# Android App Development

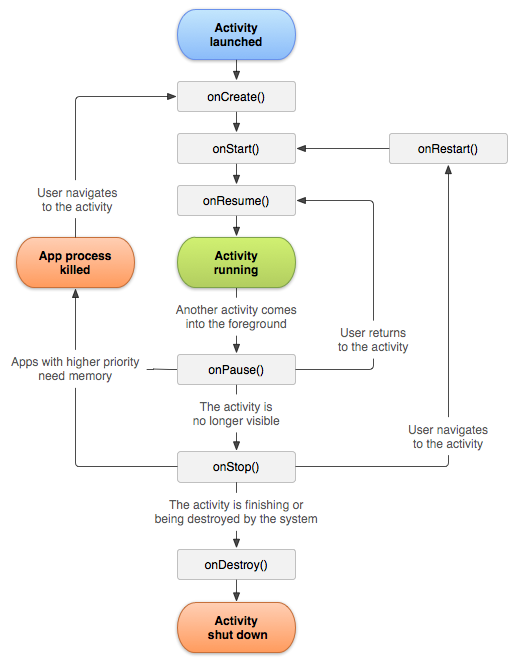
The Bananco application is implemented for Android as Android App. The provided Smartphone Sensing Framework was used to build the BananaCo App on-top because it was already capable of monitoring various sensors, such as acceleration, orientation and magnetic fields. The SSF is also capable of displaying a continuous video stream from the camera and has a sample neuronal network as classifier. To be able to build the Bananaco application using the Smartphone Sensing Framework, one needs to understand the basics in Android App development.

## Manifest

The AndroidManifest.xml contains for the operating system relevant information. For example, required permissions, as well as required features are listed here. In addition, all invokable services and activities are also listed.

In the case of the Bananoco application, the permission to access the camera is listed as ‘uses-permission’ and the main activity is listed as entry-point for the application.

## Activity-Lifecycle

In Android the application must provide an implementation that extends the Activity class and the application must adhere to the Activity Lifecycle (see Figure XX).

As the graph shows, there are multiple steps before and after the actual running state. The states before (onCreate, onStart and onResume) are used to prepare or resume resources required in the running state. For example, at the onCreate stage and in the ImageDetection activity, the camera is activated and the app prepared to be able to process incoming images. The stages onPause, onStop and onDestroy might be invoked after the running stage to tell the application to free resources or to stop processing data. For example, the onPause callback in the Bananaco application ensures that no more images from the camera are retrieved or classified to reduce battery usage.

While onCreate and onDestroy mark the lifetime boundaries of the beginning and end of the activity, the other stages marked by the callbacks are not as trivial. Once an app is no longer visible to the user, the method onStop is called, which allows the app to free all UI-relevant resources and to stop foreground tasks. It also allows the app to actively maintain remaining tasks as background tasks, such as continuing to download game assets. A call of onRestart followed by onStart allows the application to re-allocate UI-relevant resources and to prepare for user interactions. The method onPause tells the application that it is currently no longer in the foreground (another activity is hiding it) and not seen by the user. It is useful to pause foreground tasks to reduce until onResume is called, because those results are not visible to the user.

## Android Layout Definitions

<xml?>

# Graphical User Interface

* 1. This chapter discusses is split into two sections. On the first section the initial mock-up idea is displayed and discussed. The second section will present the final result with the differences to the mock-up and the reasons for the changes.

## Mock-Up

The mock-up can be seen in Figure YY. The main part of the opened Android App is dedicated to the camera live feed. This allows the user to see what is being judged. Below the live feed, two indicators were planned. The first was supposed to indicate whether the app recognized a banana in the image. The second indicator was supposed to indicate whether the banana can be eaten. As can be seen in Figure YY, the indicators used are two basic shapes: a green check mark and a red cross. With this shapes, the following total combinations can be displayed:

|  |  |  |
| --- | --- | --- |
| **Indicator 1** | **Indicator 2** | **Meaning** |
| Red cross | /Doesn‘t matter | No banana detected |
| Green check mark | Red cross | Banana detected, but do not eat |
| Green check mark | Green check mark | Banana detected, can be eaten |

## Final Result

* 1. The final implementation (see table below) also shows the live feed of the camera, but no longer uses the two indicators from the mock-up to display the ripeness value. While investigating how to train the network (link Fabian NN?) we discovered that the network can trained to judge the ripeness of the banana more precisely than ‘eatable’ and ‘not eatable’. Therefore, the indicator displays the ripeness value now, using the results ‘unripe’, ‘ripe’ and ‘overripe’ from the neuronal network. These values are displayed with an explanation and an example image in three separate columns. Finally, a merged result is displayed in the form of a progress bar.
  2. Starting on the left side, the first column shows the gauged ‘unripe’ value, the second column shows the gauged ‘ripe’ value and the third the ‘overripe’ gauged value. These values are used to indicate the lifetime of the banana in the merged progress bar. A low progress bar, ending near the first column ‘unripe’ (see Figure T1) indicates a unripe banana, while an end near the second column indicates ‘ripe’ (see Figure T2) and an ending near the third column indicates ‘overripe’ (see Figure T3). This allows the user with the first look to grasp the current state of the banana.

|  |  |  |
| --- | --- | --- |
| Figure T1: Example of unripe banana | Figure T2: Example of ripe banana | Figure T3: Example of overripe banana |

## Under the hood?

<internal details of the app?>

References

Activity Lifecycle:

<https://developer.android.com/guide/components/activities/activity-lifecycle>

<https://developer.android.com/guide/components/images/activity_lifecycle.png>