

Solution 10: Agents and boardgames

ETH Zurich

1 Observing the temperature

Solution

Listing 1: Class *OBSERVABLE_TEMPERATURE_SENSOR*

```
class
2  OBSERVABLE_TEMPERATURE_SENSOR
4  inherit
6  TEMPERATURE_SENSOR
   redefine
8     set_value ,
   default_create
10  end
12 feature {NONE} -- Initialization
14  default_create
   -- Initialize event type.
16  do
   create value_changed_event_channel.make
18  end
20 feature -- Basic operations
22  set_value (a_value: REAL)
   -- Set 'value' to 'a_value' and notify observers.
24  do
   value := a_value
26  from
   value_changed_event_channel.start
28  until
   value_changed_event_channel.after
30  loop
   value_changed_event_channel.item_for_iteration . call ([value])
32  value_changed_event_channel.forth
   end
34  end
36 feature -- Access
```

```
38 value_changed_event_channel: LINKED_LIST [PROCEDURE [ANY, TUPLE [REAL]]]
   -- Event queue where procedures can be subscribed
40
end
```

Listing 2: Class *HEATING_CONTROLLER*

```
class
2  HEATING_CONTROLLER

4 inherit

6  ANY
   redefine
8   default_create
   end

10 feature {NONE} -- Initialization
12   default_create
14   -- Initialize agent.
   do
16   adjust_heating_agent := agent adjust_heating (?)
   end

18   adjust_heating_agent: PROCEDURE [ANY, TUPLE [REAL]]
20   -- Agent

22 feature -- Element change

24   set_sensor (a_sensor: OBSERVABLE_TEMPERATURE_SENSOR)
   -- Set 'sensor' to 'a_sensor'.
26   -- Unsubscribe 'adjust_heating' from old sensor (if existing).
   -- Subscribe 'adjust_heating' to new sensor.
28   do
   if sensor /= Void then
30     sensor.value_changed_event_channel.prune_all (adjust_heating_agent)
   end
32   sensor := a_sensor
   sensor.value_changed_event_channel.extend (adjust_heating_agent)
34   end

36 feature -- Given features

38   adjust_heating (a_value: REAL)
   -- Output what to do concerning the heating.
40   do
   if a_value >= 20.0 then
42     io.put_string ("Turn off heating")
     io.new_line
44   else
     io.put_string ("Turn on heating")
46     io.new_line
```

```
48     end
      end
50  sensor: OBSERVABLE_TEMPERATURE_SENSOR
      -- Currently used sensor
52
end
```

2 Reviewing reference passing

Solution

The result of the four print instructions is:

- test
- test_foo
- test_foo_again?
- test_foo_again?

3 The final project: programming a boardgame (part 4)

Solution

Go through the code step by step and test it on your machine. Some print statements have been added to facilitate your task.

Listing 3: Class *APPLICATION*

```
1 class
  APPLICATION
3
4
5 inherit
  ARGUMENTS
6
7 create
  make
8
9
10 feature
11
  make
12
13   -- Launch the application.
14   local
15     boardgame_simulator: GAME
16   do
17     create boardgame_simulator.make
18     boardgame_simulator.play
19   end
20 end
```

