Mastermind

Asibi Ayagiba, Mads Carstensen og Andreas Drivsholm

30. november 2016

1 Forord

D3nn3 *r4pp0rt* 0mh4ndl3r 1mpl3m3n73r1ng3n 4f *M4573rm1nd* 0g d3n 3r *5kr3v37* ud3lukk3nd3 1 1337-5p34k.

Just kidding...
Denne rapport omhandler implementeringen af Master Mind som et tekstbaseret spil, der kan spilles i terminalen. I den beskrives vores tanker omkring hvordan spillet kunne implementeres, samt hvordan vi rent faktisk endte med at kode det. Vi går i dybden med de vigtigere funktioner vi har brugt, og hvordan de får spillet til at virke som en helhed.

Vi vil i denne forbindelse gerne takke vores vejleder, **L***u***L***u*, for programmerings- såvel som spirituelvejledning.

2 Introduktion

Mastermind er et kodebryder spil for to spillere: en opgavestiller og en opgaveløser. Spillet som vi kender det i dag blev opfundet i 1970 af Mordecai Meirowitz og lignede papir og blyant koderbryder spillet *Bulls and Cows*[1]. Spillet kan spilles uden kendskab til matematiske kundskaber, ved brug af intuition eller tilfædig gætterig, men en algoritme lavet af Donald Knuth kan gætte en vilkårlig opgave på højest 5 gæt. I denne opgave vil vi implementere spillet Mastermind i det funktionelle programmeringssprog F#.

3 Problemformulering

Vi skal implementere spillet Master Mind i F#. Programmet skal kunne spilles bruger mod bruger, program mod bruger i valgfrie roller og program mod sig selv. I bruger mod bruger skal opgavestilleren kunne indtaste en sekvens af 4 farver, som opgaveløseren derefter skal gætte. Hver stillet opgave og gæt skal tjekkes om de er lovlige efter spillet regler. Programmet skal kunne afgøre hvornår et gæt er gyldigt i forhold til den indtastede opgave. I program mod bruger, hvor programmet har rollen som opgavestiller, skal der automatisk kunne genereres en tilfældig opgave, og brugeren skal kunne gætte på den. I det omvendte scenarie skal programmet automatisk kunne gætte brugerens opgave. I program mod program skal både den genererede opgave og gæt ske automatisk.
Spillet skal køre i tekst-mode, og der må ikke bruges grafisk brugerflade. Vi skal bruge typerne: CodeColor, Code, answer, board og player og vi skal bruge funktionerne: makeCode, guess, validate.

4 Problemanalyse og design

Vi har valgt at opdele programmet i forskellige dele. Der skal være en startmenu, hvor brugeren kan få hjælp til reglerne for Master Mind og vælge hvilken spiltype de vil starte. Herunder skal der være 4 forskellige spiltyper: bruger mod bruger, bruger mod program, program mod bruger og program mod program. Vi har gjort os mange overvejelser omkring hvordan koden kunne struktureres for at få det nødvendige gameflow, hvor man flydende vil kunne skifte mellem startmenu og spiltyper. Moduler ville være den bedste måde at implementere dette på, men da vi ikke har lært om det endnu, er dette ikke en

[1] *https://en.wikipedia.org/wiki/Mastermind_(board_game)*

mulighed. Vi tænker at organisere de forskellige muligheder under én funktion, og så navigere imellem dem med et argument til funktionen som opfanges af if-sætninger. De 5 dele vil være:

- **0** - *Startmenu*, hvorfra man kan få hjælp til spilreglerne og vælge spiltype.
- **1** - Spiltypen *bruger mod program*, hvor programmet genererer opgaven.
- **2** - Spiltypen *bruger mod bruger*.
- **3** - Spiltypen *program mod bruger*, hvor brugeren indtaster opgaven.
- **4** - Spiltypen *program mod program*, hvor programmet selv genererer opgaven og derefter gætter.

I det følgende vil vi uddybe hvordan vi regner med at hver del af hovedfunktionen kommer til at fungere.

4.1 Startmenu

Heri vil vi lave en velkomstbesked, som en opremsning af de muligheder brugeren har. Vi regner med at man enten skal kunne vælge en af spilmulighederne eller få hjælp til reglerne for Master Mind her. Vi regner med at lave et system hvori brugeren skal indtaste et tal alt efter hvilken spiltype de vil spille. Tallet vil blive brugt til at gå ind i den underdel af hovedfunktionen hvor den tilhørende spiltype ligger.

4.2 Bruger mod program

Vi har tænkt os at starte denne spiltype med at programmet automatisk genererer en tilfældig opgave, som selvfølgelig er skjult for brugeren. Derefter gives kontrollen til brugeren og der kan gættes på mulige farve-sekvenser. Hvert gæt vil generere en respons alt efter hvor mange stifter der er placeret korrekt og har den rigtige farve (sorte stifter), og hvor mange stifter der har den rigtige farve, men en forkert placering (hvide stifter). Efter hvert gæt vil der også udskrives et 'bræt' med tidligere gæt og tilhørende respons, så brugeren har et overblik. Hvis opgaven gættes, altså at der modtages 4 sorte stifter som respons, vil brugeren have vundet og en slutside vil skrives ud på skærmen. Heri regner vi med at have muligheder for hvad brugeren nu vil foretage sig.

4.3 Bruger mod bruger

Vi har tænkt os at starte denne spiltype med at bede første bruger om et input der opfylder kravene til en gyldig opgave. Programmet tjekker om det rent faktisk gør det. Hvis det ikke gør, vil vi bede om et gyldig input. Hvis programmet får et gyldig input, vil vi udskrive, at opgaven er accepteret og rydde skærmen. Derefter er det den anden brugers tur, og her skal personen gætte på opgaven som i ovenstående sektion.

4.4 Program mod bruger

Vi ønsker at computren som gætter skal kunne gætte en vilkårlig opgave på så få gæt som muligt. Vores idé til hvordan den skal kører fungerer på følgende måde:
Først vil vi lade den gætte på fire ensfarvede stifter i en rækkefølge som vi har lavet (1: Red, 2: Green, 3: Yellow, 4: Purple, 5: White, 6: Black). Valget om at gætte i rækkefølge skyldes at vi ville minimere antallet af byt med to ensfarvede stifter, ved at vide hvor de bliver placeret. Vi vil derfor derfor fortrække at bytte stifterne på plads 0 og 3 højere end at bytte stifterne på plads 0 og 1, da der er større sandsynlighed for at de har samme farve. Hvis en opgave indholder en sort stift ville computeren minimum gætte opgaven i seks gæt, da sort er den sjette farve. Computeren skal bruge de første seks gæt til at bestemme hvilke farver opgaven indholder (givet sort er med). Dette vil vi gemme i et histogram som så bliver oversat til et syvende gæt. Her vil vi bruge rækkefølgen fra før. Så hvis en opgave hedder:

$$Red; Black; Red; Black$$

vil det syvende gæt komme ud som

$$Red; Red; Black; Black \ (svarende\ til\ 1, 1, 6, 6)$$

Dette gætte vil vi validere efter antal af sorte stifer i responsen. Vi vil derefter matche vores gæt med antal af sorte og rokkere stifterne i gættet derefter. Vi vil lave vores match with ud fra håndkøring af hvilke muligheder man kan få efter at bytte om på stifterne.

4.5 Program mod program

Vi har tænkt os at starte denne spiltype med at programmet automatisk genererer en tilfældig opgave, som udskrives på skærmen, så brugeren kan se den. Derefter vil programmet prøve at løse opgaven som beskrevet ovenfor.

5 Programbeskrivelse

I dette kapitel vil vi beskrive nogle af vores vigtigste funktioner. Meget af den indsatte kode er klippet ned i længde, da for eksempel 'printfn' sætninger ikke er så vigtige at snakke om.

5.1 Globale funktioner

Vi har en del funktioner der bruges mange gange i forskellige andre funktioner. Dem har vi valgt at definere globalt, for at undgå at skulle definere dem flere gange under hver enkelt funktion de skulle bruges i.

