



## **LOG8371- Ingénierie de la qualité en logiciel**

**Hiver 2022**

**TP2**

**Par**

**1955913 - Andi Podgorica**

**1949477 - Ming Xiao Yuan**

**1957746 - Gabriel Moreau**

**1947701 - Simon Kaplo**

**Équipe Bonaventure**

**Soumis à**

**Moutassim Abderazzak**

**21 Mars 2022**

## Table des matières

<b>1. Plan d'assurance qualité</b>	<b>3</b>
1.1 Couverture de la qualité	3
1.2 La stratégie de validation	5
1.2.1 Profiling	5
1.2.2 Monitoring	5
1.2.3 Tests de performance	6
<b>2. Préparation pour le profilage d'AntennaPod</b>	<b>7</b>
<b>3. Profilage d'AntennaPod</b>	<b>7</b>
3.1 Recherche Podcast	7
3.1.1 Consommation CPU	8
3.1.2 Consommation Mémoire	8
3.1.3 Consommation Énergétique	9
3.1.4 Détection de Ui Jank	10
3.1.5 Analyse	11
3.2 Démarrage	12
3.2.1 Consommation CPU	12
3.2.2 Consommation Mémoire	13
3.2.3 Consommation Énergétique	13
3.2.4 Détection de Ui Jank	14
3.2.5 Analyse	15
3.3 Lecture d'un podcast	16
3.3.1 Consommation CPU	16
3.3.2 Consommation Mémoire	17
3.3.3 Consommation Énergétique	18
3.3.4 Détection de Ui Jank	19
3.3.5 Analyse	20
<b>4. Benchmarking</b>	<b>20</b>
<b>5. Bibliographie</b>	<b>25</b>

# 1. Plan d'assurance qualité

## 1.1 Couverture de la qualité

Ce rapport vise la mise en place d'un plan d'assurance qualité se basant sur le critère de performance. Ce critère fait référence aux ressources nécessaires pour exécuter les différentes fonctions du système. Afin d'assurer une bonne couverture, trois sous-critères seront évalués en fonction de deux objectifs distincts. Le tableau ci-dessous présente ces objectifs et la façon dont ils seront mesurés.

### 1.1.1 Sous-critères de la qualité de l'application AntennaPod

No.	critère de qualité	Sous-critère	Objectif <sup>1</sup>	Mesures de validation <sup>2</sup>
1	Performance	<p><b><u>Comportement du temps:</u></b></p> <p>Degré auquel le temps de réponse de la fonctionnalités satisfait les exigences</p>	<p>La taux X du temps de la réponse d'une recherche d'un podcast doit être inférieur à 1. C = 2 second. (ISO/IEC FDIS 25023:2015-12 PTb-2-G)</p>	<p><b><u>La formule :</u></b></p> $X = (\sum_{i=1}^n (B_i - A_i) / n) / C$ <p>Ai: Temps à la réception de l'intrant i. Bi: Temps à la transmission de l'extrant associé i. n: Nombre d'observations. C: Temps de réponse maximum.</p>
2	Performance	<p><b><u>Comportement du temps:</u></b></p> <p>Degré auquel le temps de délai de la fonctionnalités satisfait les exigences</p>	<p>La taux X du temps de délai du démarrage doit être inférieur à 1. C = 2 second. (ISO/IEC FDIS 25023:2015-12 PTb-4-G)</p>	<p><b><u>La formule :</u></b></p> $X = (\sum_{i=1}^n (B_i - A_i) / n) / C$ <p>Ai: Temps au commencement du travail i. Bi: Temps à la fin de ce</p>

<sup>1</sup> ISO25020\_Modele\_mesure\_base

<sup>2</sup>ISO25021\_Gabarit\_metriques

				travail i. n: Nombre d'observation C: Temps de Délai maximum
3	Performance	<p><b><u>Utilisation des Ressources:</u></b></p> <p>Degré auquel la quantités d'utilisation du CPU satisfait les exigences.</p>	La moyenne X d'utilisation du CPU lors de la lecture d'un épisode doit être inférieure à 50%. (ISO/IEC FDIS 25023:2015-12 PRu-1-G)	<p><b><u>La formule:</u></b></p> $X = \sum (A_i / B_i) / n$ <p>Ai: Utilisation réelle du CPU pour une durée d'observation i. Bi: Durée d'observation i. n: Nombre d'observations.</p>
4	Performance	<p><b><u>Utilisation des Ressources:</u></b></p> <p>Degré auquel la quantités d'utilisation de la mémoire vive les exigences.</p>	Le taux X d'utilisation de la mémoire vive lors de l'ordonnancement de la liste de lecture doit être inférieur à 75%. (ISO/IEC FDIS 25023:2015-12 PRu-4-S)	<p><b><u>La formule:</u></b></p> $X = \sum (A_i / B_i) / n$ <p>Ai: Utilisation réelle de la mémoire vive pour une durée d'observation i. Bi: Durée d'observation i. n: Nombre d'observations.</p>
5	Performance	<p><b><u>Capacité:</u></b></p> <p>Degré auquel la limites maximales d'utilisateurs simultanés satisfait les exigences</p>	La taux X de la limite maximale d'utilisateurs simultanés lors de la lecture d'un épisode doit être supérieur à 2. B: 200000 (environ le nombre d'utilisateurs d'AntennaPod).	<p><b><u>La formule:</u></b></p> $X = (\sum (A_i / n)) / B$ <p>i = 1 à n A: Nombre d'utilisateurs qui peuvent accéder simultanément au système à l'observation i; n: Nombre d'observations; B: Capacité minimum d'utilisateurs simultanés.</p>
6	Performance	<p><b><u>Capacité:</u></b></p> <p>Degré auquel la limites maximales d'augmentation d'utilisateurs simultanés satisfait les exigences</p>	La taux X de la limite maximales d'augmentation d'utilisateurs simultanés effectuant un téléchargement de l'épisode doit être supérieur à 2. B: 100000 (environ le nombre d'utilisateur d'AntennaPod)	<p><b><u>La formule:</u></b></p> $X = A / B$ <p>A: Capacité d'augmentation d'utilisateurs simultanés. B: Capacité minimum d'augmentation d'utilisateurs simultanés.</p>

## 1.2 La stratégie de validation

Pour valider nos objectifs définis plus haut, différentes techniques devront être utilisées. Cette section les définit brièvement et associe nos objectifs avec la technique permettant de les évaluer.

### 1.2.1 Profiling

La première méthode de validation possible est le profilage avec échantillonnage (Sampling). Il s'agit en effet d'une forme d'analyse dynamique qui mesurera la complexité du temps et d'espaces d'AntennaPod pendant son exécution. De plus, cette méthode est intéressante dans notre analyse car nous pouvons acquérir des données de performance sans ajouter de la surcharge dans le code comme suggérerait le profilage avec l'instrumentation. À l'aide des interruptions à intervalles réguliers dans le CPU, nous pouvons donc enregistrer les instructions et ainsi connaître la fréquence d'exécution des points dans le code. Cette méthode de validation sera plus tard exécutée sur le code source d'AntennaPod et les résultats empiriques seront présentés par la suite.

### 1.2.2 Monitoring

Une autre méthode de validation possible serait le monitoring. Cette méthode permet de monitorer AntennaPod sur plusieurs niveaux, que ce soit au niveau matériel ou au niveau du système d'exploitation. En retour, ce dernier permet de mesurer l'application pendant son exécution comme le temps de réponse au niveau application ou la multilocation au niveau matériel et de l'infrastructure. Vu que la méthode du monitoring requiert des logiciels externes tels que CloudWatch d'AWS ou Ceilometer d'OpenStack, notre équipe a décidé de ne pas employer ce dernier dû aux besoins de ces infrastructures.

### 1.2.3 Tests de performance

Une dernière méthode de validation serait le test de performance. Ce dernier regroupe plusieurs types de tests comme le *Load Testing*, qui permet de tester la performance du système sous la charge attendue, le *Stress Testing*, qui teste les limites de la capacité d'AntennaPod ou le *Capacity Testing*, qui permet d'identifier la capacité maximale du système. Ces trois différents types de test sont de bonnes mesures pour évaluer le bon fonctionnement d'AntennaPod et sa capacité limite. Le *Endurance testing* et le *Spike testing* sont également efficaces pour vérifier l'application sur une longue durée et des changements subites des nombres d'utilisateurs. Un autre type de test est le *Configuration testing*, permettant de tester le système sous différentes configuration et analyser l'impact des changements. Tester les diverses configurations du système ne sont toutefois pas un objectif aussi essentiel que celles évaluées dans les tests précédents pour AntennaPod.

