Solution 7: More Peachy loops

ETH Zurich

1 Buildings for Paris

Listing 1: Class *CONSTRUCTION*

```
note
  description: "Construction class (Assignment 7)"
  date: "$Date$"
  revision: "$Revision$"

class
  CONSTRUCTION

inherit
  TOURISM

feature -- Explore Paris
  build
      -- Show map and generate buildings along all roads.
    do
      Paris.display
      fill_grid
      generate_buildings
    end

  Size: REAL_64 = 16.0
      -- Building size.

  Distance: REAL_64 = 24.0
      -- Distance from a building to the road or the next building.

  generate_buildings_along_segment (segment: TRAFFIC_ROAD_SEGMENT)
      -- Generate buildings along a segment.
    require
      segment_exists: segment /= Void
    local
      start_point, end_point, step, shift, upper, lower: TRAFFIC_POINT
      count: INTEGER
      i, j: INTEGER
      building: TRAFFIC_VILLA
    do
      from
        i := 1
      until
```

```
        i >= segment.polypoints.count
    loop
      start_point := segment.polypoints.item (i)
      end_point := segment.polypoints.item (i + 1)
      count := (start_point.distance (end_point) / Distance).floor
      step := end_point - start_point
      step.scale_to (Distance)

      shift := step.twin
      shift.rotate_rectangularly
      from
        j := 1
        upper := start_point + shift
        lower := start_point - shift
      until
        j > count
      loop
        if not grid.has_rectangle_collision (upper, Size, Size) then
          create building.make_new (Size, Size, Size, upper)
          Paris.put_building (building)
          grid.mark_rectangle (upper, Size, Size, True)
        end
        if not grid.has_rectangle_collision (lower, Size, Size) then
          create building.make_new (Size, Size, Size, lower)
          Paris.put_building (building)
          grid.mark_rectangle (lower, Size, Size, True)
        end
        j := j + 1
        upper.move_by (step)
        lower.move_by (step)
      end
      i := i + 1
    end
  end

generate_buildings
    -- Generate buildings along all roads.
  local
    r: TRAFFIC_ROAD_SEGMENT
  do
    from
      Paris.roads.start
    until
      Paris.roads.after
    loop
      r := Paris.roads.item_for_iteration.one_way
      generate_buildings_along_segment (r)
      Paris.roads.forth
    end
  end

fill_grid
```

```
      -- Create and fill grid.
    local
      l: TRAFFIC_LINE_SEGMENT
    do
      create grid.make (200, Paris.center, Paris.radius)
      from
        Paris.line_segments.start
      until
        Paris.line_segments.after
      loop
        l := Paris.line_segments.item_for_iteration
        grid.mark_polyline (l.polypoints, 7.0, True)
        Paris.line_segments.forth
      end
    ensure
      grid_exists: grid /= Void
    end

  grid: TRAFFIC_GRID
      -- Grid to detect collisions.
end
```

