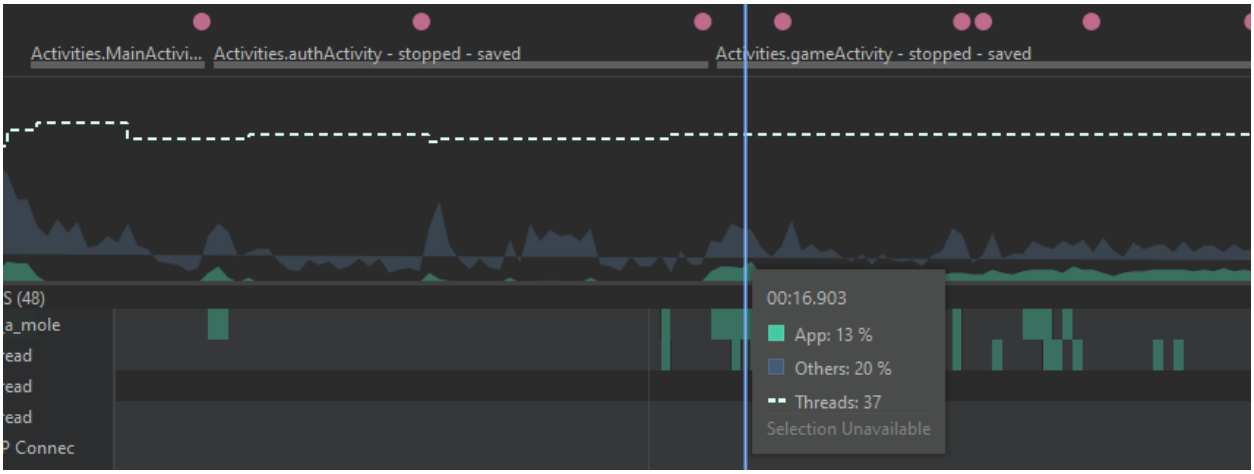
CPU Profiling:

A. Out of the code that you wrote (methods you wrote or methods you called), which consumes the most CPU time?
   a. Looking at the profiler, we find that the game activity consumes most of the CPU's time – with our runnable that sets the moles taking up most of the CPU time.

B. Screenshot:



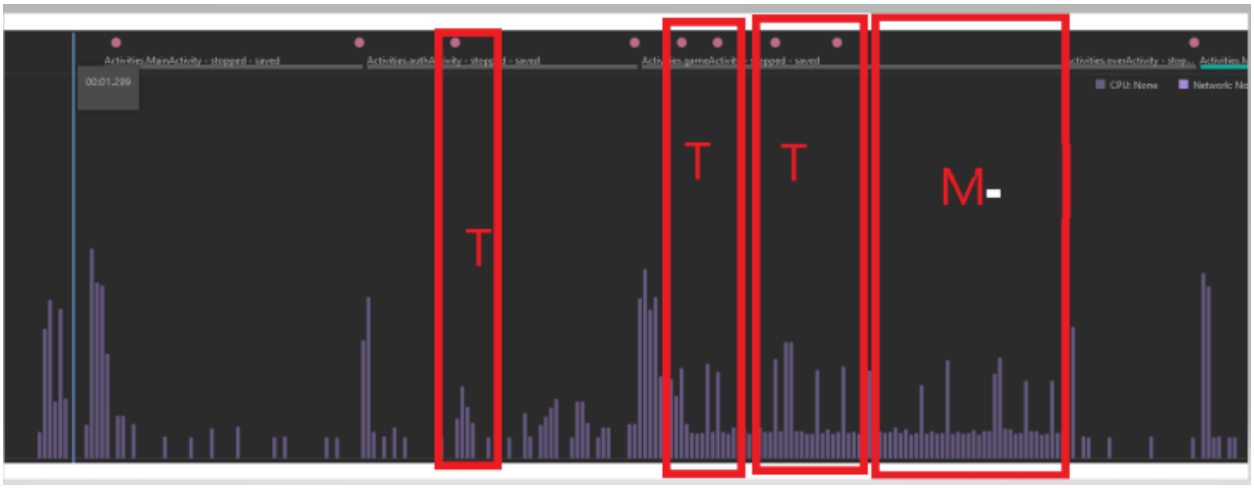
C. Discuss whether the results were expected, or if there were any surprises, and why.
   a. This is to no surprise, the game activity requires a handler, a runnable, and the model classes. Even more shocking is how lightweight the application is. It is hard to see from the screenshot – but at its worse its at roughly 15% CPU usage with around 95MB of memory being used. This trend appears to apply to the rest of the sections we're about to discuss.

