Aflevering 7

David Rasmussen Lolck, Gustav Hanehøj, Jann Johansen

20. november 2018

Indhold

Introduktion

I denne opgave implementeres spillet Awari som beskrevet i den udleverede arbejdsseddel. For at besvare opgaven tages udgangspunkt i signaturfilen og implementationsfilen fra Absalon, og så vidt muligt benyttet funktionsparadigmet. I denne rapport vil de følgende afsnit gennemgå den benyttede fremgangsmåde ift. design og implementation af selve spillet og hvilke problemer der måtte løses på vejen. Derudover gennemgås også den tilhørende white-box test.

Biblioteket

2.1 Struktur

I den udleverede implementationsfil var funktionerne `turn` og `play` allerede defineret. Altså manglede der implementering af funktionerne `printBoard`, `isHome`, `isGameOver`, `getMove` og `distribute` fra den udleverede signaturfil. Ud over disse funktioner blev der yderligere defineret `replaceAtIndex` og `updateLastPit` som hjælpefunktioner til `distribute` funktionen. Til slut blev der også lavet en simpel AI som kan spilles imod, ved at implementere funktionen `getAiMove`, og derudover `startGame` og `startAiGame` for at gøre det lettere at køre spillet fra applikationsfilen.

2.2 Kompilering og kørsel

Biblioteket kan kompileres ved hjælp af fsharp kompileren, ved at køre kommandoen `fsharpc --nologo -a awariLib.fs`. Herefter kan spillet spilles ved at køre `fsharpc awariApp.fsx && mono awariApp.exe`. Biblioteket kan testes ved at køre white-box testen med kommandoen `fsharpc awariLibTest.fsx && mono awariLibTest.exe`. Læg mærke til, at det ikke er nødvendigt at importere biblioteket gennem kommandolinjen, da dette gøres i applikationerne.

2.3 Typer

Til biblioteket skulle der vælges typer til de mange pits og boardet. Oprindeligt var det hensigten at hvert pit blot skulle være en typeforkortelse for ints, således at værdien repræsenterede antallet af bønner. Den udleverede signaturfil viste dog, at getMove kun skulle returnere et pit, hvilket ville skabe det problem, at der sagtens kunne være flere pits med lige mange bønner, og returværdien ville således ikke kunne bruges til at vide præcis hvilket pit der blev valgt. I stedet blev hvert pit til en record af typen {`index=int, beanCount=int`}.

Boardet består nu udelukkende af 14 pits, og vi valgte at organisere disse i en liste, således at typen `board` blot er en forkortelse for `pit list`

Udvalgte Implementationer

3.1 `distribute()`

Ifølge udleverede implementationsfil skulle `distribute` tage et board, en player og et pit og returnere en tuple indeholdende det nye board, spilleren hvis pit den sidste bønne landede i, og selve pittet den sidste bønne landede i. Her afviges der dog i implementeringen af funktionen, ved ikke at returnerer spilleren, der ejer det sidste pit, hvilket kommes yderligere ind på i diskussionsafsnittet.

Funktionen `distribute` virker ved at tage bønnerne fra det givne pit og dele dem ud i de efterfølgende pits mod uret. Dette gøres ved at en gang for hvert pit at beregne hvor mange bønner den vil have efter distribute, ud fra dens placering i forhold til den oprindelige pit, samt hvor mange bønner der var i denne. For at gøre dette er implementeret yderligere to hjælpefunktioner, `replaceAtIndex` og `updateLastPit`.

`replaceAtIndex` bruges til at lave små ændringer i et givent bræt. Grundet det funktionelle programmeringsparadigme, bør der ikke laves in-place ændringer i datastrukturer, men man skaber i stedet en ny datastruktur. Dette gør denne funktion, hvor det relevante pit på spillebrættet er erstattet med den ønskede pit.

`updateLastPit` tjekker om hvorvidt feltet den sidste bønne landede i kun indeholder en enkelt bønne. I så fald ville dette betyde at feltet var tomt da `distribute` kørte, og derfor skulle bønnerne i modsatte felt og feltet selv fanges og puttes over i spillerens hjemmefelt.

