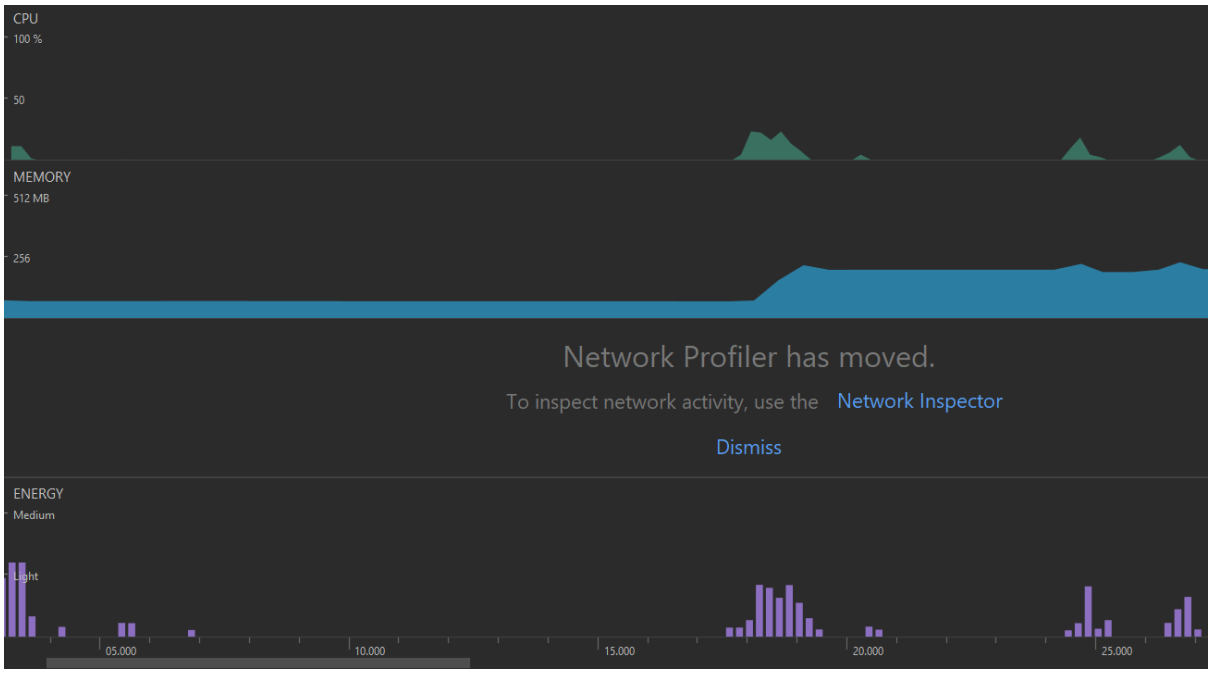
Code Profiling

Performance Testing



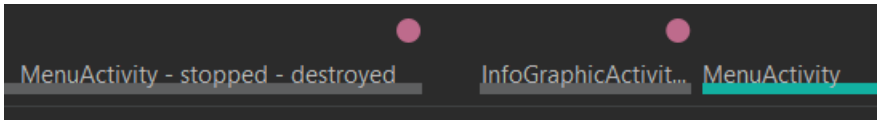
I opened the Encyclopaedia screen at 18 seconds and the memory usage spiked from ~80MB to ~225MB before settling at ~207MB. This is all the images being loaded in. This could be a problem should we have many birds in the future (like the entire Department of Conservation database). We could address this problem by using a RecyclerView which only loads the images in or just outside of view.

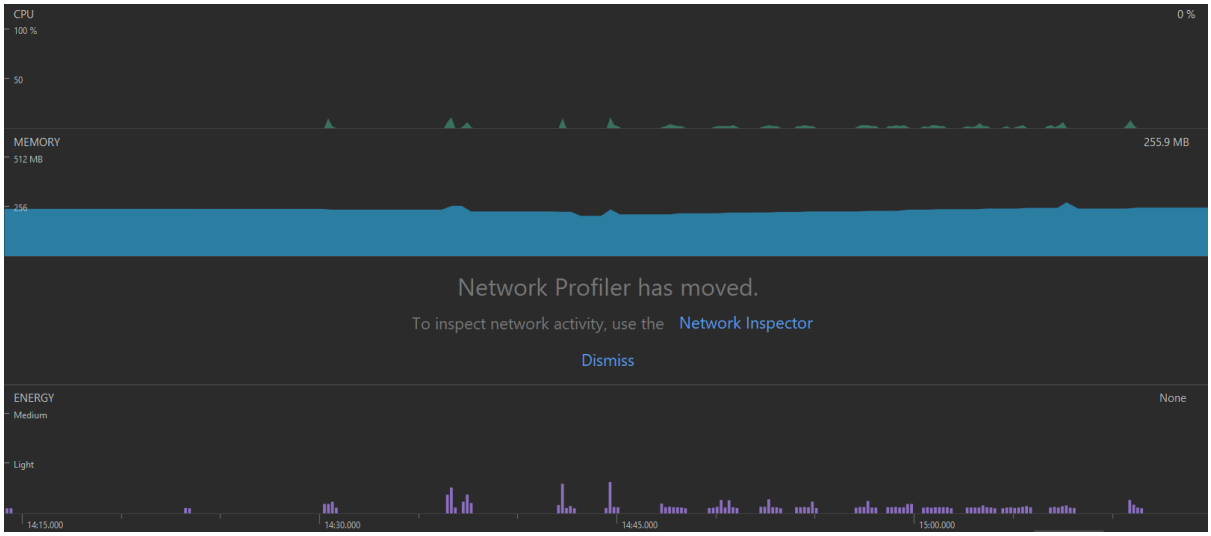
The little bumps at 25 seconds and 27 seconds is me going between the menu and encyclopaedia screens. This shows extra memory is used to facilitate this transition, but the bulk of the memory used by the encyclopaedia screen remains in the menu screen.

There is a 1-2 second lag the first time you open the encyclopaedia but every time after that is quite quick. This result explains why, because the images are not unloaded and are cached by Android, so they do not need to be loaded again.

This result is useful because we may implement a loading animation so users don't think their phone has frozen (e.g., if they had a slow phone).

Edit: it turns out we never finish()ed the encyclopaedia/infographic activity when we went to the menu so it always stayed loaded. The profiler showed us this as follows (note how the first MenuActivity is destroyed but InfoGraphicActivity is not).

