Requirements Specification Document

Team Android Optimizers
Jose Franco Baquera
Atiya Kailany
Jared Peterson

CS 448

December 7, 2019

Table of Contents

1. Introduction

The Department of Computer Science at New Mexico State University (NMSU) is currently working on a variety of research projects. One of these research projects focuses on designing and implementing a wearable interface that can control a team of virtual drones. Consequently, a mixed reality game that incorporates wearable devices and user interfaces was developed in order to test different configurations of the same system. Additional information about this project can be found in the Statement of Work document (see section 4 for a direct link).

1.1 Purpose of Product

Khalaf et al. (2018) describe in detail the purpose of the product as the following:

> The objective of the game is to find a hidden virtual object within one of the multiple structures of a physical built environment. Each structure exists physically, in whatever space the game is set, and has a virtual representation with a four-digit address. The physical structures may contain one of the following virtual elements: a clue, the hidden object, or nothing at all. The player needs to search each structure, either physically or with a drone to find out what it contains. Clues are a part of the address for the structure with the hidden object; clues may be hidden inside a structure (accessible only to the player) or may be hidden on top (accessible only to the drones). Once four unique clues are assembled, the player can use that information to identify the correct structure. This design element begins to address search and rescue methods, with the player needing to consider trade-offs between performing activities themselves or having a drone perform the activity. The hidden object functions as an analog of in-danger persons or victims, and future games may incorporate more specific features around this concept.

Ultimately, the goal of implementing this game as an Android application is to aid researchers at NMSU's Department of Computer Science by displaying an alternative map user interface using Google Maps. More specifically, the new map user interface should represent the virtual drones and other game components in a more elegant, efficient, and simple manner.

The successful completion of the project would have extensive and important real-world applications. For example, it would provide a foundation for future projects that aim to use wearable devices to control a *physical* team of drones to aid humans during both man-made and natural disasters, as well as other emergencies such as a search and rescue. Such aid to humans would not only help save lives, but also time and resources. It is important to note that such technologies are yet to be developed, meaning that researching, prototyping, and testing different configurations of user interfaces on wearable devices is still necessary until more sophisticated drones emerge on the market.

1.2 Scope of Product

For this project, we will be developing a more usable, simple, and effective map user interface.

- Map User Interface Design:

  Quick Overview – The first objective of the project is to design a new map interface using Android and Google Maps (i.e. the current interface uses NASA's WorldWind). The developed user interface will be the main interface for the game and will be displayed at all times while players operate the team of virtual drones. The map interface should, ultimately, enhance a player's experience and display relevant information about the game onto the screen.

  User Experience – The new map interface will be designed with various qualitative traits in mind. That is, the map interface should:

  - Be easy to use
  - Not have significant time lag
  - Be implemented by well-established user interface design principles
  - Display the current status of the game in an accurate manner
  - Allow users to quickly respond to changing events
  - Be simple yet effective
  - Display all relevant information without looking too cluttered
  - Allow users to find specific commands quickly
  - Have well-recognizable icons

  Information to Display – The new map interface will continuously display important information about the game, such as the following:

  - Aerial view of NMSU's campus
  - Drones' locations within the map
  - Drones' altitude (numeric)
  - Game score
  - User location
  - Human hidden objects
  - Drone hidden objects
  - Human danger zones
  - Drone danger zones
  - Drone commands such as *send*, *land*, *search*, *low altitude*, and *high altitude*

  Drone Search Enhancement – A new functionality that displays how drones search for hidden objects will be implemented. That is, this new functionality will display two user-picked coordinates on the map as an “imaginary” square area (see Figure 4).

User Stories – During the final phases of the project, user stories that highlight user experiences with the new map interface will be collected, reviewed, and used to make improvements.

- Implementation of Map Interface Using an Android Application:

  Quick Overview – The second objective of the project is to create an Android application that can display the newly designed map interface. The Android application should, ultimately, allow players to control the team of virtual drones using the touch screen capabilities of an Android tablet.

  Map Interface – The application will successfully display the new map user interface and will adhere to all the required qualitative traits/display information that was discussed in the previous bullet point (e.g. danger zones, hidden objects, etc.).

  Server / API Communication – The application will successfully connect to NMSU’s *Spitfire* server, which is responsible for providing important game information such as drones’ height and danger zones. Furthermore, the application will use Google Maps’ API to display an aerial view of NMSU’s campus.

  Drone Commands – The Android application will handle user input appropriately when issuing the following drone commands:

  - Send
  - Land
  - Search
  - Low altitude
  - High altitude

  It is worth noting that the application will have to be connected to NMSU’s *Spitfire* server in order to issue drone commands.

  Hardware – The Android application will be developed for Android tablets only. That is, the application will be tested with Android tablets and will not be implemented with smartphone usage in mind (see section 2.2 Context of Product for more information).

- Future Considerations:

  If time permits, we plan to create a unified interface that combines the map and data interfaces together. This would allow players to stay more focused on one interface instead of being distracted by having to open different tabs to check something as simple as a drone’s battery life.

