Suziddle

Emil Unn Weihe
Frederik Fredslund Lassen

24. november 2015

Forord

Opgaven er udarbejdet af Emil Unn Weihe og Frederik Fredslund Lassen i november 2015. **Der gøres opmærksom på, at koden kun kan køres med F# 4.0 eller nyere.**

Introduktion

I denne rapport er der konstrueret en Sudoku i F, som det er beskrevet i opgaveformuleringen. Denne Sudoku er er af 9 rækker og 9 kolonner med 9 regioner. Det er desuden reglen i Sudoku-spillet at ingen tal må gå igen på samme række, kolonne og region.

Indhold

1 Problemformulering **2**

2 Problemanalyse og design **2**

3 Programbeskrivelse **3**

4 Afprøvning **3**

5 Diskussion **4**

6 Konklusion **4**

7 Brugervejledning **4**

8 Programtekst **4**

1 Problemformulering

Kravene jævnfør opgavebeskrivelsen:

1. Brugeren skal kunne indtaste filnavnet for (start-)tilstanden
2. Brugeren skal kunne indtaste triplen (r, s, v), og hvis feltet er tomt og indtastningen overholder spillets regler, skal matrixen opdateres, og ellers skal der udskrives en fejlmeddelelse på skærmen.
3. Programmet skal kunne skrive matricens tilstand pa skærmen (pa en overskuelig made).
4. Programmet skal kunne foreslå lovlige tripler (r, s, v).
5. Programmet skal kunne afgøre, om spillet er slut.
6. Brugeren skal have mulighed for at afslutte spillet og gemme tilstanden i en fil.
7. Programmet skal kommenteres ved brug af fsharp kommentarstandarden
8. Programmet skal struktureres ved brug af et eller flere moduler, som I selv har skrevet
9. Programmet skal unit-testes

2 Problemanalyse og design

Brugerfladen til programmet skal bygges på en måde så brugeren, i starten af programmet, kan vælge to veje at gå ned af. Den første vej består af at kunne starte et helt nyt spil. Dette skal dog gøres fra en kendt løselig sudoku. Dette vil blive løst med prædefinerede, hardcodede template(s) som ligger sammen med programmet. Den anden vej bestårt af at kunne indlæse en tidligere spil-tilstand. Dette gøres ved samme funktionalitet som første vej, bortset fra at brugeren har mulighed for at specificere hvilken fil der skal indlæses fra. Disse to ting udgør punkt 1 i opgavebeskrivelsen.

I modulet bliver der lagt stor vægt på kodegenbrug, effektivitet og kort, forståelig kode. Dette giver sig for eksempel til kende ved at det er muligt at bruge *hint* funktionen, som skal bruges til at foreslå lovlige tripler (jævnfør, og derfor udgørende, punkt 4 i opgavebeskriven) til at indentificere om en given indtastning er lovlig, og derfor kan krav 2 løses let og elegant. Dette implementeres desuden ved at kostruere en eksisterende matrice ud fra den eksisterende matrice og returnere den, således at den kan blive brugt atter igen.

Matricens tilstand skal skrives til skærmen på en simpel, let og overskuelig måde med mindst mulig kode, selvom den selvfølgelig stadig møder krav 3.

Ved hurtigt at tjekke om matricen indeholder tomme felter, kan det afgøres om spillet er slut, jævnfør krav 5.

Spillet skal gemmes til en fil, som også skal kunne læses af programmet igen. Dette skal gøres ved at implementere standarden som specificeret i opgavebeskrivelsen: denne beskriver, at rækker skal opbevares på linjer for sig, hvor hele rækken står tegn for tegn, således at rækken er 9 tegn lang og der er 9 linjer med rækker. Programmet er derfor kompatibelt med filen fra opgavebeskrivelsen, og som en bonus er programmet kompatibelt med de andre studerendes implementationer af sudoku'en. Krav nummer 6 mødes derfor her. Programmet kan desuden afsluttes på almindelig vis ved at bruge $ctrl + c$.

I $interface$-filen skal der skrives kommentarer jævnfør F-kommentarstandarden med *summary* som kort beskriver funktionen, *param*'s som beskriver de enkelte parametre funktionen tager samt *returns* som forklarer hvad funktionen returnerer. Dette udgør krav 7, og der bliver desuden genereret en samlet XML-fil ud fra al' dokumentationen i $interface$-filen.