2 Bagels!

Listing 2: Class *BAGELS*

```
note
  description : "Bagels application"
  date : "$Date: 2008-12-29 15:41:59 -0800 (Mon, 29 Dec 2008) $"
  revision : "$Revision: 76432 $"

class
  BAGELS

create
  make

feature -- Initialization
  make
      -- Play bagels.
    local
      d: INTEGER
    do
      io.put_string ("*** Welcome to Bagels! ***%N")
      from
      until
        io.last_integer > 0
      loop
        io.put_string ("Enter the number of digits (positive):%N")
        io.read_integer
      end
      d := io.last_integer
```

```
      play (d)
    end

feature {NONE} -- Implementation
  answer: STRING
      -- Correct answer.

  play (d: INTEGER)
      -- Generate a number with 'd' digits and let the player guess it.
    require
      d_positive: d > 0
    local
      guess_count: INTEGER
      guess: STRING
    do
      io.put_string ("I'm thinking of a number...")
      generate_answer (d)
      io.put_string (" Okay, got it!%N")

      from
      until
        guess /= Void and then guess.is_equal (answer)
      loop
        io.put_string ("Enter your guess: ")
        io.read_line
        guess := io.last_string
        if guess.count = d and guess.is_natural and not guess.has ('0') then
          print (clue (guess) + "%N")
          guess_count := guess_count + 1
        else
          io.put_string ("Incorrect input: please enter a positive number with " + d.
              out + " digits containing no zeros%N")
        end
      end
      print ("Congratulations! You made it in " + guess_count.out + " guesses.")
    end

  generate_answer (d: INTEGER)
      -- Generate a number with 'd' nonzero digits and store it in 'answer'.
    require
      d_positive: d > 0
    local
      random: RANDOM
      t: TIME
      i: INTEGER
    do
      create answer.make_filled (' ', d)
      create t.make_now
      create random.set_seed (t.milli_second)
      from
        random.start
        i := 1
```

```
    until
      i > d
    loop
      answer [i] := (random.item \\ 9 + 1).out [1]
      random.forth
      i := i + 1
    end
  ensure
    answer_exists: answer /= Void
    correct_length: answer.count = d
    is_natural: answer.is_natural
    no_zeros: not answer.has ('0')
  end

clue (guess: STRING): STRING
    -- Clue for 'guess' with respect to 'answer'.
  require
    answer_exists: answer /= Void
    guess_exists: guess /= Void
    same_length: answer.count = guess.count
  local
    i, k: INTEGER
    a, g: STRING
  do
    Result := ""
    a := answer.twin
    g := guess.twin
    from
      i := 1
    until
      i > a.count
    loop
      if a [i] = g [i] then
        Result := Result + "Fermi "
        a [i] := ' '
        g [i] := ' '
      end
      i := i + 1
    end
    from
      i := 1
    until
      i > a.count
    loop
      if a [i] /= ' ' then
        k := g.index_of (a [i], 1)
        if k > 0 then
          Result := Result + "Pico "
          g [k] := ' '
        end
      end
      i := i + 1
```

```
      end
      if Result.is_empty then
        Result := "Bagels"
      end
    ensure
      result_exists: Result /= Void
    end
end
```

3 Decimal to binary converter

Listing 3: Class *DECIMAL_TO_BINARY_CONVERTER*

```
note
  description : "converter application root class"
  date : "$Date: 2008-12-29 15:41:59 -0800 (Mon, 29 Dec 2008) $"
  revision : "$Revision: 76432 $"

class
  DECIMAL_TO_BINARY_CONVERTER

create
  make

feature {NONE} -- Initialization
  make
      -- Run application.
    do
      read_input
      if not read_failed then
        write_output (to_binary (input))
      else
        print ("Reading input from " + Input_file_name + " failed")
      end
    end

feature -- Conversion
  valid_input (n: INTEGER): BOOLEAN
      -- Is 'n' a valid input to conversion?
    do
      Result := 0 <= n and n <= 100000000
    end

  to_binary (n: INTEGER): STRING
      -- Binary reresentation of 'n'.
    require
      valid_input: valid_input (n)
    local
      temp: INTEGER
    do
      if n = 0 then
        Result := "0"
```

```
      else
        from
          Result := ""
          temp := n
        invariant
          temp = n // (2 ^ Result.count).truncated_to_integer
        until
          temp = 0
        loop
          if temp \\ 2 = 0 then
            Result.precede ('0')
          else
            Result.precede ('1')
          end
          temp := temp // 2
        variant
          temp + 1
        end
      end
    ensure
      result_exists: result /= Void and then not Result.is_empty
    end

feature -- I/O
  read_input
      -- Read an integer from a file named 'Input_file_name'.
      -- In case of failure set 'read_failed'.
      -- In case of success unset 'read_failed' and store result in 'input'.
    local
      input_file: PLAIN_TEXT_FILE
    do
      read_failed := True
      create input_file.make (Input_file_name)
      if input_file.exists then
        input_file.open_read
        if input_file.file_readable then
          input_file.read_line
          if input_file.last_string.is_integer then
            input := input_file.last_string.to_integer
            if valid_input (input) then
              read_failed := False
            end
          end
        end
        input_file.close
      end
    ensure
      valid_input_if_success: not read_failed implies valid_input (input)
    end

  write_output (s: STRING)
      -- Write 's' to a file named 'Output_file_name'.
```

```
    local
      output_file: PLAIN_TEXT_FILE
    do
      create output_file.make_open_write (Output_file_name)
      output_file.put_string (s)
      output_file.close
    end

  Input_file_name: STRING = "converter.in"
      -- Input file name.

  Output_file_name: STRING = "converter.out"
      -- Output file name.

  read_failed: BOOLEAN
      -- Did last input file read operation fail?

  input: INTEGER
      -- Last integer read by a succesful input file read operation.
end
```