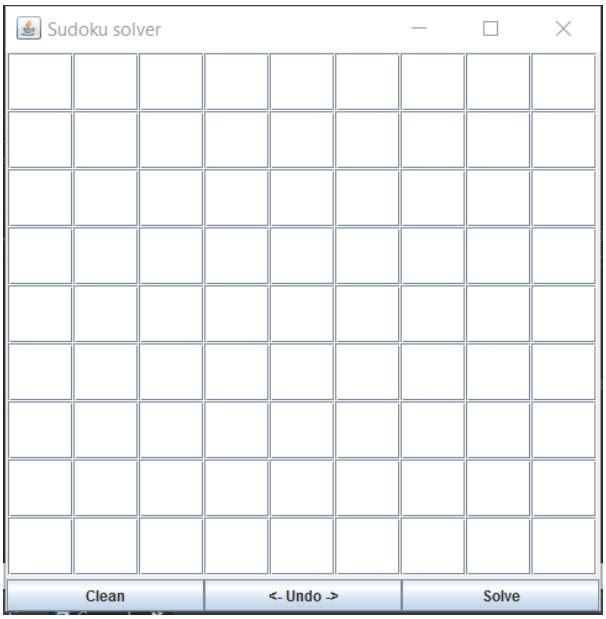
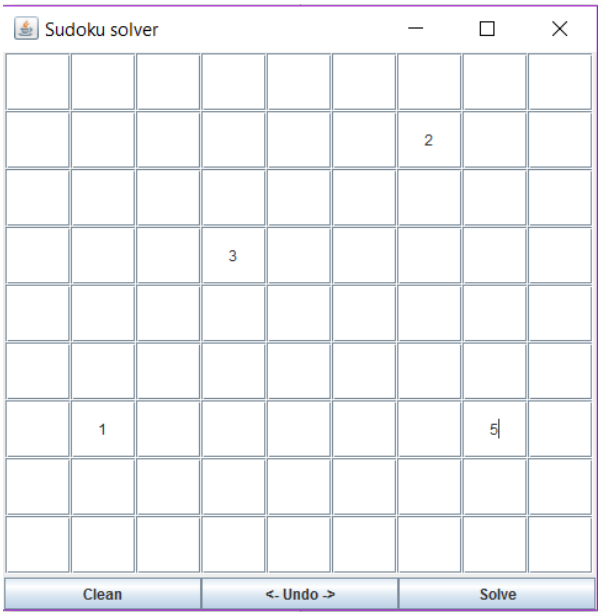
Draft Programming Expert Project : Sudoku Solver

Werking

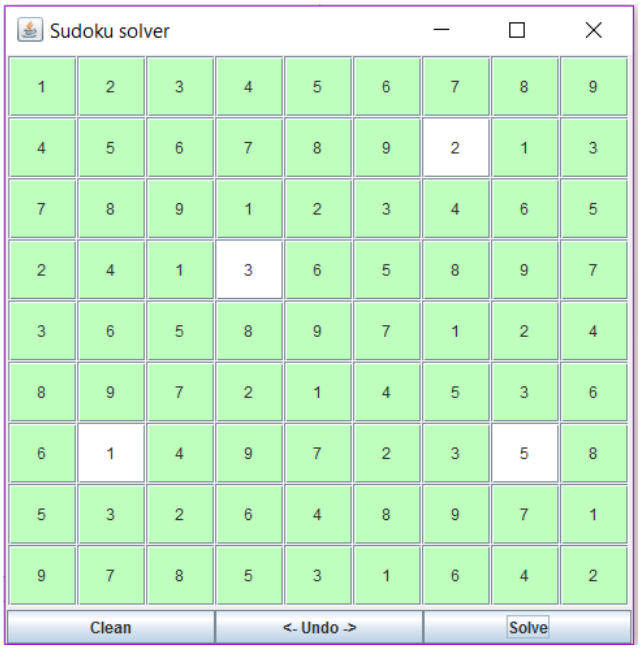
Dit is scherm dat men krijgt bij het opstarten van het programma. Hier heb je drie verschillende functies. De solve functie, waarbij de sudoku wordt opgelost. De undo functie, waarbij de vorige actie ongedaan wordt. De clean functie, waarbij de sudoku wordt gereset.