### **1.2.4 Stratégie utilisée par objectif**

No. Objectif	Stratégie	Outil
1	Benchmark	Android Studio / Emulator
2	Benchmark	Android Studio / Emulator
3	Profiling	Android Studio / Emulator
4	Profiling	Android Studio / Emulator
5	Spike Testing	Staging environnement / Emulators
6	Capacity Testing	Staging environnement / Emulators

## 2. Préparation pour le profilage d'AntennaPod

Notre équipe procède maintenant à appliquer la méthode de validation avec le profilage avec échantillonnage sur l'application AntennaPod comme mentionné dans le plan d'assurance qualité. Les étapes de préparation du profilage seront détaillées ci-dessous.

Tout d'abord, il faut avoir Android Studio d'installer avec un émulateur valide ainsi que le code source d'AntennaPod. Android Studio fournit par défaut les outils nécessaires pour effectuer le profiling d'une application. Il suffit ensuite de compiler AntennaPod et de lancer l'émulateur. La vue de profiling peut être ouverte dans l'onglet view -> Tool Windows -> Profiler. Pour effectuer la détection de UI jank, il faut enregistrer la trace du système. Cela demande une étape de plus. Une fois le profilage lancé, il faut naviguer dans l'onglet CPU, puis sélectionner l'option *System trace recording* et appuyer sur Record.

## 3. Profilage d'AntennaPod

Suite à la préparation, nous allons maintenant effectuer le profilage sur 5 des fonctionnalités d'AntennaPod en profilant leurs consommation CPU, mémoire et énergétique. Le profiling est effectué avec l'émulateur suivant:

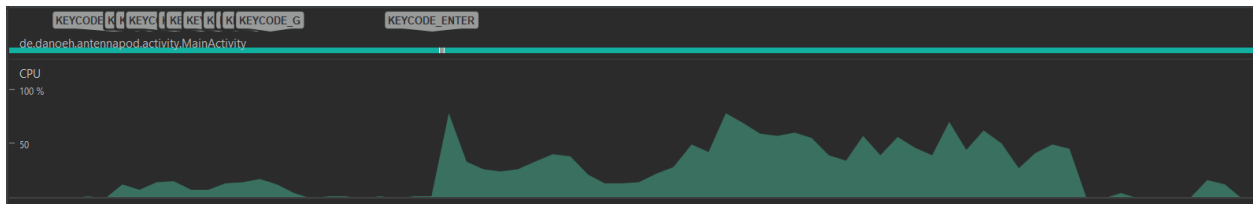
Nom	Résolution	Version API	Destination	CPU
Pixel 5	080 x 2340: 440 dpi	30	Android 11	x86

Une visualisation des résultats sera présentée pour chaque fonctionnalité.

### 3.1 Recherche Podcast

Ce test consiste à effectuer une recherche d'un nouveau podcast dans l'onglet *Add Podcast*. La partie intéressante du profiling commence dès que la touche enter est pesé et se termine une fois que les résultats de la recherche sont affichés à l'écran.

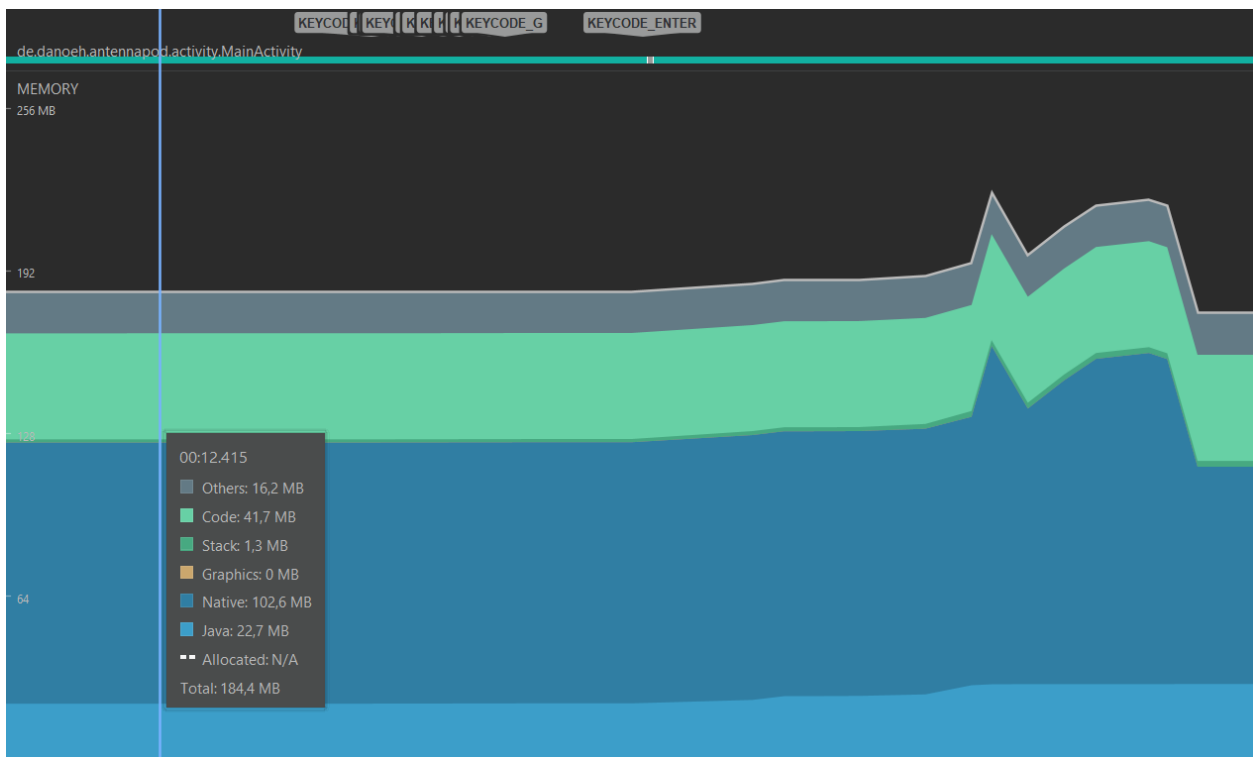
### 3.1.1 Consommation CPU



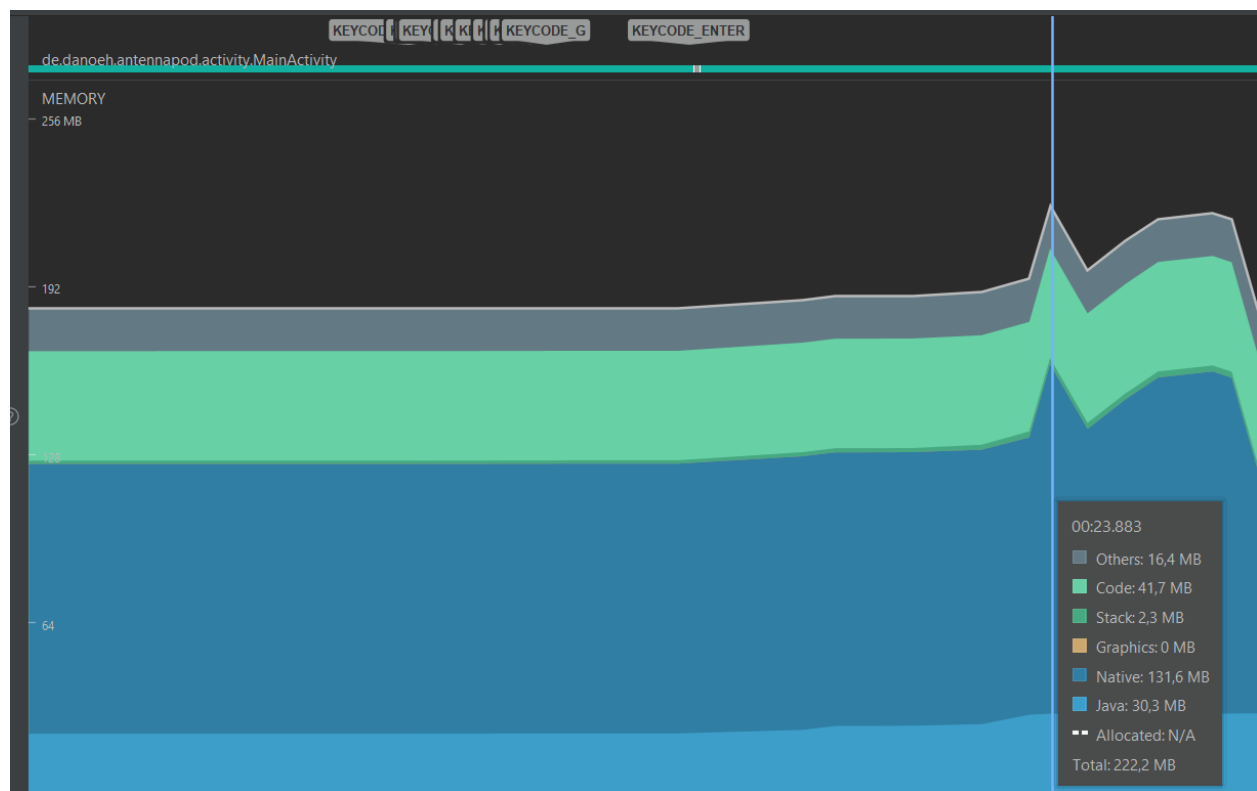
*Figure 1: Consommation du CPU lors d'une recherche*