Energy Profiling:

A. Labeled Screenshot: There are a few interesting spots in the energy profile – firstly, I split this up until three screenshots to provide clarity and because it looks nicer.
   a. To start off, we see that the energy usage spikes up when we launch and activity:

b. Next, we see the consistent usage during the game activity, this is to be expected due to the nature of the game with the handler, runnable, and the models:

c. Finally, I wanted to point out that there are also spikes in energy usage whenever I interact with the screen – additionally – there were also spikes when I missed a mole. This behavior can be seen here (T for touch) (M for misses):

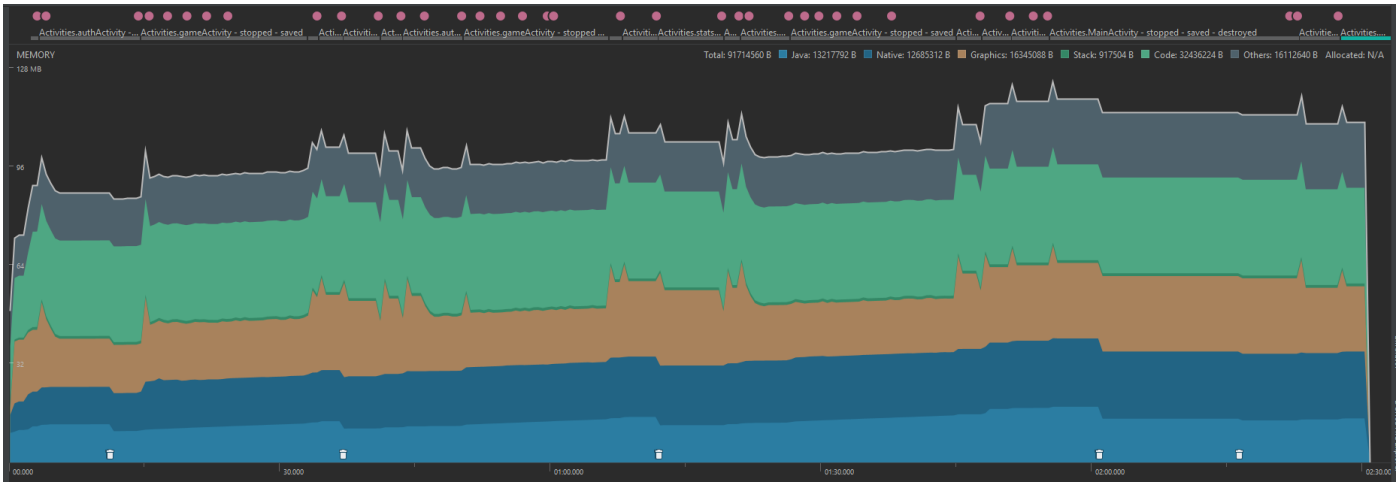


B. Based on what you see here, does your app use more or less energy than you expected? Discuss.

a. Based on this, I suspected my app would use a bit more energy – again due to all of the moving pieces, however – even at its peaks – the usage is still classified under “low energy.” The locations where usages where peaked and consisted also makes sense – the launching of the application can be seen during the first peak – you can see the device’s efforts to launch the app and run the main activity. The rest of the artifacts are also in line with what I discussed in the CPU profiler – it has consistent usage during the game activity with spikes due to the interaction – and additional usage to play audio, etc.

Memory Profiling 1:

A. What is the general trend in memory consumption of your app over time?
    a. The general trend here follows a similar trend to the CPU usage and the energy consumption – it is higher during the game activity – and gradually increases as time goes on – averaging around the 100+ MB range. You can see an increase across all buckets as the game activity runs.



B. Did you expect the trend in the previous question, or is it a surprise? Discuss.
    a. Yes, this was expected as we know there are different instances of the models and such. This is inline with the previous two profiler's results – the game activity being the more demanding activity out of the group.
C. Based on your observations, which of the memory buckets is used to store objects (i.e., the object heap)? Why?
    a. Based on what we can see, the Java bucket stores our objects for us. As the game progresses – we can see that it's usage increases with time, driving the overall usage up. Additionally, we force garbage collection – it is the bucket affected most with a clear decline (native bucket also sees similar activity, however I suspected its in conjunction with the Java bucket).

Memory Profiling 2:

A. Examine the code, how does this memory leak work?
    a. The way this memory leak works is that whenever you run this application – and you press the big red button, it essentially creates 1000000 new cat objects every single time you press the button. As time goes on, Android will clear these – however – if you press the button enough times to fill the buffer, it will cause the application to crash.
B. Force your app to run out of memory on a virtual device. How much memory is being consumed by the app when this happens? What specific error is triggered?
    a. Around 200MB were consumed when the app crashed. This seems to have been an out of memory error according to this:

```
2022-10-13 17:04:35.697 10726-10726/edu.iastate.cpre388.memoryleak E/AndroidRuntime: FATAL EXCEPTION: main
    Process: edu.iastate.cpre388.memoryleak, PID: 10726
    java.lang.OutOfMemoryError: Failed to allocate a 36920416 byte allocation with 25165824 free bytes and 25MB until OOM, target footprint 200142944, growth limit 201326592
        at java.util.Arrays.copyOf(Arrays.java:3645)
        at java.util.Arrays.copyOf(Arrays.java:3614)
        at java.util.ArrayList.grow(ArrayList.java:275)
```

C. Open the advanced settings for the virtual device (AVD manager -> drop on the right side -> Edit -> Show Advanced Settings). Scroll down to the Memory and Storage settings. Based on the settings shown here and your answer to the previous question, which memory limit is your app hitting when it crashes?
    a. According to this, our limit for dynamically allocated objects (the Heap) is around 384 MB, as seen here. Therefore the memory limit, and although I mentioned around 200MB on the previous question (Android Studio is a bit unresponsive on my laptop), the memory limit is 384MB for apps.