9i - POP

Alexander Husted

15. december 2022

Indhold

1 Problem

Der skal laves en klasse drone, samt en klasse airpace.
Der gælder følgende funktioner for dronerne:

1. Dronerne skal flyve i en direkte route fra deres start til deres destination.
2. Flyve i en konstant hastighed målt i cm/sekund.
3. Hvis to droner er mindre end 500 cm fra hindande skal de kollidere.
4. Hvis en drone når sin destination kan den ikke længere kollidere.

2 drone

Drone består af 5 members og 3 properties. De tre properties bliver defineret af de 3 argumenter som klassen tager: start_ som består af en tuple (float*float), dest_ som består af en tuple (float*float) og speed_ som består af en float værdi.

```
1  type drone (start_ : float*float, dest_ : float*float, speed_ : float) =
2      let mutable position  = start_
3      let destination = dest_
4      let mutable speed = speed_
```

Figur 1: ClassDrone

position
position bliver returneret når man kalder member: this.pos. Den bliver oprindeligt defineret af argumentet start_

```
7      member this.pos =
8          position
```

Figur 2: this.pos

destination
destination bliver returneret når man kalder member: this.des. Den bliver defineret af argumentet dest_

```
11     member this.des =
12         destination
```

Figur 3: this.des

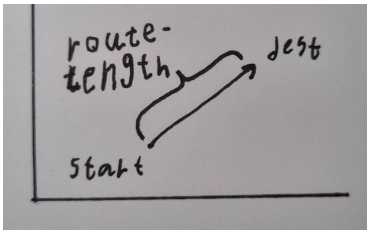
speed
Speed bliver returneret når man kalder member: this.spe. Den bliver defineret af argumentet sepeed_

```
15     member this.spe =
16         speed
```

Figur 4: this.spe

this.fly
this.fly skal returnere position efter dronen har fløjet i et sekund.
Dette gøres ved først at beregne længden af route (fig 5) ved anvendelse af formlen for euclidean distance.

$$\sqrt{(x_1 - x_2)^2 + (x_1 - x_2)^2}$$



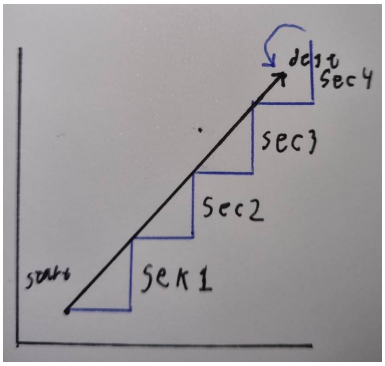
Figur 5: route

Længden divideres med hastigheden for at finde ud af hvor mange sekunder det tager at flyve hele routen.
Den oprindelige vector route divideres med tiden det tager at flyve hele routen (timefly). Herved fås en vector som symbolisere hvor langt dronen skal flyve hvert sekund. Dette lægges til position.

Problemet er nu at dronen vil fortsætte forbi destination. For at undgå dette sluttes this.fly af med et if statement: Hvis længden af routen er mindre end hastigheden så skal dronens position sætte til at være destinationenen og den nye position returneres. Ellers skal den nye position returneres.

```
19      member this.fly =
20          let route = ((fst destination) - (fst position), (snd destination) - (snd position))
21          let routeLength = sqrt(((fst route) ** 2) + ((snd route) ** 2))
22          let timeFly = routeLength / (speed)
23          let moverPrTurn = ((fst route) / timeFly), ((snd route) / timeFly)
24          position <- ((fst position) + fst (moverPrTurn)), ((snd position) + snd (moverPrTurn))
25          if routeLength <= speed
26          then
27              position <- destination
28              position
29          else
30              position
```

(a) this.fly code



(b) this.fly illustration

Figur 6: this.fly

this.atDestination
For at finde ud af hvorvidt dronen er på destinationen anvedndes et if statement: Hvis destinantionen er lig med positionen så retuner true ellers returner false.

```
34          if destination = position
35          then true
36          else false
```