On peut observer un sommet à 79% de consommation CPU pour l'application.

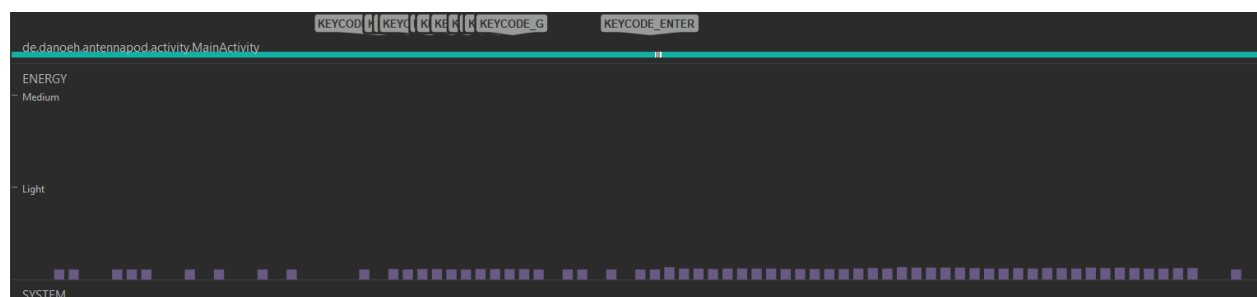
### 3.1.2 Consommation Mémoire



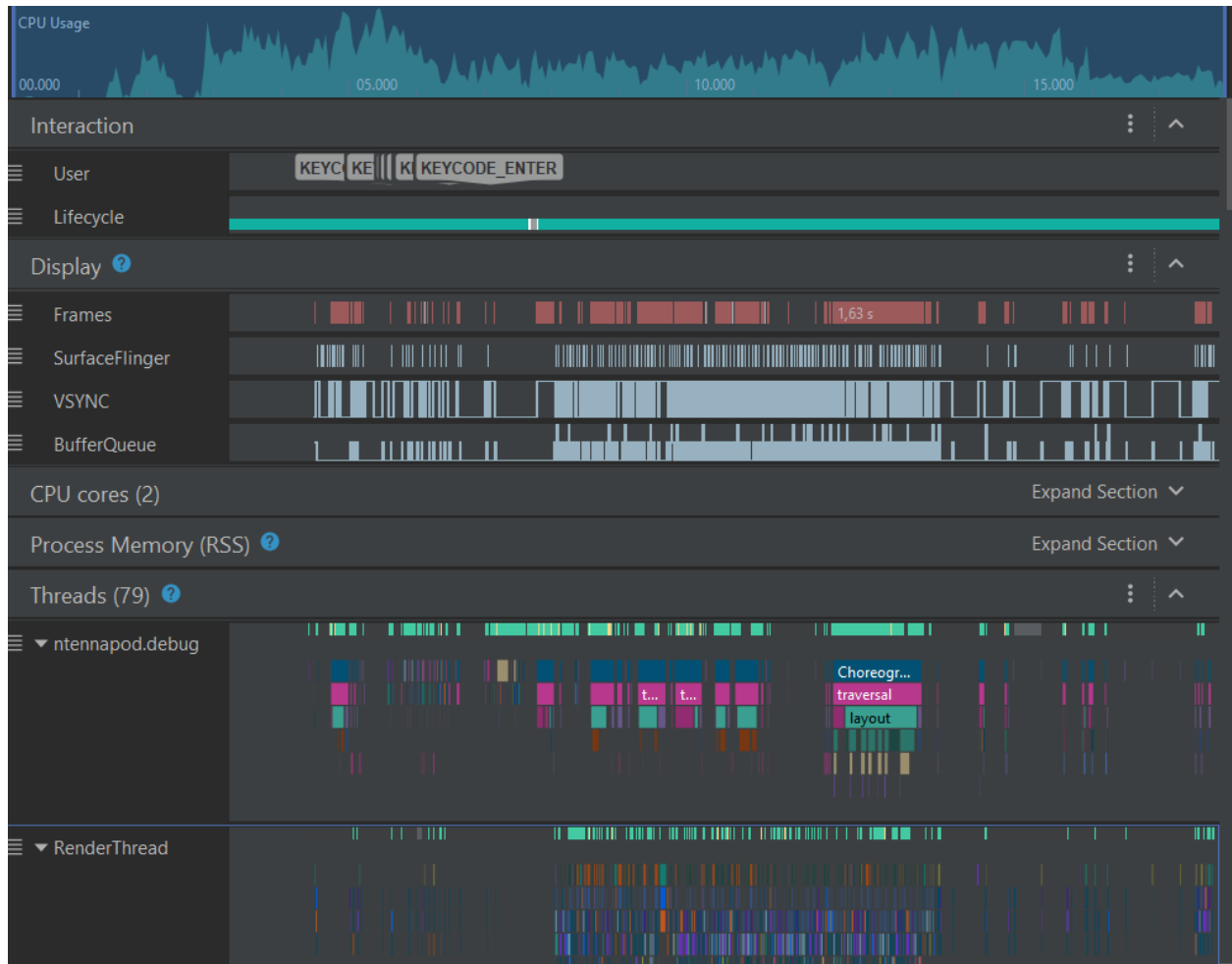


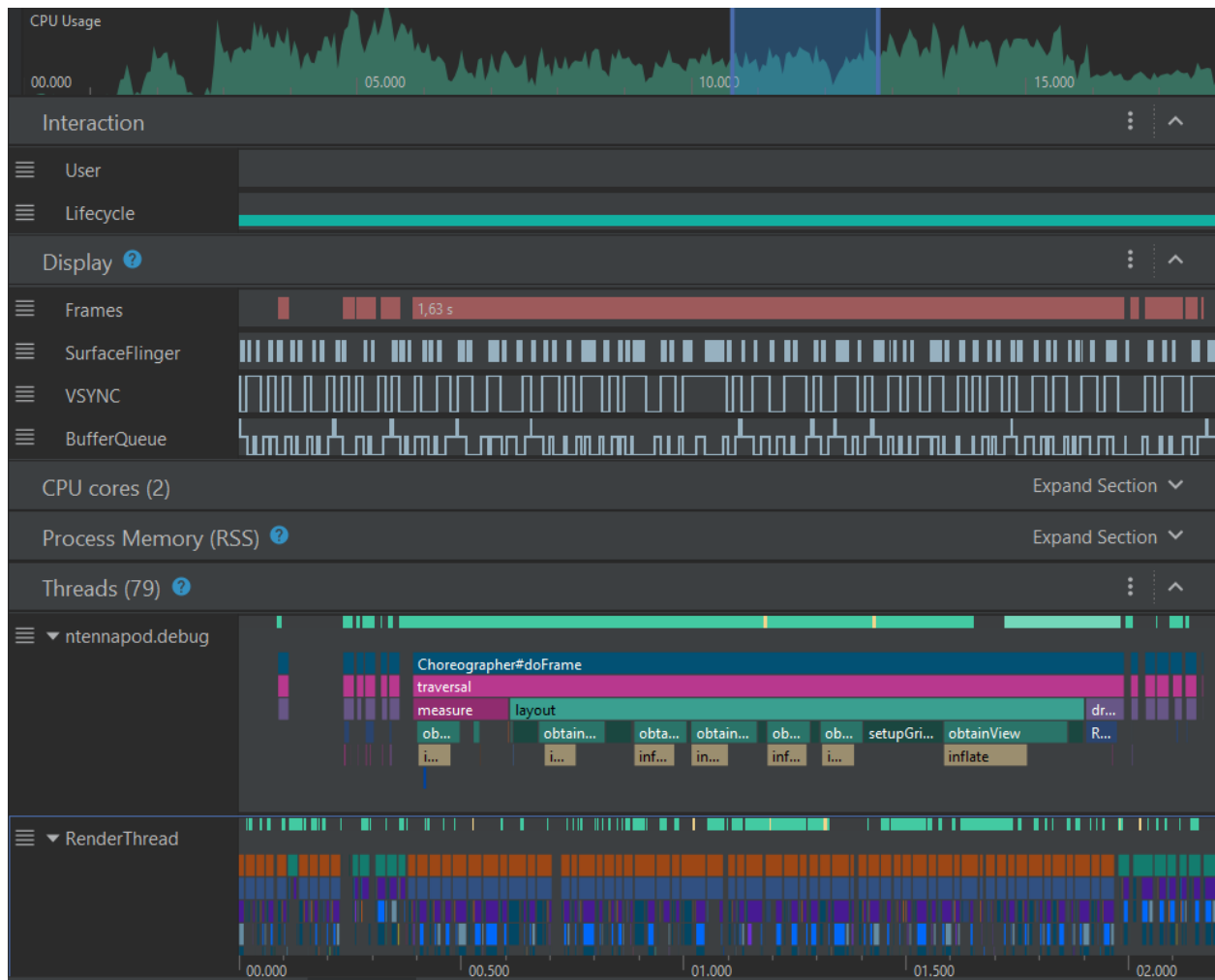


### 3.1.3 Consommation Énergétique



### 3.1.4 Détection de Ui Jank





### 3.1.5 Analyse

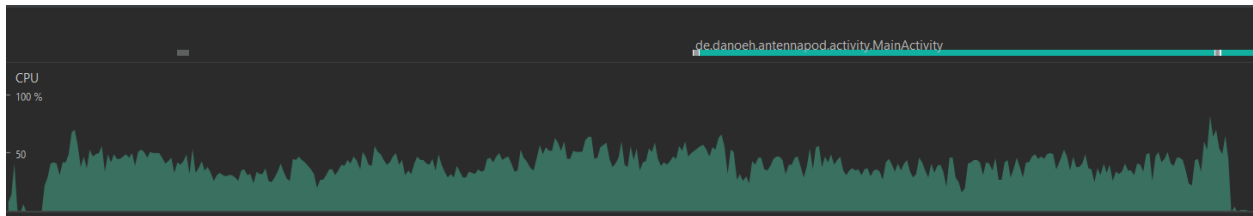
La consommation de CPU semble normale. Tout de suite au moment où la recherche commence (touche enter pesée), il y a une hausse de la consommation. Cette hausse correspond fort probablement à la construction d'une requête et l'envoi de cette dernière à un serveur. On observe ensuite une grande baisse d'utilisation. Cette baisse correspond sûrement à l'attente de l'application pour la réponse du serveur. Une fois la réponse reçue, le travail avec le CPU reprend. La consommation atteint encore une fois le sommet de 79% puis tourne autour de 40% jusqu'à l'affichage des résultats, après laquelle la consommation descend drastiquement. En ce qui concerne la mémoire, la recherche semble n'avoir eu aucun impact. C'est seulement vers la fin qu'on peut observer une légère hausse. Le système est passé de 184 à 222 MB de mémoire