- Functionalities and Changes That Lie Outside the Scope:

  There are other several functionalities and changes that lie outside the scope of work (see Figure 1 for a visual representation). Such functionalities and changes include the following:

  - Developing multiple user interfaces
  - Developing an additional tablet application
  - Developing a separate application with smartphone usage in mind
  - Implementing or modifying any algorithm that is responsible for manipulating a drone's location
  - Reconfiguring/Reprogramming NMSU's *Spitfire* or any other server that is responsible for providing important game information
  - Reconfiguring/Reprogramming the game engine that is used by *Spitfire* to simulate danger zones and hidden objects
  - Implementing more than four drones into the map interface
  - Implementing software for other wearable devices
  - Implementing other drone commands (e.g. drone speed change command)
  - All other functionalities and changes that are not mentioned explicitly in this document lie, by default, outside the scope of work

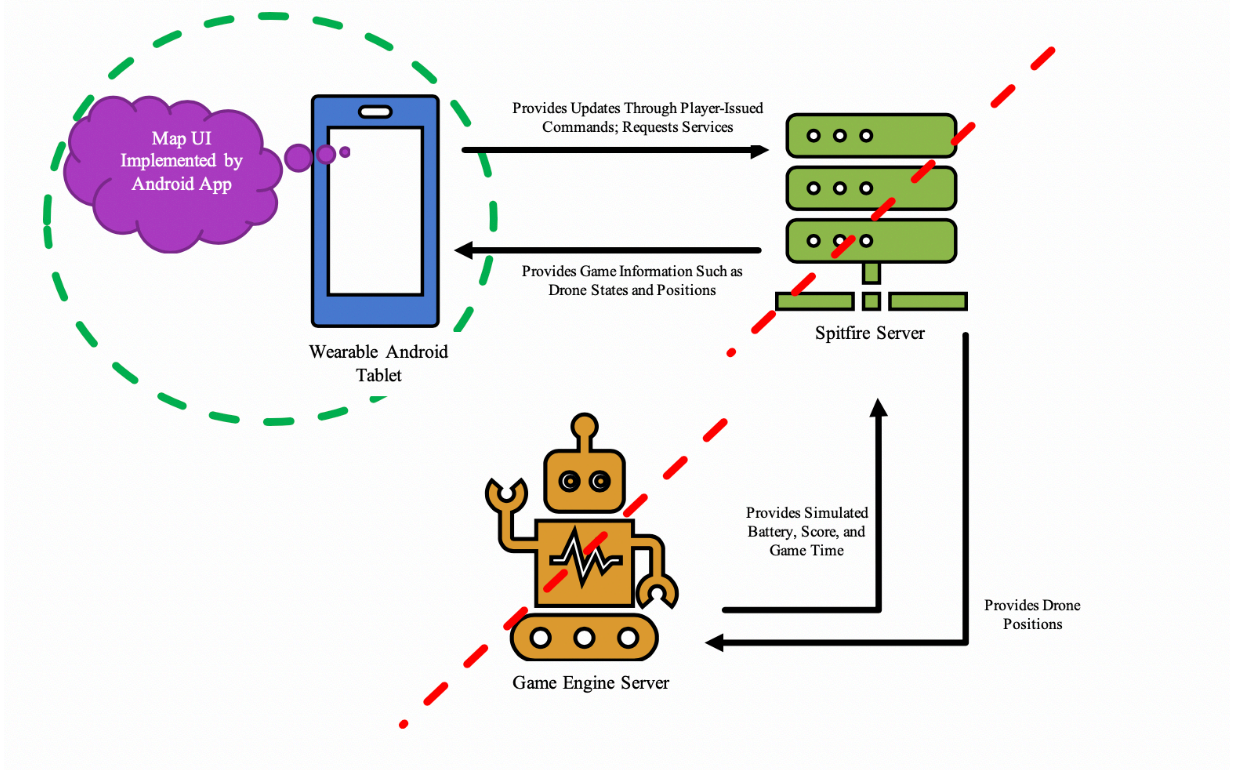

Map UI Implemented by Android App
Provides Updates Through Player-Issued Commands; Requests Services
Provides Game Information Such as Drone States and Positions
Wearable Android Tablet
Spitfire Server
Provides Simulated Battery, Score, and Game Time
Provides Drone Positions
Game Engine Server

Figure 1. Diagram illustrating the client-server and publish-subscribe architectures between the Android application and NMSU's *Spitfire* server/game engine. We note that reconfiguring or reprogramming the server and game engine is outside the scope of work (represented by a red dashed line). In contrast, everything related to the Android application is inside the scope of work (represented by a green dashed circle).

1.3 Acronyms, Abbreviations, and Definitions Used Throughout the Document

Term	Definition
API	Acronym for *Application Program Interface*; A set of routines and protocols
CPU	Acronym for *Central Processing Unit*
Danger Zone	Drone and human danger zones that can be encountered by a drone while in-flight
GUI	Acronym for *Graphical User Interface*
Hidden Object	Drone and human hidden objects that can be found by a drone while in-flight
NMSU	Acronym for *New Mexico State University*
One-to-One Button Mapping	A button will map to only one functionality (e.g. the send button command will only provide the single functionality that it was designed to implement which, in this case, is sending a drone to a specific location)
ROS	Acronym for *Robot Operating System*; An open-source operating system for robots that provides services such as hardware abstraction and message-passing between processes
ROSBridge	Protocol used to communicate to a ROS-implemented machine
Spitfire	The NMSU server that will be responsible for providing relevant game information
UML	Acronym for *Unified Modeling Language*
Virtual Drone / Drone	A drone object on the Android application interface; Does not imply an actual physical drone

Table 1. A list of the most widely used acronyms, abbreviations, and definitions used in this document.