5.1.1 randomCode

Funktionen randomCode består af variablen ***rand***, samt hjælpefunktionen **randomCodeHelper**.
Variablen ***rand*** bliver defineret ved hjælp af library funktionen **Random()**, der genererer et tilfældigt seed efter klokkeslæt.
Hjælpefunktionen **randomCodeHelper** tager et *System.Random* og et *int* input kaldet ***acc***. Den bruger Random funktionen til at finde det næste tal i seedet der ligger mellem 1 og 6. Så matches ***acc*** med enten 1-4 eller 0. Hvis det ligger mellem 1-4 matcher det tilfældigt genererede tal med enten 1, 2, 3, 4, 5 eller 6. Vi valgte en rækkefølge for farverne, så hvert tal fik en farve tilknyttet. En tilfældig farve ville blive sat foran et rekursivt kald til **randomCodeHelper** igen, men med ***acc*** talt 1 ned. Hjælpefunktionen vil altså blive ved med at køre rekursivt indtil ***acc*** er 0. Her vil den via førnævnte match returnere en tom liste, som alle farverne bliver sat ind i, fordi outputter i sidste ende vil være Farve1 :: Farve2 :: Farve3 :: Farve4 :: [].

```
let randomCode () : code =
    let rec randomCodeHelper (random : System.Random) (acc : int) =
        let random_number = random.Next(1,7)

        match acc with
        | 1 | 2 | 3 | 4 ->
            match random_number with
            | 1 -> Red    :: (randomCodeHelper random (acc-1))
            | 2 -> Green  :: (randomCodeHelper random (acc-1))
            | 3 -> Yellow :: (randomCodeHelper random (acc-1))
            | 4 -> Purple :: (randomCodeHelper random (acc-1))
            | 5 -> White  :: (randomCodeHelper random (acc-1))
            | 6 -> Black  :: (randomCodeHelper random (acc-1))
            | _ -> failwith "Kan ikke ske"
        | 0 ->
            []
        | _ -> failwith "Kan ikke ske"

    (randomCodeHelper rand 4)
```

Figur 1: Funktionen randomCode

5.1.2 makeCode

Vi har brugt funktionen **makeCode** som tager en spiller og returnerer en opgave. Til dette har vi brugt en match with som enten får inputtet *Human* eller *Computer*. Hvis makeCode får inputtet *Human* bliver koden lavet af en bruger. Her oprettes variablen ***inputCode*** som en tom liste. Brugerinputtet

læses med **ReadLine** og strengen splittes mellem hver char med funktionen **Split**. Da **Split** opretter et array oversætter vi dette med **arrayToList** og printer det ind på listen ***inputCode***. Til sidst laver vi variablen ***code*** som oversætter elementerne i ***inputCode*** (liste af chars) vha. **inputCodeToCode** og **CharToColor** til typen *codeColor*. Vi benytter en while-løkke til at tjekke om brugerinputtet er gyldigt og beder om et nyt input hvis dette ikke opfyldes. Hvis **makeCode** får inputtet *Computer* generer computeren en tilfældig kode (**randomCode**).

```
let makeCode (player : player) : code =
    match player with
    | Human ->
        let mutable inputCode = []

        inputCode <- Array.toList ((System.Console.ReadLine ()).Split(' '))

        while (userCode inputCode) = false do
            printfn "\nYou␣have␣to␣choose␣a␣valid␣color"
            printf  "\n>␣"

            inputCode <- Array.toList ((System.Console.ReadLine ()).Split(' '))

        let code = inputCodeToCode inputCode
        code
```

Figur 2: Human matchet i funktionen makeCode

5.1.3 validate

Vi har lavet funktionen **validate** som sammenligner den skjulte kode med gættet. Den returnerer *answer* som en tuple af sorte og hvide som et svar på gættet. Den benytter en **match with** til at hive det første element ud af gættet (som er et *array*) og sammenligner det med opgavens første element. Derudover er ***guessCount*** også i **validate** som tæller antal gæt.

```
let validate (hidden : code) (guess : code) : answer =
    let histoHidden = makeHisto hidden
    let histoGuess = makeHisto guess

    let rec minHisto hiddenHisto guessHisto =
        match (hiddenHisto, guessHisto) with
        | (x :: xs, y :: ys) -> (min x y) :: minHisto xs ys
        | ([], _) -> []
        | _ -> failwith "hej"

    let sumPegs = List.sum (minHisto histoHidden histoGuess)
```

Figur 3: Funktionen validate

```
let rec blackFun hidden guess =
    match (hidden, guess) with
    | (x :: xs, y :: ys) ->
                        if x = y then
                            1 + blackFun xs ys
                        else
                            0 + blackFun xs ys
    | ([], _) -> 0
    | _ -> failwith "hej"

let blackPegs = blackFun hidden guess
let whitePegs = sumPegs - blackPegs

guessCount <- guessCount + 1
if blacksValidate = 0 then
    printfn "%d:  %A" guessCount guess
else ()

(whitePegs, blackPegs)
```

Figur 4: Funktionen validate

5.1.4 guess

Vi har brugt funktionen **guess** som tager en *player* og et *board* bestående af et spils tidligere gæt. Den returnerer et nyt gæt ved input fra brugeren. Vi bruger den ikke til at få input fra computeren, da vi har brugt nogle andre funktioner. Den tjekker om inputtet svarer til ***hiddenCode***. Den sætter hvert gæt ind på *boardet* og printer *boardet*.

5.1.5 gamePlay

Funktionen **gamePlay** fungerer ved at køres med den stillede opgave som input i form af ***hiddenCode***. Brugeren laver derefter et gæt som gemmes i variablen ***humanGuess***. Gættet valideres i forhold til ***hiddenCode*** (opgaven) og outputtet gemmes i variablen ***validation*** som en tuple bestående af antal hvide * antal sorte stifter. Både gæt og ***validation***-output baseret på gættet ligges til variablen ***board***. Derefter tælles variablen ***tries*** en op, så der kan ses hvor mange gange der er gættet. Hele det her kører indtil andet element (sorte stifter) i tuplen fra ***validation*** er lig 4, altså at alle farver og positioner er korrekte. Da returneres værdien af ***tries***, så den kan bruges i den funktion der kaldte **gamePlay**.

```
let gamePlay hiddenCode =
    while snd (validation) <> 4 do
        humanGuess <- guess Human board

        validation <- (validate hiddenCode humanGuess)

        board <- board @ [(humanGuess, validation)]

        tries <- tries + 1
    tries
```

Figur 5: Funktionen gamePlay

5.2 Master Mind funktioner

Vores kode er opbygget på den måde, at det er en stor funktion, der alt efter input går ind i if-sætninger, hvori koden til de forskellige spiltyper ligger. Før disse if-sætninger er der et par underfunktioner, vi har defineret, som bruges undervejs. Dem beskriver vi her.

5.2.1 win

win er en rekursiv funktion som tager en spillertype, spiltype, computergæt og brugergæt. Funktionen printer at man har vundet og henvender sig til spilleren afhængig af om det er computeren eller en bruger. Den printer også hvor mange gæt det tog. Her benytter den sig af ***guessCounter*** hvis det er computeren og ***tries*** hvis det er brugeren. Derefter printer den en menu hvor man får mulighed for at gå til startmenuen, spille den samme spiltype eller afslutte spillet. Her har vi brugt en **match with** til at matche brugerinputtet med en af valgmulighederne. Hvis den får et ugyldigt input kører den funktionen **win** igen.

```
let rec win player gameModeChoice guessCounter tries =
      match player with
      | "Human" ->
          System.Console.Clear()
          printfn "You win!"
          printfn "It took you %d guesses!" tries

      | _ ->
          printfn "\nThe computer, Allan, wins!"
          printfn "It took it %d guesses!" guessCounter

      printfn "\nWould you like to:"
      printfn "1. Go to the startmenu?"
      printfn "2. Play the same gamemode again?"
      printfn "3. Quit the game?"
      printf ">"
      let userInput = System.Console.ReadLine ()

      match userInput with
      | "1" -> game "0"
      | "2" -> game gameModeChoice
      | "3" -> System.Environment.Exit 0
      | _ -> win player gameModeChoice guessCounter tries
```

Figur 6: Funktionen win

5.2.2 compGuess

compGuess er en funktion som indholder idéen bag vores computer som gætter. I de første et til seks gæt finder computeren ud af hvilke farvestifter der er i opgaven. Det gøres ved at gætte på en farve ad gangen, som fx: rød, rød, rød, rød. Vi har valgt at gætte i vores bestemte rækkefølge hvor rød er først og sort er sidst, for at mennesker omrokeringer i **compRearrange**. Vi har også fået løkken til er terminere, hvis alle farverne i svaret er fundet. Hvis de farvede stifter er fundet efter fx farven gul stopper while-løkken efter trejde gæt, fordi gul er den tredje farve og vi eliminerer tre ubruglige gæt på farverne lilla, hvid og sort. Vi gemmer oplysninger om antal rigtige farver i histogrammet ***temp***.

```
let mutable whiteCounter = 0
let mutable n = 0
let colors = [Red; Green; Yellow; Purple; White; Black]
while (whiteCounter < 4) && (n < 6) do
    guessCode <- [colors.[n]; colors.[n]; colors.[n]; colors.[n]]
    temp.[n] <- snd (validate hiddenCode [colors.[n]; colors.[n];
                                          colors.[n]; colors.[n]])
    whiteCounter <- whiteCounter + temp.[n]
    n <- n + 1

guessCode <- []
for i = 0 to 5 do
    while temp.[i] > 0 do
        guessCode <- List.append guessCode [colors.[i]]
        temp.[i] <- temp.[i] - 1
```

Figur 7: Udvalgte dele af funktionen compGuess

Efter løkken med de første 1-6 gæt afsluttes oversætter vi histogrammet til et "7."gæt (selvfølgelig kun det 7. hvis svaret indholder sort). Vi indfører også variablen ***blacks*** som tæller hvor mange sorte der er i et gæt. Den bruger vi i funktionen **compRearrange**. Efter 7. gæt bliver de rigtige farvede stifter omrokeret i funktionen **compRearrange**.

