Project Functional Programming: Part 2
Academic Year 2012-2013

December 13, 2012

1 Introduction

The second assignment has to be submitted before 7 January 2013 11:59 pm. Submission is done by sending your raw code files to Laurent Christophe before the deadline: `lachrist@ulb.ac.be`.

The goal of this assignment is to *simulate* a Pac-Man-like[1] game. However, instead of moving on a fixed rectangular grid, the Pac-Man and the ghosts move about in an irregularly shaped graph whose nodes correspond to locations and whose edges correspond to "tunnels" that allow Pac-Man and the ghosts to move from one location to another.

More precisely, you will have to implement a strategy to helps the ghosts catching Pac-Man while he is trying to move from a given source location to a given target location. Edge labels are positive integers and represent the units of time that are necessary for a ghost or the Pac-Man to traverse the corresponding tunnel/edge. A ghost catches Pac-Man when it is sitting on the same location/node as Pac-Man at the same moment.

2 Parsing

First of all you will have to parse the initial game configuration from a file that is specified using a subset of the DOT language. That subset is specified using the grammar shown in (Table 1). E.g., the grammar contains the production $STMTS \rightarrow STMT\ STMTS$.

Apart from the game board itself, the input file contains information like the initial position of Pac-Man and the ghosts along with the target location of Pac-Man. Besides parse errors, your program should detect the following file format errors as well:

- There is more than one source location for Pac-Man.
- There is more than one target location for Pac-Man.
- Undefined node references.

3 The Ghost's behavior

Implementing the optimal strategy to catch Pac-Man is a difficult problem. Therefore we propose the following heuristic used by the ghosts:

The global goal of the strategy is to block most of the paths Pac-Man can take to reach its target. To reduce the amount of possible paths, we consider that Pac-Man never stops and that he can not pass through a node more than once (i.e. he doesn't walk in cycles). From a given position, each ghost will move such that it will block a Pac-Man-path within a minimal time cost. When a ghost is unable to block any path, it stops moving. If multiple moves share the same minimal time cost, you should just pick one. However, you have to take into account the case where a motion blocks more than a single Pac-Man-path.

[1] `http://en.wikipedia.org/wiki/Pac-Man`

Variable	Expression	Remark
GRAPH	`graph TOKEN { STMTS }`	A directed graph is defined by a name and a sequence of statements.
STMT	`NODE ;` `EDGE ;`	A statement may either describe a node or an edge.
NODE	`TOKEN [ ATTR ATTRS ]` `TOKEN`	A node is an identifier with optional attributes.

ATTR	`source = BOOL`
	
	`target = BOOL`
	
	`ghost = NATURAL`
		Indicates if the node is the initial location of Pac-Man.
		Indicates if the node is the target location of Pac-Man.
		Indicates the amount of ghosts initially located at the node.

BOOL	`true` `false`	A boolean is either true or false ; dummy remark is dummy.
EDGE	`TOKEN CHAIN [ value = NATURAL ]`	An edge is defined by its path (sequence of node) and its value.
CHAIN	`-- TOKEN CHAIN` ϵ	A sequence of node references separated by lines.
STMTS	`STMT STMTS` ϵ	A sequence of statements.
ATTRS	`ATTR , ATTRS` ϵ	A sequence of attributes separated by comas.

Table 1: Detailed description of the import DOT format. `NATURAL` is any positive integers ; `TOKEN` is any words exclusively compound of alphabetic characters and digits. If you have any problem understanding this language, please refer to `http://en.wikipedia.org/wiki/DOT_language` or `http://www.graphviz.org/content/dot-language`

.

As the above strategy has a high computational cost, each ghost should be instantiated by a dedicated thread. To avoid the case where two ghosts aim to block the same path, the threads will have to communicate. You can use the concurrency technology taught in the course, e.g. *Software Transactional Memory*.

4 Pac-Man's behavior

As said, Pac-Man never stops (constant speed) and he may pass through a node only once (i.e. no cycles). Pac-Man knows which paths are blocked by the ghosts and will automatically pick a safe path if available. If Pac-Man cannot reach its target location safely, you can just let Pac-Man sit at its initial position.

5 Exporting the simulation

A simulation is defined by a sequence of positions of the Pac-Man and the ghosts on the given graph. The time separating the simulation steps corresponds to one time unit as defined in the introduction.

To test your strategy, you will have to export a simulation into a JSON format specified in (Table 2). The nodes should be placed using the force-oriented layout algorithm developed for the first assignment. A JavaScript program will be provided to render your output files.

