Pac-man

INGI1131 Course project (2016)

April 25, 2016

Zhongmiao Li
(*zhongmiao.li@uclouvain.be*)

Manuel Bravo
(*angel.bravo@uclouvain.be*)

This year's project consists of creating a version of the popular pac-man game. Of course, we will do this by steps, starting from a very simple version, and adding features and playability that will give you extra points!. Let's get into details:

1. The simulator

You will have to model the following components:

- The room.
- Pac-man that tries to eat all coins.
- The ghosts that wander around the room trying to eat pac-man.

A simulator has two inputs: the setup of the room and the number of lives of pac-man.

The simulation finishes whenever pac-man has eaten all coins or pac-man has lost all his lives against ghosts.

1.1. The room

The room is represented by a matrix. Each cell of the matrix contains an integer that represents what would be in the real room. The possible initial values for a cell are the following:

- 0: Empty space, which by default contains a coin
- 1: Wall
- 2: Power pellet
- 3: A ghost
- 4: The pac-man
- 5: Worm-hole end-points

An example of a valid room configuration is as follows:

```
map( r(1 1 1 1 1 1 1 1 1 1 1 5 1 1 1 1 1 1 1 1)
     r(1 3 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 3 1)
     r(1 0 1 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0 1)
     r(1 0 1 0 2 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1)
     r(5 0 1 0 1 0 1 1 0 4 0 0 1 1 0 1 0 1 0 1)
     r(1 0 1 0 1 0 0 0 0 1 1 0 1 1 0 2 0 1 0 5)
     r(1 0 1 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0 1)
     r(1 0 1 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0 1)
     r(1 3 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 3 1)
     r(1 1 1 1 1 1 1 1 5 1 1 1 1 1 1 1 1 1 1 1)
```

Similar to the original pac-man game, there are two restrictions for the configuration of a room:

(1) Each aisle (a series of continuous empty cells) can only have width of one.
(2) There should not be any inaccessible empty space. Otherwise, pac-man can never win by eating all coins.

Please, do not confuse a cell of the matrix with a cell in Oz. A matrix cell is only an element of the matrix. Remember, you are not allowed to use Oz cells.

1.2. Pac-man

Pac-man's goal is to eat all the coins placed in the room without being killed. Pac-man will be killed if he touches or is touched by a ghost. However, after eating power pellets, our pac-man can kill any ghost he encounters for a limited period of time.
After being killed, if pac-man still has extra lives, he will be respawned in his original position.

1.3. Ghosts

Ghosts are dumb, they move randomly trying to chase Pac-man. Nevertheless, after pac-man has taken a power pellet, ghosts are turned into 'scared' mode for a period of time, during which, they can be killed by pac-man.
After being killed, ghosts are respawned either in their original positions, or in a random position. A respawned ghost will not be in 'scared' mode anymore, even if they were.

1.4. Wormholes

A wormhole can have between 2 and 5 end-points. They are all connected. If pac-man or a ghost goes inside a wormhole, he gets out from one of the other end-points (chosen randomly). If a ghost or pac-man passes through a wormhole, his walking direction should be correctly adjusted so he will not be bumped between wormholes.

1.5. Interaction between components

A cell can be occupied by multiple characters. If they are all ghosts, then nothing will happen. But if pac-man encounters a ghost, then depending on their status (i.e. a ghost is in scared mode or not), either pac-man dies or the ghost dies. More complicated, if pac-man encounters a scared ghost and a normal ghost at the same time, sad story will happen: pac-man gets killed.

1.6. AI and control

Regarding how the game evolves, we are gonna do this by steps:

First, you have to build a simple version that works by rounds. We manually control pac-man either with WASD keys or arrow keys. Every time a key is pushed, a new round starts, pac-man is moved in the direction indicated by the pushed key (if possible), ghosts will move automatically (decided by the application). Ghosts are pretty dumb: they usually move in a straight line, but when they see a crossing or reach a wall, they randomly choose a direction to go. Of course, they should have less probability to choose to go backward, because that won't probably lead them to catch pac-man. If you want, you can try to make them smarter, but no need to pay much time on it!

