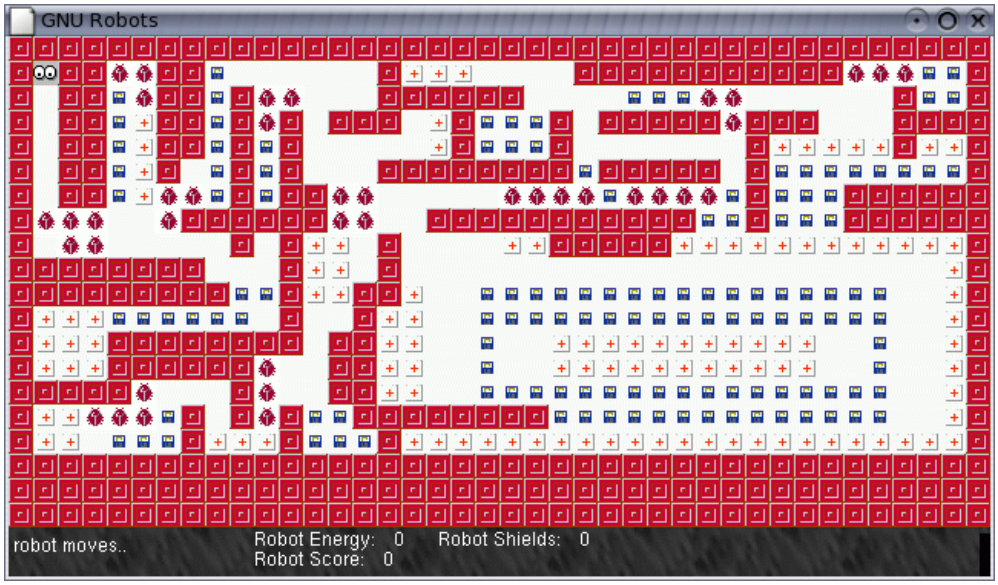
Principles of Autonomy and Decision Making Project

This text is taken from the GNU Robots manual, written by Jim Hall.

Introduction

GNU Robots is a game/diversion where you construct a program for a little robot, then set him loose and watch him explore a world on his own. The robot program is written in Scheme, and is implemented using GNU Guile.



The whole goal of GNU Robots is to get the highest score that you can.

Robot actions

But the question you're probably waiting to have answered is: "What can I make the little robot do?" That's the list of 'robot-*' functions. Here is a table of all the GNU Robots functions, called primitives, that you can use in your robot program. Each primitive has an energy cost associated with it. Note that some primitives will take an argument, and some will not. If an argument is a number, I'll print 'N' after the function name. If the argument is a GNU Robots "thing" then I'll print 'TH' after the function name. (Note that number arguments are just numbers, but "thing" arguments are strings enclosed in quotation marks.) If the primitive does not take an argument, I will print nothing.

PRIMITIVE	ENERGY	ACTION
robot-turn N	-2	Turns the robot N spaces to the right or left. If 'N' is a positive number, the robot turns to the right. If 'N' is negative, the robot turns to the left. Every turn has a cost of -2, so if you 'robot-turn 2' then the cost is -4 to your energy.
robot-move N	-2	Moves the robot N spaces forwards or backwards. If 'N' is a positive number, the robot moves forwards. If 'N' is negative, the robot moves backwards. Every move has a cost of -2, so if you 'robot-move 2' then the cost is -4 to your energy.
robot-grab	-5	Attempts to pick up what is in the space directly in front of the little robot. Note that you should not try to pick up baddies or walls. If the item is food, you increase your energy by +15 after the grab is deducted. If the item is a prize, you increase your score by +1.
robot-zap	-10	Uses the robot's little gun to zap whatever is in the space directly in front of it.
robot-smell TH	-1	Smells for a thing. If the thing is in the surrounding eight spaces, the function returns a true ('#t') value.
robot-feel TH	-1	Feels for a thing in the space directly in front of the robot. If the thing is detected in that space, the function returns a true ('#t') value. Note that you can feel a baddie, and not take any damage.
robot-look TH	-1	Looks ahead across empty spaces for a particular thing. If the thing is seen, the function return returns a true ('#t') value. Note that this is sort of a braindead function, in that it does not provide the distance to the thing. The robot cannot look behind objects, so if you look for a food item, and there is a prize item in the way, the function returns false.
robot-get-prizes	0	Returns the x, y location of all the existing prizes in the map.
robot-is-wall X Y	0	Returns `#t` if the map grid cell at (X, Y) is a wall. Returns `#f` otherwise.
robot-get-shields	0	Returns the level of the little robot's shields.
robot-get-energy	0	Returns the level of the little robot's energy.
robot-get-score	0	Returns the robot's score (how many prizes have been picked up.)
robot-get-prizes	0	Returns a list of the locations of the prizes.
robot-get-food	0	Returns a list of the locations of the food packets.
robot-get-position	0	Returns the current x/y/direction of the robot.
robot-get-map-size	0	Returns the size of the x/y size of the map.
robot-success	0	Returns the result of the last action

