

PE Taak: Performance onderzoek

Daan Bergmans

Sander Elsen

Arno Bruynseels

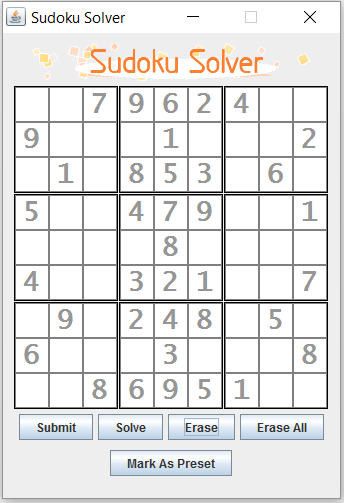
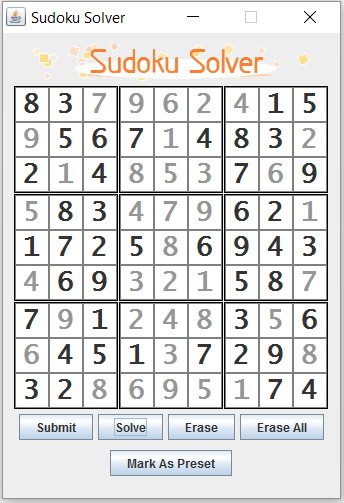
3AONC/D

2016-2017

# Sudokusolver-project op Github

Het project waar we voor gekozen hebben is een grafische sudokusolver die een sudoku kan oplossen en verbeteren.

Bron: <https://github.com/miga9/sudoku-solver-java>



# Wat is performance analysis?

## Inleiding

Indien we op zoek gaan naar wat ‘performance analysis’ betekent, waarvoor het gebruikt wordt en het nut ervan, stellen we vast dat dit concept gekend is en gebruikt wordt in talrijke domeinen[[1]](#footnote-1):

- Software & Algoritmes - Human resource management

- Sport - Verkoop

- Economie - Marketing

- Verkeer - ...

In deze analyse gaan we ons echter focussen op het domein van software en algoritmes, waaronder de programmeertaal Java.

## Beschrijving[[2]](#footnote-2)

Performantieanalyse[[3]](#footnote-3) is eigenlijk een discipline waarbij systematische observaties worden gedaan om prestaties en besluitvorming te verbeteren. Hierbij wordt alle informatie hoofdzakelijk verstrekt door middel van objectieve statistische (Data Analysis) en visuele feedback (Video Analysis).

Het doel van performantieanalyse is dus registreren wat de huidige situatie is en hoe deze zo optimaal mogelijk verbetert kan worden.

Hiervoor onderscheidt het performantieanalyse proces in het algemeen 3 verschillende stappen:

### 1 - Verzamelen van gegevens

Het verzamelen van gegevens is het proces waarbij data, over de prestaties van een uitvoerend programma, wordt verkregen.

Om dit te kunnen realiseren worden er observatietools gebruikt die gegevens kunnen meten, registreren en visueel voorstellen. Hiervoor zullen wij, in deze analyse, gebruik maken van Java VisualVM.

### 2 - Gegevens transformeren

Om data goed te kunnen bestuderen is het nodig dat de gegevens zodanig in een vorm worden omgezet dat die het mogelijk maken antwoorden te bieden. Nieuwe verzamelde gegevens voldoen zelden aan deze eisen en is het dus noodzakelijk om een transformatie te verwezenlijken.

In het software en algoritmes domein wordt men zelden met dit proces geconfronteerd omdat vrijwel alle observatietools deze taak op zich nemen.

Hier gaat Java VisualVM, de tool die we doorheen de analyse gaan gebruiken, ons helpen deze opgave volledig te automatiseren.

### 3 - Gegevens visualiseren

Nadat de data een transformatie ondergaan heeft, zou het een enorme bijdrage zijn voor de analyse dat alle data visueel voorgesteld kan worden. Gegevens visualiseren kan vele vormen aannemen. Een van de meest voorkomende vormen zijn: grafieken, diagrammen en statistieken. Het is dus belangrijk dat er goed nagedacht wordt over welke visualisatietechnieken er gebruikt zullen worden in het analyse proces.

Wederom zal Java VisualVM ons hierbij gaan assisteren. Java VisualVM ondersteunt het grafisch voorstellen van gegevens in de vorm van grafieken, diagrammen en absolute getallen.

# Waarom performantieanalyse?

Het realiseren van een performantieanalyse brengt vele voordelen met zich mee. Zoals eerder vermeldt is het een grote aanwinst om een goed inzicht te krijgen in de performantie van de software, waardoor toekomstige prestaties geoptimaliseerd kunnen worden en het nemen van beslissingen vergemakkelijkt wordt.

# Tools voor performantieanalyse

In het software en algoritme domein wordt er voornamelijk gebruik gemaakt van observatietools die de prestaties van lopende software kunnen gaan registeren, meten en visualiseren.

Bij het selecteren van welke tool men gaat gebruiken is het belangrijk om volgende aspecten te beschouwen:

* *Accuraatheid*: De tijdspanne waarin data wordt verzameld. Dit kan zowel realtime als periodiek zijn.
* *Flexibiliteit*: Hoe belangrijk is het voor de analyse dat de tool uitgebreid kan worden om bijkomende informatie te verzamelen?
* *Visualisatie*: Welke visualisatietechnieken moeten er allemaal ondersteund worden?

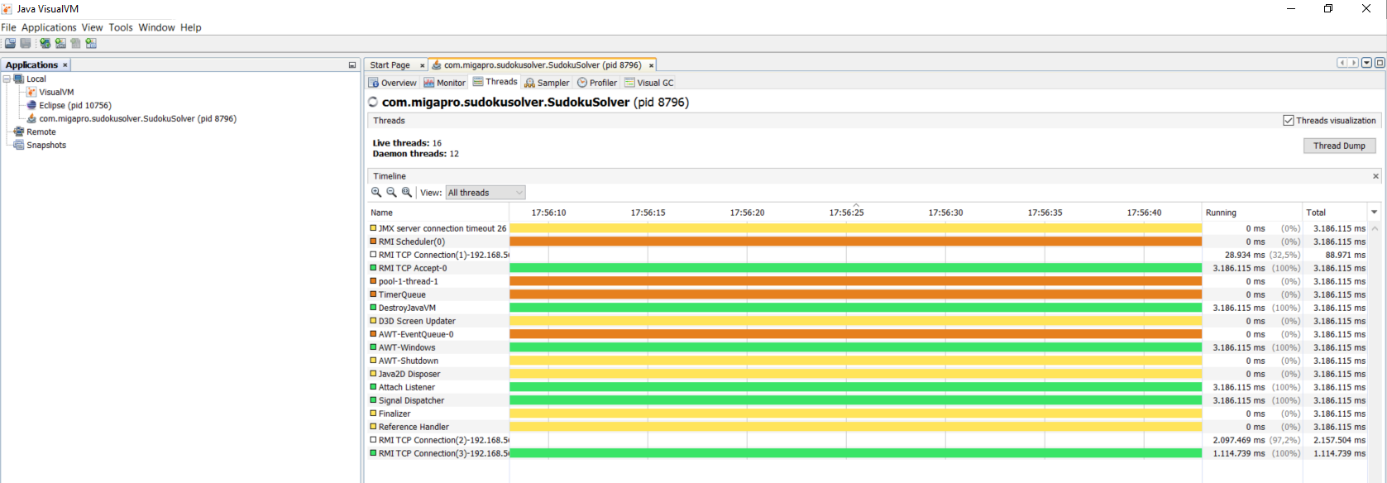
Voor Java zijn volgende observatie tools[[4]](#footnote-4) beschikbaar:

* NetBeans Profiler - JProfiler
* Java VisualVM - Java Performance Analysis Tool
* Jrockit Eclipse - Memory Analyzer
* Java Interactive Profiler - Profiler4J
* Jconsole - ...

