Verslag 1 FP

Sudoku Solver

Graphics

De graphics module bestaat uit een aantal onderdelen, die samengevoegd worden tot een enkel plaatje. Een frame bestaat uit de grijze achtergronden achter de sudoku, mogelijk een vakje dat gekleurt is om aan te geven dat er een fout gemaakt is, de lijnen die het bord definiëren, de getallen in de vakjes, en een aantal tekstjes om de besturing te tonen, en mogelijke errors die ontstaan door bijvoorbeeld IO fouten.

De GUI ondersteunt een aantal commando's:

`h`	Voer een hidden single check uit op een sudoku
`v`	Voer een visible single check uit op een sudoku
`n`	Voer een naked pairs check uit op een sudoku
`o`	Los een sudoku op volgens onze strategie
`r`	Lees een sudoku in uit een bestand
`s`	Sla een sudoku op in een bestand
`[1..9]`	Selecteer een getal, klik op een vakje om deze te plaatsen
`d`	Doodle, verwijdert en voegt notities toe, door te klikken op de notitieplekken.

Wat betreft code, in veel van de functies wordt de volledige GUI opnieuw getekend, dat is eigenlijk niet netjes, want slechts een klein deel van het scherm wordt geupdate. Helaas ontstaan er tijdens het hertekenen van delen van de GUI veel grafische glitches, waardoor in de praktijk het hertekenen van de hele GUI een beter resultaat geeft.

Ook bevat de code veel tijdelijke statusmeldingen, als een gebruiker een verkeerd cijfer in een veld probeert te zetten kleurt het veld rood. Al deze statusmeldingen vergen redelijk wat boekhouding, waardoor het updaten van de Store vaak uitgebreider lijkt dat het eigenlijk is. Voordeel is dat de GUI visueler is ten opzichte van de gebruiker, fouten worden directer getoond, en het is voor de eindgebruiker duidelijker waarom er dingen fout gaan.

Als laatste probleem zijn we de (x,y) (y,x) discussie tegen gekomen, de solver werkt vanuit (y,x), omdat in rijen denken makkelijker is dan in kolommen, maar de GUI wordt in (x,y) getekend, dus denkt in (x,y). In de praktijk bleek dit niet zo'n probleem, omdat de solver direct op een Sudoku type werkt, en eigen functies heeft voor het verkrijgen van rijen en kolommen en blokken. De GUI heeft alleen functies om individuele vlakjes binnen een Sudoku aan te passen, en gezien de solver deze functies verder niet nodig heeft, zijn deze in (x,y)

Verder bied de GUI de mogelijkheid om sudoku's te serializen en te deserializen, een sudoku wordt opgeslagen als een aantal regels, met punten op de plekken waar geen cijfer staat.
De gui kan vervolgens deze opgeslagen bestanden terugvertalen naar sudoku's die met de geimplementeerde algoritmen opgelost kunnen worden.

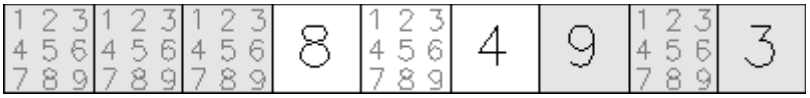
Algoritmes

De algoritmes zijn onderverdeeld in de volgende type functies:
- `getter` voor de getallen die het algoritme nodig heeft
- `filter` die gegeven de output van de getter vakjes kan aanpassen tot het goed (tussen) formaat
- `remover` die gegeven een rij een gecorrigeerde rij oplevert met behulp van de filter
- `checker` die gegeven de remover met behulp van `checkSudoku`[1] één iteratie doet over de sudoku en daarbij het algoritme dus respectievelijk op elke kolom, rij en blok uitvoert.

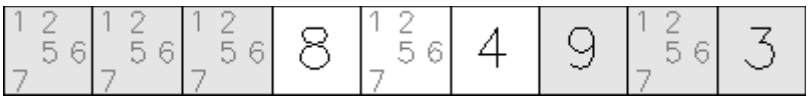
De checkers zijn bewust niet opgenomen in de beschrijvingen van de algoritmes, omdat deze slechts een aangepaste aanroep zijn van de remover.