3.2 `getMove()`

For at implementere `getMove` blev der taget udgangspunkt i den udleverede implementationsfil, derfor tages der som argument et board, en spiller og en streng, og der returneres det valgte pit. Til at starte med printes strengen til konsollen som angiver hvilken spillers tur det er, og "Again? "hvis det forrige træk resulterede i at spilleren fik en tur til.

For at få input fra spillerene kaldes `System.Console.ReadLine ()` som bindes til `inputString`. For at tjekke om den givne streng er valid kontrolleres denne med `List.contains` og en liste som indeholder tallene 1-6. Dette gøres for at sikre at `inputString` kan castes til en int, som derefter kan bruges til at vælge et pit fra det nuværende board. Gives et forbudt input bedes spilleren give et valid input og `getMove` kalder sig selv igen. For at følge det funktionelle paradigme er `getMove` defineret som en rekursiv funktion da dette tillader implementationen at afholde sig fra brugen af mutable variable. Der afviges fra den givne signaturfil idet at `getMove` tager et fjerde argument, i form af en int, som bruges til at differentiere mellem når der spilles mod en AI eller en anden spiller.

3.3 `getAiMove()`

Som en ekstra detalje ønskedes det at tilføje en computer-modstander, som man kunne spille imod, hvis man spillede alene. Dette kunne gøres på mange kompleksitetsniveauer, men en temmelig simpel greedy implementering blev valgt;

Implementationen fungerer ved at lade getMove tjekke om det er en AI der skal tage et træk, og i det tilfælde kalde getAiMove. Denne funktion tjekker så udfaldet af at vælge hver af de 6 pits og sammenligner hvor den kan få flest bønner i sit hjemmefelt. Kan den få en ekstra tur, tæller dette som et træk af værdi 5, da dette er et løst estimat for det gennemsnitlige antal bønner man kan få på 2 træk.

Da det funktionelle paradigme skal bruges, er funktionen rekursiv således at den kører 6 gange (1 gang pr. pit).

3.3.1 `displayMenu()`

Da der nu er to gamemodes (PvP og PvAI), er der nu brug for en menu til at vælge hvilken man ønsker at spille. Til dette blev der lavet en applikationsfil, hvorfra man kan starte spillet ved hjælp af en funktion kaldet displayMenu. Denne printer en simpel menu og beder så brugeren om at specificere gamemoden, som så passeres ned til biblioteksfunktionerne.

White-box testningen

For at afprøve implementationen, blev der fremstillet en white-box test til programmet. Denne white-box afprøver funktionerne:

- `isHome()`
- `isGameOver()`
- `replaceAtIndex()`
- `getHome()`
- `updateLastPit()`
- `distribute()`

Til at udfører denne white-box test, blev der dannet en funktion `createBoard()`, der kan hjælpe med at danne spillebræt, som funktionerne kan tage som input. Input og afput af koden vil ikke blive vist her, men kan ses i selve koden. Dette skyldes at funktionerne tager mange forskellige indput, og at disse kan indeholde meget data hver især, hvilket ville gøre dette afsnit uoverskueligt. En tabel over `isHome()` og `isGameOver()` kan ses i tabel 1 på den følgende side. Målet med denne Whitebox test har været at køre samtlige linjer kode i funktionerne, samt at få samtlige linjer kode kørt, samt at få samtlige mulige typer output til de forskellige funktioner. Da alle test giver true, gennemfører programmet succesfuldt whitebox-testen.