5.2.3 compRearrange

Vi har lavet funktionen **compRearrange** til at omrokere de rigtige farvede stifter efter 7. gæt. Her matcher vi antal sorte fra forrige gæt og omrokerer efter dette. Idéen bag funktionen er hele tiden at få flere sorte end tidligere, så ***guessCode*** kan blive matchet med flere sorte. Antal af sorte bliver gemt i ***blacks***. Når computeren får 4 sorte afslutter vi funktionen **compRearrange** og funktionen **win** bliver kaldt.
Med mulighed for fejl kan **compRearrange** gætte en vilkårlig opgave på højest 13 gæt.

5.3 Master Mind spiltyper

I dette underkapitel beskriver vi de funktionsdele, der køres alt efter hvilket input spillet er kørt med og hvilken if-sætning der fanger dette input.

5.3.1 Startmenu

På startmenuen har vi valgt at printe en introduktion til spillet der giver bruger mulighed for at sætte spillet i gang eller benytte sig af 'help' til at fortælle hvordan man spiller spillet. Hvis man vælger 'start' får man 4 muligheder for spilletyper: Bruger mod computer, bruger mod bruger, computer mod bruger og computer mod computer.

5.3.2 Bruger mod program

Her laver computeren en kode og brugen skal så gætte koden. Variablen ***hiddenCode*** bliver lavet med funktionen **makeCode** som bliver kaldt med inputtet *Computer*. Her genereres der en tilfædig kode med funktionen **randomCode**. Så skal bruger gætte og inputtet bliver gemt i ***board*** som bliver printet til skærmen. Til dette bruger vi funktionen **guess** og funktionen **gamePlay**.

```
let hiddenCode = makeCode Computer

let humanNumberOfTries = gamePlay hiddenCode
win "Human" "1" 0 humanNumberOfTries
```

Figur 8: Bruger mod Program

5.3.3 Bruger mod bruger

Opgaven laves gennem variablen ***hiddenCode***, der defineres som outputtet fra funktionen **makeCode** givet argumentet *Human*. Derefter defineres variblen ***humanNumberOfTries*** som outputtet fra funktionen **gamePlay** givet argumentet ***hiddenCode***. Dette output er af typen *int* og er et tal på hvor mange gæt der er foretaget. Herefter kaldes funktionen **win** med argumenterne ***"Human"***, ***"2"***, ***0*** og ***humanNumberOfTries***, fordi brugeren har vundet.

5.3.4 Program mod bruger

Funktionsdelen 'Program mod Bruger' starter med at ændre variablen ***guessCount*** til 0. Denne er lavet som en global variabel, så det er vigtigt at den nulstilles ved spilstart. Derefter laves opgaven gennem variablen ***hiddenCode***, der defineres som outputtet fra funktionen **makeCode** givet argumentet *Human*. Dette er af typen *codeColor list*. Funktionen **compGuess** kaldes herefter med argumenterne ***hiddenCode*** og ***"3"***, og den kører indtil programmet har vundet.

```
guessCount <- 0

let hiddenCode = makeCode Human

compGuess hiddenCode "3"
```

Figur 9: Program mod Bruger

5.3.5 Program mod program

Funktionsdelen 'Program mod Program' starter med at ændre variablen ***guessCount*** til 0. Denne er lavet som en global variabel, så det er vigtigt at den nulstilles ved spilstart. Derefter genereres opgaven gennem variablen ***hiddenCode***, der defineres som outputtet fra funktionen **makeCode** givet argumentet *Computer*. Dette er af typen *codeColor list*. Funktionen **compGuess** kaldes herefter med argumenterne ***hiddenCode*** og ***"4"***, og den kører indtil programmet har vundet.

6 Afprøvning og eksperimenter

Vi vil afprøve vores kode med white-box testing. Hvor vi vha test cases vil afprøve vores kode.

6.1 Afprøvning af globale funktioner

I dette underafsnit vil vi beskrive tests på nogle af de vigtigste globale funktioner i vores program. Koden vi bruger til at teste er inkluderet med afleveringen og ligger i filen: ***blackBoxTesting.fsx***

6.1.1 charToColor

Når vi tester **charToColor** laver vi et sæt af test cases. Dette sæt indeholder tupler, som hver igen indeholder et input og det forventede output når **charToColor**. Vi printer den boolske sammenligning

```
for i = 0 to testCases.Length - 1 do
    printfn "____test_%d_-_%b" (i+1) ((charToColor (fst testCases.[i])) =
                                      (snd testCases.[i]))
```

Figur 10: Test af charToColor

6.1.2 inputCodeToCode

Vi tester ligesom i **charToColor**.

6.1.3 randomCode

Kan ikke testes, da man ikke kan forudsige hvilken kode **randomCode** vil returnere. Vi tester den dog indirekte i forbindelse med *Afprøvning af spillemuligheder*.

6.1.4 userCode

Vi tester ligesom i **charToColor**.

6.1.5 makeCode

makeCode funktionen kan ikke testes med kode da den kræver brugerinput. Vi tester den dog indirekte i forbindelse med *Afprøvning af spillemuligheder*.

6.1.6 makeHisto

Vi tester ligesom i **charToColor**.

6.1.7 validate

Vi tester ligesom i **charToColor**.

6.1.8 playersInputCorrect

Vi tester ligesom i **charToColor**.

6.1.9 playersInputCorrectStartScreen

Vi tester ligesom i **charToColor**.

6.1.10 guess

Funktionen benytter sig af et brugerinput, så det er ikke muligt at lave blackbox test på guess funktionen. Vi tester den dog indirekte i forbindelse med *Afprøvning af spillemuligheder*.

6.1.11 gamePlay

Funktionen tager et bruger input og kan derfor ikke blive testet med blackbox testing. Vi tester den dog indirekte i forbindelse med *Afprøvning af spillemuligheder*.

6.2 Afprøvning af spillemuligheder

I dette underafsnit vil vi beskrive tests med de spillemuligheder vi har tilgængelige i det fulde spil. Disse kan ikke white-box testes, da de alle kræver brugerinput, så de ikke kan køres igennem udelukkende med kode. Derfor beskriver vi snarere hvordan vi har testet fysisk.

6.2.1 Bruger mod program

Vi har testet om programmet generer en ny tilfældig kode hver gang. Det har vi gjort 10 gange.
Computergenerede opgaver:

```
#1  It is: [Black; Black; Purple; Red]
#2  It is: [Black; Black; Red; Green]
#3  It is: [Black; Green; White; White]
#4  It is: [Red; Red; Black; Black]
#5  It is: [Yellow; Purple; Red; Green]
#6  It is: [Purple; Red; Purple; Green]
#7  It is: [Black; White; Green; Yellow]
#8  It is: [Red; Green; Yellow; Green]
#9  It is: [Green; Red; Yellow; Green]
#10 It is: [Green; White; White; Black]
```

De er altså tydeligvis forskellige. Man kan herefter gætte som man skal og her fanges forkerte input og man bedes indtaste gyldigt input. Når man giver et gæt der er ens med opgaven vinder man spillet som man skal.

6.2.2 Bruger mod bruger

Det er lidt svært at teste denne spilmulighed. Opgaven kan genereres på rigtig måde, forkerte input fanges og skal skrives korrekt før man kan komme videre. Man kan gætte som man skal og her fanges forkerte input også. Når man giver et gæt der er ens med opgaven vinder man spillet som man skal.

6.2.3 Program mod bruger

Her vil vi teste om en given opgave giver samme antal gæt, ved først at udregne hvor mange gæt computeren burde bruge og derefter køre programmet med den givne kode.

Opgave som ikke behøves omrokeringer:

[Red, Green, Yellow, Black]

I denne opgave regner vi med at computeren vil bruge de første seks gæt på at finde de rigtige farver, fordi sort er med. Det syvende gæt vil være det samme som den skjulte kode, så validate vil give 4 sorte stifter som respons. Så bliver koden matchet med 4 sorte i compRearrange og win funktionen bliver kaldt.

