Project 1
Logging Data and Lock Pattern

Ryan Rose (rtr29) and Walter Huang (zxh108)

September 4th, 2015

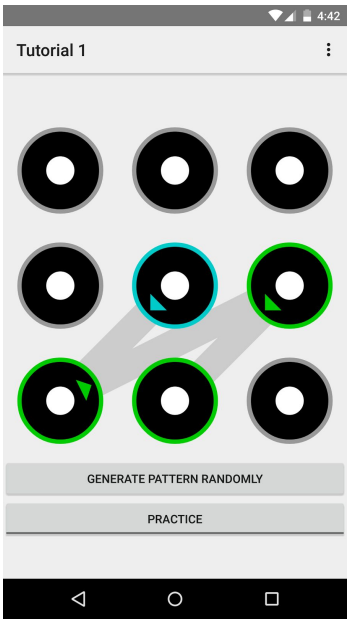
Distribution of Work:

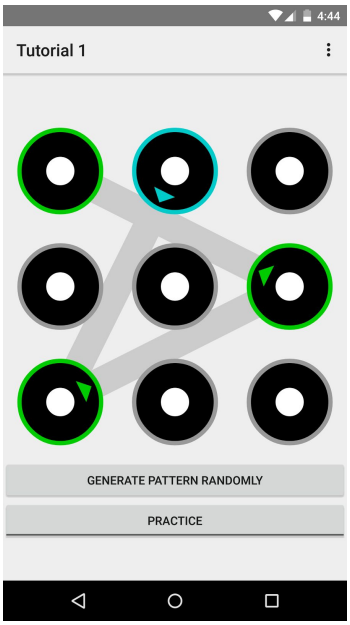
Part I: Both worked on algorithm, Walter solidifying the final algorithm and both Ryan and Walter implementing parts of it in the code.
Part II: Ryan created an initial algorithm which Walter improved and expanded, and implementation and testing was done together in pair programming.
Part III: Both studied the API on sensor function, with Ryan implementing the data collection into the algorithm Walter created in Part I.
Part IV: Walter added to the formatting done in Ryan's work of Part III, and implemented the pattern tracking and count logging. Final testing was done by both partners.

Part I

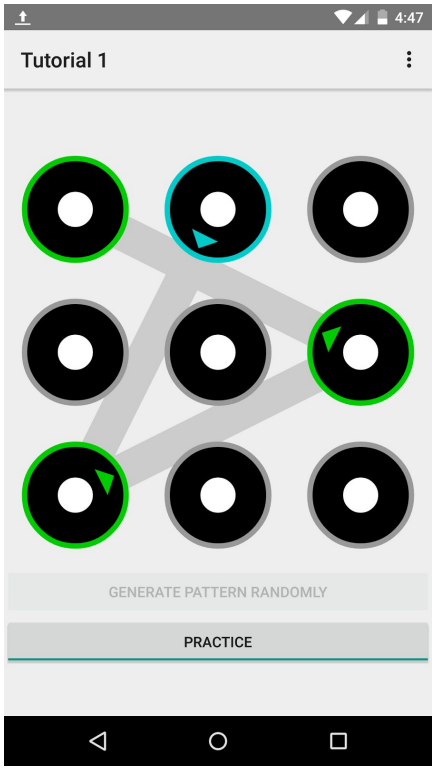
The first step of our random pattern generator is a 2D array of booleans representing the nodes chosen (true) and unchosen (false) so far in our pattern. First, a random starting node is chosen. When adding a node, we add one to a counter until reaching the pattern length that was set randomly between 3 and 5 at the beginning of the method. We then generate a random node, and pass it through a validation helper node to see if it is valid to add to the pattern. The validation checks two things: (1) that the random node does not already exist in the pattern, via the boolean array previously mentioned, and (2) that the node does not have an unused node directly between the current random node and the last node added to the pattern. We check the second step using the GCD algorithm suggested in the project guidelines. If the node passes these tests it is then added to the pattern and its corresponding location in the boolean array is set to true.

Examples:

