Performance Analysis Programming Expert Project : Sudoku Solver

# Team

Groepsleden: Bram Van Vleymen, Joran Claessens

Klas: 3AOND

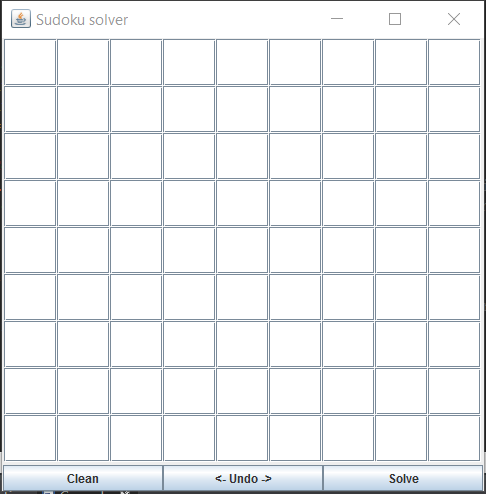
Programmeertaal: Java

Link naar Github: <https://github.com/Raidok/Sudoku-solver>

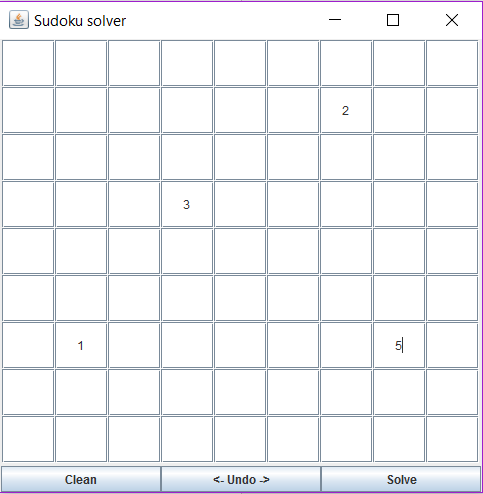
# Wat is performance analysis?

# Werking

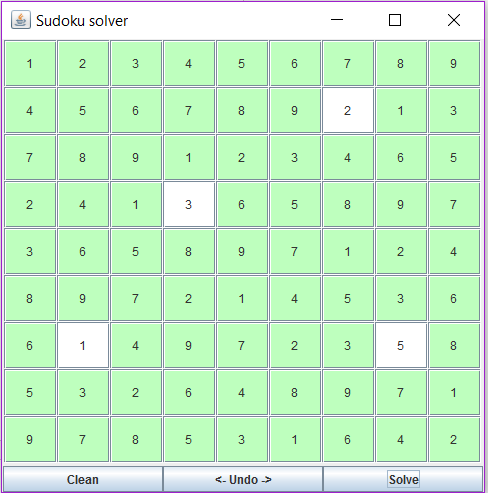
Dit is scherm dat men krijgt bij het opstarten van het programma. Hier heb je drie verschillende functies. De solve functie, waarbij de sudoku wordt opgelost. De undo functie, waarbij de vorige actie ongedaan wordt. De clean functie, waarbij de sudoku wordt gereset.



Als volgende moeten de getallen ingegeven worden. Je kan zelf kiezen welke getallen je waar gaat plaatsen.



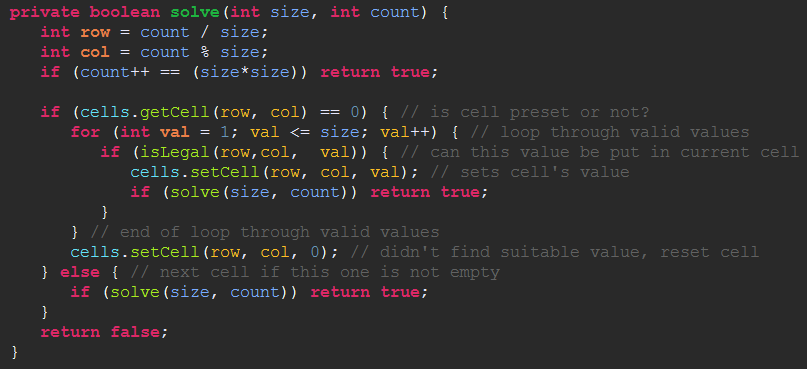
Wanneer je de getallen ingevoerd hebt en op “solve” klikt gebeurt er dit.



