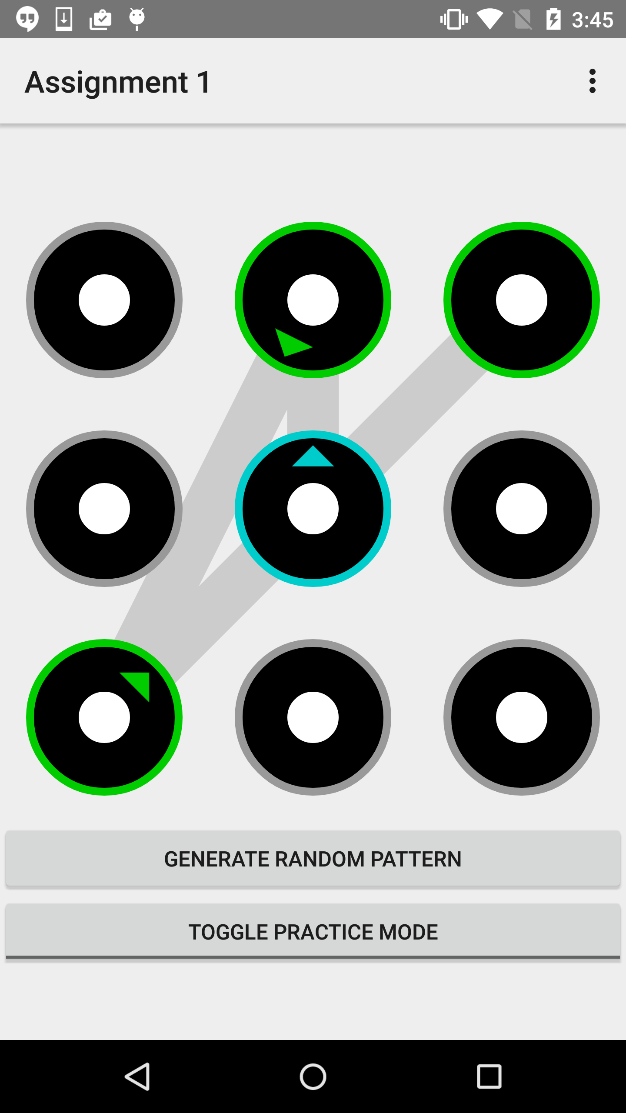
**Alan Kaan Atesoglu (aka43)**

**Harry Nelken (hrn10)**

**EECS 397 - Mobile Computing and Sensor Network**

**Assignment #1**

In this first assignment for EECS 397 we were tasked with providing the functionality for a lock screen practice activity on the Android platform. This project involved several parts, including writing data gathered from a test Android phone’s internal sensors, such as the touch, gyroscope, and accelerometer sensors, into a *.csv* file on the phone’s storage system.

**Homework Section 1**

The first section of the assignment involved generating a random pattern using the grid lock mechanism commonly found on smartphones. In doing so, there are a few rules the random pattern must comply to, the first rule being that no two grid points can be used twice in the same pattern. Another rule holds that if the nearest grid point in any direction from the last point in the pattern is unused, further grid points in that same direction are ineligible to be the next point in the pattern. To account for this, we used a random point selection process that began by trimming down a full list of the grid points so that it only contained points that were permitted by the rules. A random index is then chosen from the list and a point is added to the pattern, repeating the process until the pattern is full.