utilisée. Cette hausse est cependant très brève et revient à la normale rapidement. Il ne semble donc pas y avoir de fuite de mémoire puisque tout ce qui est utilisé est récupéré par le *garbage collector*. Il ne semble pas avoir eu d'impact significatif sur la consommation d'énergie. Pour la détection de UI Jank, toutes les frames sont en bas de une seconde excepté une, qui prend 1.63 seconde. Il s'agit du moment où la vue passe de la page de recherche à la page de présentation des résultats. La majorité du temps est passé à calculer le layout, soit 1.32 seconde. L'affichage des thumbnails pour les différents podcasts se fait en moins de 150 ms chacun et l'affichage d'une lettre dans la barre de recherche prend en moyenne 35 ms. Ces deux durées ne sont pas problématiques pour l'expérience utilisateur.

## 3.2 Démarrage

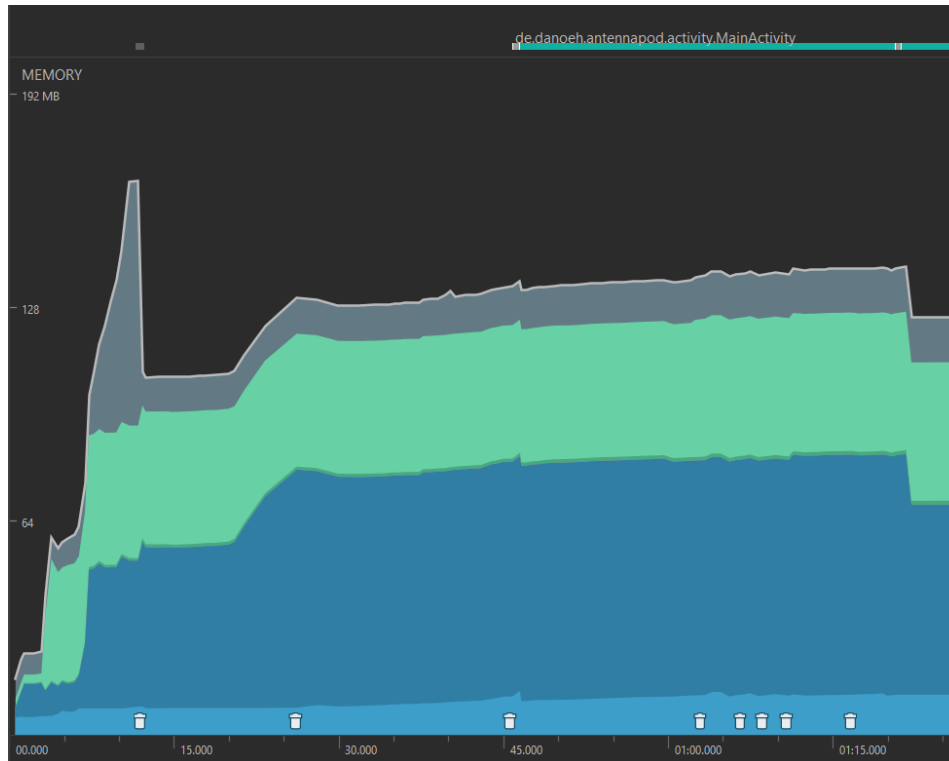
Pour ce test, l'émulateur est lancé sans AntennaPod. Par la suite, il est possible de démarrer AntennaPod et le profiler va s'attacher automatiquement à l'application pour fournir les données dès le démarrage.

### 3.2.1 Consommation CPU



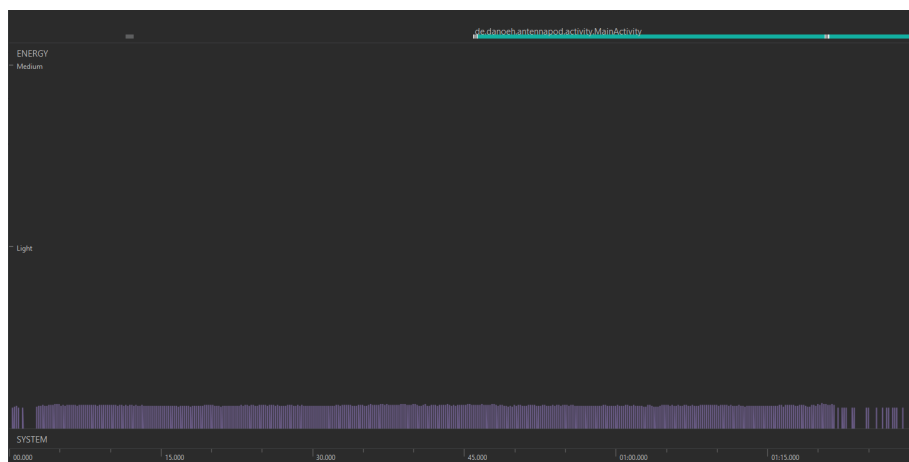
Sommet à 79% d'utilisation vers la fin. Reste majoritairement entre 30 et 60%.