Als extra functie kan je de suduko resetten door op clean te klikken, zodat je constant aan een nieuwe sudoku kan beginnen.

# Algoritme

Bij dit project wordt gebruik gemaakt van het backtrack algoritme. Hierbij moeten niet alle oplossingen bekeken worden. Bij dit algoritme zal het programma een aantal keuzes moeten maken. Wanneer het programma op een dood spoor terecht komt, zal het programma terug naar het begin gaan en andere keuzes proberen te zoeken die wel werken. Dus bij deze sudoku solver zal hij nagaan of hij deze waarde in een bepaalde cel kan gooien. Indien niet, dan zal hij verder kijken voor een andere waarde. Zo zal hij die lus blijven afgaan tot als hij een oplossing gevonden heeft. Hieronder is de algemene methode die deze sudoku solver gebruikt:



Referentie naar artikel over het backtrack algoritme:

<http://algorithms.tutorialhorizon.com/introduction-to-backtracking-programming/>

In bijlage vindt u ook een uitgebreide PowerPoint waar ze dieper ingaan op het algoritme.

# Tools

## JConsole

We hebben voor onze meetresultaten gebruik gemaakt van JConsole. Dit is een interessante tools om performantieanalyses te doen. Dit geeft een mooi overzicht van de gebruikte memory van de applicatie en de threads die er momenteel gaande zijn. In de meetresultaten vindt u enkele screenshots terug van JConsole in zijn werking.

Referentie naar JConsole documentation:

<http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html>

## VisualVM

## JProfiler

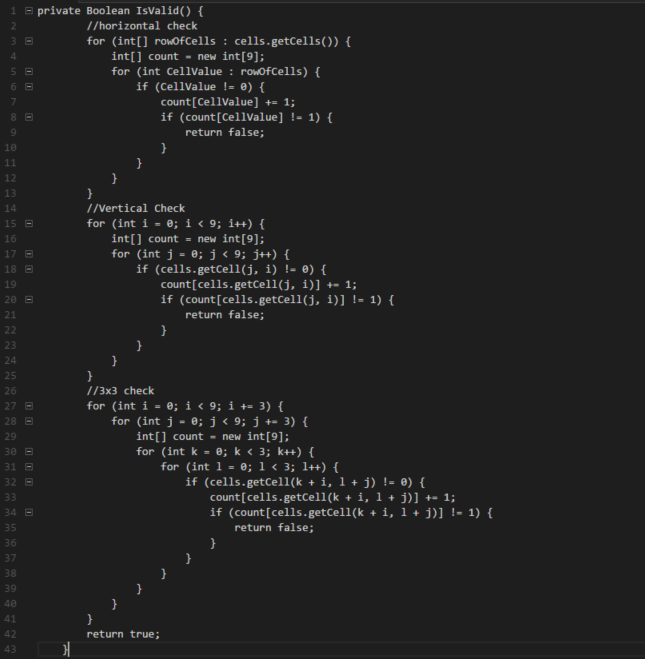
# Analyse

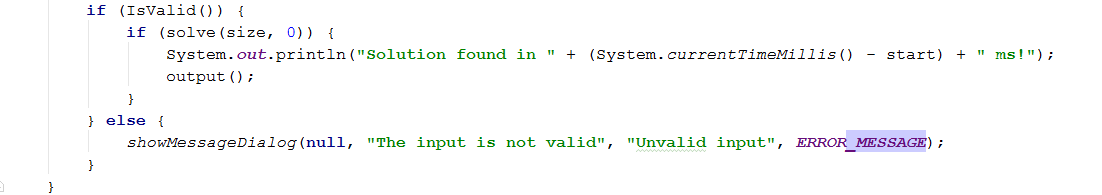
## Probleem #1: Crash programma bij foute invoer