Funktion	ID	Betingelse	Kommentar
`isHome`	1	$i.index < 7$	
	1a	t	
	1b	f	
	2	$pitPlayer = p\&\&$ $i.index\%7 = 6$	
	2a	$t\&\&t$	
	2b	$t\&\&f$	
	2c	$f\&\&?$	Kun første del køres
`isGameOver`	1	$p1Beans = 0\|\|$ $p2beans = 0$	
	1a	$t\|\|?$	Kun første del køres
	1b	$f\|\|t$	
	1c	$f\|\|f$	

Tabel 1: White Box test over `isHome` og `isGameOver`

Diskussion

I forbindelsen med udvikling af Awari, har det også skulle overvejes hvordan spillebrættet repræsenteres. De to primære muligheder var et array eller en liste. Grundet at programmet så vidt muligt skal overholde det funktionelle paradigme, og at en liste som standard overholder dette, blev det valgt som en liste. Dette gav dog nogle problemer med at lave små ændringer inden midt i listen, fx. erstatte en række specifikke indexer med andre værdier. Dette blev løst ved at lave funktionen replaceAtIndex.

Herudover er den implementerede AI langt fra perfekt. AI'en prøver ved hvert træk at udføre det træk, der giver det bedste antal bønner lige nu, men den kigger på ingen måde fremad længere end 1 træk. Dette kunne overvejes at udvides til at større antal træk, men problemet er at antallet af træk stiger eksponentielt, eftersom der for at skue n turer frem, vil være 6^n forskellige måder man kunne trække disse træk, da der er 6 træk per tur. Dette tal kunne dog sænkes væsentligt ved dynamisk programmering, da det bedste træk udelukkende er afhængigt at den nuværende brætkonfiguration, hvilket kan gemmes undervejs, i stedet for at beregne samme brætkonfiguration flere gange.

5.1 Afvigelser fra opgaveudkast

Som udgangspunkt følger det udviklede bibliotek opgaveudkastet (signaturfilen), men et par steder er der afvigelser.

Da hvert pit inderholder et index som viser dets position, er det i funktionen `isHome` ikke nødvendigt at tage boardet som et input. Dette er derfor fjernet

I det oprindelige udkast returnere `distribute` en tuple af typen `board * player * pit`, hvor spilleren repræsenterer ejeren af det pit den sidste bønne landte i, og pittet er netop det pit. Dette skyldes, at `turn` funktionen (givet i

opgaven) tjekker hvorvidt det returnerede pit er et homepit for den returnerede spiller. Dette er en fejl, da det vil give sandt for begge homepits, mens man kun ønsker sandt for sit ejet homepit. Derfor kan typen fjernes fra returværdien og funktionen rettes.

Ligeledes er der i linjen efter endnu en fejl; funktionen kalder `isGameOver b`, hvilket tjekker om det nu outdatede bræt er i en gameover tilstand. Dette bør i stedet være `isGameOver newB`, hvilket det er rettet til.

For at implementere AI funktionaliteten er det nødvendigt at `getMove` ved hvorvidt der spilles 1 eller 2 spillere. Af den grund har både `play`, `turn` og `getMove` et yderligere input som er af typen int (1 for PvP og 2 for PvAI). `play` funktionen er yderligere udvidet med en gameover besked som fortæller hvem der vandt.

Konklusion

Et bibliotek til kørsel af et Awari-spil er således lavet, og det følger i store træk det udkast som kom med opgaven. Den største afvigelse er tilføjeslen af support for en computer-modstander, som godt nok ikke er perfekt. Til biblioteket er en lille applikation udviklet, som sætter det hele i gang. Bibliotekets funktionalitet er verificeret ved hjælp af en whitebox-test, som er succesfuld, og der er ingen ummidelbare bugs i spillet. Det sagt, er koden bestemt ikke særlig elegant. Dette skyldes, at spillet er lavet med det funktionelle paradigme, som, mens det er godt til nogle ting, bestemt ikke egner sig til spil, da de ofte bygger på en state og mange loops - 2 ting som er diamentralt modsat det funktionelle paradigme. Projektet er dog lykkedes alt i alt og spillet vil åbenlyst snart blive spillet verden over.

