Sudoku

Projectgroep 8

Bram Bonné (0623825)
Wim Leers (0623800)

Universiteit Hasselt
Academiejaar 2007-2008

9 juni 2008

Inhoudsopgave

1 Beschrijving

Het doel dat we wilden bereiken bij het maken van dit programma was om een applicatie te voorzien die zowel sudoku's kan genereren als ingelezen sudoku's kan oplossen, en dit volgens de regels van sudoku die vermeld worden op Wikipedia [8].

Enkele van de vereisten die we vooropgesteld hadden zijn een zo intuïtief mogelijke interface, platformonafhankelijkheid en hints. We hebben dit alles en meer kunnen verwezelijken, met volgende feature-lijst als resultaat:

- Vijf verschillende (intuïtief juist aanvoelende) moeilijkheidsgraden.
- Zeer efficiënte oplossings- en generatiealgoritmen.
- Threading, zodat de UI ook responsief blijft gedurende de korte tijd dat er een bord gegenereerd wordt.
- Tijdelijke keuzes, zodat de gebruiker zelf kan ingeven welke zetten nog mogelijk zijn.
- Hints, waardoor voorgaande tijdelijke keuzes automatisch worden aangeduid.
- Een timer.
- Opslaan en inlezen van sudoku borden in het `csv` formaat.
- Opslaan en inlezen van een volledig sudoku spel (met de huidige spelstatus, alle gemaakte zetten en de HUD-instellingen).
- Volledig vertaalbaar programma (er wordt gekeken naar de systeemtaal om automatisch de juiste taal te kiezen).
- Oplossen van ingelezen spelborden (alsook gegenereerde borden) met behulp van twee algoritmen.
- Visuele feedback voor geldigheid en oplosbaarheid van het bord (dit kan uiteraard ook uitgeschakeld worden).
- Mogelijkheid tot afdrukken van de sudoku.
- Navigatie door middel van het toetsenbord is doorheen de gehele applicatie mogelijk (uiteraard kan dit ook door gebruik te maken van de muis).
- Exporteren van een sudoku naar een afbeeldingsbestand.
- Platformonafhankelijkheid: werkt op Windows, Mac OS X en Linux (en er zijn geautomatiseerde build scripts voor al deze platformen). Zie ook figuur 1 op de volgende pagina.
- Doxygen documentatie in alle klassen.

Figuur 1: Cross-platform Sudoku: Mac OS X, Linux (Ubuntu) en Windows

2 Handleiding

2.1 Builden

2.1.1 Linux

Er wordt een build script meegeleverd voor Linux: `build_linux.sh`. Indien u dit script uitvoert, wordt Sudoku in release mode gebuild en wordt er een .tar.gz archief van gemaakt. De enige vereiste is dat de Qt development libraries geïnstalleerd zijn.

2.1.2 Mac OS X

Ook voor Mac OS X wordt er een build script meegeleverd: `build_mac.sh`. Wanneer u dit script uitvoert, wordt Sudoku in release mode gebuild als een *Universal Binary*[1] (d.w.z. voor zowel de x86 als de PPC CPU architecturen), worden de Qt frameworks (als dylibs, dynamic libraries) in de .app package gekopieerd en wordt er tenslotte een `.dmg` disk image aangemaakt (het meest gebruikte formaat voor applicatie distributie op Mac OS X).

2.1.3 Microsoft Windows

Tenslotte is er ook voor Windows een build script voorzien: `build_windows.bat`. Dit script (in feite een *batch file*) compileert Sudoku in release mode (er wordt gebruik gemaakt van de command-line variant van Visual Studio) en daarna worden alle benodigde .dll's gekopieerd naar dezelfde map. Er wordt niet automatisch een archief of installer aangemaakt omdat deze niet standaard beschikbaar zijn op Windows. Het probleem met deze versie is dat het Qt proces na afsuiten van het programma blijft runnen (dit is een bug in Qt 4.3.2). Aangezien we voor Qt 4.4 , hebben we dus ook een script geschreven voor de MingW compiler (`build_windows_mingw.bat`), waarin dit probleem is opgelost.

[1] http://developer.apple.com/macosx/adoptinguniversalbinaries.html

2.2 Installatie

2.2.1 Linux

In Linux is het ongebruikelijk dynamische libraries te bundelen met het programma (in tegenstelling tot Mac OS X en Windows dus), maar wordt er veelvuldig gebruik gemaakt van package managers. De package manager moet gebruikt worden om de Qt libraries te installeren. Op Ubuntu wordt de *APT* [2] package-manager gebruikt, daar moet u dit commando uitvoeren: `sudo apt-get install libqt4-core`. Dit is enkel nodig indien Qt nog niet geïnstalleerd is. Op alle Linux-distributies die KDE [3] gebruiken als desktop, zal Sudoku out-of-the-box werken.

2.2.2 Mac OS X

De installatie op Mac OS X werkt net zoals eender welke andere Mac applicatie: het programma verslepen naar de `/Applications` directory en je bent klaar.

2.2.3 Microsoft Windows

Dankzij het build script worden alle `.dll`'s al meegeleverd. Je kan `Sudoku.exe` dus om het even waar plaatsen, zolang je de `.dll`'s maar meekopieert.

2.3 Spelen van een spel

2.3.1 Spelregels

> Een sudoku is een puzzel van negen bij negen vakjes met een klein aantal reeds ingevulde enkelvoudige cijfers. De kunst is de overige vakjes ook in te vullen op zo'n manier dat in elke horizontale lijn én in elke verticale kolom de cijfers 1 tot en met 9 één keer voorkomen. Bovendien is de puzzel onderverdeeld in negen blokjes van drie bij drie, die elk ook weer eenmaal de cijfers 1 tot en met 9 moeten bevatten.
>
> http://nl.wikipedia.org/wiki/Sudoku

Dit kan gedaan worden door voor een vakje alle getallen de schrappen die reeds voorkomen in zijn rij, kolom of blokje. We selecteren dan een getal definitief wanneer het het enige overblijvende getal in zijn vakje is. Het spel is afgelopen wanneer het bord volledig gevuld is.

[2] http://en.wikipedia.org/wiki/Advanced_Packaging_Tool
[3] http://www.kde.org/

Linux/Windows	Mac OS X	Actie
`Ctrl` + <getal>	`Alt` + <getal>	Duid <getal> aan als één van de tijdelijke keuzes
<getal>	<getal>	Duid <getal> aan als definitieve keuze
`Backspace`	`Backspace`	Haal een definitieve keuze van het bord
H	H	Geef alle mogelijke keuzes (hints) voor het huidige vakje
P	P	Pauzeer het spel
`Alt` + S	`Alt` + S	Laat de oplosbaarheid van het bord zien (of verberg hem terug)
`Alt` + v	`Alt` + v	Laat de geldigheid van het bord zien (of verberg hem terug)
`Ctrl` + F	`Cmd` + F	Wisselen tussen volledig scherm/venster
`Ctrl` + Q	`Cmd` + Q	Afsluiten

Tabel 1: Lijst van sneltoetsen

2.3.2 Interface

De Sudoku kan opgelost worden door het gebruik van de muis of het toetsenbord.

Bij het gebruik van de muis wordt standaard een getal als tijdelijke keuze aangeduid wanneer er op geklikt wordt binnen zijn vakje. Om een tijdelijke keuze ongedaan te maken moet gewoon dit getal opnieuw aangeklikt worden. Als er gedubbelklikt wordt op dit getal wordt het geregistreerd als definitieve keuze, die weer kan ongedaan gemaakt worden door opnieuw te dubbelklikken.

Bij gebruik van het toetsenbord kan er genavigeerd worden met de pijltjestoetsen. Tijdelijke keuzes kunnen aangeduid worden of ongedaan gemaakt worden door de `Ctrl` toets (`Alt` onder Mac OS X) in combinatie met het getal in te drukken. Voor permanente keuzes is dit gewoon het getal zelf.

Buiten het navigeren zijn er ook nog enkele andere toetsenbord sneltoetsen, waarvan de meest belangrijke[4] worden opgesomd in tabel 1.

Behalve het spelen van het spel zelf kan er ook nog geëxporteerd worden naar een aantal formaten. Het verschil tussen het opslaan en het exporteren van een bord is dat bij het opslaan de volledige spelinfo wordt weggeschreven (zoals de speeltijd), terwijl er bij het exporteren via een WYSIWYG strategie enkel het spelbord wordt weggeschreven (of naar de printer gestuurd).

3 Programmastructuur

In deze sectie bespreken we kort de gebruikte datastructuren en algoritmen. Het is echter handig om ook de Doxygen documentatie[5] bij de hand te houden, aangezien al onze klassen Doxygen documentatie bevatten.

3.1 UML

De UML diagrammen kunnen gevonden geworden op pagina's 8 - 11. Ook figuur 3.2.7 op pagina 12 geeft nog een mooi overzicht van hoe de Qt klassen

[4] De rest kan gevonden worden in het menu zelf, naast de desbetreffende keuze.
[5] Meegeleverd in de map *doc*.

met elkaar verbonden zijn.

3.2 ADT's

In deze sectie zullen we de klassenstructuur van ons programma uitleggen, waarbij we enkele belangrijke functies vermelden. Indien verdere uitleg van deze functies gewenst is, verwijzen we u graag door naar de Doxygen documentatie, waarin voor elke functie een korte beschrijving voorzien is.

3.2.1 De Board klasse

De `Board` klasse houdt in eerste instantie een sudoku puzzel bij, waarbij geen onderscheid gemaakt wordt tussen elementen die vooraf gegenereerd werden en elementen die door de gebruiker ingevuld werden. Buiten een representatie van het bord biedt deze klasse ook nog een aantal interessante functies zoals `IsValid()`, een method die kijkt of we op een geldig bord werken, en `IsValidMove()`, die ons vertelt of een zet geldig is ervan uit gaande dat het bord dat reeds is.

Verder bevat deze klasse ook nog alle mogelijke manieren om het bord weg te schrijven en in te lezen (zie ook sectie 3.4 op pagina 31) en enkele functies om het bord gemakkelijker aan te spreken en te kopiëren.

