# 163 Mobile Application

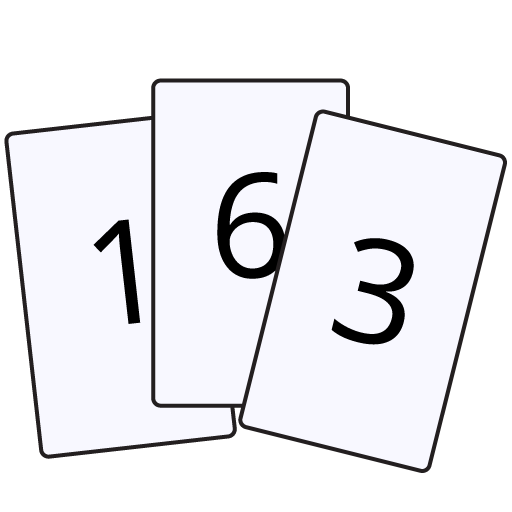
August 2016 – Present [In Progress]

## Links

Code: <https://github.com/joshuayuan/163>

Google Play: <https://play.google.com/store/apps/details?id=me.joshuayuan.a163>

## Overview

**163** is an Android Application currently available on the Google Play Store. It’s a computation based single player card game where you’re given six cards from a standard card deck, and you have to perform the four basic arithmetic operations (addition, subtraction, multiplication, division) on all six cards to reach the number 163. For example, given the set [10, 6, 10, 1, 2, 1], (10 + 6) \* 10 + 1 + 2 \* 1 = 163! There are different timed modes to play in, and for each solved set of cards, you gain a point – try to get the most points possible in the allotted time! This application is inspired by Steven Hao.

## Tl;dr

When you select the cards, which are custom button classes, they get placed or removed from a queue, which is pretty much a LinkedList. You can perform operations on the first two cards in the queue. Keep doing this and try to use all cards to get to the number 163. The cards are represented by CardNodes, which have two children CardNodes so when you combine two cards, you’re creating a new CardNode! So when you combine multiple cards together, you’re pretty much creating a binary tree structure. Let’s just play!

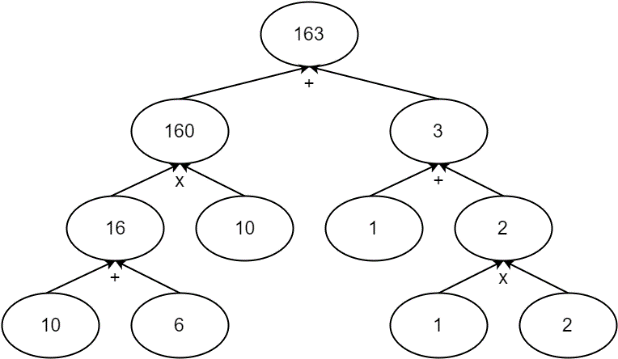
^My abridged career fair pitch to recruiters

## Development

### Start

In mid-August of 2016, I began this project as my first independent software project. I started off creating a *GameActivity* and building *activity\_game.xml*. I envisioned 6 buttons as the cards, 4 buttons for the operations, 1 for submitting, and 1 for undoing card groups. Right off the bat, I bumped into an issue I didn’t know how to solve with layouts – I wanted the third row of functional buttons to be of height wrap\_content, and the two rows of cards to equally take up the rest of the space. After more research into this, I realized that all I had to do was set the weight of the button row to equal 0, and the top two rows could have weight equal to 1 in a LinearLayout. Once the positions looked appropriate, I created a custom button appearance, *cardslot.xml*, just to make my layout look more appealing. Time to make this game work.

### Gameplay

You select at least two cards (i.e. 10 & 6), then select the operation (i.e. +). The first two selected cards then combine into one card (10 + 6 = 16). At this point, there should only be 5 clickable buttons representing the remaining 5 cards. You should still be able to select a card (16) and when possible, undo or uncombine them (16 🡪 10 & 6). When only one card remains equal to 163, then hitting the submit button would deal out six more cards.

### Flow

*MainActivity.java* is the class for the main page when the application is initially started with buttons to a tutorial/introduction page, *ReadMeActivity.java*, and the play button. The play button directs to *PlayActivity.java* which is the activity where the user selects the gameplay mode, either 1 minute, 2 minutes, 5 minutes, endless, or a custom number of minutes. After the mode is selected, a timer in *GameActivity.java* is automatically started and the game starts. Once this timer expires, the score is sent over and displayed by *ScoreScreenActivity.java*. There’s a button to return home as well.

Along the way, pressing the back button was always an issue that would mess up the flow of the game. I added key press checkers in each activity to properly redirect and finish activities when needed.

