Myrrrrr's Dice Game
Android Application

A Senior Project

presented to

the Faculty of the Computer Engineering

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science

By

Craig Leitterman

June 2011

Table of Contents

I. Abstract

Smartphone's are one the newest and fastest growing hardware platforms in recent years. The number of users playing and downloading games on the Android and iPhone application markets are growing rapidly. A majority of users including myself have been known to play common retro games such as solitaire as well as newer flash games like Angry Birds. These titles are usually "pick up and play" and provide quick entertainment while you're waiting for the bus or in between a work break. Given the huge popularity of these applications I decided to create a dice game that I used to play with my high school friends. The overall goal was to take part in this new platform, utilize my course programming experience, and create a fun quick game that can be played with a friend on a single phone or by oneself.

The objective of the game is simple; get the best score possible while still qualifying. To qualify a player must, after all their rolls, have at least a one "2" and one 4. Their score total is tallied by summing the remaining three dice. If the player fails to qualify, their score is zero. The first player begins by rolling five dice. After each roll they must set aside at least one or more dice towards their score. They then re-roll the remaining dice and once again set aside at least one or more dice until they have no more dice to roll. After the first player finishes, the second player follows the same rules above and the winner is the player with the highest score.

Although the game is simple, the application has numerous features. Some of the features are statistics tracking, accelerometer utilization for rolling, single and multiplayer support, sleek/simple user interface, and good look/feel.

II. Background

A. Why an Android Smartphone?

There are a number of reasons why I chose a Smartphone for a design platform as well as specifically the Android API. First off, for the game I was designing you'd normally need to carry around five dice in your pocket which isn't all that practical unless you're a table-top game professional. On the flip side, the application wouldn't work well on something that is also large like a desktop or laptop which is much larger than carrying around the required materials to play. As such the mobile Smartphone platform is ideal because of its size, intuitive touch/sensor control systems, and shear number of people that carry them. Smartphone's are also one of the hardware platforms I hadn't been able to design for in my computer courses at Cal Poly (Note: Recently they have added Android and iPhone technical electives in the past year or so). Although I wasn't creating a cutting-edge application, I was making one that would utilize all my previous coursework, critical thinking skills, and it would be created for an extremely useful and popular hardware set.

I specifically chose the Android platform because of its simplicity and low cost structure. There is no fee for development tools unlike the iPhone and there are numerous tutorials, examples, and forums for utilizing all of its powerful features and class libraries. Installing the Eclipse integrated development environment (IDE) and the android plug-in takes less than an hour and creating projects is relatively straightforward. I will admit it was a bit disorienting at first. Understanding how all the code works together in the tutorial takes time as well as becoming comfortable with it. I would have preferably wanted to take a course about developing for it to

learn some of the basics. However, once I was comfortable with the platform and figured out how to upload, test and run the application; development became a joy.

B. Proposals, Changes, and Some Development Advice

Going into any large project a designer needs to be flexible about their project and concept. I had a number of ideas and features that didn't make it from my proposal into the final project. My original development schedule ended up being quite different from what actually happened because it was my first application of this sort and the way you develop/implement features varies depending on importance, whether you can implement them simultaneously, complexity, and user testing. I ended up having to cut the 3D computer graphics interface because it would have taken too much time to interface properly as well as would require a physics engine to look/feel great. I also scrapped the messaging system for single player, i.e. send a message to your friend of what score you got. Instead I implemented an AI to play against, something I hadn't done before in my previous courses or electives. At the core I needed to interface with at least one of the onboard sensors and the typical choice for rolling was the accelerometer which I had some experience using in my capstone course. I also wanted to implement a stat tracking system and utilized my database technical elective knowledge for that task.

III. Design

A. Look and Feel

Before starting the project, my advisor, professor Lupo, pointed out that I should focus a lot on the look and feel of the application. The application market is quite fierce and if you release an unfinished and unpolished product, users are going to rate it poorly. Once this happens, it pushes your application down the ladder and away from being seen by other users. Since this is the case, it is good practice to create something that looks good, feels intuitive, and is as complete as possible.

