?
   1. All source code was satisfactory.
7. Is there code commented out?
   1. Some of the code has commented out code like print statements. These statements are being used for testing and debugging purposes with plans to remove them for the final version.
8. Has testing been performed?
   1. All source code has been tested. A more in depth QA for testing was completed and outlined in the Testing section of this document.

## 1.b User’s Manual *(Jared)*

### 1.b.1 Downloading the App to Your Smartphone

The application is compatible with iOS and Android devices. iOS users can download the application from the App Store while android devices can download the application from the Google Play store. *It should be noted the application is still in development and is not yet available for download on these platforms.*

### 1.b.2 Accessing the App for New Users

New users should first download the application to their smartphone. On the app startup, users will be prompted by the sign-in screen. The application requires a Gmail account to register. New users will sign-in using their Google account credentials. A successful registration will automatically sign-in the user in, load the application’s dashboard, and add their account to the application’s database. Users with an incorrect sign-in will be prompted to re-enter his/her username and password.

### 1.b.3 Accessing the App for Existing Users

Existing users already have an account in the application’s database. On startup, the application will perform one of two options:

(i) Load the application’s dashboard if automatic sign-in is enabled.

(ii) Prompt the user to sign-in with their account information.

A successful sign-in to a user’s account will load the application’s dashboard. An unsuccessful sign-in will report that either the username or password is incorrect and instruct the user to try again.

### 1.b.4 Navigating the Application

The user experience begins at the dashboard. The dashboard is the application’s main landing page and homepage for the interface. The dashboard contains the application’s global navigation. The user uses the dashboard to access the following pages:

(i) Games

(ii) Statistics

(iii) Settings

Each page contains a link back to the user’s dashboard.

The *Games* page gives the user option of choosing which game they intend to play. Players can choose from HORSE, three-point contest, two-point contest, free-throw contest, and a two-minute shootout. Clicking the icon for a game will initiate a game for the respective selection. Games collect and save the data from each game to update the user’s statistics.

The *Statistics* page stores the statistics for a user’s current session as well as their overall history. The page displays percentages for games by game type. Selecting a specific game type will generate a graph for the respective game.

The *Settings* page allows the user to adjust both application and account settings. Here a user can choose to enable/disable automatic sign-in and update other settings relative to their account and profile.

### 1.b.5 Using the User Microcontroller

In addition to interacting with the application, the user will be required to wear a microcontroller. The microcontroller is what allows the user to communicate with the system. The microcontroller has two buttons:

(i) One button to report the user’s location on the court and distance to the basket

(ii) A second button to inform the system a shot was taken

The microcontroller is designed to be worn on the forearm or attached to a pants’ waistline.

If the user presses (i), the system will determine the (x,y) position of the user on the court and inform the player of his/her location relative to the basket - near the hoop, near the sidelines, on the free throw line, etc. The system will also report the distance to the basket so the player can determine if they are too far or too close for a shot.

The user should press (ii) once they make a shot. The system is able to automatically detect if a shot is made. A successful detection will send a message to the user’s smartphone application notifying the player and update his/her statistics in the database. The button’s primary purpose is to detect if the shot is missed. If the system receives no detection of a made basket within 10 seconds since the button was pressed, the system assumes the shot is a miss. The system will then send a message to the user indicating the results. It will also update the statistics to the database.

## 1.c Administrator’s Manual *(Byron)*

Requirements

* Software
  + IOS/Android
  + Windows/OSX
  + Arduino IDE
* Hardware
  + RFM node X 4
    - Arduino Uno
    - RFM69HCW
    - Logic Converter
    - Bluetooth Module (just one)
  + Headphones
  + Smartphone

First the basketball vision app must be installed on the mobile devices. The nodes are then powered on and set their prescribed distances from one another. The player then logs in and the application is ready to use.

