Aflevering 3i

Peter Asp Hansen (glt832)

23. februar 2023

I denne afleveringsopgave benytter vi Canvas og Turtle til at lave simple grafiker vha. den viden vi har om lister.

3i0

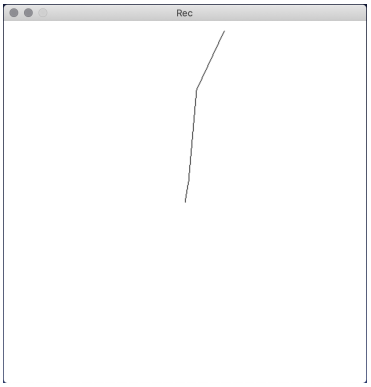
I delopgave 3i0 har skal vi konvertere listen lst til en turtleCmd list som vi kan tegne i Canvas vha. kommandoen turtleDraw. Dette gøres på fire metoder. Den første metode er vha. en rekursiv funktion kaldet fromMoveRec.

```
#r "nuget:diku.canvas, 1.0.1"
open Canvas
type move = int*int // a pair of turn and move
let lst = [(10, 30); (-5, 127); (20, 90)]

///<summary> fromMoveRec er en rekursiv funktion som laver en turtleCmd list </summary>
///<param name="lst"> En liste af tupler som er et par af turn og move </param>
///<returns> En turtleCmd liste med turtle kommandoerne Turn og Move </returns>
let rec fromMoveRec (lst:move list): Canvas.turtleCmd list=
    match lst with
    [] -> []
    |(dir,dist)::t -> Turn dir::Move dist::(fromMoveRec t)

do turtleDraw (500,500) "Rec" (fromMoveRec lst)
```

Denne funktion tager lst som input. Hvis lst er tom returneres en tom liste []. Hvis listen ikke er tom tages det første element i listen (dir,dist) og laves om til en turtleCmd list [Turn dir;Move dist]. Dernæst kaldes funktionen igen men denne gang på halen af listen. Dermed betragter vi nu det næste element i lst som (dir,dist). Dette element bliver konverteret til en turtleCmd list som vi kan concatenate med den forrige turtleCmd liste vha kommandoen ::. Dette forsætter indtil halen er tom. Med input lst returnere funktionen dermed listen [Turn 10; Move 30; Turn -5; Move 127; Turn 20; Move 90]. Vi kan tegne denne turtleCmd liste vha. kommandoen TurtleDraw.



Figur 1: rec

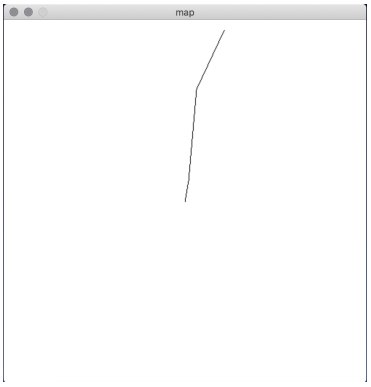
Vi gentager successen og prøver at komme frem til det samme resultat. Denne gang med list modulet List.Map. Vi definerer her en funktion kaldet fromMoveMap.

```
///<summary> fromMoveMap laver en turtleCmd list vha. List.map modulet </summary>
///<param name="lst"> En liste af tupler hvor tuplerne er par af turn og move </param>
///<returns> En turtleCmd liste med turtle kommandoerne Turn og Move </returns>

let fromMoveMap (lst:move list): Canvas.turtleCmd list =
    List.map(fun ((dir,dist):move) -> [Turn dir;Move dist]) lst |> List.concat

do turtleDraw (500,500) "map" (fromMoveMap lst)
```

Denne funktion tager også lst som input og returnere en turtleCmd list. I modsætning til funktionen fromMoveRec så er denne funktion ikke rekursiv. I stedet bruger vi List.map som benytter en anonym funktion fun som tager hvert element (dir,dist) fra lst og returnere [Turn dir; Move dist] listen. Vi har nu et antal af seperate turtle.Cmd lister som vi gerne vil concatenate. Dette gør vi ved først at bruge pipe-forward operatoren og dernæst benytte list modulet List.concat. Dette giver os igen den fulde turtleCmd liste som vi tegner vha. turtleDraw.