Til denne $interface$-fil følger også et modul, som har alle de underliggende funktioner som har med sudoku'en at gøre. Brugergrænseflade-delen og sudokuen er derfor abstraheret væk fra hinanden således, at det eneste brugergrænsefladen skal tage sig af er at oversætte mellem brugeren og det underliggende modul. Dette møder krav 8.

Der skal i programmet desuden overvejes unit-tests, og disse skal implementeres så de dækker modulet, således at sudoku-spillet er dækket for eventuelle fejl der måtte opstå under uheldige modifikationer, og således, at en given kode-ændring i modulet kan testes således at det ikke ødelægger kerne-funktionaliteten i modulet. Dette udgør krav 9.

Programmet skal desuden komme med en eksekverbar fil eller et script, der kan bruges til at køre tests, generere dokumentation, bygge eksekverbare filer samt køre spillet nemt og enkelt. Det traditionelle $GNUMakefile$ format er valgt til dette, fordi det tillader netop dette, og er standarden til denne slags opgaver.

Som det kan ses, er spillet derfor designet til at møde en analyse af kravene i problemformuleringen, og opfylder derfor kravene i samme.

3 Programbeskrivelse

I vores *module* Sudoku findes funktioner:

hints - en funktion som ud fra et koordinat på brættet, finder de værdier som kan indsættes i feltet. Først findes lister af værdierne for hhv. rækken, søjlen og regionen. Disse omdannes til *set*, hvorefter værdier fra 1 - 9 som ikke findes i disse *set*, returneres via hjælpefunktionen $missin'$. De tre lister smides ind i en *sequence*, og sendes videre til $Set.intersectMany$, som finder de fælles værdier for listerne.

isFinished - atter en funktion som anvender $missin'$, til at tjekke at alle rækkerne indeholder tal fra 1 - 9, og at brættet derfor er udfyldt / spillet er ovre.

insert - en funktion som ta'r et koordinat på brættet, og indsætter en værdi i feltet (altså, i tilfælde af at det er en korrekt værdi). Først anvendes *hints* til at finde de værdier som kan indsættes i feltet, hvorefter der tjekkes om den givne værdi er en af dem. Hvis dette ikke er tilfældet returneres $None$. Derimod, hvis det er tilfældet, *mappes* bræt-listen og værdien indsættes.

print - en funktion som ta'r brættet og printer det på en simpel vis. Der køres over alle felter i bræt-listen, hvorved deres værdier printes.

load - en funktion som anvender $System.IO.File.ReadAllLines$ til at *load*'e linjerne i en fil og konstuere en *board*-type ud fra samme. Funktionen *parse* er hjælpefunktion til denne, og *parse* tager linjerne og *map*'er dem ud over en funktion der kører $Seq.foldBack$ på hver linje og derved kører hver linje igennem *char* efter *char*. Der afgøres her hvilken *int* værdi *char*'en har, og det hele akkumuleres op i en liste af *int*'s og *mappes* tilbage til en liste af lister med heltal, som er *board*-typen.

save - en funktion som gemmer en *board*-type ned i en fil ved hjælp af $System.IO.File.WriteAllLines$. Denne funktion gør essentielt set det modsatte af *load*-funktionen, hvor de har hjælpefunktionen $stringify$ som *mapper* hver row til en linje i tekstdokumentet og på hver af disse linjer bliver en række fra sudokuen konverteres fra *int* til *char* med en $List.fold$ hvor en akkumulator af typen streng bliver kørt med rundt og der bliver sammesat strenge, således at den ligner den repræsentation som der er beskrevet i opgavebeskrivelsen.

4 Afprøvning

Der er konstrueret unit-tests dækkende:

- At gemme et sudoku spil.
- At indlæse et gemt sudoku spil.
- At bedømme om et spil er færdigt.
- At give hints til et felt.
- At indsætte et tal i et felt.