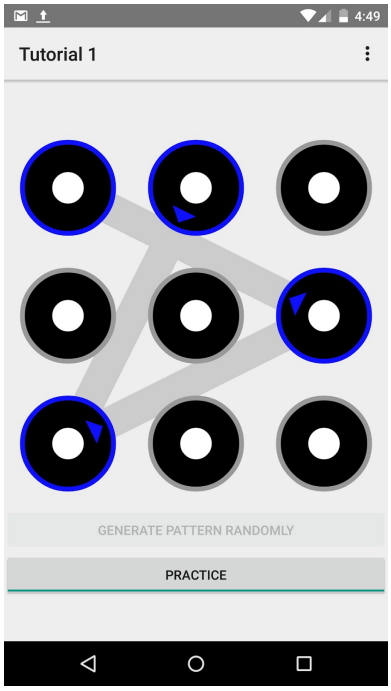




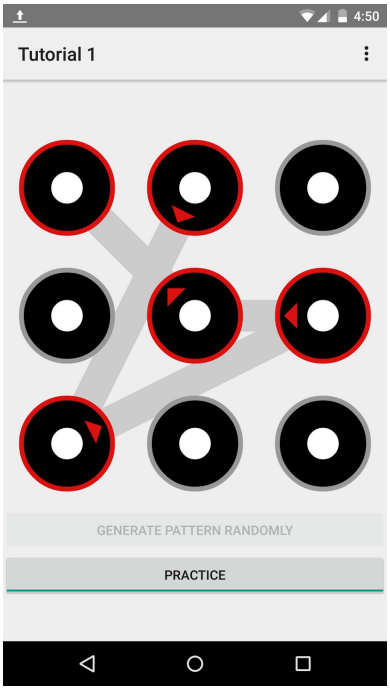
We then implemented the changes for Practice mode, including retaining the last generated pattern and disabling the “Generate Pattern Randomly” button:



Which then could be solved correctly:



Or incorrectly:



All of this interaction (clicking the buttons, drawing the patterns, feedback for correct or incorrect entries) is done on the LockPatternView. LockPatternView is a container for all the buttons and UI elements that the user interacts with. It is also the object that gets drawn

upon, and houses the methods that deal with touch interactions with the screen. An important method used frequently in Parts II-IV is onTouchEvent(). This method is called whenever someone touches the screen (ACTION_DOWN), moves across the screen (ACTION_MOVE), or lifts their finger off the screen (ACTION_UP). Any one of those three actions creates a MotionEvent which is captured and responded to by onTouchEvent.

Part II-IV:

position_X	position_Y	velocity_X	velocity_Y	pressure	size
221.0	752.0	0.0	0.0	0.3375	0.007843138
221.0	752.0	0.0012086503	0.004040606	0.6375	0.015686275
257.23547	739.07556	9.10353	-3.0179086	0.6375	0.019607844
276.48306	735.70337	12.970421	-2.096212	0.6375	0.015686275
308.71048	729.4293	21.393694	-4.0945106	0.6375	0.015686275
349.26883	722.13293	27.392069	-5.064146	0.6375	0.019607844
406.86743	712.405	35.93366	-6.300569	0.6375	0.015686275
475.61496	703.9728	43.94682	-6.6706023	0.6375	0.019607844
544.47455	698.4508	48.96317	-5.4561415	0.6375	0.019607844
619.1248	696.0	50.138954	-2.8217945	0.6375	0.015686275
699.0472	700.13086	49.101677	1.3430918	0.6375	0.015686275
782.2322	707.0244	51.01812	5.249592	0.6375	0.019607844
862.26	714.6697	51.584835	7.2351117	0.6375	0.019607844

Part II asks us to collect touchscreen tracking data, which is implemented with scalability in mind to accommodate additional sensor data. It took the longest time as we worked together. The data was captured using the built-in method onSensorChanged(), which was housed in our app's main activity. This method works in the following way: once a sensor has been registered to the application, any change in the data collected by the sensor will trigger a SensorEvent. This SensorEvent is then captured by onSensorChanged(). The SensorEvent contains information about what sensor created the event, and importantly, the data collected by the sensor. We used onSensorChanged() to capture and store data we then would write to a .csv file if the user input the correct pattern.

Part II contains a buffered writer and an array list as our "temporary storage", as each sensor update occurs, we add a new "row" to this list with all the updated data row, however, the trunk data is not written to the file with bufferedwriter until we validate it. Each row is a string with concatenated sensor input at a given timestamp. Once validation is correct, we write these strings to file, and reset the list to empty. Here we implemented some of the part 4 features so we only collect data when it is validated. The "write" action does not occur continuously, but rather executed only on validated touch_up. Keeping Part IV in mind, our bufferedwriter could be easily scaled to record data when part 4 kicks in. Part III only involved in adding additional sensor readings.

In Part IV, we directly used built in "toString()" function of the list and Point class, and the result is matching our specification. A counter is added, so that the counter is reset at each setPattern(), and the counter is incremented at each point of correct validation. The counter is

implemented without much difficulties. Instead of using raw unix milliseconds timing, we used **Timestamp** class instead. Timestamp is a built in feature that automatically format date to a nice string, so we chose to implement the timestamp attribute with this built in feature.

Here are the example of our completed Parts II-IV.

