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

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

October 21, 2022

0.1 ricochet-robots

0.1.1 Teacher's guide

Emne Classes, Objects, Methods Attributes

Sværhedsgrad Middel

0.1.2 Introduction

Denne opgave går ud på at implementere en variant af brætspillet *Ricochet Robots* (først udgivet i Tyskland under navnet *Rasende Roboter* `http://en.wikipedia.org/wiki/Ricochet_Robot`).

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

Opgaven er delt i fire dele. Den første delopgave går ud på at vise en splilleplade i terminalen. Anden delopgave går ud på at lave et klasse-hierarki til at repræsentere forskellige spileelementer bl.a. robotter. 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 jer med at skrive et mere elegant og forståeligt program.

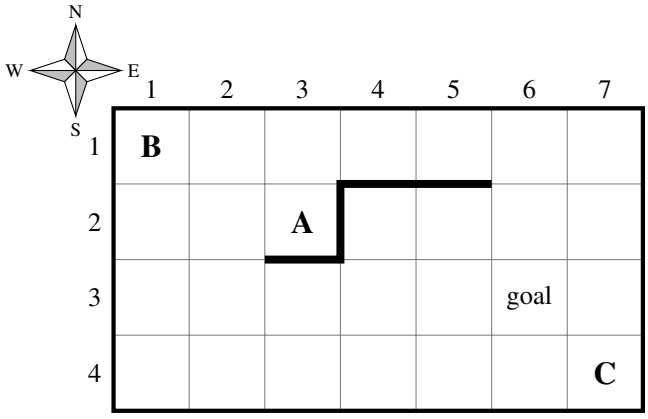
Rapport

Ud over jeres programkode skal I også aflevere en rapport (skrevet i LaTeX). I rapporten skal I beskrive implementeringen af jeres klasser, det vil sige hvilken skjult tilstand (interne variable og lignende), som jeres metoder arbejder på.

Ligeledes skal rapporten indeholde et UML diagram over klasserne i jeres løsning.

Spillets basisregler

Spillet foregår på en *plade* med $r \times c$ felter, det vil sige r rækker og c kolonner, hvor et antal *robotter* kan flyttes rundt. Hver robot starter i et separat felt på pladen, og kan derefter flyttes ved at *glide* i en af de fire retninger *nord*, *syd*, *øst* eller *vest*. Målet med spillet er at få flyttet en af robotterne hen til et *målfelt* i det laveste antal træk. Udfordringen er at når en robot starter med at glide i en retning så fortsætter den med at glide i den retning indtil, at den rammer enten en væg eller en anden robot. For at gennemføre et spil, skal en robot stoppe i et målfelt. Det tæller ikke som en løsning, hvis en robot blot glider gennem målfeltet.



Spilleplade med 4×7 felter.

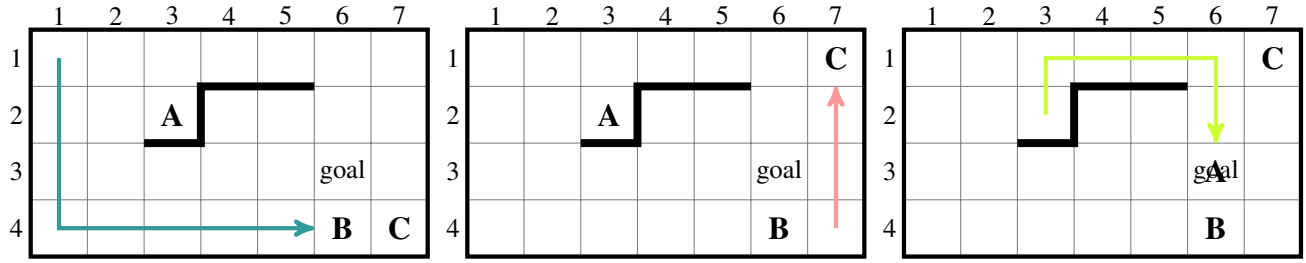
Robotter: **A** i felt (2,3), **B** i felt (1,1) og **C** i felt (3,6).

Indre vægge, tre stk:
vertikal, 1 felt, øst, startfelt (2,3);
horisontal, 1 felt, syd, startfelt (2,3);
horisontal, 2 felter, syd, startfelt (1,4).

Målfelt: (3,6)

Figure 1: Eksempel startposition

Figur **??** viser et eksempel på en startposition for et spil. Dette spil kan fx løses i seks træk: flyt **B** syd, flyt **B** øst, flyt **C** nord, flyt **A** nord, flyt **A** øst, flyt **A** syd. Der er 24 forskellige løsninger på 6 træk til spillet i Figur **??**, og ingen løsninger der er kortere end 6 træk.



0.1.3 Exercise(s)

0.1.3.1: Visning af spilleplade

For at kunne vise en spilleplade implementerer vi en klasse `BoardDisplay`, som er et gitter af felter. Hvor feltet i øverste ventre hjørne har position $(1,1)$, og første koordinat tælles op når man bevæger sig fra nord til syd (top mod bund) og anden koordinat tælles op fra vest mod øst.

