PoP - Ugeopgave 7

Christoffer, Inge og Pernille

November 8, 2018

0.1 Preface

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

0.2 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.

0.3 Problem description

We have implemented the Awari using the functional programming language F#. We were given two functions at the beginning, a turn and a play function, and the then we have made several funktions to be able to play the game. We were To be able to play the game it is necessary for the programme to get a chosen pit.

```
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
30 (*DOCUMENTATION OF isGameOver*)
31 /// <summary>
32 /// Checks whether the game is over
33 /// </summary>
34 /// <param name="b"> A board to check</param>
35 /// <returns>True if either side has no beans</returns>
36 let isGameOver (b: board) : bool =
37   match b with
```

```
38    | b when Array.forall (fun b -> (b = 0)) b.[0..5] -> true
39    | b when Array.forall (fun b -> (b = 0)) b.[7..12] -> true
40    | b -> false
41
42  let printBoard (b: board) =
43    System.Console.Clear ()
44    let esc = string (char 0x1B)
45    printf "      |"
46    for i = 12 downto 7 do
47        printf "%2i |" b.[i]
48    printfn ""
49    printf "| %2i |                          | %i |\n" b.[13] b.[6]
50    printf "      |"
51    for i = 0 to 5 do
52        printf "%2i |" b.[i]
53    printfn ""
54
55
56  (*DOCUMENTATION OF isHome*)
57  /// <summary>
58  /// Checks whether a pit is the player's home
59  /// </summary>
60  /// <param name="b">A board to check</param>
61  /// <param name="p">The player, whos home to check</param>
62  /// <param name="i">A regular or home pit of a player</param>
63  /// <returns>True if either side has no beans</returns>
64
65  let isHome (b: board) (p: player) (i: pit) : bool =
66    match i with
67    | 6 when p = Player1 -> true
68    | 13 when p = Player2 -> true
69    | _ -> false
70
71
72  (*DOCUMENTATION OF getMove*)
73  /// <summary>
74  /// Takes the pressed key as input and finds the pit of next move from the user.
75  /// </summary>
76  /// <param name="b">The board the player is choosing from</param>
77  /// <param name="p">The player, whose turn it is to choose</param>
78  /// <param name="q">The string to ask the player</param>
79  /// <returns>The indexnumber of the pit the player has chosen</returns>
80
81  let rec getMove (b:board) (p:player) (q:string) : pit =
82    printfn "%s Choose a pit between 1-6" q
83    let n = int (System.Console.ReadLine ())
84    if (1 <= n && n <= 6) then
85      match p with
86      | Player1 when not (b.[n-1] = 0) -> n-1
87      | Player2 when not (b.[n+6] = 0) -> n+6
88      | _ -> printfn "This pit is empty. Try again."
89             getMove b p q
90    else
91      printfn "This is not a valid input. Try again."
92      getMove b p ""
93
94  (*DOCUMENTATION OF checkOpp*)
95  /// <summary>
```

```
96  /// Checks pit opposit of finalPit
97  /// </summary>
98  /// <param name="b"> A board to check</param>
99  /// <param name="i">The indexnumber of the finalPit of the player who just
100 /// played his/her turn</param>
101 /// <returns>The number of beans in the pit opposite of the finalPit</returns>
102
103 let checkOpp (b:board) (i: pit) : bool =
104   if i = 13 then false
105   elif i = 6 then false
106   else
107     let Opps = (b.Length - 2) - i
108     (b.[Opps] <> 0)
109
110 (*DOCUMENTATION OF finalPitPlayer*)
111 /// <summary>
112 /// Checks whether Player1 or Player2 is the player of the final pit.
113 /// </summary>
114 /// <param name="i">The indexnumber of the finalPit of the player who just
115 /// played his/her turn</param>
116 /// <returns>Player1 or Player2</returns>
117
118 let finalPitPlayer (i: pit) : player =
119   match i with
120   | i when i <= 6 -> Player1
121   | i -> Player2
122
123
124 (*DOCUMENTATION OF distribute*)
125 /// <summary>
126 /// Distributing beans counter clockwise, capturing when relevant
127 /// </summary>
128 /// <param name="b">The present status of the board</param>
129 /// <param name="p">The player, whos beans to distribute</param>
130 /// <param name="i">The regular pit to distribute</param>
131 /// <returns>A new board after the beans of pit i has been distributed, and which player
132 //val distribute : b:board -> p:player -> i:pit -> board * player * pit
133
134 let rec distribute (b:board) (p:player) (i:pit) : board * player * pit =
135   let mutable j = i + 1
136   ///Let k be the number of pits to distribute
137   let mutable k = b.[i]
138   while k > 0 do
139     if (j <= 13) then
140       b.[j] <- (b.[j] + 1)
141       k <- k - 1
142     if (j > 13) then
143       j <- 0
144     elif k = 0 then
145       j <- j
146     else
147       j <- j + 1
148   let finalPit = j
149   if (checkOpp b finalPit) && (finalPitPlayer finalPit) = p && b.[finalPit] = 1 then
150     let Opps = (b.Length - 2) - finalPit
151     match p with
152     | Player1 -> b.[6] <- b.[6] + b.[Opps] + b.[finalPit]
153     | Player2 -> b.[13] <- b.[13] + b.[Opps] + b.[finalPit]
```

```
154       b.[finalPit] <- 0
155       b.[Opps] <- 0
156     b.[i] <- 0
157     (b, (finalPitPlayer finalPit), finalPit)
158
159 (*DOCUMENTATION OF turn*)
160 /// <summary>
161 /// Interact with the user through getMove to perform a possibly repeated turn of a play
162 /// </summary>
163 /// <param name="b">The present state of the board</param>
164 /// <param name="p">The player, whose turn it is</param>
165 /// <returns>A new board after the player's turn</returns>
166
167
168 let turn (b : board) (p : player) : board =
169   let rec repeat (b: board) (p: player) (n: int) : board =
170     printBoard b
171     let str =
172       if n = 0 then
173         sprintf "%A's move. " p
174       else
175         "Again "
176     let i = getMove b p str
177     let (newB, finalPitsPlayer, finalPit)= distribute b p i
178     if not (isHome b finalPitsPlayer finalPit)
179        || (isGameOver b) then
180       newB
181     else
182       repeat newB p (n + 1)
183   repeat b p 0
184
185
186 (*DOCUMENTATION OF play*)
187 /// <summary>
188 /// Play game until one side is empty
189 /// </summary>
190 /// <param name="b">The initial board</param>
191 /// <param name="p">The player who starts</param>
192 /// <returns>A new board after one player has won</returns>
193
194
195 let rec play (b : board) (p : player) : board =
196   if isGameOver b then
197     let esc = string (char 0x1B)
198     if b.[6] > b.[13] then
199       System.Console.WriteLine(esc + "[31;1m" + "Game over. The winner is Player 1" + es
200     elif b.[6] = b.[13] then
201       System.Console.WriteLine(esc + "[33;1m" + "Game over. It's a tie" + esc + "[0m")
202     else
203       System.Console.WriteLine(esc + "[31;1m" + "Game over. The winner is Player 2" + es
204     //printfn "Game over."
205     b
206   else
207     let newB = turn b p
208     let nextP =
209       if p = Player1 then
210         Player2
211       else
```

```
212             Player1
213         play newB nextP
```