3.2.2 De Sudoku klasse

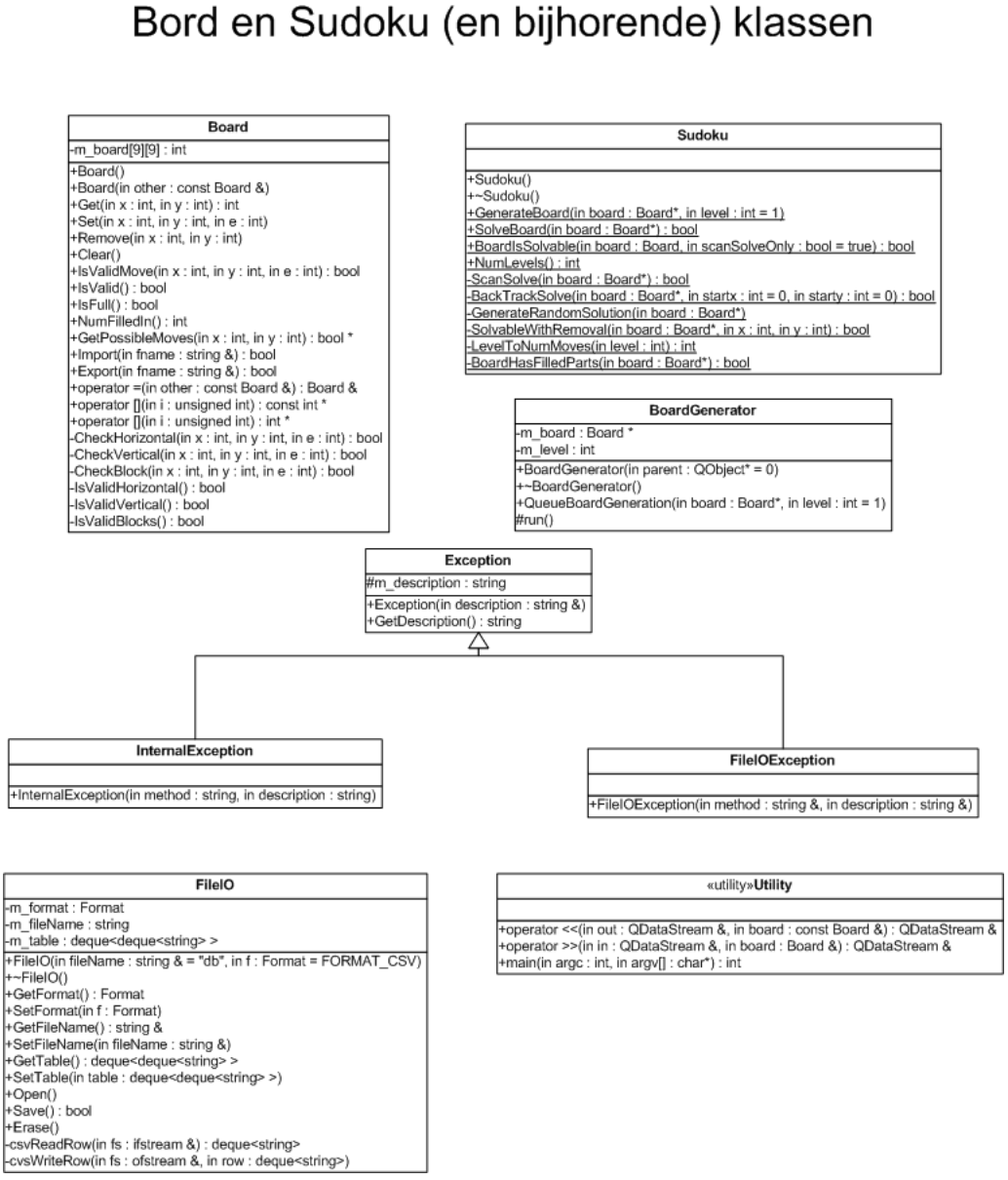
Deze klasse wordt nooit geïnstantieerd (bevat enkel statische functies) maar wordt enkel gebruikt voor de berekeningen, waarbij het telkens een pointer of referentie naar of een kopie van het bord meekrijgt. De klasse bevat enkel algoritmes, die verder behandeld worden in sectie 3.3 op pagina 22. We vermelden hier dan ook enkel dat `SolveBoard()` en `BoardIsSolvable()` verder de memberfuncties `ScanSolve()` en `BacktrackSolve()` aanroepen.

3.2.3 De PositionElement klasse

De `PositionElement` klasse wordt vooral gebruikt door de algoritmes die zich in de `Sudoku` klasse bevinden. Hij houdt simpelweg de waarde van een element bij, samen met zijn positie op het bord.

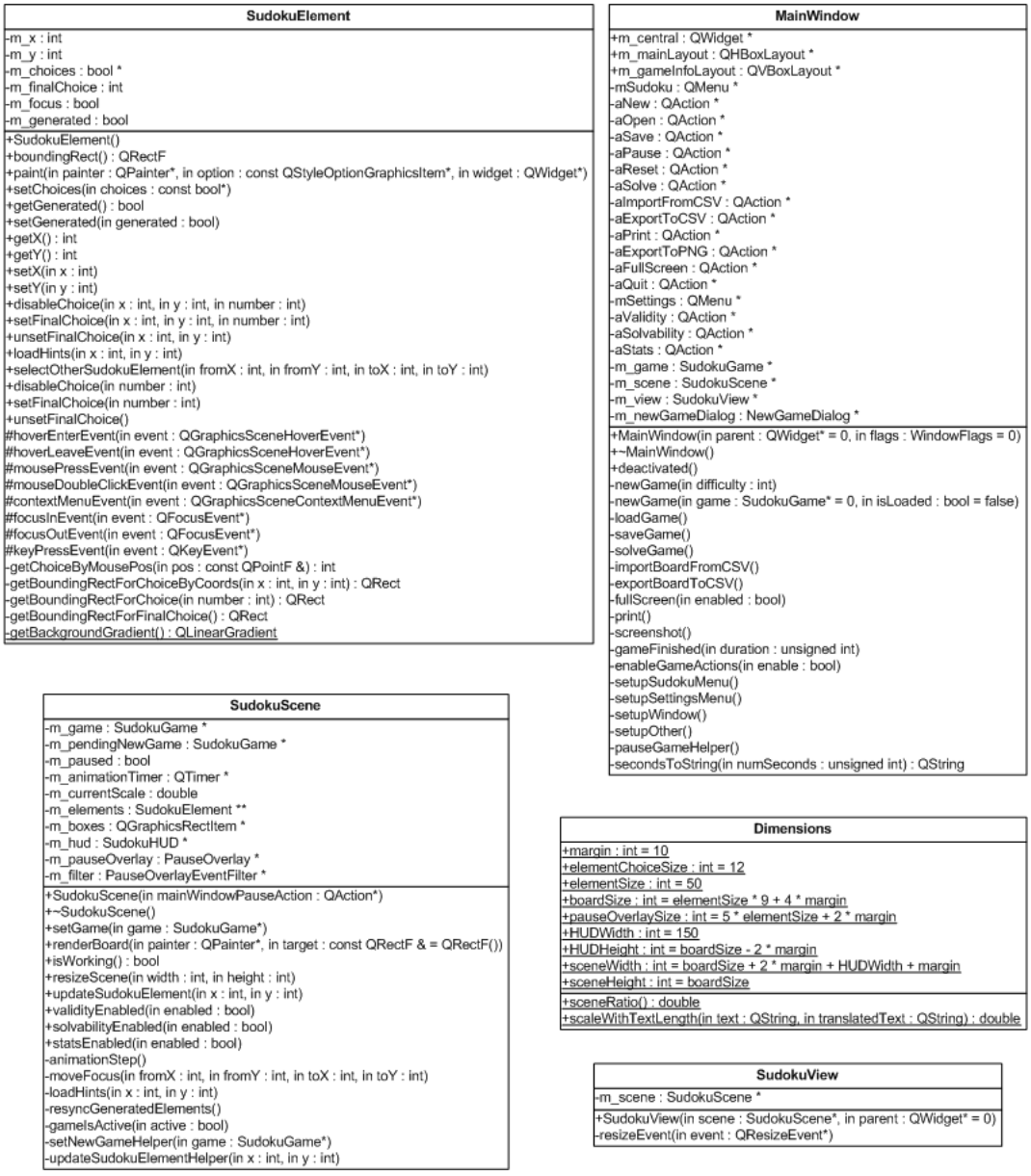
3.2.4 De BoardGenerator klasse

Dit is in feite een wrapper-klasse voor `Sudoku::GenerateBoard()`, om ervoor te zorgen dat deze functie uitgevoerd wordt in een apart (*worker*) thread. Zodra een bord gegenereerd is, stopt het thread vanzelf met bestaan. Er wordt ook bijgehouden hoelang er precies gedaan wordt over het genereren van een bord. Als je het programma via de terminal start kan je deze tijden dan ook zien (zie listing 1 op pagina 12).



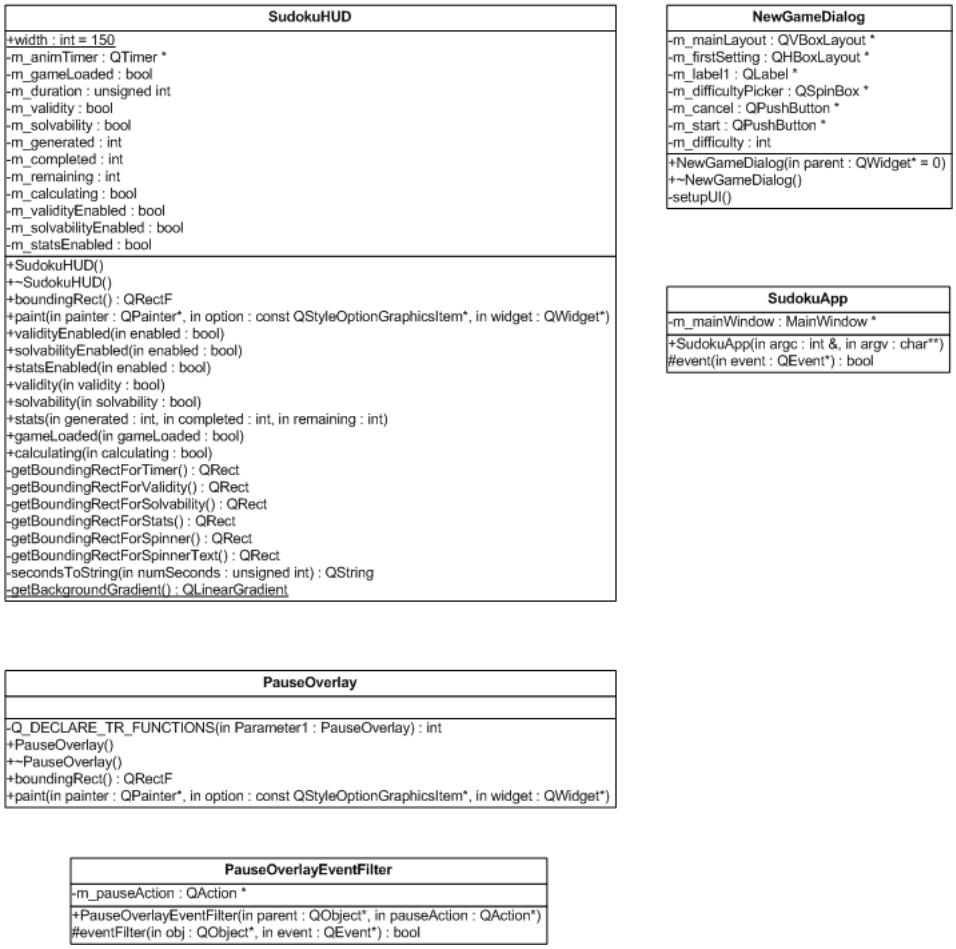
Figuur 2: UML deel 1

Qt klassen (deel 1)



