PoP Rapport 11g

Sebastian Juel Sefort - `xjz444`
Asbjørn Lorenzen - `hzs575`
Thomas Haulik Barchager - `jxg170`

4. januar 2022

1 Introduktion

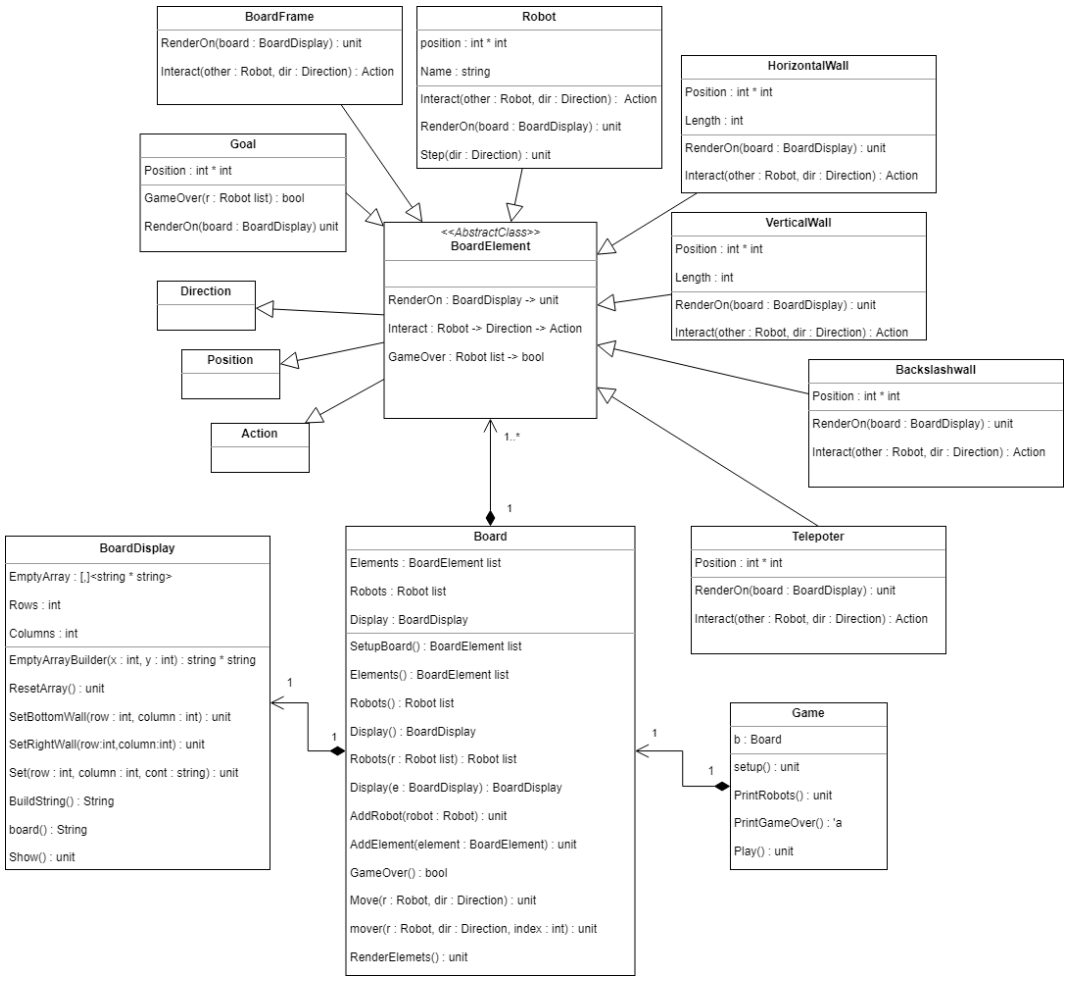
I denne rapport beskriver vi et program, der repræsenterer Spillet "Ricochet Robots"eller "Rasende Robotter". Spillet går ud på at man skal flytte ens robotter hen til målet på færrest træk muligt. Man bevæger robotterne ved at glide i en retning ind til man rammer en væg eller en anden robot.
Programmet genererer en spilleplade med tilfældige forhindringer og placerer robotterne samt målet tilfældige steder på spillepladen.