Computerens faktiske gæt:

```
1:  [Red; Red; Red; Red]
2:  [Green; Green; Green; Green]
3:  [Yellow; Yellow; Yellow; Yellow]
4:  [Purple; Purple; Purple; Purple]
5:  [White; White; White; White]
6:  [Black; Black; Black; Black]
7:  [Red; Green; Yellow; Black]
```

Vi tester om forskellige input bliver matchet korrekt med antallet af sorte svarstifter i compRearrange ved at printe en besked, hvor vi regner med at den kører.

Opgave med 2 ensfarvede stifter:

[Purple, Yellow, Yellow, Green]

Her vil computeren bruge de første 4 gæt på at finde farverne. Det 5 gæt vil den gætte på stifterne i rækkefølge: grøn, gul, gul, lilla. Her vil den matches med 2 i compRearrange og bytte grøn og lila, hvorefter den er korrekt.

Computerens faktiske gæt:

```
1:  [Red; Red; Red; Red]
2:  [Green; Green; Green; Green]
3:  [Yellow; Yellow; Yellow; Yellow]
4:  [Purple; Purple; Purple; Purple]
5:  [Green; Yellow; Yellow; Purple]
6:  [Purple; Yellow; Yellow; Green]
```

Opgave med 4 ensfarvede stifter:

[Purple, Purple, Purple, Purple]

Fordi lilla er den 4 farve regner vi med at computeren gætter det på 4 gæt.

Computerens faktiske gæt:

```
1:  [Red; Red; Red; Red]
2:  [Green; Green; Green; Green]
3:  [Yellow; Yellow; Yellow; Yellow]
4:  [Purple; Purple; Purple; Purple]
```

6.2.4 Program mod program

Her vil vi teste om programmet generer en tilfældig ny opgave hver gang spiltypen vælges. Og derefter at den løser opgaven på samme antal gæt som vi ville forudse. Vi gør dette 10 gange.
Computergenerede opgaver:

```
#1  It is: [Purple; Black; Red; Green]
    Den burde gætte det på 9 gæt. Den tog: 9 gæt.
#2  It is: [Red; Black; Red; Black]
    Den burde gætte det på 9 gæt. Den tog: 9 gæt.
#3  It is: [Red; White; Red; Purple]
    Den burde gætte det på 9 gæt. Den tog: 9 gæt.
#4  It is: [Green; White; Purple; Black]
    Den burde gætte det på 9 gæt. Den tog: 9 gæt.
#5  It is: [Purple; White; Red; Black]
    Den burde gætte det på 13 gæt. Den tog: 13 gæt.
#6  It is: [Yellow; Black; Yellow; Yellow]
    Den burde gætte det på 9 gæt. Den tog: 9 gæt.
#7  It is: [Yellow; Purple; Purple; Yellow]
    Den burde gætte det på 8 gæt. Den tog: 8 gæt.
#8  It is: [Red; Green; Yellow; Red]
    Den burde gætte det på 6 gæt. Den tog: 6 gæt.
#9  It is: [Purple; Yellow; Red; Green]
    Den burde gætte det på 11 gæt. Den tog: 11 gæt.
#10 It is: [Green; Black; Yellow; Green]
    Den burde gætte det på 9 gæt. Den tog: 9 gæt.
```

Opgaverne er tilfældige og forskellige. Og programmet løser opgaven med samme antal gæt som vi regnede med.

7 Diskussion og konklusion

Som beskrevet i afsnit 4 har vi opdelt vores program i forskellige dele under samme funktion, såsom startmenu og spilletyper, for frit at kunne hoppe imellem dem ved hjælp af argumenter til hovedfunktionen. Dette giver muligheder som f.eks. at kunne spille samme spiltype igen efter endt spil, eller at kunne gå til startmenuen efter at have vundet.
Vi har med funktionen **makeCode** gjort det muligt enten at generere en opgave automatisk eller selv indtaste en opgave der skal gættes, alt efter hvilket argument der gives.
Med funktionen **gamePlay** har vi gjort det muligt for en bruger i enten 'Bruger mod Bruger' eller 'Bruger mod program' at afgive gæt og automatisk få disse valideret samt at få respons indtil opgaven er løst.
Med funktionen **compGuess** og underfunktionen **compRearrange** har vi lavet en computer der kan gætte en vilkårlig Master Mind opgave på under 13 gæt.
Ved brug af disse funktioner samt en masse mindre, har vi formået at skabe et fuldt funktionelt tekstbaseret Master Mind program, som både kan spilles af brugere, men også af programmet selv. Vi kalder robotten der gætter for Allan.

Appendix A: Brugervejledning

Når spillet startes introduceres brugeren for en menu, hvor man enten kan skrive 'start' eller 'help'.

Start

Ved at skrive 'start' og trykke ENTER får man muligheden for at vælge mellem 4 spiltyper:
Ved at skrive '1' og trykke på ENTER vælger man 'Bruger mod Program'.

Ved at skrive '2' og trykke på ENTER vælger man 'Bruger mod Bruger'.
Ved at skrive '3' og trykke på ENTER vælger man 'Program mod Bruger'.
Ved at skrive '4' og trykke på ENTER vælger man 'Program mod Program'.

Help

Ved at skrive 'help' og trykke ENTER bliver man mødt af en vejledning i spillet regler. Herefter kan man vælge spiltype ligesom under 'start'.

Bruger mod Program

Hvis brugeren vælger 'Bruger mod Program' bliver man mødt af en meddelelse som informerer om at computeren har genereret en opgave. Tryk herefter ENTER for at starte spillet.
Herefter står der antallet af gæt, der er foretaget indtil videre. Under dette kan man se et 'bræt' bestående af tidligere gæt samt respons herpå, i form af 2 tal der repræsenterer henholdsvis antal af hvide og sorte stifter.
Nu skal brugeren gætte på en farvekombination. Dette gøres ved at indtaste 4 forbogstaver for de farver, man vil gætte på. Disse **SKAL** være adskilt af mellemrum.

Farvenavnene er skrevet på engelsk og de gyldige input at vælge imellem er som følger:

(R)ed, (G)reen, (Y)ellow, (P)urple, (W)hite, (B)lack

Et eksempel ville være: R W B Y eller r w b y. Om bogstaverne er store eller små gør altså ingen forskel. Den snu ræv vil herfra gå taktisk til værks og ellers kan der gættes tilfældigt indtil opgaven er løst.

Når opgaven løses bliver man lykønsket med beskeden 'You win!' og en besked om hvor mange gæt det tog. Herfra kan brugeren vælge mellem tre muligheder for hvad der efterfølgende skal ske. Mulighederne er som følger:

1. Gå til startmenu
2. Spil den samme spiltype
3. Afslut spillet

Disse muligheder vælges ved at skrive det respektive tal og derefter trykke ENTER.

Bruger mod Bruger

Hvis brugeren vælger 'Bruger mod Bruger' bliver man mødt af en meddelelse om at man nu skal lave den skjulte opgave. Dette gøres ved at indtaste 4 forbogstaver for farver, se **Bruger mod Program**.
Ved ugyldig indtastning bliver der skrevet en besked om at man skal vælge gyldige farver. Ved gyldig indtastning bliver der skrevet en besked om at koden er accepteret.
Herefter kan der trykkes ENTER for at rydde skærmen, og så er spillet klar til kodebryderen.
Selve gættefasen foregår ligesom i **Bruger mod Program**.

Program mod Bruger

Hvis brugeren vælger 'Program mod Bruger' bliver man ligesom i **Bruger mod Bruger** mødt af en meddelse om at man skal lave en skjult opgave. Dette foregår på samme måde.
Herefter kan der trykkes ENTER for at 'give kontrollen' over til programmet.
Skærmen bliver nu fyldt af programmets process med at løse opgaven. Hver linje er opbygget som:

$$\textit{Gætnummer: } [Farve1; Farve2; Farve3; Farve4]$$

Herefter lykønskes computeren, Allan, og der står hvor mange gæt han brugte.

Nu har brugeren følgende muligheder:

1. Gå til startmenu
2. Spil den samme spiltype
3. Afslut spillet

Disse muligheder vælges ved at skrive det respektive tal og derefter trykke ENTER.

Program mod Program

Hvis brugeren vælger 'Program mod Program' bliver man mødt af en meddelelse som informerer om at computeren har genereret en opgave, og også hvad opgaven er. For eksempel:

It is: [White; Green; Green; Green]

Herefter kan der trykkes ENTER for at 'give kontrollen' over til programmet.
Skærmen bliver nu fyldt af programmets process med at løse opgaven. Hver linje er opbygget som:

$$Gætnummer: [Farve1; Farve2; Farve3; Farve4]$$

Herefter lykønskes computeren, Allan, og der står hvor mange gæt han brugte.