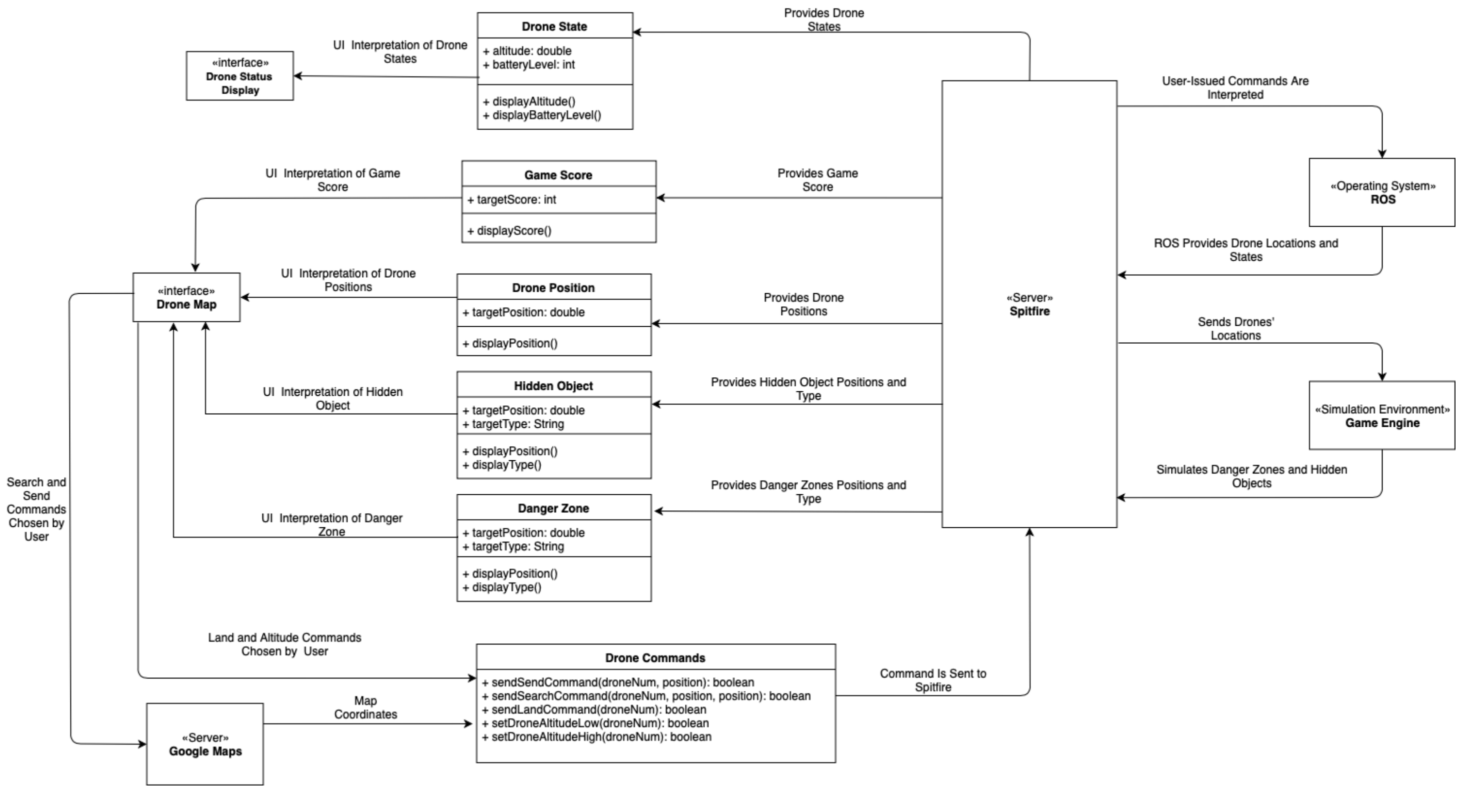
1.4 References

Khalaf, A., Pianpak, P., Alharthi, S., NaminiMianji, Z., Torres, C., Tran, S., Dolgov, I., and Toups, Z. (2018, May). "An Architecture for Simulating Drones in Mixed Reality Games to Explore Future Search and Rescue Scenarios". New Mexico State University.

2. General Description of Product

This section will provide an overview of the product and its environments, such as the domain model, context of the product, product functions, user characteristics, and assumptions/dependencies. Section 3 will delve into the details.