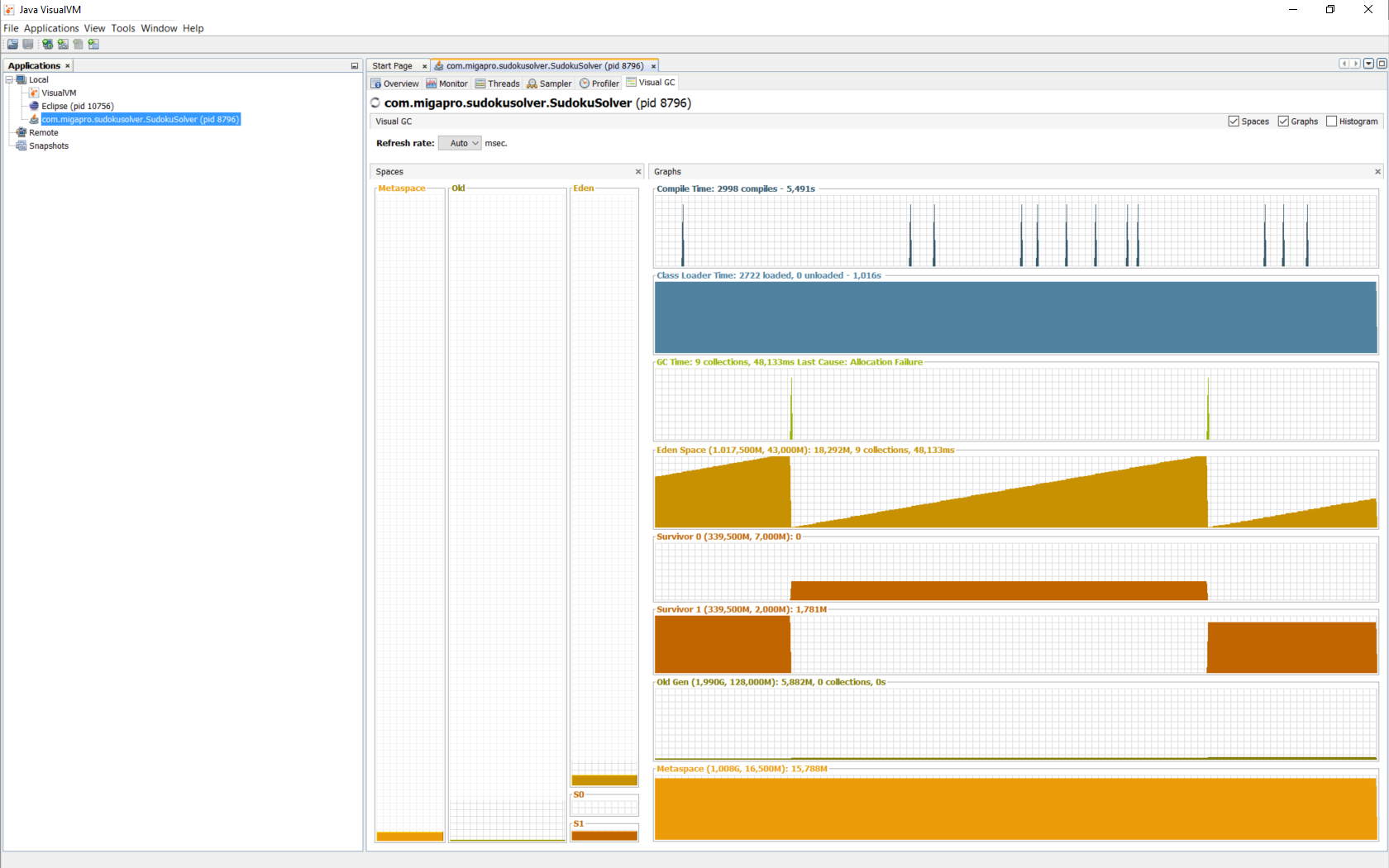
Voor de uitwerking van onze analyse hebben we gekozen voor Java VisualVM. Deze tool voldoet aan al onze belangen met bovenstaande aspecten in beraad genomen.