# 2 Testing

## 2.a Android & iOS Testing

### 2.a.1 Test Cases *(Kendall, Thomas)*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case description | Input(s) | Pass condition(s) | Fail condition(s) | Status |
| Authenticate user with Google account | User enters Google login credentials | App goes to home screen | Error message: “Wrong password. Try again.” Also allow the user to tap “Done” and exit the login page. | PASS |
| User starts new instance of app | User opens the app | App opens, (may) automatically log user in, and blanks the user’s statistics for the current game session | App fails to open, fails to automatically log the user in, or fails to blank the user’s current session statistics | PASS |
| Go to Statistics page | User taps “Statistics” on the home page | App navigates to Statistics view | App does navigate to Statistics view and stays on home page | PASS |
| Go to Games page & all views within Games page | User taps “Games” on the home page | App navigates to Games view | App does not navigate to Games view and stays on home page | PASS |
| Go to Settings page | User taps “Settings” on home page | App navigates to Settings view | App does not navigate to Settings view and stays on home page | PASS |
| Simulate a made shot | User taps “Simulate made three-pointer” or “Simulate made two pointer | Database elements are updated with new values | App fails to update database elements | PASS |
| Simulate a missed shot | User taps “Simulate missed shot” | Database elements are updated with new values | App fails to update database elements | PASS |
| Get statistic from entire history | User taps one of the buttons under “Entire History” on the Statistics page | App reads aloud the statistic value from the database | App fails to retrieve the statistic from the database, or fails to read the value aloud to the user | PASS |
| Get statistic from current game session | User taps one of the buttons below “Current Session” on the Statistics page | App reads aloud the statistic from the database | App fails to retrieve value from database or fails to read value aloud to the user | PASS |
| Tap on the “Home” button on any of the pages other than the Home page | User taps on “Home” | App navigates to Home view | App fails to Navigate to Home view and stays on current view | PASS |
| Read instructions for the current game type | User taps on “Instructions” under any of the Games views -- i.e. HORSE | App reads aloud the instructions to the user | App fails to read aloud the instructions to the user, and does nothing | PASS |
| Start a game session | User taps on “Start” under any of the Games views -- i.e. HORSE | App initializes game session and tells the user “You have started a game” | App fails to initialize the session, and/or does not tell the user that a game has started | PASS |
| Get name of the current users turn in the game of HORSE | User taps on “Who’s turn?” | App reads aloud to the user the name of the user whose turn it is | App fails to retrieve the current users turn, and/or fails to read the name aloud to the user | PASS |
| Get current user's score in the game of HORSE | User taps on “My current score” | App reads aloud the current user's score | App fails to retrieve the current user’s score, and/or fails to read the score aloud to the user | PASS |

### 2.a.2 Integration Testing *(Kendall)*

In addition to unit testing from the test cases above, the team also conducted full integration testing for both the iOS and Android applications. Example integration testing steps are described in the list below:

1. Test current session statistics across different views
   1. Log in
   2. Tap on “Statistics”
   3. Tap “Simulate made three-pointer”
   4. Tap “Simulate missed shot”
   5. Tap on “Shooting percentage” under “Current Session”
      1. The app should read aloud “Your shooting percentage for the current session is 50 percent”
   6. Tap on “Home”
   7. Tap on “Games”
   8. Tap on “HORSE”
   9. Tap “Simulate missed shot”
   10. Tap on “Home”
   11. Tap on “Statistics”
   12. Tap on “Shooting percentage” under “Current Session”
       1. The app should read aloud “Your shooting percentage for the current session is 33 percent”
2. Test current session statistics after re-logging in (note: session statistics accumulate for each game session)
   1. Log in
   2. Tap on “Statistics”
   3. Tap “Simulate made three-pointer”
   4. Tap on “Number of three pointers” under “Current Session”
      1. The app should read aloud “You have made 1 three-pointer in the current game session”
   5. Tap on “Home”
   6. Tap on “Sign Out”
   7. Tap on “Sign In” & enter credentials
   8. Tap on “Statistics”
   9. Tap on “Number of three pointers” under “Current Session”
      1. The app should read aloud “You have made 0 three-pointer in the current game session”
3. Test entire history statistics after re-logging in (note: entire history stats always accumulate)
   1. Log in
   2. Tap on “Statistics”
   3. Tap “Simulate made three-pointer”
   4. Tap on “Number of three pointers” under “Entire History”
      1. The app should read aloud “You have made a total of 1 three-pointer”
   5. Tap on “Home”
   6. Tap on “Sign out”
   7. Tap on “Sign in” & enter credentials
   8. Tap on “Statistics”
   9. Tap on “Number of three pointers” under “Entire History”
      1. The app should read aloud “You have made a total of 1 three-pointer”

### 2.a.3 Acceptance Testing *(Thomas)*

1. User logs in successfully - PASS
2. User can access Games menu - PASS
3. User can access Statistics menu - PASS
4. User can access Settings menu - PASS
5. User can start a game of HORSE - PASS
6. User can finish a game of HORSE - PASS
7. User can retrieve stats on a game of HORSE - PASS
8. User can start a game of ShootAround - PASS
9. User can finish a game of ShootAround - PASS
10. User can retrieve stats on a game of ShootAround - PASS
11. User can change settings - Not-Yet-Implemented
12. User can exit application - PASS

## 2.b Hardware Testing *(Byron)*

Configuration

In order to test the viability of real time communication, the Arduinos have to be connected to the RFM breakout chip. After soldering the headers, the logic converters and the RFM modules, the nodes were connected as per the wiring diagram detailed in the architecture specifications.

