Solution 10: Agents and boardgames

ETH Zurich

1 Air conditioning

Listing 1: Class TEMPERATURE_SENSOR

class TEMPERATURE_SENSOR

inherit ANY
    redefine
        default_create
    end

feature {NONE} -- Initialization
default_create
    -- Initialize event type.
do
        create observers.make
    end

feature -- Access
temperature: REAL
    -- Temperature value in degrees Celcius.

feature -- Status report
valid_temperature (a_value: REAL): BOOLEAN
    -- Is ‘a_value’ a valid temperature?
do
        Result := a_value >= −273.15
    end

feature -- Basic operations
set_temperature (a_temperature: REAL)
    -- Set ‘temperature’ to ‘a_temperature’ and notify observers.
require
    valid_temperature: valid_temperature (a_temperature)
do
    temperature := a_temperature
    from
        observers.start
    until
        observers.after
    loop
observers.item_for_iteration.call ([temperature])
observers.forth
end
ensure
  temperature_set: temperature = a_temperature
end

feature -- Subscription
subscribe (an_observer: PROCEDURE [ANY, TUPLE [REAL]])
  -- Add ‘an_observer’ to observers list.
do
  observers.put (an_observer)
ensure
  present: observers.has (an_observer)
end

unsubscribe (an_observer: PROCEDURE [ANY, TUPLE [REAL]])
  -- Remove ‘an_observer’ from observers list.
do
  observers.prune (an_observer)
ensure
  absent: not observers.has (an_observer)
end

feature {NONE} -- Implementation
observers: LINKED_SET [PROCEDURE [ANY, TUPLE [REAL]]]
  -- Set of observing agents.

invariant
  valid_temperature: valid_temperature (temperature)
observers_exists: observers /= Void
end

Listing 2: Class APPLICATION

class APPLICATION

create
  make

feature {NONE} -- Initialization
make
  -- Run application.
local
  s: TEMPERATURE_SENSOR
d: DISPLAY
c: HEATING_CONTROLLER
do
  create s
  create d
  create c
c.set_goal (21.5)
s.subscribe (agent d.show)
s.subscribe (agent c.adjust)

s.set_temperature (22)
s.set_temperature (22.8)
s.set_temperature (20.0)

s.set_temperature (−273.14276764)
s.set_temperature (1000)
s.set_temperature (0)
end
end

2 Debug me!

Listing 3: Class SORTED_LINKED_LIST
	note
description: "Linked list with internal cursor, where elements are sorted in ascending order."

class
SORTED_LINKED_LIST [G -> COMPARABLE]

feature -- Access
item: G
-- Value at cursor position.
require
  not_off: not off
do
  Result := active.item
end

count: INTEGER
-- Number of elements.

min: G
-- Minimum element.
require
  not_empty: not is_empty
do
  Result := first.item
end

max: G
-- Maximum element.
require
  not_empty: not is_empty
do
  Result := last.item
end
feature -- Status report

is_empty: BOOLEAN
-- Is the list empty?
do
  Result := first = Void
end

off: BOOLEAN
-- Is cursor not at a list element?
do
  Result := active = Void
end

has (v: G): BOOLEAN
-- Does list contain ‘v’?
local
  old_active: LIST_CELL [G]
do
  -- BUG 1: the position of the cursor wasn’t restored at the end of the function
  old_active := active
  search (v)
  Result := not off
  active := old_active
ensure
  not_found_in_empty: is_empty implies not Result
end

feature -- Search

search (v: G)
-- Move cursor to ‘v’ if is present.
-- Otherwise go off.
local
  next: LIST_CELL [G]
do
  search_max_less (v)
  -- BUG 2: ignored possibility of ‘v’ not in the list
  -- BUG 3: ignored possibility of ‘v’ <= ‘min’ (‘active’ = Void)
  if off then
    next := first
  else
    next := active.right
  end
  if next /= Void and then next.item = v then
    active := next
  else
    active := Void
  end
ensure
  found_or_not_found: off or else item = v
end
feature -- Element change
insert (v; G)
   -- Insert 'v' at the proper position in the list.
   local
       new: LIST_CELL [G]
do
   create new.put (v)
search_max_less (v)
   -- BUG 4: ignored possibility of 'v' <= 'min'
if off then
   if is_empty then
      -- BUG 5: forgot to update 'last' if the list was empty
      last := new
   else
      new.put_right (first)
   end
   first := new
else
   if active.right = Void then
      last := new
   else
      new.put_right (active.right)
   end
   active.put_right (new)
   end
   -- BUG 6: forgot to update 'count'
   count := count + 1
ensure
   present: has (v)
one_more: count = old count + 1end
remove (v; G)
   -- Remove one occurrence of 'v'.
require
   present: has (v)
do
   search_max_less (v)
if off then
   first := first.right
   -- BUG 7: forgot to update 'last' when emptying the list
   if first = Void then
      last := Void
   end
else
   check not last: active.right /= Void end
   active.put_right (active.right.right)
   -- BUG 8: forgot to update 'last' when removing the last element
   if active.right = Void then
      last := active
   end
end
\[
\text{count} := \text{count} - 1
\]
\[
\text{ensure}
\]
\[
\text{one_less: count = old count} - 1
\]
\end

\textbf{feature} \{NONE\} \text{--- Implementation}

first: \text{LIST\_CELL} [G]
\text{--- First cell.}

last: \text{LIST\_CELL} [G]
\text{--- Last cell.}

active: \text{LIST\_CELL} [G]
\text{--- Cursor cell.}

\text{search_max_less} (v: G)
\text{--- Move cursor to the maximum value in the list that is less than ‘v’.
}
\text{do}
\text{--- BUG 9 (a): <= instead of < (we want max less, not max less equal!)}
\text{if not is_empty and then first.item < v then}
\text{from}
\text{active := first}
\text{until}
\text{--- BUG 9 (b): > instead of >= (we want max less, not max less equal!)}
\text{active.right = Void or else active.right.item >= v}
\text{loop}
\text{active := active.right}
\text{end}
\text{else}
\text{--- BUG 10: forgot to go off if ‘min’ >= ‘v’}
\text{active := Void}
\text{end}
\text{ensure}
\text{less_than_v: not off implies item < v}
\text{next_greater_equal_v: not off and active.right /= Void implies active.right.item >= v}
\text{first_greater_equal_v: off and not is_empty implies first.item >= v}
\end

\textbf{invariant}

empty: (first = Void) = (last = Void)
last_not_linked: last /= Void implies last.right = Void
rest_is_linked: active /= Void and active /= last implies active.right /= Void
is_empty_definition: is_empty = (first = Void)
off_definition: off = (active = Void)
off_if_empty: is_empty implies off
count_non_negative: count > = 0
count_zero: is_empty = (count = 0)
sorted: active /= Void and active.right /= Void implies active.item <= active.right.item
end
3 The final project: Board game (part 4)

You can download a complete solution from 