Nu har brugeren følgende muligheder:

1. Gå til startmenu
2. Spil den samme spiltype
3. Afslut spillet

Disse muligheder vælges ved at skrive det respektive tal og derefter trykke ENTER.

Appendix B: Programtekst

```
// POP opgave 8 - Mastermind i F#
// Skrevet af Asibi Ayagiba, Mads Carstensen og Andreas Drivsholm

// De praefinerede typer vi skal bruge i vores kode.
type codeColor =
    Red | Green | Yellow | Purple | White | Black
type code = codeColor list
type answer = int * int
type board = (code * answer) list
type player = Human | Computer

/// <summary>Denne funktion soerger for at faa konverteret et bogstav om
    til
/// den passende farve.</summary>
/// <remarks>Wildcardet bliver aldrig matchet, da denne funktion altid
/// bliver kaldt med enten "R", "r", "G", etc, pga hjaelpefunktionen
/// legitColors soerger for at input altid bliver tjekket inden det bliver
/// kaldt som argument til charToColor</remarks>
/// <params>charToColor tager et parameter med navnet color som er af typen
/// string</params>
/// <returns>charToColor returnerer en farve af typen codeColor</returns>
let charToColor (color : string) =
    match color with
    | "R" | "r" -> Red
    | "G" | "g" -> Green
    | "Y" | "y" -> Yellow
    | "P" | "p" -> Purple
```

```
    | "W" | "w" -> White
    | "B" | "b" -> Black
    | _ -> failwith "Kommer ikke til at ske."

/// <summary>inputCode oversaetter en liste af bogstaver som definerer
    farver
/// til en liste af egentlige farver, altsaa er typen codeColor</summary>
/// <params>Tager et input koden som paramterer</params>
/// <returns>Returnerer en liste af typen code som er defineret som
    vaerende
/// en liste med elementer af typen codeColor.</returns>
let rec inputCodeToCode inputCode : code =
    match inputCode with
    | [] -> []
    | x :: xs -> charToColor x :: inputCodeToCode xs


/// <summary>Genererer en tilfaeldig kode. Bliver brugt naar computer
    fungerer
/// som opgavestilleren.</summary>
/// <remarks>Opretter et nyt object rand, som er defineret ved klassen
    Random.
/// Vi bruger klassens metode Next() til at generere tal mellem 1 og 6, med
/// begge tal inklusive. Vi skriver Next(1,7), men her er syvtallet
    eksklusivt.
/// </remarks>
/// <params>Denne funktion tager inten parametre</params>
/// <returns>Returnerer en en "kode". Det er en liste af 4 elementer, hvor
/// hvert element er af typen codeColor. Denne liste af farver er
    tilfaeldige
/// </returns>
let randomCode () : code =
    let rand = new System.Random()

    let rec randomCodeHelper (random : System.Random) acc =
        let random_number = random.Next(1,7)

        match acc with
        | 1 | 2 | 3 | 4 ->
            match random_number with
            | 1 -> Red    :: (randomCodeHelper random (acc-1))
            | 2 -> Green  :: (randomCodeHelper random (acc-1))
            | 3 -> Yellow :: (randomCodeHelper random (acc-1))
            | 4 -> Purple :: (randomCodeHelper random (acc-1))
            | 5 -> White  :: (randomCodeHelper random (acc-1))
            | 6 -> Black  :: (randomCodeHelper random (acc-1))
            | _ -> failwith "Kommer ikke til at ske."
        | 0 ->
            []
        | _ -> failwith "Kommer ikke til at ske."

    (randomCodeHelper rand 4)


/// <summary>Denne funktion tjekker om brugerens input er korrekt.</summary
    >
/// <params>Den tager et parameter af typen code</params>
/// <returns>Returnerer et bools true eller false, alt efter om brugerens
```

```
/// input er hhv. korrekt eller ikke korrekt.</returns>
let userCode (code : string list) =
    let legitColor col =
        match col with
        | "R" | "G" | "Y" | "P" | "W" | "B" -> true
        | "r" | "g" | "y" | "p" | "w" | "b" -> true
        | _ -> false

    let mutable returnValue = true

    if code.Length <> 4 then
        false
    else
        for i = 0 to code.Length - 1 do
            if (legitColor code.[i]) = false then
                returnValue <- false

        returnValue

/// <summary>makeCode genererer en kode af typen code.</summary>
/// <remarks>Funktionen har to branches, alt afhaengig om det er computeren
/// der fungerer som "codemaker" eller om det er en bruger.</remarks>
/// <params>Tager som parameter en spiller type som kan vaere enten Human
    eller
/// Computer</params>
/// <returns>Returnerer en kode af typen code</summary>
let makeCode (player : player) : code =
    match player with
    // Hvis "codemaker" er Computer, bliver funktionen randomCode kaldt.
    // randomCode returnerer en kode af typen code.
    | Computer ->
        randomCode ()
    // Hvis "codemaker" er Human, bliver denne branch koert.
    | Human ->
        let mutable inputCode = [] // variabel til at indeholde brugerens
            input
        printfn "\nPlease␣create␣a␣secret␣code."
        printfn "You␣can␣choose␣from␣the␣following␣colors:"
        printfn "Red,␣Green,␣Yellow,␣Purple,␣White␣and␣Black"
        printfn "Please␣enter␣four␣colors␣with␣a␣space␣between␣each,"
        printfn "and␣each␣denoted␣by␣the␣first␣character␣in␣the␣name␣of␣the
            ␣color"
        printfn "Example:␣B␣or␣b␣=␣Black"
        printfn "Example:␣r␣g␣y␣b"
        printf "\n>␣"

        // inputCode saettes til at vaere lige med brugerens input.
        // Brugerens input bestaar af 4 tegn separeret af mellemrum. Disse
            tegn
        // bliver udvindet af tekststrengen ved hjaelpe af metoden Split,
            som
        // foraarsager at inputet bliver omdannet til en liste af tegn.
        // Denne liste gemmes i den muterbare liste inputCode.
        inputCode <- Array.toList ((System.Console.ReadLine ()).Split(' '))

        // Funktionen userCode tjekker om brugeren input er korrekt.
        // Hvis inputtet ikke er korrekt, spoerges der om nyt input.
        while (userCode inputCode) = false do
```

```
            printfn "\nYou␣have␣to␣choose␣a␣valid␣color"
            printf  "\n>␣"

            inputCode <- Array.toList ((System.Console.ReadLine ()).Split('
                ' ))

        // Naar inputtet er korrekt, kaldes funktionen inputCodeToCode med
        // inputCode som argument. Dette konverterer koden fra at vaere en
            liste
        // af tegn, til at vaere en liste af typen code.
        let code = inputCodeToCode inputCode
        code

/// <summary>makeHisto laver et histo gram over en liste af farver. Lad
/// i = 1,2,3...6. Histo.[i] indeholde en heltal for hvor ofte farve i
/// forekommer. Talvaerdien for farven og selv farven, hoerer sammen som
/// de blev defineret oeverst i programkoden. Saa 0 = Red, 1 = Green, etc.
/// </summary>
/// <params>makeHisto tager et parameter af typen code</params>
/// <returns>makeHisto returnerer et histogram af farverne for den
    paagaeldende
/// liste</returns>
let makeHisto (code : code) =
    let histo = (Array.init 6 (fun _ -> 0))

    for i = 0 to code.Length - 1 do
        if code.[i] = Red then
            histo.[0] <- histo.[0] + 1
        elif code.[i] = Green then
            histo.[1] <- histo.[1] + 1
        elif code.[i] = Yellow then
            histo.[2] <- histo.[2] + 1
        elif code.[i] = Purple then
            histo.[3] <- histo.[3] + 1
        elif code.[i] = White then
            histo.[4] <- histo.[4] + 1
        else
            histo.[5] <- histo.[5] + 1

    (Array.toList histo)

let mutable guessCount = 0
let mutable blacksValidate = 0

/// <summary>validate validerer om hvor taet computerens eller brugerens
    gaet er
/// paa at vaere det rigtige - i form af hvide og sorte pinde.</summary>
/// <params>Tager to parametre. hidden er en den skjulte kode, og guess er
    den
/// kode som computer eller brugeren har gaettet</params>
/// <returns>Returnerer en tuple af typen answer som er en tuple af typen
/// (int * int). Det foerste element i tuplen er antallet af hvide pinde,
    og det
/// andet element i tuplen er antallet af sorte pinde.</returns>
let validate (hidden : code) (guess : code) : answer =
