Obroni Computer Club – A New Thing: Infon

Joachim Breitner

November 14th, 2006

We have been doing python for eight sessions now, you know a lot about it, and from here you can discover it on your own, there is enough material online. If you have problems you can still ask me for help, of course.

Today, I want to introduce a networked sife simulation environment (one could call it a game, but that doesn't sound very good at this school, I'm afraid) named "infon". On a small mazy world you control little bugs who have to eat, try not to be eaten and maybe even eat the other bugs.

The interesting thing is that all your bugs live on the same server, thus competing, and you can't contol them directly. Instead, you program the "brain" of the bugs, using a very simple programming language called "lua".

1 Viewing the game

A game of infon is running our server, `occ.soshgic.edu.gh`. To view what's going on, you can run the graphical client in `K:\Others\OCC\infon` by double-clicking on the batch file. You will see the world, maybe already a few bugs. You can grab the world with the mouse, and change the resolution with the keys 1 through 5.

2 Joining the game

To take part in the action and control the bug, you can connect to the server using telnet. Luckily, the putty program speaks telnet, so run `K:\Others\OCC\putty.exe` and connect to `occ`, using protocol "telnet" and port 1234.

You will reach a prompt where you can enter commands. You get help with "?" and you can see the current scores with "s". To join the game, enter "j". If you are re-connecting and want to continue to use your old bugs, you can now select it. Otherwise create a new player and enter a password, so you can later re-join. If you disconnect you have two minutes to re-join with your password before the player will be removed.

If you are watching the game you can see two new bugs, your bugs. You should give your player a better name than "player0": "?" tells you that the command to change the player's name is "n", so enter that and then type in a new name. You can also change the colour.

3 Controlling your bugs

Currently, your bugs sits around and do nothing but starve, which is unfortunate, because every time a bug starves you lose three points. You can actually watch them stave: There are two bars above their heads: The upper one shows how much food reserves they have (and they don't have any when the spawn) and the lower one shows the health, which is decreasing if they have no food. When this bar reaches zero, the bug dies. So let's look for food, which you can see as green stuff on the map.

First, we have to get to the food, so let's see how we can control the bug. For that, we should start by opening the code that currrently runs on the bugs. You can find it in `K:\Others\OCC\infon\player-default.txt`. As you can see, there is not much intelligence in the bug, and in the `Creature:main()` part, where the main work should be done, nothing is happening. First save the file somewhere on your H-drive so that you can make changes to it.

Let's change the behaviour a little bit. Change the last function to:

```
function Creature:main()
   self:screen_message("Hello")
end
```

To load that code on the server, you run the "b" command in your putty session, and then paste the code into the window (select it in the editor, copy, rightclick into the putty window). To finish the program entry, enter a single dot on one line. You should be back at the prompt, and your bug should suddenly say "hello".

Well, this does not bring us closer to food. So let's try to move. And because we don't know where to move, we move randomly. So look at the following code:

```
function Creature:main ()
   local x1, y1, x2, y2 = world_size()
   while not self:set_path(math.random(x1, x2), math.random(y1, y2)) do
   end
   self:begin_walk_path()
   while self:is_walking() do
      self:wait_for_next_round()
   end
end
```

To run this code, just paste it again using the "b" command. It will overwrite the old main function.

This code first get's the coordinates of the bug's world, and using `math.random`, it selects a random point in this world. This point might not be reachable (e.g. because it is a wall), and in this case the `set_path` function returns `false`. Using a `while not` function we just keep trying until we get a valid point.

If we have such a point, we tell the bug to start walking. While we are still walking, we wait for the next round. This means: We keep doing what we are doing right now for one round (which is 1/10th of a second)

Now let's try to eat something. We can only eat what we stand on, and we can find out how much food there is to take using `self:tile_food()`. So if we are walking somewhere and on our way we find some food, we might as well eat that:

```
function Creature:main()
   local x1, y1, x2, y2 = world_size()
   while not self:set_path(math.random(x1, x2), math.random(y1, y2)) do
   end
   self:begin_walk_path()
   while self:is_walking() do
      if self:tile_food() > 0 then
         self:eat()
         self:begin_walk_path()
      end
      self:wait_for_next_round()
   end
end
```

The but is still not very smart: It could search for food in the vincinity of other food. Additionally, it will lose health and it should heal itself is possible. Try it yourself!

