Project Sudoku: Analyseverslag

Wim Leers (0623800) en Bram Bonné (0623825)

27 februari 2008

1 Beschrijving van het onderwerp

1.1 Doel en interpretatie

Het doel is om een Sudoku-programma in Qt te ontwikkelen dat zowel gebruikt kan worden om het spel Sudoku te spelen (op gegenereerde spelborden) als om Sudoku's zelf op te lossen (deze kunnen dan door de gebruiker ingevoerd worden).

De spelregels die we zullen toepassen zijn deze die op Wikipedia[1] gevonden kunnen worden. Eventueel (als we tijd genoeg hebben) zouden we ook de spelvariant Hypersudoku[2] kunnen implementeren, gezien dan enkel het 'check'-algoritme aangepast zou moeten worden.

1.2 Extra's

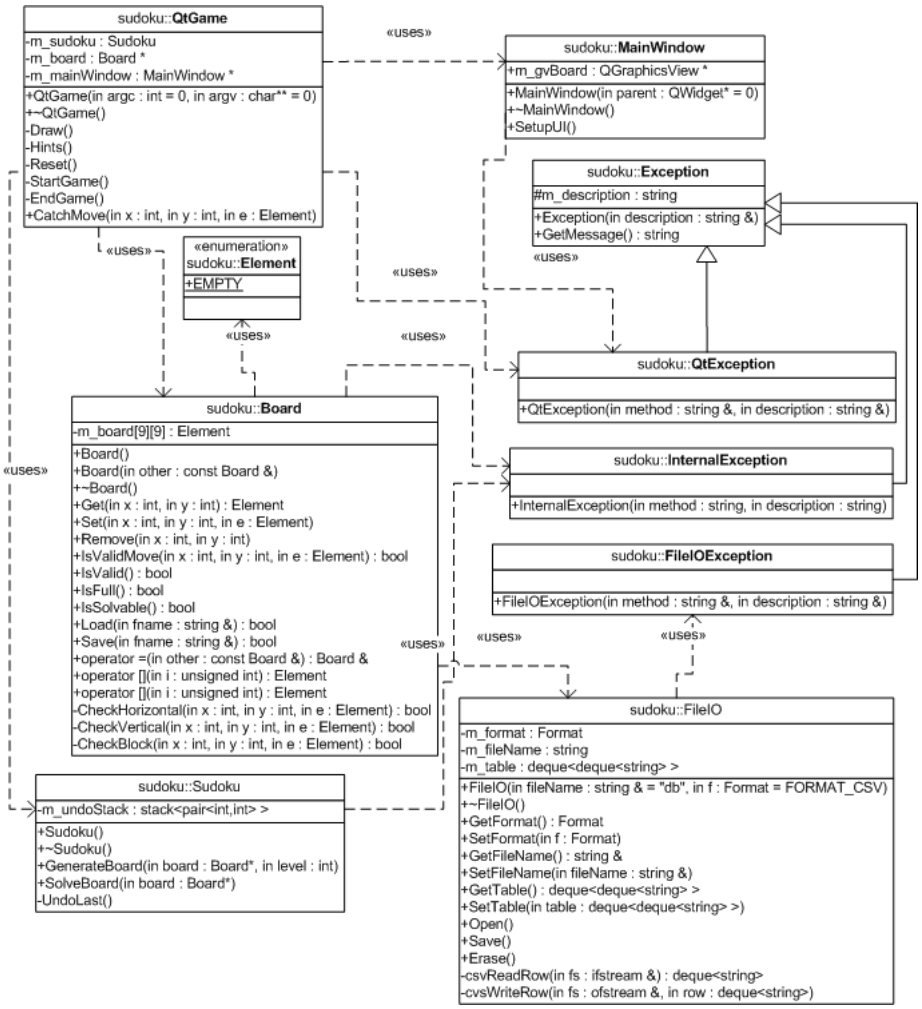
De extra's waar we momenteel aan denken zijn de volgende:

- Opslaan en inlezen van de huidige sudoku, zodat een spel later verdergezet kan worden.
- Verschillende moeilijkheidsgraden, gebaseerd op het aantal mogelijke manieren om de sudoku op te lossen.
- Hints (mogelijke zetten, zowel voor 1 geselecteerd vakje als voor het hele bord).
- Een volledig vertaalbaar programma (door middel van translation files).
- Doxygen documentatie (en dus ook Doxygen tags in de code).
- Platformonafhankelijkheid.
- Hypersudoku (als de tijd het toelaat).

