The Mastermind

PoP

Beate Berendt Søegaard
Mathias Larsen
Simon Rotendahl

Datalogi

16. december 2016

Indhold

1 Forord

I denne opgave skal vi lave en version af det populære spil Mastermind også kendt som Codebreaker. Vi skal lave spillet så man kan spille person mod person, person mod computer, og samt computer mod computer. Der er ikke nogen videre krav til den kunstige intelligens computeren skal anvende.

2 Introduktion

Spillet Mastermind er fra 1970'erne og kan spilles af to spillere. Premisen af spillet er, at den ene af spillerne laver en 'hemmelig' kombination af fire farver (også kaldet pegs), som den anden spiller så skal gætte. Hvis en spiller gætter på en korrekt farve, men på den forkerte plads får vedkommende en hvid peg, hvis farven og placeringen er korrekt, får vedkommende en sort peg.

3 Problemformulering

Vi skal implementerer spillet Mastermind i F#. Vores program skal kunne understøtte datamat vs. datamat, menneske vs. datamat og menneske vs. menneske, hvor de valgte farver bliver vist på skærmen og hvorvidt om farven er korrekt og korrekt placeret, hvilket vil resultere i hhv. en hvid eller sort peg.
Når spillet begynder vil brugeren blive prompet til at vælge en række forskellige farver, og hvordan man skal vælge dem, der skulle ikke hærske tvivl om, hvordan man starter spillet.

Vi skal implementere følgende tre funktioner:

- makeCode : player → code
- guess : player → board → code
- validate : code → code → asnwer

Vi skal også anvende følgende typer:

- 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

4 Problemanalyse og design

Der var primært to ting som, der kunne skabe problemer med implementationen.

4.1 Analyse af problemerne

4.1.1 Typer og input

For at programmet skal kunne køre fejlfrit kræver det, at vore variable er af den samme type, da Fsharp er et typestærkt sprog. F.eks. vi kan ikke bare have en funktion som tager en liste af tal som input, men for givet et array af strenge. Inputtet skal være ens med, hvad funktionen bruger.

4.1.2 Kunstig intelligens

Den anden problem/udfordring er den kunstige intelligens computeren skal bruge for at kunne løse opgaven. Vi tænkte meget over hvordan denne intelligens skulle være. Vi ønsker nemlig ikke "brute force" metoden hvor computeren bare gætter indtil den rammer det rigtige, men vi ønsker heller ikke en "picture perfect" intelligens, da det vil kede brugeren hvis computeren altid kom med det rigtige svar efter f.eks 5 runder. Så vi ønsker en kunstig intelligens som anvender oplysningen den får igennem de sorte og hvide pegs, men som stadig har en lille smule tilfældighed over sig, så den ikke bliver forudsigelig og kedelig.

4.2 Design af løsninger til problemanalysen

4.2.1 Input

Som løsning til vores problem af input, valgte vi at håndtere farverne som tal. Dvs. vi tildelte hver farven et tal, såsom "White- 0, på den måde undgik vi, at skulle forholde os til strenge af forskellige længder, hvilket vi fandt besværligt at implementere.

