Aflevering af 4g

Shatin Nguyen (hlv332), Peter Asp Hansen (glt832) & Oliver Fontaine Raaschou (vns328)

23. februar 2023

1 4g0

Ifølge opgaven, skal en robot kunne placeres på en position som er defineret type pos = int*int.

```
1 type pos = int*int //Position som bestaar af 2 int.
```

Figur 1: Defineret type = pos int*int

Opgave a

Funktionen skal tage imod to positioner og returnere et tal, defineret i linje 5. På linje 2-3 adskiller vi de enkelte elementer fra `p1:pos` og `p2:pos` så vi kan tilgås dem. Til sidst anvendes formlen $(x_2 - x_1)^2 + (y_2 - y_1)^2$ ved brug af pown funktion.[1].

```
1  ///<summary> dist tager to positioner og returnere afstanden imellem dem </summary>
2  ///<param name="p1"> Foerste position </param>
3  ///<param name="p2"> Anden position </param>
4  ///<returns> Afstand mellem de to positioner
5  let dist (p1:pos) (p2:pos) : int =
6     let (x1,y1) = p1
7     let (x2,y2) = p2
8     (pown (x2-x1) 2) + (pown (y2-y1) 2)
9  //Example
10 printfn "dist: %A" (dist (1,1) (3,3))
```

Figur 2: Løsning af 4g0a

Opgave b

Funktionen candidates skal tage en source position `(src:pos)` og en target position `(tg:pos)` og returnere en liste af positioner. Til at løse opgaven vælger vi igen at afskille source's `x,y` postion. Så laver vi en variable `a` som er afstanden fra source og target ved at anvende dist funktionen og en variable `lst` som en liste bestående af de vandrette og lodrette nabo positioner til source postionen. Vi anvender `List.filter`, hvori funktionen returnere de nabo positionerne hvor afstanden er mindre end `a`[2].

[1] Learning to Program with F, Jon Sporring, s30
[2] Learning to Program with F, Jon Sporring, s96

```
///<summary> candidates tager en source position og en target position, og returnere de
/// lodrette og vandrette nabo positionerne til source, hvor afstanden er mindre
/// </summary>
///<param name="src"> source position </param>
///<param name="tg"> target position </param>
///<returns> Returnere de nabo afstande hvori afstanden er mindre hvor afstanden er
    mindre end afstanden til target
let candidates (src:pos) (tg:pos) : pos list =
  let x,y = src
  let a = dist src tg
  let lst = [((x+1),y);((x-1),y);(x,(y+1));(x,(y-1))]
  List.filter (fun (x,y) -> (dist (x,y) tg) < a) lst
//Example
printfn "candidates: %A" (candidates (1,1) (3,3))
```

Figur 3: Løsning af 4g0b

Opgave c

Funktionen routes skal tage en source position (`src:pos`) og en target position (`tg:pos`) og returnere en liste af lister af positioner. Funktionen er rekursiv og der matches med src som stopbetingelse. Vi betragter først k som returnerer `[[tg]]` når `src=tg`. Dermed defineres endestationen i vores rute og det sidste element i hver pos liste bliver `(1,1)`. Ellers benyttes List.map modulet til a iterere over listen cand som defineres med funktion `candidates src tag`. Inden i List.map modulet benyttes List.map igen hvor den anonyme funktion `(fun j -> src::j)` som concatenator alle vores kandidater sammen til en rute (`pos list`). Til sidst bruger vi piping til at concatenate alle ruter i lister af ruter (`pos list list`).

```
///<summary> routes tager en source position og en target position og danner en liste med
     alle mulige ruter </summary>
///<param name="src"> source position </param>
///<param name="tg"> target position </param>
///<returns> en liste af lister som
let rec routes (src:pos) (tg:pos): pos list list =
  match src with
    k when src=tg -> [[tg]]
    |_->
      let cand = candidates src tg
      List.map (fun i->List.map (fun j->src::j)(routes i tg)) cand|> List.concat
//Example of use
printfn "%A" (routes (3,3) (1,1))
```

Figur 4: Løsning af 4g0c

Vi får følgende output vha. `printfn`:

```
[[(3, 3); (2, 3); (1, 3); (1, 2); (1, 1)];
[ (3, 3); (2, 3); (2, 2); (1, 2); (1, 1)];
[ (3, 3); (2, 3); (2, 2); (2, 1); (1, 1)];
[ (3, 3); (3, 2); (2, 2); (1, 2); (1, 1)];
[ (3, 3); (3, 2); (2, 2); (2, 1); (1, 1)];
[ (3, 3); (3, 2); (3, 1); (2, 1); (1, 1)]]
```

Opgave d

Vi modificere nu vores kode således, at vi i candidates funktion også kan bevæge os diagonalt. Det gør vi i en ny funktion candidates2 hvor de diagonale positioner {(x+1,y+1),(x+1,y-1),(x-1,y+1),(x-1,y-1)} er inkluderet. Dette gør vi ved at definere to lister, en liste med de gamle punkter (lst1) og en liste med de nye punkter (lst2). Dem concatenerer (@) vi til en ny liste (lst3).

```
///<summary> candidates2 tager en source position og en target position, og returnere
    alle nabo positionerne til source, hvor afstanden er mindre end afstanden
/// til target </summary>
///<param name="src"> source position </param>
///<param name="tg"> target position </param>
///<returns> Returnere de nabo afstande hvori afstanden er mindre hvor afstanden er
    mindre end afstanden til target
let candidates2 (src:pos) (tg:pos) : pos list =
     let (x,y) = src
     let a = dist src tg
     let lst = [(x+1,y);(x-1,y);(x,y+1);(x,y-1);]
     let lst2 = [(x+1,y+1);(x+1,y-1);(x-1,y+1);(x-1,y-1)]
     let lst3 = lst@lst2
     List.filter (fun i -> (dist i tg) < a) lst3
```

Figur 5: Modificerede candidates funktion

Den opdateret `routes` navngiver vi `routes2`. I den opdateret version definere vi en variablen `y` som giver listen af alle router beskrevet tidligere.
Vores ide til at løse opgaven er at finde længden af den mindste liste og filter de lister som er lig med længden. Vi anvender, `List.min` til at returnere den korteste liste[3] og `List.length` til at returnere længden af listen.[4]. Vi definere derefter en ny variable `a` som først finder den korteste liste med `List.min` af `y` og efterfølgende længden med `List.length`. Nu anvender vi `List.filter`, hvor funktionen skal tage hvert liste, og returnere de lister, hvor længden er lig med `a`.

```
///<summary> routes2 tager en source position og en target position returnere en liste af
     de korteste ruter </summary>
///<param name="src"> source position </param >
///<param name="tg"> target position </param >
///<returns > En liste med korteste liste.
let rec routes2 (src:pos) (tg:pos): pos list list =
    match src with
        k when src=tg -> [[tg]]
        |_->
            let cand = candidates2 src tg
            let y = List.map(fun i->List.map(fun j->src::j)(routes2 i tg))cand|>List.
    concat
            let a = List.length (List.min (y))
            List.filter (fun (i:pos list) -> (List.length i) = a) y
//Eksempel
printfn "routes2: %A" (routes2 (4,3) (1,1))
```

Figur 6: Opdaterede routes funktion

[3] https://fsharp.github.io/fsharp-core-docs/reference/fsharp-collections-listmodule.html
[4] Learning to Program with F, Jon Sporring, s99