**PROG 8170**

**Software Quality Assurance**

**Assignment 3**

**Hassan Nahhal**

**Chang Ho Choi**

**SungJoe Kim**

**Nicholas Collins**

**Ahmad Aljumiai**

**Outline:**

1. **Used an automated user interface clicking tool**

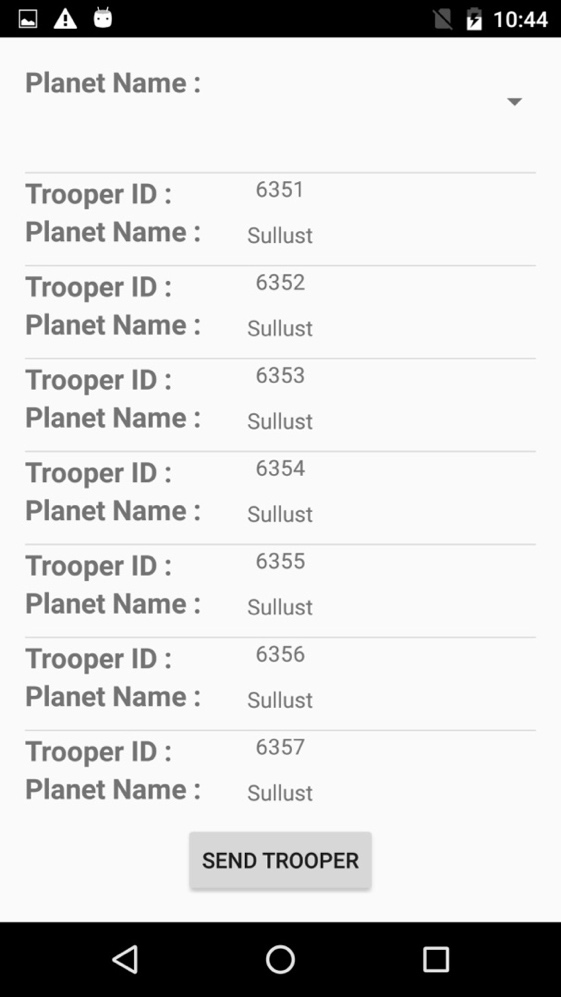
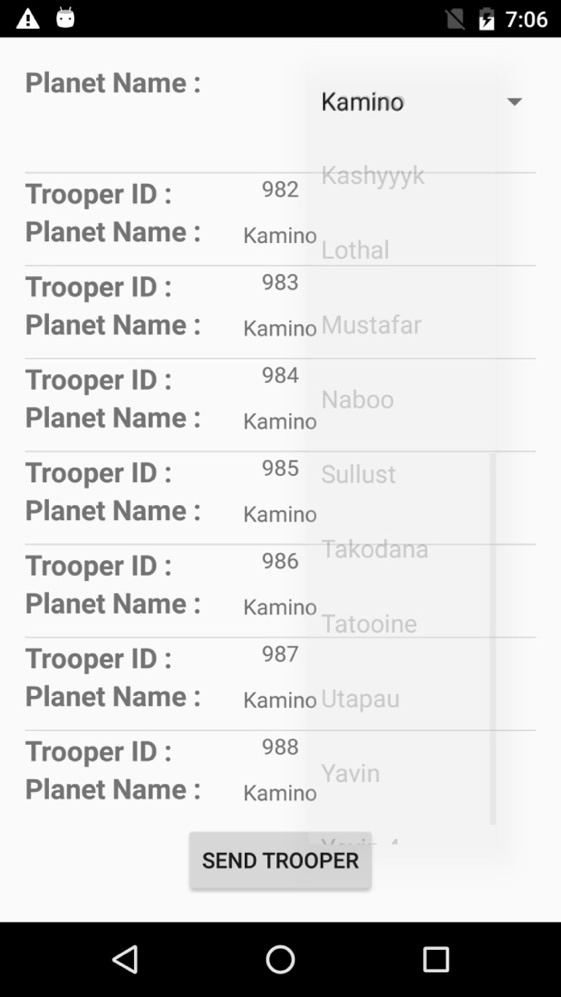
ApplicationTest is implemented from scratch and uses Rubitium Recorder in Android Studio. It uses Solo, its main class, to manipulate views and activities.

The method below will click on the spinner so it drops down the list of stars.

*solo.clickOnView(spinner);*

then click inside the spinner at an index randomly generated to select one of the stars.  
*solo.clickOnView(solo.getView(TextView.class, randInt()));*

Finally, click on the button to add the Trooper created to the database in MainActiviy.java.  
*solo.clickOnView(solo.getView(R.id.button))*

**

1. **Identify the performance metrics (mean time to failure, mean time to repair, number of storm troopers that crashes the system)**

From SQLite official site, “The maximum size of a database file is 2147483646 pages. At the maximum page size of 65536 bytes, this translates into a maximum database size of approximately 1.4e+14 bytes (140 terabytes, or 128 tebibytes, or 140,000 gigabytes or 128,000 gibibytes).”

* Approximate testing time = around 16 hours
* Mean time to failure = around 5 hours
* Mean time to repair = around 1 min (time to rerun the app)
* Mean number of storm trooper generated = around 6400

Neither the app nor the database crashed, the reason why the app stopped running is because the GC (Garbage collector) is being called a lot because of the low memory and it needs to be allocated.

