Performance Analysis Programming Expert Project : Sudoku Solver

Team

Groepsleden: Bram Van Vleymen, Joran Claessens

Klas: 3AOND

Programmeertaal: Java

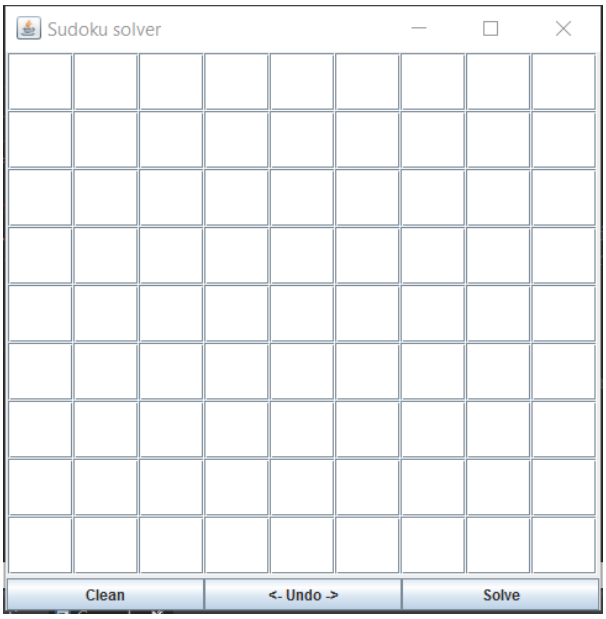
Link naar Github: https://github.com/Raidok/Sudoku-solver

Wat is performance analysis?

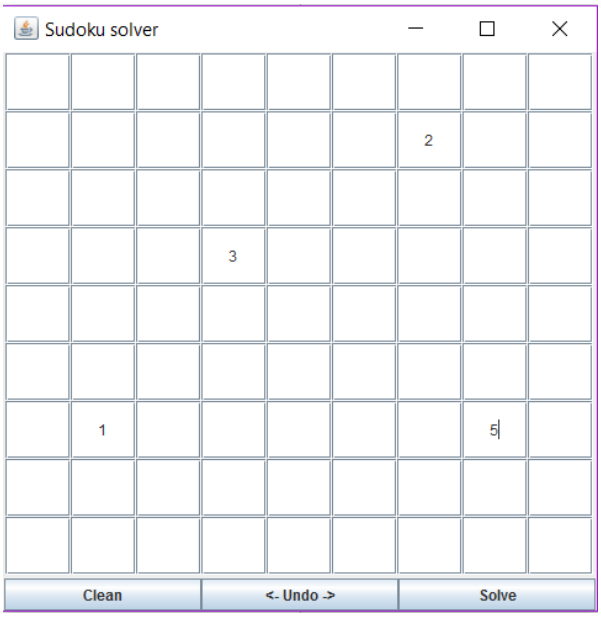
Performance analysis of profiling is het bestuderen van het geheugen of de complexiteit van een programma. Ook om te meten hoe vaak bepaalde functies worden opgeroepen en hoe lang deze functies duren. Dit word meestal gebruikt om het programma te optimaliseren. Dit word gedaan door een profiler aan het programma te hangen. Voorbeelden over profilers staat bij tools.

Werking

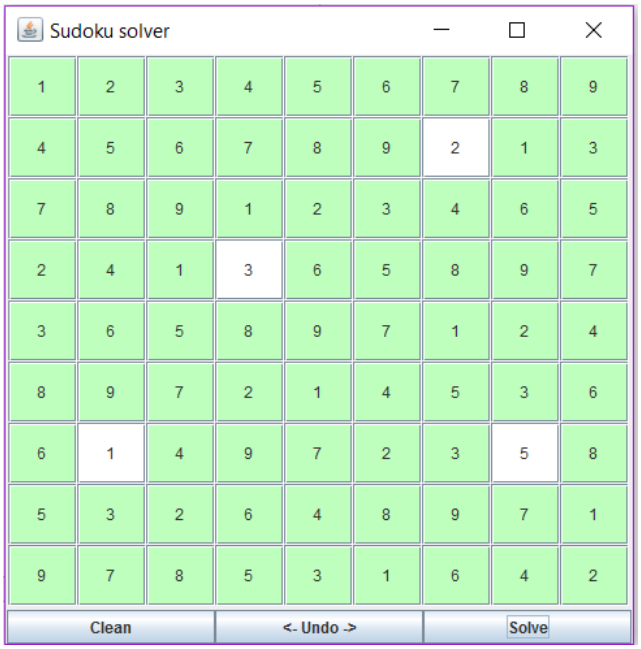
Dit is scherm dat men krijgt bij het opstarten van het programma. Hier heb je drie verschillende functies. De “solve” functie, waarbij de sudoku wordt opgelost. De “undo” functie, waarbij de vorige actie ongedaan wordt. De “clean” functie, waarbij de sudoku wordt gereset.