I began by looking at the UI of other popular games that I had installed from the Android Market and thought about what made their main menu's so great. Most applications have a title at the top of the screen with textured buttons, arranged vertically in column(s) for navigating the menu. The color scheme of the screen was indicative of what you were about to experience/play, especially the font, textures, and background. The concept of feel comes into play when the user selects a button. One game had the button expand and light up while others used opposite shading too make it look as though the button was depressed. Below, in Figure 3.1 shows the main menu from Unblock Me free and Figure 3.2 shows the menu from Find It: Disney.

Figure 3.1

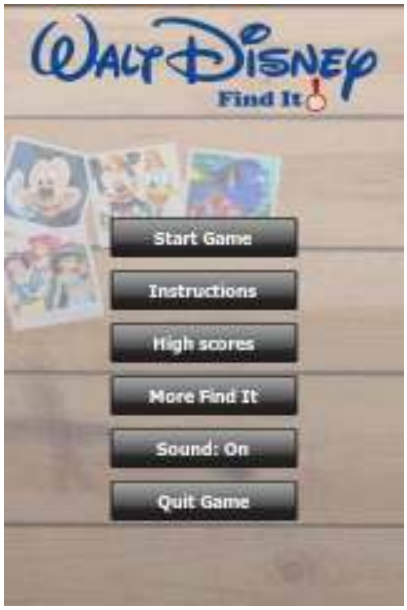

Figure 3.1

These very simple observations played an important role in designing my application main menu layout which I explain in more detail in the next section.

The look and feel also plays a part in playing the game and not just navigating the menu system. Does the user need to drag their finger? Do they just tap the screen? Can they shake the phone to roll? If so how, much force is right to trigger the event? How do they select and remove dice? There are a number of ways to implement gameplay, navigation, and features for the phone but you want all the concepts to work well together and have an overall theme that makes the application unique yet familiar and easy to use.

B. Color Scheme and Experimentation

Before I even started writing code for my application, I began by conjuring up a menu screen. Being the less than artsy individual that I am, I started my brainstorming and experimentation of color schemes in Microsoft Power Point 03'. Now most people wouldn't ever think of using Power Point for designing anything but presentations. However, I had created a number of title pages, project figures, and reports using Power Point all the way back in middle school and even up to college. I'll admit it's a bit unorthodox especially in an era of Photoshop and tons of image editing software. But I am comfortable with it, and I was able to glean some nice clip art images for the title, background, and win screen. Everything else was designed using the AutoShapes and WordArt features. Most people were incredibly surprised that all the textures were created in PowerPoint and didn't know until I told them. Another reason PowerPoint was useful was because it allowed me to create custom images without worrying about copyright issues for utilizing ones from the Internet. I could also easily layer an object such as a button, save it, and immediately test it in my application to see if it looked good.

I had originally aimed for a green, sort of felt, concept for the dice to rest on, but I couldn't find a good texture or color green that worked well. I eventually found a really nice dice image with a sharp/clean blue background. The whole image would end up being my background screen for the application's main menu. I felt that it fit so well, that I designed the color scheme for the title and buttons around the background.

The buttons also have a different image when they are pressed and by inverting the text color and shading it a different color, I was able to display a distinct button press that fits well with the theme. Below in figure 3.3 you can see the un-pressed button on the left and the pressed image on the right. It was important for the change to be distinct so you can tell when you pressed a button even if it was just a tap. I also chose to have the button shaded differently from the un-pressed one to give the impression that the button was three dimensionally pressed down.

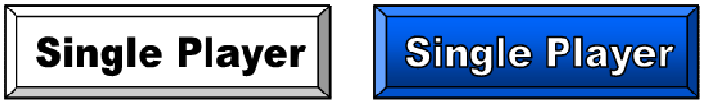

Figure 3.3

This all helps the user understand what they've pressed, make it enjoyable to press the buttons, and adds polish to the application as a whole. After putting all the elements together the final main menu can be seen in Figure 3.4.