Another issue that might arise is the heap, it might be that the heap is very big and couldn’t prevent overflow.

Last LogCat statement:

*12-08 22:31:32.717 29894-29894/? D/dbcon.getLastId():: 6357*

*12-08 22:31:32.717 29894-29894/? D/Trooper planet: Sullust*

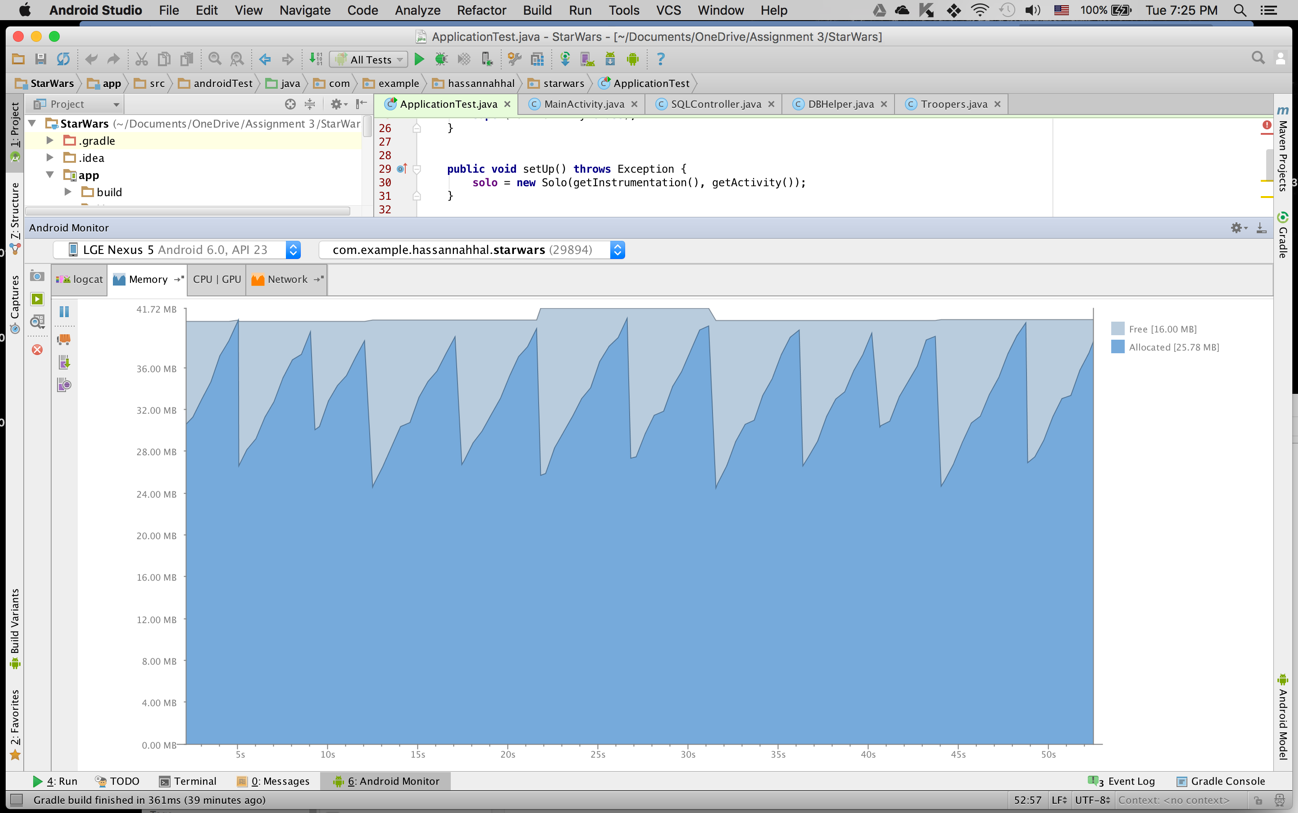
*12-08 22:31:33.880 29894-29894/? I/art: Starting a blocking GC Alloc*

*12-08 22:31:34.115 29894-29894/? I/art: Alloc sticky concurrent mark sweep GC freed 1800039(77MB) AllocSpace objects, 138(3MB) LOS objects, 7% free, 54MB/58MB, paused 852us total 233.822ms*