## Java VisualVM

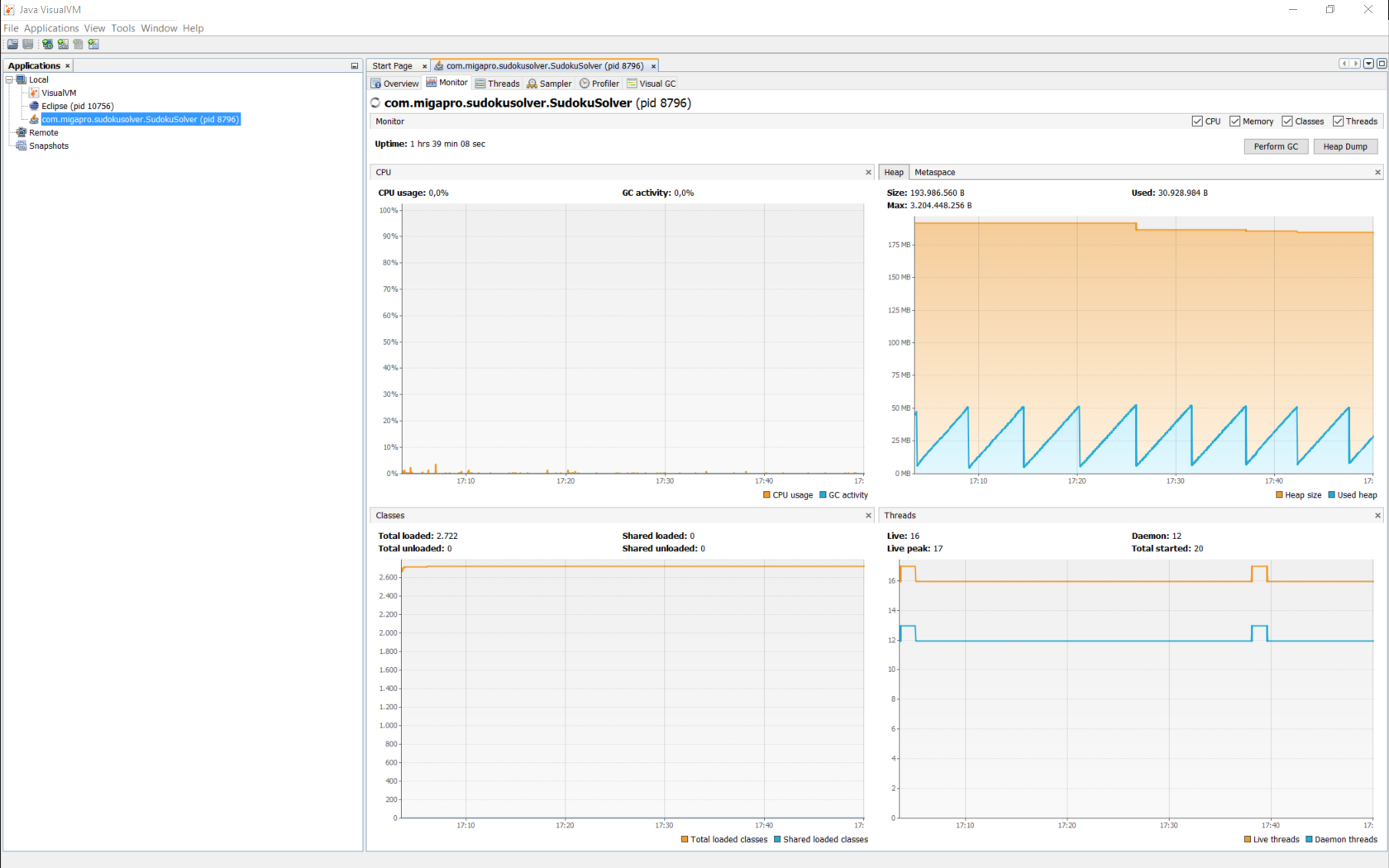
De Java VisualVM[[5]](#footnote-5) grafische user interface is een monitoring tool die voldoet aan de Java Management Extensions (JMX) specificatie. Java VisualVM maakt gebruik van het uitgebreide instrumentarium van de Java Virtual Machine (Java VM) om informatie te voorzien, over de prestaties en het gebruik van resources, van applicaties die draaien op het Java-platform.

Threads monitor

Garbagecollector monitor



Overview monitor



# Oplossingsmethode sudoku

De applicatie ondersteunt zowel het controleren van een ingegeven oplossing als het oplossen van een sudoku met een al dan niet ingegeven preset (start situatie).

Een algoritme beschrijven in een tekstuele vorm kan nogal omslachtig en onduidelijk overkomen. Hierdoor opteren wij ervoor het algoritme te beschrijven in de vorm van een stappenplan waarbij elke stap, dat het algoritme doorloopt, besproken wordt.

Bij het oplossen en controleren van de sudoku gaat het algoritme telkens starten bij de cel in de linkerbovenhoek om vervolgens van links naar rechts verder te werken.

## Controleren van een ingegeven oplossing (Submit button)

1. **Controleer de celwaarde**

*Uitleg***:** Controleer dat de huidige cel een waarde > 0 bevat.

* *Ja*: Stap 2
* *Neen*: Einde algoritme. Toon een bericht dat de suduko niet compleet is.

1. **Controleer de 3x3 box**

*Uitleg***:** Controleer dat de huidige celwaarde geen duplicaties heeft in de 3x3 box waar de cel zich in bevindt.

* *Ja*: Stap 3
* *Neen*: Einde algoritme. Toon een bericht dat de sudoku niet correct is.

1. **Controleer de rij en kolom**

*Uitleg***:** Controleer dat de huidige celwaarde geen duplicaties heeft in zijn rij of kolom. De methode die dit controleert sorteert de getallen in de rij en kolom op volgorde, waarna deze nagaat of de gesorteerde getallen tellen van 1 tot 9.

* *Ja*: Waarde van huidige cel is correct.
* *Neen*: Einde algoritme. Toon een bericht dat de sudoku niet correct is.

1. **Einde controle**

*Uitleg***:** Indien elke celwaarde gecontroleerd is en iedere cel heeft alle testen steeds doorstaan, is de ingegeven oplossing correct.

## Oplossen van een sudoku (Solve button)

1. **Controleer dat de sudoku vol is**

*Uitleg***:** Controleer dat elke cel een waarde > 0 bevat.

* *Ja*: Einde algoritme. Toon een bericht dat de suduko vol is.
* *Neen*: Stap 2

1. **Controleer dat de sudoku geldig is**

*Uitleg***:** Ga elke cel af en controleer voor iedere cel, dat een waarde > 0 bevat, of er geen duplicaties voorkomen van de huidige waarde in de 3x3 box, de rij en de kolom.

* *Ja*: Einde algoritme. Toon een bericht dat de sudoku niet geldig is.
* *Neen*: Maak de preset getallen onbewerkbaar en grijs van kleur. Ga naar stap 3.

1. **Los de sudoku op**

*Uitleg***:** Het algoritme voor het oplossen van een sudoku maakt gebruik van recursie (methode die zichzelf aanroept) en backtracking.

1. Controleer dat elke cel opgelost is

*Ja*: Het einde van de sudoku is bereikt. Einde algoritme, de sudoku is opgelost.

*Neen*: Stap 2

1. Controleer dat de cel een preset is

*Ja*: Ga naar volgende cel door zichtzelf aan te roepen met de volgende celwaarde.

*Neen*: Stap 3

1. Maak een lijst aan met cijfers aan van 1 t.e.m. 9 en schudt deze door elkaar.
2. Ga elk getal in deze lijst af en test of het huidige getal niet voorkomt in de 3x3 box, de rij en de kolom.

*Lukt*: Vul dit nummer in de cel in en ga naar de volgende cel door zichzelf aan te roepen met de volgende celwaarde.

*Faalt*: Stap 5

1. Indien er geen waarde in de lijst met nummers van 1-9 gevonden wordt dat klopt in de sudoku wordt de huidige celwaarde op 0 gezet. Vanwege recursie wordt er dan aan backtracking gedaan, wat wil zeggen dat er terug naar de vorige cel wordt genavigeerd en hiervoor een andere waarde probeert te vinden die ook klopt. Waarna het algoritme de sudoku opnieuw zal proberen oplossen met de nieuwe gevonden waarde. Dit proces kan teruggaan tot de begincel.