### Beschrijving

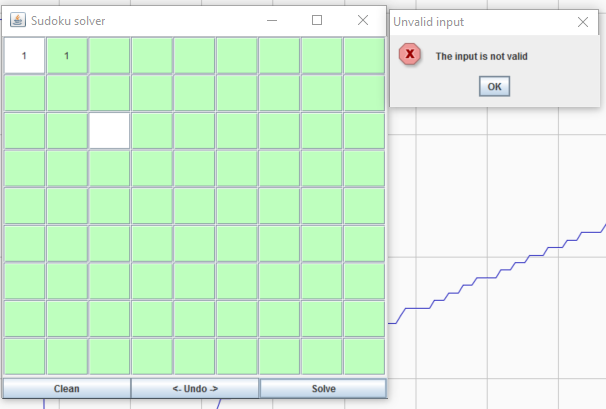
Als het programma een sudoku krijgt gegeven die onmogelijk op te lossen is, blijft het vaststeken voor 3-5 minuten omdat het programma het toch probeert op te lossen.  
Hiervoor zou een controle geschreven kunnen worden zodat dit niet gebeurt maar rechtstreeks een error word gegeven.

### Oplossing





Door hier dan gebruik te maken van een simpele if test, zorgen we ervoor dat deze fout ten alle tijden wordt opgevangen.



Nu word er rechtstreeks een foutmelding gegeven binnen de 3ms dat deze sudoku niet opgelost kan worden in de plaats van 5 minuten zonder een foutmelding.