2.1 Domain Model



Drone State
+ altitude: double
+ batteryLevel: int
+ displayAltitude()
+ displayBatteryLevel()
«interface» Drone Status Display
UI Interpretation of Drone States
Provides Drone States
User-Issued Commands Are Interpreted
«Operating System» ROS
ROS Provides Drone Locations and States
Game Score
+ targetScore: int
+ displayScore()
UI Interpretation of Game Score
Provides Game Score
«interface» Drone Map
UI Interpretation of Drone Positions
Drone Position
+ targetPosition: double
+ displayPosition()
Provides Drone Positions
«Server» Spitfire
Sends Drones' Locations
«Simulation Environment» Game Engine
Simulates Danger Zones and Hidden Objects
Hidden Object
+ targetPosition: double
+ targetType: String
+ displayPosition()
+ displayType()
UI Interpretation of Hidden Object
Provides Hidden Object Positions and Type
Danger Zone
+ targetPosition: double
+ targetType: String
+ displayPosition()
+ displayType()
UI Interpretation of Danger Zone
Provides Danger Zones Positions and Type
Search and Send Commands Chosen by User
Land and Altitude Commands Chosen by User
Drone Commands
+ sendSendCommand(droneNum, position): boolean
+ sendSearchCommand(droneNum, position, position): boolean
+ sendLandCommand(droneNum): boolean
+ setDroneAltitudeLow(droneNum): boolean
+ setDroneAltitudeHigh(droneNum): boolean
Command Is Sent to Spitfire
«Server» Google Maps
Map Coordinates

Figure 2. UML class diagram used to capture the information structure of the problem domain in the form of entities and their corresponding attributes and relationships with one another.

The Android application will have the following two "mini" interfaces: the drone status display interface and the drone map interface. In other words, the single main user interface being displayed by the application will be divided up into two sections since this was a particular requirement that was provided to us by our clients. The drone status display interface should be able to display a particular drone's altitude and battery level through the Drone State class. The Drone State class will constantly be receiving updates from *Spitfire* on the states of each drone. We note that *Spitfire* will also be constantly communicating with the game engine and ROS. Even though interacting with these two entities is outside the scope of work, it is still important to include them in the UML class diagram.

Spitfire will be constantly updating the Drone Position, Hidden Object, Game Score, and Danger Zone classes. The drone map interface will then interpret the information in these classes to display them accordingly. Furthermore, the drone map interface will be waiting for a user to issue drone commands to manipulate a chosen virtual drone. Once the user sends a drone command, the drone map interface will communicate with the Drone Commands class, which will send the required information to *Spitfire* so that it can update the state or position of the chosen drone. It is critical to note that the *search* and *send* commands will require communication with a Google Maps server first before it can communicate with the Drone Commands class. In addition, it is important to observe that the communication between the user interface and *Spitfire* will be accomplished by a "publish-subscribe" architecture (i.e. this category of architecture is also sometimes referred to as an "event-driven" architecture) that was implemented by our clients. A perfect example of this architecture in action is the Danger Zone and Hidden Object classes subscribing to *Spitfire* for any updates that are simulated by the game engine.

2.2 Context of Product

The application will be run on the Android 4.0 (Ice Cream Sandwich) operating system within an Android tablet. Furthermore, it is worth mentioning that the application will be allowed to run on almost all Android devices, though it will be designed to be specifically run on Android tablets only. The Android application will also interact with *Spitfire* to send and receive information about what to display on the map, as well as with a Google Maps server to record which map coordinates are chosen by a user at any given time. The Android tablet that the application is running on will be worn by a research participant in an office at NMSU while the research is being conducted. If the project is further expanded and implemented to control real-life drones, then we could expect the product to be used in more extreme environments (e.g. environments where search and rescue personnel work in, such as a desert or forest). However, designing such a user interface with these extreme environments in mind is outside the scope of work.

2.3 Product Functions

See section 3.2 for a detailed list of product functionalities.

2.4 User Characteristics (Capabilities and Expectations)

The application will be designed for a typical user in the sense that anyone with previous experience with smartphones, tablets, and/or video games should be able to operate the system with ease. More specifically, a user is assumed to be competent in the following list of skills:

- Interacting with a touch screen electronic device
- Being able to connect to the internet via an Android tablet
- Using visual cues to supplement instructions

Furthermore, a user is assumed to have a basic knowledge and understanding of the following list of topics:

- Mappings of button specifiers to button functionalities
- What drones are and what they do (i.e. electronic devices that fly)
- Mappings of physical objects to virtual objects

In the future, if the application is used in a real-life and practical situation, then its usability might be limited to professional personnel that has had previous training. Any icon that appears on the map should be self-explanatory, so very little is needed from the user in that sense (e.g. a virtual drone game object will resemble a physical drone; this is why users need to know what drones are in real life and how they look). Nevertheless, the user should not need any prior knowledge of programming to use the application. We also expect users to use the Android application the way it was designed to.

2.5 Constraints

In this section, we list any external constraints that outline how we need to design the Android application.