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:

```
private boolean solve(int size, int count) {
   int row = count / size;
   int col = count % size;
   if (count++ == (size*size)) return true;

   if (cells.getCell(row, col) == 0) { // is cell preset or not?
      for (int val = 1; val <= size; val++) { // loop through valid values
         if (isLegal(row,col,  val)) { // can this value be put in current cell
            cells.setCell(row, col, val); // sets cell's value
            if (solve(size, count)) return true;
         }
      } // end of loop through valid values
      cells.setCell(row, col, 0); // didn't find suitable value, reset cell
   } else { // next cell if this one is not empty
      if (solve(size, count)) return true;
   }
   return false;
}
```

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 tool 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

Ook hebben we gebruik gemaakt van VisualVM. Deze tool gaat dieper in op java applicaties. Zo kan deze een visueel beeld laten tonen van hoe de garbage collection van een java programma er kan uit zien. Niet alleen geeft de tool je een beeld van de garbage collection, maar ook memory, cpu usage …

Wij hebben deze tool vooral gebruikt als confirmatie voor de garbage collection van ons project.

Referentie naar VisualVM documentation:

https://docs.oracle.com/javase/8/docs/technotes/guides/visualvm/

JProfiler

De laatste tool die we gebruikt hebben voor ons project is JProfiler. Deze tool bestaat uit uitgebreide methodes die je een diepere inkijk geeft in jouw programma. Deze tool kan onder andere laten zien hoeveel keer een methode wordt opgeroepen, hoeveel objecten er bestaan, memory gebruik, cpu usage, de threads en veel meer. Voor ons project heeft deze tool ons het meest geholpen bij het zoeken van performantieproblemen.

Referentie naar JProfiler documentation

https://www.ej-technologies.com/products/jprofiler/docs

Analyse

Probleem #1: Crash programma bij foute invoer

Beschrijving

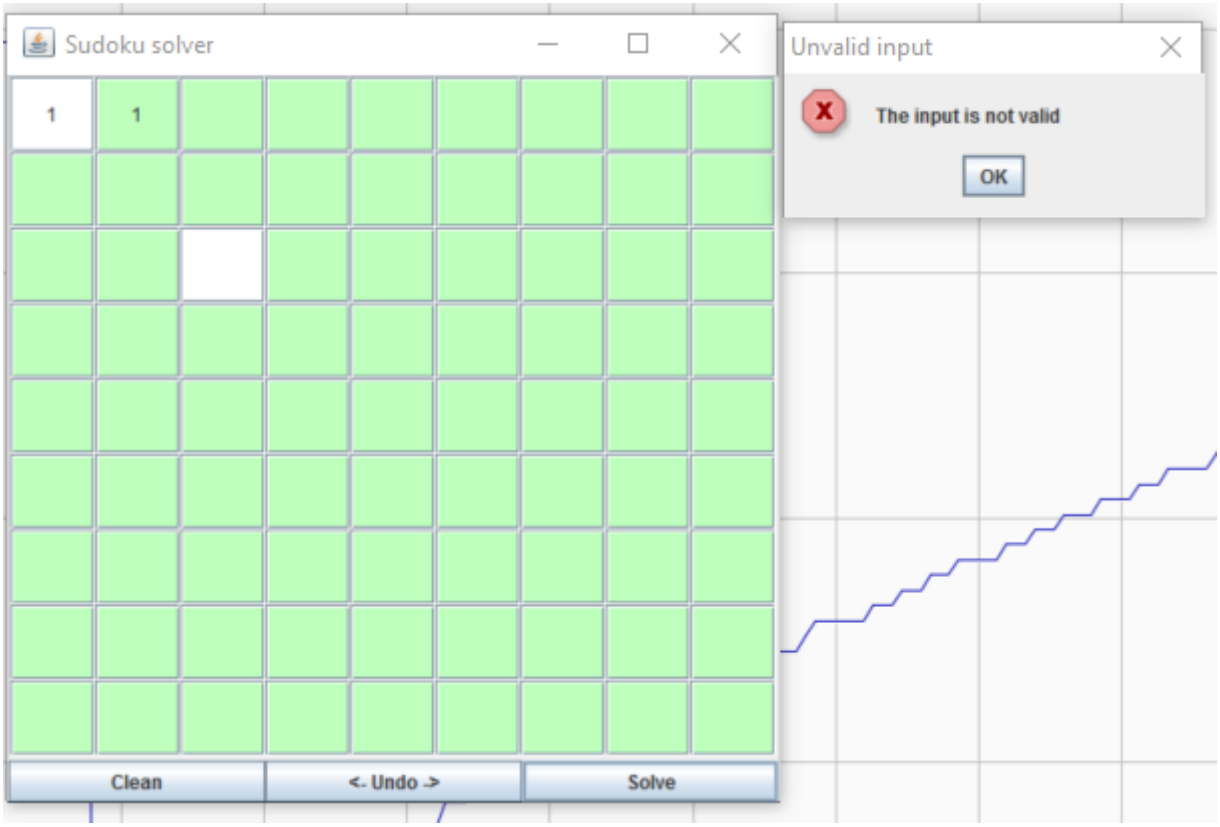
Als het programma een sudoku krijgt gegeven die onmogelijk op te lossen is, blijft het vaststeken voor 3-5 minuten omdat het programma het toch probeert op te lossen.
Hiervoor zou een controle geschreven kunnen worden zodat dit niet gebeurt, maar rechtstreeks een error word gegeven.