### Algoritme verwijzing

Wikipedia: <https://en.wikipedia.org/wiki/Sudoku_solving_algorithms>

Code My Road: <https://codemyroad.wordpress.com/2014/05/01/solving-sudoku-by-backtracking/>

Ugent: <http://lib.ugent.be/fulltxt/RUG01/002/224/798/RUG01-002224798_2015_0001_AC.pdf>

# Performantieverbeteringen[[6]](#footnote-6)

## 1 – Shuffle-methode

### Beschrijving

In de broncode wordt er voor iedere cel telkens een lijst van unieke getallen van 1 t.e.m. 9 aangemaakt en gebruikt om de sudoku op te lossen. Hierbij gaat het algoritme voor elke cel telkens de lijst af totdat er een getal gevonden is dat klopt in de sudoku.

Hoewel deze lijsten, in de broncode, willekeurig door elkaar worden geschud voordat het algoritme deze gaat gebruiken, kan het juiste getal telkens zeer snel gevonden worden (steeds één van de eerste getallen in de lijst) of kan het net zeer lang duren (steeds één van de laatste getallen in de lijst).

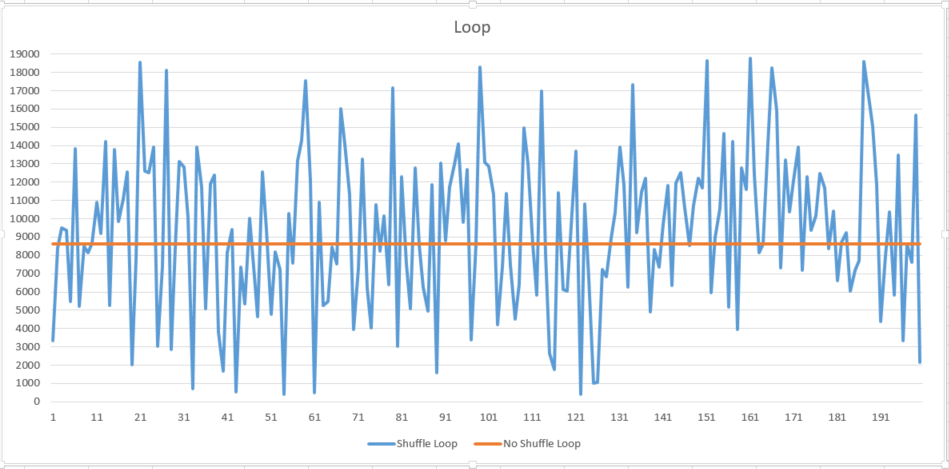
Als dezelfde sudoku dus meermaals opgelost wordt met lijsten van getallen die steeds een willekeurige volgorde hebben, zal het algoritme altijd een verschillend aantal keer de lijst doorlopen.

Omdat het algoritme functioneert met de backtracking methode, zal eveneens het aantal keer dat er aan backtracking wordt gedaan verschillen.

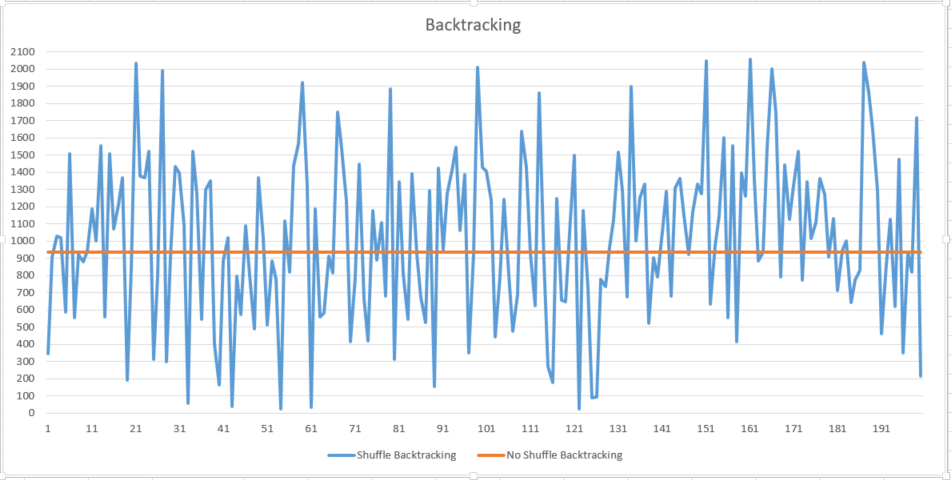
Welk effect zou het hebben op het algoritme indien de lijst niet willekeurig door elkaar wordt geschud?

### Meting

Na een sudoku 200 maal op te lossen, met en zonder de shuffle-methode, hebben we het aantal keer dat de lijst doorlopen wordt geregistreerd samen met het aantal maal dat er aan backtracking werd gedaan.

Loop

Backtracking

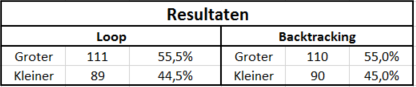


Deze resultaten hebben we in een lijngrafiek gegoten om beter inzicht te krijgen in de verzamelde resultaten.

Zoals we merken is er een constante lijn bij beide grafieken indien we geen shuffle-methode toepassen. Dit is te wijten aan de statische volgorde van de getallen in de lijst. Het algoritme zal steeds opnieuw op hetzelfde ogenblik dezelfde getallen gaan gebruiken waardoor we altijd hetzelfde resultaat bekomen.

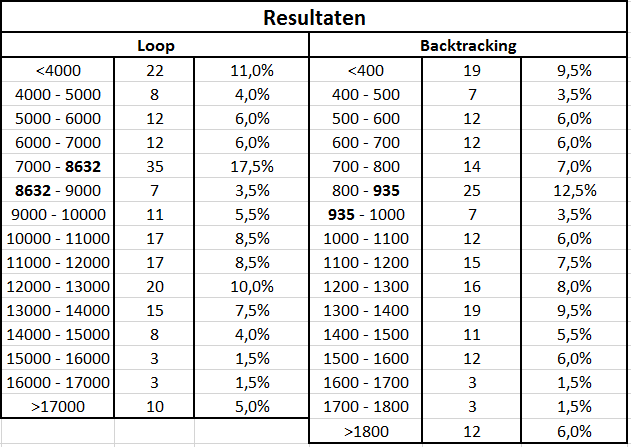
Wanneer we kijken naar de data in de grafieken die gerelateerd zijn met de shuffle-methode, nemen we meteen waar dat er zeer uiteenlopende resultaten zijn telkens de sudoku werd opgelost.

Als we gegevens nader gaan bekijken en gaan vergelijken met de waarde van de non-shuffle methode, bekomen we onderstaand resultaat.



We constateren dat in 5% van de 200 oplossingen, de shuffle-methode meer iteraties maakt zowel bij de loop als bij de backtracking.