In order to make it slightly more playable, you can move pac-man one room-cells per round and the ghosts two per round.

For this first version you can only use the declarative model abstractions extended with ports. It is not allowed to use things like NewCell, IsDet, IsFree, *classes*, or whatever language abstraction out of the declarative model and ports.

Second, let's try to make funnier! Let's make it real time. For this second version (which is actually an extension of the first), the goal is to make ghosts change their position automatically every short period of time (some milli-seconds), without waiting for pac-man to move. Try to find a frequency that makes the game playable and generates smooth transitions.

For pac-man, we should still be able to control manually it. In addition, pac-man keeps moving in the same direction unless we change the direction or the end of the corridor is reached (as in the original game). For the autonomous moves, the frequency should be slightly larger than the one for the ghosts.

For this second version, you are allowed to use the primitive isDet.

2. Project description

Your project consists of modeling and implementing a complete simulation for the description above. You have to provide a graphical user interface that allows to appreciate the interaction between the different components.

2.1. Deliverables

The project has to be done in groups of two. We strongly discourage you to work alone because of the load of the project, and also because working with somebody else will help you to see things from a different point of view, and, it will develop your social skills too.

The project must be submitted on Moodle. The submission must contain as attachment an archive file in zip or tgz format whose name is lastname1-lastname2.(zip|tgz). Of course, replace lastname with your last name.

This archive must contain the following items:

• Full source code of your program. You have to submit a functor (or a set of functors, but there should be a main one that internally loads all other functors) containing the program. The program receives certain parameters, including the map of the stage, and the number of lives.

• A README file telling some important and concise information, for example which is the main functor to execute.

• A small report of about 2 pages, explaining the architecture and design of your project, along with the concurrency issues you faced and how you solved them. The full names of the authors must appear on the first page of the report. The report must be in .pdf format. We will not accept reports in .doc or .odt formats.

Submission deadline: Friday, May 13th at 23:58

2.3. Resources

With the project description, we also provide a super simplified version of the project, which it will be a nice starting point for you. You can build your project based on the template.

Remember that you also have several resources available to get some advice during the time for the project. You have the forum on Moodle (http://www.icampus.ucl.ac.be/), ask by mail to the assistants of the course and during the lab sessions. We will also give a small sample code for QTk, and a functor template to get the arguments of the program. We will also provide some files with maps for you to test. QTk will be used for the graphical user interface (GUI). This interface is quite important because it allows us to judge and grade your project. For more information about QTK, you can refer to http://mozart.github.io/mozart-v1/doc-1.4.0/mozart-stdlib/wp/qtk/html/index.html. We will give more documentation on QTk and a sample code. For information about functors, you can refer to here:
http://www.ps.uni-saarland.de/~niehren/index.html/vorlesung/node106.html

We suggest you to use the Pickle format for the map (http://mozart.github.io/mozart-v1/doc-1.4.0/apptut/node5.html#chapter.pickle
http://mozart.github.io/mozart-v1/doc-1.4.0/system/node57.html#chapter.pickle). We provide a simple example file Pickling.oz to show how to write to a Pickle file and load from a Pickle file.

We encourage you to develop the project by yourselves. Do not copy code from other students, we use tools to compare the source code of projects (even if code is restructured and variables names are changed). If we detect that two projects are suspiciously similar, we will failed both groups.

Of course, we also encourage you to talk to other people and discuss ideas, but do not copy the code. If you do, you won't learn that much, and you won't deserve your points.

2.4 Grading

Making a playable version working by rounds will already give you 14 points. Building the real-time version upon the round-based will give you 3 more points. The rest will be based on how playable the game is, how well structured the code is, and the report.