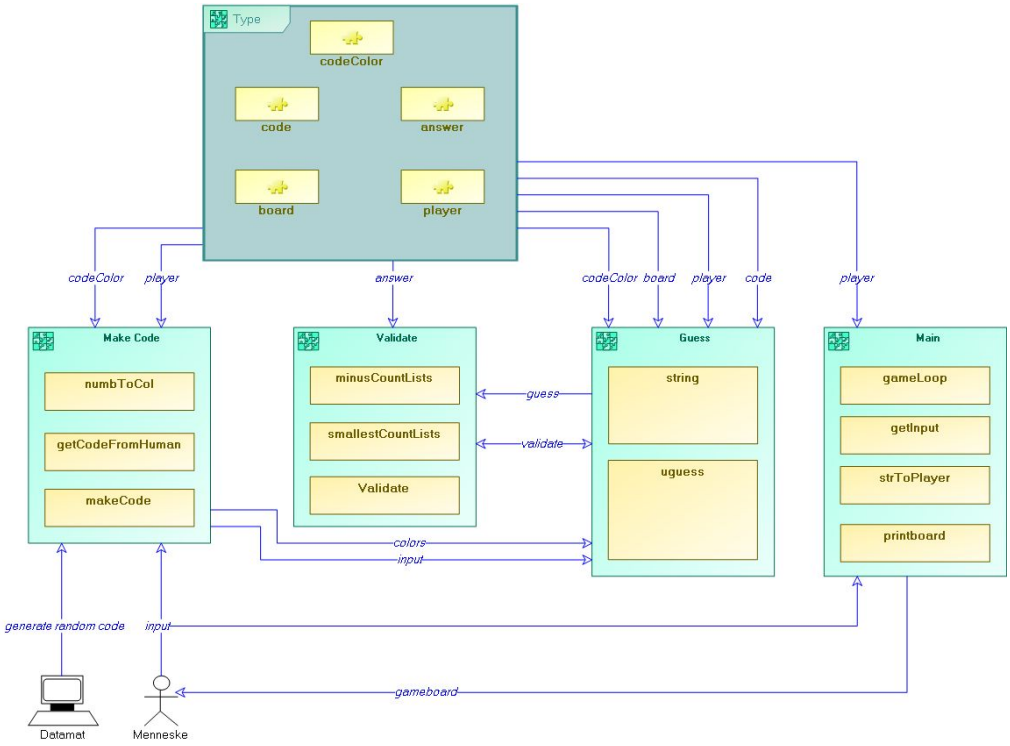
4.2.2 Kunstig intelligens

For at implementere en kunstig intelligens, lavede vi en funktion som gætter på en kombination af farver, hvor funktionen vil få enten hvide, sorte eller ingen pegs tilbage. Den information vil funktionen bruge til, at komme op med et kvalificeret bud på det næste gæt. Metode, hvorved funktionen gøre det er, at den fjerne alle de muligheder de store og hvide pegs end må udelukke. Med andre ord, bliver mulighederne for et kvalificeret bud på et gæt mindre og mindre og vil så foresætte indtil den rigtige farve kombination er gættet .

Man kunne optimere vores kunstig intelligens yderligere ved, at udregne hvor mange muligheder et gæt ville fjerne alt efter hvilken kombination af sorte og hvide pegs den får tilbage. For så at vælge den mulighed som vil have den største sandsynlighed for, at fjerne flest mulige gæt. Her sætter loven for "diminising returns" ind, den her optimering vil gøre antallet af gæt mindre, men til gengæld vil tiden det tager for datamaten at udregne sit næste gæt stige. Da vores mål ikke var at gætte svaret i så få træk som muligt var der ingen grund til, at gøre programmet langsommere.

4.3 Overordnet Design

Her ses et diagram af vores overordnet design. I diagrammet ses det hvordan programmets funktioner sender og modtager information til og fra hinanden. *For en mere detaljeret gennemgang af programmets flow se afsnit 5.1*



Figur 1: Diagram over det overordende design

5 Programbeskrivelse

5.1 Beskrivelse af programmets flow

Spillet starter med at spørger om hver af spillerne er et menneske eller en computer. Bagefter får den så spiller 1 til, at indtaste den hemmelige kode.

Efter spillerne er valgt, og den hemmelige kode er blevet lavet køre gameLoop Funktionen gameLoop holder spillet kørende. Det er den der står for at køre de andre funktioner i den rigtige rækkefølge. Denne funktion tager spilleren som skal gætte (player), den hemmelige kode (secretCode), og et board (gameBoard), af alle tidligere gæt og svar, som parametre.

Først køre den printBoard, og giver den gameBoard. Første gang vil gameBoard selvfølgelig være tomt, så der vil kun blive printet en streg. Vi syntes at stregen så meget godt ud så i stedet for, at undgå at køre den første runde, valgte vi, ikke at ændre det.

Bagefter bliver guess (se afsnit 5.2.2) så kørt med player, og gameBoard, som parametre. Hvis player er Human vil guess køre getCodeFromHuman (se afsnit 5.2.1). getCodeFromHuman tager en besked, som den så skriver til brugeren. Den tager så et input fra brugeren, som den laver om til typen "code"og returnere. guess returnere så den til gameLoop.
Hvis spilleren som skal gætter er Computer vælger funktionen en kode og returnere den.

Så bliver validate (se afsnit 5.2.3) kørt. Validate får koden som guess returnerede, og secretCode. Den sammenligner de to koder og returnere så et "answer"af sorte og hvide pløkker.