Listing 4: Class *GAME*

```
class
2  GAME
3
4 create
  make
5
6
7 feature {NONE} -- Initialization
8
```

```
make
10  -- Run application.
    local
12    i: INTEGER
    p: PLAYER
14  do
    create players.make (1, number_of_players)
16    create game_board.make (players)
    create die_1.make
18    create die_2.make
    from
20    i := 1
    until
22    i > players.count
    loop
24    create p.make ("Player" + i.out)
    p.credit (player_initial_amount)
26    p.set_location (game_board.start_square)
    players [i] := p
28    i := i + 1
    end
30    print (game_board.out)
    create a_file_printer .make
32    a_file_printer .print_game_trace (game_board.out)
    end
34
feature -- Basic operations
36
play
38  -- Start a game.
    local
40    i: INTEGER
    do
42    print ("%N%N*** Simple Snakes and Ladders Boardgame ***%N")
    from
44
    until
46    has_winner
    loop
48    from
    i := 1
50    until
    has_winner or else i > number_of_players
52    loop
    players [i].play (die_1, die_2)
54    print (game_board.out)
    a_file_printer .print_game_trace(game_board.out)
56    if players [i].location = game_board.end_square then
    find_winner
58    from
    winners.start
60    until
```

```

        winners.after
62     loop
        print ("%NThe winner(s): " + winners.item.name + " with money: " +
            winners.item.money.out + " CHF")
64         has_winner := True
            winners.forth
66     end
        end
68     i := i + 1
        end
70 end
    a_file_printer . close
72 print ("%N*** Game Over ***")
ensure
74     game_has_winner: has_winner
end
76
find_winner
78     -- Determine winner.
local
80     i: INTEGER
do
82     from
        i := 1
84     create winners.make
        winners.extend (players [i])
86     i := i + 1
invariant
88     i >= 2
        i <= number_of_players + 1
90 until
        i > number_of_players
92 loop
    if players [i].money > (winners.i_th (winners.count)).money then
94         winners.wipe_out
            winners.extend (players [i])
96     else if players [i].money = (winners.i_th (winners.count)).money then
            winners.extend (players [i])
98     end
        end
100    i := i + 1
variant
102    number_of_players - i + 1
end
104 end

106 feature -- Status
108     game_board: BOARD
        -- The game board.
110     number_of_players: INTEGER = 2
    
```

```

112     -- For testing purposes, the number of players is set to 2.
114     players: ARRAY [PLAYER]
           -- Container for players.
116
117     player_initial_amount: INTEGER = 7
118     -- Players initial amount.
120
121     die_1: DIE
           -- The first die.
122
123     die_2: DIE
           -- The second die.
124
125     has_winner: BOOLEAN
           -- Does the game have a winner?
126
127     winners: LINKED_LIST [PLAYER]
128     -- The winner(s) of the game.
129
130     a_file_printer : FILE_PRINTER
           -- Print a game representation into a plain text file .
131
132
133 invariant
134
135     game_board_exists: game_board /= Void
136
137     players_exist : players /= Void and then not players.is_empty
138
139     number_of_players_consistent: number_of_players >= 2 and number_of_players <= 6
140
141     dice_exist : die_1 /= Void and die_2 /= Void
142
143
144 end
  
```

Listing 5: Class *BOARD*

```

1 class
2   BOARD
3
4   inherit ANY
5
6   redefine out end
7
8   create
9     make
10
11 feature -- Creation
12
13   make (plrs: ARRAY [PLAYER])
           -- Create a board with squares.
14
15   require
           players_exist : plrs /= Void
  
```

```

17  local
    i: INTEGER
19  square_x, square_y: SQUARE
    do
21  players := plrs
    from
23  i := 1
    create start_square.make ("%NSquare" + i.out)
25  square_x := start_square
    print (square_x.name + " created.%N")
27  i := i + 1
    until
29  i > max_number_of_squares
    loop
31  inspect i
    when 5, 15, 25, 35 then
33  create {BAD_INVESTMENT_SQUARE}square_y.make ("Bad Investment
        Square" + i.out)
    when 10, 20, 30, 40 then
35  create {LOTTERY_WIN_SQUARE}square_y.make ("Lottery Win Square" + i.
        out)
    when 4, 14, 24, 34 then
37  create {SNAKE_HEAD_SQUARE}square_y.make("Snake Head Square" + i.out
        )
    when 3, 13, 23, 33 then
39  create {SNAKE_BODY_SQUARE}square_y.make("Snake Body Square" + i.out
        )
    when 2, 12, 22, 32 then
41  create {SNAKE_TAIL_SQUARE}square_y.make("Snake Tail Square" + i.out)
    when 7, 17, 27, 37 then
43  create {LADDER_BOTTOM_SQUARE}square_y.make("Ladder Bottom Square
        " + i.out)
    when 8, 18, 28, 38 then
45  create {LADDER_BODY_SQUARE}square_y.make("Ladder Body Square" + i.
        out)
    when 9, 19, 29, 39 then
47  create {LADDER_TOP_SQUARE}square_y.make("Ladder Top Square" + i.out)
    else
49  create square_y.make ("Square" + i.out)
    end
51  print (square_y.name + " created.%N")
    square_x.set_next (square_y)
53  square_y.set_previous (square_x)
    square_x := square_y
55  i := i + 1
    end
57  end_square := square_y
    end
59  feature -- Access
61  max_number_of_squares: INTEGER = 40
  