Figur 2: map

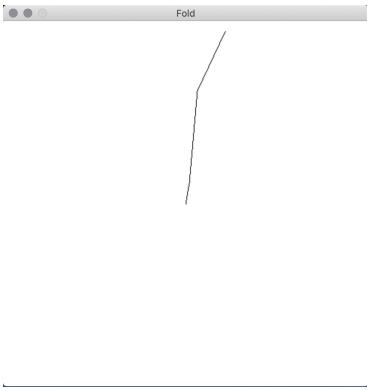
Den næste funktion kalder vi fromMoveFold. Denne funktion benytter List.fold modulet.

```
///<summary> top1 er en funktion som benyttes i List.fold modulet </summary>
///<param name="acc"> En accumulator som defineres i List.fold modulet </param>
///<param name="(dir,dist)"> En tupel bestaaende af vores par af turn og move </param>
///<returns> vores accumulator concatenated med en liste af Turn og Move
<returns>
let top1 acc (dir,dist) = acc@[Turn dir;Move dist]

///<summary> fromMoveFold laver en turtleCmd list vha. List.fold modulet </summary>
///<param name="lst"> En liste af tupler som er par af turn og move </param>
///<returns> En turtleCmd liste med turtle kommandoerne Turn og Move </returns>
let fromMoveFold (lst:move list): Canvas.turtleCmd list =
    List.fold top1 [] lst

do turtleDraw (600,600) "Fold" (fromMoveFold lst)
```

List.fold modulet tager en anden funktion, som vi kalder top1, en accumulator som i vores tilfælde er den tomme liste [] og til sidst tager den lst. Vores funktion top1 tager input acc, for accumulator, og en tupel, (dir,dist). Den returnerer acc@[Turn dir;Move dist], som er den tomme liste concatenated med vores turtleCmd liste. Vi tegner denne liste med turtleDraw.



Figur 3: fold

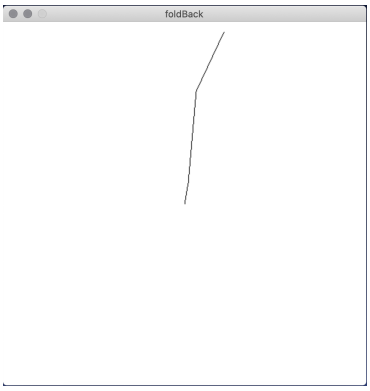
Den sidste funktion kalder vi fromMoveFoldBack. Denne funktion benytter List.foldBack modulet til at lave en turtleCmd liste.

```
///<summary> top2 er en funktion som benyttes i List.foldBack modulet </summary>
///<param name="(dir,dist)"> En tupel bestaaende af vores par af turn og move </param>
///<param name="acc"> En accumulator som defineres i List.foldBack modulet </param>
///<returns> en liste af Turn og Move concatenated med vores accumulator <returns>
let top2 (dir,dist) acc  = [Turn dir;Move dist]@acc

///<summary> fromMoveFoldBack laver en turtleCmd list vha. List.foldBack modulet </summa
///<param name="lst"> En liste af tupler som er et par af turn og move </param>
///<returns> En turtleCmd liste med turtle kommandoerne Turn og Move </returns>
let fromMoveFoldBack (lst:move list): Canvas.turtleCmd list =
    List.foldBack top2 lst []

do turtleDraw (500,500) "foldBack" (fromMoveFoldBack lst)
```

List.foldBack modulet tager ligesom List.Fold en funktion, som vi kalder top2, en accumulator, der igen er den tomme liste [] og lst. top2 tager også acc og (dir,dist) som input. Den returnerer vores turtleCmd liste concatenated med med den tomme liste. Bemærk at rækkefølgen er omvendt fra List.fold modulet. Vi tegner denne liste med turtleDraw.