Derefter bliver der checket om der var 4 sorte pløkker, og der dermed blev gættet rigtigt. Hvis der gjorde slutter spillet. Hvis ikke det var det rigtige gæt køre gameLoop sig selv. Denne gang får den en opdateret version af gameBoard, som også indeholder det seneste gæt.

5.2 Funktioner opgivet i problemformuleringen

Opgaven har stillet tre funktioner som skal være i opgave. Funktionerne er som følgende:

- makeCode : player → code
- guess : player → board → code
- validate : code → code → asnwer

5.2.1 makeCode

$$\text{makeCode} : \text{player} \rightarrow \text{code}$$

MakeCode funktionen generer en kombination af farver som datamaten vil stille som udfordring. Funktionen håndtere også farve kombinationen som den får fra brugeren, som den vil oversætte til en type som resten af programmet kan bruge. Vi har delt makeCode op i to, makeCode og dens hjælpefunktion getCodeFromHuman. Grunden til at vi har gjort det er at den også bruges i guess, og er derved en hjælpefunktion i begge.

makeCode Selve makeCode funktionen er utrolig simple. Den starter med at checke om kaldet er til et menneske eller til en computer. Hvis det er et menneske, så kalder den hjælpefunktionen getCodeFromHuman. Hvis det er en computer, så laver den en liste med 4 code typer i med tilfældige farver. Den anvender System.Random til at finde et tilfældigt tal.

getCodeFromHuman En udfordring ved denne funktion har været oversættelsen fra terminalen til noget brugbart i resten af koden. Inputtet fra terminalen vil være en streng type, og de kan være svære at parse til det man skal bruge, især når brugeren kan indtaste ekstra mellemrum osv. Vores løsning til dette

problem er at give brugeren en specifik form at indtaste deres farve kombination på. I stedet for at skrive farver som 'Rød', 'Grøn', 'Hvid' osv. Så indtaster brugeren tal fra 0 til 5, som så vil repræsentere deres ønskede farve. Grunden til at vi gjorde det, var at det er nemmere at parse en enkelt karakter i en streng, f.eks 1, end at skulle differentiere mellem forskellige længder af forskellige ord, f.eks 'Rød' og 'Hvid'. Vi lavede så en hjælpefunktion til makeCode, som hedder NumbToCol, som tager et tal og matcher den med en farve. I sidste ende returnere koden en liste af 4 farver.

5.2.2 guess

$$\text{guess: player} \rightarrow \text{board} \rightarrow \text{code}$$

Funktionen guess er den funktion der registrere brugerens gæt, eller intelligent generer computerens gæt hvis det er den der skal gætte.
Guess virker ved at: Hvis spiller 2 er Human vil guess bare køre getCodeFromHuman. Hvis spiller 2 der imod er Computer, og det er det første gæt vil det returnere koden [Red;Red;Green;Green]. Hvis ikke er det første gæt vil funktionen løbe igennem det board som den har fået, og fjerne alle de koder som ikke kan lade sig gøre, ud fra de answers den har fået for sine tidlige gæt, fra en liste af alle mulige koder der er. Når den har fjernet alle de koder der ikke kan være svaret, gætter den på den første kode som er tilbage i listen.

5.2.3 validate

$$\text{validate} : \text{code} \rightarrow \text{code} \rightarrow \text{asnwer}$$

Funktionen validate skal tage outputtet af guess og sammenligne det med outputtet af makeCode, og se om de matcher, hvis de er ens er spillet slut. Hvis de ikke er ens skal den se om gætteren skal have nogle sorte eller hvide pinde tilbage, og hvis de skal hvor mange af hver.
Validate benytter sig af List.countBy til at tælle hvor mange af de forskellige farver der forekommer i gættet og den hemmelige kode, så for at arbejde med de lister som den funktion returnere bruger vi to hjælpe funktioner. Begge hjælpefunktioner tager to lister på formen (T' * int) list

5.2.4 minusCountLists

Alle de elementer, T', bliver heltallet, som hænger sammen med den, i den anden liste trukket fra heltallet i den første liste. Hvis elementet kun forekommer i den første liste bliver der ikke trukket noget fra tallet, og hvis det kun forekommer i den anden liste ser vi bort fra det.