```

```
63     -- The max number of squares supported by the current board.
64
65     start_square: SQUARE
66         -- The start square.
67
68     end_square: SQUARE
69         -- The end square.
70
71     players: ARRAY [PLAYER]
72         -- The players in the game.
73
74     feature -- Output
75
76     out: STRING
77         -- New string containing printable representation of 'Current'.
78
79     local
80         sq: SQUARE
81         i: INTEGER
82
83     do
84         from
85             sq := start_square
86             create Result.make_from_string ("%N|")
87         until
88             sq = Void
89         loop
90             from
91                 i := 1
92             until
93                 i > players.count
94             loop
95                 if players [i].location = sq then
96                     Result.append (i.out)
97                 end
98                 i := i + 1
99             end
100             Result.append (sq.out)
101             Result.append (" |")
102             sq := sq.next
103         end
104     ensure then
105         out_not_void: Result /= Void
106     end
107
108     invariant
109     max_number_of_squares_consistent: max_number_of_squares = 40
110     start_square_exists : start_square /= Void
111     end_square_exists: end_square /= Void
112 end
```

Listing 6: Class *PLAYER*

```
class
2 PLAYER
```

```
4 create
  make
6
7 feature -- Access
8
9   name: STRING
10    -- Player name.
11
12   location: SQUARE
13    -- 'Current' location.
14
15   money: INTEGER
16    -- player current amount of money.
17
18 feature {NONE} -- Initialization
19
20   make (n: STRING)
21    -- Create a player with name.
22   require
23    name_exists: n /= Void and then not n.is_empty
24   do
25    name := n
26   ensure
27    name_set: name = n
28   end
29
30 feature -- Status setting
31
32   set_location (loc: SQUARE)
33    -- Set location for 'Current'.
34   require
35    location_exists: loc /= Void
36   do
37    location := loc
38   ensure
39    location_set: location = loc
40   end
41
42 feature -- Basic operations
43
44   play (d1,d2: DIE)
45    -- Play a turn.
46   require
47    dice_exist: d1 /= Void and d2 /= Void
48   local
49    dice_result: INTEGER
50   do
51    print ("%N" + name + " rolls dice...")
52    d1.roll
53    d2.roll
54    print ("die1: " + d1.face_value.out + " die2: " + d2.face_value.out)
```

```

    dice_result := d1.face_value + d2.face_value
56  move ( dice_result )
    end
58
move (n: INTEGER)
60  -- Move 'Current' n steps forwards.
    require
62  n_consistent: n >= 2 and n <= 12
    local
64  i: INTEGER
    do
66  from
        i := 1
68  until
        location.next = Void or else i > n
70  loop
        location := location.next
72  i := i + 1
    end
74  if location.next = Void and i <= n then
        move_back (n - i + 1)
76  end
        location.land_on (Current)
78  end

80  move_back (n: INTEGER)
    -- Move 'Current' n steps backwards.
82  require
        n_consistent: n >= 1 and n <= 12
84  local
        i: INTEGER
86  do
        from
88  i := 1
        until
90  i > n
        loop
92  location := location.previous
        i := i + 1
94  end
96  end

98  credit (amount: INTEGER)
    -- Credit amount to 'Current'.
    require
100  amount_positive: amount > 0
    do
102  money := money + amount
        print ("%N" + name + " credited with " + amount.out + " CHF. Total money:
            " + money.out)
104  ensure
        money_credited: money = old money + amount
```

```

106 end
108 debit (amount: INTEGER)
    -- Debit amount to 'Current'.
110 require
    amount_positive: amount > 0
112 do
    if is_money_enough (amount) then
114     money := money - amount
        print ("%N" + name + " debited with " + amount.out + " CHF. Total
            money: " + money.out)
116     else
        money := 0
118     print ("%N" + name + " debited with " + amount.out + " CHF but did not
            own enough money. Total money: " + money.out)
        end
120 ensure
    money_debited: money = (old money - amount).max (0)
122 end

124 feature {NONE} -- Implementation

126 is_money_enough (amount: INTEGER): BOOLEAN
    -- Is 'Current's money enough for withdrawing 'amount'?
128 do
    Result := money - amount >= 0
130 end

132 invariant

134 name_exists: name /= Void and then not name.is_empty
    money_non_negative: money >= 0
136 end
  
```

Listing 7: Class *DIE*

```

1 class
    DIE
3
    create
5     make

7
    feature {NONE} -- Initialization
9
        make
11     -- Create a die with valid initial face value.
        do
13         face_value := rand.item \\ 6 + 1
        end
15
    feature -- Access
  