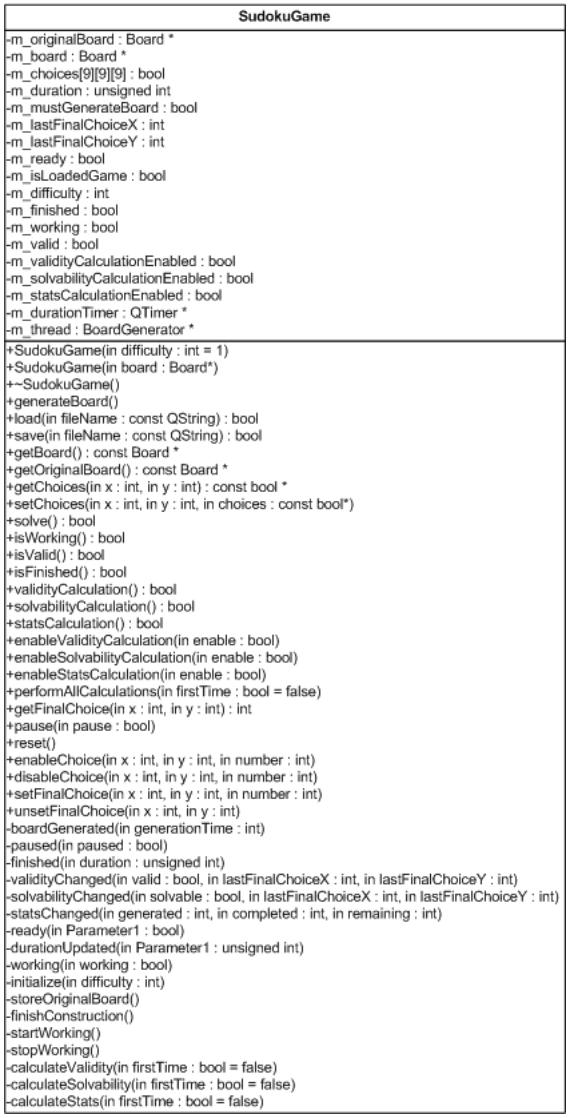
Figuur 3: UML deel 2

Qt klassen (deel 2)



Figuur 4: UML deel 3

Qt klassen (deel 3)



Figuur 5: UML deel 4

Listing 1: De tijd nodig om een nieuw bord te genereren is zichtbaar in de terminal.

```
Start generating new board...
Finished generating new board in worker thread in 18 ms.
Game is ready!
Start generating new board...
Finished generating new board in worker thread in 12 ms.
Game is ready!
Start generating new board...
Finished generating new board in worker thread in 63 ms.
Game is ready!
```

3.2.5 De FileIO klasse

Deze klasse herbruiken we uit ons vorige project [13]. Hij zorgt voor het wegschrijven en inlezen van het spelbord in het `csv` bestandsformaat. Deze klasse wordt gebruikt door de `Import()` en `Export()` methods van de `Board` klasse.

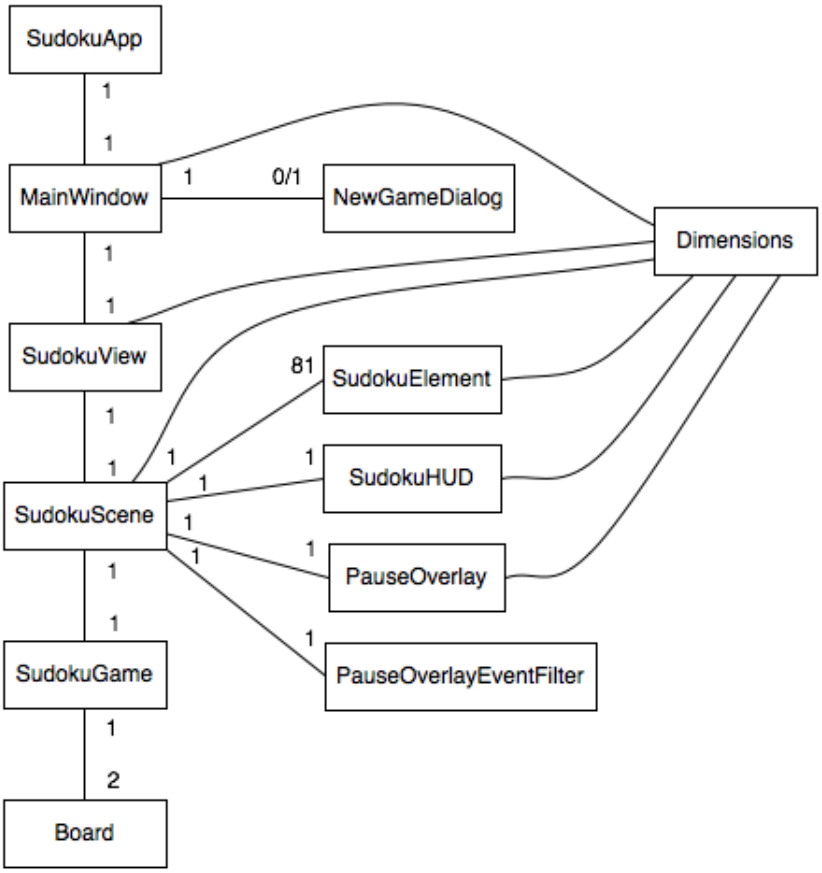
Voor meer informatie over de bestandsformaten verwijzen we door naar sectie 3.4 op pagina 31.

3.2.6 De Exception klassen

Deze drie klassen (`Exception`, met daarvan afgeleid `FileIOException` en `InternalException`) worden gebruikt om excepties af te handelen.

3.2.7 GUI architectuur

Om meteen een duidelijk idee te geven van de klasse-opbouw van de GUI, is het handig om met een diagram te beginnen:



Figuur 6: GUI architectuur diagram

De `SudokuApp` klasse is een afgeleide van `QApplication` en is dus het startpunt van de hele applicatie. Het bevat één `MainWindow` klasse die onder andere het hoofdvenster en de menu's onderhoudt. Wanneer de gebruiker een nieuw spel start wordt er een `NewGameDialog` object aangemaakt. Dit is een modal dialog. Bijgevolg wordt de gebruiker verplicht om eerst dit venster te behandelen voordat hij verdergaat met andere acties in het programma. Daarom is het ook logisch dat dit object enkel gedurende zijn gebruiksduur bestaat: als de gebruiker klaar is met dit dialoogvenster, wordt het object weer vernietigd.

Dan komen we nu bij het interessante gedeelte. De `MainWindow` klasse bevat één `SudokuView`, één `SudokuScene` en één `SudokuGame` object en is de eigenaar van al deze objecten. Maar een `SudokuView` object heeft uiteraard een `SudokuScene` object nodig om iets zinnig te kunnen weergeven, dus wordt een pointer naar het `SudokuScene` object meegegeven. Op zijn beurt kan het `SudokuScene` object slechts een Sudoku spel weergeven indien er een `SudokuGame`

object is waaruit het de nodige informatie kan halen. Wanneer een nieuw spel gestart wordt, blijven de `SudokuView` en `SudokuScene` objecten onaangetast: er wordt simpelweg een nieuwe pointer naar een `SudokuGame` object ingesteld in het `SudokuScene` object.

De `SudokuScene` bevat 81 (één voor ieder vakje) `SudokuElement` objecten, één `SudokuHUD` object (waarin de timer, de geldigheid, oplosbaarheid en statistieken van de huidige puzzel getoond worden). Ook bevat het een `PauseOverlay` object en een `PauseOverlayEventFilter` object. Daarover meer in secties 3.2.18 en 3.2.19 op pagina 22; voorlopig is het genoeg om te weten dat deze beide objecten gedurende de hele levenstijd van het `SudokuScene` object ook bestaan, maar slechts gebruikt worden wanneer nodig.

Ten slotte dan nog de `SudokuGame` klasse: deze bevat alle spellogica (minus de algoritmes, want die zitten in de `Sudoku` klasse als static methods).

3.2.8 SudokuApp

`SudokuApp` is een afgeleide van de `QApplication` klasse. Deze klasse afleiden is in de meeste applicaties overbodig, maar in ons geval was het noodzakelijk: we wilden de applicatie automatisch pauzeren indien de focus van de gebruiker naar een andere applicatie wisselt. Om dit te bereiken, overriden we het `event()` slot, zodat de gewenste reactie op het `QEvent::ApplicationDeactivate` event kan worden voorzien.

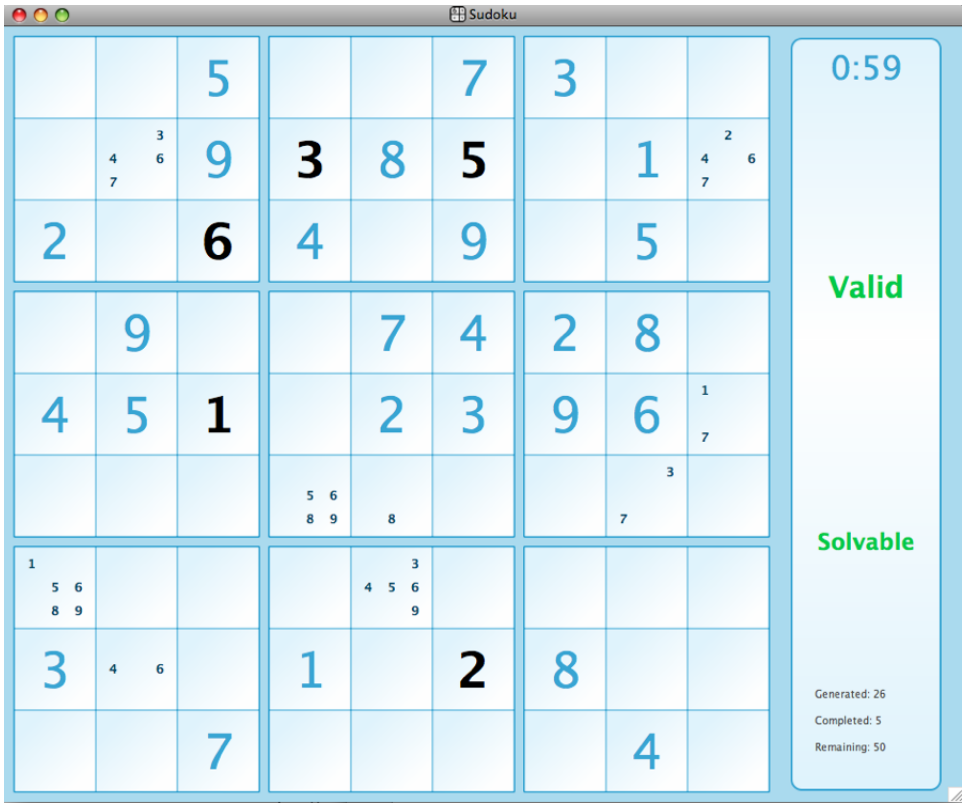
Jammer genoeg bleek achteraf dat dit event niet ondersteund wordt op Linux, of althans niet door het X Window System[6] (dat alomtegenwoordig is in Linux distributies), in tegenstelling tot wat de documentatie beweert [14]. We gebruiken het dus enkel onder Mac OS X en Windows.

3.2.9 Dimensions

Deze klasse bevat uitsluitend static public members. Er zijn geen variabelen, enkel constanten. Deze constanten bepalen de *verhoudingen* van de GUI. Zo is het dus mogelijk om de GUI er compleet anders te laten uitzien zonder enige verandering in een ander bestand (voor een voorbeeld, zie figuur 7). *Iedere* klasse die iets voorziet dat zichtbaar is voor de gebruiker, maakt gebruik van deze constanten!

