The Feebs War

Gustavo Henrique Milaré

March 15, 2008

Abstract

The Feebs War is a modified version of Planet of The Feebs `http://www.cliki.net/Planet\%20of\%20the\%20Feebs`, a game made for people learn and improve their lisp and code manipulation tecniques. The graphics are now displayed using Lispbuilder `http://lispbuilder.sourceforge.net`'s libraries, so the problems with portability from CMUCL and X Window Sistem do not exist anymore. Also the code is cleaner and more extensible.

Contents

1 Introduction

The Feebs are intelligent and hostile creatures that live inside maze tunnels. They also have no mercy with each other, so they frequently throw a letal flame from through their mouth, getting rid of their opponent and eatting the carcass left. But throwing flames have an energy cost, so they must keep tracking for food.

This game is intended to help lisp newbies (or maybe a little more advanced lispers) to learn lisp. A player must create a function that receives what his/her feeb is seeing and feeling, and returns what it will do next. To create the better feeb, one can create variables to store data from previous moves (or not), and can also use all the power of lisp to improve his/her creature. But the most important is to make good choices and be aware of danger!

1.1 Changes from *Planet of the Feebs*

Many changes were made from the original game, but, if you have any feeb definition and you want to use it, it should be easy to adapt the brain function to the new rules.

The main reason of this project is that *Planet of the Feebs* is really interesting for (not just) newbies to learn lisp, but the difficulties to install, unportability and the ausence of competitors make it difficult to someone to be interested in making a feeb. So, I hope that making these adjustments and maybe creating some contests over the web make people be more interested in learning lisp.

So, these are (some of) the changes:

- The graphics are not based on X Window Sistem anymore, but on *Lisp-builder*, and there are no CMUCL's event handler. This way, the code is more portable and graphics can be improved. Just creating some image files of a feeb and your feeb is much more personalized!
- Every element of the map (including walls) is a list, so the brain of a feeb doesn't need to test all the time if the element is an atom or a list (wich, in my opinion, is really boring, unlispy and unnecessary in this case). That was only a reason to duplicate code and work, adding no results at all...
- Many functions and variables are changed and others were added
- Documentation is more objective than the one provided with *Planet of the Feebs*, and is fully compatible with the code. This way it is easier to understand the game.
- Security is improved. Now it the behavior functions are allowed to store and change structures and vectors passed to it. The parameters can't be change by those functions while inside the game.

- It is possible now to extend the rules: the code is object oriented and new rules, special moves, change the behavior of flames, etc, can be done by adding new classes and/or methods. This manual is just the beginning! enditemize

2 The Game

2.1 Overview

Your feeb's objective is to survive and kill other feebs. It is inside a maze of tunnels. Every turn, all feebs lose one unit of energy, and maybe starves. Your feeb is able to move forward, turn left, right or around, flame, peek around a corner, eat something or just wait. After all feebs move, the flames thrown before also move (or dissipate), carcasses may rot and mushrooms may grow, accordingly to some rules.

The game rules are defined by parameters. These parameters can be read by the command **(get-feeb-parm** 'parameter**)** To see all parameters, values and also all the documentation, one can use **(list-parameter-settings)**. Using **(change-feeb-parm**'parameter value**)** gives the possibility to change them (but not during the game) and **(documentation** 'parameter 'feeb-parm**)** can be used to know them. Just remember that every probability must be a rational number (like 1/2).

But don't panic! These parameters are just for one to know how the game is going to be, but in the begining there is no need to explicitly use them when creating the brain of a feeb. The best way to create a feeb is watching a game (among system feebs), improving it (it is defined in file brains.lisp) a little more, testing the changes...

These are some global parameters:

'game-length Number of turns the game will last.

'points-for-killing How many points some feeb earn for killing someone.

'points-for-dying How many points some feeb earn for dying (usually negative).

'maze-x-size Horizontal size of the maze.

'maze-y-size Vertical size of the maze.

2.2 Throwing flame

If a feeb decides to throw a flame, if it is prepared to and has enough energy, the next turn there will be a flame in the square in front of the feeb, and it will see it, so the feeb shouldn't move forward. For a few turns, the feeb will not be able to throw flames. Each turn, the flame moves forward destroing mushrooms and killing feebs it encounters, transforming them into carcasses. If there is a wall, the flame can reflect, and, if so, it will turn 180 degrees.

