

# PoP - Ugeopgave 7

Christoffer, Inge og Pernille

November 9, 2018

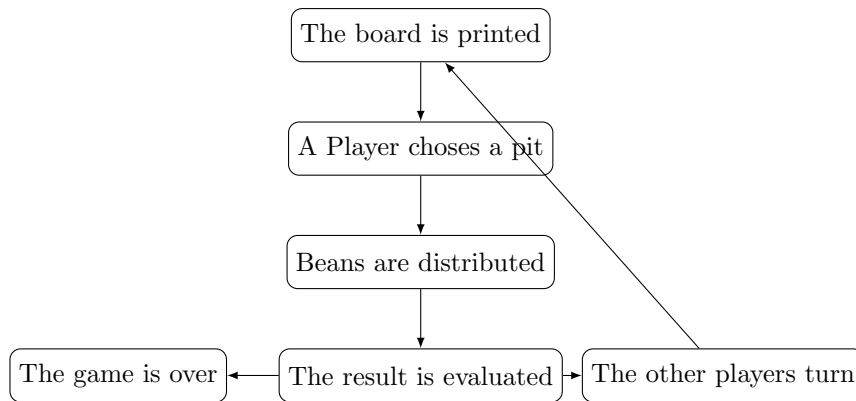


Figure 1: Flow of Awari

## Preface

As a part of the Programming and Problem Solving course, we, three Computer Science students at Copenhagen University, build the game Awari.

## Awari

Awari is an ancient two player game that resembles the beloved Kalaha. The main objective of Awari is to capture the most beans. The game consists of a board with 6 pits and one home pit for each player. Each of the 6 x 2 pits consists of 3 beans. The players must in turn take the amount of beans from a pit on his or her side of the board and distribute them in the following pits. The game continues until one of the two players has no beans left, and the winner is the player who captured the most beans in his or her homepit.

## Problem description

We have implemented the Awari using the functional programming language F#. We were given two functions as a start, a turn and a play function, and a signature file with type indication for several minor functions. To be able to play the game we have created functions to print the board, to distribute the beans, to check for game over etc. All the functions will be elaborated in this rapport.

## Problem analysis and design

### flow

We designed our programme starting by focusing on the board.

1).

```

1  module Awari
2
3  /// A game is played between two players
4  type player = Player1 | Player2
5
6  /// A board consisting of pits.
7  type board = int array
8
9  /// Each player has a set of regular pits and one home pit. A pit holds zero or
10 /// more beans
11 type pit = int
12
13 (*DOCUMENTATION OF printBoard*)
  
```

```

14  /// <summary>
15  /// Prints the board
16  /// </summary>
17  /// <param name="b"> A board to be printed </param>
18  /// <returns>() – it just prints</returns>
19  /// , e.g.,
20  /// <remarks>
21  /// Output is for example,
22  /// <code>
23  ///      3  3  3  3  3  3
24  ///      0                                0
25  ///      3  3  3  3  3  3
26  /// </code>
27  /// Where player 1 is bottom row and rightmost home
28  /// </remarks>
29  let printBoard (b: board) =
30      System.Console.Clear ()
31      let esc = string (char 0x1B)
32      printf "      |"
33      for i = 12 downto 7 do
34          printf "%2i |" b.[i]
35      printfn ""
36      printf " | %2i |                                | %i |\n" b.[13] b.[6]
37      printf "      |"
38      for i = 0 to 5 do
39          printf "%2i |" b.[i]
40      printfn ""
41
42
43  (*DOCUMENTATION OF isGameOver*)
44  /// <summary>
45  /// Checks whether the game is over
46  /// </summary>
47  /// <param name="b"> A board to check</param>
48  /// <returns>True if either side has no beans</returns>
49  let isGameOver (b: board) : bool =
50      match b with
51      | b when Array.forall (fun b -> (b = 0)) b.[0..5] -> true
52      | b when Array.forall (fun b -> (b = 0)) b.[7..12] -> true
53      | b -> false
54
55
56
57  (*DOCUMENTATION OF isHome*)
58  /// <summary>
59  /// Checks whether a pit is the player's home
60  /// </summary>
61  /// <param name="b">A board to check</param>
62  /// <param name="p">The player, whos home to check</param>
63  /// <param name="i">A regular or home pit of a player</param>
64  /// <returns>True if either side has no beans</returns>
65
66  let isHome (b: board) (p: player) (i: pit) : bool =
67      match i with
68      | 6 when p = Player1 -> true
69      | 13 when p = Player2 -> true
70      | _ -> false
71

