

Learning to Program with F#  
Exercises  
Department of Computer Science  
University of Copenhagen

Jon Sparring, Martin Elsmann, Torben Mogensen, Christina Lioma

November 27, 2020

## 0.1 roguelike

### 0.1.1 Lærervejledningn

**Emne** Classes, Objects, Methods Attributes

**Sværhedsgrad** Middel

### 0.1.2 Introduktion

Denne opgave går ud på at lave et såkaldt retro-style *roguelike* spil. Et roguelike går ud på spilleren skal udforske en verden og løse nogle opgaver, ofte er denne verden et underjordisk fantasy *dungeon* befolket af monstre der skal nedkæmpes.

I denne opgave skal der arbejdes med at lave et object-orienteret design, som gør det nemt at udvide spillet med nye skabninger og spil-mekanismer.

Opgaven er delt i tre dele, samt en fjerde frivillig del. I den første delopgave skal der arbejdes med at implementere en *canvas* til at vise vores verden. I den anden delopgave går ud på at lave et klassehierarki til at repræsentere skabninger og genstande i verden. Endelig skal der i den tredje delopgave arbejdes mod at samme de forskellige dele til et samlet spil. Derudover er der i fjerde del en række forslag til udvidelser.

I det følgende er der kun givet minimums-krav til hvilke metoder og properties I skal implementere på jeres klasser. I må gerne lave ekstra metoder eller hjælpe-funktioner, hvis I synes det kan hjælpe til at skrive et mere elegant og forståeligt program.

### 0.1.3 Opgave(r)

#### 0.1.1: Terminal Brugergrænseflade

For at vise spillets verden implementer vi en klasse Canvas, som er et gitter af felter. Hvor feltet der er i øverste ventre hjørne har position (0,0).

Hvert felt har en char, og så *kan* feltet have en forgrundsfarve, og det *kan* have en baggrundsfarve.

Implementér en klassen Canvas som har følgende signatur:

```
type Color = System.ConsoleColor
type Canvas =
  class
    new : rows:int * cols:int -> Canvas
    member MaxX : int
    member MaxY : int
    member Set : x:int * y:int * cont:char * fg:Color *
      bg:Color -> unit
    member Show : unit -> unit
  end
```

Det vil sige:

- En konstruktør der tager antal rækker og koloner som argumenter.
- To properties der angiver det maksimale række-indeks, MaxX, og kolonne-indeks, MaxY.
- en metode Set til at sætte indhold og farver på et felt.
- en metode Show til at vise en canvas i terminalen.

I rapporten skal I beskrive jeres designovervejelser, samt redegøre for hvilken skjult tilstand en canvas har.

**Hints:** Brug følgende funktionalitet fra standard-biblioteket:

- `System.Console.ForegroundColor <- System.ConsoleColor.White` til at sætte forgrundsfarven til hvid.
- `System.Console.BackgroundColor <- System.ConsoleColor.Blue` til at sætte baggrundsfarven til blå.
- `System.Console.ResetColor()` til at sætte farverne i terminalen tilbage til normal.

### 0.1.2: Genstande og Skabninger

Vi bruger klassen Entity til at repræsentere genstande og skabninger i vores verden. Disse skal kunne renderes på en canvas. Tag udgang i følgende erklæring:

```
type Entity() =  
  abstract member RenderOn : Canvas -> unit  
  default this.RenderOn canvas = ()
```

Hvis I får behov for det må I gerne tilføje tilstand og en anden default implementering af RenderOn til Entity.

Til at repræsentere spilleren bruger vi klassen Player:

```
type Player =  
  class  
    inherit Entity  
    new : ...  
    member Damage : dmg:int -> unit  
    member Heal : h:int -> unit  
    member MoveTo : x:int * y:int -> unit  
    member HitPoints : int  
    member IsDead : bool  
  end
```

En spiller er død hvis de har mindre end nul hit points. En spiller har et maksimum hit points de kan helbredes op til.

