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

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

December 1, 2020

0.1 roguelike

0.1.1 Lærervejledningn

Emne Classes, Objects, Methods Attributes

Sværhedsgrad Middel

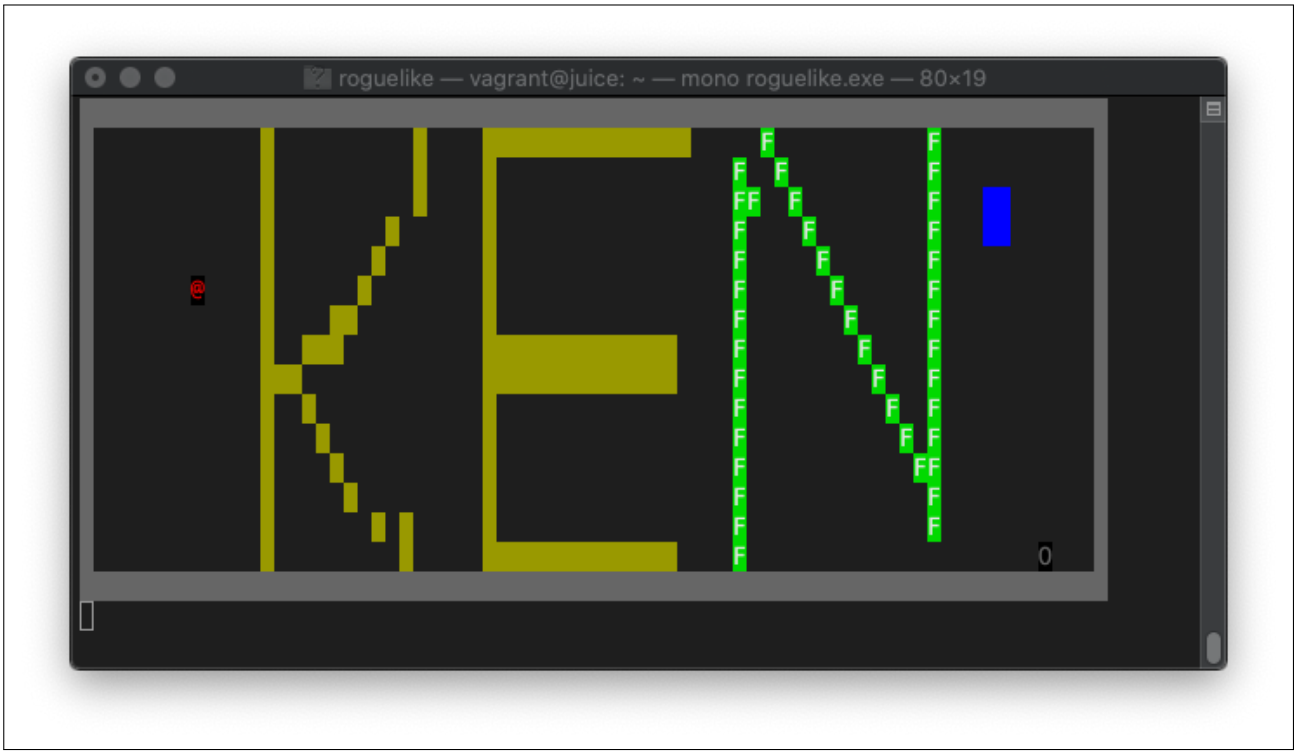
0.1.2 Introduktion

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

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

Opgaven er delt i fire dele. I den første delopgave skal der arbejdes med at implementere en *canvas* i terminalen til at vise vores verden. Anden delopgave går ud på at lave et klasse-hierarki til at repræsentere skabninger og genstande i verden. Endelig skal der i den tredie delopgave arbejdes mod at sætte de forskellige dele sammen til et samlet spil. Fjerde del indeholde en række forslag til udvidelser, hvoraf I skal implementere mindst to.

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: Brugergrænseflade i Terminal

For at vise spillets verden implementer vi en klasse `Canvas`, som er et gitter af felter. Hvor feltet i øverste ventre hjørne har position $(0,0)$, og x-koordinatet tælles op mod højre, og y-koordinatet tælles op når man bevæger sig fra top mod bund.

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 Set : x:int * y:int * c:char * fg:Color * bg:Color ->
   unit
    member Show : unit -> unit
  end
```

Det vil sige:

- En konstruktør der tager antal rækker og koloner som argumenter.
- 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 hvilke data en canvas har.

Hints: Til `Show` skal I bruge følgende funktionalitet fra standard-biblioteket:

- `System.Console.ForegroundColor <- System.ConsoleColor.White` til at sætte forgrundsfarven til hvid (kan også bruges til andre farver).
- `System.Console.BackgroundColor <- System.ConsoleColor.Blue` til at sætte baggrundsfarven til blå (kan også bruges til andre farver).
- `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 udgangspunkt 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 (data og properties), metoder 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 og skabninger, som spilleren kan interagere med, bruger vi den abstrakte klasse `Item` [1]:

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

Den måde en spiller interagere med en genstand på, er ved at gå ind i genstanden (det kommer vi tilbage til i næste delopgave). Til dette skal vi bruge `FullyOccupy` 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 fem 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.
- `Exit` vejen ud af dungeon!

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:

[1] Kfl: **Find på bedre navn**

(a) Vis hvordan verden ser ud, samt om der er eventuelt er 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 eller hvis spilleren har fundet `Exit` vis et afslutningsskærmbillede 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 rendere 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 brugere trykkede på op-pilen.

0.1.4: Udvidelser

Lav mindst 2 udvidelser til spillet og beskriv dem i jeres rapport. Følgende er nogle forslag til udvidelser, men I må gerne selv lade fantasien råde.

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