    // Dette er de to variable som lagrer histogrammerne for den skjulte
        kode
    // og brugerens/computerens gaet.
```

```
    let histoHidden = makeHisto hidden
    let histoGuess = makeHisto guess

    // Her sammenligner men de to histogrammer. Summen af pinde (hvide +
       sorte
    // pinde) regne ved at finde min(hiddenHisto[i], guessHisto[i]) for
    // i = 0,1,2,..5
    let rec minHisto hiddenHisto guessHisto =
        match (hiddenHisto, guessHisto) with
        | (x :: xs, y :: ys) -> (min x y) :: minHisto xs ys
        | ([], _) -> []
        | _ -> failwith "Kommer␣ikke␣til␣at␣ske"

    // sumPegs indeholder summen af hvide og sorte pinde
    let sumPegs = List.sum (minHisto histoHidden histoGuess)

    // Antallet af sorte pinde findes med blackFun funktionen.
    // For i = 0,1,2,3, hvis hidden[i] = guess[i] er den i'te pind den
       rigtige
    // farve paa det rigtige sted.
    // funktionen returnerer et heltal mellem 0 og 4, begge inklusive.
    let rec blackFun hidden guess =
        match (hidden, guess) with
        | (x :: xs, y :: ys) ->
                            // Hvis hidden[i] og guess[i] er samme farve,
                            // skal der returneres en mere sort pind.
                            if x = y then
                                1 + blackFun xs ys
                            else
                                0 + blackFun xs ys
        | ([], _) -> 0
        | _ -> failwith "hej"

    // blackPegs betegner antallet af sorte pinde.
    let blackPegs = blackFun hidden guess

    // Antallet af hvide pinde kan regnes som summen af sorte og hvide
       pinde
    // minus antallet af sorte pinde.
    let whitePegs = sumPegs - blackPegs

    // Holder styr paa hvor mange gaet man har brugt.
    guessCount <- guessCount + 1


    if blacksValidate = 0 then
        printfn "%d:␣␣%A" guessCount guess
    else ()

    // Returnerer en tuple med antallet af hvide- og sorte pinde.
    (whitePegs, blackPegs)

// Tjekker om brugeren har valgt det rigtige input naar han/hun skal vaelge
// game mode.
// Brugeren bliver bedt om at vaelge et af de fire game modes. Hvis
   brugeren
// indtaster noget andet, vil han/hun faa at vide han/hun skal vaelge et
   tal
```

```
// mellem 1 og 4
let playersInputCorrect number =
    match number with
    | "1" | "2" | "3" | "4" -> true
    | _ -> false

// Tjekker om brugeren vaelger et gyldigt valg paa den foerste
   velkomstskaermen.
let playersInputCorrectStartScreen input =
    match input with
    | "help" | "Help" | "start" | "Start" ->
        true
    | _ ->
        false

/// <summary>Funktionen guess tager mod et brugergaet og returnerer en
/// kode</summary>
/// <remarks>Funktionen har et parameter af typen player, saa den baade
/// kan tage gaet fra en bruger og et computer. Dette stod der i opgaven
/// at den skulle. Men gaettet fra computeren bliver lavet et andet sted
/// i koden, og derfor matcher player altid med Human og aldrig med
/// Computer. Vi satte bare Computer branchen til at returnerer en liste
/// med farven Red, da guess skal returnere en variable af typen code
/// </remarks>
/// <params>guess har to parametere. Player er af typen player, og bliver
   brugt
/// til at finde ud af om gaettet kommer fra en computer eller en bruger.
   board
/// er af typen board, og tager variable af typen board. Board er en liste
   med
/// elementer som har typen (code * answer).</params>
/// <returns>guess returnerer en kode hvis brugerinputtet er korrekt</
   returns>
let guess (player : player) (board : board) : code =
    match player with
    | Human ->
        let mutable userInput = []

        // Printer spillebraettet indeholdende tidliger gaet og dets
           svarpinde.
        printfn "\nYour␣board␣so␣far:"

        for i = 0 to board.Length-1 do
            printfn "%A" (board.[i])

        printf "\n>␣"

        userInput <- Array.toList (System.Console.ReadLine().Split(' '))

        // validerer bruger input
        while (userCode userInput) = false do
            printfn "\nInvalid␣input\n"
            printf  ">␣"
            userInput <- Array.toList (System.Console.ReadLine().Split(' ')
                )

        inputCodeToCode userInput
    | Computer ->
```

```
        [Red]

/// <summary>Funktioen gamePlay koerer indtil brugeren har gaettet rigtigt
/// </summary>
/// <remarks>Funktionen koerer kun naar det er brugeren der fungerer som
/// kodeloeser</remarks>
/// <params>Tager som parameter hiddenCode af typen code. Den bruger dette
/// parameter til at sammeligne de gaet der bliver lavet med den hemmelige
/// kode, saa den ved hvornaar den skal terminere</params>
/// <returns>Returnerer et heltal med hvor mange forsoeg brugeren har brugt
    paa
/// at gaette koden.</returns>
let gamePlay hiddenCode =
    let mutable board : board = []
    let mutable humanGuess : code = [Red]
    let mutable validation = (0,0)
    let mutable tries : int = 0

    // Det er foerst naar validate returnerer (0,4) (Naar der er 4 sorte
       pinde)
    // at loekken terminerer, og brugeren har vundet spillet.
    while snd (validation) <> 4 do
        System.Console.Clear()

        // Printer hvor mange gaet man har brugt indtil videre.
        printfn "You␣have␣used␣%d␣guesses␣so␣far." tries

        humanGuess <- guess Human board

        validation <- (validate hiddenCode humanGuess)

        // Spillebraettet bliver opdateret med det nye gaet
        board <- board @ [(humanGuess, validation)]

        tries <- tries + 1
    tries

/// <summary>game funktionen er "main" funktionen i vores program. Den
   fungerer
/// ved at den kalder sig selv naar der skal startes et nyt spil - derfor
   er den
/// rekursivt defineret.</summary>
/// <param name="choice">Choice fortaeller hvilket
/// game mode der skal spilles. choice kan antage foelgende vaerdier:
///   0: Startskaerm
///   1: Human vs. Computer, hvor computer som "codemaker"
///   2: Human vs. Human
///   3: Computer vs. Human, hvor brugeren fungerer som "codemaker"
///   4: Computer vs Computer
/// </param>
let rec game choice =

    /// <summary>Win funktionen bliver kaldt naar enten brugeren eller
       computer
    /// har gaettet rigtigt, og derfor vundet spillet.</summary>
    /// <param name="player">Parameteret player bruges til at afgoere om
       det er en bruger eller
    /// computeren som har vundet.</param>
```

```
/// <param name="gameModeChoice">
/// Parameteret gameModeChoice bruges til at finde ud af hvilket
    gamemode
/// brugeren/computer spillede da han/hun/den vandt. Dette bliver brugt
/// naar man skal spille igen, for saa kalder man game med dette
    parameter,
/// hvis man oensker at spille samme game mode igen.
/// </param>
/// <param name="guessCounter">guessCounter er hvor mange gaet
    computeren har brugt</param>
/// <param name="tries">tries er hvor mange gaet brugeren har brugt.</
    param>
/// <returns>Returneringsvaedien afhaenger af hvad brugeren taster ind
/// efter overstaaet spil</returns>
let rec win player gameModeChoice guessCounter tries =
    match player with
    | "Human" ->
        System.Console.Clear()
        printfn "You␣win!"
        printfn "It␣took␣you␣%d␣guesses!" tries

    | _ ->
        printfn "\nThe␣computer,␣Allan,␣wins!"
        printfn "It␣took␣it␣%d␣guesses!" guessCounter

    printfn "\nWould␣you␣like␣to:"
    printfn "1.␣Go␣to␣the␣startmenu?"
    printfn "2.␣Play␣the␣same␣gamemode␣again?"
    printfn "3.␣Quit␣the␣game?"
    printf ">␣"
    let userInput = System.Console.ReadLine ()

    match userInput with
    | "1" -> game "0" // Her ledes brugeren tilbage til startskaermen
    | "2" -> game gameModeChoice // Her spilles det samme game mode
       igen
    | "3" -> System.Environment.Exit 0 // Her afsluttes spillet

    // Her kalder funktionen sig selv, med de nuvaerende parametre som
    // argumenter. Den goer dette i stedet for en fejlmeddelelse, og
    // spoerger saa brugeren igen om hvad han/hun vil.