```

```
17   face_value: INTEGER
19   feature -- Basic operations
21   roll
23     -- Roll die
24   do
25     rand.forth
26     face_value := rand.item \\ 6 + 1
27   end
29 feature {NONE} -- Implementation
31   rand: RANDOM
32     -- Pseudo-random number generator.
33   local
34     t: TIME
35     seed: INTEGER
36   once
37     create t.make_now
38     seed := (t.fine_seconds * 1000).rounded
39     create Result.set_seed (seed)
40     Result.start
41   end
43 invariant
44   six_sided_die : face_value >= 1 and face_value <= 6
45 end
```

Listing 8: Class *FILE_PRINTER*

```
class
2  FILE_PRINTER
4  create
5    make
6
7  feature -- Initialization
8
9    make
10     -- Initialize 'Current'.
11   do
12     create internal_file .make_open_append ("game_trace.txt")
13   end
14
15  feature -- Access
16
17
18 feature -- Basic operations
19
20   print_game_trace (s: STRING)
```

```
    -- Print game trace on flat file .
22  require
    game_trace_exists: s /= Void and not s.is_empty
24  do
    internal_file . putstring (s)
26  end

28  close
    -- Close file.
30  do
    internal_file . close
32  end

34 feature {NONE} -- Implementation

36  internal_file : PLAIN_TEXT_FILE

38 invariant
    internal_file_exists : internal_file /= Void
40
end
```

Listing 9: Class *SQUARE*

```
1 class
  SQUARE
3
  inherit
5  ANY

7 redefine out end

9 create
  make

11 feature {NONE} -- Initialization
13
  make (n: STRING)
15    -- Initialization for 'Current'.
  require
17    name_exists: n /= Void and then not n.is_empty
  do
19    name := n
  ensure
21    name_set: name = n
  end

23 feature -- Access
25
  name: STRING
27    -- The square name.

29 next: SQUARE
```

```

    -- The next square.
31  previous: SQUARE
33  -- The previous square.

35 feature -- Status setting

37  set_next (sq: SQUARE)
    -- Set next square.
39  require
    square_exists: sq /= Void
41  do
    next := sq
43  ensure
    next_square_set: next = sq
45  end

47  set_previous (sq: SQUARE)
    -- Set previous square.
49  require
    square_exists: sq /= Void
51  do
    previous := sq
53  ensure
    previous_square_set: previous = sq
55  end

57 feature -- Basic operations

59  land_on (p: PLAYER)
    -- Action to be executed when player lands on 'Current'.
61  require
    player_exists: p /= Void
63  do
    print ("%N" + p.name + " landed on " + name)
65  end

67 feature -- Output

69  out: STRING
    -- New string containing printable representation of 'Current'.
71  do
    Result := create {STRING}.make_empty
73  ensure then
    out_not_void: Result /= Void
75  end

77 invariant
    name_exists: name /= Void and then not name.is_empty
79  end
end
```

Listing 10: Class *LADDER_BODY_SQUARE*

```
class
2  LADDER_BODY_SQUARE

4  inherit
   SQUARE
6
   redefine land_on, out end
8
   create
10  make

12  feature -- Access

14  amount_to_be_debited: INTEGER = 5
      -- amount to be debited.
16
   feature -- Basic operations
18
   land_on (p: PLAYER)
20     -- Action to be executed when player lands on 'Current'.
       do
22     precursor (p)
         p.debit (amount_to_be_debited)
24     ensure then
         player_debited: p.money = (old p.money - amount_to_be_debited).max (0)
26     end

28  feature -- Output

30  out: STRING
      -- New string containing printable representation of 'Current'.
32  do
     Result := "H"
34  ensure then
     out_not_void: Result /= Void
36  end
end
```

Listing 11: Class *LADDER_BOTTOM_SQUARE*

```
1  class
   LADDER_BOTTOM_SQUARE
3
   inherit
5   SQUARE
   redefine land_on, out end
7
   create
9   make

11  feature -- Basic operations
```

```
13 land_on (p: PLAYER)
    -- Action to be executed when player lands on 'Current'.
15 do
    precursor (p)
17 p.move (2)
    end
19
feature -- Output
21
    out: STRING
23    -- New string containing printable representation of 'Current'.
    do
25        Result := "="
    ensure then
27        out_not_void: Result /= Void
    end
29 end
```