Er zijn ook 2 statische methods: eentje om de aspect ratio op te vragen (dit kan niet in een constante omdat een `static const double` initializer niet toegelaten wordt door de `g++` compiler), en eentje om een tekst lengte ratio te verkrijgen. Deze laatste method wordt gebruikt om de grootte van tekst automatisch te schalen naargelang de lengte van de vertaling. Dit wordt bijvoorbeeld gebruikt in de `PauseOverlay` klasse: "Paused" wordt "Gepauzeerd" in het Nederlands en zonder herschaling zou dit niet passen.

[6] http://en.wikipedia.org/wiki/X_Window_System



Figuur 7: Dimensions::elementSize 80 i.p.v. 50 pixels. Alle andere verhoudingen zijn onaangetast; het geheel rendert nog steeds mooi. Zie figuur 14 op pagina 27 ter vergelijking.

3.2.10 MainWindow

Deze klasse dirigeert de werking van de GUI: ze bevat het hoofdvenster (zoals de naam al aangeeft), start een `NewGameDialog` wanneer nodig, bevat `SudokuView`, `SudokuScene` en `SudokuGame` objecten en leidt alle signalen in goede banen naar de juiste ontvangers.

Het `SudokuView` object wordt als het central widget ingesteld, waardoor er geen gebruik moet gemaakt worden van `QLayout` of afgeleides daarvan. Zo bekomen we ook een zeer speelse GUI, het is tenslotte geen kantoorsoftware.

3.2.11 NewGameDialog

Dit is een van de eenvoudigste klassen in het hele project. Deze klasse is afgeleid van `QDialog` en voorziet een `newGame(int difficulty)` signal, waarmee de `MainWindow` klasse de gewenste moeilijkheidsgraad ontvangt. Het is een modal dialog, waardoor de gebruiker de rest van de applicatie niet meer kan gebruiken totdat hij deze dialoog afgehandeld heeft. De `Qt::Sheet` window flag [2] is ingesteld om de applicatie meer als een standaard Mac applicatie te laten aanvoelen. Dankzij deze flag schuift de dialog uit het hoofdvenster.

3.2.12 Model-View architectuur

Het eigenlijke spel wordt beheerd door de `SudokuView`, `SudokuScene` en `SudokuGame` klassen (en de klassen waar deze 3 op steunen uiteraard). Samen vormen ze een implementatie die dicht bij het *Model-View-Controller* design pattern [7] aanleunt. Het is geen strikte implementatie daarvan, want in Qt definieer je typisch widgets (*view*) én maak je tegelijk de verbindingen die reacties op de input van de gebruiker realiseren (*controller*). Het is dus vrij logisch dat je een mix krijgt van view en controller. Men spreekt dan van het *Model-View* design pattern[8]. Qt heeft echter wel een extra abstractie om meerdere instanties van views (meerdere views die dezelfde scene observeren[9]) mogelijk te maken, maar daar maken wij geen gebruik van. Een voorbeeld hiervan zou een voor- en zijaanzicht van dezelfde scene kunnen zijn.

In ons geval is `SudokuGame` het *model* (bevat de business logic) en `SudokuScene` de *view*.

3.2.13 SudokuView

Dit is een eenvoudige afgeleide van de `QGraphicsView` klasse, maar accepteert enkel een `SudokuScene` object in tegenstelling tot het standaard gedrag, dat eender welk `QGraphicsScene` object aanvaardt. Wat we wel even willen opmerken is dat alle render hints en alle optimalisatie flags zijn ingeschakeld. Dat laatste

[7] http://en.wikipedia.org/wiki/Model-view#Pattern_description

[8] http://doc.trolltech.com/4.3/model-view-introduction.html#the-model-view-architecture

[9] http://doc.trolltech.com/4.3/graphicsview.html#the-graphics-view-architecture

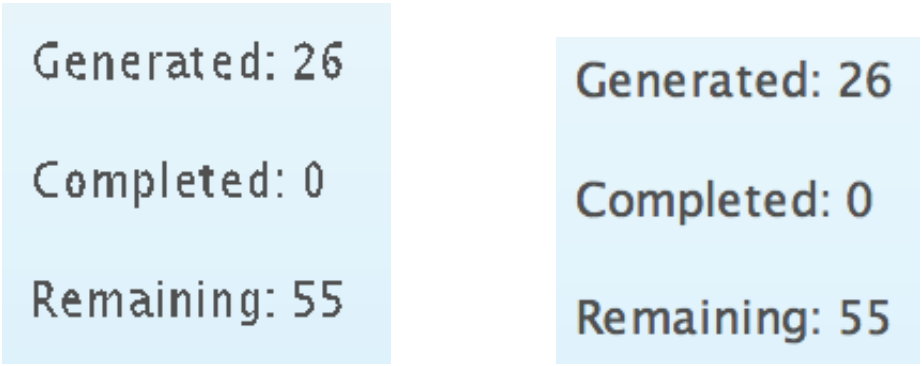
impliceert dat alle code die op het scherm tekent zeer juist geïmplementeerd is, anders zouden er artifacts optreden. Een korte uitleg bij iedere flag:

1. `QGraphicsView::DontClipPainter`: schakelt alle impliciete (automatische) clipping uit. Dit vereist dat je code ten alle tijde binnen de *bounding rectangles* tekent.

2. `QGraphicsView::DontSavePainterState`: `QGraphicsView` beschermt de toestand van de painter (een `QPainter` object), door telkens de toestand voor het aanroepen van de tekencode te herstellen na de aanroep. Dit vereist dat je de painter toestand telkens zelf correct instelt.

3. `QGraphicsView::DontAdjustForAntialiasing`: wanneer anti-aliasing is ingeschakeld (wat bij ons het geval is), kan het zijn dat er stukjes buiten de bounding rectangles getekend worden. De `QGraphicsView` klasse lost dit op door de bounding rectangles in alle richtingen met 2 pixels uit te breiden. Dat betekent uiteraard dat er ook telkens meer werk moet gebeuren. Nadat we deze flag inschakelden, kon er op slechts één plaats een artifact gevonden worden (in de animatie van `SudokuHUD`), en dat is opgelost door een gelijkaardige techniek. Het verschil is dat er nu slechts één item met een grotere bounding rectangle getekend wordt!

Daarnaast is er nog één zeer opmerkelijke eigenschap van deze klasse: op Windows wordt hij gerenderd door middel van OpenGL. Op Mac OS X en Linux wordt geen OpenGL gebruikt omdat het er ten eerste significant trager werkt en er ten tweede een hoop artifacts voorkomen. Ook zien de lettertypen er erg slecht uit (zie figuur 8). Dit is dus een ander voorbeeld van een platform-specifieke optimalisatie.



(a) Mét OpenGL (b) Zonder OpenGL

Figuur 8: Renderen met/zonder OpenGL op Mac OS X.

3.2.14 SudokuScene

De spelweergave wordt verzorgd door deze klasse. `QGraphicsScene` is de base class, zoals de naam al doet vermoeden. In een `QGraphicsScene` horen `QGraphicsItem` objecten thuis: deze items zijn de objecten die je in de scene kunt zien en waarmee je kan interageren. Om geavanceerdere objecten te maken, moet je echter zelf afleiden van deze klasse en dan zelf gaan tekenen (d.m.v. `QPainter`). Zo hebben we voor dit spel 3 klassen afgeleid van `QGraphicsItem`:

1. `SudokuElement`: om één element (puzzelvakje) in een sudoku puzzel voor te stellen (zie sectie 3.2.15 op de pagina hierna)
2. `SudokuHUD`: deze toont alle informatie i.v.m. het huidige spel (zie sectie 3.2.16 op pagina 20)
3. `PauseOverlay`: dit item wordt toegevoegd telkens het spel gepauzeerd is en wordt daarna weer vanzelf verwijderd (zie sectie 3.2.18 op pagina 21)

Deze klasse bevat de volgende items (zie ook opnieuw figuur 6 op pagina 13):

1. 81 `SudokuElement` objecten, één per puzzelvakje
2. 9 `QGraphicsRectItem` objecten, één per groep van 9 puzzelvakjes (dit is dan ook meteen de enige door Qt meegeleverde `QGraphicsItem`-afgeleide die gebruikt wordt)
3. 1 `SudokuHUD` object
4. 1 `PauseOverlay` object; dit wordt enkel echt toegevoegd aan de scene wanneer het nodig is, dus wanneer het spel gepauzeerd is. Zoals de naam al doet vermoeden, heeft het `PauseOverlayEventFilter` object hier veel mee te maken. Zie respectievelijk secties 3.2.18en 3.2.19 op pagina 22 voor meer uitleg over deze klasses.

De ongeveer honderd items [10] in de scene worden – indien je hier niet specifiek rekening mee houdt – bij iedere update aan de scene, allemaal hertekend. Dat is natuurlijk zeer inefficiënt. Gelukkig bevat Qt al het wapen om de strijd mee aan te gaan: `QGraphicsScene->update(QRectF(x, y, w, h))`. Daarbij moet je natuurlijk wel op ieder moment accuraat kunnen berekenen wat de bounding rectangles zijn van ieder stukje dat kan geüpdatet worden. Bovendien is onze GUI geheel herschaalbaar, wat dus impliceert dat de positie en grootte van íeder item berekend wordt a.h.v. de `Dimensions` klasse (3.2.9), anders zou het stoppen met werken.

Maar ook de `QGraphicsItem->update(QRectF(x, y, w, h))` method bestaat, dus binnen een item kan óók slechts een stuk hertekend worden. Bijvoorbeeld bij de animatie in de `SudokuHUD` klasse (3.2.16)wordt dit gebruikt om bij de geanimeerde vooruitgangsindicator enkel dat stuk te hertekenen. Dit leverde een performance boost op van ongeveer 60%!