### 3.2.2 Consommation Mémoire

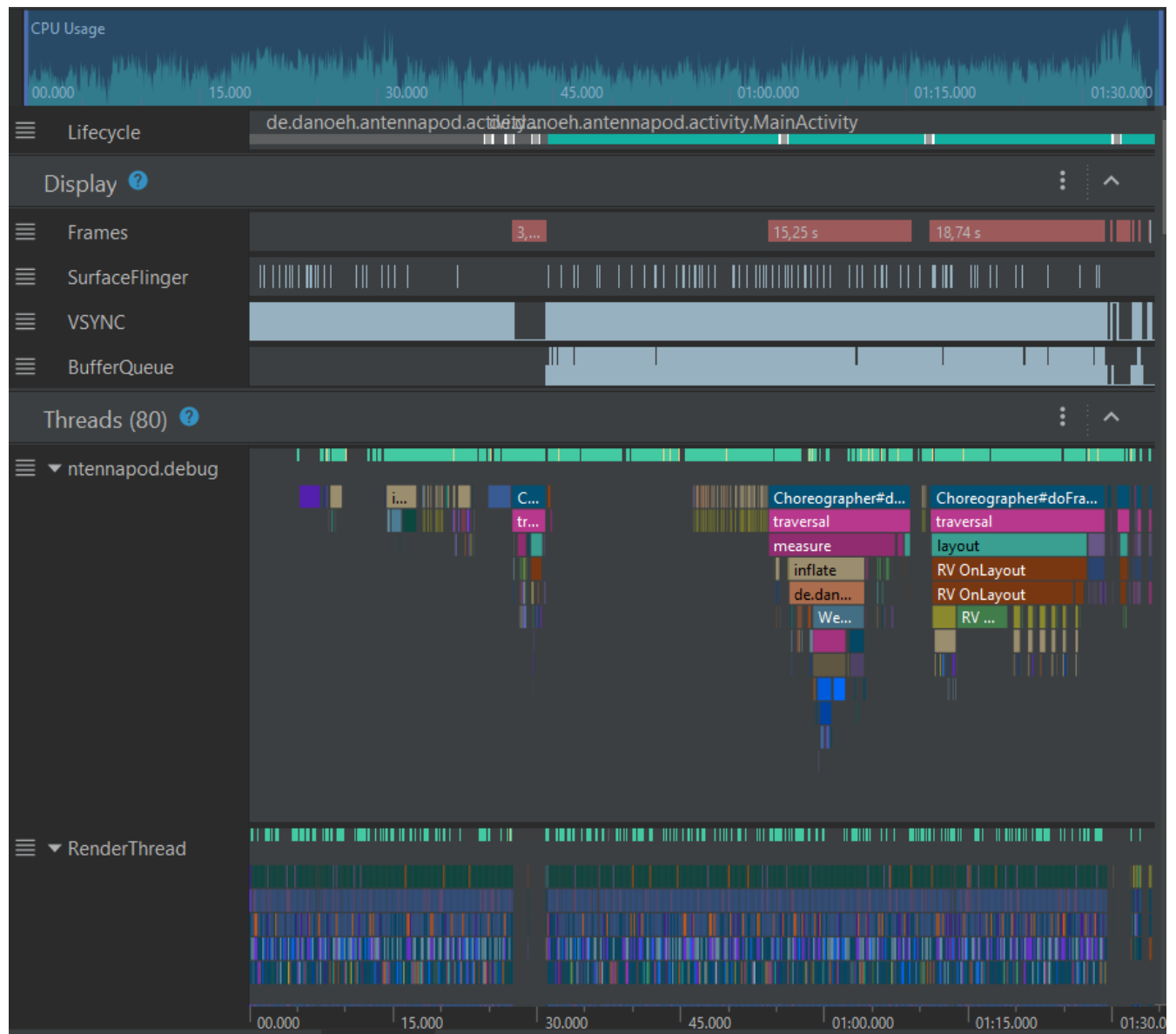


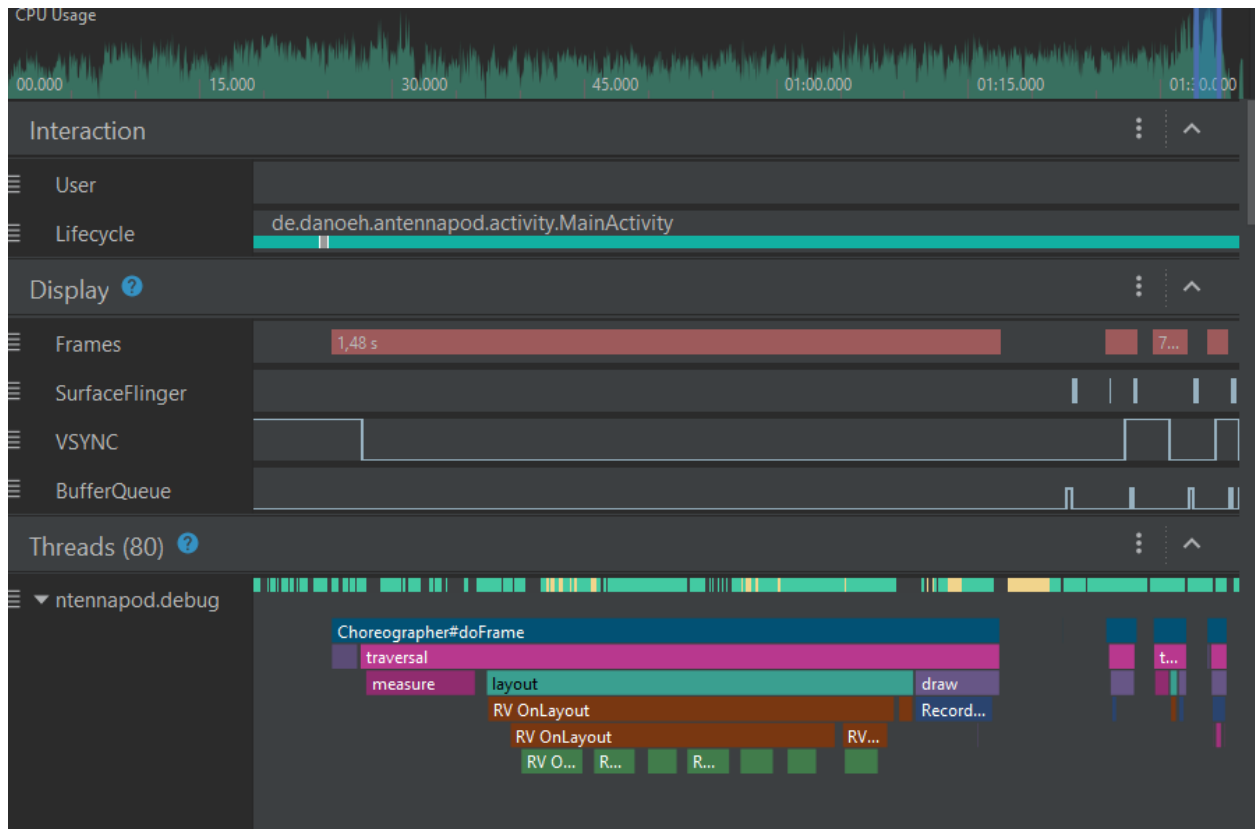
\*Sommet à 162 MB, après 30 secondes, tourne autour de 135 MB.

