

Assignment 7: More loops

ETH Zurich

Hand-out: 7 November 2008
Due: 13 November 2008

Goals

- Use loops and conditionals to solve tasks.
- Use nested loops.

1 Buildings for Paris

In this task, you will generate buildings for Paris that are placed along its roads. *TRAFFIC_ROAD* represents streets, railtracks, or lightrail tracks that public transportation lines use as their medium to operate on. A *TRAFFIC_ROAD* can be a one-way road in which case it only contains one *TRAFFIC_ROAD_SEGMENT* (accessible through the feature *one_way*) or a two-way road in which case there are two *TRAFFIC_ROAD_SEGMENTS* (feature *one_way* and feature *other_way*). A *TRAFFIC_ROAD_SEGMENT* connects an *origin* (starting station) with a *destination* (end station) and contains a set of points defining the shape of the segment. All two-way roads of the Paris map have the same set of points for the first and the second segment (but in inverted order). The points are accessible through the feature *polypoints* in *TRAFFIC_ROAD_SEGMENT* (inherited from *TRAFFIC_SEGMENT*). Figure 1 illustrates this. To iterate through the polypoints of a road segment, you can use the following scheme:

```
local
  road: TRAFFIC_ROAD
  seg: TRAFFIC_ROAD_SEGMENT
  coord: TRAFFIC_POINT
  i: INTEGER
do
  road := Paris.roads.item (1) -- Access the first road in Paris
  seg := road.one_way -- Access the first segment of the road
from
  i := 1
until
  i > seg.polypoints.count
loop
  coord := seg.polypoints.item (i)
  -- Do something with coord
  i := i + 1
end
end
```

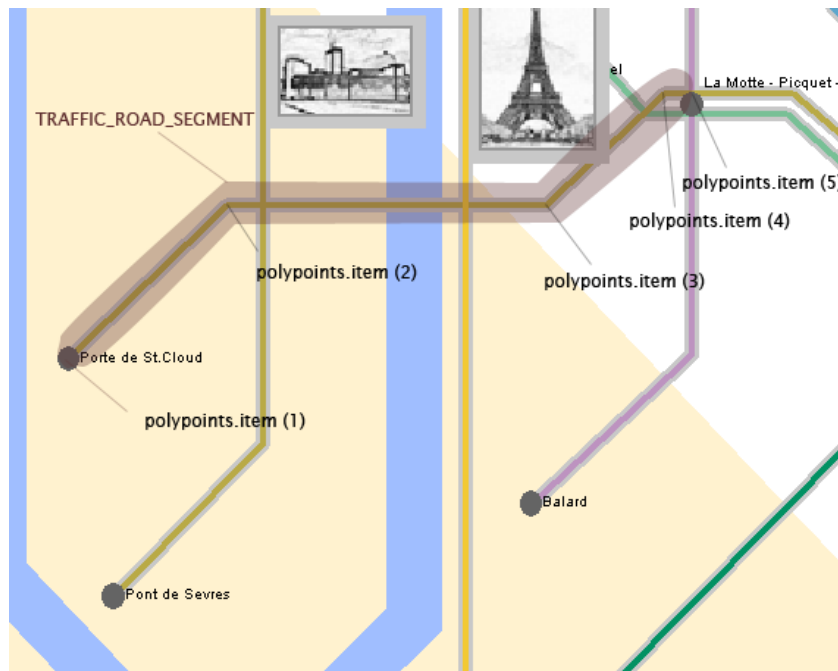


Figure 1: Example of a `TRAFFIC_SEGMENT` and its `polypoints`.

To do

1. Download http://se.ethz.ch/teaching/2008-H/eprog-0001/exercises/assignment_7.zip and extract it in `traffic/example`. You should now have a new directory `traffic/example/assignment_7` with `assignment_7.ecf` directly in it (it is important that the location corresponds to the description here!).
2. Open and compile this new project. Open class `CONSTRUCTION`.
3. Implement feature `generate_buildings_along_segment`. It should generate buildings of type `TRAFFIC_VILLA` along the road segment given as argument. The result should be two polylines of buildings parallel to the polyline of the road segment: one above it with distance 24.0 and one below also with distance 24.0. The distance between two buildings next to each other on a row should also be 24.0. Figure 2 shows a schematic image of it. Note that each consecutive point pair `polypoints.item (i)` and `polypoints.item (i+1)` defines a straight line. The first two buildings for each point pair are placed on the perpendicular axis to this straight line going through `polypoint.item (i)`. The number of buildings for each point pair and row is the rounded down number of buildings that would fit on the line defined by the point pair. Test your implementation of `generate_buildings_along_segment` by calling it from feature `build` with an object of type `TRAFFIC_ROAD_SEGMENT` as argument (see the code above for hints on how to access a road segment). Don't forget to add the generated buildings to `Paris`.
4. Implement feature `generate_buildings` that does the same for all roads in Paris. Call `generate_buildings` in feature `build`.
5. **Optional:** The algorithm you implemented in Step 4 will put buildings on top of existing buildings and public transportation lines. To prevent this you can use a `TRAFFIC_GRID`.