### Cards

One of the most important aspects of this project I had to design was the representation of cards in the game. Because my cards were Android Buttons, I could use onClickListeners in *GameActivity.java* to detect when the cards were selected. But I couldn’t have a direct mapping between a card and a button because this game requires constant generation and degeneration of different cards, and a button could not just be removed from the layout, when a card wasn’t present. To solve this, I created two new classes. *CardSlot.java* is a custom button class, which extends Android’s Button. There are always six CardSlots representing the six buttons, or slots for cards. Each CardSlot has an instance of a CardNode, from *CardNode.java*, where a CardNode is the representation of a card.

The CardNode class stores five important fields – two CardNode pointers which are either null if the card has no children or are pointers to the two CardNodes which created this CardNode, an int representing the CardNode’s current position in the queue to display, another int storing either the card’s original card slot or 0 if it was a combined card, and a double for the current value of the card. With the CardNode data structure, I’m able to combine and break down other CardNodes in a way similar to a binary tree structure.

At the start of the game, six CardSlots are initialized as the six buttons. Six CardNodes are generated from a deck, and placed in the CardSlots and their positions, values, etc, all assigned appropriately. When a CardSlot, the custom button class, is clicked, the queue’s position is displayed on the top left corner of the button. To implement this feature, I overrode the onDraw() method of Android Buttons to draw the appropriate graph one through six based on the CardNode’s position in the queue.

### Queue

The most challenging feature to implement was definitely the queue. The purpose of a queue is so that players could select multiple cards at once (select up to six cards) and rapidly perform operations on them (five operations) without having to reselect the cards instead of having to make three clicks (select two cards and an operation) for each computation. The former method can reduce six cards to one card in eleven taps whereas the latter implementation would result in fifteen taps, arguably slower.

The queue has complicated functions. When only one card is in the queue, any operation would do nothing, preserving the card in the queue still. When two cards are in the queue, an operation would combine the two cards into one card, emptying the queue. If more than two cards are in the queue and an operation is performed, the first two cards would combine into one card, with the new card now at the first position of the queue, and the rest of the cards would decrement towards the front of the queue. Also, if a card in the queue is deselected, it should be removed from the queue and update the position of cards in the queue after it. With all this in mind, I created my own Queue class, *Queue.java*, to represent this queue data structure.

This Queue class would be responsible for creating new CardNodes and also deconstructing two CardNodes back into the queue from a single CardNode. I used Java.util’s LinkedList to store the CardNodes and every time an update was made to the queue, I would iterate through the LinkedList and update each CardNode appropriately.

### Shuffling

I first implemented each of the six cards to just drawn as a random number from 1 to 13. Later I implemented a virtual 52 card deck and drew six cards without replacement for each set, only refreshing and reshuffling the deck every three draws (18 cards). Turns out that the second implementation results in sets which are much more difficult to solve because double digit cards become much more uncommon, and double digit cards are generally very useful for this game. I created a Deck class in *Deck.java* to abstract away some of the deck operations. The shuffling and dealing details may still be revised again in the future.

## Future Plans

Here’s a list of features or changes I’d like to make in future versions in no particular order:

* UI/UX Redesign (colors, shapes, filters, etc)
* Card to also show children cards values with their operation, instead of just its current value.
* Local high scores
* Global high scores
* Automatic solution generator
* “Three-click” function (select one card, select an operation, select second card).
* Improve state saving bugs

## Random (read at your own risk)

“Is there always a guaranteed solution?”   
“No… Usually, but probably not always.”

“So how do I know if this set of cards is unsolvable?”

“Just… try harder.”

Why 163? Steven just played it like that. It’s just a big prime number, that’s not too big.

This is the first project I’ve worked on entirely individually and I’m extremely proud of it! I have big dreams and aspirations that I hope I can all include eventually. I’m didn’t start developing this game so that it could be popular, but because this game is one I’ve been consistently playing by myself since high school (with actual cards). After taking CS61A and CS61B in the spring 2016, for the first time in forever, I finally had faith in my programming ability even having taken Java and APCS courses in high school. Even having worked on Android Applications before in teams, I honestly never really understood how it all worked together, so this application was definitely the test for myself to prove whether or not I could take Java and Android and create a functional app. Sure did!

Why did I write so much (~1600 words)? It’s winter break, and I wanted to be as transparent and as detailed about this project for myself. I understand that only my parents and maybe a few others will read even a majority of this text, but I still do it kind of for safe keeping, and kind of for the personal sense of project fulfillment.

Thank you!!

# Electric Longboard

November 2015 – Present [In Progress]

## Overview

Turns out Berkeley’s campus takes a lot more effort to walk across than I expected. While walking is a great form of exercise, I’d prefer not to show up to class all out of breath and sweaty, so I could invest in a bike, or a longboard, but instead I went overboard and am currently building a customized motorized longboard. The idea is like this: get a longboard deck with trucks, get a motor, attach the motor to the wheels with a motor mount, pulleys, and a belt, connect the motor to an electronic speed controller (ESC), which should connect to batteries and an Arduino, get a battery management system (BMS) for balancing and charging the batteries, connect a wireless controller with the Arduino, and finally attach all these parts to the board!