### 3.2.3 Consommation Énergétique



### 3.2.4 Détection de Ui Jank





### 3.2.5 Analyse

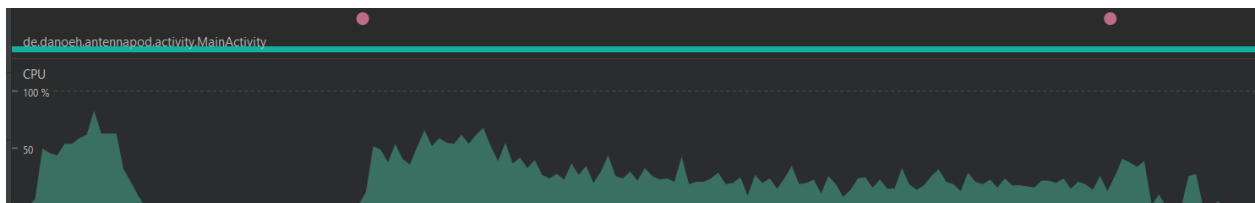
Le profiling de cette opération s'est effectué sur une plus grande période que la précédente. On peut bien observer que l'utilisation du CPU reste relativement stable tout au long du démarrage. On peut remarquer un extremum au début et à la fin du démarrage et que la moyenne de la première moitié du processus est légèrement plus élevée que la deuxième moitié. Une fois le démarrage terminé, l'utilisation du CPU descend drastiquement. Il est tout à fait normal que le démarrage d'une application demande un peu plus de CPU. On remarque quelque chose de plus intéressant pour la consommation de mémoire. Les premières secondes du démarrage demandent beaucoup de mémoire. Après la première passe du *garbage collector*, une bonne partie de la mémoire se fait libérer. Le reste de l'opération de démarrage demande environ 135 MB de mémoire. Si jamais il y avait des problèmes de consommation de mémoire lors du démarrage, il serait intéressant d'inspecter ce qui se passe dans les premières secondes. Toutefois, le plus haut niveau est inférieur au niveau moyen de l'activité précédente (Recherche de Podcast), il n'y a donc sûrement aucun problème. De plus,

la mémoire se fait libérer rapidement. Encore une fois, la consommation d'énergie reste très faible tout au long de l'activité. En ce qui concerne la détection de UI Jank, on peut observer qu'on passe 34 secondes pour dessiner deux frames. Cela est dû au fait que la majorité du travail se fait du côté du CPU pour initialiser l'application. Ces deux frames représentent la page de loading ainsi que la page d'accueil dans aucun contenu affiché. La deuxième capture d'écran se concentre sur l'affichage du contenu initial. La vue prend au total 1.48 seconde pour être mise à jour. On peut observer que la majorité du temps est consacré au calcul du layout, qui représente 944 ms versus 188 ms pour le draw. Simplifier le layout de la page d'accueil pourrait permettre d'accélérer l'affichage de celle-ci, mais ça reste une amélioration négligeable à comparer aux 34 secondes d'attente pour que le système soit initialisé.

### 3.3 Lecture d'un podcast

Ce test consiste à faire jouer un podcast déjà téléchargé. Il suffit de naviguer dans l'ongle épisode et de jouer n'importe quel podcast s'y trouvant. La partie intéressante du profiling commence dès que la touche play est pesé et se termine une fois que la touche pause est pesé. Sur les captures d'écran suivantes, les deux cercles rouges représentent ces événements respectivement. Le profiling s'est effectué sur une période d'environ 20 secondes.