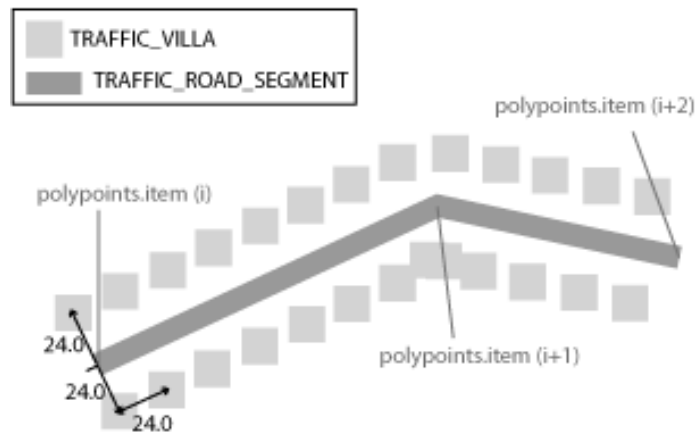


Figure 2: Buildings along a road segment

The grid needs to be the same size as Paris (using its center and radius) and contains $n * n$ fields that can be marked as occupied. The feature `fill_grid` of class `CONSTRUCTION` creates and initializes such a grid for you and you can use it to create and add a building to Paris only if it doesn't collide with an existing item. See the comment at the end of feature `generate_buildings_along_lines` for an explanation on how to do this.

Hints

Again, you will need to do vector calculations that are available from class `TRAFFIC_POINT`. For a short description of this class, have a look at assignment 6. Additionally, you might find the feature `normalize` of `TRAFFIC_POINT` useful to get a vector of length 1.

To hand in

Hand in the code of class `CONSTRUCTION`.

2 Loop exercise

To do

You are to write a program that plays a game known as Bagels. It is a variation of Mastermind that uses the digits 1 to 9 (no zeros).

In Bagels, the program generates an n -digit number that the user will try to guess. The user is asked at game startup with what value of "n" she or he wishes to work.

After each guess, the application gives some clues as to how close the user is to getting the answer right. If there is a right digit in the right position, it should say FERMI. If there is a right digit but in the wrong place, it will say PICA. For example, if the number is 123 and the guess 329, it will report FERMI PICA. If the number is 123 and the guess 312, it will report PICA PICA PICA. Each digit will match only once. If the answer is 999 and the guess 912, it will report FERMI. If the answer is 912 and the guess 999, it will also report FERMI. Notice that it might report several FERMIs or PICAs in one feedback message. If the answer is 222 and the guess is 262, it should say FERMI FERMI (not FERMI PICA). If the answer is 11234511 and guess 19111191 the result should be FERMI FERMI PICA PICA (because two of the four searched 1s are FERMI and another two are present, but at the wrong position).

The application should list all of the FERMIs first and then all of the PICAs. If the answer is 12345 and the guess 31245, it will say FERMI FERMI PICA PICA PICA. Because of this, the user can't assume anything from the order of the clues.

The application should require the user to give exactly "n" digits per guess.

Hints

The class *STRING* provides many useful features for string manipulation and we encourage you to store the number to guess as an object of type *STRING* instead of as an *INTEGER*. To generate a random answer, you will have to use the class *RANDOM* again. It might be the case that the class *TIME* is not available in your project. To add it to your project, open the project settings dialog from the menu (**Project** ▸ **Project settings**). Click on the tree item **Groups** and use the keyboard shortcut **Ctrl+L**. In the list of default libraries, choose **time** and click **OK**. Recompile your project and you will be able to use *TIME*.

To hand in

Hand in your class text.

3 BNF-E in BNF-E

BNF-E is a formal language which can itself be described by a grammar in BNF-E.

To do

Describe the syntax of the BNF-E language with a grammar in BNF-E. You can use the following production as a starting point:

```
BNF_E_Grammar ::= {Production "%N" ...}+  
Production ::= .....  
.....  
.....  
.....
```

To hand in

Hand in your BNF-E grammar.