## Probleem #2: Stijgende memory

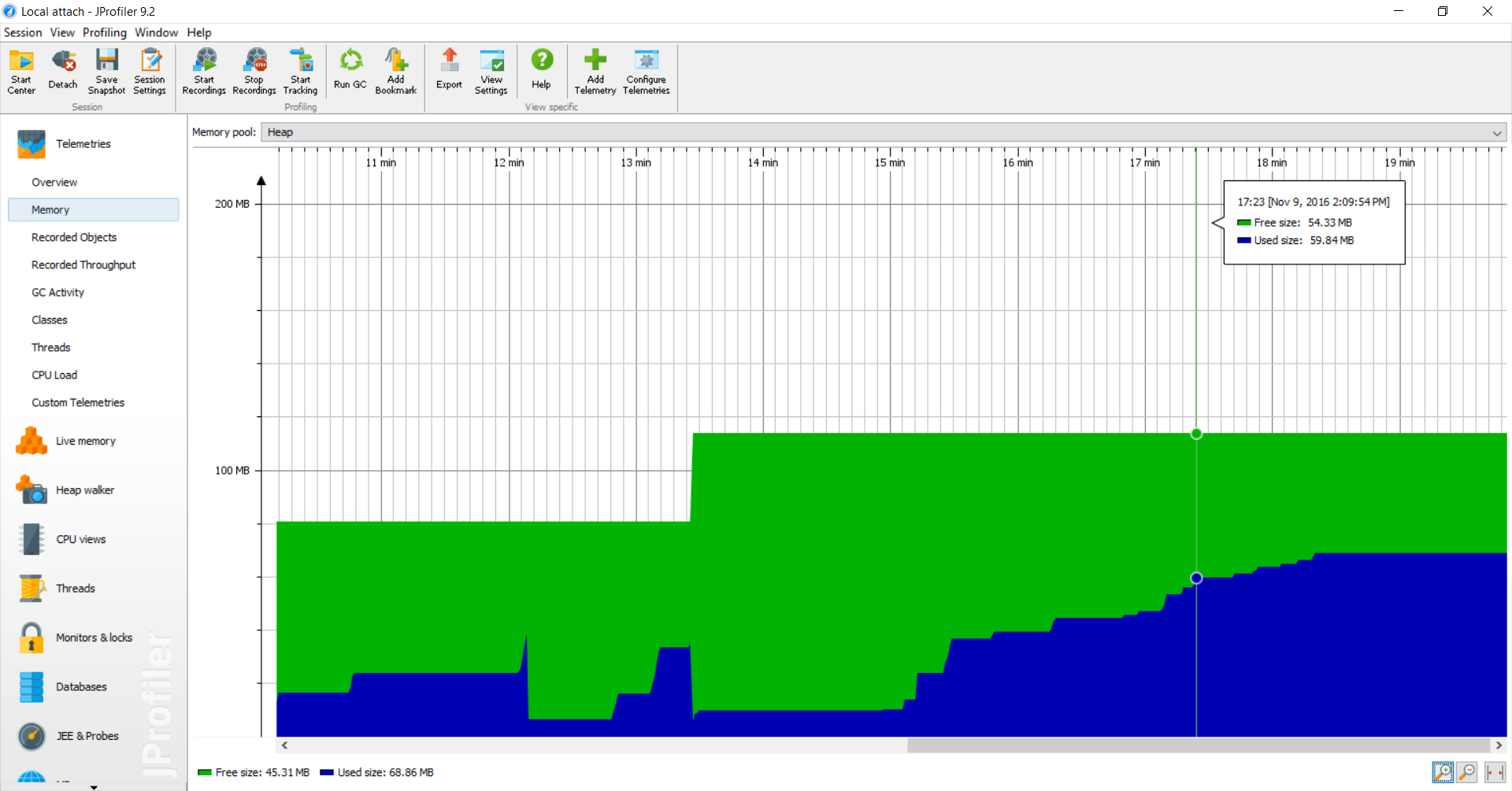
### Beschrijving

We hebben gemerkt dat als we een memory analyse deden op het programma dat de memory hoger en hoger werd. Soms als de memory een piek bereikt wordt de heap verhoogd en gebeurt er een garbage collect, zodat er de volgende keer meer memory gebruikt kan worden. Het is onnodig dat dit programma veel memory gebruikt, dus zochten wij hier een oplossing voor. Dit komt voor wanneer er telkens op “solve” wordt geklikt.