Figure 3.4

C. XML, Navigation, and Activities

Once you have a color scheme, concept, and theme you need to write the XML code to place all your buttons, pictures, and widgets. The XML coding is essentially your front end development and allows you to place objects either linearly or relative to one another. These objects can be text, buttons, image buttons, images, input screens and more. In Figure 3.4 (above) I ended up using and ImageView and four ImageButtons. As you would guess, ImageView's are for images and ImageButtons's are textured buttons. I can then specify attributes for each object in order to place it. Below I have given an example of a button used in my main menu.

```
<Button
android:layout_width="180dp"
android:layout_height="50dp"
android:layout_centerHorizontal="true"
android:padding="20dp"
android:id="@+id/singlePlayer"
android:background="@drawable/singleplayer_button"
android:onClick="singlePlayer"/>
```

As a developer I can specify a number of things including the width, height, placement, padding between objects, id of the object, a texture, and even a method associated with the item when it has been pressed. The background attribute in this case is linked to another xml file that specifies specifically what image is shown when the button is being held down, focused, or not touched at all. This is the feature that allowed me to create buttons that effectively animate or change when pressed. The id attribute is also powerful because it allows the developer to access this UI space within the java code. Also of note is the onClick attribute which is another way of setting up a listener for the button that simply runs the method whose name is specified by the attribute. This only works if the method is "void" and has a single "view" as an input. In my main menu, the onClick runs a method that starts another

activity. Activities allow you to separate you application into modular components. For my application I have four activities total, one for the main menu, one for gameplay (both single and multiplayer), one for displaying how to play, and one for displaying game statistics. Each of these activities only controls the functions described by their name, each is only reachable from the main menu, and each returns back to the main menu (except the main menu which returns out of the application). Figure 3.5 below shows how the activities are linked together.

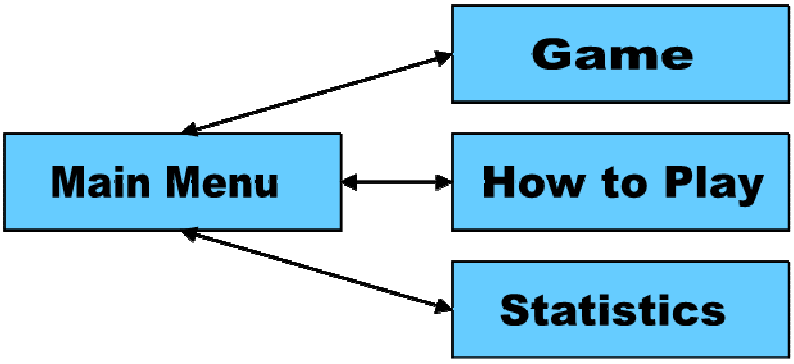

Figure 3.5

D. Game Layout and Gameplay

Selecting either single player or two players starts the same activity in the application. This activity is aware of which button was pressed from the main menu and runs a few different methods accordingly. Since there was a lot of the same variables and methods used between single player and multiplayer, I chose not to split them into separate activities. The one trick with this was creating heavily variable UI on the fly. This was easier than I thought because all I had to do was hide the objects that weren't going to be used and update images and game values accordingly.

Upon starting up the game activity the user sees five slots that hold the dice towards their score, two text views of scores, and a roll button at the bottom. The user

can also roll by shaking the phone; please see section IV part C “Accelerometer and Shake to Roll” for more detail. Figure 3.6 shows the initial screen and Figure 3.7 shows the screen after the user has chosen to roll and the animation is finished. To learn more about the animation and threading please see section IV part A “Threading and Animation.”