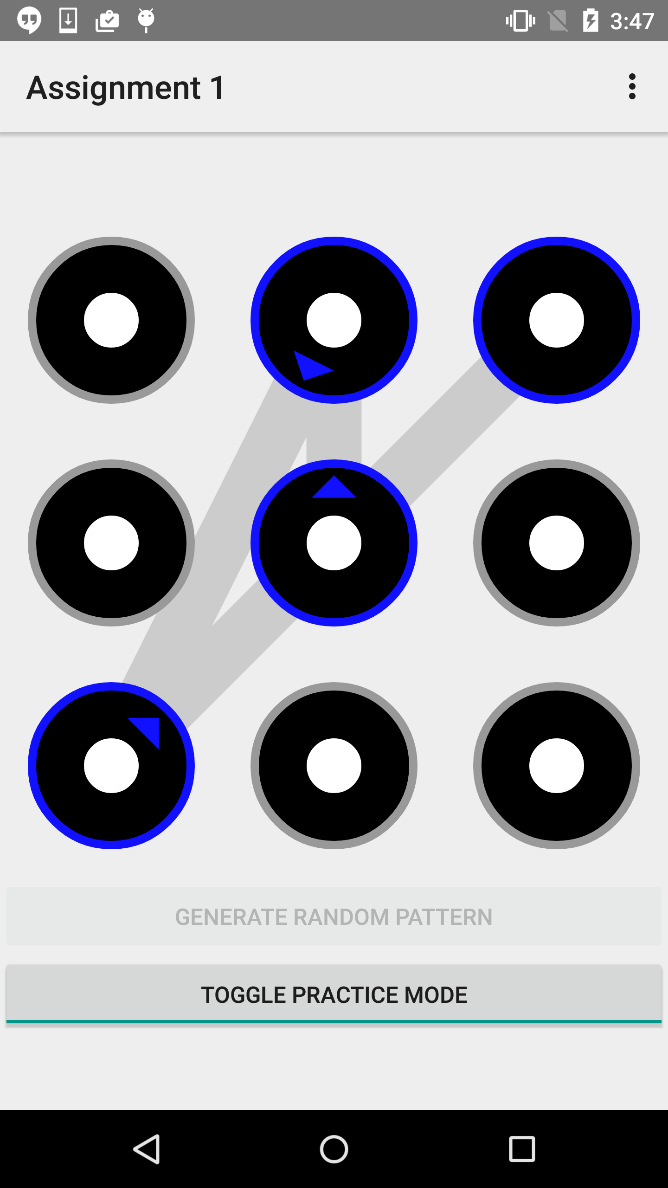
Specifically, we accomplished “two away” issue by comparing all candidate points to the last point in the pattern each time a point was added. We first subtracted the x-values of the last point in the pattern and the candidate point currently being considered. If the absolute value equaled 2, we know that there may be a point in between the two. To check the diagonal case, we check if the absolute difference between the two same points’ y-values is also 2, in which case we determine that the point “in between” is the center point. If the y-value of the candidate point is instead the same as the last point in the pattern’s y-value, the point “in between” is determined to be the point at (x - 1) of the same y, the straight line across case. If the y-values are neither equal nor 2 apart, then there is no “in between” point. If an “in between” point was determined, the pattern generated thus far is checked to see if this point is contained. If the “in between” point *is* contained, then the candidate point being considered remains eligible. This same logic, without the diagonal case check, is applied to the y-value coordinates of the last pattern point and a candidate point, covering the straight line up and down case. This process continues until the max pattern size is reached.

Figure 1. A randomly generated lock-pattern acquired by pressing the “Generate Random Pattern” button.

Figure 2. A successful attempt to re-enter the randomly generated pattern.

A small part of this section involved designing the UI to handle setting up the practice mode and generating a pattern on button clicks. To this extent, our UI uses the “setPattern” method of the “LockPatternView” class, using the result of a call to the “generatePattern” method of the “PatternGenerator” class. In addition, we disable the “Generate Random Pattern” button when the Practice mode is toggled on and enabled when toggled on, also calling the “LockPatternView” class’s “setPracticeMode” method appropriately with each toggle. The latter of these actions enables the user to practice drawing their generated pattern in real time. We also made some adjustments to the string values in the Android project to make the button text’s more appropriate.

**Homework Section 4**

This section, as well as Sections 2 and 3 before it, involved accessing the phone’s sensor data and manipulating it. By means of various event listeners, the start and end of every touch, as well as changes of the touch position in between the start and end, are recognized by the Android environment in the recurring method “onTouchEvent”, the event having several possible state values covering the beginning, middle, and end of every touch. Similarly, changes in the values of the various sensors embedded in an Android phone can also be recognized and observed through the method “onSensorChanged”. In this way, the position, velocity, pressure, and size of each touch are temporarily stored in buffers, along with readings from the phone’s accelerometer, magnetometer, rotation, linear acceleration, gyroscope, and gravity sensors. Upon the end of a touch, the resultant pattern entry was tested for validity against the last randomly generated pattern. If the practiced pattern matched the generated pattern, the values stored in the buffers were confirmed as valid, later to be written to the .csv file at the end of the practice session. Sensor data accumulated during a touch that did not result in a matching pattern was discarded.  Finally, a better appreciation of the sensors in the lifecycle of the android application was gained from needing to activate the sensors and disable them as well.  These were done at specific times to minimize stress on the battery of the phone, which the sensors draw heavily from.  The sensors were registered on the “onCreate” during the application’s creation stage, however they needed to be disabled when not in use, so within the “onPause” method, which is called if the application is navigated away from, the sensor manager is disabled and the sensors stop collecting data since they are not being used.  Upon resumption of the application, the sensors would be re-added to the sensor manager, allowing for the application to resume as expected.