### Meetresultaten

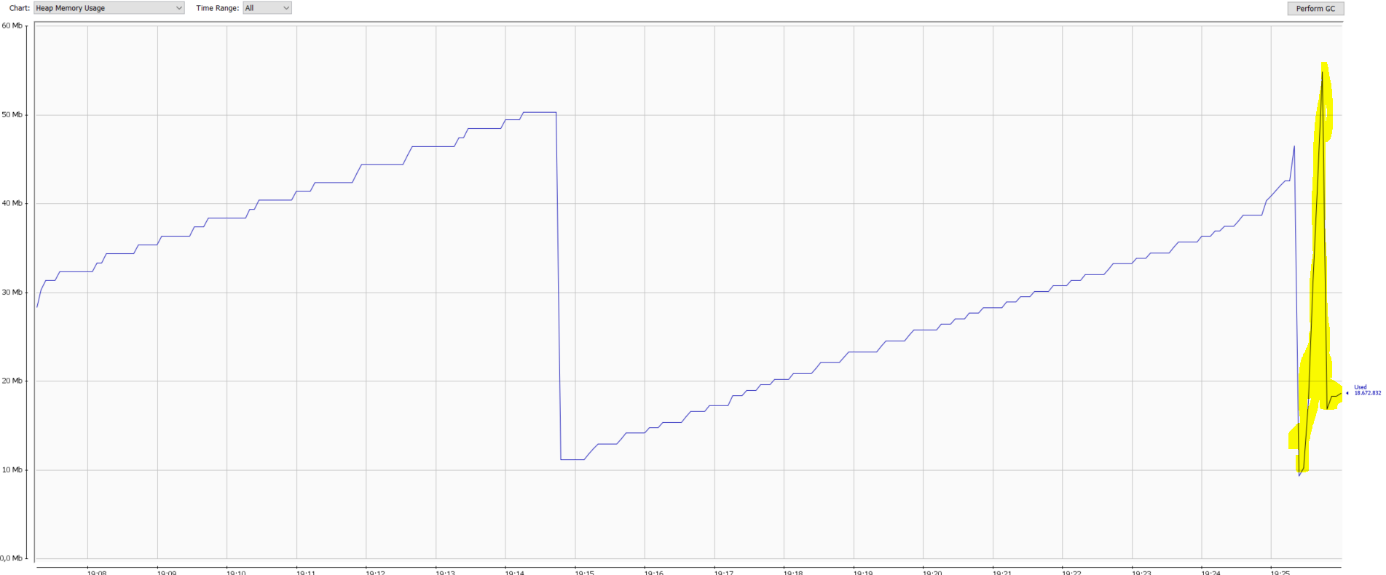
Zoals u hieronder kan zien, start het met een maximum heap size van onder 100mb. Wanneer we op solve blijven klikken en de memory in de lucht blijft gaan, merken we op dat de maximum heap size verhoogd wordt boven de 100mb. Daarnaast merken we ook op wanneer de heap size verhoogd wordt dat de memory ook hoger kan gaan dan normaal.