Here are the features:

* 192kV Brushless Outrunner Motor with 36:17 gear ratio for top speeds of 22mph
* Two 5000mAh 3S 40C LiPo batteries for a range of 6 miles
* FVT 120A ESC, and Battman BMS by Raphael Chang
* Custom machined two-piece aluminum motor mount to allow for belt tensioning
* Kama Wireless Nunchuck Controller with WiiChuck Adapter to interface with Arduino
* Vacuum formed plastic enclosure, mounted on custom machined aluminum plate
* Caliber Trucks and a 36x9.5” longboard deck

# 3D Scanner

January 2016 [Completed]

## Links

Code: <https://github.com/anuragmakineni/laser_scanner>

Devpost: <https://devpost.com/software/3d-scanner-j4fb3y>

## Overview

For PennApps XIII, one of the largest student-run hackathons, my team consisting of Raphael Chang, Nikunj Khetan, Anurag Makineni, and myself built a functional 3D Scanner. Having two out-of-date LIDARs laying around, we brainstormed for a project which could incorporate these two LIDARs and settled on the idea of a 3D Scanner. Throughout the next 36 hours at the University of Pennsylvania, we designed and assembled the frame, wrote code, played in the snow, got 10 hours of sleep, and tested our device. We were a **Top 10 Hack** and won the **Best Use of Rapid Prototyping Award**!

## Development

The hardware was designed and built from scratch, and the code was written in C++ and Python, built on top of ROS. The scanner used two LIDARs, one mounted on top and one on the side. As the turntable spun the object, the LIDAR scans were projected into a point cloud using the angle data from the servo motor. The point clouds from the two LIDARs were then merged using the ICP algorithm, which combines the common points from the two clouds into one final point cloud. The point cloud could be exported into MeshLab, which converted it into an STL file for applications like 3D printing. We used ROS to read the LIDAR data and spin the servo motor, and PCL (point cloud library) to process the point clouds. I developed software in Python and assembled the frame of the scanner.

# Autonomous Suitcase

September 2016 [Completed]

## Overview

My second experience at PennApps was PennApps XIV, where my team consisting of Nikunj Khetan, Megan Lau, Owen Li, and myself built a drivetrain for a suitcase to autonomously follow someone around. We had a single longboard truck with dual motors mounted to it, and a spare Microsoft Kinect (3D Camera) lying around so we decided to build a device that could track a person with a purpose, such as a self-following suitcase! The next 36 hours were spent working on this project under the mentorship of Rapahel Chang, Anurag Makineni, and Brent Yi, and were able to scrap together a semi-functional machine – it could track a person and the motors would move it in the general correct direction, but in the end, we ran out of time to finish testing and fine tuning the controls.

## Development

The drivetrain frame was laser cut out of quarter inch plywood with glue and tight finger joints to fix all the pieces together. Two brushless outrunners were mounted to a longboard truck for a single reduction drivetrain, which was then mounted at an angle to the frame. The two motors each had a VESC (Vedder’s Electronic Speed Controller) and was controlled via an ODROID-XU4. The code was written in Python, built on top of ROS.

The Microsoft Kinect, a 3D Camera, interfaced with ROS through a library we found online. It published a linear and an angular velocity Vector based on the point cloud it sees to our drivetrain controller which would interpret these two values and spin the right and left motors independently such that the drivetrain would move at the defined speed and direction. I worked on the code and electronics system for this project.

# Drip

October 2014 [Completed]

## Links

Devpost: <https://devpost.com/software/drip-2v5f4>

## Overview

Drip is an Android mobile application to promote charitable giving. It monitors the user’s credit card transactions and rounds up each transaction to the next dollar value and calculates the price difference. This difference, or change, is donated to charities which you’ve selected. Tony Peng, Alex Schmakov, and I made this developed this application in native Android for Cal Hacks I, and won the “Best High School Hack” Award. I worked on the graphics and front-end side of this application.

# CheckedIn

November 2015 [Completed]

## Links

Devpost: <https://devpost.com/software/checkin-97sgrn>

Code: <https://github.com/brentyi/CheckedIn>

## Overview

CheckedIn is a mobile application developed in Meteor.js with Cordova plugins for Cal Hacks II. CheckedIn is an attendance taking application, which uses Bluetooth in mobile devices to check whether or not a user is actually present. If a mobile device is present within Bluetooth distance, then that user is presumed to be present. I was on a team with Bradley Chee, Ziqi Chen, Billy Lu, and Brent Yi, and worked on the Bluetooth plugins.

# About

Hi! I’m Joshua Yuan, a second year (sophomore) studying for a B.A. in Computer Science from UC Berkeley. Welcome to my website where you can learn about my personal projects and learn more about me!

Contact me via joshuayuan@berkeley.edu if you have any questions or comments!

Check out my other profiles below.

Github: <https://github.com/joshuayuan>

LinkedIn: <https://www.linkedin.com/in/joshua-yuan>

Devpost: <https://devpost.com/joshuayuan>