[1] http://en.wikipedia.org/wiki/Sudoku
[2] http://en.wikipedia.org/wiki/Hypersudoku

2 Analyse

2.1 Klassediagram



2.2 ADT's

Board Dit is een klasse die een array van enums (met waarden `EMPTY` of 0-9) en de operaties hierop voorziet, alsook enkele kleine checkfuncties die betrekking hebben tot het bord zelf. Deze functies zijn `isValidMove()`, `isValid()` en `isSolvable()`. Ook kan hij een array inlezen uit, of wegschrijven naar een bestand (met behulp van de `FileIO` klasse).

Sudoku Deze klasse voorziet de algoritmes voor het spel. De oplosser voor het bord (`SolveBoard()`) zetten we hierin, alsook de `GenerateBoard()` functie. Beiden krijgen een pointer naar een `Board` element mee waarop ze moeten werken.

QtGame Dit is de klasse waarin alle interactie met de gebruiker (het Qt-gedeelte) alsook het spelverloop geregeld wordt. Deze klasse heeft dus

de nodige signals en slots om het spelverloop in goede banen te leiden. In de slots (maw: wanneer de gebruiker een bepaalde actie uitvoert) wordt natuurlijk gebruik gemaakt van de methods van de Sudoku klasse. Deze klasse bevat ook het spelbord.

MainWindow Deze klasse is – zoals de naam al aangeeft – de klasse voor het venster dat de andere Qt widgets bevat. Voor het belangrijkste element, het spelbord, gaan we hoogstwaarschijnlijk een `QGraphicsView` gebruiken. Op deze manier kunnen we op een zeer flexibele manier het spelbord vormgeven en kunnen we geavanceerde interacties voorzien. Deze bespreking is slechts een *inschatting* van hoe de klassestructuur van het Qt-gedeelte er zal gaan uitzien: het is onmogelijk om nu al te zeggen hoe het er precies zal gaan uitzien.

FileIO: De klasse die gebruikt wordt voor de FileIO zelf, dus niet voor het schrijven naar een bord. Gegevens die uit een bestand komen of in een bestand gaan worden via queues behandeld. We hergebruiken deze klasse uit ons vorig project (Reversi).s

Exception: De standaard exception klasse. Deze wordt nooit op zichzelf geïnstantieerd, maar kent de afgeleide klassen `QtException`, `InternalException` en `FileException`, die respectievelijk fouten bij de interface, het interne gedeelte en het lezen/schrijven naar bestanden opvangen.

2.3 Algoritmes

De moeilijke algoritmes bevinden zich voornamelijk in de `Sudoku` en de `Board` klassen. Deze bespreken we dan ook hier.

De belangrijkste van deze functies is de `isValidMove()` functie, gezien deze gaat kijken of het zetten van een element op een bepaalde plaats nog een geldig bord oplevert. Deze functie zal ervan uit gaan dat het bord dat wordt meegegeven reeds een geldig bord is, om op die manier zo efficiënt mogelijk de huidige zet te kunnen controleren. Hij gaat dan voor de meegegeven zet in de parameters kijken of het vakje al ingevuld is en daarna onderzoeken of hetzelfde getal al horizontaal, verticaal of in het 3x3 blokje voorkomt. De functie wordt gebruikt door zowat alle functies die iets te maken hebben met het controleren, het oplossen, of het genereren van het bord.

```
if (veld leeg)
    return (CheckHorizontal() && CheckVertical() && CheckBlock());
return false;
```

Waarbij we gebruik maken van het feit dat de compiler aan 'lazy evaluation' zal doen en als dusdanig alleen `checkVertical()` zal oproepen als `checkHorizontal()` `true` teruggaf. `checkHorizontal()` ziet er dan ongeveer zo uit:

```
for (i = 0; !foutGevonden && i < 9; i++)
    foutGevonden = (bord[rij][i] == toeTeVoegenElement);
return !foutGevonden;
```

De functie `isValid()` doet wat hij zegt. Hij krijgt gewoon een bord mee en gaat hierop kijken of er geen dubbele elementen voorkomen op plaatsen waar

het niet mag (door gebruik te maken van de `isValidMove()` functie op elke plaats van het bord).