En plade har altid alle ydre vægge. For at håndtere indre vægge, så kan hver felt have en nedre væg og en højre væg, men ikke andre vægge. Hvert felt kan vise en tekststreng på maksimalt to tegn. Se Figur **??** for at se en visning af spillet fra Figur **??**.

Implementér klassen `BoardDisplay` med følgende signatur:

```
type BoardDisplay =
  class
    new : rows:int * cols:int -> BoardDisplay
    member Set : row:int * col:int * cont:string -> unit
    member SetBottomWall : row:int * col:int -> unit
    member SetRightWall : row:int * col:int -> unit
    member Show : unit -> unit
  end
```

Det vil sige:

- En konstruktør der tager antal rækker og koloner som argumenter.

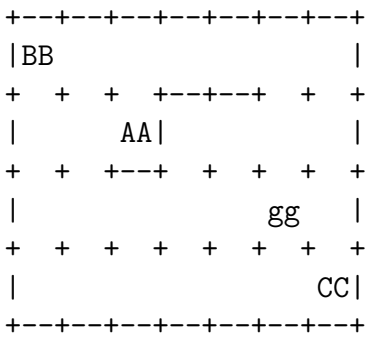


Figure 2: Eksempel på visning af spilleplade.

- en metode `Set` til at sætte indhold i et felt.
- to metoder `SetBottomWall` og `SetRightWall` til at sætte indre vægge for et felt.
- en metode `Show` til at vise en canvas i terminalen.

I rapporten skal I beskrive jeres designovervejelser, samt redegøre for hvilke data klassen `BoardDisplay` har.

0.1.3.2: Spilelementer

Vi bruger den abstrakte klassen `BoardElement` til at repræsentere spilelementer og klassen `Robot` til repræsentere robotter. Tag udgangspunkt i følgende erklæringer:

```
type Direction = North | South | East | West
type Action =
  | Stop of Position
  | Continue of Direction * Position
  | Ignore

[<AbstractClass>]
type BoardElement() =
    abstract member RenderOn : BoardDisplay -> unit
    abstract member Interact : Robot -> Direction -> Action
    default __.Interact _ _ = Ignore
    abstract member GameOver : Robot list -> bool
    default __.GameOver _ = false

and Robot(row:int, col:int, name:string) =
    inherit BoardElement()
    member this.Position = ...
    override this.Interact other dir = ...
    override this.RenderOn display = ...
    member val Name = ...
    member robot.Step dir = ...
```

Hvor typen `Direction` bruges til at angive en retning i forhold til spillepladen. Typen `BoardElement` bruges til at repræsentere spilelementer som ikke er robotter, denne klasse har tre abstrakte metoder: `RenderOn` til at rendere et element på et `BoardDisplay`; `GameOver` som bruges til at afgøre om et spil er gennemført, baseret på robotternes positioner; `Interact` som bruges at afgøre et spilelements indflydelse på en robot, inden den flyttes. Typen `Action` bruges til at specificere hvilken indflydelse et spilelement har på en robots flytning.

Klassen `Robot` bruges til at repræsentere robotter. Robotter er også spilelementer, da de kan påvirke andre robotter. Det vil sige, at klassen skal implementere metoderne fra `BoardElement`. Derudover har en robot:

- To properties: `Position` et par, (r,c), der angiver hvor på pladen robotten er og `Name` der angiver robottens navn, som blandt andet kan bruges til at rendere en robot på et `BoardDisplay`.
- Metoden `Step` som bruges at flytte robotten et felt i retningen `dir`. Dette er blot en hjælpemetode til at ændre `Position`, da `Step` *ikke* skal tage hensyn til andre spilelementer.
- Metoden `Interact` bruges til at stoppe en anden robot der forsøges at flytte ind i robotten. Fx, hvis `other` (argumenter til `Interact`, den anden robot) står i felt $(1,4)$ og er på vej vest, og robotten (`this`) står i felt $(1,3)$, så skal `Interact` returnere `Stop(1,4)`.

Implementér følgende fire konkrete klasser, der alle nedarver fra `BoardElement`, til at repræsentere indre og ydre vægge samt målfelt:

- `Goal` der skal have en konstruktor der tager to heltals argumenter, r og c, som angiver et målfelt. Metoden `GameOver` skal returnere `true`, hvis en robot er stoppet ovenpå målfeltet.
- `BoardFrame` bruges til at repræsentere rammen for en spilleplade (de ydre vægge). Klassen skal have en konstruktor der tager to heltals argumenter, r og c, som angiver størrelsen på pladen.
- `VerticalWall` bruges til at repræsentere en indre vertikal væg. Klassen skal have en konstruktor der tager tre heltals argumenter, r, c og n, som angiver startfelter for væggen, (r,c), samt længden af væggen, n. Hvis n er positiv løber væggen fra nord til syd, ellers løber den fra syd mod nord. Væggen er på østsiden af startfeltet.
- `HorizontalWall` bruges til at repræsentere en indre horisontal væg. Klassen skal have en konstruktor der tager tre heltals argumenter, r, c og n, som angiver startfelter for væggen, (r,c), samt længden af væggen, n. Hvis n er positiv løber væggen fra vest til øst, ellers løber den fra øst mod vest. Væggen er på sydsiden af startfeltet.