[10] http://labs.trolltech.com/blogs/2006/08/30/4-million-items-is-it-really-possible

Listing 2: Scene resizing algoritme dat de aspect ratio behoudt

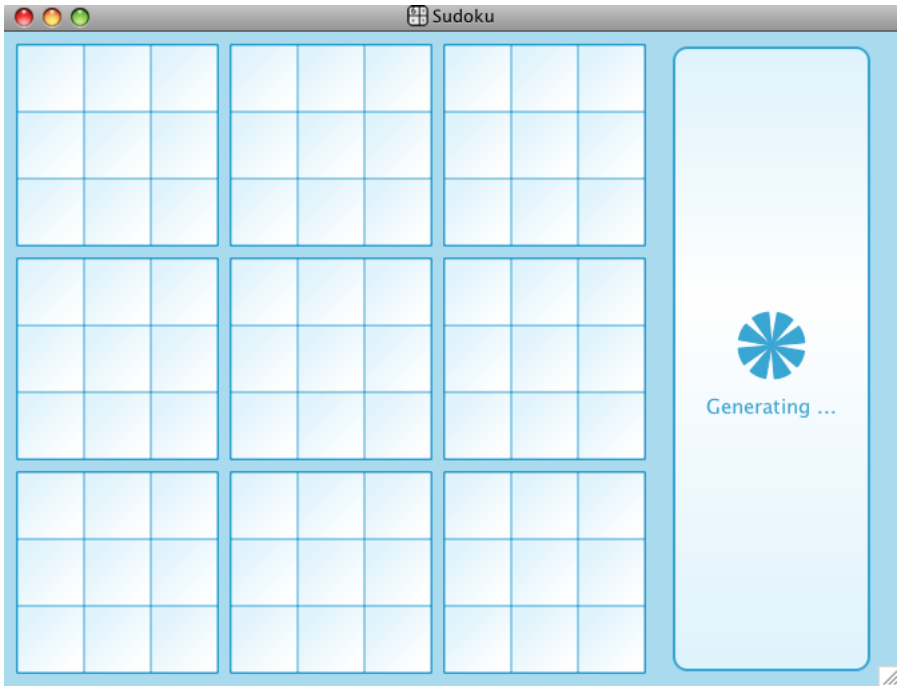
```
void SudokuScene::resizeScene(int width, int height) {
    setSceneRect(0, 0, width, height);

    // If the new aspect ratio (width to height) is greater than the initial
    // one, we must scale using width, otherwise using height, to maintain
    // the aspect ratio of the scene (and as a result to keep all items of the
    // scene visible).
    double scale;
    if ((double) width / height > Dimensions::sceneRatio())
        scale = (double) height / Dimensions::sceneHeight;
    else
        scale = (double) width / Dimensions::sceneWidth;

    // Store in CPU registry because this will be used to scale *every* item
    // in the scene!
    register double scaledScale = scale / m_currentScale;

    QGraphicsItem * item;
    foreach (item, this->items()) {
        item->setPos(item->pos() * scaledScale);
        item->scale(scaledScale, scaledScale);
    }

    m_currentScale = scale;
}
```

Je zou verwachten dat in een klasse als deze geen speciale berekeningen zitten. `SudokuScene` is echter een uitzondering: de sudoku applicatie laat namelijk toe om het venster te resizen. Dat klinkt niet buitengewoon, maar schijn bedriegt: een gewone resize functie zou geen rekening houden met de aspect ratio. De `resizeScene(int width, int height)` method die je in listing 2 vindt doet dat echter wel, en is slechts een twintigtal regels groot, maar zeer krachtig!

3.2.15 SudokuElement

Deze klasse is afgeleid van `QGraphicsItem`, maar wordt daarvoor nog afgeleid van `QObject`. Er is communicatie nodig tussen instanties van deze klasse en de scene en daarvoor moet er altijd eerst afgeleid worden van `QObject`. Een `QGraphicsItem` is standaard een klasse zonder interactie: het bevat enkel code om te tekenen. Er wordt dus door Qt verondersteld dat de interactie gebeurt op scene niveau, maar in ons is het veel handiger om dat in deze klasse te doen: anders zou de `SudokuScene` klasse onhandelbaar groot worden.

Deze klasse meer in detail bespreken zou langer zijn dan de implementatie te lezen, omdat er zoveel verschillende dingen gebeuren. Hover-, keypress-, mousepress- en focusevents worden ondersteund. Er zijn verschillende *states* die worden bijgehouden, om zo het element op de juiste manier te tekenen. De interactie met het spel verloopt grotendeels via deze klasse en het is dus ook dankzij deze klasse dat dat zo intuïtief gaat.



Figuur 9: De animatie in `SudokuHUD`.

3.2.16 SudokuHUD

Ook deze klasse wordt eerst afgeleid van `QObject` en daarna pas van `QGraphicsItem`, om dezelfde reden als de `SudokuElement` klasse. Ze zorgt enkel voor het tonen van de huidige toestand van het spel. Toch heeft ook deze klasse enkele noemenswaardige features.

Om de kader met afgeronde hoeken te tekenen, moest er gebruik gemaakt worden van een `QPainterPath`. `QPainter::DrawRoundRect()` is nutteloos omdat het enkel vierkanten met ronde hoeken kan tekenen. Als je er een rechthoek van maakt, krijg je een ellips! Dit is opgelost in Qt 4.4, waar deze method verwijderd is en `QPainter::DrawRoundedRect()[15]` is toegevoegd ("Rounded" versus "Round").

Daarnaast is er ook een animatie (zie figuur 9), die getoond wordt wanneer een bord gegenereerd wordt (in een apart thread, zie sectie 3.2.4 op pagina 7). Zo is het op alle momenten duidelijk voor de gebruiker dat de applicatie niet gestopt is met werken. Animaties in Qt zijn mogelijk door een `QTimer` en te combineren met een statische variabele (om bij te houden wat de vorige positie was).

3.2.17 SudokuGame

Kort samengevat: deze klasse bevat alle logica om het spel in de juiste banen te leiden. Het is het *model* in de Model-View architectuur (zie 3.2.12 op pagina 16).

Maar ook deze klasse heeft enkele opmerkelijke eigenschappen. Zo zijn er bijvoorbeeld 2 manieren om een instantie van dit object aan te maken:

1. `SudokuGame::SudokuGame(int difficulty)`: deze constructor wordt gebruikt om nieuwe spellen aan te maken, van de opgegeven moeilijkheidsgraad. De constructor zelf maakt echter enkel een *leeg* bord aan. Het is aan de programmeur om te bepalen wanneer het echte bord gegenereerd moet worden, door `SudokuGame::start()` aan te roepen. Deze maakt een nieuw bord aan door `QueueBoardGeneration(m_board, m_difficulty)` aan te roepen, waardoor het genereren van het bord in het `BoardGenerator` object en dus in een ander thread gebeurt (een `QThread` is een afgeleide van `QObject`[3]). Wanneer het bord gegenereerd is, zendt het `BoardGenerator` object een `boardGenerated(int generationTime)` signaal uit, welke wordt opgevangen in de `SudokuGame` klasse. Vervolgens stopt de `SudokuGame` klasse de animatie in de `SudokuHUD` (dmv het `working(bool)` signaal) en stuurt het het `SudokuGame::ready(bool)` signaal uit. Het spel is dan klaar om gespeeld te worden.

2. `SudokuGame::SudokuGame(Board * board)`: deze constructor is een heel stuk eenvoudiger dan in gebruik dan de vorige, omdat er geen threading aan te pas komt. Hij ontvangt namelijk gewoon een gegenereerd bord, en wordt dus gebruikt bij het importeren van bestaande borden, bijvoorbeeld om `csv` bestanden (zie sectie 3.4.1 op pagina 31) in te lezen.

Omdat Sudoku slechts door 1 persoon tegelijk gespeeld wordt, is de spellogica zelf zeer eenvoudig: zolang het bord niet volledig én geldig is ingevuld, is het spel niet gedaan. We moeten dus enkel de `Board::IsFull()` en `Board::IsValid()` methods aanroepen om dit te controleren.

Ten slotte is de rest van deze klasse voornamelijk gevuld met het omzetten en opvangen van signals en slots, het berekenen van de door de gebruiker gewenste informatie die in de `SudokuHUD` komt (maar wel énkel de informatie die ook daadwerkelijk gebruikt wordt!), en het aan `SudokuScene` melden van veranderingen op het spelbord.

3.2.18 PauseOverlay

De `PauseOverlay` klasse is een simpele afgeleide van `QGraphicsItem`. Er stelde zich echter een probleem: we wilden onze applicatie vertaalbaar maken, maar de `tr()` functie is enkel beschikbaar in afgeleides van `QObject` en `QCoreApplication`. Gelukkig voorziet Qt ook hier een oplossing: door de macro `Q_DECLARE_TR_FUNCTIONS(PauseOverlay)` toe te voegen (vergelijkbaar met de `Q_OBJECT` macro), is de `tr()` functie beschikbaar binnen de klasse alsof er niets aan de hand is.

3.2.19 PauseOverlayEventFilter

Een event filter dient, zoals de naam al zegt, om events te filteren: je kan kiezen welke events je een speciale behandeling wil geven en welke door de filter heen geraken (en welke dus op de normale manier verder afgehandeld worden).

De `PauseOverlayEventFilter` wordt geïnstalleerd op de scene (door `SudokuScene` zelf) telkens het spel gepauzeerd wordt. Dit voorkomt dat input van toetsenbord en muis de `SudokuElement` objecten bereikt. Slechts door een dubbelklik (opgevangen door `PauseOverlayEventFilter` zelf) of door het gebruik van pauzeersneltoets `P` (wordt opgevangen door `MainWindow`) kan het spel dan verdergezet worden.

3.2.20 Evolutie

Het eindresultaat ziet er een stuk beter uit dan de eerste versies. Dit is logisch, maar als je weet dat dit de eerste keer was dat we met QGraphicsView en de verwante klasses hebben gewerkt, is dat eens zo aannemelijk. In de eerste fase hebben we erop gefocust om vooral een duidelijke en goed werkende GUI te hebben (zie figuren 10 en 11 op pagina 24). Daarna, toen de HUD moest toegevoegd worden (figuur 12 op pagina 25), hebben we beroep gedaan op Kaj Heijmans om ons van een mooier kleurenschema te voorzien (figuur 13 op pagina 26). We zijn zeer tevreden over het eindresultaat, dat je kan zien op figuur 14 op pagina 27. Kaj heeft ook het icoontje ontworpen. Programmeurs hebben zelden of nooit goede design skills, daarom dat we ervoor gekozen hebben om daarvoor iemand anders in te schakelen. Ten slotte zijn wij het die het hebben geïmplementeerd, dus vonden we het niet erg om dit aan iemand anders toe te vertrouwen.

3.3 Algoritmes

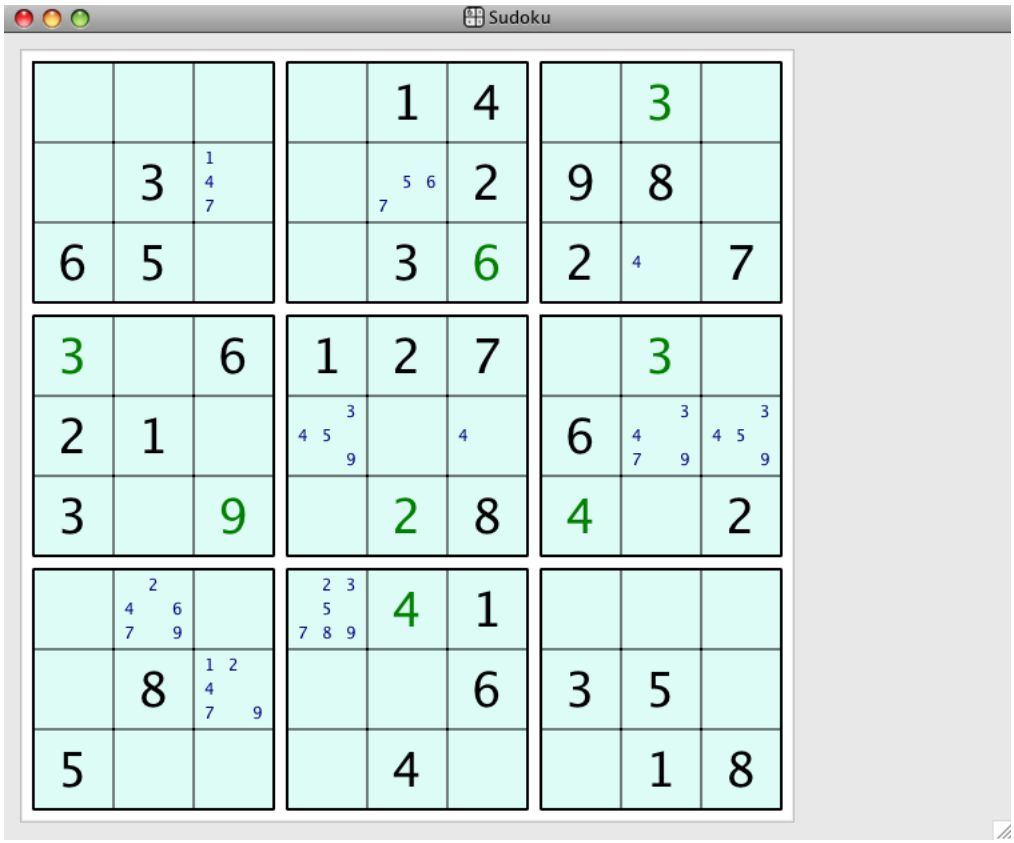
In onze applicatie hebben we ernaar gestreefd zo efficiënt mogelijke algoritmen te ontwikkelen, zonder dat deze efficiëntie ten koste moest gaan van de intuïtieve manier waarop ons generatiealgoritme werkt. We hebben er dan ook voor geopteerd om geen puzzels tijdens het spel te genereren, of tijdelijke puzzels bij te houden. Alles moest gegenereerd worden in een voor de gebruiker aanvaardbare tijd.

