12.2-x Lotto example sketch – OLD

Description

The next example is an experiment on playing Lotto with two different strategies: choosing random numbers or playing the same numbers each time.

In this simplified version of Lotto you have to guess seven different numbers in the interval from 1 to 34. You may submit one bet each week. At the end of the week the Lotto system chooses randomly seven different winner numbers, and the winning players are those that have submitted a bet with these winner numbers.

The players are represented by parallel processes, where each player for each week (if he/she is in mood for playing) submit a bet to lotto. A bet is represented by an object of class `Bet`:

```
class Player: MonitorProcess
   kind: ref String
   myBet: ref Bet
   if (inMoodForPlayingSameNumbers) :then
      kind := "same"
   :else
      kind := "random"
   myBet :=
      Bet(clock.now, this(Player).kind,
          sevenRandomNumbers)

   cycle -- for each week
      if (inMoodForPlaying) :then
         if (kind = "random") :then
            myBet :=
               Bet(clock.now, this(Player).kind,
                   sevenRandomNumbers)
         lotto.submitBet(myBet)
      clock.waitAweek
```

In real life the players are interested in whether they win or not, as winning comes with money. As we are only interested in what kind of strategy wins first/most, we represent the kind of players by a String with the kinds “random” and “same”. When a winner is found we are only interested in the kind of the winning player. The bets are therefore represented by objects of class `Bet`, with the kind of player and seven numbers. Bets are only valid in the week where they are made, so a bet also has a value variable representing the time of the bet:

```
class Bet(timeIssued: var TimeOfDay,
   playerKind: var String,
   numbers: ref Array(7, Integer):
```

Each week each player decides first if to play this week and second if to play seven random numbers or not. In the experiment this is simply represented by two functions that use a random function that delivers a random value in a given interval:

```
inTheMoodForPlaying -> m: var Boolean:
   m := random(0,1) <> 0

inTheMoodForPlayingSameNumbers -> m: var Boolean:
   m := random(0,1) <> 0
```

The function `sevenRandomNumbers` is also based on the `random` function:

```
sevenRandomNumbers -> srn: ref Indexed(7, Integer):
   inx: var Integer
   srn := Array(7, Integer)
   next:
```

```
         n := random(1,34)
         if (srn.has(n)) :then
           restart(next)
         :else
            srn.put(n):at[inx]
            inx := inx + 1
            if inx < 7 :then
               restart(next)
```

The Lotto is represented by an object `lotto`. This object keeps the bets being submitted, it has the winning bet, and the deadline for submitting bets. As players in parallel submit bets by calling the method `submitBet`, the `lotto` object is defined as a monitor with `submitBet` as an entry method. When a player has submitted a bet, it has to wait for a week, represented by the method `waitAweek`:

```
   lotto: obj Monitor
      bets: obj Set(Bet)
      winningBet: ref Bet
      deadline: var TimeOfDay

      clearBets: entry
         bets.clear

      submitBet(b: ref Bet): entry
         if ((b.timeIssued <= deadline
) and
            ( b.timeIssued >= (deadline - clock.oneWeek) :then
               bets.insert(b)

      findWinningBets
         noOfRandomWinners, noOfSameWinners: var
 Integer
         winningBet:= Bet(clock.now, "winning bet",
                          sevenRandomNumbers)
         -- check if any bet matches winningBet
         bets.scan
            if (current.numbers =
                winningBet.numbers) :then
               if (current.playerKind = "random") :then
                   noOfRandomWinners := noOfRandomWinners + 1
               :else
                   noOfSameWinners := noOfSameWinners + 1
               -- print noOfRandomWinners, noOfSameWinners
```

The experiment is represented by a `BasicSystem` object. It generates a number of `Player` objects, starts the lotto by setting the deadline, and then starts the generated `Player` objects. It then repeatedly waits a week before finding winning players, and then clear the bets and set a new deadline.

```
lottoExperiment: obj MonitorSystem
   -- all the classes, methods and objects introduced above, i.e
   -- Player, RandomNumbersPlayer, SameNumbersPlayer, Bet
   -- inTheMoodForPlaying, inTheMoodForPlayingSameNumbers
   -- sevenRandomNumbers, lotto

   maxNoOfPlayers: val 100000
   inx: var Integer
   players: obj Set(Player)

   cycle -- generate players
      if (inx < maxNoOfPlayers) :then
         inx := inx + 1
         players.insert(Player)

   -- starting lotto and the players:
   lotto.deadline := clock.now + clock.oneWeek
   -- deadline must be set before starting the players
   players.scan
```

```
      current.start

   cycle
      clock.waitAweek
      lotto.findWinningBets
      lotto.clearBets
      lotto.deadline := clock.now + clock.oneWeek
```

The whole experiment relies on the availability of a clock that can deliver the time now (represented by a value of type `TimeOfDay`).