5 Diskussion

Skulle man forbedre denne sudoku er det oplagt at gøre følgende:

- Sudokuen kunne printes endnu mere overskueligt end den på nuværende tidspunk bliver. Dette er dog på bekostning af kodelinjer og kodeoverskuelighed, men dette vil gøre sudokuen mere præsentabel og nemmere at finde rundt i visuelt.
- Fejl-håndteringen kunne være mere ekstensiv - det er muligt at ramme *exceptions* ved et uheld, og dette må i sagens natur ikke kunne ske - i hvert fald kun i et lille omfang.
- Et udvalg af start-tilstande kunne tilbydes fra start og blive tilfældigt valgt - endda auto-genereret. Sidstnævnte er dog en lidt mere kompleks handling. Som en sidenote til første, kunne templates være direkte kompileret ind i exe-filen med kompilerns resource-flag.

6 Konklusion

Det kan ses i problemanalysen at alle krav fra problemformuleringen er mødt i designet, og spillet er programmelt konstrueret udfra samme design - derfor kan det konkluderes - desuden ved hjælp af unit-test'ene - at programmet opfylder de stillede krav.

7 Brugervejledning

Programmet kan bygges ved hjælp af *make build*, køres ved hjælp af *make run* samt køres og bygges sekventielt ved hjælp af *make dev*. Tests køres desuden med *make test*.

Når programmet starter er der muligt at vælge enten at starte et spil og indlæse et eksisterende spil. Vælges den første mulighed vil man blive kastet lige ud i et spil, mens der i indlæsnings-tilfældet vil komme en prompt hvor man vil blive bedt om at skrive navnet på filen der skal indlæses - herefter vil spillet blive indlæst, og man vil nu være i samme situation som når man startede et nyt spil.

Inde i selv spillet kan man køre tre forskeklige kommandoer: *help*, *save* og *hint*. *Help* printer samme hjælpemeddelelse som brugeren får i starten af spillet, som forklarer hvordan spillet spilles, og essentielt hvilke kommandoer der findes. *Save* gemmer spillet - der promptes om et filnavn og dettes huskes, således at der kun skal trykkes enter alle efterfølgende gange der skal gemmes, i tilfælde af at filnavnet ønskes at være det samme. Til sidst er der *hint* som åbner en promt hvor der skal specificeres hvilet felt (række, søjle) der ønskes hjælp til.

Til sidst er der den vigtigste funktion: indsætning af værdier i sudokuen. Dette gøres ved at bare at skrive tal i rækkefølgen: række nummer, søjle nummer og værdien der ønskes indsat. For eksempel 247 for række nummer 2, søjle nummer 4 og værdien der skal indsættes 7. Bemærk at der her bliver fjerne alle ikke-tal, og derfor er det også muligt at skrive 2,4=7 eller (2, 4)=7 eller 2 4 7.

8 Programtekst

Listing 1: sudoku-interfacet, sudoku.fsi

```
1  module Sudoku
2
3  type row = list<int>
4  type board = list<row>
5
6  /// <summary><c>hints</c>-funktionen finder de  v rdier  som kan  i n d s t t e s  i
       et  g i v e n t  f e l t .</summary>
7  /// <param name = "f e l t"></param>
8  /// <param name = "b r t"></param>
9  /// <returns><c>set</c> af  v rdier .</returns>
10 val hints : (int * int) -> board -> Set<int>
```

```
11
12 /// <summary><c>isFinished</c> tjekker om alle  r kker  indeholder tal fra 1
       - 9.</summary>
13 /// <param name = "b r t"></param>
14 /// <returns>Hvorvidt om spillet  er  ovre.</returns>
15 val isFinished : board -> bool
16
17 /// <summary> I n d s tter  en  v rdi .</summary>
18 /// <param name = "f e l t"></param>
19 /// <param name = " v rdi "></param>
20 /// <param name = "b r t"></param>
21 /// <returns>Shizz'</returns>
22 val insert : (int * int) -> int -> board -> board option
23
24 /// <summary>Printer  Soduko  plade.</summary>
25 /// <param name = "b r t"></param>
26 val print : board -> unit
27
28 /// <summary>Loader  et  b r t  fra  f i l .</summmary>
29 /// <param name = "filnavn"></param>
30 /// <returns>Et  b r t</returns>
31 val load : string -> board
32
33 /// <summary>Gemmer  et  b r t  til  f i l .</summmary>
34 /// <param name = "filnavn"></param>
35 /// <param name = "b r t"></param>
36 val save : string -> board -> unit
```