Programming of the Arduino was done in the Arduino IDE. The language that the Arduino is configured with is C like, and is compiled and flashed onto the Arduino at the time of build. For this RFM breakout chip, we used a series of open libraries, RFM64, created by low powerlabs. These libraries take the thumb2 instructions and formats them to a higher level object oriented model. Libraries must be installed directly to the Arduino IDE for them to be referenced. Complete code attached.

### 2.b.1 Test Plan

As per specifications, the following list of critical points were tested to ensure the integrity of the signal strength between nodes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case description | Input(s) | Pass condition(s) | Fail condition(s) | Status |
| Test modes power | Arduino Uno,  DC/LiPo/USB | Node able to operate without being attached to PC | Node does not operate without being tethered | PASS |
| Test node Configuration | Arduino Uno, RFM64HCW | Able to address and read from the devices on a single node. | Node is unable to configured | PASS |
| Test player communication | Arduino Uno, X2  RFM64HCW X2 | Receives packets from different nodes and stores signal strength information | Two nodes are unable to communicate | PASS |
| Test Beacon node communication | Arduino Uno, X2  RFM64HCW X2 | Continuously updates its RSSI information to the player node | Unable to continuously send or otherwise communicate | PASS |
| Acceptance Test: User makes a shot wearing device | Arduino Uno, X2  RFM64HCW X2 | While wearing the device the player makes a shot, and the information is recorded in the database | Any piece in the design flow failing. | NOT YET IMPLEMENTED |

DC/LiPo/USB Power Testing:

Typically, an Arduino is powered through USB or DC power adapter. However, the player will be moving around, which will require the node that the player wears to operate wirelessly. The Arduino UNO has multiple ways to address this issue.

The code that operates the UNO must still be flashed onto its internal storage initially, but after that, every time the device is powered on, the code will execute once the reset button is pushed. To test this, the following code was flashed to the Arduino:

Int LED = 13;

Blink(LED,1000);

In this case, 13 is the pin number that is associated with the onboard LED on the Arduino and Blink(<pin>,<delay>). The blink function is in the Arduino Uno library; the delay is given in milliseconds. Therefore, a delay value of 1000 will toggle the voltage of that pin from high to low every second.

After getting the code flashed to the Arduino, and confirming its success over USB, tests were then moved onto DC adapter, and LiPo battery. The results were consistent when powered by each source.

Test node configuration

After installing RFM64 libraries to the Arduino IDE, configuration values for the Arduino. Working from the low power labs, libraries for the RFM64 breakout chip, the configuration of all nodes require the following values:

* NetworkID: The network of nodes
* NodeID: The individual node ID
* ToNodeID: The node to send to
* UseAck: Send acknowledgement upon receiving a packet
* Frequency: The frequency of the RFM transceiver
* LED: The pin on the Arduino corresponding the onboard LED
* GND: The pin on the Arduino corresponding to ground
* Baudrate: The speed at which packets are sent via COM port

Once these values are set, and flashed to the Arduino, communication to radio module opens. To test this connection, temperature was read from the internal thermostat of the radio module, successfully.

Test player communication

This is a test of the RFM module’s ability to receive a packet and send an acknowledgement to multiple nodes in the network. The act of sending a packet gives an RSSI value which is a determination of the intensity of a signal. When a message is received, radio.recieveDone(), is pulled high. This allows the values for RSSI values to be updated when they receive a message.

To test this, the player node was sent packets at 915MHz and the behavior was observed. Messages were being successfully received, and their RSSI values are successfully being stored in an array data member.

Test beacon communication

This works as the other handshaking process to the player nodes .recieveDone() function. Using the native delay function in the Arduino UNO delay(n), n being the amount of time you want to delay in milliseconds, the beacon nodes are set to send its position information to the player node every second upon launch. The structure of the code is similar to that of the player node. With the exception of the addressing. All beacon nodes are set to address the player node.

To test this, after flashing the code to the node, and external power source was hooked to the player node and moved around 100 square foot area. A steady stream of data was then sent to player node, and the RSSI values reported to console.

### 2.b.2 Integration Testing

1. Test node configuration
   1. Boot Arduino IDE
   2. Connect Arduino via USB
      1. Set the baud rate
      2. Determine the COM port
   3. Upload code
   4. Reset Arduino
   5. Press button to return temperature
2. Test node to node communication
   1. Power on nodes.
   2. Press reset button on player node.
   3. Press reset button on beacon node.
   4. Begin broadcasting
   5. Wait for acknowledgement.
   6. Store RSSI information from response

### 2.b.3 Acceptance Testing

The proper plan for acceptance testing would be to let the user experience all aspects of the system working together. For example, user making a shot while wearing the player node. This functionality is not yet available so it fails.

## 2.c Quality Review for Testing *(Thomas)*

1. All test cases passed

The quality of the Android and iOS applications and hardware code are satisfactory as they complete their desired tasks and pass all required test cases, with the exception of those which have yet to be implemented. New features and functionality will be added and consequently new test cases will be added to test said features without changing the main test flow of our testing approach.