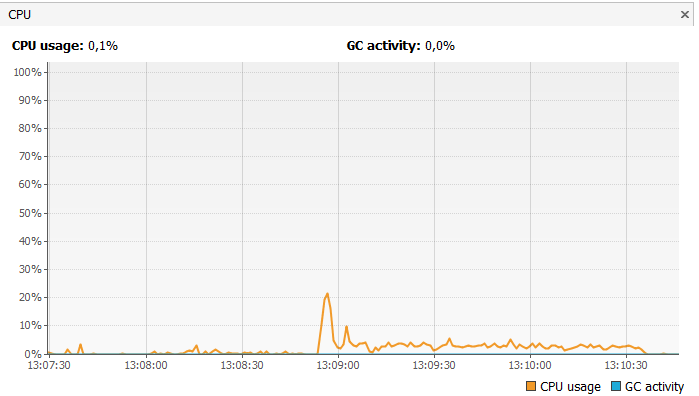
Indien we dan naar absolute aantallen van iteraties gaan kijken, stellen we vast dat het grootste aantal iteraties, die een winstmarge hebben, zowel bij de loop als bij backtracking maar net onder de constante liggen. Hoewel dit een winst aan performantie blijkt te zijn, is dit in tegenstelling met de iteraties die hoger liggen dan de constante. Het grootste aandeel hiervan ligt dan veel hoger boven de constante waarde.



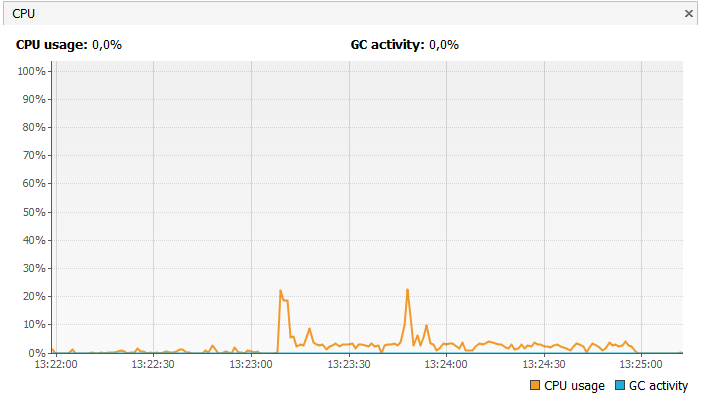
We kunnen bovenstaande resultaten nu ook gaan mappen op het CPU gebruik om daadwerkelijke performantieverbeteringen waar te nemen.

Met Java VisualVM hebben we het CPU gebruik geregistreerd bij zowel het gebruik met als zonder de shuffle-methode.

Geen shuffle-methode



Met shuffle-methode



Uit bovenstaande metingen blijkt er inderdaad een performantieverschil merkbaar te zijn. Zonder het gebruik van de shuffle-methode volgt het CPU gebruik een vrij constante lijn, wat overeenstemt met het aantal iteraties.

Bij het CPU gebruik met shuffle-methode zien we echter een paar uitschieters. Dit is te verklaren door de willekeurige volgorde van de lijst die op dat moment toevallig een zeer slechte schikking had waardoor er harder moest gewerkt worden om het correcte getal te vinden.

### Vaststelling

We kunnen dus besluiten dat er een deel van de tijd met de shuffle-methode een winst aan performantie gegenereerd wordt. Echter wordt er bij 5% van alle oplossingen een groter aantal iteraties uitgevoerd. Wanneer er meer iteraties worden overlopen, ligt dit aantal gemiddeld hoger dan het aantal iteraties die er gewonnen worden bij een winstmarge.

Dit resulteert over het algemeen in een hoger CPU gebruik met kans op één of vele uitschieters die intensief CPU gebruik vergen.

Het is dus aan te raden om geen shuffle methode te gebruiken om de zekerheid te garanderen over een constante CPU gebruik zonder kans op uitschieters.

## 2 – Primitieve datatypes en wrapperklassen

### Beschrijving

De Java virtuele machine werkt met de heap en de stack. Dit zijn 2 geheugenopslagplaatsen die een verschillende werking hebben wat betreft geheugenallocatie en dataopslag.

De heap

Op een heap kunnen data-elementen (vaak objecten) worden opgeslagen en verwijderd. Dit verwijderen gebeurt door de garbage collector als het geheugen van de heap vol is of indien het data-element niet meer nodig is en afgebroken wordt.

Voor het alloceren van geheugenblokken maakt de heap gebruikt van dynamische allocatie. Dit wil zeggen dat een geheugenblok zowel gealloceerd en vrij gemaakt kan worden op elk gegeven moment.

Een applicatie bevat meestal 1 heap.

De stack

De stack is een gereserveerde geheugenruimte die verdeeld is in frames. Er wordt telkens een stackframe aangemaakt wanneer er een methode wordt uitgevoerd. Elk stackframe bevat de referentie van lokale variabelen (primitieve datatypes) en andere relevante data. Een stackframe wordt terug vrij gemaakt als de methode van het gealloceerde frame afgelopen is.

De stack wordt altijd gereserveerd in een LIFO (last in first out) volgorde; de laatst gereserveerde blok zal altijd het volgende blok zijn dat vrij gemaakt wordt.

Elke thread in een applicatie krijgt een stack toegewezen.

Besluit

Door de verschillende werking van geheugenallocatie en dataopslag zal de stack dus sneller werken dan de heap.

Indien we bovenstaande aspecten in beraad gaan nemen, kunnen we hiermee rekening gaan houden in de java code. In java bestaan er wrapperklassen die afgeleid zijn van de primitieve datatypes. Deze wrapperklassen zijn echter objecten die op de heap worden opgeslagen. Dit in tegenstelling tot de primitieve datatypes die worden bijgehouden op de stack.

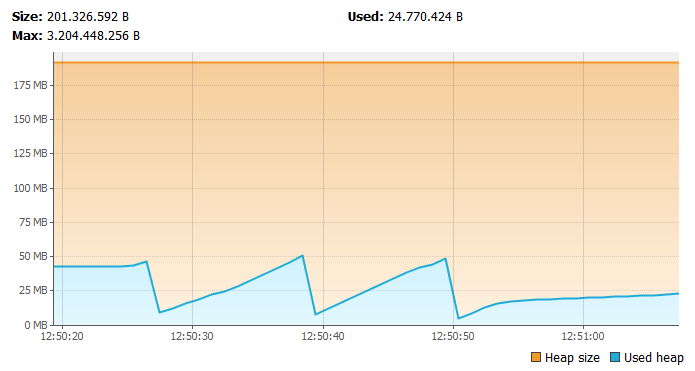
Het zou dus efficiënter zijn in termen van geheugengebruik om primitieve datatypes te gebruiken i.p.v. wrapperklassen.