2.5.1 Deadlines

- The Android application needs to be completed by December 6, 2019

2.5.2 Code Design

- The source code needs to be moderately commented and well documented since Ahmed Khalaf and Dr. Toups will continue to work on the system after it is delivered

- The source code needs to be as modular as possible since Ahmed Khalaf and Dr. Toups might decide to add more functionality to the system in the future

2.5.3 *Spitfire* NMSU Server

- The Android application will need to get all of the game's information from *Spitfire* (i.e. there exists a strong connection between *Spitfire* and the application)

2.5.4 Google Maps Map

- The user interface must display an aerial view of NMSU's campus using Google Maps

2.5.5 Hardware / Operating System

- The application needs to run on most Android tablets; Performance will vary, however, and will depend on which tablet is used (see section 2.2 Context of Product for more information).

2.6 Assumptions and Dependencies

It is assumed that the application will be run by users on Android devices only since no other OS devices are supported within the scope of this project. Performance-wise, there is no guarantee that the application will run on old devices, or devices with low CPU performance.

3. Special Requirements

This section will contain detailed and specific information on what the final Android application must do. It will cover external interface requirements, functional requirements, performance requirements, design constraints, and quality requirements.

3.1 External Interface Requirements

This section will describe in detail the external interface interactions that the Android application must have. More detail is given in the following subsections.

3.1.1 User Interfaces

The Android application will interact with only one external user interface.

- *Google Maps User Interface* – The application will interact directly with a Google Maps user interface that will display an aerial map of NMSU's campus (see Figure 3). It is worth noting that this interface will display nothing more than a static visual representation of the campus in "map" mode only (i.e. "satellite" mode will not be supported) and will have no Google labels such as street and building names.

3.1.2 Hardware Interfaces

The Android application will interact with only one external hardware interface.

- *Android Tablet* – The key hardware component that the application must interact with is an Android tablet. In essence, user input will be detected using only the Android tablet's touch screen capabilities. A required model or brand of an Android tablet will not be specified in this document since the application will be expected to run on an Android tablet whose platform version is at least Android 4.0 (see section 3.1.3). It is estimated that such application that expects that minimum API level will run on approximately 99% of Android devices. It is critical to mention that the application will be allowed to run on almost all Android devices, but it will be designed to be specifically run on Android tablets only.

3.1.3 Software Interfaces

The Android application will directly interact with other software packages and components. The following list describes such interfaces in more detail:

- *Maps SDK for Android* – The Android application will use Google Maps' API to access the Google Maps servers. The API will allow the Android application to do two basic things:
    1. Display a static aerial view of NMSU's campus by using four fixed map coordinates (see section 3.1.1)

2. Use user input from the GUI's map section to determine which specific location/map coordinates were chosen by the user

In essence, the API will take user input and return the map coordinates that were chosen (i.e. latitude and longitude). The required version of Maps SDK for the system will be the one released on February 6, 2019. More information about this API can be found in the following link: https://developers.google.com/maps/documentation/android-sdk/support

- *Android 4.0 (Ice Cream Sandwich)* – The Target Android (Platform) Version is the API level of the Android device where the application expects to run. This ensures that the application will continue to work as expected since the Android device will use this setting to enable or disable any compatibility behaviors. The application will be designed to interface with all Android devices that have an operating system that is Android 4.0 or newer. More information about the minimum Android version that the tablet must have in order to run the application can be found in the following link:
https://developer.android.com/about/versions/android-4.0.html

- *ROSBridge* 2.0 – In order to communicate with *Spitfire* to send player-issued commands and receive game information, the Android application will have to use ROSBridge 2.0, which is a package and API that provides a JSON interface to ROS. This package and API will allow communication between ROS and non-ROS programs/platforms (see section 3.1.4). More information about ROSBridge 2.0 can be found in the following link:
http://wiki.ros.org/rosbridge_suite

- *Android Studio 3.5 (Android Emulator)* – The application will be developed with Android Studio 3.5 and will use Java (not Kotlin) as the main programming language. In addition, Android Studio will not be needed by the user while he or she plays the game but will be used by the development team to create an Android Virtual Device (AVD). The AVD will then be used by the Android Emulator to simulate Android devices on the computer in order to test the application on a variety of devices and Android API levels.

3.1.4 Communication Interfaces

The Android application will be handling communication, both directly and indirectly, with other entities (see Figure 5 for an abstract visual representation). The following list describes these entities and any protocol requirements in more detail:

- *Spitfire (Directly)* – The Android application will communicate directly with NMSU's *Spitfire* server, which is connected to the CS network (i.e. *spitfire.cs.nmsu.edu* through socket number 9090). In order to communicate with the server, the Android application will have to use the ROSBridge 2.0 protocol, which provides several operations such as message compression/transformation and authentication. The connection will remain active until the user exits the application. More information about ROSBridge's 2.0 protocol can be found in the following link: *https://github.com/RobotWebTools/rosbridge_suite/blob/groovy-devel/ROSBRIDGE_PROTOCOL.md*

- *Google Maps Servers (Indirectly)* – The Android application will have a constant connection to a Google Maps server by using Maps SDK for Android (i.e. Google Maps' API for Android development). The protocol requirements are outlined in the official Google documentation found in the following link: *https://developers.google.com/maps/documentation/ios-sdk/reference/*

- *Game Engine Server (Indirectly)* – The Android application will indirectly communicate with a game engine server that is responsible for providing only a small part of the game's logic. This communication will not be explicit since *Spitfire* will serve as an intermediary between the Android application and the game engine server (see Figure 1). Because this has more to do with *Spitfire's* configuration, and configuring servers is outside the scope of work, no protocols involving the game engine server will be described in this document.