Figur 4: foldBack

3i1

I delopgave 3i1 er vi blevet foræret funktionen tree som tegner et fraktal træ i midten af canvas. Vi benytter denne funktion i en ny funktion kaldet randomTree, som placerer træet et tilfældigt sted på Canvas.

```
#r "nuget:diku.canvas, 1.0.1"
open Canvas

///<summary> tree tager en int vaerdi og rekursivt generere en liste. </summary>
///<param name="sz"> int vaerdi som definere st rrelsen af traeet.</param>
///<returns> output er en turtleCmd list. </returns>
let rec tree (sz: int) : Canvas.turtleCmd list =
    if sz < 5 then
        [Move sz; PenUp; Move (-sz); PenDown]
    else
    [Move (sz/3); Turn -30]
        @ tree (sz*2/3)
        @ [Turn 30; Move (sz/6); Turn 25]
        @ tree (sz/2)
        @ [Turn -25; Move (sz/3); Turn 25]
        @ tree (sz/2)
        @ [Turn -25; Move (sz/6)]
        @ [PenUp; Move (-sz/3); Move (-sz/6); Move (-sz/3)]
        @ [Move (-sz/6); PenDown]

//Vi definerer variabler til turtleDraw
let w = 600
let h = w
let sz = 100

//turtleDraw tager listen fra tree og tegner et tr   i Canvas.
turtleDraw (w,h) "Tree" (tree sz)

let rnd = System.Random ()
///<summary> randomTree tager en int vaerdi og rekursivt generere en liste. </summary>
///<param name="sz"> int vaerdi som definere stoerrelsen af traeet.</param>
///<returns> output er en turtleCmd list. </returns>
let randomTree (sz:int): Canvas.turtleCmd list=
    let (dist:int) = rnd.Next 300
    let (dir:int) = rnd.Next 360
    [PenUp; Turn(dir); Move(dist); Turn(-dir); PenDown]
    @tree sz
    @[PenUp; Turn(180+dir); Move(dist); Turn(-dir+180); PenDown]

//turtleDraw tager listen fra randomTree og laver et trae i Canvas
// hvor placeringen er tilfaeldig.
turtleDraw (w,h) "randomTree" (randomTree sz)
```

randomTree tager input sz som definerer størrelsen af træet. Den gør brug af System.Random() der genererer et tilfældigt tal vha. modulet Next. Vi genererer to tilfældige tal. Det første kalder vi dist for distance og kan have værdi mellem 0 og 300. Det andet kalder vi dir for direction og kan have værdi mellem 0 og 360. Dernæst bruger vi disse værdier samt turtle kommandoerne PenUp og PenDown til at rykke startplaceringen af træet i retningen dir med afstanden dist. Yderligere sikre vi også træet vender i den rigtige retning før vi concatenator med funktionen tree. Til sidst concatenator vi med endnu en turtleCmd liste som returnerer vores turtle til start. Dermed kan vi tegne et træ i en tilfældig position med turtleDraw.

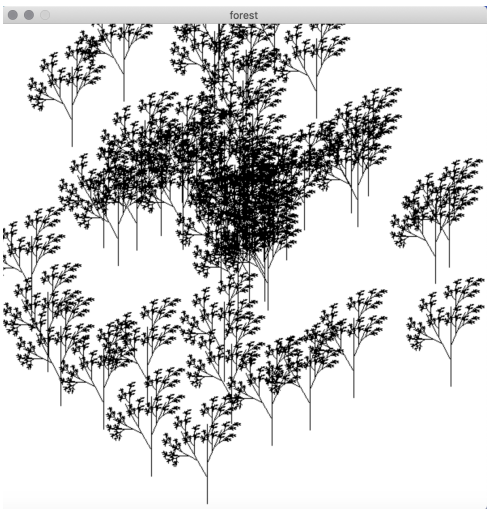
Figur 5: randomTree

Vi laver nu en ny funktion som tegner n træer med størrelse sz i en tilfældig position. Funktionen kalder vi forest og den tager input sz og n og returnere en turtleCmd liste.

```
///<summary> forest tager to int vaerdier og rekursivt generere en liste. </summary>
///<param name="sz"> int vaerdi som definere stoerrelsen af et trae.</param>
///<param name="n"> int vaerdi som definere antallet af traeer. </param>
///<returns> output er en turtleCmd list. </returns>
let rec forest (sz:int) (n:int):Canvas.turtleCmd list =
    match n with
    0 -> []
    |_-> randomTree sz
        @forest sz (n-1)

let n=50
//turtleDraw tager listen fra forest og tegner n traeer i Canvas.
turtleDraw (w,h) "forest" (forest sz n)
```

forest er en rekursiv funktion. I funktionen bruger vi match-with med n. Hvis n=0 returneres den tomme liste []. Ellers kaldes funktionen randomTree som vi concatenator med funktionskaldet forest sz (n-1) hvorved funktionen kaldes igen og igen indtil n=0. Dermed generer vi en turtle.Cmd liste med n tilfældigt placerede træer. Vi tegner disse træer vha. turtleDraw.



Figur 6: forest