```

```

72
73 (*DOCUMENTATION OF getMove*)
74 /// <summary>
75 /// Takes the pressed key as input and finds the pit of next move from the user.
76 /// </summary>
77 /// <param name="b">The board the player is choosing from</param>
78 /// <param name="p">The player, whose turn it is to choose</param>
79 /// <param name="q">The string to ask the player</param>
80 /// <returns>The indexnumber of the pit the player has chosen</returns>
81
82 let rec getMove (b:board) (p:player) (q:string) : pit =
83     printfn "%s Choose a pit between 1-6" q
84     let n = int (System.Console.ReadLine ())
85     if (1 <= n && n <= 6) then
86         match p with
87         | Player1 when not (b.[n-1] = 0) -> n-1
88         | Player2 when not (b.[n+6] = 0) -> n+6
89         | _ -> printfn "This pit is empty. Try again."
90                 getMove b p q
91     else
92         printfn "This is not a valid input. Try again."
93         getMove b p ""
94
95 (*DOCUMENTATION OF checkOpp*)
96 /// <summary>
97 /// Checks pit opposit of finalPit
98 /// </summary>
99 /// <param name="b"> A board to check</param>
100 /// <param name="i">The indexnumber of the finalPit of the player who just
101 /// played his/her turn</param>
102 /// <returns>The number of beans in the pit opposite of the finalPit</returns>
103
104 let checkOpp (b:board) (i: pit) : bool =
105     if i = 13 then false
106     elif i = 6 then false
107     else
108         let Opps = (b.Length - 2) - i
109         (b.[Opps] <> 0)
110
111 (*DOCUMENTATION OF finalPitPlayer*)
112 /// <summary>
113 /// Checks whether Player1 or Player2 is the player of the final pit.
114 /// </summary>
115 /// <param name="i">The indexnumber of the finalPit of the player who just
116 /// played his/her turn</param>
117 /// <returns>Player1 or Player2</returns>
118
119 let finalPitPlayer (i: pit) : player =
120     match i with
121     | i when i <= 6 -> Player1
122     | i -> Player2
123
124
125 (*DOCUMENTATION OF distribute*)
126 /// <summary>
127 /// Distributing beans counter clockwise, capturing when relevant
128 /// </summary>
129 /// <param name="b">The present status of the board</param>

```

```

130  /// <param name="p">The player , whos beans to distribute</param>
131  /// <param name="i">The regular pit to distribute</param>
132  /// <returns>A new board after the beans of pit i has been distributed , and which player
133  //val distribute : b:board -> p:player -> i:pit -> board * player * pit
134
135  let rec distribute (b:board) (p:player) (i:pit) : board * player * pit =
136      let mutable j = i + 1
137      ///Let k be the number of pits to distribute
138      let mutable k = b.[i]
139      while k > 0 do
140          if (j <= 13) then
141              b.[j] <- (b.[j] + 1)
142              k <- k - 1
143          if (j > 13) then
144              j <- 0
145          elif k = 0 then
146              j <- j
147          else
148              j <- j + 1
149      let finalPit = j
150      if (checkOpp b finalPit) && (finalPitPlayer finalPit) = p && b.[finalPit] = 1 then
151          let Opps = (b.Length - 2) - finalPit
152          match p with
153          | Player1 -> b.[6] <- b.[6] + b.[Opps] + b.[finalPit]
154          | Player2 -> b.[13] <- b.[13] + b.[Opps] + b.[finalPit]
155          b.[finalPit] <- 0
156          b.[Opps] <- 0
157          b.[i] <- 0
158          (b, (finalPitPlayer finalPit), finalPit)
159
160  (*DOCUMENTATION OF turn*)
161  /// <summary>
162  /// Interact with the user through getMove to perform a possibly repeated turn of a play
163  /// </summary>
164  /// <param name="b">The present state of the board</param>
165  /// <param name="p">The player , whose turn it is</param>
166  /// <returns>A new board after the player's turn</returns>
167
168
169  let turn (b : board) (p : player) : board =
170      let rec repeat (b: board) (p: player) (n: int) : board =
171          printBoard b
172          let str =
173              if n = 0 then
174                  sprintf "%A's move. " p
175              else
176                  "Again "
177          let i = getMove b p str
178          let (newB, finalPitsPlayer, finalPit)= distribute b p i
179          if not (isHome b finalPitsPlayer finalPit)
180              || (isGameOver b) then
181              newB
182          else
183              repeat newB p (n + 1)
184      repeat b p 0
185
186
187  (*DOCUMENTATION OF play*)

```

```

188  /// <summary>
189  /// Play game until one side is empty
190  /// </summary>
191  /// <param name="b">The initial board</param>
192  /// <param name="p">The player who starts</param>
193  /// <returns>A new board after one player has won</returns>
194
195
196  let rec play (b : board) (p : player) : board =
197      if isGameOver b then
198          let esc = string (char 0x1B)
199          if b.[6] > b.[13] then
200              System.Console.WriteLine(esc + "[31;1m" + "Game over. The winner is Player 1" + es
201          elif b.[6] = b.[13] then
202              System.Console.WriteLine(esc + "[33;1m" + "Game over. It's a tie" + esc + "[0m")
203          else
204              System.Console.WriteLine(esc + "[31;1m" + "Game over. The winner is Player 2" + es
205          //printfn "Game over."
206          b
207      else
208          let newB = turn b p
209          let nextP =
210              if p = Player1 then
211                  Player2
212              else
213                  Player1
214          play newB nextP

```