Figure 3.6

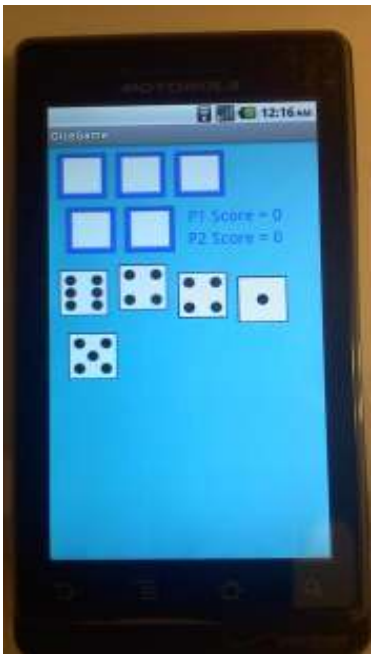

Figure 3.7

After completing a roll, the user selects the dice he wants to keep by simply taping a die. I chose to make the dice “Image Buttons” to allow quick and easy de/selection of dice. After the user selects a die the application moves it to the next open slot at the top. The image button is then removed from the remaining rolled dice. This is done in order to prevent the user from selecting it again. The player can also remove a die from a filled socket by simply tapping it. This will then return the die back to the bottom. The player can only perform this action for dice that they have

rolled on this turn. This is because once the player rolls again, the dice in the final slots at the top become locked and can no longer be removed. Once the user has selected at least one or more dice on this roll, they can continue their turn by rolling again via the roll button or shaking the phone. The roll button is only visible if you can roll. Figure 3.8 shows what happens when the user selects a die. The text in this figure also indicates the players score is DNQ ("did not qualify"). This text updates telling the user their current score based on the dice they are holding. Once the player has at least one "2" and one "4" it then begins to tally their score from the remaining currently held dice. Figure 3.9 demonstrates this transition.

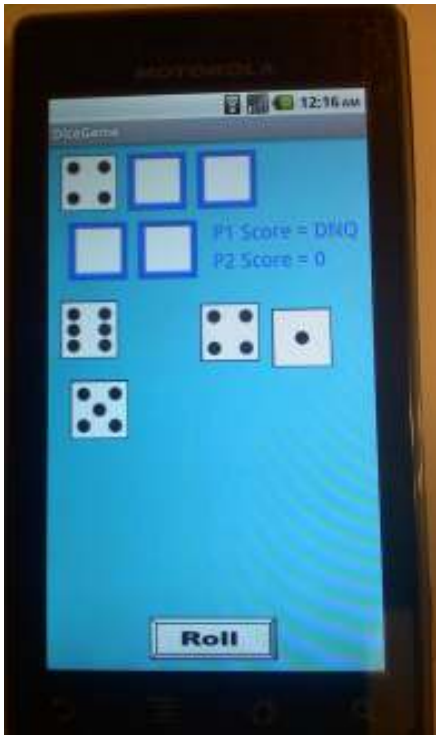

Figure 3.8

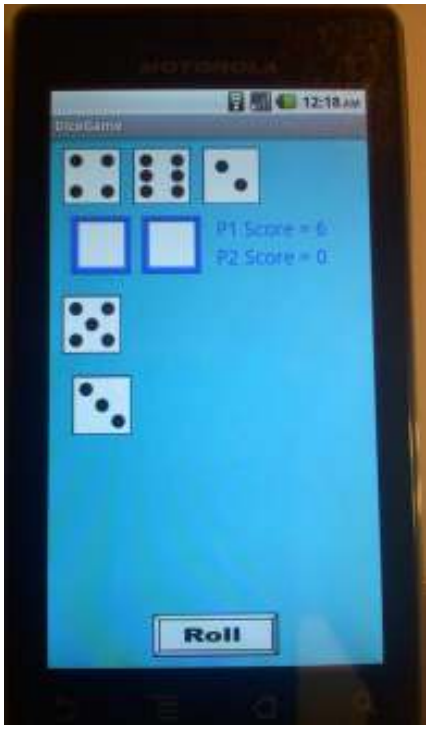

Figure 3.9

The player then continues to roll, taking at least one dice before rolling again. Once they have selected their last dice the user is then shown a finish button to complete their turn. At this point, the program will prepare for a new round and start

the AI (if single player mode) or allow player two to start clean like in Figure 3.6. For more information about the AI, see section IV part B “Artificial Intelligence.” After completing the game and comparing both scores, Figure 3.10 shows an example of one “win” screen with a button to end the game. Touching this button returns the user to the main menu.