Het programma start op met een memory van ongeveer 9mb. Dit wordt pas collected wanneer er een piek wordt bereikt. Wanneer de heap size wordt verhoogd kan de memory zelfs tot 80mb gaan. Dit kan hoger en hoger blijven gaan telkens als de heap size wordt vergroot.



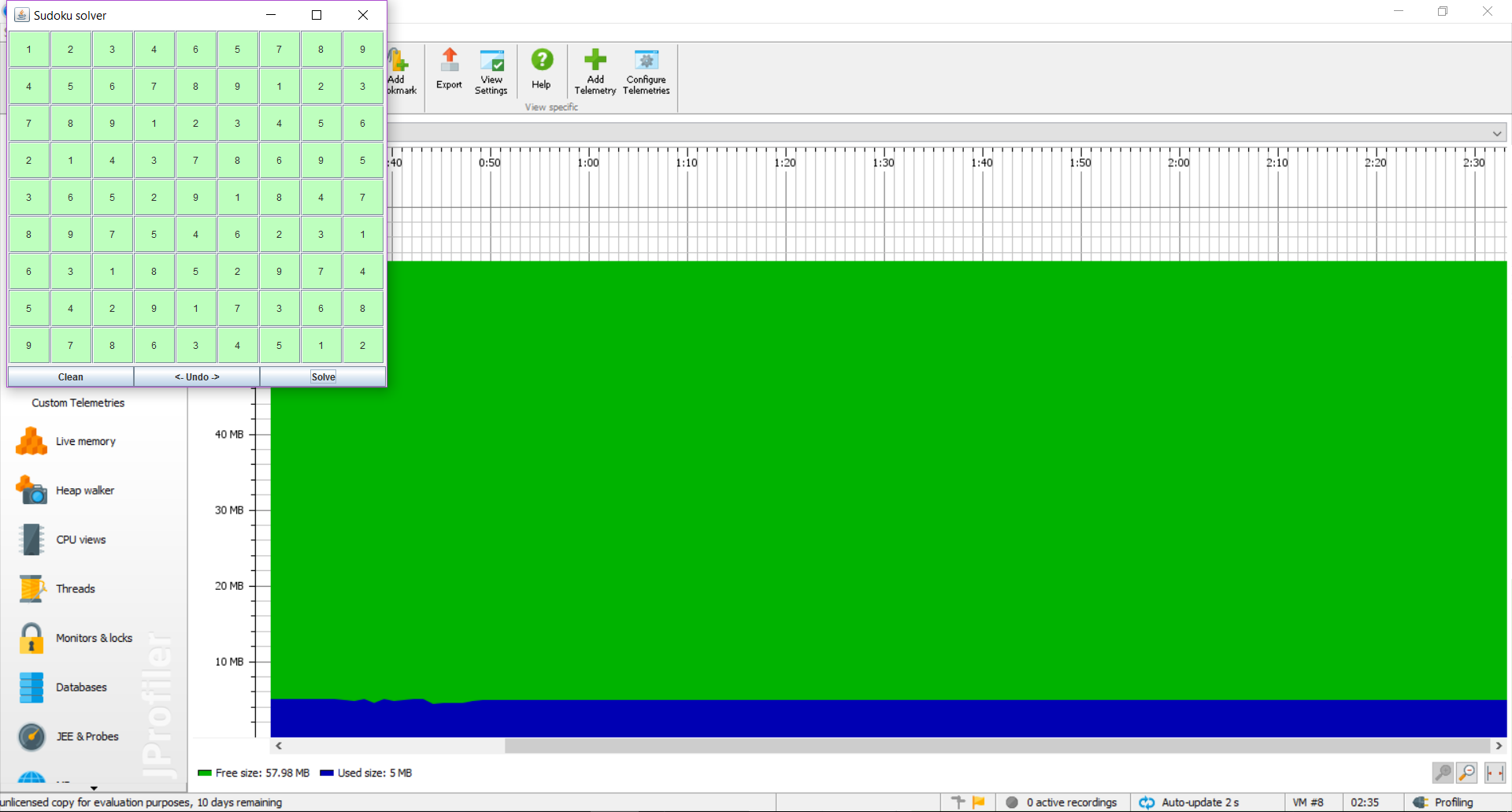
Groen: Maximum heap size

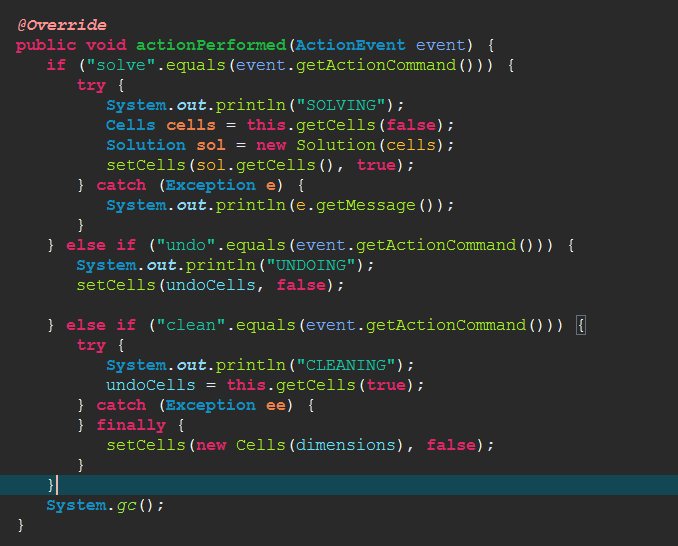
Blauw: Memory gebruik



### Oplossing

Onze oplossing hiervoor is om het programma een geforceerde garbage collect te laten doen (Dit kan u op de laatste regel zien van de tweede screenshot). Telkens wanneer er een knop wordt gedrukt in het programma zal garbage colllection optreden zodat we niet zoveel memory krijgen na een bepaalde tijd. Zoals u ook kan zien op de eerste screenshot gebruikt het programma bijna altijd maar 5mb in plaats van 80mb zoals in onze meetresultaten.