Once a feeb is killed (or starves), in it's place in the maze there will appear a carcass. The feeb goes to the end of the dead feebs line. When the carcass rots, the first feeb in line reincarnates. So, dying is not so terrible.

These are the parameters related to flames:

'flame-energy Amount of energy lost after throwing a flame.

'fireball-guaranteed-lifetime Number of turns that a fireball is guaranteed not to dissipate, unless it encounters a wall.

'fireball-dissipation-probability Probability of the flame to dissipate each turn after the apropriate time.

'fireball-reflection-probability Probability of the flame to reflect when encountering a wall.

'flame-no-recovery-time Number of turns that a feeb cannot fire.

'flame-recovery-probability Probability of the feeb to recover the hability to throw a flame, after the apropriate time.

2.3 Eating food

There are two kinds of food, carcasses and mushrooms. Carcasses usually give less energy than mushrooms, and may rot, but, while it does not rot, a feeb can feed as long as it wishes. Mushrooms disapear after being eaten. By eating food, the feeb will be able to recover energy, wich is important because, if a feeb stays with 0 or less units of energy, it starves.

These are the quantities:

'mushroom-energy Amount of energy recovered when the feeb eats a mushroom.

'carcass-energy Amount of energy recovered each turn that the feeb eats a carcass.

'carcass-guaranteed-lifetime Number of turns that a carcass will surely not rot. After these turns, it can rot, depending on probabilities.

'carcass-rot-probability Probability of the carcass to rot, after the apropriate time.

'maximum-energy Maximum amount of energy that a feeb can have eating.

'starting-energy Amount of energy a feeb has when it reincarnates.

'number-of-mushrooms Quantity of mushrooms that exist in the maze.

3 The Feeb

A feeb needs four things: a name, a brain and a set of graphics (optional).

- The name, a string.
- The brain is a function that decides what the feeb will do next, based on what it is seeing and feeling.
- The set of graphics is an image file (of format BMP, JPEG, PNG, and any others that supported by SDL_image).

One can create a feeb calling **(define-feeb** name brain **:graphics** graphics**)**. If name is already used, a warning will be signaled, and the old feeb will be substituted. Calling **(list-of-feebs)** will return the list of the feebs (names only) that will be defined when the game begins. **(delete-feeb** name**)** will delete the feeb with this name from this list, and **(delete-all-feebs)** will clear it.

3.1 Possible decisions

After processing the information available, the brain will take a decision. If this decision is not one of the decisions listed down, a warning will be signaled, and the result will be like **:wait**. Then, if someone are testing a brain function, he or she will be able to know if something goes wrong.

The possible values that the brain function can return are these:

:move-forward Move one square forward, unless there is a wall in front of the feeb.

:turn-left Turn 90 degrees to the left.

:turn-right Turn 90 degrees to the right.

:turn-around Turn 180 degrees.

:flame Throw a flame. The flame will be created next turn in the front square of the feeb.

:wait Do nothing in this turn.

:peek-left Peek to the left around a corner. The creature does note actually move, but, in the next turn, the creature will have the same vision that it would have if he had moved one step foward and turned left (and one step back because the feeb needs to see what is actually in front of it). Peeking used so a feeb can analize a corridor before trespassing it.

:peek-right Peek to the right around a corner, analogous to **:peek-left**.

:eat-carcass Eat a carcass if there is any available in the feeb's square. The amount of the parameter **'carcass-energy** is restored to the feeb's energy.

:eat-mushroom Eat a mushroom if there is any available in the feeb's square. The amount of the parameter **'mushroom-energy** is restored to the feeb's energy.

3.2 Information available

The brain of a feeb must take five arguments; I'll call them *status*, *proximity*, *vision*, *vision-left* and *vision-right*.

3.2.1 Status

Every time the brain is called, it receives some useful information through **status**.

The structure **status** keeps information about the feeb itself. Also it has information of the previous movement of the feeb. To access them, one must call:

(name *status***)**

The name of the feeb.

(facing *status***)**

Where the feeb is facing to, one of the constants provided: **north**, **east**, **south** or **west**, wich are 0, 1, 2 and 3 respectivelly.

(x-position *status***)**

The horizontal position of the feeb, starting with 0 and increasing to east. If **'sense-location-p** is nil, it returns nil instead.