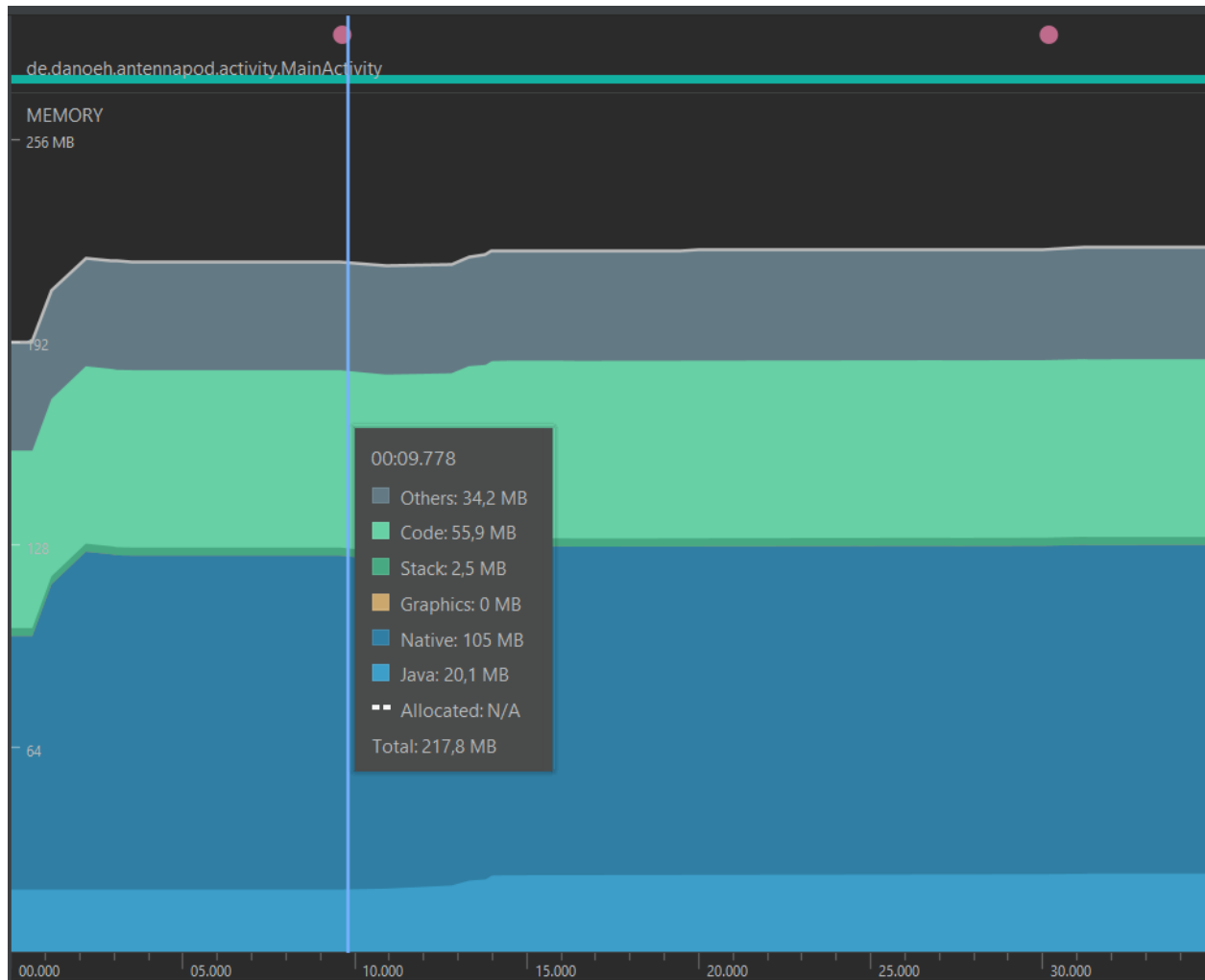
#### 3.3.1 Consommation CPU



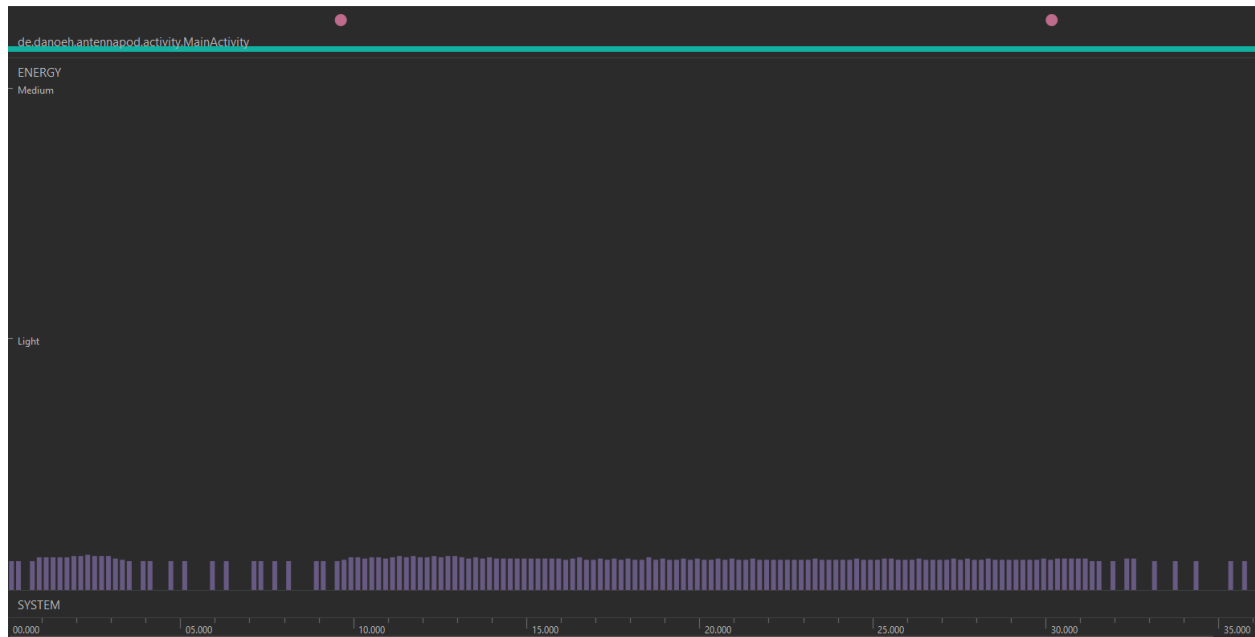
Sommet à 69% d'utilisation de CPU, tourne ensuite autour de 30% d'utilisation.



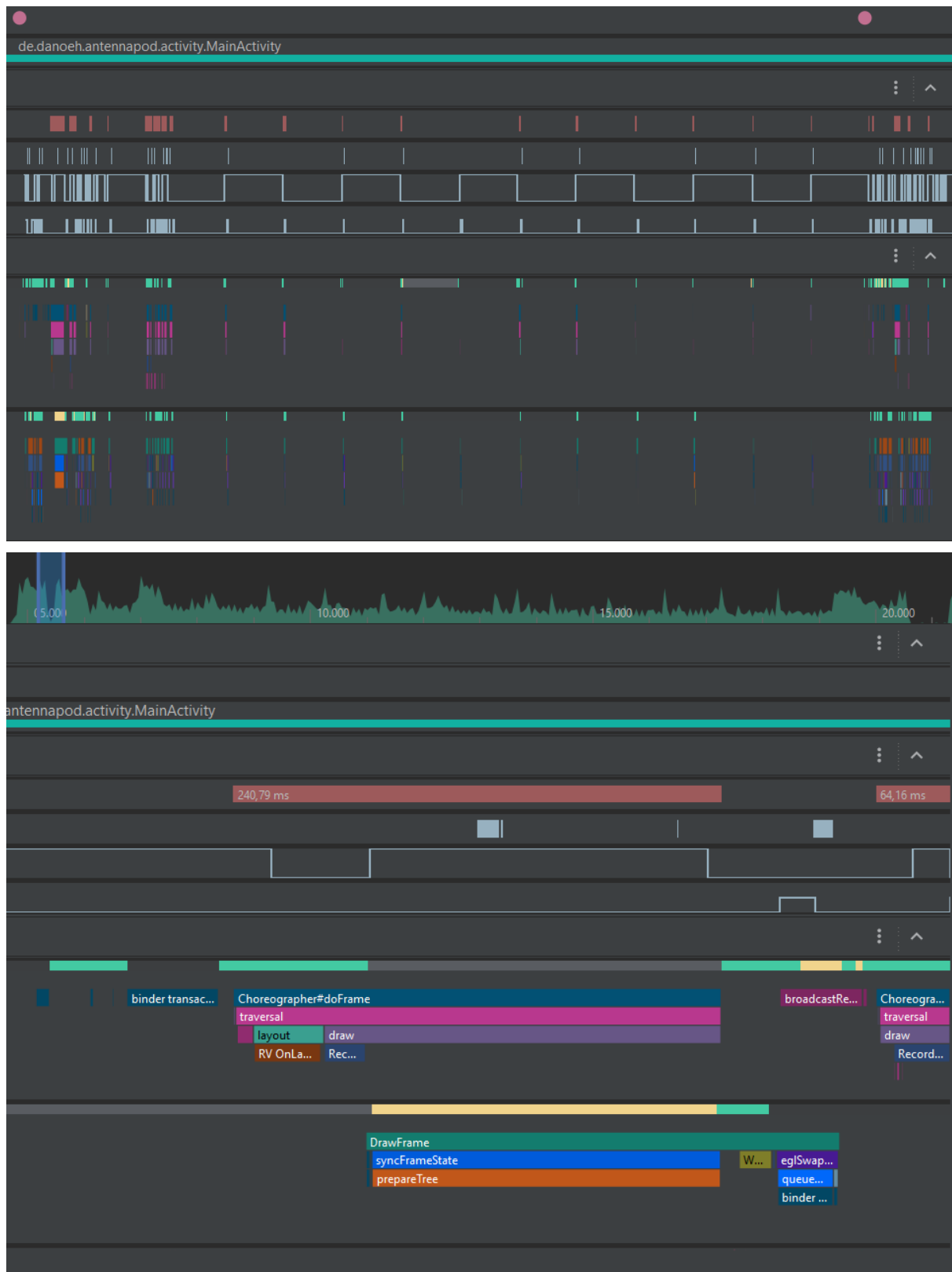
### 3.3.2 Consommation Mémoire



### 3.3.3 Consommation Énergétique



### 3.3.4 Détection de Ui Jank



### 3.3.5 Analyse

Pour la troisième activité, le profilage ne semble pas avoir fait ressortir de comportement alarmant. Pour l'utilisation du CPU, il y a un sommet à 69% au début, puis l'utilisation se stabilise autour de 30%. En ce qui concerne la mémoire, il n'y a pratiquement eu aucun changement entre avant et après le démarrage de la lecture du podcast. Il est même possible que la légère augmentation observée soit liée à une autre tâche qui se déroulait en arrière plan n'étant pas liée avec la lecture. Comme pour les deux autres activités, la consommation d'énergie est restée très faible tout au long du profilage. Comme l'écoute de podcast est la fonctionnalité d'AntennaPod, il est très rassurant de voir que ça ne cause pas d'augmentation de l'utilisation des ressources significatives. Pour la détection de UI Jank, on peut voir qu'il y a surtout des changements de frame au début et à la fin. Cela est causé par l'animation du bouton play/pause. Pendant la lecture, le seul changement qui survient est avec la barre de progression. Il y a une frame qui a pris plus de temps à dessiner que les autres (environ le double) avec 240,79 ms. Toutefois, même avec le détail de l'exécution des threads, il est difficile d'identifier la source de cette longueur.

## 4. Benchmarking

D'abord, le Benchmarking est un moyen de tester les performances de notre application AntennaPod. Nous exécutons des «benchmarks» pour analyser et déboguer les problèmes de performance tout en s'assurant de ne pas introduire des régressions dans les modifications récentes. Alors, Android propose deux librairies et approches de «benchmarking» afin d'analyser et tester différents types de situations dans l'application AntennaPod, soit le *Microbenchmark* et le *Macrobenchmark*.

Premièrement, le *Microbenchmark* est conçu pour mesurer le travail du CPU qui évalue les meilleures performances (*Just in Time*). Cette librairie analyse des performances de situations spécifiques à AntennaPod, mais non celles qui pourraient concerner des problèmes généraux du système. En effet, le *Microbenchmark* permet de traiter une

structure de données complexe ou de traiter un algorithme spécifique lourd qui se fait appeler plusieurs fois au cours de l'exécution de l'application. De plus, il est possible de mesurer certaines parties de l'interface utilisateur comme par exemple le temps nécessaire qu'il faut pour inflater un «layout». Par contre, le *Microbenchmark* ne nous dira pas si nous améliorons un goulot d'étranglement ou le temps de démarrage de l'application par exemple.[7]


En second lieu, le *Macrobenchmark* permet de mesurer les interactions de l'utilisateur telles que le démarrage et l'interaction avec l'interface utilisateur. À l'opposé du *Microbenchmark*, le *Macrobenchmark* offre un contrôle direct sur l'environnement et la performance à tester, c'est-à-dire qu'il nous permet de contrôler la compilation, le démarrage ainsi que l'arrêt de l'application. Cela a pour but de mesurer directement le démarrage ou le défilement réel de l'application au lieu de mesurer que des fonctions spécifiques avec tous les accès au disque en cache.[7]

Pour notre application AntennaPod, nous avons décidé de choisir la librairie *Macrobenchmark* car elle nous permet d'analyser des performances spécifiques à une activité que nous avons choisie. En effet, nous avons choisi d'analyser la lecture de podcasts. L'objectif du benchmark est de voir s'il y a des problèmes de performances lors de la lecture de podcasts dans AntennaPod. Effectivement, lorsque nous avons fait le *Profiling* dans la section d'auparavant, nous avons remarqué que l'opération coûteuse était la lecture de podcasts. Elle mérite alors d'être optimisée.