5.2.5 reduceCountLists

Denne funktion tager det mindste tal til hvert element, eller fjerner det hvis der er mindre end 1 af det element, i en af listerne. Så hvis et element ikke forekommer i en af listerne vil det bliver fjernet, og af dem som er tilbage tager vi det mindste af de to tal som høre til det element. Validate laver først en ny liste, blacks, som kun indholder de farver som står rigtigt (sorte svarstifter). Så tæller vi hvor mange af de forskellige farver der er i blacks med List.countBy,

Cblacks. Så bruger vi minusCountLists på Cblacks og en List.countBy liste af gættet, så får vi hvor mange af de forskellige farver der er i gættet uden dem som også stod på den rigtige plads, Cguess. Vi gør de samme med svaret, og får Cans. Nu kan vi så bruge reduceCountLists til at få listen af de farver, og antallet af dem, som er i både svaret og gættet. Så lægger vi antallene af de forskellige farver sammen, og så har vi antallet af de hvide svarstifter. Antallet af de sorte svarstifter er bare længden af listen blacks.

6 Test — Afprøvning

Når man køre programmet vil man blive spøgte om man vil se testene eller spille spillet. Vi har testet de dele af funktionerne der ikke skal bruge et bruger input.

6.1 validate

Vi tester validate med forskellige set koder som giver forskellige answers.
Vi har valgt ikke, at teste validate med koder af andre længder end 4 da det er op til funktionerne der tager imod input fra brugeren, at sikre sig de ikke tage imod ugyldige koder.

6.2 makeCode

MakeCode tester vi om der bliver returneret endegyldig kode når player er Computer.
Vi har også testet om det brugeren skriver er det svar som funktionen får, det det har vi gjort ved, at starte spillet, vælge spiller 1, og 2, som Human, og indtaste en kode. Derefter gætter vi på den samme kode. Vi kan se funktionen virker, da vi vinder spillet.

6.3 guess

Her tester vi igen om en gyldig kode bliver returneret når player er Computer.
Vi har tested bruger input på samme måde som makeCode.

6.4 minusCountList

Vi har testede funktionen med 2 lister hvor de begge indholder de samme 3 elementer, så vi kan se de bliver trukket fra hinanden korrekt.
Derefter tester vi funktionen hvor både a, b, eller begge to er tomme, og ser vi får de resultater vi forventer. I disse tests bruger vi også forskellige typer så vi kan se det også virker som forventet.

6.5 reduceCountLists

Først tester vi med de samme lister som MinusCountList, og ser, at vi får en liste som kun indholder de elementer som der var et positivt antal af i begge lister. Af dem får vi også det mindste af de to tal. Bagefter tester vi om den returnere en tom liste hvis a eller b er tomme.

6.6 getcodeFromHuman

Vi har testede denne funktion ved at køre den, og printe dens return værdi. På den måde kunne vi se, at funktionen gav det svar vi forventet, når vi gav den et input på den form den forventet. Vi testede også funktionen med ulovlige inputs. Her fungeret den som forventet, og spurgte brugeren om, at give et lovligt input.

6.7 printBoard

Her testede vi funktionen ved at give den forskellige boards, og se hvad der blev der blev printede til konsollen.

7 Brugervejledning

Hvordan spiller man Mastermind med vores program.

7.1 Intro:

Ved programmets start vil du bliver spurgt om hvor vidt spiller 1 og spiller 2 skal være en personer eller kunstig intelligens.

7.2 Menneske vs. Menneske:

Efter du nu har valgt f.eks menneske vs. menneske vil spilleren der skal give udfordringen blive bedt om at gøre det vha. tal, med forklarende tekst der viser hvad du eller din modstander skal taste for specifikke farver, pas dog på, for udfordringen vil blive vist på skærmen mens den indtastes, så der kan snydes. Efter dette vil spilleren der skal løse udfordringen blive bedt om at gætte på en farve kombination, og computeren vil hjælpe til ved at udregne hvor mange sorte og hvide pegs han får for det gættet, og der skal gættes på nu. Således forsættes spillet indtil koden knækkets.