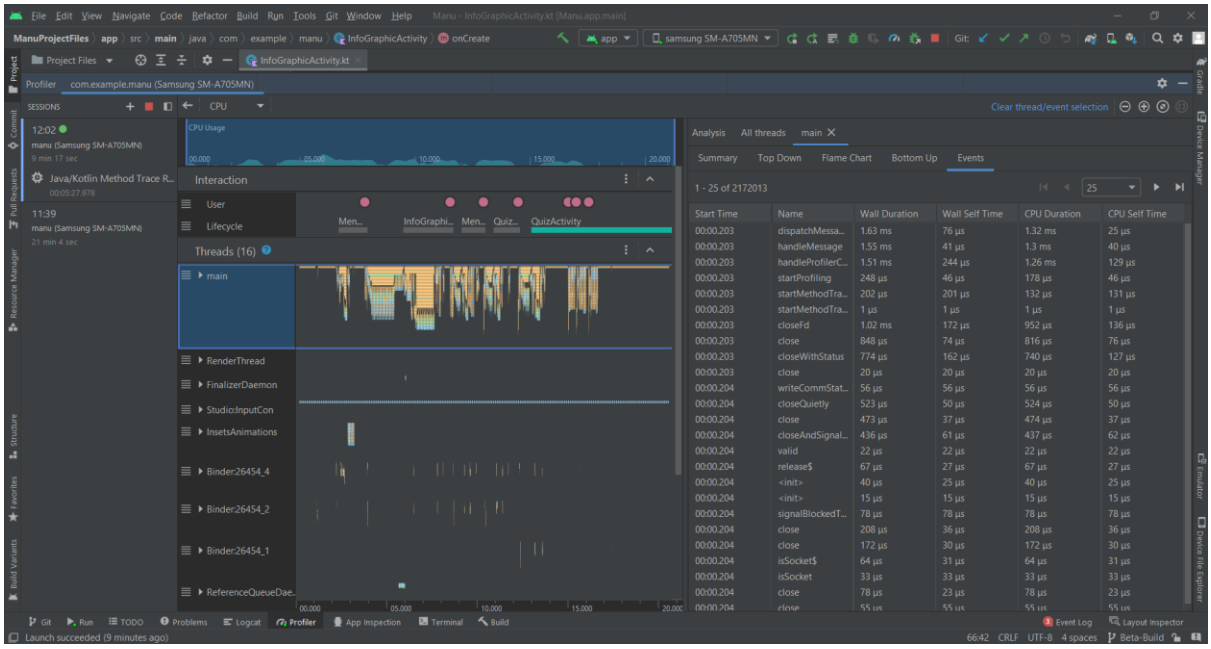




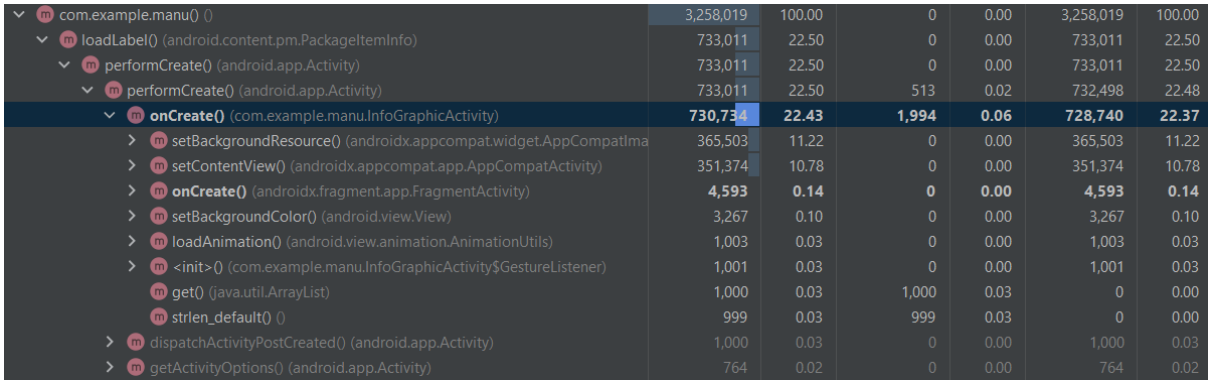
We pressed play at 14:42, selected the photo quiz at 14:45 and answered all ten questions before pressing play again. Clearly, the code that generates and runs the quiz is efficient and does not use much CPU power.

Code Profiling

I did recorded some Kotlin method traces which were quite interesting.



The above shows me playing around on the app. The user row me tapping on the screen and the lifecycle shows the active activity. As seen before, there is a long gap between the menu and infographics screen and a lot of code is involved it loading it, and this code continues consistently for much longer than later events (like me answering questions in QuizActivity).



com.example.manu() ()	3,258,019	100.00	0	0.00	3,258,019	100.00
loadLabel() (android.content.pm.PackageItemInfo)	733,011	22.50	0	0.00	733,011	22.50
performCreate() (android.app.Activity)	733,011	22.50	0	0.00	733,011	22.50
performCreate() (android.app.Activity)	733,011	22.50	513	0.02	732,498	22.48
onCreate() (com.example.manu.InfoGraphicActivity)	**730,734**	**22.43**	**1,994**	**0.06**	**728,740**	**22.37**
setBackgroundResource() (androidx.appcompat.widget.AppCompatIma	365,503	11.22	0	0.00	365,503	11.22
setContentView() (androidx.appcompat.app.AppCompatActivity)	351,374	10.78	0	0.00	351,374	10.78
onCreate() (androidx.fragment.app.FragmentActivity)	**4,593**	**0.14**	**0**	**0.00**	**4,593**	**0.14**
setBackgroundColor() (android.view.View)	3,267	0.10	0	0.00	3,267	0.10
loadAnimation() (android.view.animation.AnimationUtils)	1,003	0.03	0	0.00	1,003	0.03
<init>() (com.example.manu.InfoGraphicActivity$GestureListener)	1,001	0.03	0	0.00	1,001	0.03
get() (java.util.ArrayList)	1,000	0.03	1,000	0.03	0	0.00
strlen_default() ()	999	0.03	999	0.03	0	0.00
dispatchActivityPostCreated() (android.app.Activity)	1,000	0.03	0	0.00	1,000	0.03
getActivityOptions() (android.app.Activity)	764	0.02	0	0.00	764	0.02

As seen above, I was able to profile code that we wrote explicitly, rather than code from the various Android packages. The onCreate() method is called when the activity is loaded. I could inspect how long certain processes took like setting the background image (the ferns). The profiler told me how long it took as a portion of its calling method (because this view was top-down).