Obroni Computer Club – More Infon

Joachim Breitner

December 5th 2006

Welcome to our last session. Next week is the last week of the semester, and no club sessions will be held.

Last session, we started playing with "infon", and created bots that run around and eat. This time, we want to push it further and explore more of the things you can do with infon, such as converting your bug to a stronger bug, spawning new bugs and attacking other bugs.

Don't miss the last section with some final words.

1 Remember?

I recommend you also open last sessoin's document, for reference. Here is a quick summery on how to get started:

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.

To take part in the action and control the bug, you can connect to the server using telnet. Run `K:\Others\OCC\putty.exe` and connect to `occ`, using protocol "telnet" and port 1234. To join the game, enter "j". You can then change your name with the "n" command.

If you have written your bugs code, upload that to the server, by running the "b" command in your putty session, and then pasting 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.

2 More kinds of bugs

Currently, the infon worlds knows three kind of bugs:

2.1 Small Bug (type 0)

This is the bug you start with. It has medium speed (at least when healthy). It can only attack the flying bug, and it can convert to either the small bug or the flying bug.

2.2 Large Bug (type 1)

If you upgrade a small bug to type 1, it becomes the large bug. It is slower, but more robust and can attack any other kind of bug. It is the only bug that can spawn new bugs, and it can only convert back to the small bug.

2.3 Flying Bug (type 2)

If you upgrade a small bug to type 2, it becomes the flying bug. It is the fastest, can fly over walls and water, but is weak when attackt and can not attack. It can only convert back to the small bug.

3 Converting

Converting costs food and energy, so you only want to convert healthy bugs. The code fragment could look like this:

```
if not (self:type() == 1) and self:food() > 0.6 * self:max_food()
                      and self:health() > 60 then
   self:set_screen_messsage('morph')
   self:convert(1)
end
```

You can find out your current type using the `self:type()` function. If you want to have the possibility to abort the conversion, for example because you are being attacked, you might want to use the non-blocking functions `self:set_conversion(type)`, `self:begin_converting()` and `is_converting()`

4 Spawning

It would be boring to have only two bugs all the time. Luckily, we can spawn new bugs if we have a healthy bug. A sample code might be:

```
if self:type() == 1 and self:food() > 0.8 * self:max_food()
                and self:health() > 80 then
  self:screen_message('daddy')
  self:begin_spawning()
  while self:is_spawning() do
    self:wait_for_next_round()
  end
end
```

5 Attacking

Once you have a large bug, you can find and attack other bugs. The idea is to use the function `self:nearest_enemy()` to locate the enemy, then try to catch it and start to

attack it. I won't give you any example code, but the functions are documented in the reference, so nothing should stop you from raging war against the other bugs!

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.

`self:convert(type)` will convert to the given type. Make sure you have plenty of food and health before trying that, as otherwise the conversion will fail and you will lose the invested food!

`self:attack(enemy)` will attack the given target (which is a player id returned by `self:nearest_enemy()`) as long as it can. It stops either when the bugs is dead or walked away.

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:begin_converting()` and `self:is_converting()`
- `self:begin_attacking()` and `self:is_attacking()`
- `self:begin_feeding()` and `self:is_feeding()`

Feeding means giving food to a another bug of you, which of course must be close to yours. Both attacking and freeding require that you set the target of the action first, using `self:set_target(id)`, and to convert, you have to tell the game what to convert to using `self:set_conversion(type)`, whereas type is one of 0, 1 or 2.

`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.

`self:nearest_enemy()` returns the nearest enemy, and information about it. Use it like this:

```
enemy_id, x, y, player = self:nearest_enemy()
```

Then enemy_id is the enemy bug's number (which can be used in `self:set_target()` or `self:attack()`), x and y the coordinates (for example, to walk to) and player is the number of the player the bugs belongs to.

7 Example Bot

Here is a bot that just tries to stay alive, eat as much as possible and spawn as often as possible. It does not kill, it does not flee and it does not make points – that's up to you!

```
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 and self:food() < self:max_food() then
      self:screen_message('hmm')
      self:eat()
      if self:health() < 80 then
        self:heal()
      end
      self:begin_walk_path()
    end
    if self:health() < 20 or (self:food() == self:max_food() and self:↵
        health() < 100) then
      self:screen_message('heal')
      self:heal()
      self:begin_walk_path()
    end
    if not (self:type() == 1) and self:food() > 0.9 * self:max_food() ↵
        and self:health() > 90 then
      self:screen_message('morph')
      self:convert(1)
      self:begin_walk_path()
    end
    if self:type() == 1 and self:food() > 0.8 * self:max_food() and ↵
        self:health() > 80 then
      self:screen_message('daddy')
      self:begin_spawning()
      while self:is_spawning() do
        self:wait_for_next_round()
      end
      self:begin_walk_path()
    end
    self:screen_message('search')
    self:wait_for_next_round()
  end
end
```

8 Final Words

This was the last time the Obroni Computer Club came together. I hope that all of your could benefit from it in one way or the other, maybe by learning useful things, maybe by having some fun, maybe be having a different view on computers. Although I had planned to be here for a whole year, it did not work our. If you have questions why, or questions related to what we have done in the club, or any other kinds of questions, feel free to e-mail me at mail@joachim-breitner.de. I will you the best during your years at SOSHGIC and especially the time "in the wild" afterward.

Joachim Breitner