Visible Singles

Het Visibles Singles algoritme streept alle opties die al zijn ingevult in een rij weg. Er wordt eerst een lijst opgevraagd met reeds ingevulde getallen op met `getNumbers`, zo levert de volgende rij de getallen 8,4,9 en 3 op.



Vervolgens verwijdert het algoritme alle gevonden getallen uit de lijst met mogelijke opties (als er meer dan één optie is) en krijgen we dus het volgende resultaat:



[1] Zie hoofdstuk Solver

Hidden Singles

Het Hidden Singles algoritme zoekt naar een vakje in een rij die als enige nog de mogelijkheid heeft voor een getal. Eerst moeten dus de opties worden opgezocht die nog maar 1 keer voorkomen. Dit wordt gedaan met `getHiddenSingles` die, gegeven een rij, een lijst oplevert met alle elementen daarvan die slechts één keer voorkomen.

De volgende rij levert de lijst van 6,2,8,9,1,3 op met `getHiddenSingles`:

Vervolgens kunnen deze getallen ingevuld worden met `filterHiddenSingles`, die gegeven een vakje en de getallen een ingevuld vakje teruggeeft indien mogelijk. Hierbij wordt vanzelfsprekend rekening gehouden met vakjes die reeds zijn ingevuld, deze hoeven immers niet aangepast te worden.

Vervolgens kan `removeHiddenSingles` gebruik maken van `filterHiddenSingles` om de hele rij te controleren indien er in die rij hidden singles worden gevonden met `getHiddenSingles`.

Als bovenstaande row wordt doorlopen met het Hidden Singles algoritme zal deze het volgende resultaat opleveren:

Naked Pairs

Het Naked Pair algoritme zoekt naar twee vakjes met twee identieke mogelijkheden, bijvoorbeeld twee vakjes die beiden als enige mogelijkheden 7 en 6 hebben. In een rij met een naked pair kunnen er dus geen andere vakjes zijn die deze mogelijkheden kunnen hebben.

Als eerste moet er worden gekeken welke vakjes slechts twee opties hebben. Vervolgens wordt de lijst van deze vakjes samengenomen, gesorteerd en gegroepeerd. We houden dan een lijst over als deze: [[[1,2],[1,2]],[[3,4],[3,4],[3,4]]]. Hier worden alle groepen met een lengte van twee uit gehaald. Zo houden we alleen een lijst van paren van paren over die wordt omgezet naar een formaat waarin het vergeleken kan worden met de vakjes die we zoeken: ([[1,2]]).

Dit alles wordt gedaan door de functie `getNakedPairs` die het paar [6,7] oplevert gegeven de volgende rij:

Vervolgens worden de gevonden paren gebruikt om een vakje aan te passen met `filterNakedPairs`. Bij Naked Pair is het echter zo dat deze niet ingevuld worden, maar dat

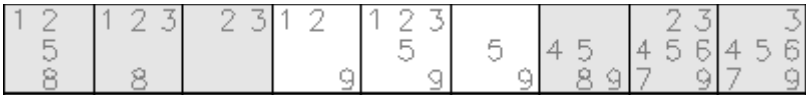
de gevonden getallen weggestreept worden uit de vakjes die niet één van de gevonden paren zijn.

Ten slotte is er de `removeNakedPairs` functie die de Naked Pairs verwijdert indien deze voorkomen in de rij. Gegeven de bovenstaande rij levert deze functie de volgende rij op:

Hidden Pairs

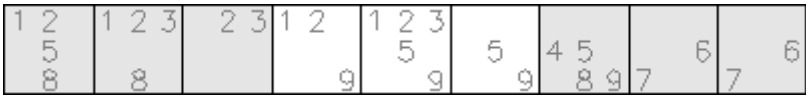
Het Hidden Pair algoritme zoekt naar een paar van vakjes die gezamenlijk de enige in de rij zijn die een tweetal mogelijkheden bevatten. Deze vakjes zullen dan ook slechts dit tweetal mogelijkheden als mogelijkheden hoeven te hebben.