Programmet er udarbejdet i det objekt-orienteret programmeringsparadigme, og er skrevet i F#

2 Problemanalyse og design

Til at arbejde med vores "Ricochet Robots"har vi lavet mange forskellige klasser i F# til at holde styr på forskellige elementer i spillet. Udover de forskellige klasser har vi brugt 3 typer, Direction, som er en retning, Position, som en tuple af 2 ints, og Action, som fortæller om robotten skal stoppe eller fortsætte. Til at holde styr på spillets forskellige elementer har vi en abstraktklasse BoardElement, som alle boardets elementer nedarver fra. Dette gør vi nemt kan holde styr på og tilføje elementer til spillet.

Udover at have lavet mange forskellige klasser i F# har vi også lavet et UML diagram over alle klasserne og hvordan de hænger sammen. Dette gjorde at vi havde overblik over programmeringsfasen og kunne se hvad vi manglede eller hvad der havde for høj kobling. Nedenfor ses vores UML diagram over spillet "Rasende Robotter".



Figur 1: Færdig lavet UML diagram

Programmet spilles i terminalen og spilles ved hjælp af bruger input. Man kan have så mange forskellige

robotter man har lyst til, men hver robot skal have et navn på 2 karakterer.

3 Programbeskrivelse

I vores program har vi mange forskellige klasser. Den første klasse vi har arbejdet med er BoardDisplay. Denne klasse bygger selve boardet og hvordan det skal vises i terminalen. Her har vi gjort brug af et Array2D til at kunne holde styr på spillets fælter. Vi har sat det op så hvert felt er 2 linjer med 3 karakterer lang. Her er et eksempel på hvordan det skal fortolkes:

```
AA|
--+
for hvert felt:
    main string "AA|"
    bottom string "--+"
```

For at kunne få boardet til at bygge har vi gjort brug af en funktion EmptyArrayBuilder som matcher på en tuple (x,y) til at bestemme hvilken tekststreng vores Array2D skal have. Herefter bruger vi Array2D.init til at lave rammen om spillet så alle 4 kanter ikke er åbne så robotten flyver ud af siderne. Nedenunder vises hvordan BoardDisplay med størrelsen 4x7 ser ud inden elementer er tilføjet til boardet.

```
+--+--+--+--+--+--+--+
|                    |
+  +  +  +  +  +  +  +
|                    |
+  +  +  +  +  +  +  +
|                    |
+  +  +  +  +  +  +  +
|                    |
+--+--+--+--+--+--+--+
```

Efter BoardDisplay har vi lavet en abstraktklasse BoardElement. Denne klasse er til for at holde styr på at vores elementer alle har samme funktioner og kan tilføjes til boardet nemt. I vores spil har vi gjort brug af 7 forskellige element klasser. En Robot, Goal, BoardFrame, VerticalWall, HorizontalWall, Forwardslashwall og Teleporter.

Hvert element har nedarvet fra BoardElement klassen og derfor har de alle metoderne:

```
RenderOn : BoardDisplay -> unit
Interact : Robot -> Direction -> Action
GameOver : Robot list -> bool
```

I robot klassen har vi gjort brug af match-statements til at fortælle robotten hvilken vej til skal bevæge sig når brugeren har givet input. Samme har vi gjort i BoardFrame, VerticalWall og HorizontalWall, her har vi også gjort brug af match-statement til at få robotten til at stoppe med at bevæge sig når den rammer en væg. Klassen Goal er til for at have et mål, men også så en spiller kan vinde spillet. Dette sker ved at én robot er på feltet med "GO". Klassen Teleporter vil teleporterer en robot til et tilfældigt felt på boardet når robotten rammer feltet med "TP" på. Vi har som et design valg valt at robotterne alle sammen initialiseres til at være i toppen til venstre af boardet når spillets startes.

Klassen Board er lavet til at skabe et BoardDisplay og bagefter bede de forskellige spilleelementer om at rendere sig selv på det ved at kalde deres RenderOn() metode. I vores Board klasse har vi lavet en funktion SetupBoard() som instantierer alle spillets elementer, hvorefter den tilføjer dem alle til en liste med BoardElements.

Klassen Game laver et Board med (rows,cols) som størrelsen på boardet. Game har 3 metoder, PrintRobots() som printer alle robotter i spillet, PrintGameOver() som printer win-condition til terminalen og slukker programmet og til sidst har den metoden Play() som gør at vi kan spille selve spillet. Metoden Play() har 2 rekursive funktioner chooserobot() og moveloop(). chooserobot() bruges til at skifte mellem robotterne i spillet. moveloop() funktionen tager input fra spillerens piletaster og fortæller hvilken retning robotten skal bevæge sig ud fra det input. Udover at den bevæger robotten tjekker den også om man har vundet spillet eller ej.