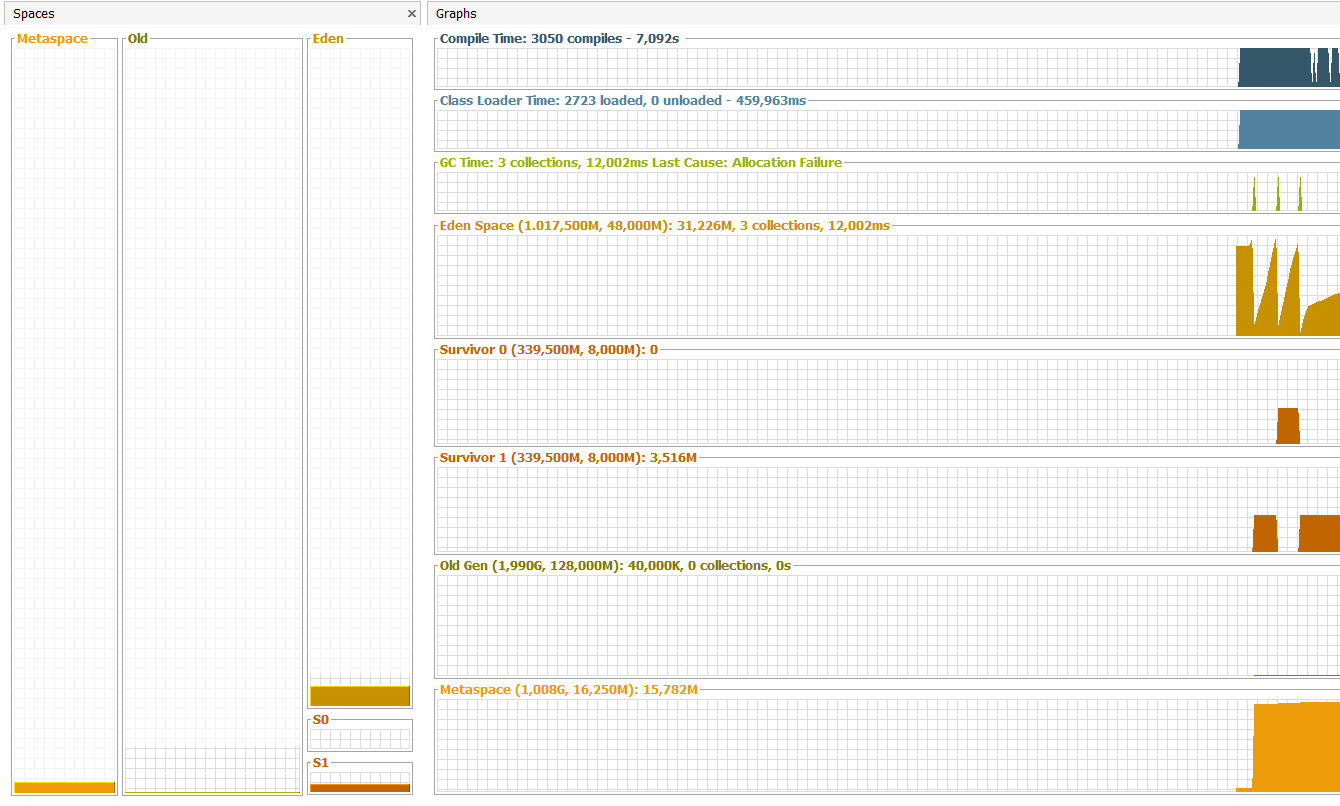
### Meting

We hebben de sudoku 50 maal opgelost met zowel wrapperklassen als primitieve datatypes.

Bij het uitvoeren van deze metingen hebben we gebruik gemaakt van de Java VisualVM heap monitor om het geheugengebruik van de heap te visualiseren.

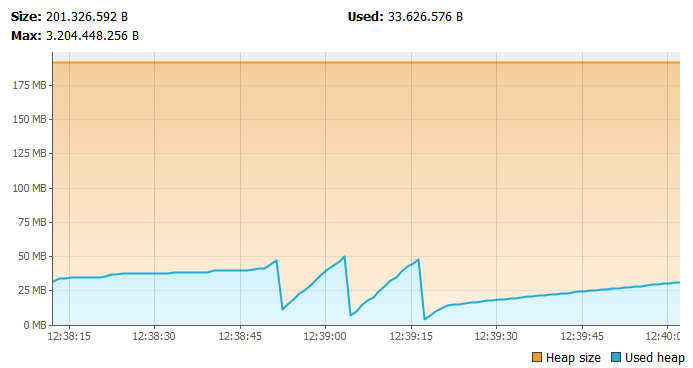
Wrapperklassen

Heap monitor

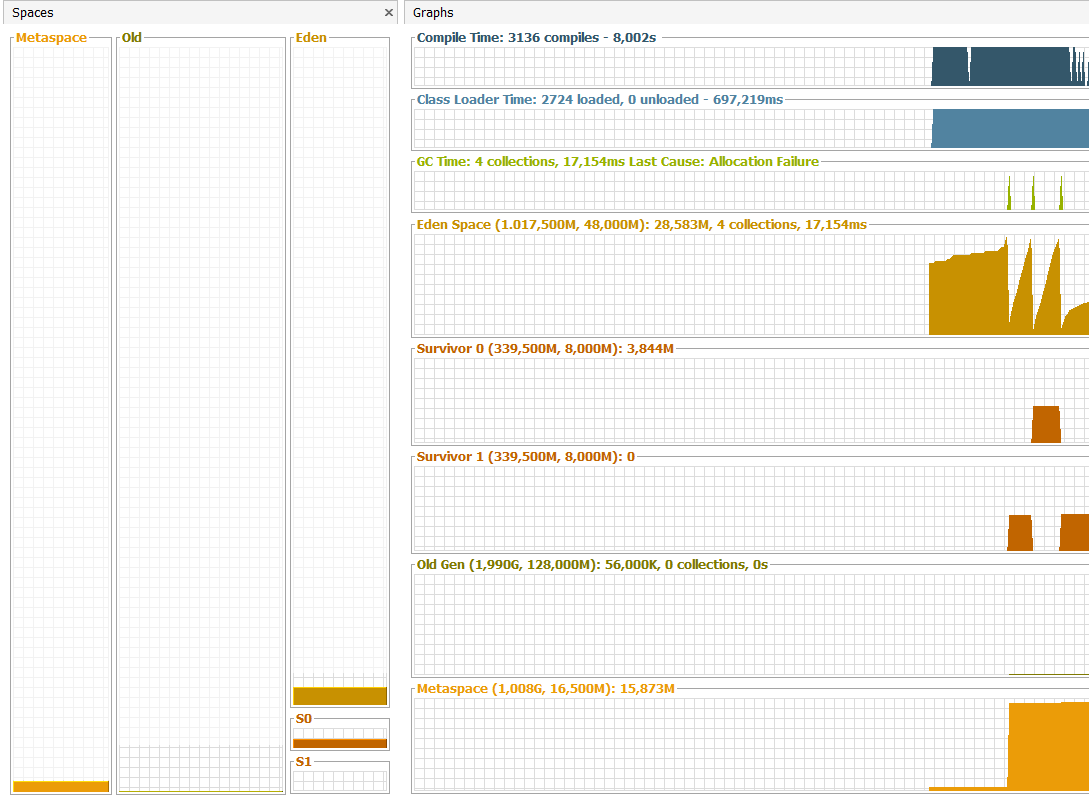
GC monitor

Primitieve datatypes

Heap monitor



GC monitor



De grafieken en statistieken op de heap –en GC monitor zien er vrijwel hetzelfde uit bij beide resultaten. Samen hebben ze opwaartse bewegingen, wat het ophopen en creëren van object voorstelt, en dalingen die het opruimen van de niet meer overlevende objecten in kaart brengt.