Er worden in onze applicatie verschillende algoritmen gebruikt, die zich voornamelijk bevinden in de Sudoku klasse. De drie belangrijkste algoritmen worden hieronder kort besproken.

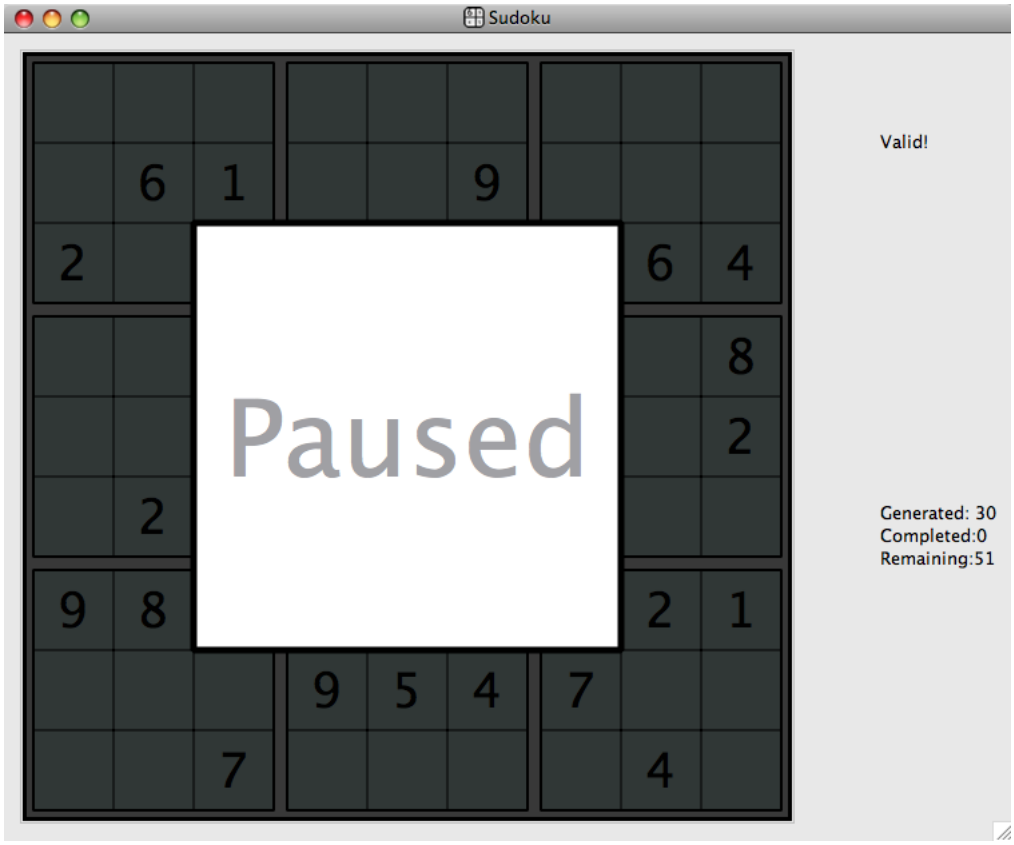
3.3.1 ScanSolve()

`ScanSolve()` is het algoritme dat de sudoku op een intuïtief aanvoelende manier zal proberen op te lossen. Dit is ons efficiëntste oplossingsalgoritme.

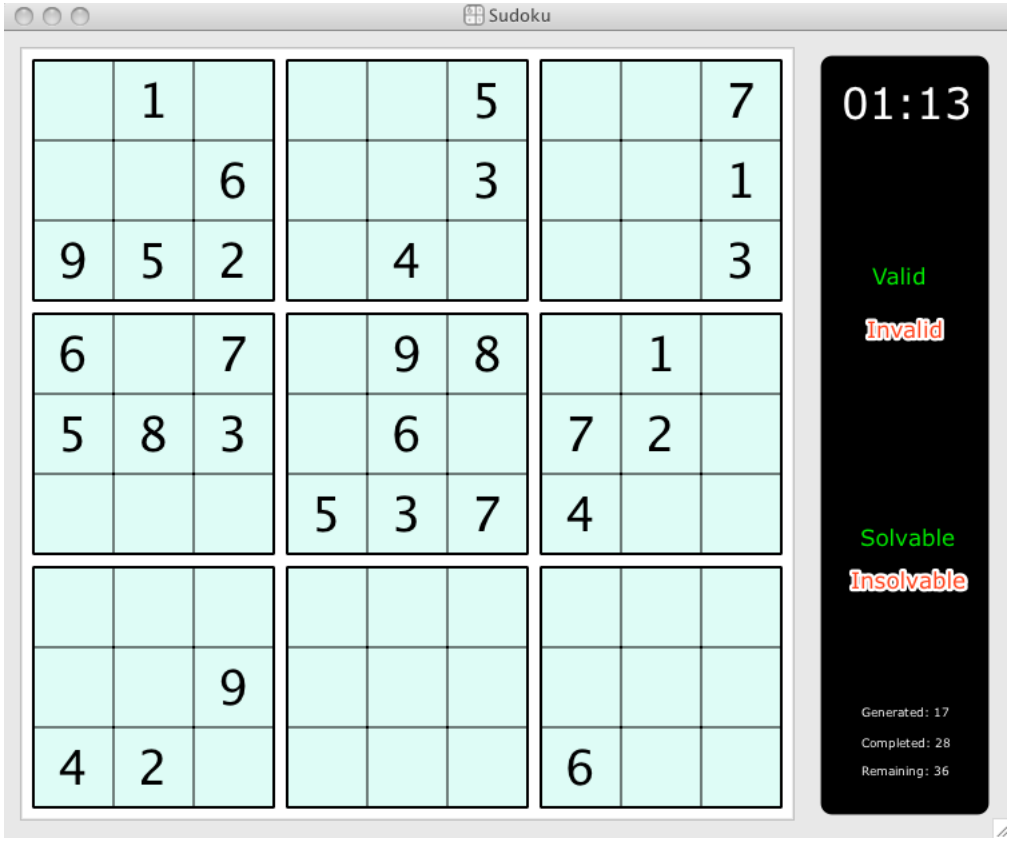
Het algoritme werkt door voor elke positie op het bord een array van booleans aan te maken waarin het bijhoudt welke zetten nog mogelijk zijn voor die positie. Dit gebeurt door voor elk van deze elementen alle nog mogelijke (niet uit het array verwijderde) zetten te proberen. De zetten die niet mogelijk zijn schrappen



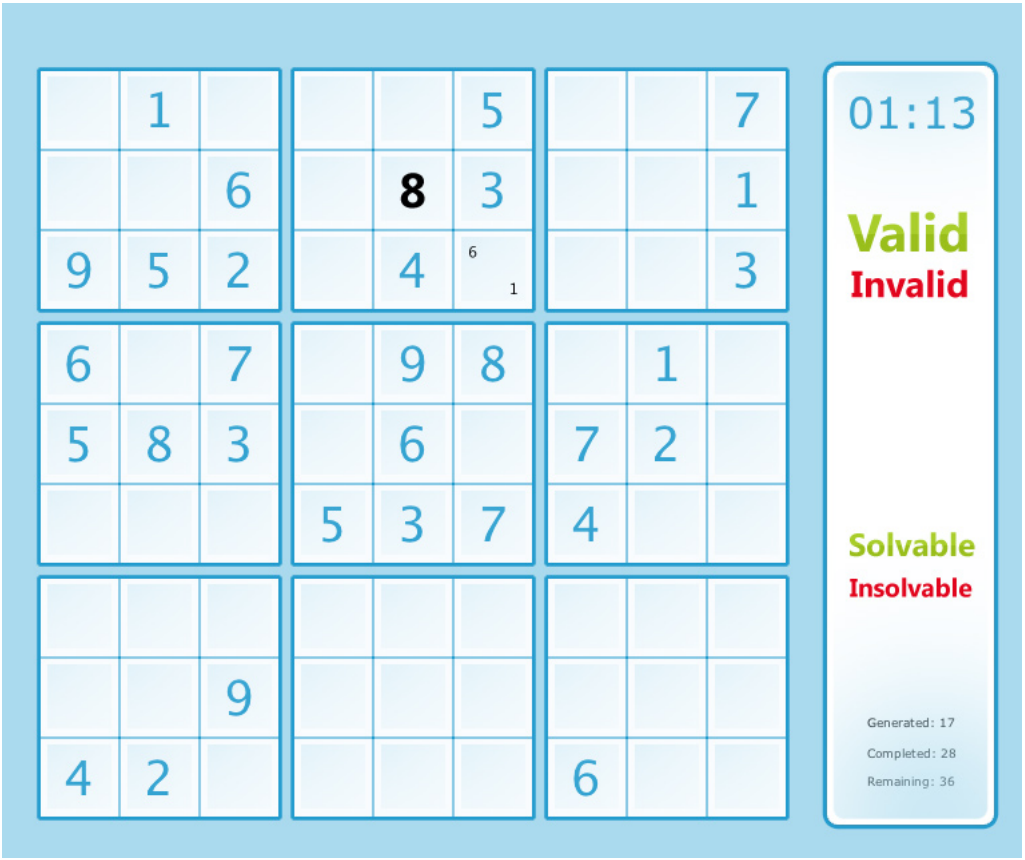
Figuur 10: Eerste versie van de Sudoku GUI - nog zonder HUD.



