Lab 1 Project Report

Matthew Swartwout James Zhang

September 23, 2015

1 Implementing Pattern Generator

1.1 Contributions

Matt Swartwout - Algorithm development, implementation, debugging
James Zhang - Algorithm development, debugging

1.2 Explanation of code and understanding

For Part 1 there were two sections of code that changed. The first is the getPattern method in the PatternGenerator class. This was a straightforward implementation of the algorithm that was described in the lab assignment. First the pattern length is determined using a random number generator between the set maximum and minimum values. After that three ArrayLists are instantiated: pattern, availablePoints, candidatePoints. pattern is the current state of the lock pattern, availablePoints is the list of all points not in the pattern, and candidatePoints is the list of points that are a valid selection for the next node in the lock pattern. A node is then randomly selected from availablePoints to be added to the pattern, and it is then removed from available points. Then the algorithm described in the lab is implemented, which removes points from the candidatePoints list if they are not a valid choice for the next node. It does this by comparing the GCD of the coordinates of the last node in the current lock pattern and the candidate node, and then seeing if there are unused nodes in between them if necessary. Once all invalid nodes have been removed from the candidate nodes list a next node is randomly selected, added to the pattern and removed from availablePoints. This repeats until a pattern of the correct length has been generated. The second part of code that was modified was the onCreate method of ALPActivity. ALPActivity is the base Activity of the app, and is what runs the LockScreenView, PatternGenerator, and all the other classes. onCreate is what is run when the Activity is created and is where the Practice and Generate Pattern buttons are created. In this case, there was no logic associated with the Buttons, so we had to add in that logic. For the Generate Pattern button, this meant added the OnClickListener, which responds when the button is clicked. When the button is clicked it calls getPattern to generate a pattern, and then calls setPattern to set the lock pattern to that

new, generated pattern. Then the LockScreenView is invalidated. Invalidating the view forces it to be redrawn, which is necessary to display the new pattern. Changing the Pattern toggle button was also likewise simple. This is done using an OnCheckedChangeListener. When the button is changed it has a boolean that indicates whether or not the button is set. setPracticeMode is called with this boolean as an argument, which then enables or disables practice mode. After this is done the view is again invalidated.

2 Capturing MotionEvent Data

2.1 Contributions

Matthew Swartwout - Implementation and debugging James Zhang - None

2.2 Explanation of code and understanding

For Part 2, the modified code mainly resides in onTouchEvent in the LockPatternView. onTouchEvent is run every time a touch event occurs. The onTouchEvent function uses a switch statement. There are three major motion events that are each cases: ACTION_DOWN, ACTION_MOVE, and ACTION_UP. If the MotionEvent is ACTION_DOWN, the user has initialized contact with the screen. Velocity in this case will be 0 for both x and y directions. The method will then write the timestamp, positions, velocities, pressure to a String using a StringBuilder. Because velocity is 0 these are hardcoded into the String, but a VelocityTracker is initialized so that if an ACTION_MOVE occurs the velocity can be recorded. If the MotionEvent is ACTION_MOVE, the user has moved their touch from one position to another. In this case the event is added to the VelocityTracker and the velocity is computed. A StringBuilder is once again used to create a comma-separated String of all the data. If the MotionEvent is ACTION_UP, the user has ended their contact with the screen. In this case, once again, the velocities are zero, so the data written is the same as the ACTION_DOWN method. In addition, the VelocityTracker must be recycled and nulled out, so that it can be re-instantiated on the next ACTION_DOWN event. In all methods, the final String created by the StringBuilder is sent to the ALPActivity method writeToFile, which was created for this assignment. writeToFile takes two String arguments, a filename and data. It uses openFileOutputStream to create a FileOutputStream to the file with the filename that was passed in (this will create the file if it doesnt exist, or append to the file if it does exist). It then writes the data to the file, and closes the FileOutputStream once the data is written.

3 Capturing Sensor Data

3.1 Contributions

Matthew Swartwout - Implementation and debugging James Zhang - None

3.2 Explanation of code and understanding

Part 3 was very similar to Part 2 in that a StringBuilder was used to collect data and the writeToFile method was re-used to create the CSV file. The difference here is that the data from Part 3 is coming from the phone sensors and not from a MotionEvent. To do this, ALPActivity was modified to implement SensorEventListener, which enables it to access sensor data. In onCreate, a SensorManager is created, and then an Accelerometer, Magnetometer, Gyroscope, Rotation, Linear Acceleration, and Gravity sensor are all registered with the SensorManager. Additionally, every sensor returns a float array with an x,y, and z value. In onCreate an 3 entry float array is created for every sensor to store the most recent values. Because ALPActivity now implements SensorEventListener, two methods must be created: onAccuracyChanged and onSensorChanged. onAccuracyChanged is left blank, but onSensorChanged is implemented. onSensorChanged is called every time a sensors value changes. It contains a switch statement that determines which sensor value was changed. Each case calls a helper function which updates the most recent value (stored in the float array) for whichever sensor was updated. After every update, createSensorString is called, which uses a StringBuilder (very similar to Part 2) to create a String containing all the comma-separated sensor values. Then writeToFile is reused to write the sensor data to the CSV file.