7.3 Menneske vs. Datamat

Efter du har valgt menneske vs. computer vil du blive bedt om at indtaste din udfordring til computeren. For at minimere dårlige oplevelser med spillet, så regner spilet selv ud hvor mange hvide og sorte pegs den får tilbage for hver svar, derved vil spillet ikke kunne lide under unødige tastefejl. Computeren vil nu gætte løs indtil koden er knækket og til sidst fortælle hvor svær den var ved at fortælle dig hvor mange gæt den brugte på at regne det ud.

7.4 Datamat vs. Datamat

Efter du har sat spillet til Computer vs. Computer, så sæt dig tilbage og nyd den hæsblæsende action, mens computeren gætter koden du ikke selv har indtastet!

8 Konklusion

Vores Mastermind spil virker. Det er muligt at spille person mod person, person mod computer, og samt computer mod computer. Vi har også fået lavet en effektiv, men stadig sjov kunstig intelligens, der gør at brugeren vil mærke at jo svære hans opgave er jo flere ture vil computeren bruge. Vi har også lavet de funktioner problemformuleringen sat som krav, samt brugt de typer beskrevet i problemformuleringen.

9 Bilag

```
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

//Validate
//
///<summary>
///Tager to lister af formen (T' * int) list, dem som List.CountBy
///retunere, og finder de elementer i b som ogsaa er i a, og
///traekker b tallet fra a tallet.
///</summary>
let minusCountLists a b =
    List.fold (fun acc (elem, count) ->
                try
                    (elem,count - (snd (List.find (fun (elem2, _) ->
                                                    elem2 = elem) b)))::acc
                with
                    _ -> (elem,count)::acc
                ) [] a
///<summary>
///Tager 2 lister paa formen, (T' * int) list, som dem retuneret af
///List.CountBy. Retunere en ny liste paa samme form, som indholder de
///elementer i a som ogsaa findes i b, og tallet i tuplen er stoerrer end
///0 i begge liste. Hvis elementet findes i begge lister, og tallene er
///stoerrer end 0, vil tallet i den nye liste vaere det mindste af de to tal.
///</summary>
let smallestCountLists a b =
    List.fold (fun acc (elem, count) ->
                try
                    let Cb = snd (List.find (fun (elem2, _) ->
                                elem2 = elem) b)
                    if Cb < 1 || count < 1 then acc
                    else
                        if Cb < count then (elem, Cb)::acc
                        else (elem, count)::acc
                with
```

```
                    _ -> acc
                ) [] a

///<summary>
///Sammenligner et gaet og svaret, og giver antallet af sorte og hvide pinde
///</summary>
///<params name="answer">
///Det rigtige svar, af typen code som er 4 lang
///</params>
///<params name="guess">
///Gaettet, af typen code som er 4 lang
///</params>
///<returns>
///Et answer
///</returns>
let validate answer guess =
//bestem sort
    let blacks = List.fold2 (fun acc elem1 elem2 ->
                             if elem1 = elem2 then elem1::acc
                             else acc) [] answer guess
//find hvid
    let Cblacks = (List.countBy (fun elem -> elem) blacks)
    let Cguess = minusCountLists (List.countBy (fun elem -> elem) guess)
                                 Cblacks
    let Cans = minusCountLists (List.countBy (fun elem -> elem) answer)
                               Cblacks

    let whites = smallestCountLists Cans Cguess
    (whites.Length, blacks.Length)
////////////////////////////////////////////////////////
//MakeCode
//
///<summary>
///Funktionen numbToCol, tager et tal og matcher det med en farve.
///</summary>
///<param name="chalNumb">Er tallet der skal laves til en farve.</param>
let numbToCol chalNumb =
    match chalNumb with
    | 0 -> Red
    | 1 -> Green
    | 2 -> Yellow
    | 3 -> Purple
    | 4 -> White
    | 5 -> Black
    | 48 -> Red
    | 49 -> Green
    | 50 -> Yellow
    | 51 -> Purple
    | 52 -> White
    | 53 -> Black
```

```
    | _ -> failwith "You gone done fucked up"
///<remarks>
///Vi har 48 til 53, da det er unicode for 0 til 5, hvilket er noedvendigt