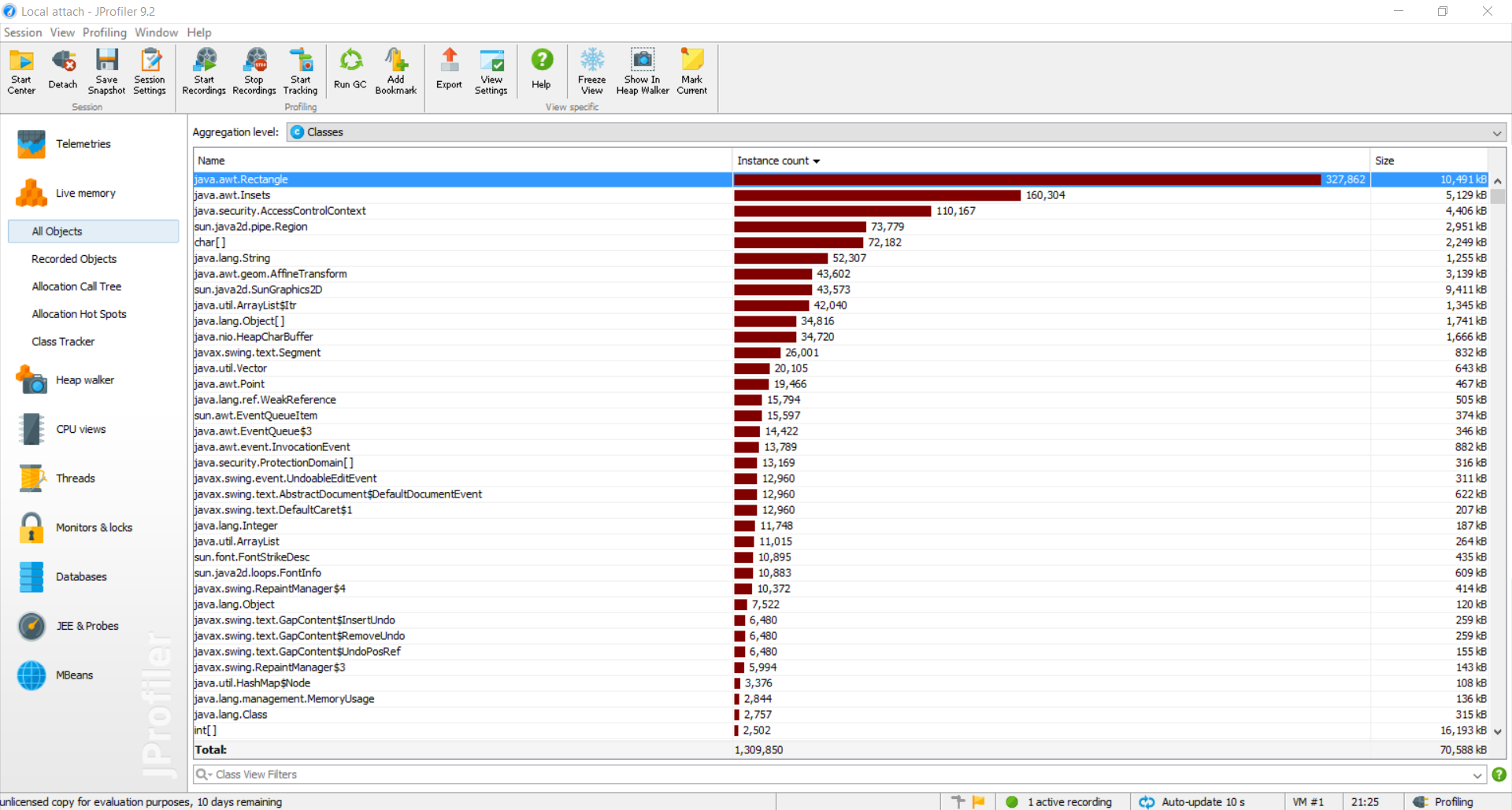
## Probleem #3: Objecten verdwijnen niet

### Beschrijving

Als gevolg van een stijgende memory merken we ook op dat de objecten niet worden collect. Dus telkens als er een sudoku opgelost wordt, gaat hij telkens opniew dezelfde objecten aanmaken, terwijl er niks gedaan wordt met de oude objecten zoals u subiet wel gaat merken bij de meetresultaten.