Oplossing

```
private Boolean IsValid() {
        //horizontal check
        for (int[] rowOfCells : cells.getCells()) {
            int[] count = new int[9];
            for (int CellValue : rowOfCells) {
                if (CellValue != 0) {
                    count[CellValue] += 1;
                    if (count[CellValue] != 1) {
                        return false;
                    }
                }
            }
        }
        //Vertical Check
        for (int i = 0; i < 9; i++) {
            int[] count = new int[9];
            for (int j = 0; j < 9; j++) {
                if (cells.getCell(j, i) != 0) {
                    count[cells.getCell(j, i)] += 1;
                    if (count[cells.getCell(j, i)] != 1) {
                        return false;
                    }
                }
            }
        }
        //3x3 check
        for (int i = 0; i < 9; i += 3) {
            for (int j = 0; j < 9; j += 3) {
                int[] count = new int[9];
                for (int k = 0; k < 3; k++) {
                    for (int l = 0; l < 3; l++) {
                        if (cells.getCell(k + i, l + j) != 0) {
                            count[cells.getCell(k + i, l + j)] += 1;
                            if (count[cells.getCell(k + i, l + j)] != 1) {
                                return false;
                            }
                        }
                    }
                }
            }
        }
        return true;
    }
```

```
        if (IsValid()) {
            if (solve(size, 0)) {
                System.out.println("Solution found in " + (System.currentTimeMillis() - start) + " ms!");
                output();
            }
        } else {
            showMessageDialog(null, "The input is not valid", "Unvalid input", ERROR_MESSAGE);
        }
    }
```