3.2 Functional Requirements

The user will interact with the game through one main GUI that will be displayed at all times during gameplay on a tablet via an Android application. The following list of functional requirements will describe specific behaviors that the system will perform as part of the entire problem solution.

3.2.1 Provide a Google Maps Overlay

About 80% to 85% of the user interface will display a Google Maps overlay of NMSU's campus. It is important to note that not all of the campus will be displayed (see Figure 3). The Google Maps overlay will be displayed at all times and will allow users to pick a specific location on campus to send a drone.

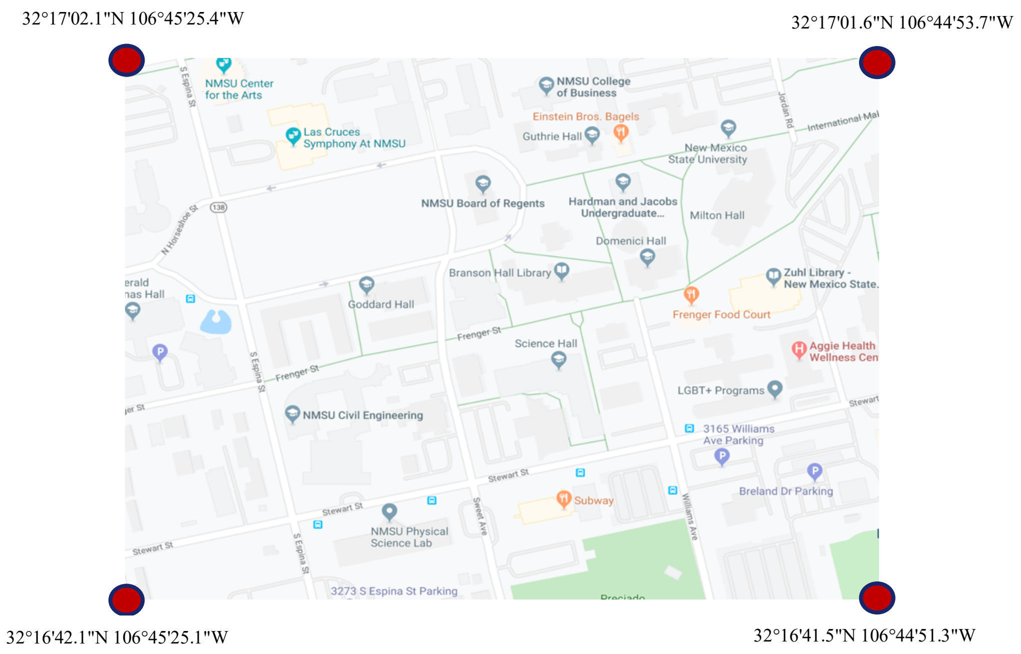


Figure 3. A visual representation of what the Google Maps section of the user interface will display. We note that the screenshot and coordinates are approximations and may change slightly. In addition, the final Google Maps overlay will have no Google labels such as street and building names.
Source: Google (n.d.). [Aerial view of NMSU’s campus using Google Maps]. Retrieved November 1, 2019.

3.2.2 Provide Building and Street Overlays

Layers of green geometric shapes and pink lines will be placed on top of the Google Maps map in order to make buildings and streets more noticeable against the busy background. These geometric shapes and lines will align with the buildings and streets found on the Google Maps map with at least 85% accuracy. We note that more priority will be placed on the green geometric shapes that outline NMSU buildings than the pink lines.

3.2.3 Drone Selection

Four distinct options that map one-to-one to four unique virtual drones will be displayed on the user interface. This will allow a user to choose which drone to manipulate. For example, if a user chooses drone “1,” then every manipulation that is chosen right after will be applied to drone “1” only. We note that this functionality will be implemented by a bootstrap carousel. Furthermore, a user will be allowed to manipulate only one drone at a time.

3.2.4 Drone Commands

Three distinct buttons that map one-to-one to three unique drone commands (i.e. *send*, *land*, and *search*) will be displayed on the user interface at all times. This will allow a user to manipulate one drone (after being chosen) at a time in one of the following ways:

1. Send a drone to a user-specified location within the Google Maps map
2. Land a drone that is in flight
3. Make a drone search a particular "imaginary" square area (see Figure 4)

It is worth noting that sending a drone to a location will require one map coordinate while sending a drone to search will require two map coordinates. Making a drone land will not require map coordinates since the user will only need to press the *land* button.