Eerst moet er gezocht worden naar mogelijkheden die slechts twee maal voorkomen door `getHiddenPairs` die dit doet door de rij samen te nemen, te sorteren, te groeperen en vervolgens te filteren op paren met een lengte van twee. De functie levert bij de volgende rij [6,7] op:



Deze gevonden getallen zijn er helaas minder makkelijk uit te filteren dan bij vorige functies. Er moet door `filterHiddenPairs` namelijk eerst gekeken of er de gevonden getallen daadwerkelijk voorkomen in paren van twee. Hiervoor nemen we de intersection (hetgeen ze gemeen hebben) van elk vakje met de lijst met gevonden getallen. Deze intersection kan dan weer gebruikt worden om de paren te vinden door de rij te filteren op intersecties met lengte twee, vervolgens te sorteren, te groeperen en weer te filtreren op lengte twee. Wat je hierdoor overhoudt is de lijst van intersecties met lengte twee die tweemaal voorkomen in de rij. Aangezien de remover deze paren ook nodig heeft worden deze naast de volgende intersectie-rij ook gegeven:

Gegeven de intersectie en de lijst met paren kunnen nu de overbodige mogelijkheden worden verwijderd door `removeHiddenPairs`. Dit wordt gedaan door de rij en de intersectie naast elkaar te leggen. Waar de intersectie een vakje bevat dat ook in de lijst staat met paren wordt het paar gekozen. Is dit niet het geval dan wordt gewoon het oude vakje gekozen. Na het wegstrepen houd je dus de volgende rij over:



Solver

Om überhaupt de algoritmes te gebruiken moeten alle mogelijke opties worden bijgehouden. Dit wordt gedaan door de `prep` functie die op alle lege vakjes alle mogelijkheden ([1..9]) zet. In deze geprepareerde sudoku kunnen vervolgens opties worden weggestreept.

Om de sudoku op te lossen is het beter om de berekeningen op te delen in kleine stapjes. Zo hoeft niet elk algoritme een volledige sudoku uit elkaar te kunnen halen en opnieuw in elkaar te zetten.

Dankzij de functie `checkSudoku` wordt het schrijven van een algoritme een stuk makkelijker. Met behulp van deze functie kan namelijk een hele sudoku gecontroleerd worden met een meegegeven algoritme dat slechts een rij hoeft te kunnen controleren. De kolommen worden gecontroleerd, dan de rijen. Daarna worden alle blokken één voor één uit de sudoku gehaald, omgezet naar een rij, gecontroleerd, teruggezet naar een blok en vervolgens teruggeplaatst in de sudoku met behulp van de `checkBlock`, `getBlock`, `blockToRow`, `rowToBlock` en `setBlock` functies.

Omdat het ook handig is om zo ver mogelijk te kunnen komen met een algoritme hebben we de functie `recCheck` gemaakt die recursief een functie aanroept op de sudoku tot de sudoku twee iteraties achtereen hetzelfde blijft.

Met deze functies hebben we een solver geschreven in twee stappen, namelijk `solve` en `solveWith`.

`solveWith` krijgt een lijst met functies die uitgevoerd worden op de sudoku. Zodra een algoritme geen nieuwe sudoku oplevert wordt er een moeilijker algoritme gebruikt. Als een algoritme wel een nieuw resultaat oplevert is er een kans dat er weer vakjes kunnen worden ingevuld en wordt er dus teruggegaan naar de simpelere algoritmes. Als zelfs het moeilijkste algoritme geen nieuwe resultaten geeft wordt het resultaat tot dan toe teruggegeven, dit kan ook de ingevulde sudoku zijn.

`solve` roept slechts `solveWith` aan met de lijst van checkers van bovengenoemde algoritmes met `recCheck` en de geprepareerde sudoku.