4 Logging Data on Lock Pattern Success

4.1 Contributions

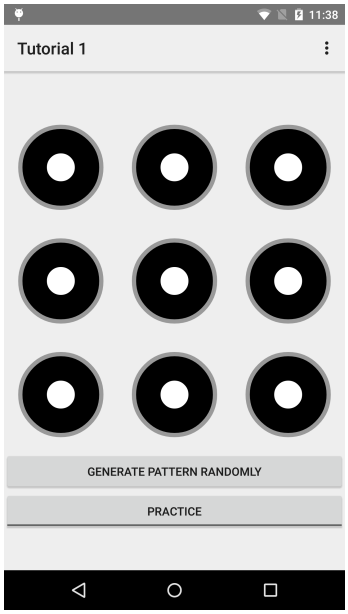
Matthew Swartwout - Implementation and Debugging
James Zhang - None

4.2 Explanation of code and understanding

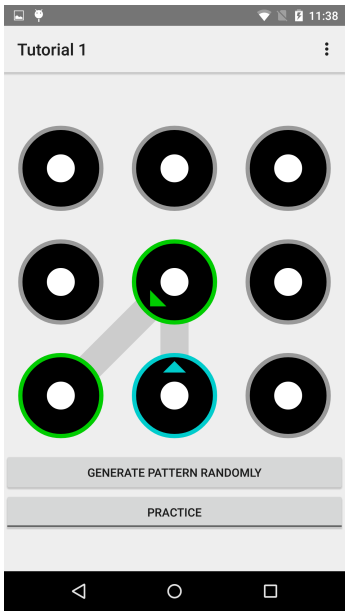
Part 4 was a synthesis of Parts 1, 2, and 3. It involved modifications of ALPActivity and LockPatternView. First, in our ALPActivity we modified the createDataString method to not always write to the file. It appends any info to the string, and then stores that. We created a separate writeToFile method that takes the stored string and writes it to the file. This allows for the program to buffer a long string, and then choose to write to it or throw it away. Next, in our LockPatternView we had to change onTouchEvent and testPracticePattern. In onTouch event we modified it to call createDataString on touch events. This

doesn't write the string to the file, just appends the MotionEvent and sensor data at the time of that event to the string. Then in the testPracticePattern method we added logic to write the file. If the practice pattern is successfully entered then it writes to the string to the file and clears the data buffer. If it is not successful it doesn't write and only clears the data buffer.

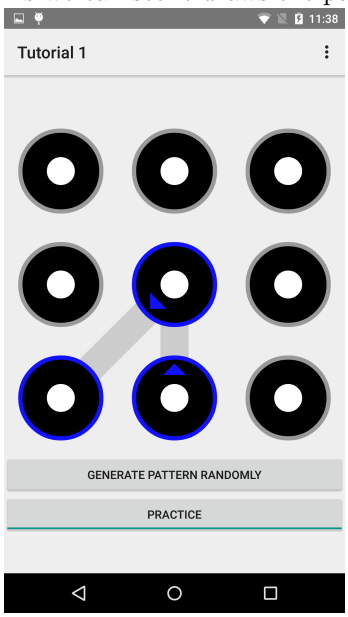
5 Screenshots



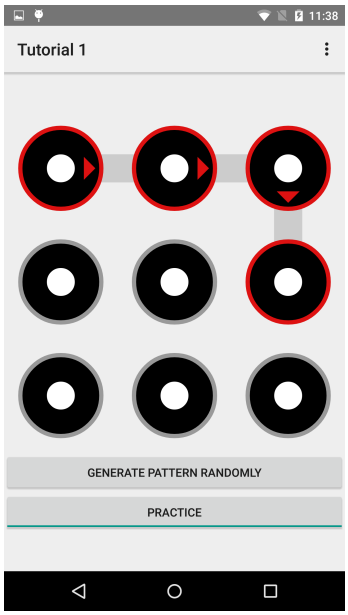
This screenshot shows the default screen when the app is opened, with no pattern generated and not in practice mode.



This screenshot shows the screen after a random pattern has been generated. As we can see it draws the pattern on the screen so that the user can learn it.



This screenshot shows what happens after a successful lock pattern entry. The nodes that were drawn turn blue. We can also see in this image that practice mode is engaged by seeing that the Practice toggle button is highlighted.



This screenshot shows what happens after a successful lock pattern entry. The nodes that were drawn turn red.

6 Project Report Contributions

Matthew Swartwout - Writing, editing
James Zhang - None