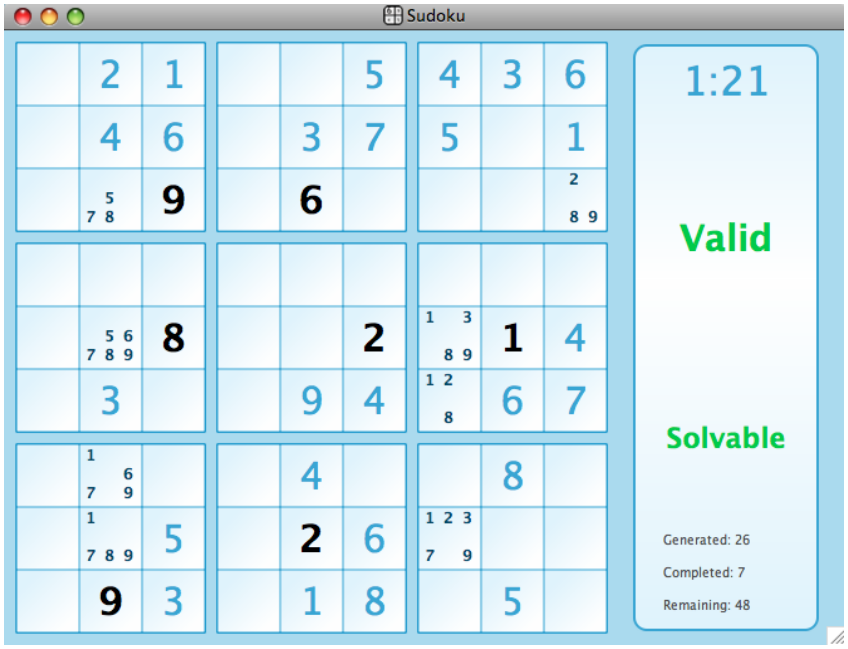
Figuur 11: Initiële versie van de HUD en het `PauseOverlay`.



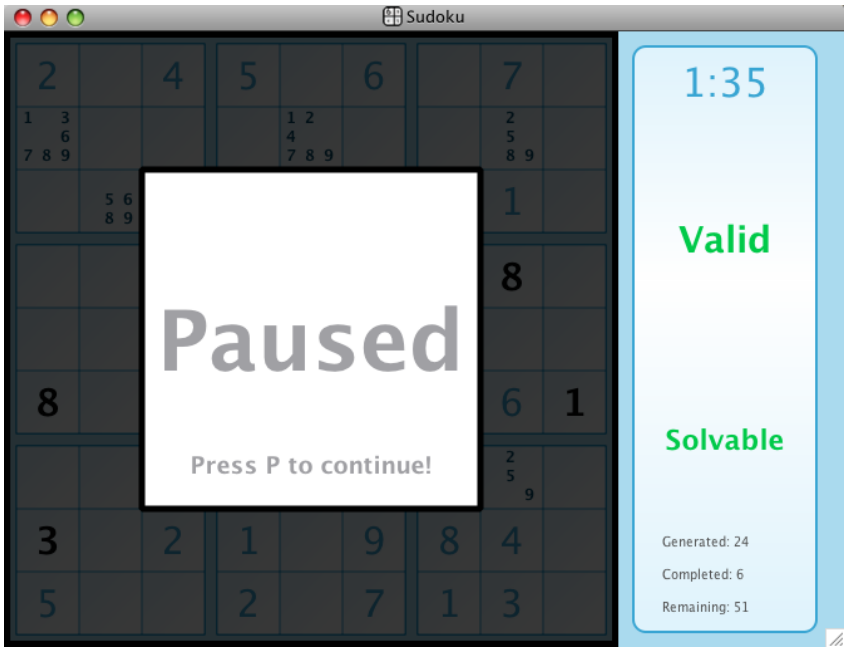
Figuur 12: Mock-up van de HUD.



Figuur 13: Mock-up van nieuw kleurenschema in overleg met Kaj Heijmans.



(a) Aan het spelen.



(b) Gepauzeerd.

Figuur 14: Geüpdatete GUI op basis van de mockup uit figuur 12 en het kleurenschema uit figuur 13.

Listing 3: ScanSolve()

```
while (!board->IsFull() && found) { // While possible moves
   found = false; // Keep if we have filled in an element
   for (int y = 0; y < 9; ++y)
      for (int x = 0; x < 9; ++x) {
         numPossible = 0; // We keep the number of possibilities
         for (int e = 1; e < 10; ++e) { // Looking for possible
                                        // solutions per box
            if (possibleSolutions[x][y][e]) {
               possibleSolutions[x][y][e] = board->IsValidMove(x, y, e);
                  if (possibleSolutions[x][y][e]) // If it is still valid
                     ++numPossible;
            }
         }
         if (numPossible == 1) {
            for (int e = 1; e < 10; ++e)
               if (possibleSolutions[x][y][e])
                  board->Set(x, y, e);
               found = true;
         }
      }
}
return board->IsFull();
```

we uit dit array. Wanneer een vakje nog maar één mogelijkheid heeft, wordt deze mogelijkheid permanent ingevuld, zodat `ScanSolve()` met dit verder opgeloste bord kan verderwerken.

Dit proces wordt herhaald zo lang er nog vakjes op het bord zijn waarvan mogelijkheden weggeschrapt kunnen worden. Als dit niet meer lukt, en het bord is vol, dan heeft `ScanSolve()` de puzzel opgelost. Is het bord niet vol, dan moeten we `BackTrackSolve()` gebruiken om de puzzel verder op te lossen.

Deze functie maakt ook gebruik van de method `IsValidMove()` uit de `Board` klasse. Die kijkt simpelweg in alle richtingen (alsook in het vierkant) of het getal al voorkomt.

Het belangrijkste stuk uit dit algoritme kan in Listing 3 teruggevonden worden.

3.3.2 BackTrackSolve()

Dit algoritme is het zwaardere oplosalgoritme in ons programma. Het zal dan ook enkel gebruikt worden wanneer de puzzel niet oplosbaar is door `ScanSolve()`. Ook zal voor elke uitvoering van `BackTrackSolve()` `ScanSolve()` sowieso aangeroepen worden, gezien `BackTrackSolve()` hier altijd voordeel uit haalt, en ook omdat we dan zeker weten dat `ScanSolve()` de puzzel niet kan oplossen. Dit voorbereindende werk gebeurt door de `SolveBoard()` functie.

Sudoku's die niet kunnen opgelost worden door `ScanSolve()` zullen echter niet gegenereerd worden door ons programma (omdat dit geen *echte* sudoku's zijn). Het `BackTrackSolve()` algoritme zal dan ook enkel gebruikt worden voor

Listing 4: BackTrackSolve()

```
/* Find the first element that is not filled in,
 * and set foundzero accordingly */

if (!solved && foundzero) { // Backtrack only if 0 found
    for (int e = 1; e < 10 && !solved; ++e)
        if (board->IsValidMove(x, y, e)) {
            board->Set(x, y, e);
            if (BackTrackSolve(board, x + 1, y))
                solved = true;
            else
                board->Remove(x, y);
        }
}

return solved;
```

het oplossen van borden die de gebruiker zelf heeft ingelezen (bijvoorbeeld via `csv`) en die niet oplosbaar zijn door `ScanSolve()`.

Het algoritme zelf is een eenvoudig backtracking algoritme met enkele sudoku-specifieke aanpassingen. Zo zal het een pointer naar een bord en een startpositie op het bord als parameters krijgen. Er wordt dan gezocht naar het eerstvolgende element dat niet ingevuld is, tenzij we een vol bord hebben (dit is dan namelijk geldig dankzij de vorige backtracking stappen, en dankzij de geldigheidscontrole in `SolveBoard()`). Hierna probeert het algoritme voor alle geldige zetten op deze plaats recursief verder op te lossen. Als dit lukt wordt de zet definitief ingevuld (we werken immers met een pointer naar het bord) en wordt er verder recursief gewerkt. Als we bij het recursief oplossen een vol bord krijgen wordt dit gemeld aan de oproepende functie, zodat we weten dat het bord opgelost is. Lukt het niet om een geldig vol bord te krijgen, dan verwijderen we het element van de huidig behandelde locatie, zodat de oproepende functie een ander getal kan proberen.

Het belangrijkste stuk uit dit algoritme is afgebeeld in Listing 4.

3.3.3 GenerateBoard()

Het `GenerateBoard()` algoritme is het meest uitgebreide algoritme van onze applicatie. De bedoeling van dit algoritme is in eerste instantie puzzels met een zo intuïtief mogelijk aanvoelende moeilijkheidsgraad te genereren, en deze puzzels toch zo 'willekeurig' mogelijk houden. We hebben hiervoor een aanpak gekozen die erg verschilt van de standaard methoden om sudoku's te genereren. In plaats van bijvoorbeeld het aantal elementen of het aantal iteraties de moeilijkheidsgraad te laten bepalen, hebben we er namelijk voor geopteerd om deze te laten afhangen van het aantal mogelijke manieren waarop het bord op te lossen valt. Dit wordt meteen duidelijker, wanneer we het algoritme bespreken.

Het algoritme zelf kan beter uitgelegd worden als we het onderverdelen in enkele stappen:

1. Eerst wordt er een volledig opgelost bord gegenereerd. Hiervoor zorgt de `GenerateRandomSolution()` functie. Deze zal 25 willekeurige getallen op willekeurige plaatsen van het bord zetten. We proberen dan het bord op te lossen. Als dit niet zou lukken, worden er gewoon andere willekeurige getallen geprobeerd. De reden dat dit er exact 25 zijn komt voort uit uitvoerig testen van verschillende waarden, waarbij er bij 25 geplaatste getallen gemiddeld het minste tijd werd besteed aan het genereren van een oplossing. Er zijn 2 redenen waarom we, wanneer het bord niet oplosbaar blijkt, telkens een nieuw bord genereren in plaats van gewoon incrementeel te werken met minder dan 25 getallen:

   (a) Het bord is nu meer 'random' dan we op een incrementele manier verwezenlijkt zouden krijgen met het gebruik van oplossingsalgoritmen.

   (b) Deze methode duurt effectief gemiddeld minder lang (bij het genereren van bijvoorbeeld 100 borden).

2. Hierna gaan we (ook weer op willekeurige plaatsen) elementen wegnemen uit het bord, waarbij we telkens kijken of het bord nog steeds oplosbaar blijft door het `ScanSolve()` algoritme. Elk van de verwijderde elementen zetten we tevens in een lijst (`undoList`), die we later nog nodig zullen hebben. Als er geen elementen meer verwijderd kunnen worden zonder dat het bord onoplosbaar wordt, gaan we naar de volgende stap.

3. Dit is de belangrijkste stap van ons algoritme. Hierin gaan we er namelijk voor zorgen dat er een duidelijk verschil is tussen de verschillende niveaus. Dit doen we door opnieuw een willekeurig element van het bord te verwijderen, en dit op een `redoStack` te zetten. We hebben dan een onoplosbaar bord (daarvoor zorgt de voorgaande stap), dat we terug oplosbaar gaan proberen te maken. Zo krijgen we meerdere *paden* volgens welke het bord opgelost kan worden.
   Het opnieuw oplosbaar maken van het bord gebeurt door één voor één elk element van de voorheen aangemaakte `undoList` terug te zetten tot er een element is waarbij ScanSolve() er weer in slaagt het bord op te lossen. Als dit niet lukt wordt dit element weer van het bord gehaald, en gaan we verder met het volgende element op de `undoList`. Als dit echter wel lukt, zal het element van de `undoList` gehaald worden en op de `redoStack` geplaatst worden, zodat we het na uitvoering van dit stuk van het algoritme terug op het bord kunnen plaatsen.
   Deze stap van het algoritme blijven we een aantal keren herhalen, waarbij het aantal bepaald wordt door het niveau.