`solveBoard()` zal een bord volledig invullen. Hij werkt door het gebruik van backtracking om zo de mogelijkheden af te gaan. Een stukje pseudocode:

```
if (!board.IsFull()) {
   for (alle mogelijke zetten) {
      if (board.IsValidMove( te proberen element )) {
         board.Set( dit element );
         if (SolveBoard())
            return true;
         else
            undoZet();
      }
   }
   if ( geen van de zetten gelukt )
      return false;
}
else
   return true;
```

De functie `generateBoard()` zal gebruik maken van `solveBoard()` om een bord te genereren. Eerst wordt (door gebruik te maken van willekeurige nummers en `solveBoard()`) een uitgespeeld spel gegenereerd. Hierna gaat `generateBoard()` elementen wegnemen tot op een moment dat er nog maar een aantal (dat afhangt van de moeilijkheidsgraad) mogelijke zetten zijn.

```
/* Elementen in random volgorde/met random waarde
 * proberen te plaatsen, zodat bord random genoeg is */
solveBoard(tempboard);
// triedElems[][] houdt bij welke elementen geprobeerd zijn
while (tempBoard.isValid() && !isFull(triedElems)) {
   do {
      randomX = rand() % 9 + 1;
      randomY = rand() % 9 + 1;
   } while (triedElems[randomX][randomY]);
   triedElems[randomX][randomY] = true;
   tempboard.RemoveElem(randomX, randomY);
   if (!tempboard.IsSolvable())
      undoLast(); // Werkt met stack
}
/* MAXNIVEAU is de hoogste moeilijkheidsgraad
 * We willen het aantal mogelijkheden van de gekozen
 * moeilijkheidsgraad hebben, undoLast() zet dan een element
 * terug, zodat het gemakkelijker wordt. */
for (int i = 0; i < MAXNIVEAU - moeilijkheidsgraad; i++)
   UndoLast();
```

2.4 Bestandsstructuren

Het enige dat moet opgeslagen/ingelezen kunnen worden is de huidige spelsituatie. Deze zullen we wegschrijven in het *csv* formaat (9 rijen, 9 kolommen), zodat het overal gemakkelijk geopend kan worden. Lege vakjes worden weergegeven door een spatie zodat het bestand er ook 'natuurlijk' uitziet in een tekst-editor (doordat een *csv* bestand gewoon een aantal rijen zijn met waarden die gescheiden worden door komma's).

3 Taakverdeling en Planning

Tot nu toe hebben we enkel de (voorlopige) klassedefinities (samen) gemaakt. Dit betekent dat de `Exception` klassen klaar zijn. Aangezien we de `FileIO` klasse uit ons vorige project zullen hergebruiken, is deze ook af. De taakverdeling voor de rest van het project ziet er als volgt uit, maar is onderhevig aan veranderingen:

Wim zal zowel het grafische gedeelte als het spelverloop voor zijn rekening nemen. Dit houdt in dat hij ervoor zal zorgen dat het spel van begin tot eind gespeeld kan worden.

Bram zorgt er dan voor dat het interne gedeelte van het spel werkt, dit wil zeggen: de volledige `Board` en `Sudoku` klassen. Hij houdt zich dus bezig met de algoritmen om een bord te genereren of op te lossen en met deze die kijken of het bord nog oplosbaar is op een bepaald moment in het spel. Ook zal hij zorgen voor de documentatie.

De planning ziet er (voorlopig) ongeveer zo uit:

einde paasvakantie Basisstructuur af, dat wil zeggen: zorgen dat de klassen er uitzien zoals ze zullen zijn aan het einde van het project.

week 2 trimester 3 Ervoor zorgen dat de algoritmes werken, zodat er een spel gespeeld kan worden (eventueel enkel via test-cases door de ontwikkelaars).

week 5 trimester 3 Spel moet speelbaar zijn door iedereen. Mogelijk is nog niet alle functionaliteit beschikbaar, maar de belangrijkste functies zouden af moeten zijn.

week 7 trimester 3 Volledige implementatie af, zodat we kunnen beginnen met het testen van speciale gevallen om de laatste bugs eruit te werken en zodat we aan het eindverslag en de presentatie kunnen beginnen.