///naar brugeren indtaster deres opgave i terminalen.
///</remarks>
///<returns>En Color type, eller kaster en undtagelse.</returns>

///<summary>Haandtere inputtet fra spilleren naar de skal gaette</summary>
let rec getCodeFromHuman msg =
    printfn "%s: \n (x,y,z,c) where x,y,z,c are your colors. Write
    0 for Red
    1 for Green
    2 for Yellow
    3 for Purple
    4 for White
    5 for Black" msg
    let input = System.Console.ReadLine()
    try
        List.init 4 (fun i -> numbToCol(int(input.[2*i+1])))
    with
        | _ -> getCodeFromHuman msg

///<remarks>
///u defineres som System.Random() for at System.Random() kan
///bruges flere gange i streg uden at give det samme tal.
///</remarks>
let u = System.Random()

///<summary>
///Funktionen makeCode lave en opgave/challenge
///til spilleren eller computeren.
///</summary>
///<param name="player">
///parametren er hvor vidt spilleren er et
///menneske eller en datamat, vha. typen player.
///</param>
let makeCode player =
    if player = Computer then
        [numbToCol(u.Next(0, 4));
        numbToCol(u.Next(0, 4));
        numbToCol(u.Next(0, 4));
        numbToCol(u.Next(0, 4))]
    else
        getCodeFromHuman "Give your challenge as the following example"
///<returns>En challenge/opgave af typen 'code'.</returns>


////////////////////////////////////////////////////////
//Guess
//
```

```
///<summary>
///Laver en codeColor om til en streng
///</summary>
let string (color:codeColor) : string =
    match color with
    | Red -> "Red"
    | Green -> "Green"
    | Yellow -> "Yellow"
    | Purple -> "Purple"
    | White -> "White"
    | Black -> "Black"
///<summary>Printer et board til consolen</summary>
let printBoard (gameBoard: board) =
    for i in gameBoard do
        let (guess, result) = i
        printfn "%-8s %-8s %-8s %-8s Whites: %d Blacks: %d"
                                (string(guess.[0]))
                                (string(guess.[1]))
                                (string(guess.[2]))
                                (string(guess.[3]))
                                (fst result) (snd result)
    printfn "-------------------------------------------------------------"
///<summary>
///Hvis player er Human faar getCodeFromHuman deres gaet, ellers
///finder computeren ud af det koder der er mulige ud fra dens andre svar,
///og retunere et af dem. Hvis det er foerste tur retunere computeren
///koden [Red;Red;Green;Green]
///</summary>
///<params name="player">En spiller enten computer eller mennekse</param>
///<params name="gameBoard">
///De gaet og svar der har vaeret indtil videre i spillet
///<params>
///<returns>En kode</returns>
let uguess (S:codeColor list list) player (gameBoard:board) =
    let removeWrong combs (guess:code) (answer:answer) =
        List.filter (fun comb -> (validate comb guess) = answer) combs
    match player with
    | Human -> getCodeFromHuman "Please enter your guess"
    | Computer when gameBoard.Length > 0 ->
        (List.fold (fun acc elem ->
                    removeWrong acc (fst elem) (snd elem)
        ) S gameBoard).[0]
    | _ -> [Red;Red;Green;Green]

//En liste af alle mulige koder
let mutable S = []
for p = 5 downto 0 do
    for j = 5 downto 0 do
        for k = 5 downto 0 do
            for l = 5 downto 0 do
```

```
                S <-  [numbToCol p;
                       numbToCol j;
                       numbToCol k;
                       numbToCol l] :: S
///<remark>
///Opretter guess som opgaven beskriver den skal
///vaere, da den lige nu forventer
///en liste af alle mulige kode
///kombinationere for, at virke
///</remark>
let guess = uguess S

////////////////////////////////////////////////////////////
//
///<summary>
///Faar et input fra brugeren, tjekker om det er lovligt, hvis ikke spoerger
///den igen
///</summary>
let rec getInput msg validInputs =
    printfn "%s" msg
    let input = System.Console.ReadLine()
    if (List.exists (fun elem -> elem = input) validInputs) then
        input
    else
        getInput msg validInputs