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

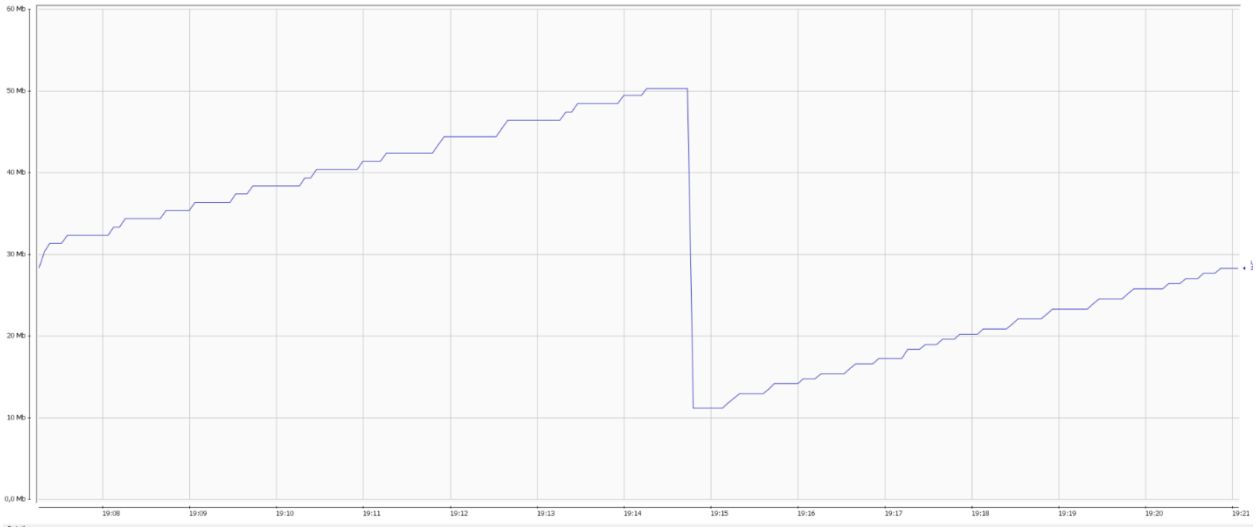
We hebben voor onze meetresultaten gebruik gemaakt van JConsole. Dit is een interessante tools om performantieanalyses te doen. Dit geeft een mooi overzicht van de gebruikte memory van de applicatie en de threads die er momenteel gaande zijn. In de meetresultaten vindt u enkele screenshots terug van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

Referentie naar JConsole documentation:

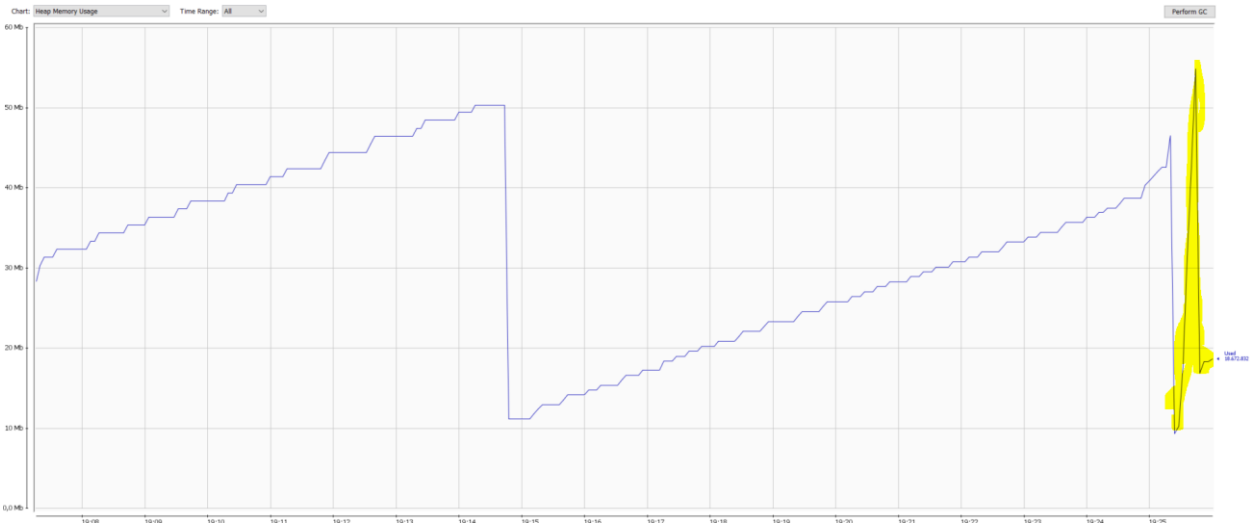
http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html

Meetresultaten

Als het programma wordt opgestart gebruikt het geleidelijk aan meer geheugen, ook al word er niks mee gedaan. Ik heb niet gevonden hoe dit komt omdat het programma eigenlijk niks doet. Maar ik vermoed dat dit de ui is.



Als ik op de undo button blijf klikken schiet het geheugen omhoog maar deze wordt ook weer opgenomen over de 50mb. Bij het oplossen van de sudoku gebeurt hetzelfde. Als er op clear word gedrukt word de ui wel leeggemaakt maar de waarden worden in een andere variable gezet zodat de undo button nog steeds werkt. De undo button werkt ook nog wanneer de garbage collection gebeurt is.



Voorstellen tot verbetering

1) Als in het programma een sudoku krijgt gegeven die onmogelijk op te lossen is blijft het hangen voor 3-5minuten omdat het toch probeerd deze optelossen.
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);
        }
    }
```

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

2) De ui gebruikt telkens meer memory zonder dat er in het programma wat gedaan wordt. De ui zou anders opgebouwd kunnen worden zodat dit niet gebeurt.

3) 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.