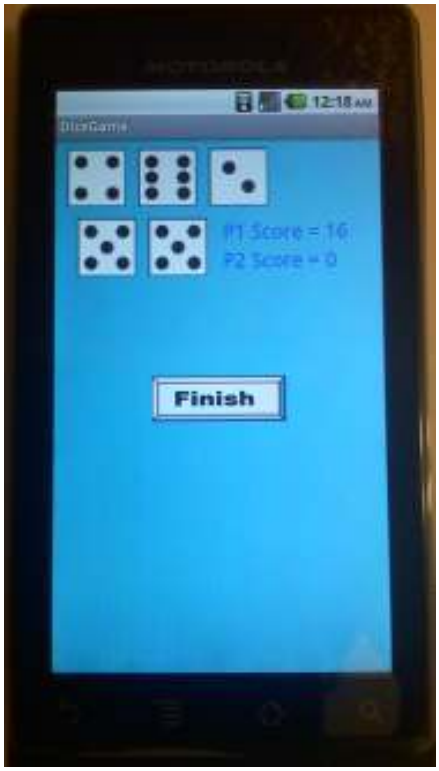

Figure 3.9

Figure 3.10

After completing a game, statistics are recorded for both player 1 and player 2’s score. It also tallies the AI’s score (if single player) as well as who won. More detail on the statistics is explained in section IV part D “Statistics: SQLite Database.” During development there was a lot of testing of edge cases to prevent the user from accidently break the application. See the testing section of this report for more information.

III. Features

A. Animation and Threading

I knew from the beginning of the project I wanted some sort of roll animation. After deciding not to go the three dimensional graphics route, I came up with two alternatives. The first was to have the dice randomly move around the screen through a number of set places and change value to give the impression they were being tossed around. The other option was to opt for a more slot machine like animation. Where the dice stand still and change value every fraction of a second. Both animations provided a sense of drama between rolls as the user would hope the roll animation would stop on the dice they want. In the end the slot machine option was chosen due to time constraints.

To get the animation to work was quite interesting. At first I created a regular method to perform this animation. However, this didn't work because the UI is running in a separate thread from the activity calling the method. What ends up happening is that the program only updates the UI only after the method is complete. To work around this I created an Asynchronous task that performs the roll animation and publishes its progress every $1/5^{th}$ of a second. This thread is started when the user rolls either via the accelerometer or button. As it updates the UI, the regular roll method is creating the actual/final values that are used by the player after the animation. Checks are also set in place to prevent the user from selecting or removing dice during this animation.

B. Artificial Intelligence

When the project was first proposed this was not one of the features. Originally I was going to have the application setup a messaging system to allow you to text your friend your score. However I found this a bit lame and of course your friend could totally make up a score and such. As I thought more about it, the AI seemed like a fun idea. I hadn't done any AI programming yet (i.e. hadn't taken the Cal Poly course), so I thought I might learn here. It was a good first start because the game isn't overly complex. In the brainstorming phase I realized you can play the game conservatively or aggressively. I started implementation on the conservative option and it turned out to win quite a lot of times which is why I ended up with a single AI option.

When it was the AI's turn, I wanted it to show the player exactly what moves the computer was making. This of course is a type of animation and as such the AI is actually run as a separate thread even though it appears to take its turn sequentially after the user. Each turn the AI prioritizes getting a single "2" and a single "4" first. If it has those, it takes all the 5's and 6's. Otherwise it grabs the highest single value. A lot of the play testers, including myself really enjoyed the single player experience, the feedback and speed (9/10^{ths} second) at which it animates. Moving forward I want to make the UI more robust and possibly add in the option for a more aggressive AI that takes extra 6's even when it hasn't finished qualifying yet.