Tout d'abord, afin de faire la bonne configuration, nous avons suivi les étapes sur comment écrire un *Microbenchmark* selon Android Studio. En effet, nous avons commencé par ajouter la librairie *benchmark* au module *build.gradle* comme illustré dans la figure 4.1.1. Ensuite, nous avons désactivé le débogage dans le fichier *AndroidManifest.xml* et mis à jour l'élément *application* afin de forcer la désactivation temporaire du débogage. Nous le démontrons dans la figure 4.1.2. Une fois ceci terminé, nous avons synchronisé le tout pour faire la mise à jour de la librairie ainsi que la désactivation du débogage en appuyant sur le bouton *Sync now*. La prochaine étape

que nous avons fait, c'était d'ajouter une instance de la classe *Benchmarkrule* venant de la librairie que nous venons d'installer. *Benchmarkrule* sert à benchmarker du code dans un émulateur/appareil Android. Alors, nous avons ajouté cette instance dans notre fichier de test de la classe *PlaybackTest* illustré dans la figure 4.1.3. Nous avons également ajouté un test de benchmark dans le fichier de *PlaybackTest* afin de tester la lecture de podcasts à l'aide du *Microbenchmarking* illustré dans la figure 4.1.4. [8]

Cependant, notre test a échoué et nous ne sommes pas parvenus à analyser la lecture d'un podcast à l'aide du *Benchmark*. En effet, nous n'arrivons pas à faire passer le test et arriver à des résultats car nous obtenons des erreurs. L'une des erreurs est que la dépendance que nous avons installée n'est pas reconnue et que le Benchmark n'est pas reconnu. Nous pouvons observer cet échec dans la figure 4.1.5.



```
dependencies {
    androidTestImplementation("androidx.benchmark:benchmark-junit4:1.1.0-beta03")
    implementation project(":core")
    implementation project(":event")
    implementation project(':model')
    implementation project(':net:sync:gpoddernet')
    implementation project(':net:sync:model')
    implementation project(':parser:feed')
    implementation project(':playback:base')
    implementation project(':playback:cast')
    implementation project(':storage:database')
    implementation project(':ui:app-start-intent')
    implementation project(':ui:common')
    implementation project(':ui:i18n')
    implementation project(':ui:statistics')
```

#### 4.1.1: Ajout de la librairie Benchmark dans le *build.gradle*

```

<application
    android:debuggable="false"
    tools:ignore="HardcodedDebugMode"
    tools:replace="android:debuggable"
    android:name="de.danoeh.antennapod.PodcastApp"
    android:icon="@mipmap/ic_launcher"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:label="AntennaPod Debug"
    android:backupAgent=".core.backup.OpmlBackupAgent"
    android:restoreAnyVersion="true"
    android:theme="@style/Theme.AntennaPod.Splash"
    android:usesCleartextTraffic="true"
    android:supportsRtl="true"
    android:logo="@mipmap/ic_launcher"
    android:resizeableActivity="true"

```

4.1.2: Désactivation du débogage dans l'élément application du fichier *AndroidManifest.xml*

```

/**
 * Test cases for starting and ending playback from the MainActivity and AudioPlayerActivity.
 */
@LargeTest
@IgnoreOnCi
// @RunWith(Parameterized.class)
@RunWith(AndroidJUnit4.class)
public class PlaybackTest {
    @Rule
    public BenchmarkRule benchmarkRule = new BenchmarkRule();
    public ActivityTestRule<MainActivity> activityTestRule = new ActivityTestRule<>(MainActivity.class, initial

    @Parameterized.Parameter(value = 0)
    public String playerToUse;
    private UIUtils uiTestUtils;
    protected Context context;
    private PlaybackController controller;

    @Parameterized.Parameters(name = "{0}")
    public static Collection<Object[]> initParameters() {
        return Arrays.asList(new Object[][] { { "exoplayer" }, { "builtin" }, { "sonic" } });
    }
}

```

4.1.3: Ajout de l'instance *BenchmarkRule*

```

@Test
public void benchmarkSomeWork() throws Exception {
    final BenchmarkState state = benchmarkRule.getState();
    while (state.keepRunning()) {
        setContinuousPlaybackPreference(false);
        uiTestUtils.addLocalFeedData( downloadEpisodes: true);
        activityTestRule.launchActivity(new Intent());
        setupPlaybackController();
        playFromQueue( itemId: 0);
        Awaitility.await().atMost( timeout: 5, TimeUnit.SECONDS)
            .until(() -> controller.getStatus() == PlayerStatus.INITIALIZED);
    }
}

```

#### 4.1.4: Écriture du Test avec *Benchmark* afin de tester la lecture d'un podcast

```

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':app:dataBindingMergeDependencyArtifactsFreeDebugAndroidTest'.
> Could not resolve all files for configuration ':app:freeDebugAndroidTestRuntimeClasspath'.
   > Could not resolve androidx.benchmark:benchmark-junit4:1.1.0-beta03.
      Required by:
         project :app
      > Cannot find a version of 'androidx.benchmark:benchmark-junit4' that satisfies the version constraints:
         Dependency path 'AntennaPod:app:unspecified' --> 'androidx.benchmark:benchmark-junit4:1.1.0-beta03'
         Constraint path 'AntennaPod:app:unspecified' --> 'androidx.benchmark:benchmark-junit4:{strictly 1.0.0}' because of the following reason: freeDebug
      > Could not resolve androidx.benchmark:benchmark-junit4:{strictly 1.0.0}.
         Required by:
            project :app
         > Cannot find a version of 'androidx.benchmark:benchmark-junit4' that satisfies the version constraints:
            Dependency path 'AntennaPod:app:unspecified' --> 'androidx.benchmark:benchmark-junit4:1.1.0-beta03'
            Constraint path 'AntennaPod:app:unspecified' --> 'androidx.benchmark:benchmark-junit4:{strictly 1.0.0}' because of the following reason: freeDebug

* Try:
> Run with --stacktrace option to get the stack trace.
> Run with --info or --debug option to get more log output.
> Run with --scan to get full insights.

* Get more help at https://help.gradle.org

BUILD FAILED in 51s

```

#### 4.1.5: Échec du Test sur la lecture d'un podcast à l'aide du *Benchmark*



## 5. Bibliographie

[1] AntennaPod – *The Open Podcast Player*. (2022). En ligne. Disponible sur:

<https://antennapod.org/>

[2] About – *AntennaPod*. (2022). En ligne. Disponible sur:

<https://antennapod.org/about/>

[3] Contribute – *AntennaPod*. (2022). En ligne. Disponible sur:

[https://antennapod.org/contribute/?fbclid=IwAR2sqvQ3qDyQsmm7Lkc4GuGV5ZGccww\\_WESxTkD  
DfuSqK2bhzd4t6OyYBM8](https://antennapod.org/contribute/?fbclid=IwAR2sqvQ3qDyQsmm7Lkc4GuGV5ZGccww_WESxTkD<br/>DfuSqK2bhzd4t6OyYBM8)

[4] Normalisation des exigences. *Hydro Québec*. Disponible sur:

<https://docs.google.com/spreadsheets/d/0B4cNbsMq3e3bWkIBNEtPTkxMMUE/edit?resourcekey=0-drvZ>

[5] ISO25020\_Modele\_mesure\_base

[6] ISO25021\_Gabarit\_metriques

[7] Benchmark your app. *Android Developers*. (2022). En ligne. Disponible sur:

<https://developer.android.com/studio/profile/benchmarking-overview>

[8] Writing a Microbenchmark. *Android Developers*. (2022). En ligne. Disponible:

<https://developer.android.com/studio/profile/microbenchmark-write>