Programmet køres ved først at compile Robots-modulet, derefter robots-game.fsx. Til sidst kører man filen med mono. Dette gøres ved at indtaste følgende kommandoer i terminalen:

```
fsharpc -a robots.fs
fsharpc -r robots.dll robots-game.fsx
mono robots-game.exe rows columns robots
```

Når man køre programmet med mono robots-game.exe, skal man angive spillets rækker og kolonner samt hvor mange robotter spillet skal have. Spillets robotter skal være angivet med store bogstaver og SKAL være 2 karaktere lang, f.eks. AA, BB, CC. Et eksempel på kørsel af programmet med mono

```
mono robots-game.exe 4 7 AA BB CC
```

Giver os et spil med 4 rækker og 7 kolonner samt 3 robotter, kan ses nedenfor.

```
+--+--+--+--+--+--+--+
|AA BB CC              |
+  +  +  +  +  +  +  +
|        TP   \        |
+--+--+--+  +  +--+  +
|TP    |GO        |     |
+  +  +  +  +  +  +  +
|     |                |
+--+--+--+--+--+--+--+
"
1: "AA"
2: "BB"
3: "CC"
```

4 Afprøvning og eksperimenter

Under programmeringsfasen har vi testet programmet hver gang vi lavede en ny klasse for at sikre os at det virkede som vi ville have det skulle. Dette har gjort vi har fundet adskillige problemer med spillet. Først skulle vi finde ud af hvordan vi kunne bruge Array2D til at holde styr på felterne og hvordan felterne skulle printes til terminalen. Herefter har vi testet hvordan robotterne bevægede sig ud fra forskellige match-statements.

Da vi skulle til at lave ekstra forhindringer lavede vi 2, Forwardslashwall og Teleporter. Forwardslashwall havde vi lidt problemer med at få den til at virker. Første gang vi prøvede dette element crashede vores program, men efter adskillige tests har vi fået det til at fungerer som vi havde tænkt. Teleporter var dog lidt sværre. Vi fandt hurtigt ud af at Teleport elementer kunne få en robot til at teleporterer oven på en anden robot. Dette skaber store problemer da robotterne ikke må være oven i hinanden. Teleporteren har et problem hvis den er lige op ad en kant/væg. Hvis en robot er på vej ind i en teleporter som vil sende robotten ind mod væggen så vil spillet bare fryse da robotten ikke kan være uden for banen.

Vi fandt også ud af at programmet nogen gange kan generere et board med Goal elementet oven i en teleporter. Som vist nedenfor kan et board ses med goal oven i en robot eller en teleporter.

```
+--+--+--+--+--+--+--+--+
|AA BB|CC DD|           |
+  +  +  +  +  +  +  +  +
|TP    |      |          |
+--+  +  +  +  +  +  +  +
|    TP       |          |
+  +  +  +  +  +  +  +  +
|          \              |
+  +  +  +  +  +  +  +  +
|                         |
+--+--+  +  +  +  +  +  +
|                         |
+  +  +  +  +  +  +  +  +
|                         |
+--+--+--+--+--+--+--+--+
"
1: "AA"
2: "BB"
3: "CC"
4: "DD"
```

Udover problemer med teleporter elementet havde vi også problemer med at få Forwardslashwall til at fungere som vi havde tænkt. De første tests fik programmet til at crashe, ignoerere væggen og sende i forkert retning. I vores version af programmet er der stadig få problemer med væggen der gør spillet crashes, vi ved ikke helt hvorfor men vi tror det har noget at gøre med at den er sat for tæt på en kant eller anden væg.

Igennem hele programmeringsfasen har vi lavet metoder i de enkelte klasser til at teste om koden er rigtigt. Vi har feks lavet en metode i klassen BoardDisplay der hedder "Print()"som meget simpelt printer funktionen EmptyArray for at tjekke om den har lavet det rigtige Array2D vi skulle bruge.

5 Diskussion og konklusion

Vi havde fået til opgave at lave spillet "Rasende Robotter". Dette program er skrevet i programmeringssproget F#. For at lave vores program har vi gjort brug af objektorienteret programmering til at holde styr på spillets elementer. Inden programmeringsfasen har vi sat os og lavet et UML diagram over spillets klasser og funktioner. Dette har gjort os kloge på hvad der skulle være klasser og hvad der skulle være funktioner. Gennem programmeringsfasen har vi gjort brug af at køre funktioner manuelt, dette har hjulpet os til at få skrevet korrekt kode så funktionerne opnår de krav og specifikationer der er givet.