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

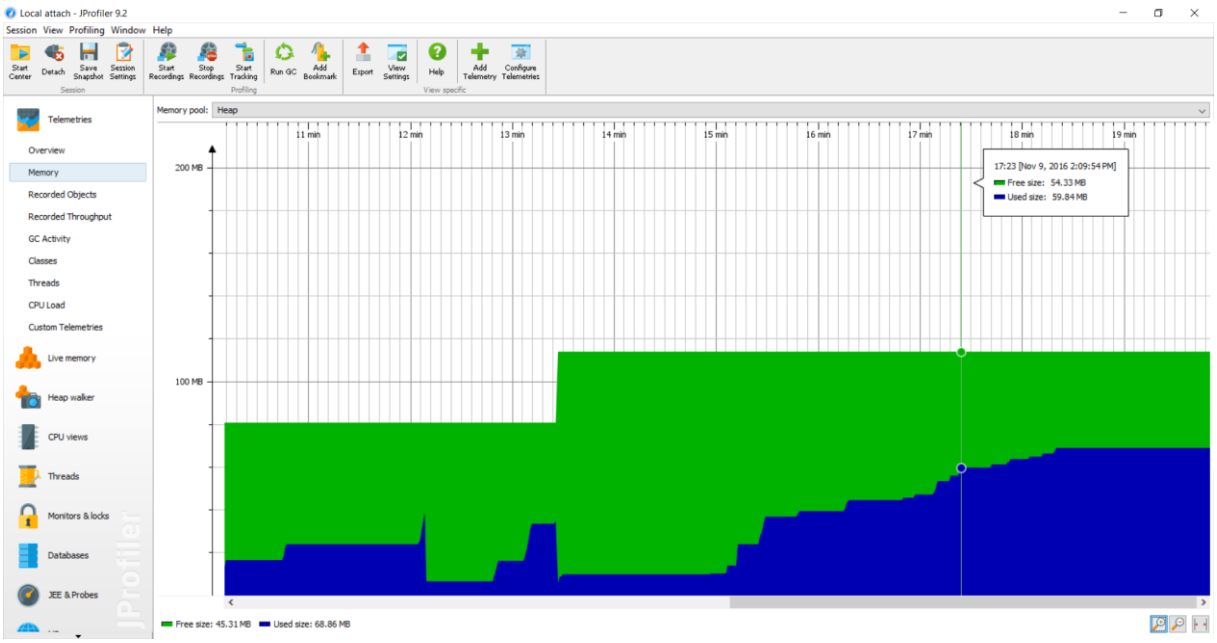
Beschrijving

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 een knop wordt geklikt.

Meetresultaten

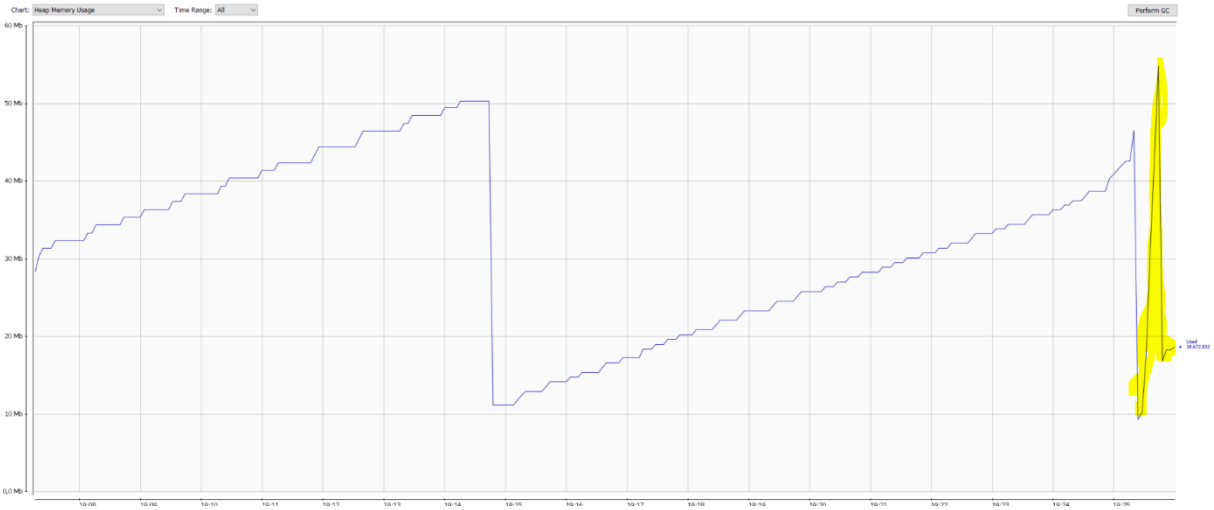
Zoals u hieronder kan zien, start het met een maximum heap size van onder 100mb. Wanneer we op solve blijven klikken en de memory in de lucht blijft gaan, merken we op dat de maximum heap size verhoogd wordt boven de 100mb. Daarnaast merken we ook op wanneer de heap size verhoogd wordt dat de memory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