Til at repræsentere genstande som spiller kan interagere med bruger vi den abstrakte klasse Item<sup>1</sup>:

```
type Item =  
  class  
    inherit Entity  
    abstract member FullyOccupy : unit -> bool  
    abstract member InteractWith : Player -> unit  
  end
```

---

<sup>1</sup>Kfl: Find på bedre navn

Den måde en spiller interagerer med en genstand på, er ved at gå ind i genstanden (det kommer vi tilbage til i næste delopgave). Til dette skal vi bruge `FullyOccupied` til at sige om genstanden fylder feltet helt ud eller om spilleren kan stå i samme felt som genstanden. Metoden `InteractWith` bruges dels til at genstanden kan have effekter på spilleren, og retur-værdien siger om genstanden stadigvæk skal være i verden (`true`) efter interaktionen, eller om den skal fjernes (`false`) fra verden.

Implementér følgende fire konkrete klasser der nedarver fra `Item`:

- `Wall` der fylder et helt felt, men ellers ikke har effekter på spilleren.
- `Water` der ikke fylder feltet helt ud, og helbreder med to hit points.
- `Fire` der ikke fylder feltet helt ud, og giver ét hit point i skade ved hver interaktion med spilleren. Når spilleren har interageret fem gange med ilden går den ud.
- `FleshEatingPlant` der fylder feltet helt ud, og giver fem hit point i skade ved hver interaktion med spilleren.

### 0.1.3: Verden

Implementer klassen `World`:

```
type World =  
  class  
    new : ...  
    member AddItem : item:Item -> unit  
    member Play : unit -> unit  
  end
```

Metoden `AddItem` bruges til at befolke verden med ting som spilleren kan interagere med. Typisk inden spillet går i gang.

Metoden `Play` bruges til at starte spillet, og tager sig af interaktionen med brugeren via terminalen. Spillet er tur-baseret og foregår på følgende vis:

- (a) Vis hvordan verden ser ud, samt om der er eventuelt sket noget for spilleren
- (b) Hent brugerens træk
- (c) Afgør hvilke `Items` som brugeren eventuelt interagerer med, samt hvad det betyder for hvad spillerens position og helbred er.
- (d) Hvis spilleren er død vis et afslutningsskærm billede og stop spillet, ellers start forfra.

#### Hints:

- Det er en vigtig pointe at `World` ikke tager sig af at rendere spilleren og `Items` i verden, men blot skaber en canvas, de kan render sig selv på.
- Brug `System.Console.Clear()` at fjerne alt fra terminalen inden verden vises.
- Brug `Console.ReadKey(true)` til at hente et træk fra brugeren
- Hvis `key` er resultatet fra `Console.ReadKey` så er `key.Key` lig med `System.ConsoleKey.UpArrow`, hvis brugeren trykkede på op-pilen.

### 0.1.4: Frivillige Udvidelser

- `Teleport`, lav en teleport der flytter spilleren fra et sted i verden til et (evt tilfældigt) andet sted i verden.

- Udvid Item så de kan påvirke verdenen. Fx, så kunne `FleshEatingPlant` sætte en stikling (en ny `FleshEatingPlant`) i et ledigt felt ved siden af den, hver tredje tur den ikke interagerer med spilleren.
- Monstre der kan bevæge sig rundt i verden, fx tilfældigt hvis de er langt fra spilleren, men går mod spilleren hvis de er tæt på.
- Udvid Player med et *inventory*, så man kan samle ting op i verden og flytte rundt på dem. Det kan fx bruges til at spilleren skal finde en nøgle for at komme gennem en dør.
- Udvid Canvas til at kunne vise emoji. Det kan gøres ved at hvert felt kan indeholde en string frem for kun en char, og så skal I være opmærksomme på at emoji ofte fylder det samme som to almindelige tegn.
- Lad fantasien råde