(y-position *status***)**

The vertical position of the feeb, starting with 0 and increasing to south. If **'sense-location-p** is nil, it returns nil instead.

(peeking *status***)**

If it is **:peek-left** or **:peek-right**, it means that the current *vision* provided is result of a previous **:peek-left** or **:peek-right** command of the same feeb. Otherwise, it is **nil**. Note that *proximity* is *not* affected.

(line-of-sight *status***)**

Indicates the amount of valid entries in *vision*. It actually means that **(aref** *vision* **(line-of-sight** *status***))** will return **'(:rock)**.

(ready-to-fire *status***)**

If **T** indicates that the feeb is ready to fire. If **Nil** indicates it is not.

(aborted *status***)**

Related with timing. Returns **T** if the last move of feeb was aborted because of timing issues.

(last-move *status***)**

The feeb's previous move, or **:dead** if it has just reincarnated.

3.2.2 Proximity and vision

The brain receives also information about what is near the feeb and what the feeb sees. We note that, contrary to *Planet of the Feebs*, it is safe to change anything inside these structures, so you are alowed to keep them stored and to modify them as you wish.

The structure *proximity* has the contents of the squares near the feeb (not affected by peeking) with these fields:

(my-square *proximity***)**

Contents of the feeb's current square.

(left-square *proximity***)**

Contents of the right square of the feeb.

(right-square *proximity***)**

Contents of the left square of the feeb.

(rear-square *proximity***)**

Contents of the square behind the feeb.

The vector *vision* has the contents of the squares that are in front of the feeb. For example, **(aref** *vision* 0**)** will return the contents of the square in front of the feeb, **(aref** *vision* 1**)** will return the contents of the next square, and so on. As said before, **(aref** *vision* **(line-of-sight** *status***))** will be the first **'(:rock)** encountered. All subsequents square, like **(aref** *vision* **(**+ 1 **(line-of-sight** *status***)))**, will be garbage and should not be used.

The contents of one square returned by any of these calls is either a list of elements, a wall **'(:rock)** (i.e. a list with one element, a **:rock**) or **()** if the square is empty. Each element of the square is one of these:

- **Feeb image.** One can call **(feeb-image-p** element**)** to see if element is a feeb image.
- **Fireball image.** One can call **(fireball-image-p** element**)** to check if element is a fireball image.
- **:carcass**. If there is a **:carcass** in the square of the feeb (i.e. in **(my-square** *proximity***)**), the call **:eat-carcass** will make the feeb eat it.
- **:mushroom**. Analogous to **:carcass**. A mushroom appears randomly in places previously marked in the map.

3.2.3 Feebs and fireballs images

Both fireballs and feebs that are given to the brain function are not the real ones, but just images with contents that the brain function can access. It is allowed to keep and change its contents because they won't be used internally.

These are the accessors available (they read and change the fiels):

(feeb-image-name feeb-image**)**

The name of the feeb. (Maybe you know it's weakpoints?)

(feeb-image-facing feeb-image**)**

The facing of the feeb. This way the brain function can see if the feeb-image either sees the feeb which is playing or not.

(feeb-image-peeking feeb-image**)**

Returns **:peek-left** or **:peek-right** if the feeb is peeking to (its) left or right, or **nil** if not.

(fireball-image-direction fireball-image**)**

The direction where the fireball image is going to.

3.2.4 Vision-left and vision-right

vision-left and *vision-right* are vectors similar to vision, but they are less precise in the contents. Also their valid contents are limited by **(line-of-sight** *status***)**, so **(aref** *vision-left* **(line-of-sight** *status***))**, for example, will return **:unknown**.

Note that feebs that are not peeking, mushrooms and carcasses are *not* be detected by these vectors. Also, if there is a feeb peeking to the opposite side, it won't be detected either. The elements in **vision-left** and **vision-right** are lists containing these elements:

:peek-letf	This means that in that square there is a feeb peeking to (its) left.
:peek-right	This means that in that square there is a feeb peeking to (its) right.
:rock	This square is just a wall. In this case, this is the only element in the square.

3.3 Extra functions provided

Before making the brain of your feeb, you might want to take a look at the Extra functions that are available in the file *definitions/extra.lisp*. The only thing you need so you can use them is to see their code and know what they do.