Listing 2: sudoku-modulet, sudoku.fs

```
1  module Sudoku
2
3  type row = list<int>
4  type board = list<row>
5
6  let span = [1 .. 9]
7  let missin' = Set.difference (set span)
8
9  let rec transpose = function
10     | (_ :: _) :: _ as list -> List.map List.head list :: transpose (List.
           map List.tail list)
11     | _ -> []
12
13 let region (r, s) (list : board) =
14     let r, s = (r / 3 * 3, s / 3 * 3)
15     List.collect (fun (x : row) -> x.[s .. s + 2]) list.[r .. r + 2]
16
17 let hints (r, s) (list : board) =
18     seq [set list.[r] |> missin'
19          set (transpose list).[s] |> missin'
20          set (region (r, s) list) |> missin'] |> Set.intersectMany
21
22 let isFinished (list : board) =
23     List.forall (fun x -> (missin' (set x)).Count = 0) list
24
25 let insert (r, s) v list =
26     if not ((hints (r, s) list).Contains v) then None else
27     List.mapi (fun i x -> if i <> r then x else
```

```
28                              List.mapi (fun i x -> if i <> s then x else v) x)
                                    list |> Some
29
30  let print list =
31      let case0 x value = if x = 0 then " " else value
32      List.iteri (fun i x -> List.iter (fun x -> printf " %s %s" (case0 x (x.
            ToString ())) (case0 i "|")) (i :: x)
33                              printfn "\n
                                    +--+--+--+--+--+--+--+--+--+") (span
                                     :: list)
34
35  let parse input =
36      List.map (fun x -> Seq.foldBack (fun y acc -> (if y = '*' then 0 else (
            System.Convert.ToInt32 y) - 48) :: acc) x []) input
37
38  let stringify list =
39      List.map (fun x -> List.fold (fun acc y -> acc + if y = 0 then "*" else
             y.ToString ()) "" x) list
40
41  let load file =
42      System.IO.File.ReadAllLines file |> List.ofArray |> parse
43
44  let save file list =
45      System.IO.File.WriteAllLines (file, stringify list)
```

Listing 3: tests, test.fsx

```
1   let test (id, b) = printfn "%s: %b" id b
2
3   let izzle =
4       [[5; 3; 0; 0; 7; 0; 0; 0; 0];
5        [6; 0; 0; 1; 9; 5; 0; 0; 0];
6        [0; 9; 8; 0; 0; 0; 0; 6; 0];
7        [8; 0; 0; 0; 6; 0; 0; 0; 3];
8        [4; 0; 0; 8; 0; 3; 0; 0; 1];
9        [7; 0; 0; 0; 2; 0; 0; 0; 6];
10       [0; 6; 0; 0; 0; 0; 2; 8; 0];
11       [0; 0; 0; 4; 1; 9; 0; 0; 5];
12       [0; 0; 0; 0; 8; 0; 0; 7; 9]];
13
14  let finished =
15      [[1; 2; 3; 4; 5; 6; 7; 8; 9];
16       [1; 2; 3; 4; 5; 6; 7; 8; 9];
17       [1; 2; 3; 4; 5; 6; 7; 8; 9];
18       [1; 2; 3; 4; 5; 6; 7; 8; 9];
19       [1; 2; 3; 4; 5; 6; 7; 8; 9];
20       [1; 2; 3; 4; 5; 6; 7; 8; 9];
21       [1; 2; 3; 4; 5; 6; 7; 8; 9];
22       [1; 2; 3; 4; 5; 6; 7; 8; 9];
23       [1; 2; 3; 4; 5; 6; 7; 8; 9]];
24
25  test ("[save] t01", Sudoku.save "test.txt" izzle = ())
26  test ("[save] t02", Sudoku.save "test2.txt" finished = ())
27
28  test ("[load] t01", Sudoku.load "test.txt" = izzle)
29  test ("[load] t02", Sudoku.load "test2.txt" = finished)
30
31  test ("[isFinished] t01", Sudoku.isFinished izzle = false)
32  test ("[isFinished] t02", Sudoku.isFinished finished = true)
```

```
33
34 test ("[hints] t01", Sudoku.hints (0, 2) izzle = set [1; 2; 4])
35 test ("[hints] t02", Sudoku.hints (0, 3) izzle = set [2; 6])
36 test ("[hints] t03", Sudoku.hints (0, 0) finished = set [])
37
38 test ("[insert] t01", Sudoku.insert (0, 2) 3 izzle = None)
39 test ("[insert] t02", Sudoku.insert (0, 2) 2 izzle <> Some izzle)
40 test ("[insert] t03", Sudoku.insert (0, 0) 1 finished = None)
```