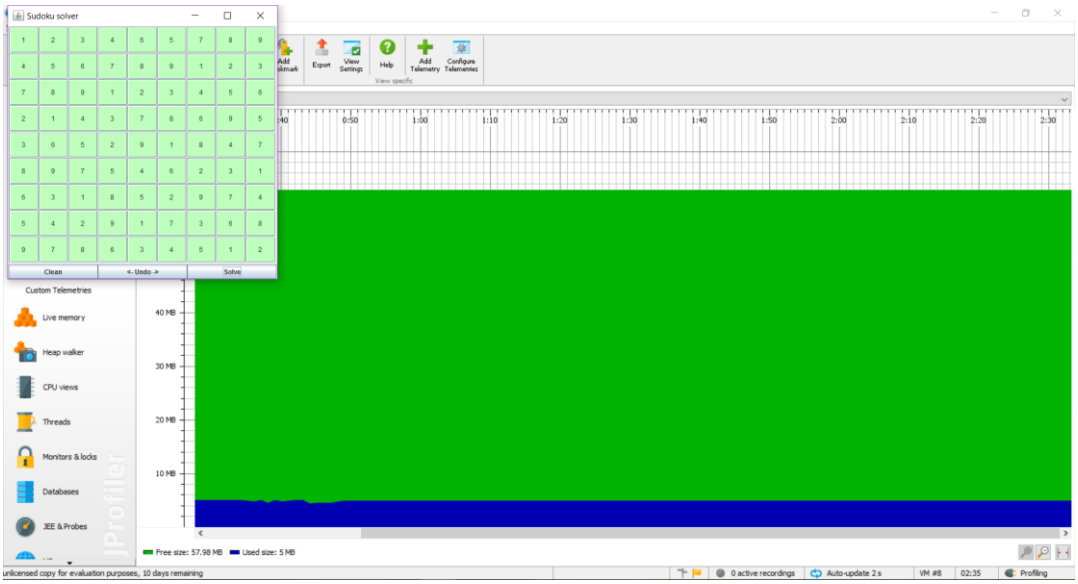
Groen: Maximum heap size

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 er 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.



```java
@Override
public void actionPerformed(ActionEvent event) {
    if ("solve".equals(event.getActionCommand())) {
        try {
            System.out.println("SOLVING");
            Cells cells = this.getCells(false);
            Solution sol = new Solution(cells);
            setCells(sol.getCells(), true);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    } else if ("undo".equals(event.getActionCommand())) {
        System.out.println("UNDOING");
        setCells(undoCells, false);

    } else if ("clean".equals(event.getActionCommand())) {
        try {
            System.out.println("CLEANING");
            undoCells = this.getCells(true);
        } catch (Exception ee) {
        } finally {
            setCells(new Cells(dimensions), false);
        }
    }
    System.gc();
}
```