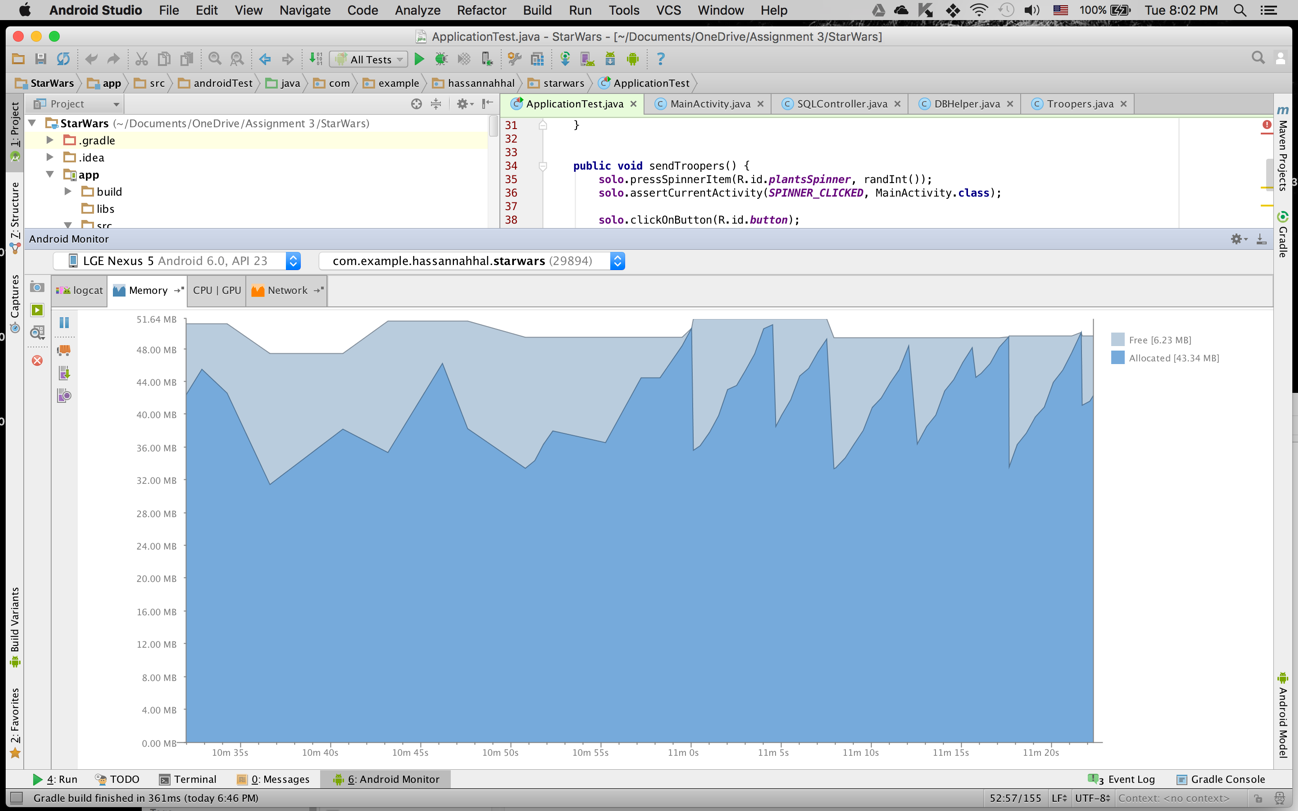
1. **Identify the performance metrics of the user interface (CPU load and memory usage graph)**

In terms of memory usage, until almost 1200 Troopers were loaded, the usage was normal and at 20.00 MB.

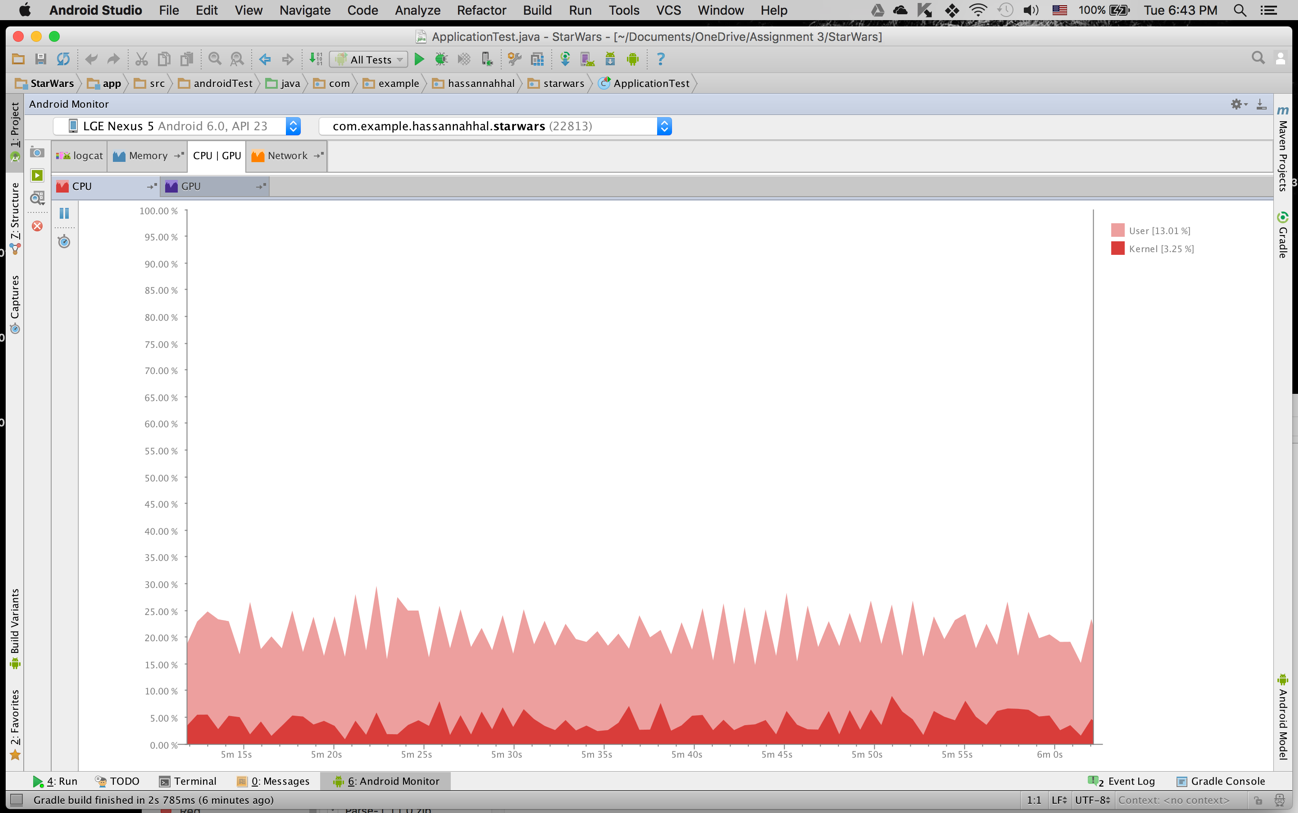
However, when the database had around 2100 Troopers, memory usage increased till it reached almost 40.00 MB.