No.	TimeStamp	TYPE_ACCELEROMETER_X	TYPE_ACCELEROMETER_Y	TYPE_ACCELEROMETER_Z	TYPE_MAGNETIC_FIELD_X	TYPE_MAGNETIC_FIELD_Y	TYPE_MAGNETIC_FIELD_Z	TYPE_GYROSCOPE_X	TYPE_GYROSCOPE_Y
1	2015-09-13 23:04:11.689	0.45489502	6.22731	7.388504	-13.792419	-8.021545	-5.5480957	0.07563782	-0.0021209717
2	2015-09-13 23:04:11.75	0.26335144	6.294342	7.005432	-10.914612	-8.532715	-5.052185	0.07563782	-0.0021209717
3	2015-09-13 23:04:11.767	0.26335144	6.294342	7.005432	-10.914612	-8.532715	-5.052185	0.07563782	-0.0021209717
4	2015-09-13 23:04:11.784	0.26335144	6.294342	7.005432	-10.914612	-8.532715	-5.052185	0.07563782	-0.0021209717
5	2015-09-13 23:04:11.801	-0.37828064	6.6630554	7.8769226	-10.914612	-8.532715	-5.052185	0.07563782	-0.0021209717
6	2015-09-13 23:04:11.817	-0.37828064	6.6630554	7.8769226	-10.914612	-8.532715	-5.052185	0.07563782	-0.0021209717
7	2015-09-13 23:04:11.834	-0.37828064	6.6630554	7.8769226	-10.914612	-8.532715	-5.052185	0.07563782	-0.0021209717
8	2015-09-13 23:04:11.851	-0.3830719	6.38533	8.010986	-10.914612	-8.532715	-5.052185	0.07563782	-0.0021209717
9	2015-09-13 23:04:11.868	-0.3830719	6.38533	8.010986	-10.914612	-8.532715	-5.052185	0.07563782	-0.0021209717
10	2015-09-13 23:04:11.885	-0.3830719	6.38533	8.010986	-10.745239	-9.384155	-4.5562744	0.14274597	0.059646606
11	2015-09-13 23:04:11.903	-0.3830719	6.38533	8.010986	-10.745239	-9.384155	-4.5562744	0.14274597	0.059646606

LINEAR_ACCELERATION_Y	TYPE_LINEAR_ACCELERATION_Z	TYPE_GRAVITY_X	TYPE_GRAVITY_Y	TYPE_GRAVITY_Z	position_X	position_Y	velocity_X	velocity_Y	pressure	size	mCurrentPattern	counter
081543	0.1304779	0.36166382	6.1776276	7.6075745	805.0	1250.0	0.0	0.0	0.3625	0.011764707	[Point(1, 2), Point(0, 2), Point(1, 1), Point(2, 0)]	0
081543	0.1304779	0.36166382	6.1776276	7.6075745	778.4536	1243.8739	-4.6554656	-1.074508	0.3625	0.011764707	[Point(1, 2), Point(0, 2), Point(1, 1), Point(2, 0)]	0
80176	-0.43214417	0.071517944	6.391159	7.4375763	748.8493	1239.5968	-22.75624	-5.015224	0.3625	0.015686275	[Point(1, 2), Point(0, 2), Point(1, 1), Point(2, 0)]	0
80176	-0.43214417	0.071517944	6.391159	7.4375763	708.9171	1234.2375	-27.273445	-3.9131613	0.3625	0.015686275	[Point(1, 2), Point(0, 2), Point(1, 1), Point(2, 0)]	0
80176	-0.43214417	0.071517944	6.391159	7.4375763	665.536	1225.8804	-29.29976	-5.5651655	0.3625	0.011764707	[Point(1, 2), Point(0, 2), Point(1, 1), Point(2, 0)]	0
80176	-0.43214417	0.071517944	6.391159	7.4375763	621.31024	1218.5465	-29.661558	-5.536076	0.3625	0.015686275	[Point(1, 2), Point(0, 2), Point(1, 1), Point(2, 0)]	0
80176	-0.43214417	0.071517944	6.391159	7.4375763	581.6277	1212.3623	-28.234882	-4.975876	0.3625	0.011764707	[Point(1, 2), Point(0, 2), Point(1, 1), Point(2, 0)]	0
80176	-0.43214417	0.071517944	6.391159	7.4375763	542.49835	1208.3947	-24.534	-3.460331	0.3625	0.015686275	[Point(1, 2), Point(0, 2), Point(1, 1), Point(2, 0)]	0
80176	-0.43214417	0.071517944	6.391159	7.4375763	507.50583	1206.2816	-21.202366	-1.3186834	0.3625	0.015686275	[Point(1, 2), Point(0, 2), Point(1, 1), Point(2, 0)]	0
80176	-0.43214417	0.071517944	6.391159	7.4375763	473.51706	1206.0	-19.216131	0.25001886	0.625	0.019607844	[Point(1, 2), Point(0, 2), Point(1, 1), Point(2, 0)]	0
80176	-0.43214417	0.071517944	6.391159	7.4375763	440.93292	1209.9333	-18.177671	2.2755387	0.625	0.015686275	[Point(1, 2), Point(0, 2), Point(1, 1), Point(2, 0)]	0