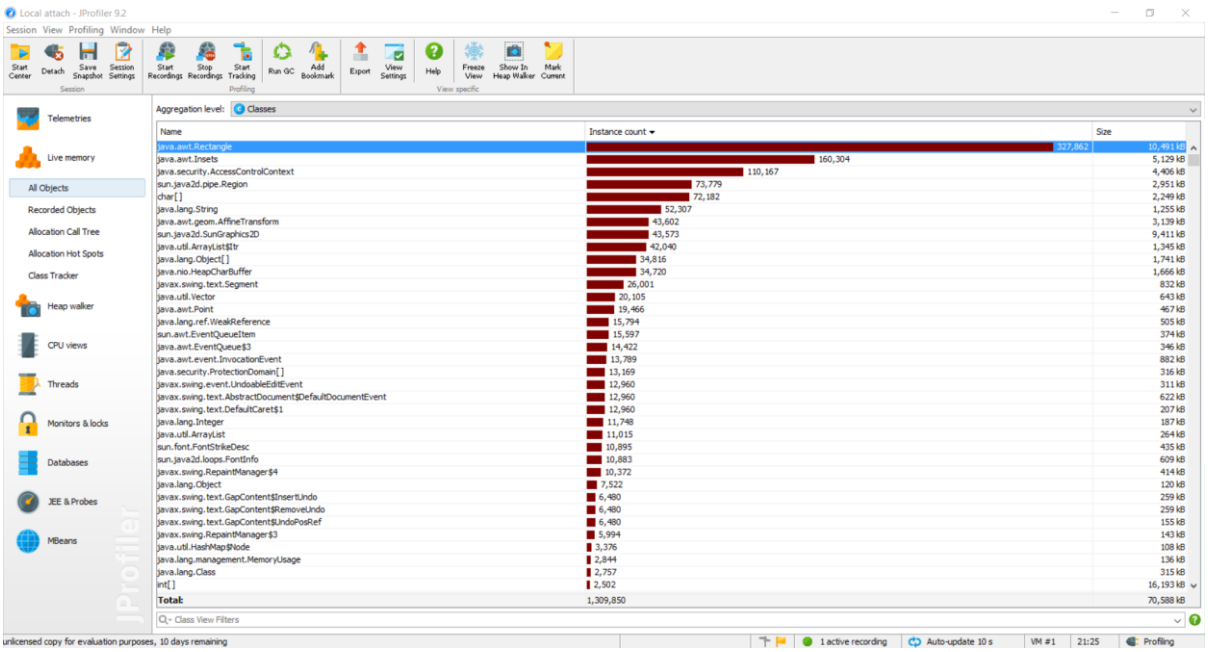
Probleem #3: Objecten verdwijnen niet

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 opnieuw dezelfde objecten aanmaken, terwijl er niks gedaan wordt met de oude objecten zoals u straks wel gaat merken bij de meetresultaten.

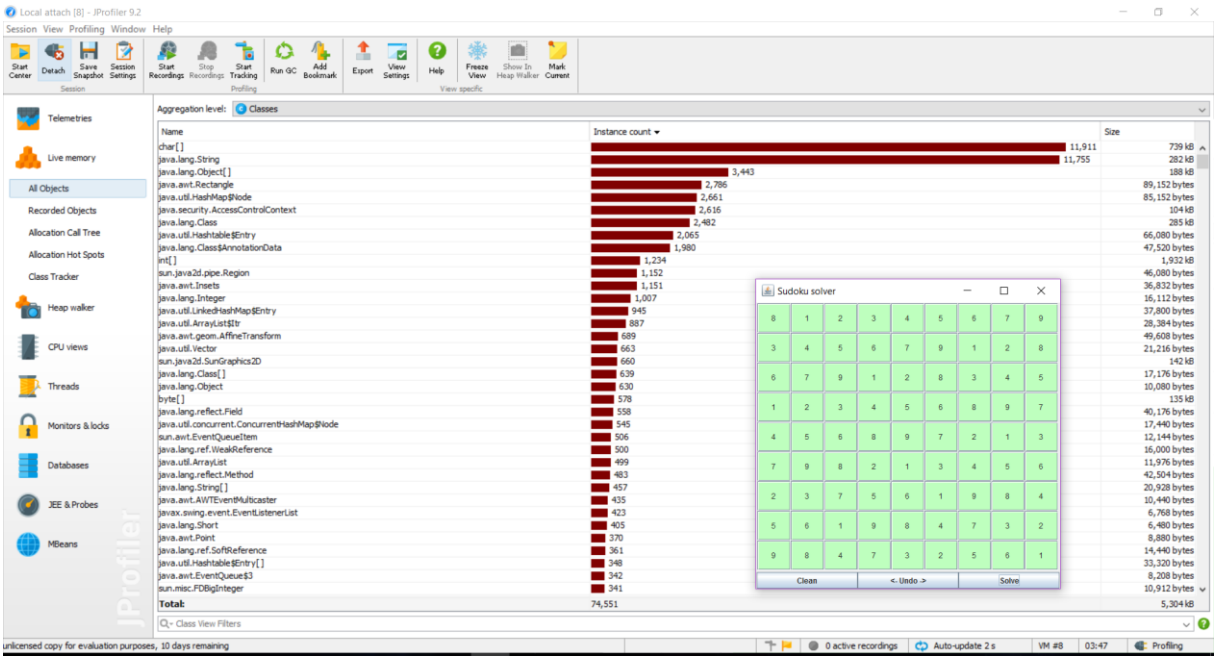
Meetresultaten

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.



Reflectie

Joran Claessens

Toen we aan dit project begonnen had ik totaal geen idee hoe we er aan moesten beginnen. We wisten wel hoe en wat we hiervoor moesten gebruiken, maar mijn eerste gevoel gaf aan dat dit niet zo simpel ging zijn.