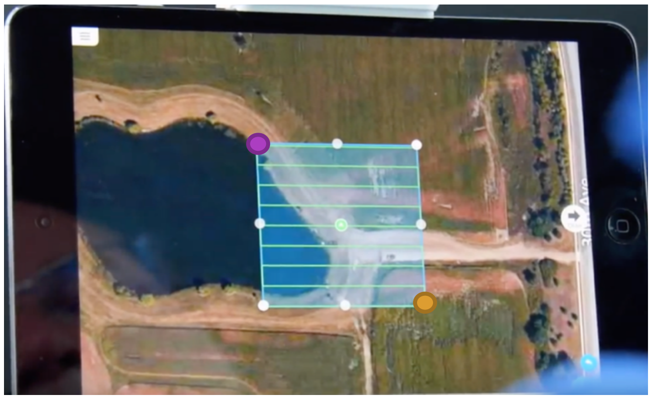


Figure 4. Screenshot demonstrating how the "imaginary" square area will be selected by using two user-chosen coordinate points. In this example, the user would have selected the purple and orange coordinates. The selected drone would then search the perimeter of this "imaginary" square.
Source: Andrews, A. [Aeroworks Productions]. (2016, February 27). DJI Phantom 3 Pro | Search and Rescue | Drone Deploy | S.W.A.R.M. [Video File]. Retrieved from https://www.youtube.com/watch?v=0l4MLWJxxy0

3.2.5 Drone Altitude

Two distinct buttons that map one-to-one to two unique drone altitude states (i.e. *low altitude* and *high altitude*) will be displayed on the user interface at all times. This will allow users to manipulate the altitude of a chosen drone currently in flight. More specifically, low altitude will correspond to 15 meters while high altitude to 25 meters.

3.2.6 Game Icons

Abstract objects that are implemented in the software will always be represented by concrete game icons on the user map interface. Such icons will include *drone hidden objects*, *human hidden objects*, *human danger zones*, *drone danger zones*, and *virtual drones*.

3.2.7 Symbols Guide

A section that will serve as a guide (or legend) for users will be displayed on the user interface. The guide will describe and label each game icon being displayed on the map (see section 3.2.6 for a list of game icons).

3.2.8 User Score

The user score will be displayed on the user interface at all times and will properly reflect the current status of the game.

3.2.9 User Location

The specific user location of where the Android application is being used will be displayed on the map interface.

3.2.10 Dynamic Information on the Map

While a user is playing the game, several icons on the map will be dynamically changing. More specifically, dynamic icons might appear, disappear, and/or change location. A dynamic icon is one of the following: *virtual drone*, *search path square area*, *human danger zone*, *drone danger zone*, *drone hidden object*, or *human hidden object*.

3.2.11 *Spitfire* Server Interaction

The Android application will connect with *Spitfire* using the ROSBridge 2.0 protocol. In essence, the application will send which virtual drone and command was selected by a user to *Spitfire*. Such interactions are summarized in the following table:

User Command	Android Application Output / *Spitfire* Input	*Spitfire* Action
Select Drone	Which virtual drone (i.e. 1 to 4) was selected.	Will apply any future drone manipulation commands to the selected drone.
Send Drone	One map coordinate.	Will modify the selected drone's position linearly.
Land Drone	Land command.	Will modify the selected drone's state.
Drone Search	Two map coordinates.	Will modify the selected drone's position nonlinearly for a short period of time.
Drone Altitude	High or low altitude.	Will modify the selected drone's altitude.

Table 2. List of user commands that will require the Android application to communicate with *Spitfire*.

Spitfire will also keep track of the locations of hidden objects and danger zones. More specifically, all of the game's information (e.g. drone positions, drone altitude, danger zones, etc.) will be stored and manipulated in *Spitfire*. Furthermore, the Android application must not only send user-issued drone commands to *Spitfire,* but also "subscribe" to any game updates that are "published" by *Spitfire*.

3.2.12 Google Maps Server Interaction

The Android application must connect to Google Maps' servers through Maps SDK for Android (i.e. Google Maps' API for Android). The application will then send map coordinate requests to the Google Maps servers any time they are needed to manipulate a chosen virtual drone. A perfect example of this interaction is when users send a drone to a location within the map:

1. A user chooses a drone, selects a location within the map, and presses the send command
2. The Android application sends a request to a Google Maps server asking which map coordinate was chosen
3. The Google Maps server returns the appropriate coordinate
4. After receiving the map coordinate, the Android application will redirect it to *Spitfire* in order for it to update the chosen drone's location

3.3 Performance Requirements

The following list of requirements will describe in detail the performance thresholds that the system must adhere to and will mainly focus on how fast the Android application processes user input. It is important to note that performance will vary heavily and will depend on the user's internet connection speed, network congestion, the type of Android tablet chosen, and how efficient the Google Maps, *Spitfire*, and game engine servers are. Because configurating servers is outside the scope of work and we cannot assume much about users' internet connection speeds/which tablets users might use, we can only make a more general list of performance requirements that focus more on ranges rather than specific values. Furthermore, it is worth mentioning that these are desirable qualities for the system but are not necessary for it to function.

- The Android application should run consistently at 30 to 40 frames per second
- The Android application should display the main GUI in 5 to 8 seconds after it is opened (i.e. this directly involves to how fast the application can start up)
- The Android application should process user input and provide appropriate feedback in 2 to 5 seconds (e.g. sending a chosen drone to a close location within the map should be completed in this time period)

3.4 Design Constraints

This section will describe in detail any constraints that will force specific design decisions. The following list describes these constraints in more detail:

- The Android application must run on any Android tablet that has Android 4.0 or newer as an operating system
- The Android application must enforce "publish-subscribe" and "client-server" architectures between it and the applicable servers (see Figure 5)
- The system must use the ROSBridge protocol to communicate with *Spitfire*
- The system must be implemented with modular software in order to allow for future modifications and changes
- All available user functionality will be accessible only through the Android application and will be directly accessed through the main user interface (i.e. the system must be coherent)
- User stories will need to be gathered and used to make any positive modifications in the system.