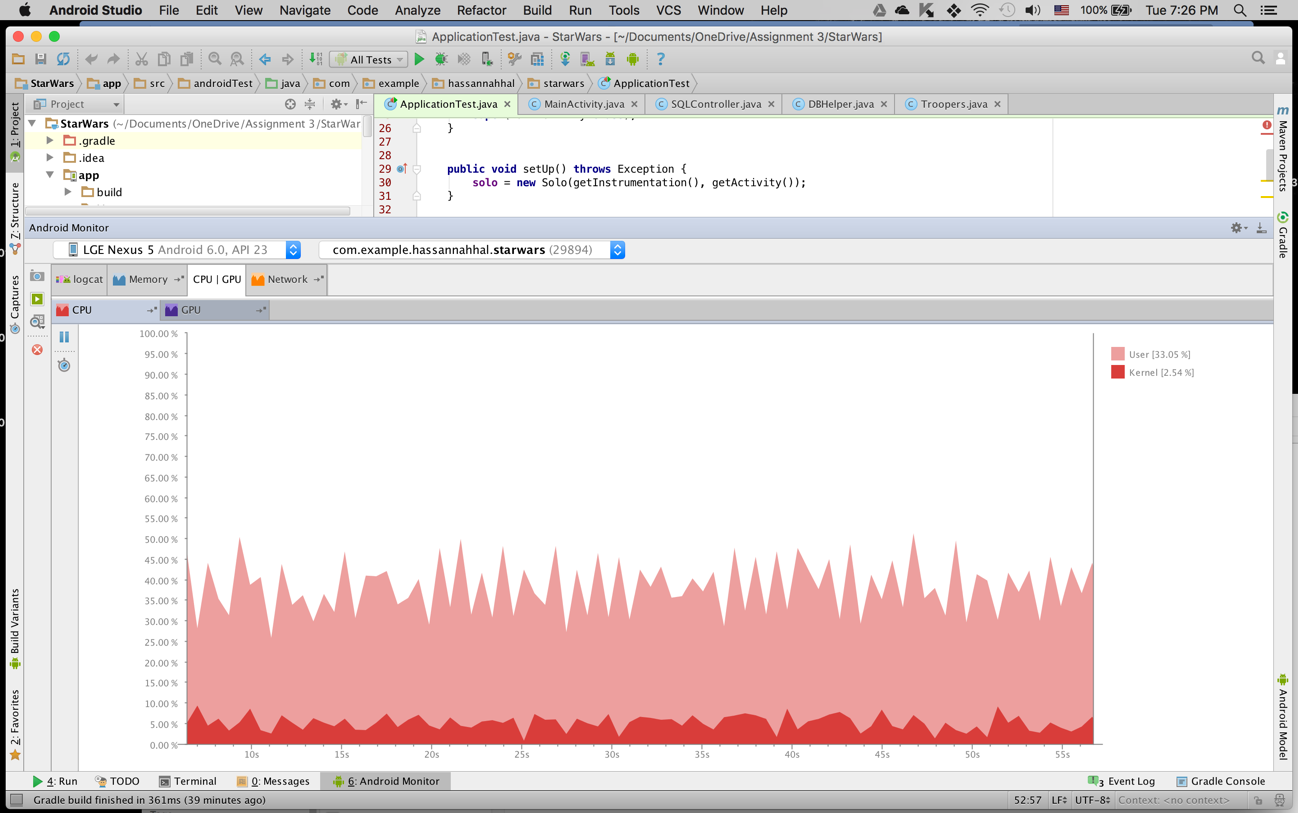
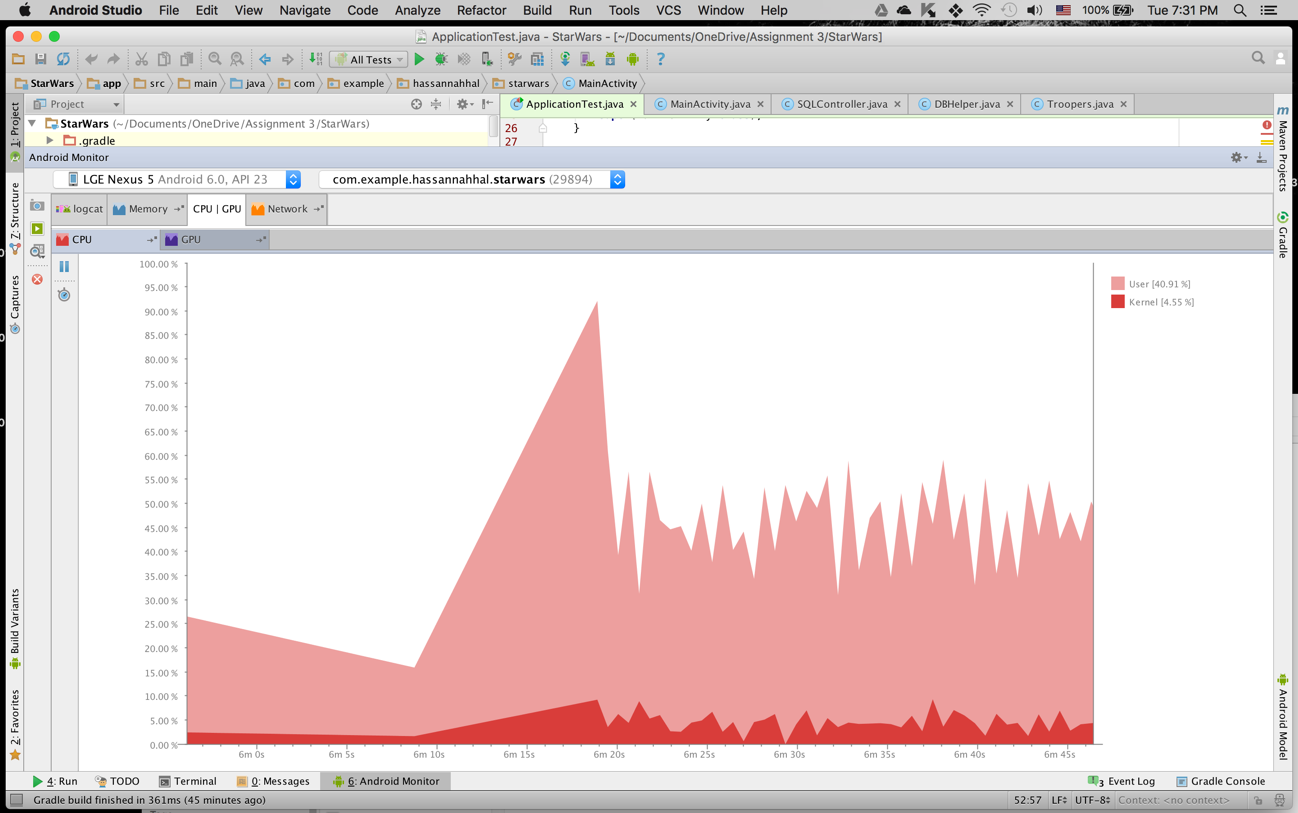
At 3500 Trooper, memory usage increased to 48.00MB