Zo waren onze eerste problemen dat we niet goed de performantieproblemen eruit konden halen. We zochten meer eigenlijk fouten in de code dan fouten van de performantie van het programma. Voor mij was dit een hele uitdaging om het programma op een hele andere manier te bekijken.

Eens we echt de tools aan het analyseren waren, zagen wij hier en daar misschien iets wat een normaal programma niet zou doen. Zo hebben we onder andere gemerkt dat de memory steeds de lucht bleef in gaan. Terwijl we dachten dat voor zo een licht programma dit niet mogelijk was. Dus hebben we hiervoor een oplossing gevonden waardoor het niet meer zoveel memory gebruikt.

Ik denk dat het toepassen van performance analysis dat dit toch heel handig kan zijn voor later. Als een klant een programma opent, wilt hij ook dat het snel gaat. Anders gaat het bedrijf hierdoor klanten kunnen verliezen. Dit is zeker iets om mee te nemen voor een toekomstige job, omdat dit altijd een meerwaarde kan geven aan het bedrijf.

Daarnaast is het ook zo bij het toepassen van deze analyse, dat er veel tijd erin gestoken moet worden. Als een bedrijf beschikt over een redelijke zware app, waar dan een performance analysis wordt op toegepast, zal dit toch heel veel tijd moeten innemen, waardoor het publishen van de app moet worden uitgesteld. Dit kom je ook onder andere tegen in online services dat uit heel veel gebruikers bestaat. Kan het systeem al deze gebruikers aan? Moeten hier maatregelen worden genomen? Hoe zit het dan met de kosten? Dit kan misschien voorkomen worden wanneer er dus gebruik wordt gemaakt van performance analysis.

Uit dit project kan ik toch concluderen dat dit ik dit zeker moet meenemen naar mijn volgende projecten. Als ik een programma schrijf zal ik meer rekening houden met de performantie. Zo heb ik in een vorig project meegekregen van mijn teamleden dat het programma toch een beetje traag was. Op dat moment zelf had ik totaal geen idee om het sneller te maken. Met de kennis die ik in dit project heb vertoefd, zou ik misschien toch een manier gevonden hebben om het programma sneller te laten werken.

Bram Van Vleymen

Toen we begonnen aan het project nadat we de sudoku solver hadden gevonden, dacht ik hoe gaan we hier ooit een probleem in vinden. Het programma is zo klein en op het eerste zicht was er geen probleem. Het moment dat we begonnen te analyseren met een tool zagen we toch al wat problemen. We waren sudoku's aan het oplossen en de memory bleef maar omhoog gaan. Het programma gebruikte 14mb bij het opstarten en na een paar keer oplossen al meer als 50mb. Dit hebben we dan ook opgelost door de garbage collector zelf te laten runnen.

Wat ik uit dit project geleerd heb is dat de problemen van een programma niet altijd te zien zijn door een gebruiker. Maar vaak op de achtergrond liggen.

Zelf ben ik ook gaan kijken naar programma's die ik geschreven heb. Hier had ik ook een memory leak terug gevonden en een ander programma waar de memory ook de hele tijd omhoog ging. Dit was natuurlijk normaal voor dit programma omdat deze om de 30 minuten data van een api afhaalden. Maar na dat deze verwekt was bleef dit gewoon staan. Nadat de data verwerkt was heb ik ook zelf de garbage collector laten runnen en toen was dit opgelost.

Persoonlijk vind ik performance analysis heel belangrijk. Niet per se in de sukoku solver, want hier ga je er niet veel van merken tenzij je het programma 100keer ofzo hebt openstaan. Maar in het programma wat ik zelf had geschreven. Het ram gebruik ging elk half uur omhoog met 100mb en dat is een groot probleem voor een programma dat 24/7 moet aanstaan. Omdat dit gewoon crashte op het moment dat het geheugen van de pc heeft gebruikt. Als je het programma niet analyseert valt dit niet op en dan wanneer de crash gebeurt is het een groot vraagteken.

Het was zeker een zinvol project en vanaf nu zal ik elke tool toch kort analyseren die ik zelf schrijf. Het heeft ook mijn gedachte gang bij het programmeren veranderd. Ik denk niet meer hoe kan ik dit het beste doen maar hoe kan ik dit het beste doen zodat het programma performant blijft. Want hier een paar megabyte bespaart lijkt niet veel, maar als dit op meerdere plaatsen gebeurt begint dit al snel op te tellen.