Hoewel er grafisch geen verschil zichtbaar is omwille van het beperkt gebruik aan wrapperklassen, is het toch een zeer goede ‘beste practice’ om primitieve datatypes te gebruiken i.p.v. wrapperklassen.

Deze ‘best practice’ hanteren zal ervoor zorgen dat als de applicatie uitgebreid wordt er een winst aan performantie en geheugengebruik zal zijn. In de sudokusolver applicatie resulteert dit dus in een verbetering op gebeid van schaalbaarheid.

### Vastelling

In de sudokusolver applicatie zal er geen tot een erg klein performantieverschil merkbaar zijn door het weinig gebruik van wrapperklassen. Anderzijds door de ‘beste practice’ toe te passen en primitieve datatypes te gebruiken i.p.v. wrapperklassen waar het kan, wordt de applicatie automatisch meer schaalbaar en zal er een winst aan performantie zijn indien deze uitgebreid wordt.

## 3 – Callable solve-method

### Beschrijving

Java is een programmeertaal dat multithreading ondersteunt.

Multithreading is een concept dat het uitvoeren van verschillende taken op hetzelfde moment mogelijk maakt. De taken worden op een verschillende thread geplaatst en simultaan uitgevoerd.

Indien een applicatie gebruikt maakt van een GUI (grafische user interface), is het gebruik van multithreading zeker aan te raden. De taakverdeling zal er dan voor zorgen dat de GUI niet zal bevriezen als er taken op de achtergrond uitgevoerd worden.

Het simultaan uitvoeren van taken zal dus minder tijd in beslag nemen en is dan ook zeer aantrekelijk op gebied van performantie.

In de broncode van de sudoku applicatie wordt er geen multithreading toegepast en bijgevolg zal elke taak die aangesproken wordt zich na elkaar uitvoeren.

Om deze reden hebben we de zwaarste taak die de sudokusolver moet uitvoeren in een aparte thread gezet. Hier kunnen de berekeningen dan simultaan uitgevoerd worden zonder dat de GUI bevriest.

### Meting

Hoewel de taak nu in een aparte thread zit is het niet echt mogelijk om een observatie uit te voeren waarin we de performantieverbetering gaan meten. Dit is omwille van de recursieve berekening die wordt gebruikt bij het oplossen van een sudoku.

Een recursieve berekening kan niet opgesplitst worden in meerdere threads omdat de methode altijd de waarde nodig heeft van de vorige aanroep.

Een vaststelling zou wel mogelijk zijn mocht er een taak simultaan uitgevoerd moeten worden met de solve-methode. Dit is echter niet het geval in de broncode en is een meting uitvoeren dus overbodig.

De grootste winst aan performantie, bij het gebruik van de aangemaakte thread, is de responsiviteit van de GUI. De grafisch user interface bevindt zich nu op een andere thread dan de berekening en zal dus niet meer bevriezen bij het uitvoeren van de zware recursieve berekening.

### Vastelling

Het gebruik van multithreading is noodzakelijk als men een perfomante applicatie wil opbouwen. Hierdoor zullen meerdere taken simultaan uitgevoerd kunnen worden waardoor er een grote tijdswinst zal zijn.

Hoewel de sudokusolver niet sneller zal werken door de geïntegreerde mulithreading, zal de taakverdeling er toch voor gaan zorgen dat de GUI responsief blijft bij het uitvoeren van de solve-methode.

## 4 – GenerateRandomNumbers-method

### Beschrijving

Zoals als eerder vermeld gebruikt de solve-method een lijst met getallen van 1 t.e.m. 9 om via het backtracking algoritme een sudoku op te lossen.

In de originele broncode wordt deze lijst telkens opnieuw aangemaakt en geïnstantieerd wanneer er een nieuwe aanroep wordt gedaan op de recursieve methode. Dit aanmaken gebeurt in de generateRandomNumbers-methode waarin origineel ook de shuffle-methode wordt gebruikt om de nieuw aangemaakte lijst door elkaar te schudden.

We hebben in voorgaande verbetering (1) echter al vastgesteld dat de shuffle-methode niet performant is. De enige taak die de generateRandomNumbers-methode nu nog rest is het aanmaken van nieuwe lijsten met getallen van 1-9 in vaste volgorde.

Dit is echter onnodig werk dat niet hoeft geleverd te worden aangezien dezelfde lijst opnieuw gebruikt wordt bij elke iteratie van de recursieve methode.

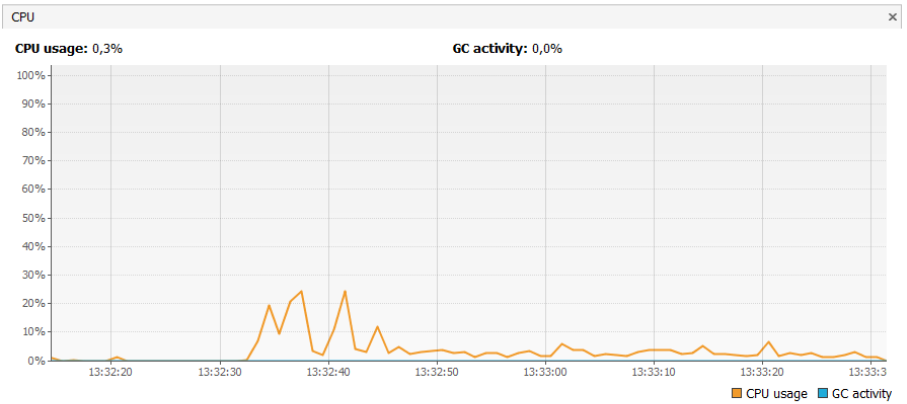
Het zou dus veel voordeliger zijn om eenmaal een statische lijst aan te maken en deze te gebruiken doorheen de hele oplossingsmethode i.p.v. steeds dezelfde lijst opnieuw te genereren.