4 Debugging

It might happen that although your program looks fine, your bug just sits there and does nothing. Or it does something, but for some reason not what you want. Then it is important to find out whats happening.

To find out about programming errors, for example typos in the method names, use the “i” command. If there was an error in the code of one of your bugs, you will find it there.

Otherwise it helps to put in `p("Some message")` commands in your code. This will print the message to putty, and you can easily follow which code is doing what and when.

5 A note on lua syntax

Lua is such a simple language that there should be no need for explanation for python-cracks like you. But a few hints:

- Blocks like `while ... do`, `function` or `if ... then` have to be ended by `end`.
- Dependening on what you want to with variables, you should create them in one of these ways:
  1. If you want it to be used only in the current function, put a `local` in front of it when you use it the first time.
  2. If the variable should be shared by all your bugs, just use it without anything special.
  3. If it should be separate for any of your bugs, but available in all functions, then name it `self.name_of_variable`.

For mor information on lua, see `http://www.lua,org/`.

6 Infon Reference

From here on, you can explore the program by yourself. Here is a reference of some of the functions you might want to use. For more information, read `http://infon.dividuum.de/`

I will only cover eating and movement. If you want to find out more about attacking and evolving, read the website. We might cover it next week.

6.1 Blocking actions

These actions are called "blocking" beacause once you start it, the bug will happily keep doing it until it is done, and (almost) nothing will stop it.

`self:moveto(x,y)` will start moving the bug to the specified coordinate, and not return until we are there. If that is not possible, for example, because we would walk onto a wall, the function will immediatelly return, returning the value "false".

`self:heal()` will start healing the bug. That means that the health bar is increasing, but the food bar is decreasing. It will run until it can't heal any more (either fully recovered or out of food).

`self:eat()` will eat all the food on the current tile, and won't return until it is all eaten or the bug can't eat any more.

6.2 Non-blocking actions

You can also program your bug non-blocking, which is a bit harder, but more powerful. Non-blocking means that you tell the bug what to do next, but it won't be doing it until you call `self:wait_for_next_round()`. This allows you to change your mind while doing something, for example, starting to eat while going somewhere else.

For every possible action there is a "begin"-function that sets your bug to do something, and an "is"-function to check if he is (still) doing this thing. They are:

- `self:begin_idling()` and `self:is_idle()`
- `self:begin_walk_path()` and `self:is_waling()`
- `self:begin_healing()` and `self:is_healing()`
- `self:begin_eating()` and `self:is_eating()`

`self:screen_message("Message")` will print the message next to your bug on the screen, good to scare other bugs or see what your bug is doing.

`self:set_path(x,y)` sets the path for the `self:begin_walk_path()` function.

`p("message")` prints the message in your putty window, useful for debuggin.

`self:wait_for_next_round()` will, obviously, wait for one round, and actually give the bug time to do anything.

6.3 Getting information

To make your decisions, you have to find out things about your environment.

`self:pos()` returns your current coordinates (e.g. `x,y = self:pos()`)

`self:speed()` returns your maximum speed in coordinate units per second.

`self:health()` returns your current health level (0-100)

`self:food()` returns your current food store level.

`self:max_food()` returns your maximum food store level.

`self:tile_food()` returns the amount of food on the current tile.

7 Outlook

The bugs can now run around and eat stuff, which is nice. But with Infon, there is more to explore:

- The “King Of The Hill”-tile (KOTH) in the middle: Whoever sits on it for a while get’s points for that. But beware, you might be easy game for attacking bugs.
- There are three kind of bugs, type 0, which can only eat and eventually convert to type 1 or type 2 (who then could convert back if they want).
- Type 1 bugs can actually attack and kill other bugs, and make more points in doing so.
- Type 2 can fly over walls and water.
- You can feed (give food to) your other bugs, for example to feed one who has conquered the KOTH.
- Events: You can define functions to react on certains things like being attacked or spawning.

I will keep the server running (at least until the next power outage, please remind me if I forget to restart it), and I hope that by next week some smart bugs are running around!