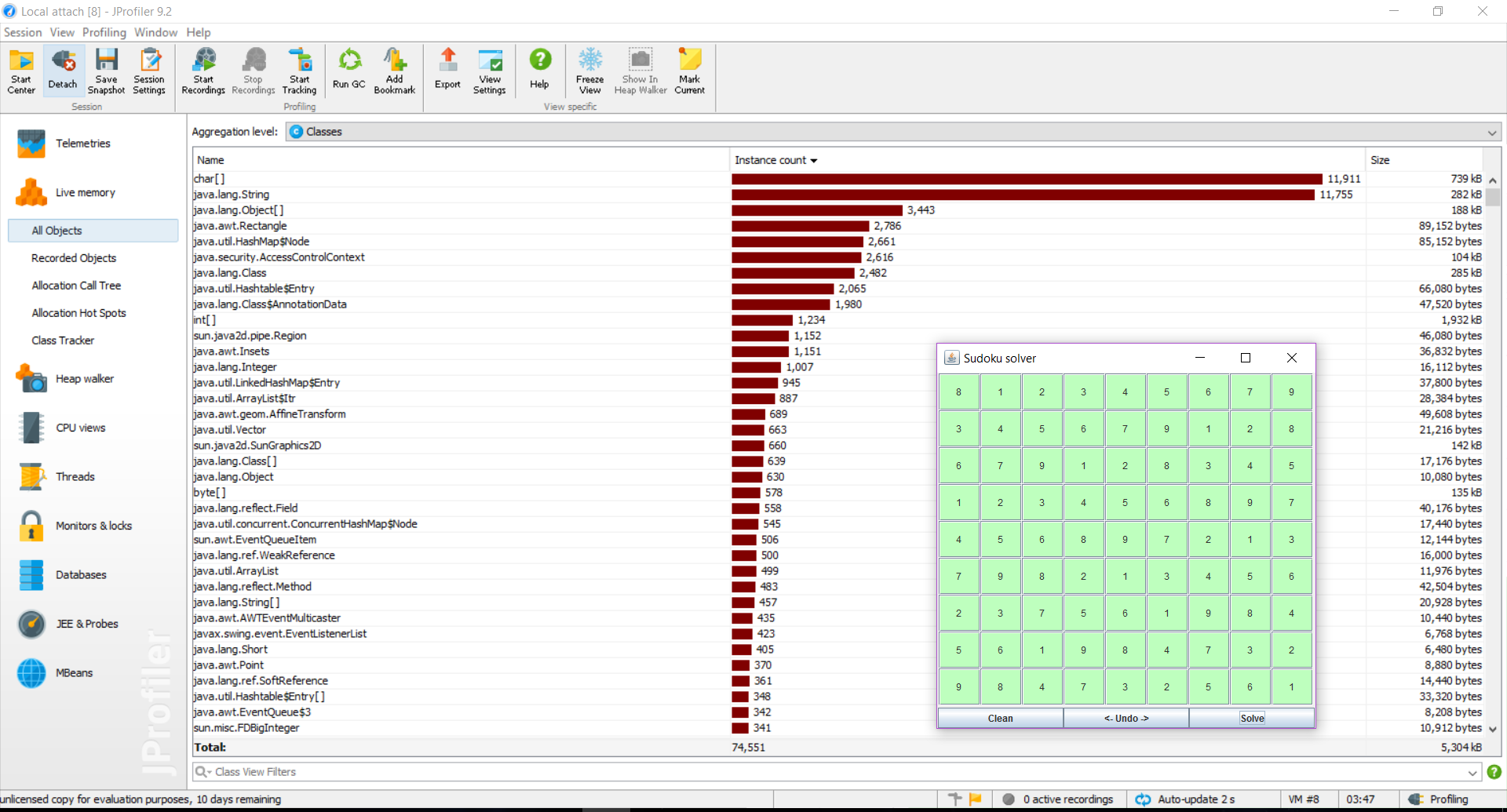
### Meetresultaten

Zoals men hier kan zien, worden er tamelijk veel rectangle’s aangemaakt. Dit is nog lichtjes uitgedrukt. Na een goede aantal keer op de “solve” button te hebben geklikt zijn er maar liefst in de 320.000 rectangle objecten. Wat al meer dan 10mb gebruikt van de memory. Dit is dus de oorzaak waarom het programma zoveel memory in gebruik neemt. Dit geldt ook voor al die andere objecten die getoond worden.



### Oplossing

De oplossing bij probleem 2 heeft er hier ook voor gezorgd dat alle rectangle’s en andere objecten collected worden en dat het memory gebruik hierdoor veel minder is. Nu zien we dat er nog maar in de 2000 rectangle’s bestaan.



## Probleem #4:

### Beschrijving

### Meetresultaten (als ge geen meetresultaten hebt moogt ge dit puntje wegdoen)

### Oplossing

Het oplossen van de sudoku gebeurt meestal binnen de 3-5ms. Dit zou versneld kunnen worden door gebruik te maken van multithreading. Hierdoor zouden ook meerdere oplossingen kunnen terug gegeven worden want nu krijg je altijd maar één oplossing terug terwijl er veel meer mogelijk zijn.

# Reflectie

## Joran Claessens:

## Bram Van Vleymen: