AN ABSTRACT OF THE THESIS OF
Levi G. Hayden for the degrees of Honors Baccalaureate of Science in Mechanical Engineering and Honors Baccalaureate of Science in Marketing presented on August 26, 2014. Title: Development of a Mobile Application for Android Platforms.

Abstract approved: ____________________________________ Ronald Metoyer

The ability to develop mobile applications for smartphones and tablets is an increasingly valuable skill in today’s labor market. This thesis details the development of the mobile application Drone Drop from beginning to finish. I first describe the background research conducted to learn about programming languages and other tools used by developers. I explain the design process for a simple, yet challenging game. I highlight the general steps for code implementation, including class structuring and troubleshooting. I prepare Drone Drop for publication on the Google Play Marketplace by creating graphics and audio as well as by signing the development file with a secure developer key.

Upon completion of this thesis, I have successfully hosted an original application on the global Android marketplace and have received downloads from all over the world. I have tested in-app advertisement through Google AdMob and have completed a trial marketing campaign through Google AdWords.

To try Drone Drop, visit https://play.google.com/store/apps/details?id=com.fmi.game.android on an Android device or search for “Drone Drop” by Island Studios on Google Play.

Keywords: Android, Mobile, Platform, Application, Game, Development, Google, Play

Corresponding E-mail Address: Levi.G.Hayden@gmail.com
Development of a Mobile Application for Android Platforms
By
Levi G. Hayden

A PROJECT
Submitted to
Oregon State University
University Honors College

In partial fulfillment of
the requirements for the
degrees of

Honors Baccalaureate of Science in Mechanical Engineering (Honors Scholar)
and
Honors Baccalaureate of Science in Marketing (Honors Scholar)

Presented August 26, 2014
Commencement September 2014
Honors Baccalaureate of Science in Mechanical Engineering and Honors Baccalaureate of Science in Marketing project of Levi G. Hayden presented on August 26, 2014.

APPROVED:

_______________________________
Mentor, representing Computer Science

_______________________________
Committee Member representing Computer Science

_______________________________
Committee Member, representing Mechanical Engineering

_______________________________
Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

_______________________________
Levi G. Hayden, Author
Acknowledgements

I would like to thank my mentor, Dr. Ronald Metoyer, for taking a gamble on this thesis project. It is not every day that a mechanical engineering or marketing major walks into your office and asks you to mentor the development of a mobile application.

I would like to thank James Cho from Kilobolt.com for sparking my thesis idea. While I had always wanted to make mobile applications, his tutorials showed me that I could in fact build an Android application from ground zero without an immense background in computer science.

I would like to thank the BadLogicGames.com development team and community for creating such a useful Java framework as libGDX. Without their open source, cross-platform development framework, this thesis project would likely still be in development.
# Table of Contents

Acknowledgements ................................................................................................................................. i  
Preface ................................................................................................................................................. 1  
Introduction ........................................................................................................................................ 2  
  Problem Definition ................................................................................................................................. 2  
  Statement of Purpose ............................................................................................................................... 3  
Method .................................................................................................................................................... 4  
  Technical Research ................................................................................................................................. 4  
  Selection of Language .................................................................................................................................. 6  
  Selection of Libraries/API ......................................................................................................................... 7  
  Game Design ........................................................................................................................................... 8  
Implementation ..................................................................................................................................... 11  
  Class Structure ..................................................................................................................................... 11  
  Media Development ................................................................................................................................. 15  
  Debugging ............................................................................................................................................ 16  
  Publishing v1.1 ......................................................................................................................................... 17  
  Monetization .......................................................................................................................................... 18  
  Update v1.2 ............................................................................................................................................ 19  
  Test Marketing ........................................................................................................................................ 20  
Results ................................................................................................................................................... 22  
  Marketplace Statistics ............................................................................................................................... 22  
Conclusions ........................................................................................................................................... 25  
  Summary ............................................................................................................................................... 25  
  Recommendations for Future Development .......................................................................................... 27  
Bibliography ......................................................................................................................................... 29  
APPENDICES ......................................................................................................................................... 30  
  Appendix A – Class Structure Flow Diagram ......................................................................................... 31  
  Appendix B – Rendering Examples of Drone Drop Game States .......................................................... 32
Preface

With over 900 million activated devices worldwide by May 2013, the Android mobile platform presents a growing opportunity for application developers (Bort, 2013). For those who are looking to turn their ideas into a source of income, the Google Play Marketplace has enabled developers to connect directly with users to sell their own products or to generate revenue through advertisement. With minimal barriers to entry, the Android mobile platform is the best platform for new developers to begin application development.

My motivation for creating a mobile application is a deep rooted desire to understand how everything works. From a very young age, I have been interested in creating video games to entertain others. Ever since ‘smart’ devices started to grow in popularity, I had wanted to create a mobile application. In the summer of 2013, I interned at Glumac in Corvallis, Oregon. As part of my internship, I was asked to develop a mobile application for iOS to be used by the head of the company. When I began messing with application source code, I knew that I was capable of producing my own app.

When my last undergraduate year began, I made the decision to focus my honors thesis around something that I would be proud of, something I had always wanted to do, and something that would build professional skills to improve my future career.

So I made an app.
Development of a Mobile Application for Android Platforms

Introduction

With thousands of free resources on the web, mobile application development is a near limitless playground for motivated individuals. Free and Open Source Software (FOSS) communities have grown alongside industries protected by patents and trademarks in this digital age (Paula Hunter, 2013). These FOSS communities not only create software available to everyone, but they seek the improvement of software worldwide. With websites such as GitHub.com and StackOverflow.com, beginner developers have the opportunity to seek help from professional developers. Despite a technological environment that essentially begs for more developers, a stigma of exclusivity in application production persists.

Problem Definition

In this thesis project, I will develop a mobile application, beginning to end, for the Android Mobile platform. The mobile application development process will include background research into coding languages, existing applications, and marketplace restrictions. After selecting a language and various libraries to support game development, I will design a game upon which the application will be based.
When the game design has been conceptualized, application development will transition into actual implementation. In the implementation phase, the application will start to take form through a structure of code modules which make up the game’s framework. By filling in the framework and by creating media elements (audio and visuals), the application will be finished in its most basic form. After testing and debugging, the application will be packaged and hosted on the Google Play Marketplace.

In order for mobile application development to be a viable endeavor, the application must be monetized. The last few steps of the project (application development) will be the introduction of a monetizing method, updating the application, and marketing. For the purposes of this thesis, marketing will only be conducted on a testing basis to collect information for future use.

**Statement of Purpose**

The purpose of this thesis is to show the general steps that can be taken by anyone who is attempting to develop their first mobile application. With no formal training in application development, I hope to show how almost anyone can break into the mobile application industry with their own ‘at-home’ development process. I intend to show that application development is a low barrier-to-entry market and that the main limitation to success is one’s own creativity.
Method

This section details the steps I took before any actual coding took place. In order to successfully develop an application, it is important to determine which coding languages are capable of doing what you need. Because each coding language has its own benefits, the selection of a language that best fits the project’s needs can greatly reduce the labor required by the developer. Additionally, within each language, there may be libraries or frameworks that can be referenced to significantly reduce the back end work required. Lastly, by establishing a game design before code creation begins, we are able to avoid scope creep and focus on the application features that are most important.

Technical Research

The first step to develop a mobile application for the Android platform is to learn how applications are created in general. Application are developed using one or more coding languages that are often compiled or translated in an integrated development environment (known as an IDE in the programming world). This IDE is your interface into your code and helps organize the many different components that may be part of a project. As applications typically depend on more than one code language, the IDE creates the ability to seamlessly switch between languages and provides the ability to
see how information (in the form of variables) flows between different sets of code.

The IDE that is appropriate for any given project is dependent on the language(s) it uses.

The code languages that are most common for Android application development are Java, C++, and Python (StackOverflow, 2010). However, there are also application development ‘studios’ that allow the use of JavaScript, HTML5, and CSS3 to create cross-platform applications. Two of these coding platforms I considered were Titanium Studio and Apache Cordova. Both of these platforms boast that they do not require the use of native code (e.g. Java, Objective-C, C++) and can significantly reduce the development time of a mobile application. These platforms, which use web style programming converted for mobile applications, also have the benefit of being able to be used on most mobile devices (e.g. Android, iOS, BlackBerry) (The Apache Software Foundation, 2013) (Appcelerator, 2014).

Another option for Android mobile application development is to use native code. While some developers say that using native code makes app development harder than it needs to be, it also gives the developer more flexibility in his options. Even though someone who selects native code (typically Java in Android’s case, not to be confused with JavaScript) forgoes the ability to use all-in-one platforms like Titanium Studio, they get access to thousands of free, open source libraries which are essentially shortcuts.

An application programming interface (API) is a coding tool used to specify how software modules should communicate with one another. For example, if a developer wants to communicate with a database owned by Google (whether it is for their search
services or to place a Google owned advertisement within their application) they would use a Google API to be an interface between the application and the database’s software. Many mobile applications use a graphics API to communicate between the native code and a device’s graphic rendering system. Custom APIs, such as the libGDX API, which is free and open source, give developers an inside track to both 2-D and 3-D rendering.

The last segment of research revolves around game design. Everyone wants to have the next big thing (e.g. Flappy Bird) but recreating something that already has been done is a bit mundane (albeit potentially lucrative). Through research, I found that free mobile games that were simple, intuitive, but difficult, tended to have high potential for widespread use. Without a large marketing budget, in depth games that required thousands of people playing in order for it to be entertaining are not feasible. Social features, also called viral features (e.g. leaderboards, score sharing), often help a game become more popular, but are often not used to their full potential.

Selection of Language

The language selected for this project is Java with the Eclipse IDE. The reason that this language was chosen is because it is used in many fields ranging from mobile and desktop applications to corporate computing solutions (Oracle). While experience with JavaScript, HTML5, and CSS3 would be beneficial for a future in web development, an application made with the previously mentioned 3rd party ‘studios’ would be limited
in its features. Additionally, during my research, I was severely unimpressed with example mobile applications made with each ‘studio’. I believe that experience with Java will be a stronger foundation in application development than with a scripting language (e.g. JavaScript, CSS). While an application created in Java is typically not easily convertible to iOS platforms (which most commonly uses Objective-C), the next section will detail a framework that can make it much easier to have cross-platform compatibility.

**Selection of Libraries/API**

Because Java on its own isn’t easy nor is impressive, a selection of libraries and API’s is required to give an application a high level of function without ‘re-creating the wheel’. For the project, a framework created by BadLogicGames called libGDX was selected. This framework/API creates easy cross-platform capability (Android, iOS, desktop, and HTML), the ability to render 2D or 3D games by using camera views, and coding structure that efficiently manages memory (Zechner, 2013).

This project also makes use of a library called Universal Tween Engine that helps create mathematically controlled graphical fades. While a whole library for a single feature seems excessive, the ability to smoothly control a sprite’s opacity is much more difficult than it may seem.

The Google Play Services reference library was selected for this project to create an opportunity for monetization and to use features provided through the Google Play
Marketplace such as leaderboards and achievements. While these features are not included in the scope of this thesis, it is important to include so that future updates can be made.

**Game Design**

After selecting a code language (Java) and a main reference API (BadLogicGames’ libGDX), I began brainstorming on what the application should be. It was already determined that the application should be for fun, a game, and that its complexity should be appropriate for a developer’s first app. An online tutorial from Kilobolt.com provided a detailed walkthrough on how to make a ‘Flappy Bird’ clone using the libGDX API/library. However, making a clone of an already popular game that has hundreds of knockoff versions on the Android Play Store wouldn’t be rewarding enough.

My goal was to create a concept of a simple game that would require more skill than ‘Flappy Bird’ yet be almost as simple. I decided to use a two button control system and a side-scrolling scene. One control was to be a vertical control; to keep an object aloft a la ‘Flappy Bird.’ The other button needed to control a function that allowed skilled players to perform well and inexperienced players to at least perform. The second button became the ability to drop a ‘package’ from the flying object.

At this point, the game had its basic features; some flying object was able to drop some falling objects. The next step was to create an objective. By visualizing the application in its completed state, the logical objective would be to drop ‘packages’ onto
some sort of targets. The interaction for these targets was not important at this point; the flying object could be trying to bomb/destroy targets blow, deliver shipments to customers, or be an alien trying to abduct people with a ray beam.

With an objective established (making ‘packages’ collide with another object below), game over conditions needed to be made. Without some criteria that caused the game to end, the application wouldn’t be a challenge and would have no purpose. Since the player now has two types of objects to control, and a set of objects they cannot control (the targets), an interaction between the three can define success or failure. Because a player can choose when to drop packages, it made sense that a package that was wasted, or that didn’t collide with a target, would be considered a failure. With the intention of making the game difficult, this failure was determined a game over condition. Additionally, if a target ‘escaped,’ or made it off the screen without a collision interaction, the player would fail and the game would end. Naturally, because the player has complete control over their flying object, a ground collision or a collision with a target, was also determined to be a game over condition. Lastly, because the application needs to be limited to the screen area of its native device, flying an object off-screen would also be a game over condition.

To make things even more interesting, four ‘pick-ups,’ or floating objects that may or may not be acquired by the player, were added so that the player would have more to think about and couldn’t get into a safe rhythm. Because the pick-ups were added to force the player to make decisions and not to necessarily make the game more
difficult, both positive and negative effects were added. The four pick-ups are: an extra point, a penalty point, a speed boost, and instant death.

To score and give quantification to a player’s success, a point system was added used. Each successful package-target collision would increment the player’s score by one. An extra point pick-up gives the player another point while a penalty point removes a point. Each player’s high score will be saved on their device so they can show off how well they have done. Additionally, each player’s lifetime score will be logged for the purposes of seeing how much they have played and, in the future, be used to unlock additional graphic/audio sets.

The last component of game design to be discussed is theme. To this point, all components are generic objects that have some type of interaction. To an end-user, a game without a theme is boring: no one wants to have a game of squares dropping squares and avoiding squares. Because I believe drones are going to become popular in the future and there has been news about Amazon delivering packages via drone service, this applications theme became “Drone Drop,” where the user controls a drone trying to deliver packages. As mentioned previously, there is the possibility for graphic sets (e.g. aliens, planes, birds) to be available as rewards, but for the initial version, only the drone theme will be available.
Implementation

With background research completed, languages and components selected, and the game designed, the next logical step toward creating and publishing a mobile application is to code it. The Android SDK (software development kit) was installed, eclipse booted up, libraries installed, and application construction began.

Class Structure

A Java program is a collection of codes called classes that initialize objects (which are instances of these classes) which interact with each other creating an intended effect. To organize how an application’s codes interact, it is helpful to map out interactions in a class structure. The class structure for this application (Drone Drop) is shown in Appendix A. Please note that the class structure only refers to classes created by the application developer and many more classes are being referred to behind the scenes.

As shown in Appendix A, there are seven main ‘families’ of classes (organized as packages in Eclipse). The runtime operation of the mobile application starts at the AndroidLauncher.java (specified in the AndroidManifest.xml file) which is called by the mobile device. The AndroidLauncher class creates two views, one for an advertisement and one for the actual game, and tells DroneDropGame.java to start. The DroneDropGame class extends a class type called Game (this is where libraries and API’s
When a class extends another class, it inherits all of the methods, or functions of that class. When the DroneDropGame class is started by the AndroidLauncher class, a large amount of background code takes place (code created by other developers), but ultimately does whatever is written in the DroneDropGame.java file. When the game activity is created, the application tells an AssetLoader to load all of the graphics and sounds needed and triggers a screen to be displayed called SplashScreen.java. This screen uses the three TweenAccessor classes and a sprite loaded by the AssetLoader class to fade a specified image in and out (not shown). Upon the fade's (called a ‘tween’) completion, the GameScreen is loaded.

When the GameScreen (which also fades in using the three TweenAccessor classes) is loaded, the application is running to its full capability. The rest of the application is governed by a game loop as defined by a screen class (which GameScreen.java implements, meaning that it includes all of the methods of a screen and can be treated as one). The game loop in its basic form can be seen in Figure 1.
Figure 1 - Drone Drop’s basic game loop. This repeating series updates the game, plays audio, and re-renders all graphics as dictated by the game’s objects. Collisions and resulting events are checked within each update method.

As can be seen in Figure 1, most of the game is controlled by a simple loop of updating the ‘world’ (created by the GameWorld class) and telling the renderer (created by the GameRenderer class) to display the appropriate graphics. The complexity of the game loop lies in nested methods; when one object’s method causes many other objects’ methods to be called. For example, whenever the world is updated, it must tell each of its objects (two backgrounds, one drone, two grounds, three packages, three pickups, three targets, and one scrollhander) to update. Additionally, the world must check to see if any of these objects are colliding and handle the situation on a case-by-
case basis. Only when all objects are updated and collisions checked may the renderer start to render the objects to the screen. Keep in mind that this happens over 60 times per second.

When the application is showing its game screen view (the one used for gameplay), it has four different states (ready, game over, tutorial, and alive) that each change how the game is updated and rendered as shown in Appendix B. For example, when in a ready state the drone controls itself, the background and ground objects scroll, but targets do not. When in a game over state, none of the objects scroll, but a “game over” sprite is displayed.

To allow user touch inputs, the game screen makes an InputHandler object that implements InputProcessor (another libGDX class). This object sits idle while it waits for the user to interact with the mobile device. When a physical button is pressed or the screen touched, the input handler interprets the interaction and runs code appropriate for the situation. For example, when the screen is tapped, the device passes a position vector (X and Y) to the code which then is checked by the code for a certain command. If the tap falls in the lower left quadrant of the device and the game is in the “alive” state, the drone will be told to ‘fly’ (which means that its velocity is set to a certain value).

The last sets of classes to be discussed are the UIObjects and the AdComponents. The UIObjects package contains only one type of button, which provides a toggling on/off effect. These buttons are used to allow the user to turn on/off game music and sound effects. The ActionResolver class provides an interface to be passed through the
launcher (which controls that game and advertisement views) to the GameScreen class so that advertisements can be turned on and off depending on the game state. Without this class, it is difficult to pass control of the AndroidLauncher class to the rest of the application causing it to be effectively invisible and inaccessible. By using ActionResolver as an implementation class, we can pass it (and other devices’ launchers) through to the game without creating cross platform conflicts.

**Media Development**

To represent the game objects to the user in a meaningful manner, sprites and textures are used to visually display information (mainly the size and position of the drone, its packages, and the targets). Supporting textures such as the ground and the background help give the application a sense of completion and attractiveness. The last type of graphics is user interface elements such as buttons and text overlays. These elements help the user understand how to interact with the application as well as make it visually appealing.

To create the required graphics, images were adapted from free-to-use sources online and manipulated with Adobe Photoshop to create a pseudo-realistic feel. The main artistic principle used when creating the graphic set was to make sure that all of the components fit together and did not clash.

Because the game was designed around a concept that could have multiple themes, it is important to make graphics according to a standard framework. In the
libGDX library, an asset loader can load a texture from which different texture regions are then based upon. By standardizing this texture, additional graphic sets can be used by changing only which texture is loaded (each texture would have a different theme but would contain the same set of regions). While this step is not critical to the success of the application in its initial form, it will reduce a tremendous amount of work when additional graphic sets are used.

In order to deliver a completed application, a set of sound effects as well as background music must be selected. Because I have little to no experience in sound effect creation and do not have the software to properly create background music (nor the time to learn, that could be a whole new thesis project), I used outside resources to fulfill the project’s needs. For sound effects, I selected effects by online users IEDlabs and Timgormly under the Attribution license as well as a couple under a public domain license (Creative Commons). For background music, my friend Ricky Navarrete used Apple Garage Band to create a short loop to provide an exciting ambience.

**Debugging**

As with any project, there are bound be complications that arise throughout a project. With application development, problems often take form as bugs, or unintended/unwanted behaviors within a program. When an application is being published for public use, it is important that the project is completely devoid of function ruining bugs and bugs that damage the user’s experience.
While developing an application, some bugs become immediately apparent and can usually be diagnosed on the spot. For a new developer, these bugs tend to serve as a learning instrument. When the developer understands what caused a bug and implements a solution, they often learn new techniques that can be applied elsewhere in the application. Throughout the development of Drone Drop, I found it best to address a bug as soon as it arose, instead of leaving it to fix later on.

Before publishing an application to the general public, it is common to have users (besides the developer(s)) test the app and try to make bugs arise. The Google Play Marketplace actually has a feature that allows for a selected group of Android users to download an update before everyone else. However, a private debug .apk (Android’s installation package) can also be shared with friends so that an application can be tested and iteratively debugged without involving the 3rd party marketplace. While it is tempting to keep all testing in-house with the developers, outside users have a tendency to find more bugs than a developer who knows what is supposed to happen.

**Publishing v1.1**

To publish the first version of Drone Drop on the Google Play Marketplace, a developer must first create a Google Play Developer account which costs $25 for life. This account will allow a user to publish applications to the Google Play Marketplace as well as get access to Google APIs and services such as the ability to sell applications (which takes more work). Once an account is established, the project .apk file can be
exported via Eclipse IDE and signed using a password protected key (step-by-step instructions are provided in the Eclipse keystore wizard). The purpose of ‘signing’ an application is to protect it, mark that it is yours, and to keep incorrect applications from accidentally being uploaded over the wrong file.

After the project .apk is uploaded and is being processed by Google Play, the next step is to create product details (which are required by Google before your application can be published). These product details include a title, a short description, a full description, and a few graphic assets (screenshots, icons, and feature pieces). The developer must also select what type of application it is, its category, and its content rating. In Drone Drop’s case, the application is a game in the casual category and is rated for use by everyone. Lastly, the developer must provide either a website or an email address that users may use to contact the developer for app related reasons.

**Monetization**

For an application to be a viable project in the real world, it needs to bring some sort of value to the developer. No organization will commission an application that does not create a benefit of any kind. In the corporate world, an organization may pay developers to create an application that saves salespeople time in preparation or helps close sales. In the commercial world, a retailer may create an application that improves their brand image or that allows consumers to shop online. In the app development
world, organizations tend to create applications that can be sold to users or that can
generate advertisement revenue.

For Drone Drop (a free-to-play game), the monetization structure is to create
advertisement revenue. By including an advertisement view in the application
(mentioned in the Class Structure section), Drone Drop will show the user an
advertisement in the menu and game over screens. These advertisements, provided
and managed by Google AdMob, share a portion of profit with the developer. Every
time an advertisement is shown it is called an advertisement impression. When these
impressions are clicked, the organization that purchased the ad space (managed by
Google) typically pays a small amount. Whenever one of these ‘clicks’ turns into the
activity intended by the advertising organization (such as buying a product or signing up
for a newsletter), it is called a conversion. In most cases, an organization pays per
advertisement click, but in some cases pay per conversion.

Update v1.2

Once an application is successfully published on Google Play, the developer may
want to update the application to fix bugs, provide additional content, or to restructure
its monetization. With Drone Drop, version 1.1 did not include any advertisements
because priority was on having the application published. In order to generate revenue,
an update needed to be ‘pushed out.’ Thankfully, Google Play Marketplace makes this
relatively simple. The application needs to be re-packaged, re-signed, and re-uploaded
(see Publishing v1.1). Given that these steps were taken correctly, the new update can be published to the marketplace and users can begin downloading the new version. It should be noted, however, that the method upon which the application is updated on a user’s device varies on an individual basis. Depending on the Android version a person is using, the application may or may not be automatically updated. To force updates, the initial version of the application can check to make sure it is up-to-date. This method sometimes backfires as it gives the user an easy opportunity to uninstall the application completely.

**Test Marketing**

The last ‘phase’ of development taken for Drone Drop was the initial phases of application marketing. There is a saying in the marketing world, “any free publicity is good publicity.” While this is hotly debated, free marketing to friends and family is always great. With social networks in abundance, it is easy to link acquaintances to the application. Drone Drop at this time, unfortunately, lacks a ‘viral feature,’ or a feature that encourages users to share the application with people they know. Because Drone Drop lacks this feature, it has not been widely promoted beyond personal contacts (it would be inefficient and lead to poor returns on investment). Additionally, it will be much more effective to do a marketing push after the application is available on the iOS platform as reaching out to users unable to use the application would be wasteful.
To test the ‘public’ marketing environment (e.g. the advertisements that are being put on Drone Drop to generate revenue), a test marketing campaign has been started on AdMob (which is a sub-division of AdWords by Google). The test is being conducted at $.01 per click with a daily budget of $.50 to $2.50 a day (but hasn’t been enabled on a long term basis). The results of the advertising campaign test will be described in the results section.
**Results**

By going through the process outlined previously in this thesis, I, under the publishing name “Island Studios,” have produced and published a fully functioning mobile application on the Google Play Marketplace. Drone Drop, an Android game, tests the skills and coordination of the user who simultaneously controls a flying quadcopter and times payload releases. To complicate the process, the user must also be aware of four types of pick-ups that may help or harm their progress. Because the application has been uploaded to the Google Play Marketplace and has been enabled internationally, most Android users worldwide have access to Drone Drop and may download it for free at their own leisure.

**Marketplace Statistics**

An advantage of hosting the application on the Google Play Marketplace (besides the universal access it provides to users) is the toolset that Google provides to its developers. For instance, any Google developer can track the total number of downloads as well as current installations of each of their applications. Developers can easily link their application to marketing campaigns through AdMob/AdWords by Google (as I did for Drone Drop). The Google Developer Console is a single point of access to change the marketplace listing of each of the developer’s apps.
At the time of writing (August 14, 2014), Drone Drop has been installed 94 times and remains installed on 45 devices. While a rate of application retention of less than 50 percent seems low, many developer forums suggest that any retention ratio above 25 percent is actually decent. Drone Drop has an application rating score of 4.22 out of 5 from a total of 9 reviews. While most current installations (64%) are from within the United States, Drone Drop has been downloaded by users worldwide (including users in Bangladesh, Pakistan, Germany, and Jamaica, to name a few).

Drone Drop’s test marketing (through AdMob by Google) has resulted in 12 of the 94 total downloads. Drone Drop’s current advertisement campaign is set to pay $0.01 per user click. With $5.65 spent to this point, Drone Drop’s advertisement has been shown 56,849 times, clicked on 565 times, and has led to 12 downloads. While this might seem like a shockingly low conversion rate (2.12%) and an even lower click-through-rate (.99%), these numbers seem to be about average compared to what other developers have mentioned. Although the rate will fluctuate heavily because the sample size is small, presently it costs about $0.50 to convert one user through AdMob.

Users that are ‘converted’ through mobile advertisement on AdMob seem to be harsher critics of applications and are often willing to leave reviews on content. While reviews are a valuable source of feedback, it seems that these types of users often leave unsubstantiated, negative reviews. For example, one user complained about the lack of instructions despite the first thing any user is shown is a short tutorial.

The revenue earned through the display of advertisements is very low (a few cents) since update version 1.2 introduced ads by AdMob. As advertisement revenue is
directly tied to the number of clicks an ad receives, revenue can be increased by increasing the number of downloads and active users. If 100 users play every day and see 2 different advertisements, the application is only creating 200 ad impressions. If we compare this to what Drone Drop has spent on advertisements, only about 1% of impressions will lead to a click. The amount that can be earned per click depends on the advertisement (Drone Drop’s ads only cost $0.01 per click) and the cut that Google takes for providing the service.
Conclusions

I personally believe that this project was very rewarding. I enjoyed learning how to make an application idea become a reality. By using my new knowledge, I believe I have grown as a professional in this high-tech, high-speed world. The next two sections describe specific lessons learned by undertaking this project and laying groundwork for steps forward (that I am excited to take!)

Summary

One of the things I intended to ‘prove’ by completing this thesis was that anyone can be a developer as long as they have the motivation and desire to learn. The internet provides a myriad of free resources to help new, intermediate, and advanced developers; one only needs to look for them. With the rise of free and/or open source software, our ever advancing world is practically begging for new coders. By creating my own Android application, I learned more about coding and software development than I thought I would. Learning a new coding language can take a long time, but like any other language (Spanish, French, etc.) we can’t expect to become proficient overnight.

Looking at technical lessons learned throughout this project, the biggest recommendation I have to new programmers is to address errors as they arise and to actually read the error logs. I’m sure I spent a good part of a couple Friday nights
chasing down a bug, only to scroll through the error logs, track the problem to a specific line of code, and to realize I accidentally capitalized a letter incorrectly. Yes, Google is your friend and StackOverflow probably has the exact answer to the error you are experiencing, but often your solution will lie right in your IDE.

Commenting your code is a tremendously important habit to learn. While I was the only developer who worked on Drone Drop, the ability to look at a section of code and understand how it relates to the whole application is very helpful. Especially when you are learning a new language (Java in my case), comments serve as a quick translation to your native language.

The last conclusion, or lesson learned, is to have fun with it. Developing Drone Drop was not easy. Drone Drop took many hours, but when it started to take shape it all became worth it. Feel free to tweak, twist, and explore your own code; it is YOUR project after all. Take obstacles in strides: the harder the bug is to fix, the more you will probably learn in the process.

For those interested in experiencing Drone Drop first hand, the application is free to download in the Google Play Market place. Search Drone Drop in the Google Play Marketplace and look for “Drone Drop” by “Island Studios” or use the following URL to go directly to it.

Drone Drop by Island Studios
Available for free on Google Play!
Recommendations for Future Development

As the scope of this thesis was limited by time constraints, it only included the development of an application for the Android platform. Additionally it did not integrate any social (viral) features and did not use many of the services provided by Google (such as leaderboards and achievements). It does not include any extra ‘skin’ packages, or alternative graphics. This thesis does not include a widespread marketing plan. This thesis does not include additional applications that can cross-promote with Drone Drop. All of these are things that I plan to fix as time permits.

The first step after successful defense of this thesis is to integrate social features provided by Google Play Services. I believe that the ability to share high scores with friends will greatly increase its potential to become popular. With features (and prompts) to suggest that a user rates the application in the Play Store, more users will review the application which will help its placement in the search listing. Adding achievements and leaderboards will increase the depth of the application and give users a reason to play it more often.

After adding social features, I would love to start the development of alternative graphics for Drone Drop. By restricting these graphic packages to users with higher scores or more lifetime points, it will encourage players to become better at the game, increasing their own playtime and the likelihood they will share the application with their friends.

After creating additional content, I will work on developing the iOS version of Drone Drop. I believe as it is structured, Drone Drop should not be too difficult to port
to an Apple App Store supported version. In order to get Drone Drop on iOS devices, however, an Intel based system with OSX must be used. I have set a personal goal to have the iOS version of Drone Drop completed by the end of 2014.

After Drone Drop is available on iOS, I will start promoting the application on social media, at events, and through web channels (e.g. AdMob). I plan to conduct a brief business analysis to see if pursuing application development is viable in the long-term. If the spread of Drone Drop is successful, and application development proves to be a worthwhile use of my time, I will continue to develop applications on the side of full-time employment.
Bibliography


Creative Commons. (n.d.). *Attribution 3.0 Unported*. Retrieved from CreativeCommons.org: http://creativecommons.org/licenses/by/3.0/legalcode


APPENDICES
Appendix A – Class Structure Flow Diagram
### Appendix B – Rendering Examples of Drone Drop Game States

<table>
<thead>
<tr>
<th>Screen: GameScreen</th>
<th>State: Ready</th>
<th>Active Objects: Drone, Background, Ground</th>
<th>UI Buttons: Enabled</th>
<th>AdView: Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen: GameScreen</td>
<td>State: Tutorial</td>
<td>Active Objects: Drone, Background, Ground</td>
<td>UI Buttons: Enabled</td>
<td>AdView: Enabled</td>
</tr>
<tr>
<td>Screen: GameScreen</td>
<td>State: Alive</td>
<td>Active Objects: All</td>
<td>UI Buttons: Disabled</td>
<td>AdView: Disabled</td>
</tr>
<tr>
<td>Screen: GameScreen</td>
<td>State: Dead</td>
<td>Active Objects: Drone, Background, Ground</td>
<td>UI Buttons: Disabled</td>
<td>AdView: Enabled</td>
</tr>
<tr>
<td>Screen: SplashScreen</td>
<td>State: N/A</td>
<td>Active Objects: N/A</td>
<td>UI Buttons: Disabled</td>
<td>AdView: Disabled</td>
</tr>
</tbody>
</table>