4. De laatste stap bestaat er dan enkel nog in om te kijken of er genoeg elementen konden terug gezet worden op het bord (zodat het gewenste niveau bereikt is). Als dit zo is, moeten we de gewiste elementen nog terugplaatsen en is het algoritme afgelopen. Is dit niet zo, dan beginnen we helemaal opnieuw van het begin. Dit lijkt misschien geen goede

beslissing, maar uitproberen leert ons dat het beter is om het hele algoritme te herstarten, dan om verdere aanpassingen te maken aan het huidige bord (gezien dit voor lange recursieve stukken in het algoritme zou zorgen). Ons algoritme is overigens nog steeds erg efficiënt (gemiddeld rond de 25ms voor het genereren van een bord op een 1.7Ghz processor).

Tot slot volgt in listing 5 nog het belangrijkste stuk uit dit algoritme (het stuk waarin we elementen terug op het bord plaatsen om het niveau te bepalen). We excuseren ons voor de onoverzichtelijkheid. Het algoritme zelf kan natuurlijk altijd gevonden worden in de broncode, waar de formatting beter is.

3.4 Bestandsstructuren

3.4.1 CSV

We voorzien de mogelijkheid om het spelbord te exporteren (en in te lezen) in het `csv` formaat. Dit is een veelvoorkomend formaat, waarin gegevens van elkaar gescheiden worden door komma's of newlines. Bij ons is dit niet anders: een rij van het bord is een regel in het `csv` bestand, met de getallen voor deze rij gescheiden door komma's.

Dit formaat is vooral handig voor het importeren in andere toepassingen (bijvoorbeeld spreadsheet programma's), en om borden in te lezen die de gebruiker zelf heeft ingegeven. Zo kan de gebruiker bijvoorbeeld een bestaande sudoku puzzel uit een krant in zijn favoriete teksteditor ingeven, om hem vervolgens te laten oplossen door ons programma.

3.4.2 Serialized

De geserialiseerde import/export functie wordt gebruikt om volledige spellen op te slaan en in te lezen. Dit houdt in dat niet enkel het bord, maar ook gegevens zoals de speeltijd en de tijdelijke keuzes worden opgeslagen, samen met welke gegevens de gebruiker als zichtbaar heeft ingesteld in de HUD.

3.4.3 PNG

Er is ondersteuning om een spelbord te exporteren naar het PNG formaat. Dit kan handig zijn om een Sudoku te delen met iemand anders, of om later af te drukken. Enkel de Sudoku puzzel zelf, dus niet de HUD, wordt hierbij geëxporteerd. De achtergrond wordt transparant ingesteld. Om dit te vereenvoudigen – want hetzelfde moet gebeuren bij het afdrukken van een Sudoku puzzel – is er de `SudokuScene::renderBoard()` method. Deze method rendert het bord naar een `QPainter` object. Dit `QPainter` object werd daarvoor aangemaakt met als paint device een `QImage`. Zo wordt er dus letterlijk gerenderd naar een afbeelding.

Listing 5: GenerateBoard()

```
while (reinsert > 0 && stillFoundOne) {
  stillFoundOne = false;

  int yoffset = rand() % 9; // So we don't always start erasing the first few elements
  int xoffset = rand() % 9;

  for (int y = 0; y < 9 && reinsert > 0 && !undoList.empty(); ++y)
    for (int x = 0; x < 9 && reinsert > 0 && !undoList.empty(); ++x) {
      int realx = (x + xoffset) % 9; // The real x and y positions
      int realy = (y + yoffset) % 9;

      PositionElement pe(realx, realy, board->Get(realx, realy));
      redoStack.push(pe);
      board->Remove(realx, realy); // Remove the element from the board
                                   // and try to solve it again

      bool foundOne = false;
      list<PositionElement>::iterator it;
      for (it = undoList.begin(); it != undoList.end() && !foundOne;) {
        board->Set(it->GetX(), it->GetY(), it->GetE());
        if (BoardIsSolvable(*board, true)) { // Board is solvable by ScanSolve()
          PositionElement addableElement(it->GetX(), it->GetY(), it->GetE());
          redoStack.push(addableElement);

          it = undoList.erase(it); // Remove from the undoList and add to the redoStack
          --reinsert;
          foundOne = true;
          stillFoundOne = true;
        }
        else
          it++;

        board->Remove(realx, realy); // Removed in BOTH CASES!

        if (!foundOne && !redoStack.empty()) {
          board->Set(redoStack.top().GetX(), redoStack.top().GetY(), redoStack.top().GetE());
          redoStack.pop();
        }
      }
  }
}
```

Figuur 15: Vertaald Sudoku spel.

3.5 Overige

3.5.1 Vertaalbaarheid

Het hele programma is vertaalbaar. Qt biedt hier uitstekende ondersteuning voor [4]. We leveren alvast een Nederlandse vertaling mee. De juiste taal wordt automatisch gekozen aan de hand van de system locale.

3.5.2 Slechts één afbeelding en gradiënten

Het programma oogt mooi en aangenaam, maar toch zit er in het hele programma slechts één afbeelding: het programma icoon. Alle gradiënten die je ziet (in alle `SudokuElementen` en de `SudokuHUD`) worden dynamisch gegenereerd. Deze gradiënten worden bovendien slechts éénmaal gegenereerd.

3.5.3 Instellingen onthouden

Alle instellingen in het programma, dus de HUD-instellingen van het spel en de moeilijkheidsgraad in het dialoogvenster om een nieuw spel te starten, worden onthouden door middel van `QSettings`.

3.5.4 Cross-platform problemen en Qt bugs

Er zijn verscheidene problemen die optraden omdat we Sudoku cross-platform hebben gemaakt:

1. Normaal gezien zou renderen d.m.v. OpenGL probleemloos moeten werken, maar het werkt enkel betrouwbaar op Windows. Zie sectie 3.2.13 op pagina 16.

2. Het programma werkte probleemloos in Linux en Mac OS X, maar wanneer het in Windows uitgevoerd werd, crashte het steevast omdat er iets mis zou zijn met de iterators. Uiteindelijk bleek dat iterators op Windows lichtjes anders werken dan wij gewoon waren onder Linux en Mac OS X.

3. Ook threading werkte zonder problemen in Linux en Mac OS X, maar crashte in Windows, meerbepaald bij het beëindigen van een thread ging het mis. Uiteindelijk hebben we dit opgelost door gebruik te maken van het standaard `QObject` destructie model om threads succesvol te beëindigen op alle platformen.

4. X11 ondersteunt het `QEvent::ApplicationDeactivate` event niet, waardoor de applicatie niet automatisch gepauzeerd werd wanneer het de focus verliest. Zie sectie 3.2.8 op pagina 14 voor meer details.

5. Er waren problemen met het opvangen van keyboard events op Mac OS X: de combinatie `ALT`+6 wordt nóóit opgevangen, en de combinatie `ALT`+7 soms.

6. Het `QGraphicsItem::contextMenuEvent()` wordt helemaal niet aangeroepen op Mac OS X, maar wel op Linux.

7. Wanneer Sudoku werd afgesloten op Windows, bleef het proces nog actief: het programma werd niet correct afgesloten. Dit gebeurde zowel met de MingW[11] als met de Visual Studio 2005 compiler, telkens met versie 4.3.2 van Qt. Na een upgrade naar versie 4.4 van Qt was dit opgelost.

4 Taakverdeling en planning

4.1 Taakverdeling

4.1.1 Bram Bonné

Board	alles
Sudoku	alles
algemeen	vertalingen, op punt stellen van verslagen

[11] http://www.mingw.org/

Qt laden en opslaan van het spel

platformen Linux

4.1.2 Wim Leers

Qt alles behalve het hierboven genoemde

threading alles

FileIO alles (herbruikt uit vorig project: Reversi, voor het vak Object Georiënteerd Programmeren)

algemeen GUI gedeelte van het verslag

platformen Mac OS X en Windows (alsook Linux build script)

4.2 Logboek / Planning

Om dit project te realiseren hebben we verschillende stadia doorlopen. Deze beschrijven we hieronder.

Aan het begin van het project hebben we eerst uitvoerig geanalyseerd en besproken wat de klassenstructuur zou worden. We stelden toen reeds een voorlopig UML klassendiagram op, hetgeen weinig gewijzigd is doorheen het project. Ook maakten we toen afspraken over hoe de verschillende delen van het programma met elkaar zouden interageren. Tenslotte bepaalden we de structuur van de `svn` repository en plaatsten we hier de header files in die we samen gemaakt hadden.

Tijdens de paasvakantie is er nog bijkomend opzoekingswerk gebeurd met betrekking tot de algoritmen, en begon Bram met pseudocode op te stellen van de verschillende oplossings- en generatiealgoritmen, zodat het na de paasvakantie nog een kwestie van enkele uren was om de eerste versies van deze algoritmes werkende te hebben.

Ons eerste doel was zo snel mogelijk ervoor te zorgen dat er een spel gespeeld kon worden, zonder dat dit al soepel moest verlopen. Dit realiseerden we in korte tijd zodat het veel gemakkelijker werd om de algoritmen te testen en verder aan te passen.

De volgende stap was om het geheel wat gebruiksvriendelijker te maken, en al stilaan aan de extra's te beginnen. Het opslaan en inlezen was in een mum van tijd klaar (mede dankzij het hergebruiken van code uit ons vorige project) en ook hints werden geïmplementeerd.

Daarna is er vooral zeer veel tijd gestoken in de GUI, die nog sterk uitgebreid moest worden ten opzichte van de basis versie. Die basis versie was namelijk geschreven om makkelijker te kunnen testen. Toen de GUI op een punt kwam dat hij er als een echt bruikbare applicatie begon uit te zien, werd er begonnen met het cross-platform testen. Hierbij zijn verscheidene problemen aan het licht gekomen (zie ook sectie 3.5.4).

In de finale fase moesten we ons bijgevolg vooral bezig houden met finale aanpassingen: Wim met het mooier maken van de GUI, Bram met het sturen van onze applicatie naar testgebruikers, die feedback moesten verzorgen over de generatie van de puzzels. De feedback bleek hoofdzakelijk positief, dus er moesten slechts kleine aanpassingen gebeuren aan de algoritmen. Ook de vertalingen werden afgewerkt, en de laatste bugs werden opgelost.

Referenties

[1] http://doc.trolltech.com/4.3

[2] http://doc.trolltech.com/qq/qq18-macfeatures.html

[3] http://doc.trolltech.com/4.3/threads.html#threads-and-qobjects

[4] http://doc.trolltech.com/4.3/i18n.html

[5] http://websvn.kde.org/trunk/KDE/kdegames/kreversi

[6] http://doc.trolltech.com/4.3/qabstractscrollarea.html#setViewport

[7] http://thesmithfam.org/blog/2007/02/03/qt-improving-qgraphicsview-performance/

[8] http://en.wikipedia.org/wiki/Sudoku

[9] http://www.eddaardvark.co.uk/sudokusolver.html

[10] http://www.di-mgt.com.au/sudoku.html

[11] http://www.techfinesse.com/game/sudoku_solver.php#desc

[12] http://edwinchan.wordpress.com/2006/01/08/sudoku-solver-in-c-using-backtracking/

[13] http://code.google.com/p/reversi-school/

[14] http://trolltech.com/developer/task-tracker/index_html?id=156410&method=entry

[15] http://doc.trolltech.com/4.4/qpainter.html#drawRoundedRect