3.4 Changing the map layout

It is possible to change the layout of the map by calling **(change-layout** new-layout**)**. There are a few predefined mazes that are in variables ***maze-0*** (which is set by default) throw ***maze-5***. In a layout, 'X' represents a wall, 'e' represents a feeb entry point (there will be as many entry points as feebs in the maze at the same time), 'm' represents a mushroom site and ' ' is a blank space.

If you want to create a new map, you can start by an empty template of any size that is provided by **(make-template** x-size y-size**)**, or you can get a reandom map calling **(generate-maze** x-size y-size **:density** density**)** The density is a number, recomended to be between 0.25 and 0.45, which tells the portion of the maze should be blank spaces. The function quits after a while if it doesn't meet this portion. See its documentation for more details and options.

3.5 Graphics

With this version of the game, it's possible to choose the graphics of a feeb when creating it, so your feeb will be more personalized.

The graphic of a feeb is defined by an image file, which should have three colunms by eight lines of pictures of the same size. The four first lines must be the animations of the feeb walking up, left, down and right, respectively.

The next four lines must be the pictures of the feeb flaming up, left, down and right, respectively. To see an example, see "default-feeb.png".

After creating the image file, you must call **(create-graphics** path-to-image-file**)**. If you now how to work with sdl surfaces in lispbuilder, you may use the function with a surface instead of a image file; or you can call **(create-graphics** path-to-image-file nil**)** if the surface should not be freed after the call. The result must be the third argument given to define-feeb.

3.6 Starting the game

The game loop is started by calling **(simple-play)**.

4 Contests

I sugest that you see this chapter only after you have created at least a basic brain feeb, which is better than the (simple) provided brain, or if you want to participate of a contest or a game with your friends.

4.1 Map

It is possible to get the maze map during the game, but with only the corridors. Note that the function that gets the map is purposely a little slow, so, invoking it too many times in a contest that uses timing atributes is not a good idea; anyway, it is possible to invoke this function before defining the feeb, and store its value somewhere. Also note that the map returned does not have any information about what is really in the maze, but only the possible ways.

To get the map, one can call **(get-maze-map)**. This function will return **nil** if parameter **'may-get-maze-map-p** is also **nil**. Otherwise, the map returned is an array, so that calling **(aref** map x y**)** will get the contents in the position (x,y) (like euclidean but inverting the y axis). The contents of a cell could be one of these:

:mushroom-place A mushroom patch, i.e. when a mushroom is reincarnate, it could reincarnate here.

:feeb-entry-place A feeb entry, i.e. if a carcass rots a feeb can appear here.

:rock A wall. Feebs cannot come to this place.

nil An "empty" place, i.e. neither of the previous.

This map can safelly be used since **(get-maze-map)** makes a new copy every time it is called.

4.2 Timing

There are also some timing atributes that can be given to the game. The more time the feeb takes make a decision, greater is the probability of its command to be aborted.

To make this available, someone must set these parameters:

'slow-feeb-noop-switch If is non-nil, there is a possibility that the move of a feeb is aborted according to its function time.

'slow-feeb-noop-factor The probability of the feeb to abort will be this factor times the amount of time the feeb takes to have a decision, divided by the total time taken by all the feebs in the current turn or divided by a reference time.

'reference-time Time taken by reference if non-nil.

'points-for-slow-down Points earned when a feeb's move is aborted due to slowness.

4.3 Sense of location

Some accessors related to position and orientation of the feeb can be turned off.

These are the parameters:

'sense-location-p If nil, **x-position** and **y-position** will return nil when someone tries to invoke it. Otherwise return the position.

4.4 Changing the rules

To change the rules of the contest, they must be changed before the feebs are defined, because in a feeb definition it could use the values of the variables to make a global strategy.

All the parameters, values and documentation that can be listed using **(list-parameter-settings)** can be changed using **(change-feeb-parm name value)**, which is deactivated during the game. Also, they all have documentation about themselves, so feel free to use **(documentation** 'parameter 'feeb-parm**)** and see what each parameter does. Documentation is available to external functions as well.

5 Reference

Fahlman, S. E. ***Planet of the Feebs*** *- A Somewhat Educational Game.* `ftp://ftp.csl.sri.com/pub/users/gilham/feebs/feebs.tex`.