Listing 12: Class *LADDER_TOP_SQUARE*

```
1 class
    LADDER_TOP_SQUARE
3
    inherit
5    SQUARE

7 redefine land_on, out end

9 create
    make
11
feature -- Basic operations
13
    land_on (p: PLAYER)
15    -- Action to be executed when player lands on 'Current'.
    do
17        precursor (p)
    end
19
feature -- Output
21
    out: STRING
23    -- New string containing printable representation of 'Current'.
    do
25        Result := ">"
    ensure then
27        out_not_void: Result /= Void
    end
29 end
```

Listing 13: Class *LOTTERY_WIN_SQUARE*

```
1 class
```

```

  LOTTERY_WIN_SQUARE
3
  inherit
5  SQUARE

7  redefine land_on, out end

9  create
   make
11
  feature -- Access
13
   amount_to_be_credited: INTEGER = 10
15   -- amount to be debited.

17  feature -- Basic operations

19  land_on (p: PLAYER)
   -- Action to be executed when player lands on 'Current'.
21  do
   precursor (p)
23   p.credit (amount_to_be_credited)
   ensure then
25   player_credited: p.money = old p.money + amount_to_be_credited
   end
27
  feature -- Output
29
   out: STRING
31   -- New string containing printable representation of 'Current'.
   do
33   Result := "$"
   ensure then
35   out_not_void: Result /= Void
   end
37 end
```

Listing 14: Class *BAD_INVESTMENT_SQUARE*

```

1  class
   BAD_INVESTMENT_SQUARE
3
  inherit
5  SQUARE

7  redefine land_on, out end

9  create
   make
11
  feature -- Access
13
   amount_to_be_debited: INTEGER = 5
```

```

15     -- amount to be debited.
17 feature -- Basic operations
19   land_on (p: PLAYER)
      -- Action to be executed when player lands on 'Current'.
21   do
      precursor (p)
23     p.debit (amount_to_be_debited)
      ensure then
25     player_debited: p.money = (old p.money - amount_to_be_debited).max (0)
      end
27
28 feature -- Output
29
30   out: STRING
      -- New string containing printable representation of 'Current'.
31   do
32     Result := "#"
      ensure then
33     out_not_void: Result /= Void
      end
34
35 end

```

Listing 15: Class *SNAKE_BODY_SQUARE*

```

1 class
2   SNAKE_BODY_SQUARE
3
4   inherit
5   SQUARE
6
7   redefine land_on, out end
8
9   create
10    make
11
12   feature -- Access
13
14     amount_to_be_credited: INTEGER = 5
15     -- amount to be debited.
16
17   feature -- Basic operations
18
19     land_on (p: PLAYER)
      -- Action to be executed when player lands on 'Current'.
20     do
21       precursor (p)
22       p.credit (amount_to_be_credited)
23     ensure then
24       player_credited: p.money = old p.money + amount_to_be_credited
25     end
26
27

```

```
feature -- Output
29
  out: STRING
31  -- New string containing printable representation of 'Current'.
  do
33    Result := "-"
  ensure then
35    out_not_void: Result /= Void
  end
37 end
```

Listing 16: Class *SNAKE_HEAD_SQUARE*

```
1 class
  SNAKE_HEAD_SQUARE
3
  inherit
5  SQUARE

7 redefine land_on, out end

9 create
  make
11
  feature -- Basic operations
13
  land_on (p: PLAYER)
15  -- Action to be executed when player lands on 'Current'.
  do
17    precursor (p)
    p.move_back (2)
19    print ("%N" + p.name + " landed on a snake tail!" + p.location.out)
  end
21
  feature -- Output
23
  out: STRING
25  -- New string containing printable representation of 'Current'.
  do
27    Result := ""
  ensure then
29    out_not_void: Result /= Void
  end
31 end
```

Listing 17: Class *SNAKE_TAIL_SQUARE*

```
1 class
  SNAKE_TAIL_SQUARE
3
  inherit
5  SQUARE
```

```
7 redefine land_on, out end

9 create
  make
11
12 feature -- Basic operations
13
14   land_on (p: PLAYER)
15     -- Action to be executed when player lands on 'Current'.
16     do
17       precursor (p)
18     end
19
20 feature -- Output
21
22   out: STRING
23     -- New string containing printable representation of 'Current'.
24     do
25       Result := ""
26     ensure then
27       out_not_void: Result /= Void
28     end
29 end
```