1. Code has been refactored

Both the Android and iOS applications have been refactored to ensure unnecessary overhead or issues for future releases and testing. This will ensure that unknown bugs or defects will not be exposed due to poor code management, and will help with code maintenance and reliability. This also improves code performance as un-needed code is eliminated.

1. Code has been commented

For both applications we made sure that the code was heavily, but usefully documented to ensure easy location and understanding of code functionality. The main purpose of this was for code maintenance and future feature implementation. It was also extremely helpful for the User Interface view code as that portion of code is subject to the most changes.

# 3 Metrics

## 3.a iOS Metrics *(Kendall)*

* The original number of estimated story points for developing the iOS application was roughly 50 points. The team estimated each story with story points ranging between values 1-5, where 5 is the most complex. After completing all stories, some story points were misjudged. This was to be expected since none of the team had any previous experience with iOS development.
* Total number of lines of code for the iOS application, to date, is roughly 1000 lines. XCode automatically generates code that is placed in default files. The 1000 lines metric counts lines of code that the team has actively developed.
* Weighted methods per class metric:
  1. AppDelegate: 11
  2. ViewController: 9
  3. StatisticsController: 13
  4. GamesController: 2
  5. HorseController: 9
* Depth of inheritance = 2
  1. All ViewControllers inherit from the AppDelegate
* Product size:
  1. Number of story points: 100
  2. Number of test cases: 12 + integration testing
  3. Number of lines of code: ~1000
* Defects:
  1. There is a bug in getting the current user’s information if they are automatically logged in. If the user signs in, the global variables are set that contain user information. However, if the user if automatically signed in, then those global variables aren’t set. A solution to this is to use the Firebase library call: `FIRAuth.auth()?.currentUser` The application is able to successfully authenticate a user, but the above function always returns nil.

## 3.b Android Metrics *(Thomas)*

* The original number of estimated story points for developing the Android application was roughly 50 points. The team estimated each story with story points ranging between values 1-5, where 5 is the most complex. This was to be expected since none of the team had any previous experience with Android development.
* Total number of lines of code for the Android application, to date, is roughly 2000 lines. The 2000 lines metric counts lines of code that the team has actively developed.
* Weighted methods per class metric:
  1. Dashboard: 5
  2. Games: 3
  3. Horse: 7
  4. Settings: 1
  5. ShootAround: 6
  6. SignInActivity: 6
  7. Statistics: 2
* Views (Metric - # of Widgets per view)

a. Activity\_dashboard.xml: 4

b. Activity\_games.xml: 3

c. Activity\_settings.xml: 9

d. Activity\_sign\_in.xml: 2

e. Activity\_statistics.xml: 1

f. Activity\_horse.xml: 22

g. Activity\_shootaround.xml: 22

* Depth of inheritance = 2
  1. Dashboard inherits from AppCompatActivity
  2. All other classes inherit from Activity
* Product size:
  1. Number of story points: 100
  2. Number of test cases: 12 + integration testing
  3. Number of lines of code: ~2000
* Defects:
  1. N/A - All functionality currently works as expected.

## 3.c Hardware Metrics *(Byron)*

* It was hard to build a metric to assign story points to this because of the difficulty in getting hardware components to work. Therefore the risk was divided into two separate sections, the hardware configuration portion, this includes wiring and soldering, which we gauged to be around 70 points, and the software that is used to configure the hardware. This initialization is light on code, because it only setting pin values, and storing signal information in an array as it comes in, therefore we estimated the risk would be much lower, 20s.
* Each node has it’s own code that is compiled for it. The code is almost identical between nodes for the exception of the addressing information. The code is based around the low power labs initialization and RFM64 libraries. Without the libraries, there is about 185 lines of code per node.
* Weighted by component frequency
  1. Arduino - 5
  2. RFM - 5
  3. Button - 10
  4. Battery - 1
  5. Serial COM communication - 1
* Depth of inheritance = 1
  1. All data members are inherited from RFM64 radio class object
* Product size
  1. Number of story points: 70
  2. Number of test cases: 6 + integration testing
  3. Number of lines of code: ~185 per node
* Defects
  1. Cannot reassign Arduino button from reset
  2. The maximum db intensity is -25 which doesn’t seem to change within the first 2 feet of the node.
  3. Executing code once package is received must be done by acknowledgement.

# 4 Developer’s Notes and Website *(Jared)*

Our team’s website can be viewed at <https://prodigiousmelon.github.io/>. Included on the site is each team member’s individual developer notebook, which is found under the specific team member’s name.

Word Count:

Jared - 894

Thomas - 439

Byron - 1284

Kendall - 1127