Variable	Expression	Remark
SIM	{"graph":GRAPH,"target":NATURAL,"steps":[STEP STEPS]}	The complete simulation ; "target" is the index of the node that Pac-Man should reach.
GRAPH	{"nodes":[NODE NODES],"edges":[EDGE EDGES]}	The graph which is a constant of the simulation. The nodes and the edges, will be referred to using their index inside their respective list (indexes start at zero).
NODE	{"x":FLOAT,"y":FLOAT}	A node that is represented by its Cartesian coordinates (obtained by the force-oriented layout).
EDGE	{"from":NATURAL,"to":NATURAL,"value":NATURAL}	"from" is the index of the node which is the starting point of the edge ; "to" is the index of the node which is the end point of the edge ; "value" is the weight of the edge.
STEP	{"pacman":POS,"ghosts":[POS POSS]}	A step of the simulation that is represented by the positions of Pac-Man and the ghosts.
POS	{"inside":NATURAL,"at":NATURAL}	"inside" is an edge index, "at" is the position of the object on the edge (starting from the source of the edge).
NODES	, NODE NODES ϵ	A sequence of nodes separated by comas.
EDGES	, EDGE EDGES ϵ	A sequence of edges separated by comas.
STEPS	, STEP STEPS ϵ	A sequence of steps separated by comas.
POSS	, POS POSS ϵ	A sequence of positions separated by comas.

Table 2: Detailed description of the export JSON format. NATURAL is any positive integers, FLOAT is any positive floating-point numbers. If you have problems understanding this language, please refer to http://en.wikipedia.org/wiki/JSON.

6 A complete example

In this section we illustrate a possible execution of the presented assignment:

```
>> ./pacman input.dot output.json
```

The input is detailed in Figure 1; the output (text) is detailed in Figure 3. Moreover the output is (graphically) detailed in Figure 2. As said, a program will be provided for you for rendering the textual JSON output.

```
graph pacman {
    a [source=true];
    b1;
    b2;
    c [ghost=1];
    d1;
    d2;
    e1;
    e2;
    f [target=true];
    g [ghost=1];
    a -- b1 [value=4];
    b1 -- c [value=1];
    c -- d1 -- e1 -- f [value=3];
    a -- b2 -- c -- d2 -- e2 -- g [value=3];
    e2 -- f [value=10];
}
```

Figure 1: Contents of the file `input.dot`. This represents the input which is the initial configuration of the simulation.

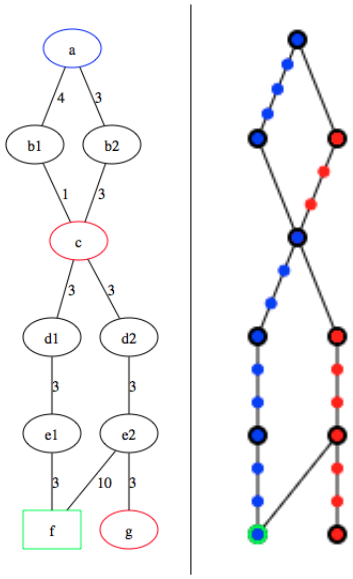

Figure 2: On the left, a graphical representation of the `input.dot` file presented in Figure 1; on the left a graphical representation of the output simulation presented in Figure 3. The path of Pac-Man is indicated in blue, the ghost's paths are indicated in read and the target is shown in green. This case shows that the heuristic is not optimal.

```
{"graph":{"nodes":[{"x":100, "y":50},
                   {"x":80, "y":100},
                   {"x":120, "y":100},
                   {"x":100, "y":150},
                   {"x":80, "y":200},
                   {"x":120, "y":200},
                   {"x":80, "y":250},
                   {"x":120, "y":250},
                   {"x":80, "y":300},
                   {"x":80, "y":300}],
          "edges":[{"from":0, "to":1, "value":4},
                   {"from":1, "to":3, "value":1},
                   {"from":3, "to":4, "value":3},
                   {"from":4, "to":6, "value":3},
                   {"from":6, "to":8, "value":3},
                   {"from":0, "to":2, "value":3},
                   {"from":2, "to":3, "value":3},
                   {"from":3, "to":5, "value":3},
                   {"from":5, "to":7, "value":3},
                   {"from":7, "to":8, "value":10},
                   {"from":9, "to":7, "value":3}]},
 "target":8,
 "steps":[
 {"pacman":{"inside":0, "at":0}, "ghosts":[{"inside":6, "at":3}, {"inside":10, "at":0}]},
 {"pacman":{"inside":0, "at":1}, "ghosts":[{"inside":6, "at":2}, {"inside":10, "at":1}]},
 {"pacman":{"inside":0, "at":2}, "ghosts":[{"inside":6, "at":1}, {"inside":10, "at":2}]},
 {"pacman":{"inside":0, "at":3}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":3}]},
 {"pacman":{"inside":1, "at":0}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":2}]},
 {"pacman":{"inside":2, "at":0}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":1}]},
 {"pacman":{"inside":2, "at":1}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":0}]},
 {"pacman":{"inside":2, "at":2}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":0}]},
 {"pacman":{"inside":3, "at":0}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":0}]},
 {"pacman":{"inside":3, "at":1}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":0}]},
 {"pacman":{"inside":3, "at":2}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":0}]},
 {"pacman":{"inside":4, "at":0}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":0}]},
 {"pacman":{"inside":4, "at":1}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":0}]},
 {"pacman":{"inside":4, "at":2}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":0}]},
 {"pacman":{"inside":4, "at":3}, "ghosts":[{"inside":6, "at":0}, {"inside":8, "at":0}]}]
}
```


Figure 3: Contents of the file `output.json`. This represents the output of your program which is the result of your strategy to catch the Pac-Man.