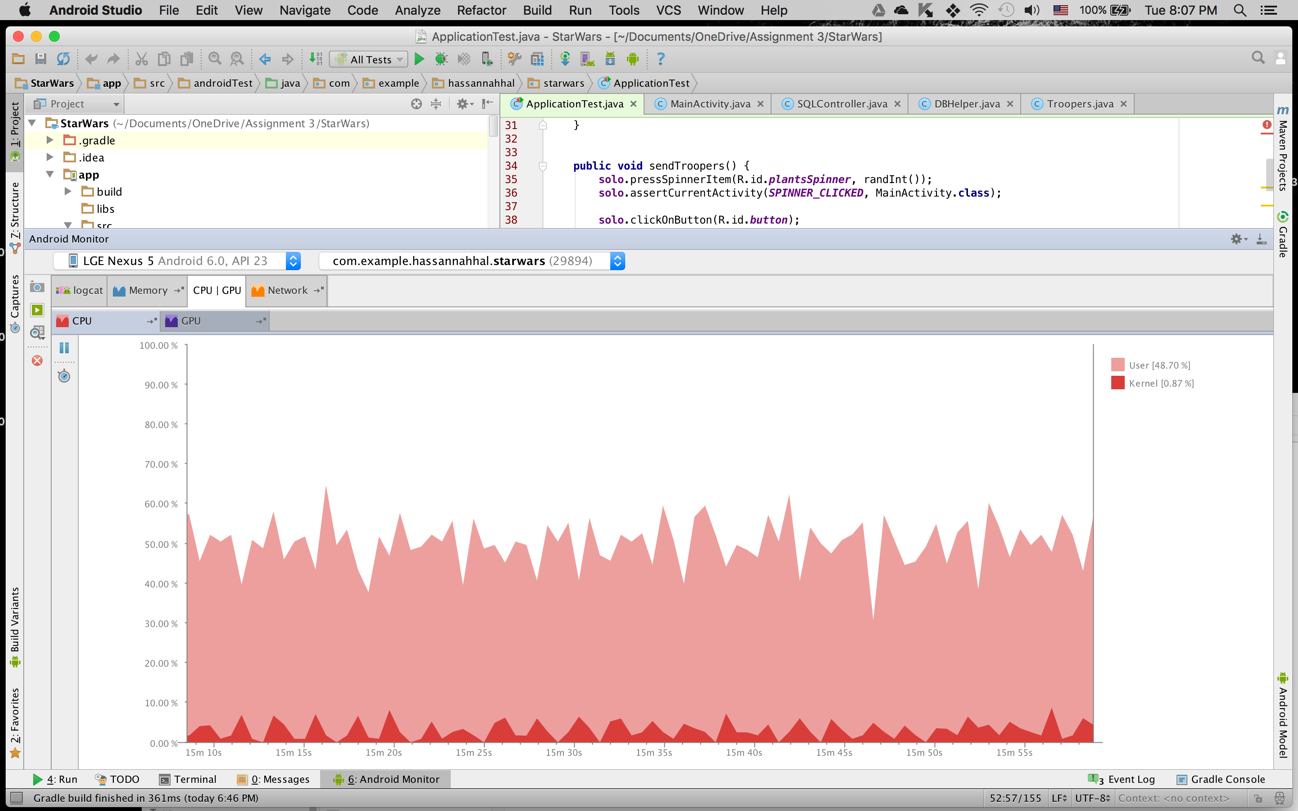
CPU was normal and under 35% till 1200 Trooper.