    | _ -> win player gameModeChoice guessCounter tries

/// <summary>Funktionen compGuess bliver kaldt naar det er computeren
    der
/// skal loese en kode</summary>
/// <param name="hiddenCode">Opgaven der skal loeses</param>
/// <param name="gameChoice">Hvilken gamemode der bliver kaldt fra.
    Bruges til at spille samme type
/// igen fra winskaerm</param>
/// <remarks>Funktionen koeres kun naar computeren skal gaette</remarks
    >
let compGuess hiddenCode gameChoice =
    //Variable der bruges under funktionen
    let mutable guessCode : code = []
    let mutable validation = (0,0)
    let mutable whitePegs : int = 0
    let mutable blackPegs : int = 0
```

```
//Laver en variabel med 6 elementer, der alle er 0'er.
let temp = (Array.init 6 (fun _ -> 0))

//Foerste 6 gaet
// Hvis koden bestaar af fire ens farver, kan programmet gaette det
    paa
// i+1 gaet, hvis koden har vaerdi i, hvor Red = 0, Green = 1, etc.
let mutable whiteCounter = 0
let mutable n = 0
let colors = [Red; Green; Yellow; Purple; White; Black]
//Laver gaet med samme farve paa alle pladser, en farve af gangen
while (whiteCounter < 4) && (n < 6) do
    guessCode <- [colors.[n]; colors.[n]; colors.[n]; colors.[n]]
    //Midlertidigt histogram der bruges til at sammensaette 7. gaet
       . Indsaetter tal svarende
    //til hvor mange sorte stifter der modtages som respons paa
       gaet paa plads svarende til
    //farvenummer
    temp.[n] <- snd (validate hiddenCode [colors.[n]; colors.[n];
                                          colors.[n]; colors.[n]])
    whiteCounter <- whiteCounter + temp.[n]
    n <- n + 1



//Sammensaetter 7. gaet
guessCode <- []
for i = 0 to 5 do
    while temp.[i] > 0 do
        guessCode <- List.append guessCode [colors.[i]]
        temp.[i] <- temp.[i] - 1

//Tjekker hvor mange sorte der gives som respons paa sammensatte 7.
    gaet.
//Hvis opgaven er den samme farve paa alle 4 pladser, vil
   programmet bruge et ekstra
//gaet, selvom den har det rigtige gaet. Derfor aendrer vi
   variablen blacksValidate, som goer at
//guessCount ikke taeller op naar den er forskellig fra 0.
let mutable blacks =
    guessCount <- guessCount - 1
    blacksValidate <- 1
    snd (validate hiddenCode guessCode)

//aendrer den tilbage saa gaet taeller op igen ved hver validate
blacksValidate <- 0

if blacks = 4 then
    let mutable s = 0
    //Hvis alle farverne er den samme, bruges der ikke et gaet.
    for i = 0 to 5 do
        if guessCode.[0..3] = [colors.[i]; colors.[i]; colors.[i];
           colors.[i]] then
            s <- s + 1
        else ()
    //Men hvis opgaven nu er [Red; Red; Green; Green] vil
       programmet have den efter andet
    //gaet. Den skal dog stadig bruge et gaet paa rent faktisk at
```

```
        gaette paa netop det.
    if s = 0 then
        //Bruges udelukkende for at bruge et gaet
        validate hiddenCode guessCode |> ignore
    else ()
else
    blacks <- snd (validate hiddenCode guessCode)

/// <summary>Tjekker antal sorte og omrokerer farverne i
    positionerne</summary>
/// <param name="guess">Bruges naar programmet kaldes rekursivt til
    have gemme nuvaerende
/// vaerdi af guessCode</param>
/// <returns>Kalder win funktionen hvis der gives 4 sorte stifter
    som respons, ellers
/// omrokerer den farver</returns>
let rec compRearrange guess =
    match blacks with
    | 4 ->
        win "Comp" gameChoice guessCount 0
    | 2 ->
        guessCode <- [guessCode.[3]] @ guessCode.[1..2] @ [
            guessCode.[0]]
        // Bliver til 4 2 3 1
        blacks <- snd (validate hiddenCode guessCode)

        match blacks with
        | 4 ->
            win "Comp" gameChoice guessCount 0
        | 2 ->
            if guessCode.[0] = guessCode.[2] && guessCode.[1] =
                guessCode.[3] then
                guessCode <- guessCode.[0..1] @ [guessCode.[3]] @ [
                    guessCode.[2]]
                blacks <- snd (validate hiddenCode guessCode)
                if blacks = 4 then
                    win "Comp" gameChoice guessCount 0
                else
                    guessCode <- [guessCode.[1]] @ [guessCode.[0]]
                        @ [guessCode.[3]] @ [guessCode.[2]]
                    blacks <- snd (validate hiddenCode guessCode)
                    win "Comp" gameChoice guessCount 0
            elif guessCode.[0] = guessCode.[2] then
                guessCode <- guessCode.[0..1] @ [guessCode.[3]] @ [
                    guessCode.[2]]
                blacks <- snd (validate hiddenCode guessCode)
                win "Comp" gameChoice guessCount 0
            else
                guessCode <- [guessCode.[1]] @ [guessCode.[0]] @
                    guessCode.[2..3]
                blacks <- snd (validate hiddenCode guessCode)
                win "Comp" gameChoice guessCount 0
        | 0 ->
            guessCode <- List.rev guessCode
            blacks <- snd (validate hiddenCode guessCode)
            win "Comp" gameChoice guessCount 0
        | 1 ->
            guessCode <- [guessCode.[2]] @ [guessCode.[1]] @ [
```

```
                guessCode.[3]] @ [guessCode.[0]]
            // Bliver til 3 2 1 4
            blacks <- snd (validate hiddenCode guessCode)
            match blacks with
            | 4 ->
                win "Comp" gameChoice guessCount 0
            | 2 ->
                guessCode <- [guessCode.[0]] @ [guessCode.[2]] @ [
                    guessCode.[1]] @ [guessCode.[3]]
                blacks <- snd (validate hiddenCode guessCode)
                win "Comp" gameChoice guessCount 0
            | 0 ->
                guessCode <- [guessCode.[2]] @ [guessCode.[3]] @ [
                    guessCode.[0]] @ [guessCode.[1]]
                // Bliver til 1 4 3 2
                blacks <- snd (validate hiddenCode guessCode)
                win "Comp" gameChoice guessCount 0
            | 1 ->
                guessCode <- [guessCode.[1]] @ [guessCode.[2]] @ [
                    guessCode.[0]] @ [guessCode.[3]]
                // Bliver til 2 1 3 4
                blacks <- snd (validate hiddenCode guessCode)
                match blacks with
                | 4 ->
                    win "Comp" gameChoice guessCount 0
                | 2 ->
                    guessCode <- guessCode.[0..1] @ [guessCode.[3]]
                        @ [guessCode.[2]]
                    win "Comp" gameChoice guessCount 0
                | 0 ->
                    guessCode <- [guessCode.[1]] @ [guessCode.[0]]
                        @ [guessCode.[3]] @ [guessCode.[2]]
                    // Bliver til 1 2 4 3
                    blacks <- snd (validate hiddenCode guessCode)
                    win "Comp" gameChoice guessCount 0
                | _ -> printfn "Fejl paa 2. indre"
            | _ -> printfn "Fejl paa 2. midt"
        | _ -> printfn "Fejl paa 2. ydre"
    | 1 ->
        guessCode <- [guessCode.[3]] @ guessCode.[1..2] @ [
            guessCode.[0]]
        // Bliver til 4 2 3 1
        blacks <- snd (validate hiddenCode guessCode)
        match blacks with
        | 2 ->
            if guessCode.[0] = guessCode.[2] then
                guessCode <- [guessCode.[1]] @ [guessCode.[0]] @
                    guessCode.[2..3]
                blacks <- snd (validate hiddenCode guessCode)
                if blacks = 4 then
                    win "Comp" gameChoice guessCount 0
                else
                    guessCode <- [guessCode.[0]] @ [guessCode.[3]]
                        @ [guessCode.[2]] @ [guessCode.[1]]
                    blacks <- snd (validate hiddenCode guessCode)
                    win "Comp" gameChoice guessCount 0
            else
                compRearrange guessCode
```

```
            | 1 ->
                if guessCode.[0] = guessCode.[2] then
                    guessCode <- [guessCode.[0]] @ [guessCode.[3]] @ [
                        guessCode.[1]] @ [guessCode.[2]]
                    blacks <- snd (validate hiddenCode guessCode)
                    win "Comp" gameChoice guessCount 0
