# Android App Development

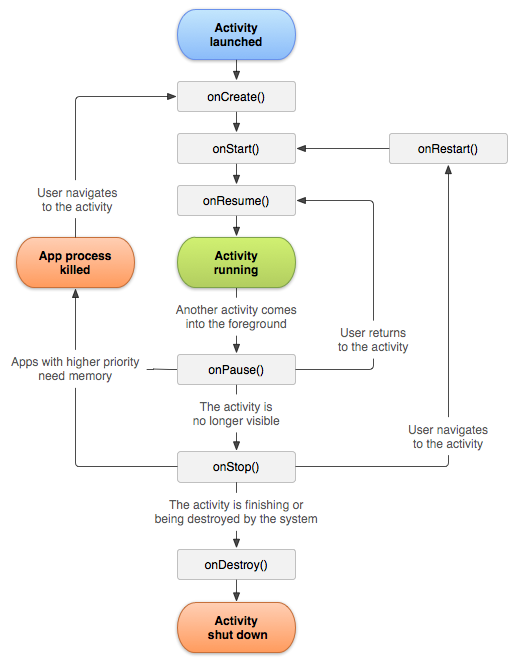
The BananCo application is implemented as App for Android. The workshop provided a reference Android App called “Smartphone Sensing Framework” which was already capable of monitoring various sensors, such as acceleration, orientation and magnetic fields as well as displaying a continuous video stream from the camera with a neuronal network as classifier. The continuous video stream with classifier was used as foundation for the BananCo Android App. Although Java[footnote regarding kotlin?] is used for writing Android Apps, Android enforces additional requirements. For a basic understanding, this chapter will cover some of those.

## Manifest

The file ‘AndroidManifest.xml’ contains relevant information for the operating system. For example, required permissions, as well as required features are listed here. In addition, all invokable services and activities need to be listed. Otherwise those services and activities cannot be used at runtime.

In the case of BananoCo, 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 class inside the App responsible for handling the UI is called ‘Activity’. Besides displaying the UI and handling user input, it must also adhere to the Activity Lifecycle (see Figure XX). To implement custom behavior, one needs to write a class that extends the ‘Activity’ class and overwrite methods where a custom behavior is desired.

As the graph for the Activity Lifecycle shows, there are multiple steps before and after the running state. The steps before (onCreate, onStart and onResume) are used to prepare or re-activate resources required in the running state. For example, at the onCreate stage and in the ImageDetection activity, the connection to the camera is initialized but not activated before onResume was called. The callbacks 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 the connection to the camera is not destroyed before onStop.

The methods callbacks onCreate and onDestroy mark the lifetime boundaries of an activity and are trivial in regards of pre- and post-conditions. If the method onCreate is invoked, one can be sure that no other callback was invoked yet. Once onDestroy is invoked, one can be sure that the Activity won’t receive any further notifications. All other invocations might occure more often and with under other circumstances. 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 or dialog is hiding it) and cannot be 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

For Android Apps it is common to have UI elements defined in a resource file, a so called layout XML file. For each element, depending on the chosen layout, the position and size, default texts or values and ids can be set. The activity will load this layout definition in the onCreate method and retrieve UI elements through their ids. This allows great flexibility because the layout details can be adjusted without interfering with the UI logic. In addition, Android offers the possibility to silently load different XML files depending on the screen orientation if provided. This allows one to easily adjust or totally change the layout depending on the screen orientation without having to change anything in the activity, as long as the ids of all UI elements remain the same.

# 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 |

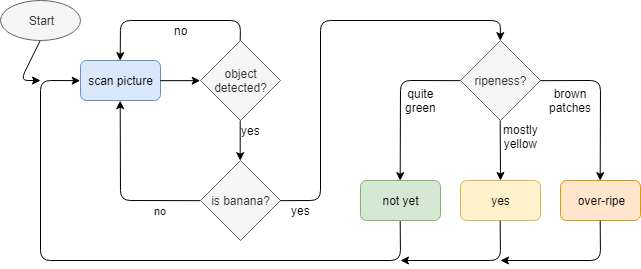
## Final Result

* 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.