when the number of Troopers exceeded 2100 Troopers, it hit 50% use.

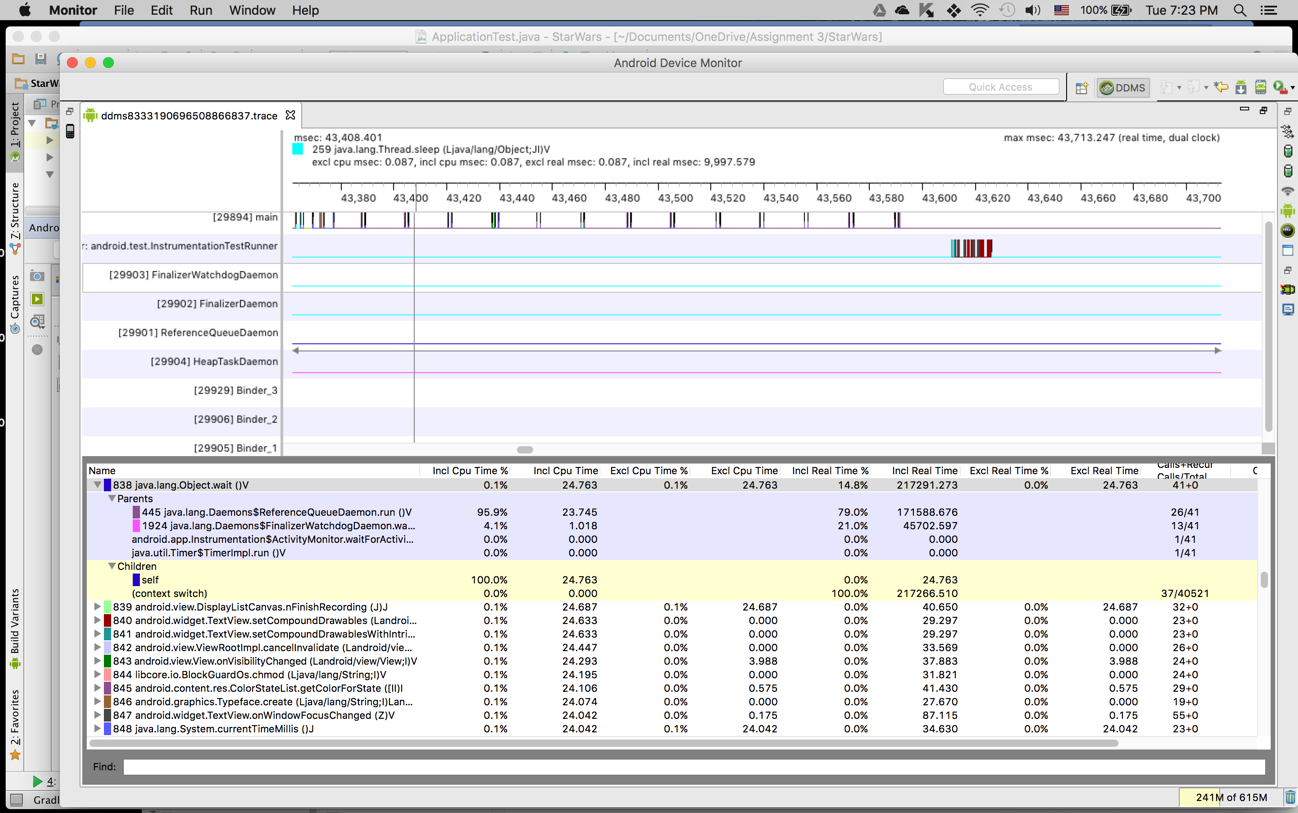
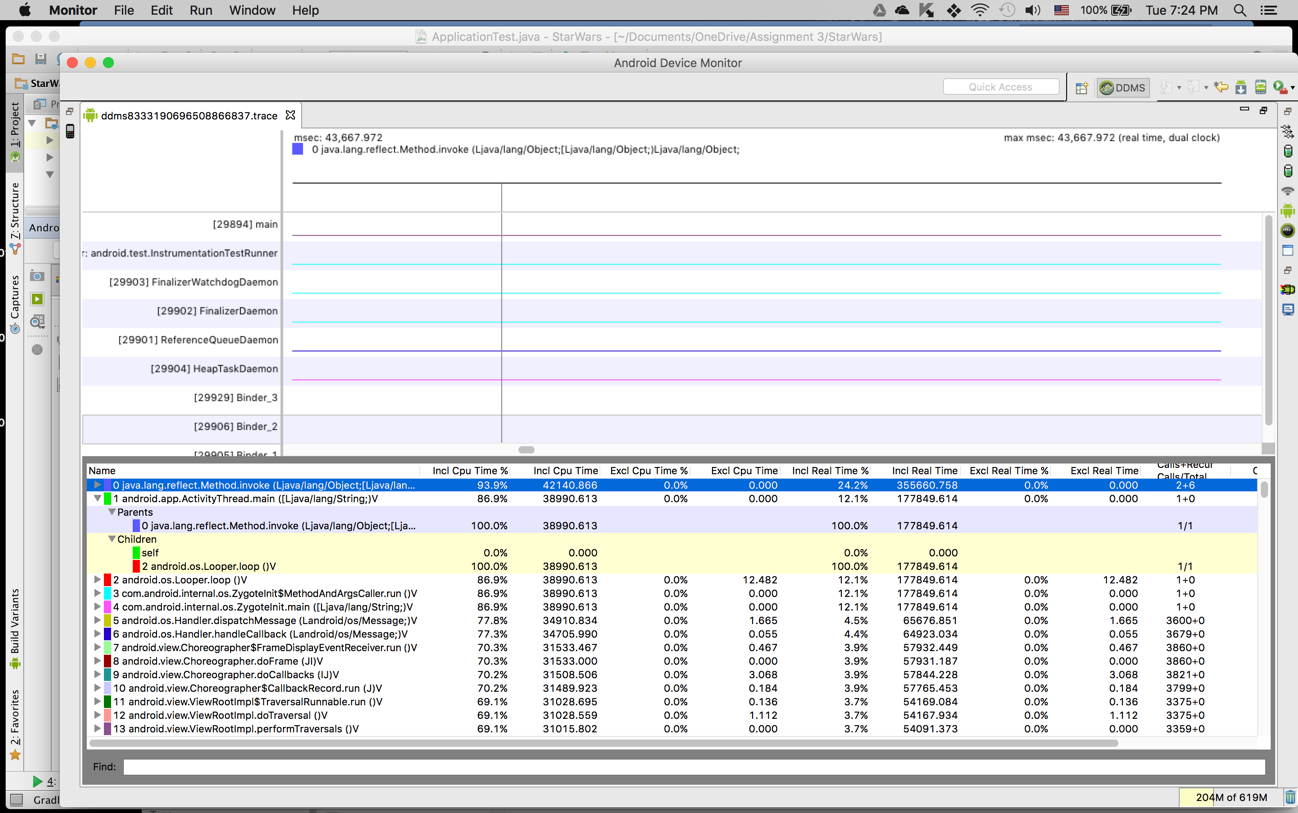