The end of part 4 is what brought together the entire project.  Though not much of the code had to be modified for the “PatternView” to understand when to draw the pattern and recognize that the input pattern was correct, it was significantly easier to understand what to modify after familiarizing ourselves with the application from the prior homework sections.  This familiarity was also useful when refactoring the application to store the sensor data into buffer lists, due to the behavior of writing to the file upon toggling practice mode into “off”, it had been implemented in a very similar way.

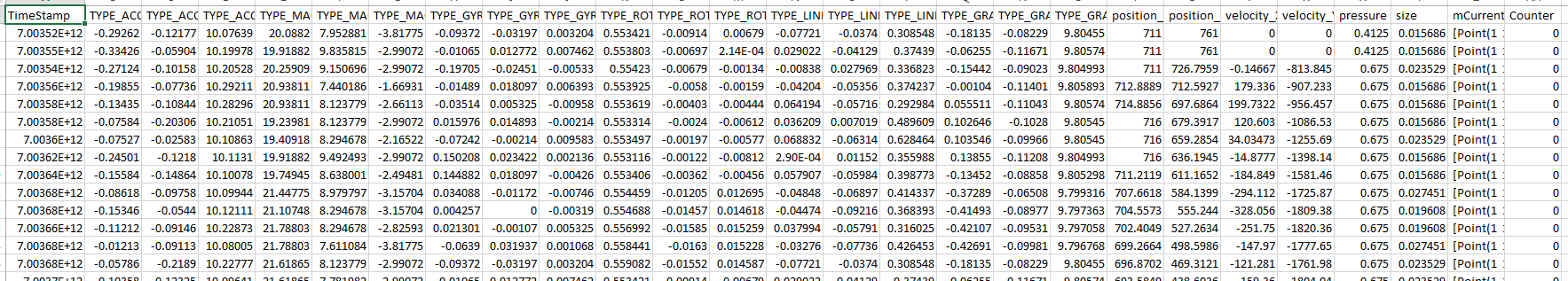
The last substantial portion of the fourth section of homework was implementing the counters and pattern recording that were also to be written to the .csv file.  This was not very challenging as their values needed to be adjusted on the specific “UP” property of a touch event.  This recorded an integer indicating the attempt at practicing the pattern, which as specified in the earlier sections was only saved if the pattern itself was correct. 

Figure 3. The outputted .csv file that is recorded when touching the screen in practice mode and saved when the user leaves practice mode. It only shows the data for successful unlock attempts.

**Division of Work**

The work was divided on a relatively even split. While we both made sure to understand the concepts at hand in regards to each of the individual homework sections, we worked on most sections entirely individually. We began the assignment by pair programming our way through the first section. The heavy logic that was the focus of that assignment was something we decided to work through together, avoiding having to explain individual solutions to one another, instead collaborating on each decision made. We then split up the work.

The second and third sections, dealing with trying out the touch and sensor event listeners, were handled mostly by Alan. He integrated the listener behavior into the existing codebase and began printing the touch and sensor information through the Android log system. Once the foundation for gathering this information was laid down, Alan also setup the writing of the stored values to a .csv file. During this time, Alan maintained possession of the Nexus 6 that we received in order to test the hardware sensors embedded in the phone, while I took the time to familiarize myself with the event cycle of the sensors, gaining further familiarity with the code in preparation for the final step.

The fourth and final section was handled by Harry. He implemented the buffer system, saving only sensor data accumulated during a successful practice touch, which required refactoring some of the code written in the second and third sections. Harry handled the maintenance of the counter state for each touch event and added the additional columns necessary to display the current pattern and the counter values in the .csv file.

Amidst the division of work, both partners were actively involved in the debugging and review of each other’s code contributions. Much effort was placed into understanding the existing codebase so as to avoid reinventing crucial components that had already been implemented, making the distribution of effort between both partners to be quite balanced.

Notes:

1. The .csv file’s save location is at the root, located along side DCIM, Download, and other user files. The true location is “/storage/emulated/0/assignment1datadump.csv”.