///<summary>
///Aendre stregene til typen player
///</summay>
let strToPlayer str =
    match str with
    | "H" | "h" -> Human
    | "C" | "c" -> Computer
    | _ -> failwith "Invalid player type"

///<summary>
///Loopen af spillet, stopper naar spillet er vundet.
///</summary>
///<parmas name="players">
///Spilleren som skal gaette det rigtige svar
///</params>
///<parmas name="secretCode">
///Koden som skal gaettes
///</params>
///<parmas name="gameBoard">
///Alle tidligere gaet og svar
///</params>
///<returns>
///Har ikke nogen return, printer en besked hvis spillet er vundet
///</returns>
```

```
let rec gameLoop player secretCode gameBoard =
    printBoard gameBoard
    let currentGuess = guess player gameBoard
    let answerPins = validate secretCode currentGuess
    match answerPins with
    | (0, 4) -> printfn "The code was guessed in %d turns" (gameBoard.Length)
    | _ -> gameLoop player secretCode ((currentGuess, answerPins)::gameBoard)

////////////////////////////////////////////////////////
//Tests
let runTest() =
//minusCountListsT
    printfn "minusCountLists tests:"
    printfn "[(\"1\", 20), (\"2\", 30); (\"4\", 0)], [(\"1\", 7);\
(\"2\", 10); (\"4\", 5)] =\n[(\"4\", -5); (\"2\", 20); (\"1\", 13)]: %A\n"
        ((minusCountLists [("1", 20); ("2", 30); ("4", 0)]
            [("1", 7); ("2", 10);("4", 5)]))

    printfn "[],[(1, 7); (2, 10); (4, 5)] = []: %A"
        ((minusCountLists [] [("1", 7); ("2", 10);("4", 5)]))

    printfn "[], [] = []: %A"  (minusCountLists [] [])
    printfn "[(Red, 7); (Green, 10); (White, 5)], [] =
[(Red, 7); (Green, 10); (White, 5)]: %A\n"
        (minusCountLists [(Red, 7); (Green, 10); (White, 5)] [])
//smallestCountLists
    printfn "smallestCountLists tests:"
    printfn "[(\"1\", 20), (\"2\", 30); (\"4\", 0)], [(\"1\", 7); \
(\"2\", 10); (\"4\", 5)] =\n[(\"2\", 10); (\"1\", 7)]: %A\n"
        ((smallestCountLists [("1", 20); ("2", 30); ("4", 0)]
            [("1", 7); ("2", 10);("4", 5)]))

    printfn "[],[(1, 7); (2, 10); (4, 5)] = []: %A"
        ((smallestCountLists [] [("1", 7); ("2", 10);("4", 5)]))

    printfn "[], [] = []: %A"  (smallestCountLists [] [])
    printfn "[(Red, 7); (Green, 10); (White, 5)], [] = []: %A\n"
        (smallestCountLists [(Red, 7); (Green, 10); (White, 5)] [])

//ValidateT
    printfn "Validate tests:"
    printfn "[Red; Green; Black; White]\
, [Red; Green; Black; White] = (0, 4): %A"
            (validate [Red; Green; Black; White] [Red; Green; Black; White])
    printfn "[Red; Black; Black; White]\
, [Red; Green; Black; White] = (0, 3): %A"
            (validate [Red; Black; Black; White] [Red; Green; Black; White])
    printfn "[Yellow; Black; Black; Black]\
, [Red; Green; Black; Yellow] = (1, 1): %A"
            (validate [Yellow; Black; Black; Black]
```

```
                [Red; Green; Black; Yellow])
    printfn "[Yellow; Green; White; Black]\
, [Black; White; Green; Yellow] = (4,0): %A"
                (validate [Yellow; Green; White; Black]
                [Black; White; Green; Yellow])

//makeCodeT
    printfn "MakeCode tests:"
    printfn "returns a list of 4 codeColours if player is computer: %A"
                (((makeCode Computer):code).Length = 4)

//GuessT
    printfn "Guess tests:"
    printfn "returns a list of 4 codeColors if player is computer: %A"
                (((guess Computer []):code).Length = 4)

////////////////////////////////////////////////////////////
printfn "Welcome to Mastermind."
match getInput "Enter 't' to see tests, or 'g' to play the game"
                ["t";"g"] with
| "t" -> runTest()
| "g" ->  let players =
            (strToPlayer (getInput
                "Is the first player a Human or Computer?(Enter H or C):"
                ["H"; "C"; "h"; "c"]),
                strToPlayer (getInput
                        "And the second player (Enter H or C):"
                        ["H"; "C"; "h"; "c"])
            )
          let secretCode = makeCode (fst players)
          gameLoop (snd players) secretCode []
| _ -> failwith "Got wrong input, getInput is broken."
```