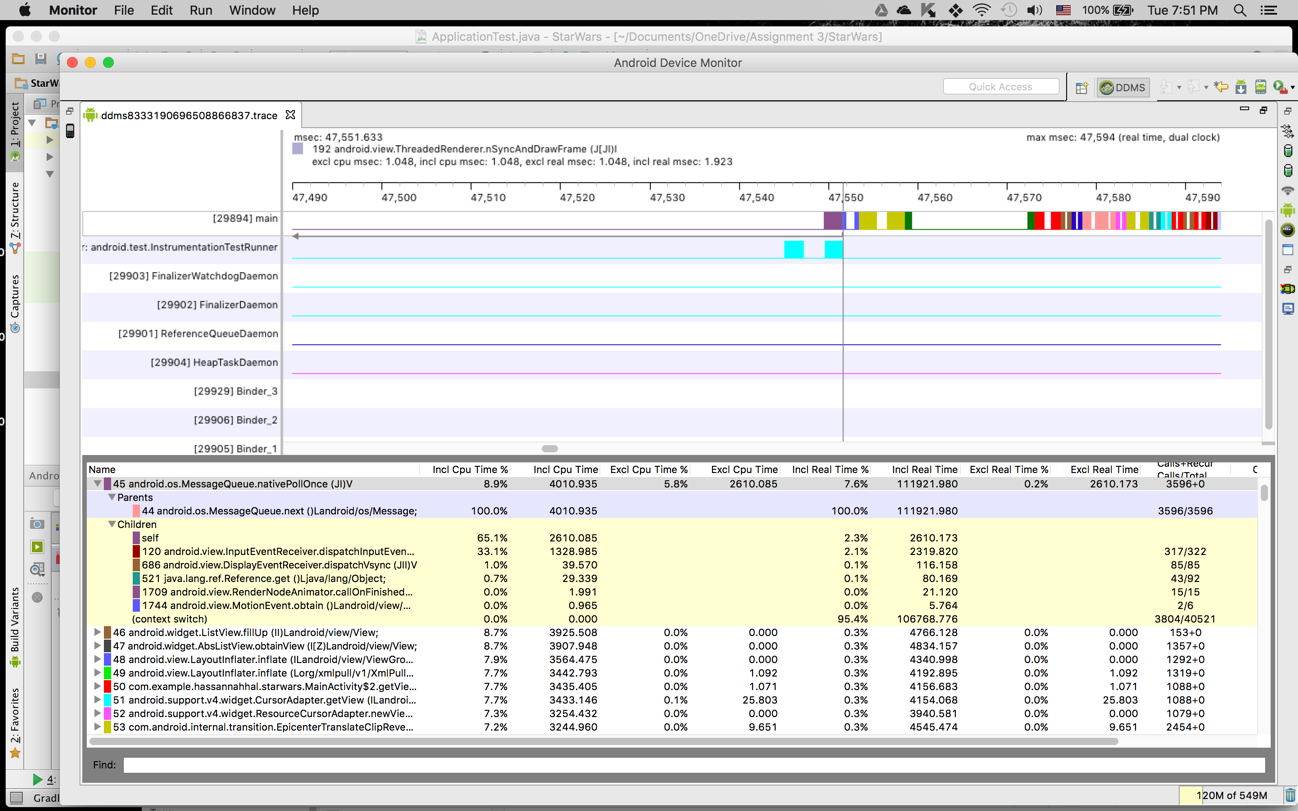
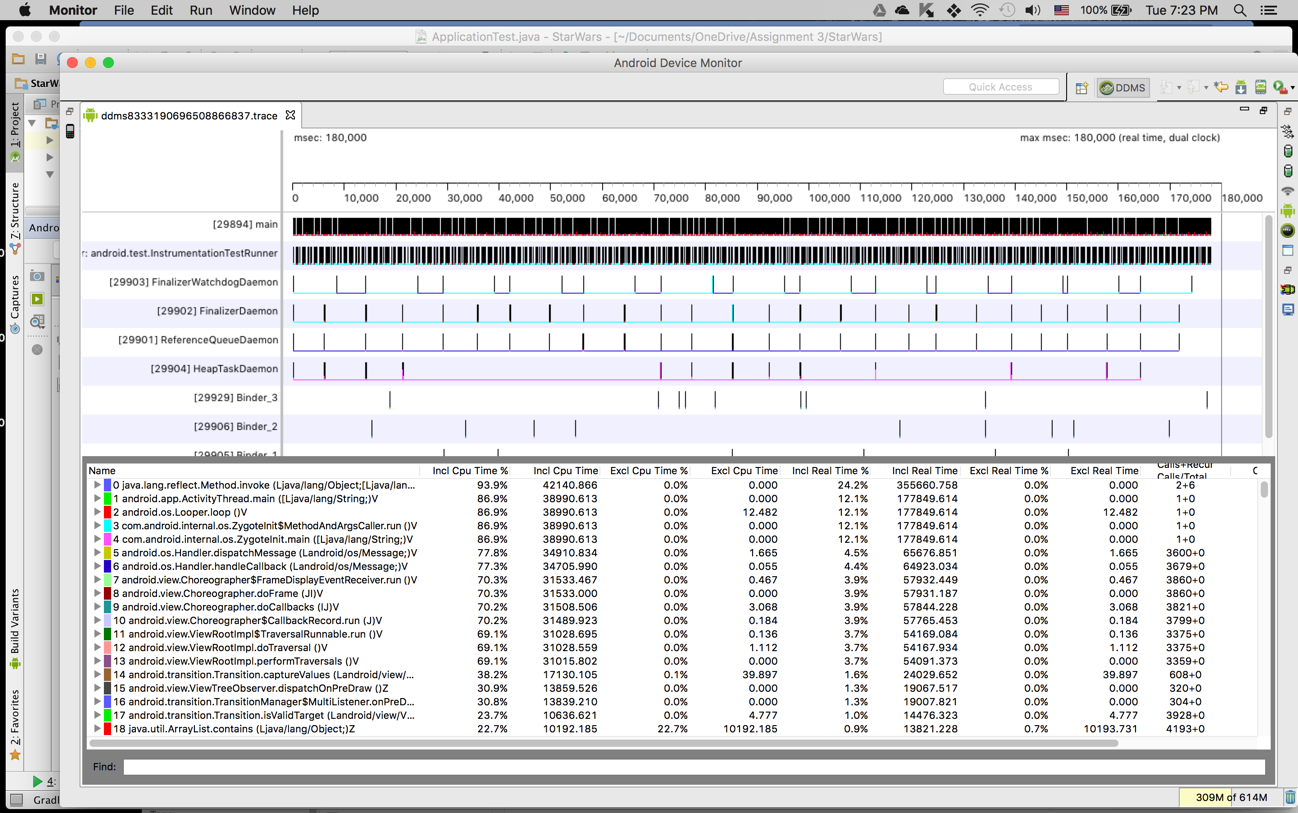
at 3500 Troopers, CPU usage increased to 55%



Using Android Device Monitor and DDMS (Dalvik Debug Monitor Server) we could know real time data on which methods are being executed and the ones that use the most of the memory and CPU.

The images show how the usage started to change and increase when the number of troopers increased. Zooming into the graph shows more data about each method execution, moving left and right in the graph allows us to monitor the data in respect to time.





1. **Implement the database and the user interface successfully**

In terms of saving troopers, we implemented SQLite database in android including 2 rows, TROOPER\_ID and TROOPER\_STAR, TROOPER\_ID is auto incremented by SQLite whereas TROOPER\_STAR is being selected from a spinner that contains all the stars that have been supplied to us.

To keep track of numbers of Troopers created, we log to logcat the id of the last element in the cursor, hence it will be the last id of troopers generated.