Appendix A - Kildekode

,

7.1 awariLib.fs

```
1  module Awari
2  type player = Player1 | Player2
3  type pit = {
4    index : int
5    beanCount: int
6  }
7  type board = pit list
8
9  let boardSize = 14
10
11 // Hjælpefunktion, der udskriver spillepladen.
12 // Player1 har den øverste række, mens Player2 har den nederste
13 let printBoard (b:board) : unit =
14     let pitsToString acc i = acc + sprintf "%3i" i.beanCount
15     let player2String = List.fold pitsToString "␣" (List.rev b.[7..12])
16     let player1String = List.fold pitsToString "␣" (b.[0..5])
17     let homePitString = sprintf "%i%21i" b.[13].beanCount b.[6].beanCount
18     printfn "%s\n%s\n%s\n" player2String homePitString player1String
19
20 // Hjælpefunktion til at bestemme om et
21 // bestemt felt er en bestemt spillers hjemmefelt
22 let isHome (p:player) (i:pit) : bool =
23     let pitPlayer = if i.index < 7 then Player1 else Player2
24     (pitPlayer = p) && (i.index%7=6)
25
26 // Hjælpefunktion til at bestemme om en brætkonfiguration
27 // medfører at spillet er slut
28 let isGameOver (b:board) : bool =
29   let count (x:int,y:int) i:pit : int*int =
30     if i.index%7<>6 then
31       if i.index >6 then (x + i.beanCount,y)
32       else (x,y+i.beanCount)
33     else (x,y)
34
35   let (p1Beans,p2Beans) = List.fold count (0,0) b
36   (p1Beans = 0 || p2Beans = 0)
37
38
39 // Hjæplefunktion til at erstatte feltet
40 // i et bestemt index med et givent felt.
41 let replaceAtIndex (index:int) (ni:pit) (b:board) : board =
42     let rep (currentIndex:int) (i:pit) =
43         if currentIndex = index then ni else i;
44     List.mapi rep b
45
46 // Hjælpefunktion til at få hjemmefeltet for en given spiller
47 let getHome (b:board) (p:player):pit =
48     match p with
49     |Player1 -> b.[6]
50     |Player2 -> b.[13]
51
52 // Opdaterer det sidste felt, hvis dette felt kun
53 // indeholder 1 bønne, dvs. distribute endte i et tomt felt
54 // ellers returnerers blot den givne konfiguration af felter
55 let updateLastPit (b:board) (p:player) (i:pit) : (board*pit) =
56     if i.beanCount = 1 && i.index % 7 <> 6 then
57         let home = getHome b p
58         let oppositePit = b.[12 - i.index]
59         let newHomePitCount = home.beanCount + i.beanCount + oppositePit.beanCount
```

```
60          let updatedBoard =
61              b
62              |> replaceAtIndex home.index {home with beanCount = newHomePitCount}
63              |> replaceAtIndex oppositePit.index {oppositePit with beanCount = 0}
64              |> replaceAtIndex i.index {i with beanCount = 0}
65          (updatedBoard,i)
66      else
67          (b,i)
68
69
70  // Hovedfunktionen til at uddele bønnerne på et
71  // specifikt felt ud til alle de efterfølgende felter,
72  // samt fjerne de oprindelige bønder i startfeltet.
73  let distribute (b:board) (p:player) (i:pit) : (board*pit) =
74
75      // Hjælpefunktion til at bestemme hvordan
76      // bønnerne i et givent felt cp ændre sig efter
77      // bønnerne i "op" er blevet fordelt.
78      // op er den pit som alle bønnerne kommer fra.
79      let addBean (op:pit) (cp:pit) : pit =
80          let indexDiff = (cp.index - op.index - 1 + boardSize) % boardSize
81
82          // Skulle det valgte felt indeholde nok bønder
83          // til at komme hele vejen rundt adderes
84          // dette tal til alle felter.
85          //
86          // Herudover sættes op til nu at have 0 bønner
87          let rounds = op.beanCount / boardSize
88          let beanSum = (if cp <> op then cp.beanCount else 0) + rounds
89
90          // Hvis der var nok bønner i op til at nå det nuværrende felt,