Listing 4: brugerfladen, sudizzle.fsx

```
1  let help() =
2    printfn "%s" """
3    Spillet underst tter f lgende kommandoer:
4
5    help  - bringer denne hj lp frem
6    save  - gemmer spillet
7    hint  - giver et hint til et felt
8
9    Desuden spilles der ved fx. at skrive:
10
11   (3,2)=2
12   322
13   3,2,2
14
15   Hvor de tre tal repr senterer, i denne r kkef lge:
16
17   - r kke
18   - s jle
19   - v rdi
20
21   God spillelyst!
22   """
23
24 let stripNonNumbers input =
25   String.map (fun c -> if System.Char.IsNumber(c) then c else char(0))
       input
26
27
28 let mutable savedAs = ""
29
30 let saveGame sudoku =
31   printfn "%s" """
32   Indtast det filnavn du vil gemme som.
33   """
34
35   printf "[%s]> " savedAs
36
37   let input = System.Console.ReadLine()
38
39   if savedAs <> "" && input = "" then
40     Sudoku.save savedAs sudoku
41   else
42     Sudoku.save input sudoku
43     savedAs <- input
44
45   printfn "Spillet blev gemt!"
46
47 let displayHint sudoku =
```

```
48    printfn "%s" """
49    Indtast feltet du vil have et hint for: r,s
50    """
51
52    printf "hint > "
53
54    let input = System.Console.ReadLine()
55
56    let numbers = stripNonNumbers input
57    if String.length numbers = 2 then
58      let r = int(numbers.[0]) - 49
59      let s = int(numbers.[1]) - 49
60
61      printfn "%A" (Sudoku.hints (r, s) sudoku)
62    else
63      printfn "Forkert indtasting af hint!"
64
65  let rec loop sudoku =
66    // Print the sudoku to screen
67    Sudoku.print sudoku
68
69    // Keep asking the user for input with recursion
70    let rec interrogate() =
71      printf "> "
72      let input = System.Console.ReadLine()
73      match input with
74      | "save" -> saveGame sudoku; interrogate ();
75      | "help" -> help(); interrogate ();
76      | "hint" -> displayHint sudoku; interrogate ();
77      | _ ->
78        let numbers = stripNonNumbers input
79        if String.length numbers = 3 then
80          let r = int(numbers.[0]) - 49
81          let s = int(numbers.[1]) - 49
82          let v = int(numbers.[2]) - 48
83
84          printfn "Indtasting r:%d s:%d v:%d" r s v
85
86          match Sudoku.insert (r,s) v sudoku with
87          | Some x -> loop (x)
88          | None ->
89            printfn "Forkert indtastning"
90            interrogate()
91          ()
92        else
93          printfn "Forkert indtastning"
94          interrogate ()
95    interrogate()
96
97  let newGame() =
98    loop(Sudoku.load "./templates/01.txt")
99
100
101 let loadGame() =
102   printfn "%s" """
103   Indtast venligst navnet p  det gemte spil
104   """
105
```

```
106     printf "> "
107     let input = System.Console.ReadLine()
108
109     help()
110
111     input |> Sudoku.load |> loop
112     ()
113
114
115 (* Main entry point for application *)
116 [<EntryPoint>]
117 let main args =
118     printfn "%s" """
119     Velkommen til SuDizzle, du har nu f lgende valgmuligheder:
120
121     - 'start'   for at starte et nyt spil
122     - 'indl s'  for at indl se et tidligere spil
123     """
124
125     // Keeps asking the user for input with recursion
126     let rec interrogate() =
127         printf "> "
128         let input = System.Console.ReadLine()
129         match input with
130             | "start"   -> help(); newGame()
131             | "indl s"  -> loadGame()
132             | _ ->
133                 printf "%s" "Forkert input, pr v igen!\n";
134                 interrogate()
135
136     interrogate()
137
138     // Return 0 to indicate success
139     0
```