                else
                    guessCode <- [guessCode.[1]] @ [guessCode.[2]] @ [
                        guessCode.[0]] @ [guessCode.[3]]
                    blacks <- snd (validate hiddenCode guessCode)
                    win "Comp" gameChoice guessCount 0
            | 0 ->
                guessCode <- [guessCode.[3]] @ [guessCode.[2]] @ [
                    guessCode.[1]] @ [guessCode.[0]]
                compRearrange guessCode
            | _ -> printfn "Fejl paa 1."

        | 0 ->
            guessCode <- guessCode.[3] :: guessCode.[0..2]
            // Bliver til 4 1 2 3
            blacks <- snd (validate hiddenCode guessCode)
            match blacks with
            | 4 ->
                win "Comp" gameChoice guessCount 0
            | 1 ->
                if guessCode.[0] = guessCode.[3] then
                    guessCode <- [guessCode.[0]] @ [guessCode.[3]] @ [
                        guessCode.[1]] @ [guessCode.[2]]
                    blacks <- snd (validate hiddenCode guessCode)
                    //bliver til 4 3 1 2
                    win "Comp" gameChoice guessCount 0
                else
                    compRearrange guessCode
            | 2 ->
                if guessCode.[0] = guessCode.[3] then
                    guessCode <- [guessCode.[0]] @ [guessCode.[3]] @ [
                        guessCode.[2]] @ [guessCode.[1]]
                    blacks <- snd (validate hiddenCode guessCode)
                    //bliver til 4 3 2 1
                    win "Comp" gameChoice guessCount 0
                else
                    compRearrange guessCode
            | 0 ->
                compRearrange guessCode
            | _ -> printfn "Fejl paa 0."
        | _ -> printfn "Fejl paa ydre"
    compRearrange guessCode

if choice = "0" then
    //Startskaerm

    System.Console.Clear()

    printfn "Hey. And welcome to the game Mastermind. If this is your"
    printfn "first time, you are strongly encourged to write \"help\", 
        and you"
    printfn "will receive a help screen explaining the goal of the 
        games"
```

```
printfn "and␣the␣controls.␣If␣you␣are␣already␣familiar,␣you␣can␣
    simply"
printfn "just␣type␣'start'␣to␣start␣the␣game.\n"
printf  ">␣"

let mutable userInputStartScreen = System.Console.ReadLine ()

//Indtil brugeren skriver enten help eller start, kommer man ikke
    videre
while (playersInputCorrectStartScreen userInputStartScreen) = false
    do
    printfn "\nPlease␣enter␣either␣\"help\"␣or␣\"start\"\n"
    printf  ">"
    userInputStartScreen <- System.Console.ReadLine ()


if userInputStartScreen = "help" || userInputStartScreen = "Help"
    then
    System.Console.Clear()
    printfn "THE␣GOAL␣OF␣MASTERMIND"
    printfn "The␣goal␣is␣to␣guess␣the␣secret␣code␣made␣by␣the␣\"
        codemaker\""
    printfn "A␣code␣consists␣of␣4␣coloured␣pins,␣withe␣the␣
        following"
    printfn "color:␣Red,␣Green,␣Yellow,␣Purple,␣White,␣and␣Black."
    printfn "After␣each␣of␣your␣guess,␣you␣get␣a␣pair␣of␣white␣and␣
        black"
    printfn "pegs,␣telling␣you␣how␣much␣off␣you␣are."
    printfn "The␣number␣of␣white␣pegs␣tells␣how␣many␣of␣the␣pegs␣in
        ␣your"
    printfn "guess␣is␣the␣right␣color,␣but␣in␣the␣wrong␣place.␣And␣
        the"
    printfn "number␣of␣black␣pegs,␣tells␣the␣how␣many␣of␣the␣pegs␣
        in"
    printfn "your␣guess␣is␣right,␣both␣in␣terms␣of␣color␣and␣
        position."
    printfn "\nFor␣example,␣if␣the␣hidden␣code␣is:␣R␣G␣B␣P"
    printfn "and␣your␣guess␣is␣R␣B␣W␣W,␣your␣answer␣in␣white␣and␣
        black"
    printfn "pegs,␣will␣be␣shown␣to␣the␣right.␣In␣this␣example␣it␣
        will"
    printfn "look␣like␣this:"
    printfn "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣W␣B"
    printfn "R␣B␣W␣W␣␣␣␣␣␣␣␣␣␣␣␣␣1␣1"
    printfn "When␣your␣guess␣has␣4␣black␣pegs,␣it␣means␣your␣are␣
        guessed"
    printfn "the␣secret␣code,␣and␣won␣the␣game."
    printfn "Now␣there␣is␣really␣only␣one␣thing␣left,␣who␣is␣to␣
        play?"

printfn "\n"
printfn "The␣different␣game␣modes:\n"
printfn "(1)␣Human␣vs␣Computer,␣with␣computer␣acting␣as␣\"codemaker
    \""
printfn "(2)␣Human␣vs␣Human"
printfn "(3)␣Computer␣vs␣Human,␣with␣human␣acting␣as␣\"codemaker\""
printfn "(4)␣Computer␣vs␣Computer"
printfn "\nChoose␣a␣number␣between␣1-4\n"
```

```
        printf  ">␣"

        let mutable playersInput = System.Console.ReadLine ()

        //Indtil man skriver tal mellem 1 og 4 kommer man ikke videre
        while (playersInputCorrect playersInput) = false do
            printfn "\nYou␣have␣to␣choose␣a␣number␣between␣1␣and␣4\n"
            printf  ">␣"

            playersInput <- System.Console.ReadLine ()
        //Kalder spilfunktion rekursivt med indtastede tal som argument
        game playersInput

    elif choice = "1" then
        //Human vs Computer

        System.Console.Clear()

        printfn "You␣chose␣the␣game␣mode␣Human␣vs␣Computer,␣where␣the␣
            computer"
        printfn "acts␣as␣the␣\"codemaker\""

        //Opgaven generes automatisk med et kald til makeCode og gemmes i
            variablen hiddenCode
        let hiddenCode = makeCode Computer

        printfn "\nThe␣computer␣has␣now␣generated␣a␣code.\n"
        printf "Press␣ENTER␣to␣begin␣playing..."
        let p = System.Console.ReadLine()

        //Kalder funktionen gamePlay der foerst terminerer naar opgaven er
            loest
        let humanNumberOfTries = gamePlay hiddenCode
        //Kalder win funktionen. Giver den player, hvilken spiltype der
            lige blev spillet og hvor
        //mange forsoeg det tog.
        win "Human" "1" 0 humanNumberOfTries

    elif choice = "2" then
        //Human vs Human

        System.Console.Clear()

        printfn "You␣chose␣the␣game␣mode␣Human␣vs␣Human"

        //Opgaven laves ved et kald til makeCode og gemmes i variablen
            hiddenCode
        let hiddenCode = makeCode Human

        printfn "\nCode␣accepted.\n"
        printf "Press␣ENTER␣to␣give␣control␣to␣player␣2..."
        let p = System.Console.ReadLine()

        //Kalder funktionen gamePlay der foerst terminerer naar opgaven er
            loest
        let humanNumberOfTries = gamePlay hiddenCode
        //Kalder win funktionen. Giver den player, hvilken spiltype der
            lige blev spillet og hvor
```

```
        //mange forsoeg det tog.
        win "Human" "2" 0 humanNumberOfTries

    elif choice = "3" then
        //Computer vs Human
        //guessCount skal nulstilles da det er en global variabel
        guessCount <- 0

        System.Console.Clear()

        printfn "Computer vs Human, with human acting as \"codemaker\""

        //Opgaven laves ved et kald til makeCode og gemmes i variablen
            hiddenCode
        let hiddenCode = makeCode Human

        printfn "\nCode accepted.\n"
        printf "Press ENTER to give control to the computer..."
        let p = System.Console.ReadLine()
        let k = System.Console.Clear()

        //Funktionen compGuess kaldes med opgaven og spiltypen. Den
            terminerer foerst naar opgaven er
        //loest, og win kaldes videre fra den.
        compGuess hiddenCode "3"

    else
        //Computer vs Computer
        //guessCount skal nulstilles da det er en global variabel
        guessCount <- 0

        System.Console.Clear()

        printfn "You chose the game mode Computer vs Computer"

        //Opgaven generes automatisk med et kald til makeCode og gemmes i
            variablen hiddenCode
        let hiddenCode = makeCode Computer

        printfn "\nThe computer has now generated a code.\n"
        printfn "It is: %A\n" hiddenCode
        printf "Press ENTER to begin the game..."
        let p = System.Console.ReadLine()
        let k = System.Console.Clear()

        //Funktionen compGuess kaldes med opgaven og spiltypen. Den
            terminerer foerst naar opgaven er
        //loest, og win kaldes videre fra den.
        compGuess hiddenCode "4"

game "0"
```