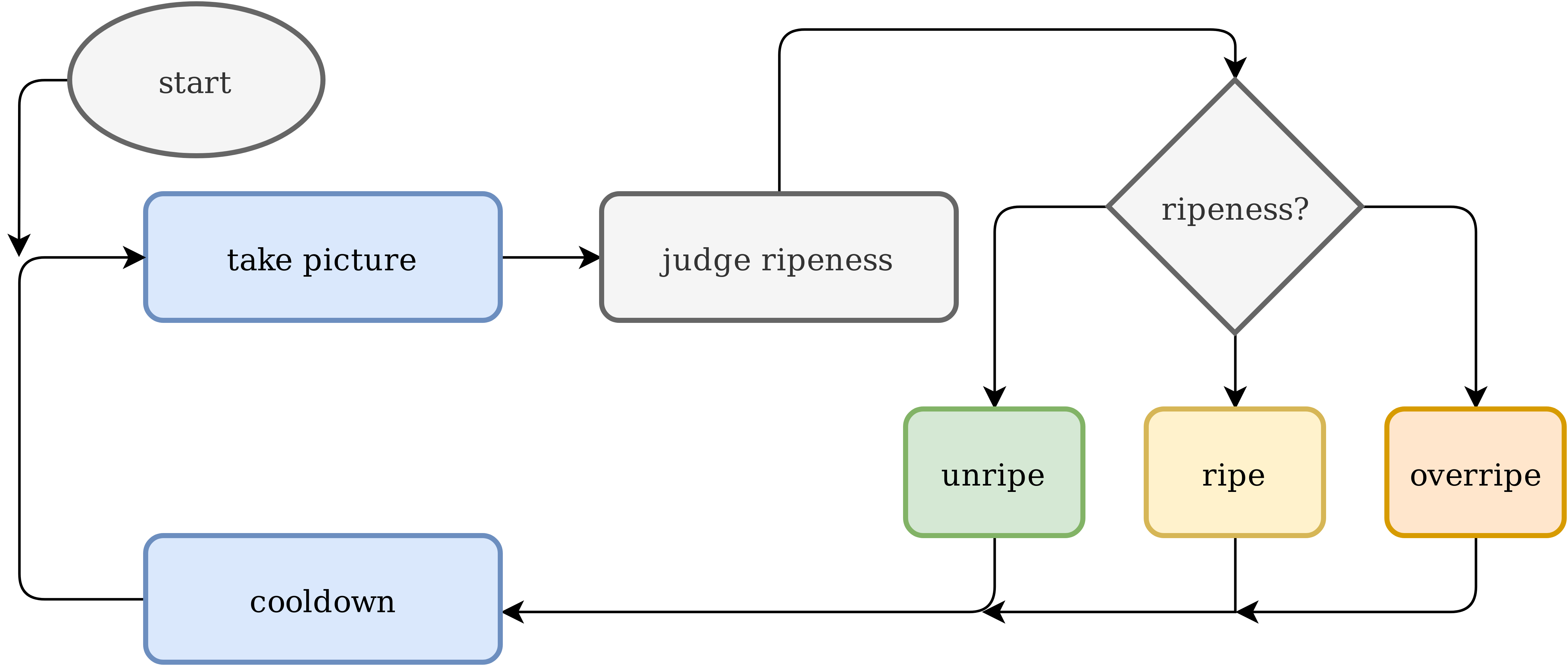
|  |  |  |
| --- | --- | --- |
| Figure T1: Example of unripe banana | Figure T2: Example of ripe banana | Figure T3: Example of overripe banana |

## Operation Principle

* 1. As mentioned in chapter [mock-up] in the early stages, the App was supposed to have an object detection before the ripeness of the banana is judged. This change did not only surface in the UI differing from the UI mock-up but also the flowcharts.

Figure O1

* 1. Figure O1 shows the early flowchart draft while Figure O2 shows the flow chart of the implemented app. In addition to the removal of the object detection, a new step has been added. The ‘cooldown’ step is supposed to reduce the average load over time on the CPU of the smartphone. This shall prevent overheating and CPU throttling as well as increasing battery lifetime. This change was a result of early tests, where the smartphone got uncomfortably warm and where the battery was drained although it being connected through USB to the computer.
  2. The following listing shall explain each step in more detail:
  + ‘take picture’: Takes the current picture from the camera live feed
  + ‘judge ripeness’: Takes the selected picture and feeds it to the neuronal network to retrieve a judgment for the ripeness.
  + ‘ripeness?’, ‘unripe’, ‘ripe’ and ‘overripe’: Depending on the results, update the UI
  + ‘cooldown’: Idle for a bit (500ms) to allow the smartphone to cool down

Figure O2

References

Activity Lifecycle:

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

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

<https://kotlinlang.org/docs/tutorials/kotlin-android.html>