Figur 7: route

3 airSpace

Airspace er den klasse som kontrollere alle dronerene på en gang. Den skal bestå af 5 members og 1 property.

```
38  type airSpace () =
39      static let mutable droneList = []
```

Figur 8: this.drones

this.drones
Returnere listen af droner.

```
42      member this.drones =
43          droneList
```

Figur 9: this.drones

this.dronesDist
Tager to argumenter drone1 og drone2, hvortil den beregner distancen mellem de to.
Dette opnås ved at definere to lokale variabler a og b, som hver opbevare en af dronernes position. Derfra anvendes formlen for euclidean distance til at beregne afstanden.

```
48      member this.dronesDist (drone1: drone, drone2: drone) =
49          let a = drone1.pos
50          let b = drone2.pos
51          let dronesDistance = sqrt((((fst a) - (fst b))**2.) + (((snd a) - (snd b))**2.))
52          dronesDistance
```

Figur 10: this.dronesDist

this.addDrone
Tager 3 argumenter, tilsvarende til klassen drone, hvortil den tilføjer en ny drone til listen droneList. Dette opnås ved at oprette en lokal variabel addedDrone, som består af en drone list med et element, nemlig klassen drone med argumenterne a, b og c tilsvarende til start_, dest_ og speed_. Den nye liste bliver sammensat med droneList og droneList returneres.

```
58      member this.addDrone(a, b, c) =
59          let addedDrone = [drone (a, b, c)]
60          droneList <- addedDrone @ droneList
61          droneList
```

Figur 11: this.addDrone

this.flyDrones
Denne member skal få alle dronerne i listen til at flyve i et sekund.
Dette opnås ved at pipe dronelist til List.map, hvorfra den køre member this.fly på alle elementer i listen.

```
64      member this.flyDrones =
65          droneList |> List.map (fun x -> x.fly)
```

Figur 12: this.flyDrones

this.collide
Denne member skal kontrollere om der er nogle af dronerne i listen, som er mindre en 500 meter fra hindanden.
Dette opnås ved at oprette tre lokale variabler:

1. **dronesInAir:** Denne variabel tager droneList og filtrerer alle de droner til hvor x.atDestination = false. Altså står man tilbage med en list udelukkende af de droner som ikke i luften.

2. **pairedDrones:** Denne variabel tager dronesInAir og opretter parvise tupler ved brug af List.allPairs. Her vil nogle af tuplerne bestå af ens droner. Derfor sorteres de tupler til hvor dronerne i første og anden koordinat ikke er ens.

3. **collided:** Nu hvor vi har en liste af tupler med alle flyvende droner. Kan de tupler hvor afstanden mellem dem er mindre end 500 sorteres til. Derfra laves alle tuplerne om til lister ved anvendelse af List.map hvorfra de bliver konkateneret til en lang liste med List.concat. Der er dog stadig et problem, en drone kan godt kollidere med mere end en drone, samt to droner kan kollidere både som (x,y) og som (y,x). Derfor anvendes List.distinct til at fjerne dubletter. Nu består collided kun af en liste over droner som kollidere.

Som side effekt til this.collide fjernes alle dronerne i collided fra dronelist. Derefter returneres listen collided.

```
68      member this.collide =
69          let dronesInAir =
70              this.drones |> List.filter (fun x -> x.atDestination = false)
71          let pairedDrones =
72              (List.allPairs dronesInAir dronesInAir) |> List.filter (fun x -> fst x <> snd x)
73          let collided =
74              pairedDrones |> List.filter (fun x -> this.dronesDist x < 500)
75              |> List.map (fun (x,y) -> [x;y]) |> List.concat |> List.distinct
76          droneList <- droneList |> List.except (collided)
```

Figur 13: this.collide

4 Assert

klassen Assert har en member: info. Info er en statisk member som tager 4 argumenter. En string et a og et b som generisk type, samt en operater, der kontrollere forholdet imellem a og b. Denne operater returnere bool(sandt/falsk).
Der bliver defineret en variabel i, som enten returnere "pass"eller "fail"afhængigt af forholdet mellem a og b. info returnere et printfn "sting: sandt/falsk"