Opgavebesvarelse 9i

Peter Asp Hansen (glt832)

23. februar 2023

I denne afleveringsopgave betragtes der droner som bevæger sig fra punkt A til punkt B med en bestemt hastighed. I forsøget på at udnytte vores viden om klasser vil vi derfor danne to klasser kaldet `Drone` og `Airspace`. Vi indleder med at beskrive nogle hjælpe funktioner som bruges i vores klasser.

```
let vectorLength (x:vec)(y:vec): int =
let len = (pown (fst y - fst x) 2 + pown (snd y - snd x) 2)
int(sqrt(float(len)+0.5))

let addPosition (a:vec)(b:vec)(s:int): vec=
    let len = vectorLength a b
    let d = float(s)/float(len)
    let x = float(fst a) + float(d) * (float(fst b) - float(fst a))
    let y = float(snd a) + float(d) * (float(snd b) - float(snd a))
    (int(x+0.5),int(y+0.5))

```

Figur 1: Hjælpefunktioner

`vectorLength` udregner afstanden mellem to vektorer x og y.
`addPosition` bestemmer en ny position baseret på hastighed og retningen fra a til b.

Vi er nu klar til at beskrive `Drone`. Denne klasse skal først og fremmest bruges til senere hen at tilføje droner til vores airspace. Da drone er et objekt, kan vi både beskrive nogle særlig egenskaber for dronen samt nogle særlige handlinger. De er henholdsvis kaldet properties og methods. Hver drone **skal** have følgende properties:

- Position (som returnerer koordinatet for startpositionen)
- Speed (som returnerer dronens hastighed)
- Destination (som returnerer den endelige destination)

Bemærk at det er tilladt at inkludere flere properties, hvilket er udnyttet. Det kommer vi ind på senere i rapporten. Inden da gennemgår vi også hurtigt de obligatoriske methods:

- Fly() (som opdatere Position dronen har bevæget sig i 1s)

  Her benyttes if-statements til at inddele i tilfælde. Hvis boolean værdien for landed eller crashed er true returneres `()` da dronen nødvendigvis ikke behøver at flyve længere. Ellers hvis afstanden mellem startpositionen og destinationen er mindre end eller lig hastighed, opdateres statussen for landed til true og p får samme værdi som d. ellers opdateres p vha hjælpefunktionen addPosition med inputs p, d og s.

- AtDestination() (returnere boolean værdierne true hvis dronen har nået sin destination ellers returneres falsk)

Yderligere er der også defineret en række mutable værdierne som vi kan opdatere vha. vores methods. Disse værdier er

- p (dronens position)
- d (dronens destination)

- s (dronens hastighed)
- landed (status for dronen, har boolean værdi true hvis p=d)
- crashed (status for dronen, har boolean værdi true hvis dronen er kollideret med en anden drone)

```
type Drone (startpos:vec,dest:vec,speed:int,name:string) =
    let mutable p = startpos
    let mutable d = dest
    let mutable s = speed
    let mutable landed = false
    let mutable crashed = false
    member this.Crashed = crashed
    member this.Crash() = crashed <- true
    member this.Position = p
    member this.Speed = s
    member this.Destination = d
    member this.AtDestination():bool = landed
    member this.Fly () =
        if (landed=true) || (crashed=true) then
            ()
        elif (vectorLength p d)<= s then
            landed <- true
            p <- d
        else
            p <- addPosition p d s
    override this.ToString () = sprintf "%A" (p,name)
```

Figur 2: klassen Drone

Vi har valgt at inkludere en ekstra property kaldet `crashed` og en ekstra method kaldet `Crash`. `crashed` tjekker om dronen er styrtet og returnere den tilhørende boolean værdi. `Crash` ændre boolean værdien af `crashed` til true aka vi styrter dronen. Bemærk at vi benytter override. Dette hjælper os med at få et læseligt output i `Airspace()` som består en drones position og navn.
Vi demonstrere lige at `Drone` virker ved at teste `Fly()`

```
let d = Drone((15,40),(40,15),2,"d")
d.Fly()
printfn "%A" d.Position
```

Dette giver os et output (16, 39) som forventet.

Vi beskriver nu klassen drone Airspace(). Denne klasse indeholder en mutable liste kaldet drones. Vi kan tilføjer droner til denne liste vha en method kaldet `addDrone`. Klassen indeholder også property kaldet `Drones` som viser listen af droner. Derudover har vi også en række methods:

- droneDist (bestemmer afstanden mellem to drone positioner)
- FlyDrones() (kalder Fly() på alle elementer i drone list)
- addDrone (drone) (allerede beskrevet)
- Collide (t) (dronerne sættes i bevægelse og der analyseres om de kollidere med hinanden)
  Vi bruger for-loops til at sammenligne alle drone par og tjekker om de kolliderer vha. en if-statement. Hvis afstanden mellem et drone par er mindre end eller lig 5 og ingen af dronerne er nået deres destination, kollidere de og de tilføjes til mutable `(Drone*Drone) list` kaldet collision. Derudover ændres `crashed` for begge droner til true. Dermede kan vi også fjerner dronerne fra `drones` vha List.filter.

```
type Airspace() =
    let mutable drones : Drone list = []
    //measures the distance between two given drones
    member this.Drones = drones
    member this.droneDist (d1: Drone)(d2:Drone) =
        let pos1 = d1.Position
        let pos2 = d2.Position
        vectorLength pos1 pos2
    //advances the position of all drones by one second
    member this.FlyDrones () =
        for i in [0..(drones.Length-1)] do
            drones.[i].Fly()
    //adds a drone to the collection
    member this.addDrone drone=
        drones <- drones@[drone]
    member this.Collide (t:int): (Drone*Drone) list =
        let mutable collision: (Drone*Drone) list = []
        for x=0 to (t-1) do
            this.FlyDrones()
            for i=0 to (drones.Length-1) do
                for j=i+1 to (drones.Length-1) do
                    if (this.droneDist (drones.[i])(drones.[j]) <= 5 && not ((drones.[i].
    AtDestination()) || drones.[j].AtDestination())) then
                        collision <- collision @ [drones.[i],drones.[j]]
                        drones.[i].Crash()
                        drones.[j].Crash()
                        ()
                    else
                        ()
            drones <- List.filter (fun d -> not d.Crashed) drones
        collision
```

Figur 3: klassen Airspace

Vi tjekker om `Airspace()` virker ved at teste `Collide(t)`

```
let AS = Airspace()
let d1 = Drone((0,40),(40,15),2,"d1")
let d2 = Drone((0,10),(0,40),2,"d2")
let d3 = Drone((0,20),(0,40),1,"d3")
AS.addDrone d1
AS.addDrone d2
AS.addDrone d3
printfn "%A" (AS.Drones)
printfn "%A" (AS.Collide 5)
printfn "%A" (AS.Drones)
```

Figur 4: test af `Airspace()`

Her tilføjes tre droner d1,d2 og d3 til vores `Airspace()` AS. Vi printer først listen `drones`. Derefter printes listen `collisions`. Til sidste printes listen `drones`. Dette giver os følgende output:

```
[((0, 40), "d1"); ((0, 10), "d2"); ((0, 20), "d3")]
[(((0, 20), "d2"), ((0, 25), "d3"))]
[((10, 35), "d1")]
```

Bemærk at d2 og d3 kolliderer og de er derfor blevet fjernet fra `drones`

Jeg havde desværre ikke tid til at lave en Assert klasse til at teste med.