91          // forøges antallet af bønner her med 1.
92          if indexDiff < op.beanCount then
93              {cp with beanCount = beanSum + 1}
94          else
95              {cp with beanCount = beanSum}
96
97      if i.beanCount = 0 then (b,i)
98      else
99          let lastIndex = (i.index + i.beanCount) % boardSize
100         let nb = List.map (addBean i) b
101         let ni = nb.[lastIndex]
102         updateLastPit nb p ni
103
104
105 let rec getAiMove (b:board) (p:player) (cIndex:int) (cMax:int) (maxIndex:int)=
106   let (hypoBoard, hypoFinalPit) = distribute b p b.[cIndex]
107   let hypoHome = getHome hypoBoard p
108   let home = getHome b p
109   let homeDiff = hypoHome.beanCount-home.beanCount
110   let hypoMax = (if (isHome p hypoFinalPit) then 5 else homeDiff)
111
112   if (cIndex % 7) = 5 then
113     if hypoMax > cMax then
114       b.[cIndex]
115     else
116       b.[maxIndex]
117   else
118     if hypoMax > cMax then
119       getAiMove b p (cIndex+1) hypoMax cIndex
120     else
121       getAiMove b p (cIndex+1) cMax maxIndex
122
123
124 // Funktionen der får et input fra brugeren, samt validerer dette input
125 let rec getMove (gameType:int) (b:board) (p:player) (s:string) : pit =
126   if (gameType = 1 || (gameType = 2 && p=Player1))then
127     printfn "%s" s
```

```
128       let inputString = System.Console.ReadLine ()
129       if not (List.contains inputString ["1";"2";"3";"4";"5";"6"]) then
130         do printfn "Incorrect input, try again with one of the pits 1-6"
131         getMove gameType b p s
132       else
133         let inputPit = int (inputString)
134         match p with
135         | Player1 -> b.[inputPit-1]
136         | Player2 -> b.[inputPit+6]
137     else
138       do printfn "Ai's move:"
139       getAiMove b p 7 0 0
140
141   // Funktionen der holder styr på turen, dvs om den nuværrende spiller
142   // skal have en tur til, eller om det er en ny spiller
143   let turn (gameType : int) (b : board) (p : player) : board =
144     let rec repeat (b: board) (p: player) (n: int) : board =
145       printBoard b
146       let str =
147         if n = 0 then
148           sprintf "%A's move:" p
149         else
150           "Again? "
151       let i = getMove gameType b p str
152       let (newB, finalPit)= distribute b p i
153       if not (isHome p finalPit)
154          || (isGameOver newB) then
155         newB
156       else
157         repeat newB p (n + 1)
158     repeat b p 0
159
160   // Funktionen der skifter til ny spiller, skulle den gamle
161   // ikke længere have sin tur.
162   let rec play (gameType : int) (b : board) (p : player) : board =
163     if isGameOver b then
164       printBoard b
165       match (b.[6].beanCount - b.[13].beanCount) with
166       | n when n > 0 -> printfn "Game over! The winner is player 1!"
167       | n when n < 0 -> printfn "Game over! The winner is player 2!"
168       | _            -> printfn "Game over! It was a draw!"
169       b
170     else
171       let newB = turn gameType b p
172       let nextP =
173         if p = Player1 then
174           Player2
175         else
176           Player1
177       play gameType newB nextP
178   // Starter spillet med standard brætopsætning
179   let startGame () =
180       let b = List.init 14 (fun x -> {index = x; beanCount = if x % 7 = 6 then 0 else 3})
181       let p = Player1
182
183       play 1 b p |> ignore
184
185   let startAiGame () =
186       let b = List.init 14 (fun x -> {index = x; beanCount = if x % 7 = 6 then 0 else 3})
187       let p = Player1
188
189       play 2 b p |> ignore
```