Vi skal sikkert fjerne Date og TimeOfDay her da de er tidligere.
Og her er så den berømte AnyTime

```
class Date(year, month, day: var integer): Value
   setDate(y, m, d: var integer):
      year := y
      month := m
      day := d
   asString -> s: var String:
      s := year + "." + month + "." + day

class TimeOfDay(th: var Time.Hours): Value
   - :
      in p: var TimeOfDay
      out r: var Time.Hours
      r.magnitude := th.magnitude - p.th

class AnyTime(d: var Date, td: var TimeOfDay(0 hours)): Value
   - :
      in p: var AnyTime(Date, TimeOfDay)
      out r: var Time.Hours
      r.magnitude := td.th.magnitude - p.td.th


clock: obj
      now -> var t: TimeOfDay:
         ...
      oneWeek -> var h: Time.Hours:
         h := 7 * 24 hour

      waitAweek:
         ...
```

+++ complete code:
Er det BasicSystem eller MonitorSystem? Monitor bruges tidligere så vel MonitorSystem?

```
lottoExperiment: obj BasicSystem

   sevenRandomNumbers -> srn: ref Indexed(7, Integer):
      inx: var Integer
      srn := Array(7, Integer)
      next:
         n := random(1,34)
         if srn.has(n) :then
           restart(next)
         :else
            srn.put(n):at[inx]
            inx := inx + 1
            if inx < 7 :then
               restart(next)

   inTheMoodForPlaying -> m: var Boolean:
      i: var Integer
      i := random(0,1)
      m := random(0,1) <> 0
```

```
   inTheMoodForPlayingSameNumbers -> m: var Boolean:
      i: var Integer
      i := random(0,1)
      m := random(0,1) <> 0

   class Bet(timeIssued: var TimeOfDay,
       playerKind: ref String, numbers: ref Indexed(7, Integer)

   class Player: Process
      kind: ref String
      myBet: ref Bet
      if inMoodForPlayingSameNumbers :then
         kind := "same"
      :else
         kind := "random"
      myBet :=
         Bet(clock.now, this(Player).kind,
             sevenRandomNumbers)
      cycle -- for each week
         if inMoodForPlaying :then
            if (kind = "random") :then
               myBet :=
                  Bet(clock.now, this(Player).kind,
                      sevenRandomNumbers)
            lotto.submitBet(myBet)
         clock.waitAweek

   lotto: obj Monitor
      bets: obj Set(Bet)
      winningBet: ref Bet
      deadline: var TimeOfDay

      clearBets: entry
         bets.clear

      submitBet(b: ref Bet): entry
         if ((b.timeIssued <= deadline
) and
            (b.timeIssued >= (deadline - clock.oneWeek) :then
               bets.insert(b)

      findWinningBets
         noOfRandomWinners, noOfSameWinners: var
 Integer
         winningBet:= Bet(clock.now, "winning bet",
                          sevenRandomNumbers)
         -- check if any bet matches winningBet
         bets.scan
            if (current.numbers =
                winningBet.numbers) :then
               if (current.kind = "random") :then
                   noOfRandomWinners := noOfRandomWinners + 1
               :else
                   noOfSameWinners := noOfSameWinners + 1
               -- print noOfRandomWinners, noOfSameWinners


   maxNoOfPlayers: val 100000
   inx: var Integer
   players: obj Set(Player)

   cycle -- generate players
      if (inx < maxNoOfPlayers) :then
         inx := inx + 1
         players.insert(Player)

   -- starting lotto and the players:
```

```
lotto.deadline := clock.now + clock.oneWeek
-- deadline must be set before starting the players
players.scan
   current.start

cycle
   clock.waitAweek
   lotto.findWinningBets
   lotto.clearBets
   lotto.deadline := clock.now + clock.oneWeek
```