C. Accelerometer and Shake to Roll

From the beginning I wanted to implement a shake to roll feature for the project. Although it's a very obvious feature to have for any sort of dice game, I felt that my experience with accelerometers from my capstone project would prove useful in integrating this feature. I was pleasantly surprised that the accelerometer was a lot easier to interface with when compared to my previous projects. The Android libraries have easy to use classes that manage its sensors including the accelerometer. There is in fact a method that checks when the accelerometer data is changed. I ended up doing a check to see if the net force in any direction exceeded three g's of acceleration. If this was the case it would initiate the roll sequence animation as well as the roll method. I had to put in flags to prevent the user from spamming the roll as well as rolling when they weren't supposed to be able to. I felt breaking the net force was the easiest way to interface with my current roll animation and methods. I settled on 3g's after testing because I didn't want the user to accidentally roll and I didn't want it to be to difficult to shake. I also decided not to enforce continual shaking in order to allow the user to see the animation and to prevent over zealous shaking that could cause the phone to fly out of their hand.

D. Statistics: SQLite Database

One of the core features outlined in the proposal was the implementation of statistics for the game. I wanted to do this for a couple of reasons. I was curious as to what the most common roll, average score, and whether or not the person who rolls first or second has an advantage over the other. When I was first implementing the statistics I fiddled around with two different solutions, one way using files and the other using a database. For a while I was struggling with both but decided on the database because it's a much better solution. I had taken the introduction to database course but I didn't understand the abstraction used to interface with the Android database. A few tutorials and examples later, I implemented a simple solution to store information in the rows of a three column table. I also initially ran into problems accessing the table because the database persists even if a new version of the program is uploaded. This is obviously done to keep data across multiple versions but gave me a few problems when trying to install on a new machine where the table hadn't been created or populated. I ended up adding a check at the beginning of the program to see if the table exists and if it doesn't, it creates it and populates the table with zeros.

V. Testing

Testing code and programs on the Android is incredible powerful and relatively easy. I was able to run both on the virtual machine as well as the phone itself. If any segmentation faults arise there is a debugger that provides a back trace built into Eclipse. I was able to test my program very often and in a “hands on” manner. This allowed me to focus a lot on placement, look and feel, and gameplay. The platform encourages a modular design full of separate activities and methods. I was glad my courses at Cal Poly stressed testing and designing this way; it really saved me a lot of time debugging my application. Working on this platform with its robust tools ensured that I was never stuck because of a bug for too long.

Armed with the Androids many tools I was able to self test the application and beta test with my friends very often. There were a lot of edge cases to catch and I thought about them in advance. On numerous occasions I tried to break the application whenever I could. I even encouraged my beta testers to try and mash all the on screen buttons and shake it to see if they could find any problems. The other great part of being able to test my application often with my classmates was because it allowed me to get immediate feedback about features. I could ask them about certain design choices and get their opinion. All the feedback was great during development because it made changes and implementing new features easier and ensured there wasn’t too much backtracking or erasing. I also wanted to make sure the product was very polished because once it’s released on the Android Market, its ratings will determine if it gets seen or downloaded.

VI. Conclusion

This project was very enjoyable and a great experience for a number of reasons. First, I set out and met my goals of becoming reoriented with Java as well as designing for a platform I hadn't yet tried. Second I was able to utilize a lot of the experience and knowledge that I had learned from my previous courses at Cal Poly. Third, the project has encouraged me to make newer better programs for this platform in the future and it gives me confidence in creating personal applications that can be used for controlling things at home.

It was also a great project of discovery and exploration. I had always wanted to know what it was like to create an Android application because Smartphone's are so dynamic and useful. I had no experience of how to design or make this type of application. As a solo project it is a great choice because there is a lot of easily accessible information about how to utilize and understand the Android. There is a strong community that helps new developers create fantastic projects.

As for future work on the project there is always room for improvement. I can always improve the AI, add more statistics, improve the roll animation, and add sounds and graphics. Like all projects and industry there is always plenty of room for improvement; with time usually being the limited factor. In closing, I encourage others to try making an application of their own and revel in the power of computers and computer programming.