### Meting

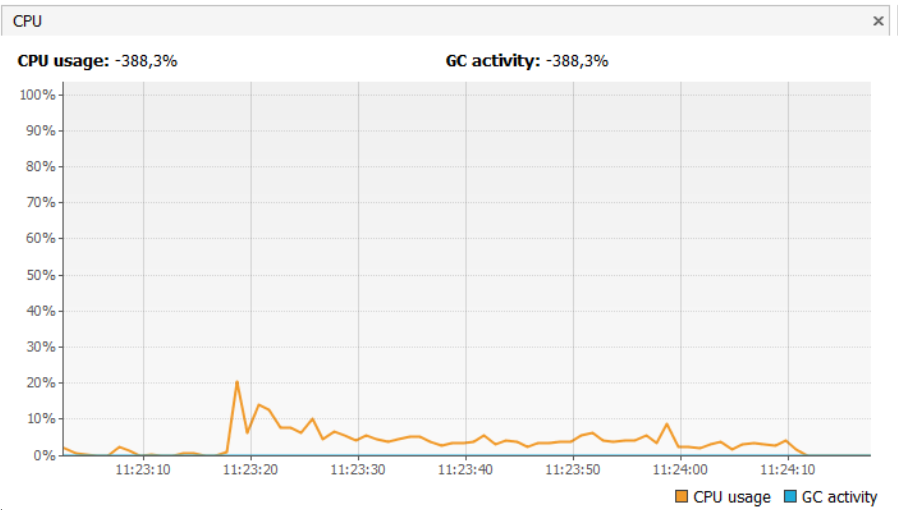
Om eventuele verbeteringen waar te nemen hebben we de sudoku 100 maal opgelost met zowel een statische lijst als de generateRandomNumbers-methode die steeds een nieuwe lijst produceert.

Voor het monitoren hebben wederom beroep gedaan op de faciliteiten van Java VisualVM om het CPU-gebruik te registreren.

Statische lijst



GenerateRandomNumbers-method



Zoals men ziet ligt het gemiddelde CPU-gebruik over de totale uitvoering hoger bij de generateRandomNumbers-methode. Dit is ook een logisch gevolg van het extra overbodige werk dat de ‘central processing unit’ moet uitvoeren.

### Vastelling

Door het achterhalen van de voorgaande shuffle-methode hebben we nu ook de hele generateRandomNumbers-methode kunnen weglaten aangezien deze enkel nog overbodig werk uitvoerde.

Wegens deze gehele performantieanalyse hebben we het gemiddelde CPU-gebruik bij het oplossen van een sudoku naar benden kunnen brengen.

# Reflectie

## Arno Bruynseels

Voor de aanvang van deze opdracht was ik mezelf er niet van bewust dat performantieanalyse een activiteit is die in vele verschillende domeinen is gekend. Het verbaasde me dan ook dat er zo frequent aan performantieanalyse gedaan wordt.

Nu realiseer ik me hoe belangrijk en welke grote impact performantieanalyse kan hebben voor een applicatie, algoritme of om het even welke zaak waarnaar er onderzoek wordt gedaan.

Door het uitwerken van de taak heb ik mezelf kunnen verdiepen in heel het verloop van wat er zich achter de code afspeelt. Dit vond ik een unieke en aangename ervaring omdat dit niet elke dag op school wordt aangeleerd.

Tijdens de opdracht hebben we ook kennis gemaakt met de verschillende tools die beschikbaar zijn om het proces van performantieanalyse te ondersteunen. Ik vond het zeer leerrijk om al deze tools met elkaar te vergelijken en te kijken wat de voordelen of nadelen zijn van een bepaalde tool t.o.v. van een andere.

Nu ik weet wat performantieanalyse inhoud, wat de impact ervan kan zijn en de voordelen ken die het aanbiedt, zal ik in de toekomst tijdens het coderen deze zaken altijd in het achterhoofd houden waardoor ik kritischer zal kijken naar wat ik heb geschreven heb.

## Sander

Ik had er nog nooit bij stil gestaan hoe groot de impact van kleine foutjes kunnen zijn op een programma.

Ik dacht dat als alle code ongeveer juist was dat het wel snel genoeg zou zijn, en als het moest dan voegden we maar wat threads toe hier en daar.

Ik heb nu leren inzien dat dit dus helemaal niet het geval is, en dat er echt wel veel tijd in kruipt om een programma optimaal te doen functioneren.

Ook heb ik nu een visuele weergave gezien van hoe de heap en stack bewegen tijdens het werken van een programma en wat er allemaal achter de schermen gebeurd in een Java programma.

Elke tool heeft ook zijn eigen voor- en nadelen wat een heel ander beeld geeft op de performantie.

Door veel op te zoeken voor dit project heb ik veel geleerd over best practices in java en waarom je sommige dingen beter wel of niet doet.

Ik ga me hier zeker nog wat verder in verdiepen.

## Daan

Ik heb altijd al het gevoel gehad dat onze (grote) projecten niet altijd even snel werkten, en het STUFV-project van vorig jaar had het nog meer bevestigd (bvb ophalen van alle gemeentenamen). Daarom ben ik blij dat we ook een project gehad hebben omtrent performance analyse aangezien wij hier nog nooit echt iets over gezien hebben in de lessen (buiten threading). Door actief op te zoeken blijven deze minimale zaken ons ook bij.

Met de verschillende tools (en achterliggende lessen) hebben we kunnen ontdekken wat de performantie van een programma wel en niet kan beïnvloeden, zaken waar we geen gedacht van hadden en die moeilijk zijn om op te merken, maar wel een grote impact kunnen hebben.

Door deze opdracht zal ons sneller bijblijven wat we mogen en niet mogen doen. Want uiteindelijk wat voor de klant telt is dat het programma doet wat hij vraagt en dat ie het snel doet.

# Bronnen

(A) Verschillende domeinen voor performantieanalyse:

* <http://www.eis2win.co.uk/expertise/performance-analysis/>
* <http://www.economicsonline.co.uk/Managing_the_economy/Measuring_performance.html>
* <http://www.smtware.com/services/traffic-analysis>
* <http://www.businessdictionary.com/definition/performance-analysis.html>
* <https://en.wikipedia.org/wiki/Profiling_(computer_programming)>
* <http://www.sportni.net/performance/sports-institute-northern-ireland/performance-science/performance-analysis/>

(B) Definitie performantieanalyse:

* <http://www.eis2win.co.uk/expertise/performance-analysis/>

(C) Beschrijving performantieanalyse:

* <https://www.mcs.anl.gov/~itf/dbpp/text/node107.html>

(D) Observatietools:

* <https://blog.idrsolutions.com/2014/06/java-performance-tuning-tools/>

(F) Java VisualVM:

* <http://www.oracle.com/technetwork/articles/java/javamanagement-140525.html>
* <https://visualvm.github.io/features.html>

(G) Performantieverbeteringen:

* <http://www.appperfect.com/support/java-coding-rules/optimization.html>
* <https://blog.jooq.org/2015/02/05/top-10-easy-performance-optimisations-in-java/>

1. (A) Verschillende domeinen voor performantieanalyse [↑](#footnote-ref-1)
2. (C) Beschrijving performantieanalyse [↑](#footnote-ref-2)
3. (B) Definitie performantieanalyse [↑](#footnote-ref-3)
4. (D) Observatietools [↑](#footnote-ref-4)
5. (F) Java VisualVM: [↑](#footnote-ref-5)
6. (G) Performantieverbeteringen [↑](#footnote-ref-6)