Hvis I får behov for det må I gerne tilføje tilstand (data og properties) samt flere metoder til alle klasser. Men de skal kunne bruges med de angivende minimums specifikationer.

0.1.3.3: Interaktion

Implementer klassen `Board`:

```
type Board =
  class
    new : ...
    member AddRobot : robot:Robot -> unit
    member AddElement : element:BoardElement -> unit
    member Elements : BoardElement list
    member Robots : Robot list
    member Move: Robot -> Direction -> unit
  end
```

Metoden `AddElement` bruges til at sætte en spilleplade op. Typisk inden spillet går i gang. Property `Elements` bruges til at få en liste af alle spilelementer (inklusiv robotter), og `Robots` bruges til at få en liste af alle robotter. Et `Board` har altid et `BoardFrame` spilelement.

Metoden `Move` bruges til at flytte en robot. En robot flyttes ved at der fortages et antal skridt med robotten i en givet retning. Inden hvert skridt løbes gennem alle spilelementer (undtagen robotten selv), og metoden `Interact` kaldes for hvert element. Hvis alle spilelementer returnere `Ignore` kan robotten flyttes eet felt i den givne retning, og robotten forsøges at flyttes endnu et felt. Hvis et spilelement returnerer `Stop pos`, stoppes robottens flytning i felt `pos` (som ikke nødvendigvis er robottens nuværende position). Hvis et spilelement returnerer `Continue dir pos` fortsætter robotten fra felt `pos` med retning `dir` (bemærk at ingen af de obligatoriske spilelementer bruger `Continue`).

Implementer klassen `Game`:

```
type Game =
  class
    new : Board -> Game
    member Play: unit -> int
  end
```

Metoden `Play` bruges til at starte spillet, og tager sig af interaktionen med brugeren via terminalen, returværdien er hvor mange træk der blev brugt. Spillet foregår på følgende vis:

(a) Vis hvordan pladen ser ud, hvor mange træk der er brugt indtilvidere, navnene på robotterne, samt evt anden information som I finder relevant.

(b) Lad brugeren vælge en robot (fx ved at skrive navnet på robotten), herefter kan robotten flyttes rundt ved brug af pile-tasterne indtil at der tastes enter.

(c) Når en robots træk er slut, får alle spilelementer mulighed for at afgøre om et spil er slut. Hvis spillet er slut, vis et afslutningsskærmbillede og stop spillet.

Klasserne `Board` og `Game`, samt de andre klasser fra de andre delopgaver, skal være i filen `robots.fs` i modulet `Robots`. Lav derudover en fil `robots-game.fsx`, der som minimum laver et spil og kalder `Play`, så vi kan prøve dit spil.

Hints:

- Det er en vigtig pointe at `Board` *ikke* holder styr på hvor de forskellige spilelementer er, det skal de selv holde styr på. Ligeledes skal `Board` *ikke* tage sig af at rendere spilelementer, men skal blot skabe et `BoardDisplay`, og bagefter bede de forskellige spilelementer om at rendere sig selv på det.
- Brug `System.Console.Clear()` at fjerne alt fra terminalen inden brugergrænseflade vises.
- Brug `Console.ReadKey(true)` til at hente tryk på piletasterne fra brugeren
- Hvis `key` er resultatet fra `Console.ReadKey` så er `key.Key` lig med `System.ConsoleKey.UpArrow`, hvis brugeren trykkede på op-pilen.

0.1.3.4: Udvidelser

Lav mindst to udvidelser til spillet og beskriv dem i jeres rapport. Følgende er nogle forslag til udvidelser, men I må gerne selv lade fantasien råde. Hvis I laver udvidelser som kræver ændringer til de eksisterende typer og klasser, så skal I diskutere hvorfor denne slags ændringer kan være problematiske i jeres rapport. Hvis I laver nye spilelementer, så skal jeres rapport indeholde en regelbeskrivelse for de nye elementer som hvis man spillede en fysisk udgave af spillet, fx hvad der sker hvis to robotter ender i samme felt (kan ikke fysisk lade sig gøre).

- Teleport, lav en teleport der flytter en robot fra et sted i verden til et andet sted i verdenen.

- Vægge der kan bevæge sig.
- Bomber som der eksploderer ved sammenstød med en robot, og fjerner nærtliggende vægge.
- Udvid `BoardDisplay` og `Game` til at kunne vise farver og emoji. Vær opmærksom på at emoji ofte fylder det samme som to almindelige tegn. Brug

  ```
  System.Console.BackgroundColor <- System.ConsoleColor.Blue
  ```

  til, fx, at sætte baggrundsfarven til blå.
- Mulighed for at loade forskellige spil/startposition fra en fil.