7.2 awariApp.fsx

```
1  #r "awariLib"
2  open Awari
```

```
3
4  let rec displayMenu () =
5      printfn "Welcome to Awari! \nPlease choose your gamemode (1/2)"
6      printfn "1: Player vs. Player"
7      printfn "2: Player vs. Ai"
8      let input = System.Console.ReadLine()
9      match input with
10     | "1" -> startGame ()
11     | "2" -> startAiGame ()
12     | _   -> printfn "Please try again\n"
13              displayMenu ()
14
15
16 displayMenu ()
```

7.3 awariLibTest.fsx

```
1  #r "awariLib"
2  open Awari
3  // Ikke alle funktioner bliver testet, da funktionerne,
4  // der skaber interaktion med brugeren har sideeffekter,
5  // der er svære at tjekke fra programmets side, uden at
6  // ændre i koden.
7
8  // Funktion til at hjælpe med bræt konstruktion
9  let createBoard f =
10    List.init 14 (fun x -> {index = x; beanCount = f x})
11
12
13
14 let isHomeTest () =
15     let b = createBoard id
16     // 1a & 2a
17     printfn "Player 1 Home Test:    %b" (isHome Player1 b.[6] = true)
18     // 1b && 2a
19     printfn "Player 2 Home Test:    %b" (isHome Player2 b.[13] = true)
20     // 2b
21     printfn "Wrong Home a:          %b" (isHome Player1 b.[13] = false)
22     // 2c
23     printfn "No Home:               %b" (isHome Player1 b.[3] = false)
24
25 let isGameOverTest () =
26     let ba =  createBoard (fun x -> if x < 6 then 0 else 1)
27     // 1a
28     printfn "Player 1 No Beans:     %b" (isGameOver ba = true)
29     let bb = createBoard (fun x -> if x > 6 then 0 else 1)
30     // 1b
31     printfn "Player 2 No Beans:     %b" (isGameOver bb = true)
32     // 1c
33     printfn "Not game over:         %b" (isGameOver (createBoard id) = false)
34
35 let replaceAtIndexTest () =
36     let z x = 0
37     let f x = if x = 3 then 5 else 0
38     let b = createBoard z
39     let nb = createBoard f
40     printfn "Replace at board:      %b" (replaceAtIndex 3 {beanCount = 5; index = 3} b = nb)
41
42 let getHomeTest () =
43     let b = createBoard id
44     printfn "Player 1:              %b" (getHome b Player1 = b.[6])
45     printfn "Player 2:              %b" (getHome b Player2 = b.[13])
46
47 let updateLastPitTest () =
48     let f (l: int list) x = l.[x]
49
50     let listba  = [0;0;0;0;0;1;0;2;1;0;0;0;0;0]
```

```
51      let listnba = [0;0;0;0;0;0;3;0;1;0;0;0;0;0]
52      let ba = createBoard (f listba)
53      let nba = createBoard (f listnba)
54      printfn "Update last pit empty: %b" (fst (updateLastPit ba Player1 ba.[5]) = nba)
55
56
57      let listbb  = [0;0;0;0;1;2;0;2;1;0;0;0;0;0]
58      let listnbb = [0;0;0;0;1;2;0;2;1;0;0;0;0;0]
59      let bb = createBoard (f listbb)
60      let nbb = createBoard (f listnbb)
61      printfn "Update last pit bean:  %b" (fst (updateLastPit bb Player1 bb.[5]) = nbb)
62  let distributeTest () =
63      let f (l: int list) x = l.[x]
64
65      let listba  = [0;0;0;4;0;1;0;2;1;0;0;0;0;0]
66      let listnba = [0;0;0;0;1;2;1;3;1;0;0;0;0;0]
67      let ba = createBoard (f listba)
68      let nba = createBoard (f listnba)
69      printfn "Distribute non-empty:  %b" (fst (distribute ba Player1 ba.[3]) = nba)
70
71
72      let listbb  = [0;0;0;0;1;2;0;2;1;0;0;0;0;0]
73      let listnbb = [0;0;0;0;1;2;0;2;1;0;0;0;0;0]
74      let bb = createBoard (f listbb)
75      let nbb = createBoard (f listnbb)
76      printfn "Distribute empty:      %b" (fst (distribute bb Player1 bb.[1]) = nbb)
77
78  do isHomeTest ()
79  do isGameOverTest ()
80  do getHomeTest ()
81  do replaceAtIndexTest ()
82  do updateLastPitTest ()
83  do distributeTest ()
```