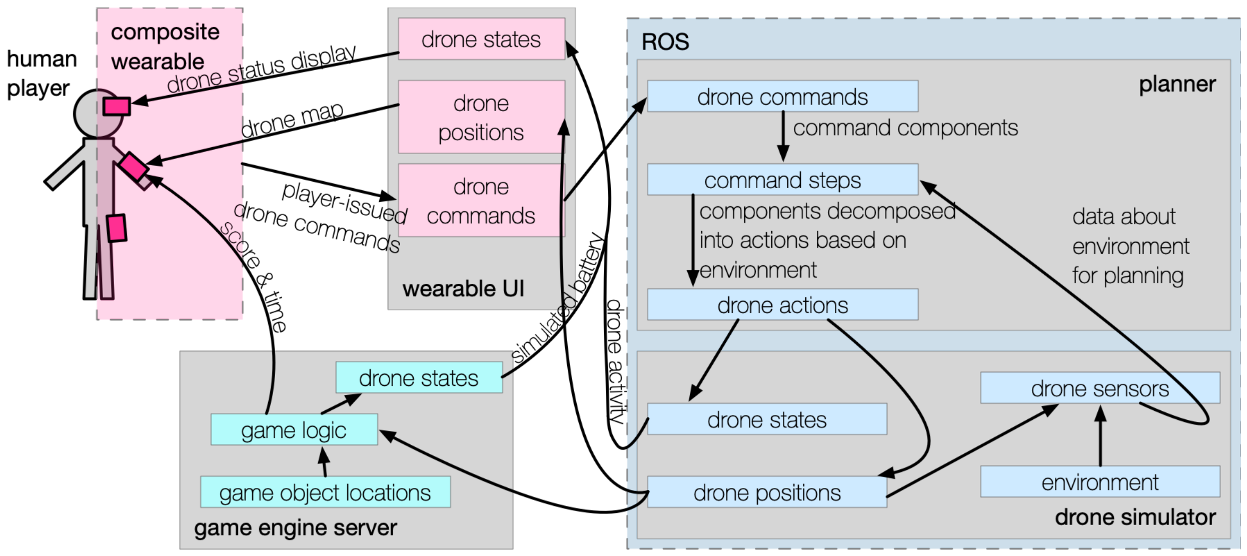


Figure 5. The original system architecture diagram that was provided by our clients. The diagram clearly emphasizes the importance of implementing an Android application that enforces "publish-subscribe" and "client-server" architectures. We note that the modifying, reprogramming, and reconfiguring of the Gameworld UI, game engine server, and drone simulator (i.e. Spitfire) is outside the scope of work.

3.5 Quality Requirements

This section describes in detail what users are going to expect, in terms of quality, while interacting with the system. For this particular project, we will focus more on the map user interface and the specific qualitative/non-functional requirements it must adhere to in order to enhance user experience. The final main user interface should:

- Be easy to use – Users should figure out how to accomplish a specific task in less than 2 seconds

- Not have significant time lag – Dynamic game icons and menus that change constantly should not lag (see section 3.3 for specific frames per second requirements)

- Be implemented by well-established user interface and game design practices (see section 4 for more information)

- Allow users to quickly respond to changing events – Users should figure out how to correct a mistake or how to interact with the system in less than 2 seconds

- Be simple yet effective – Users should not feel overwhelmed by system functionalities and the system shall not have unnecessary functionalities that confuse the user

- Display all relevant information without looking too cluttered – The user interface must have enough "white space" between buttons and other icons.

- Have well-recognizable icons – Users must be able to recognize the functionality of a button or icon in less than 1 second

- Allow users to identify buildings and streets in less than 1 second

- Warn users if connection is lost with *Spitfire* or Google Maps' servers

- Not require training or special knowledge – Users should know how to play the game after (at most) three trial runs

3.6 Other Requirements

No other requirements need to be specified at this time.

4. Appendices

This section will detail information that was not appropriate to put in-line in any other section.

- *Design Practices* – Practices on how to design effective user and game interfaces will be taken from the following book:

  Jenny Preece, Yvonne Rogers, Helen Sharp. ***Interaction Design: Beyond Human-Computer Interaction***, 4th Edition. Wiley. 2015.

- *ROS Bridge Android Library Recommendation* – A recommendation was made to choose the following ROSBridge Android library while implementing the system: *https://github.com/xbw12138/rosbridge*

- *Statement of Work* – The following link can be used to access the project's *Statement of Work* document:

  https://eltnmsu-my.sharepoint.com/:b:/g/personal/jose5913_nmsu_edu/EZ-cpMLMQypNppmWFABqVagBhKiNphEzzEbQVM9Ruo0Vcg?e=W4KQwa