Sudo Ku  
Een sudoku solver voor Functioneel Programmeren, Opdracht 1.

De eerste opdracht voor het vak Functioneel Programmeren betrof het maken van een sudoku solver in een functionele programmeertaal. Door onze ervaring met de programmeertaal Haskell tijdens het practicum, is er ook gekozen om de solver in deze taal te schrijven.

De geschreven solver voldoet in elk geval aan de definitie die er wordt gegeven door het Van Dale woordenboek:

**su·do·ku** [sudoku] (dem; meervoud: sudoku's)  
puzzel waarbij cijfers (ook wel: letters) volgens bepaalde regels moeten worden ingevuld in een diagram [Van Dale, 2011]

Er zijn twee regelsets die ondersteund worden. Allereerst wordt de ‘standaard’ sudoku ondersteund, waarbij in een rij, kolom of 3x3-sector niet hetzelfde getal tweemaal voorkomt. Daarnaast wordt de ‘X’-sudoku ondersteund, waarbij bovendien in de diagonalen niet tweemaal hetzelfde getal voor mag komen.

## Model

Voordat er aan de implementatie werd begonnen, is een aantal keuzes gemaakt voor wat betreft de interface en implementatiecomplexiteit. Om het bord intern te representeren is er gekozen om gebruik te maken van een eenvoudige typeclass met de velden op het bord en het speltype. Een veld bestaat uit een x- en y-coördinaat, een waarde en de opties die er nog zijn voor het veld. Wegens efficiëntie is hier ook nog de sector opgeslagen waarin het veld zich bevindt; het is niet noodzakelijk om dit expliciet op te slaan, maar het vereenvoudigt het verdere gebruik van de velden wel.

## Interface

De interface ondersteunt, naast het invoeren van waarden op het bord, het opslaan en laden van spellen, het wisselen tussen speltypen, het oplossen van het bord en het geven van een hint (één zet).

Er wordt onder andere gebruik gemaakt van de modules FPPrac.Graphics (waaronder Graphics.Gloss) en FPPrac.Events. Door middel van een ‘event handler’ worden aanroepen vanuit de interface doorgestuurd naar de hoofdmodule. Er zijn twee van zulke handlers; één voor interne taken in de Graphics module (waaronder het opslaan en laden van de spellen) en één voor de spelspecifieke taken in de hoofdmodule (waaronder het oplossen van een spel).

## Core

Om een oplosstrategie te bepalen is er gekeken naar verschillende strategieën om een sudoku op te lossen. De meeste strategieën die te vinden zijn, maken gebruik van een bordperspectief. Omdat de implementatie de opties per veld bekijkt, is een aantal omgezet naar een veldperspectief. Elke strategie werkt de opties van een veld bij. Wanneer een strategie opties aan heeft gepast, is het noodzakelijk om alle strategieën nogmaals uit te voeren. De geïmplementeerde strategieën zijn:

* **Exclusion**: ingevulde cijfers in kolom, rij of sector worden uit de opties gehaald.
* **Singles**: als een cijfer alleen in de opties van dit veld voorkomt en niet in de kolom, rij of sector, zal deze als enige optie overblijven.

Deze strategieën zijn uitgebreid met een diagonaal-controle voor de ‘X’-sudoku.

Een oplossing wordt verkregen door de velden te sorteren op het aantal overgebleven opties en vervolgens door de lijst van onopgeloste velden te lopen. Wanneer een veld na het toepassen van de eerdergenoemde strategieën nog maar één optie heeft, wordt deze meteen ingevuld. Mochten er meerdere opties zijn, dan moet er helaas gebruik worden gemaakt van backtracking en zal de lijst van opties één voor één afgelopen worden. Als een optie geen positief resultaat oplevert, wordt deze niet gebruikt en wordt de volgende optie uitgewerkt. Door de manier van aanroepen en lazy evaluation in Haskell worden niet alle opties uitgewerkt als er meerdere oplossingen zijn.

Door het efficiënte gebruik van backtracking heeft het toevoegen van meerdere strategieën eerder een negatief dan positief effect op de oplossnelheid. Bij elke strategie is de afweging gemaakt tussen de kosten en baten van een implementatie ervan. Gebleken is dat zelfs een simpele strategie (zoals de singles-strategie) erg efficiënt geïmplementeerd moet worden om een snelheidswinst te behalen.

## Resultaat

Het blijkt dat de eenvoudigste sudoku’s zonder backtracking op te lossen zijn. Alleen voor de moeilijkere sudoku’s is dit vereist.

De oplostijden van enkele sudoku’s zijn hieronder weergegeven in seconden. Deze zijn verkregen door het meten van de tijd met een stopwatch. Het was helaas niet haalbaar om een timing in Haskell werkend te krijgen.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | Easy1 | Hard1 | Escargot2 | Minimal3 | Hardest4 |
| Normaal | 0.6 | 0.6 | 24.8 | 1.6 | 90.4 |
| ‘X’ | 1.6 | 4.2 | - | 4.7 | 96.1 |

1. Deze zijn willekeurig van internet gehaald. Er is daarom niet met zekerheid te zeggen hoe ‘easy’ of ‘hard’ deze sudoku’s dan wel niet zijn.
2. AI Escargot is een sudoku-puzzel gemaakt door een Finse wiskundige; "Escargot demands those tackling it to consider eight casual relationships simultaneously, while the most complicated variants attempted by the public require people to think of only one or two combinations at any one time."
3. Normaal: Dit is een willekeurige minimale sudoku met 17 hints.  
   ‘X’: Dit is een willekeurige minimale sudoku met 12 hints.
4. Omdat voorgaande tests ook door mensen zijn op te lossen is er ook een test gedaan met een sudoku die alleen door een computer gedaan kan worden.   
   Normaal: Deze is gevonden op Wikipedia en speciaal ontworpen om een computer er zo lang mogelijk over te laten doen.  
   ‘X’: Dit is een willekeurige sudoku met 16 hints die erg moeilijk blijkt te zijn.