And what kinds of "things" are out there to be detected by 'robot-smell', 'robot-feel' and 'robot-look'? Here is a list of all possible things in the GNU Robots world:

THING	DESCRIPTION	
	baddie	A nasty little critter that is generally bad for your health. Either leave these alone, or zap them. Don't try to pick them up, or you will inflict damage to your shields. Don't bump up against them–that doesn't feel too good, either.
	space	An empty space. There's nothing interesting here.
	food	Yum! A health item that will help to restore +20 points of your robot's energy levels. (15 points after the 5 points for the grab is deducted.)
	prize	Pick these up! This will add +1 point to your score. Remember, the goal of GNU Robots is to get the highest score!
	wall	An obstacle. You can't zap these, so you better go around them. Trying to grab a wall does nothing to you, but bumping up against one will inflict damage to your shields.

To increase your score, you need to pick up prizes. So let's create a function that allows our little robot to pick up some prizes.

```
(define (grab-prize)
  (if (eqv? (robot-feel "prize") #t) (grab-and-move))
)
```

The 'eqv?' statement compares the truth value of two expressions. In this case, we compare a test expression against a literal true value ('#t'). Our test expression is '(robot-feel "prize")' which will return a '#t' value if the little robot is able to detect a prize in the space immediately in front of it.

If the 'eqv?' statement is successful (the two expression are the same, or in other words the robot is able to feel a prize in the space right in front of it) then Scheme will execute the 'grab-and-move' function.

In plain English: if the robot detects a prize in the space immediately before it, the robot will pick it up, and move forward into that space.

But every time you make the little robot take an action, you cause it to use energy. To restore its energy levels, it needs to find and consume food. So let's create a function similar to the above that picks up a food item:

```
(define (grab-food)
  (if (eqv? (robot-feel "food") #t) (grab-and-move))
)
```

As you can see, the 'grab-food' function is virtually identical to the 'grab-prize' function. I don't think you need me to explain the difference.

From here, you are on your way to creating a cool robot that you can set loose in GNU Robots. It's just a matter of picking up some more of the flow control structures like do-while loops and more of the if-then tests. Then you'll have a fine robot program to play with.

How to run GNU Robots

We have provided multiple versions of GNU Robots. Some run under X Windows, such as you might use at a Linux workstation. The other versions are text-based version, and can be run on remote athena workstations from any machine, such as a Windows terminal.

To get GNU Robots on athena, log into an athena workstation and copy the GNU Robots directory to your local directory:

```
% cp /afs/athena.mit.edu/course/16/16.410/GNU_Robots-16.413 .
```

You can download the linux version of GNU Robots to your local machine from `http://web.mit.edu/16.410/www/GNU_Robots-16.413-Linux.tar.gz`. This will give you a tar file that you can then unpack using:

```
% tar xzf GNU_Robots-16.413-Linux.tar.gz
```

If you look inside the directory (either under athena or under linux), you will see three files called `robots`, `xrobots` and `robots_logfle`. The first, `robots` is a program that runs your code with an ASCII graphical interface. This is the program you most likely want to run on an athena machine.

The second program, `xrobots` runs your code with much prettier graphical interface under X-Windows. This is the program you most likely want to run under windows.

The third program, `robots_logfile` runs your code without any interface at all, but just reports the result of each move.

The fourth program, `interactive_robots` lets you run your code with a prompt, reporting the result of each move.

The final program, `xinteractive_robots` lets you run your code with a prompt and also gives you an X windows GUI.

Running a Robot Interactively

You can run the robot interactively by running `interactive_robots` or `xinteractive_robots`. The `xinteractive_robots` version will pop up a gui, but the simple `interactive_robots` version will just run in text mode. The interactive programs give you a standard scheme prompt, and allow you to run commands. For example:

```
nickroy@mapleleaf:[src] 24>./interactive_robots -f ../maps/maze.map
GNU Robots, version 1.0
Copyright (C) 1998,1999,2000 Jim Hall <jhall1@isd.net>
GNU Robots comes with ABSOLUTELY NO WARRANTY
This is free software, and you are welcome to redistribute it
under certain conditions; see the file ‘COPYING’ for details.
Loading Guile ... Please wait
Map file: ../maps/maze.map
GNU Robots starting..
thing ‘#’ added at 0,0
thing ‘#’ added at 1,0
...
thing ‘#’ added at 38,19
thing ‘#’ added at 39,19
guile> (robot-turn 0)
#t
guile> (robot-is-wall 1 1)
#f
guile> (robot-look "baddie")
#f
guile>
```

All commands are activated for the interactive programs: you can run any command at any time.

Running a Robot Program

You run a robot program by specifying a map file and a Scheme instruction file. The map file should be self-explanatory: it describes where the obstacles are and where the baddies are, etc. The Scheme instruction file provides the code that you will write in order to control your robot, using the operators described above.

An example session might be:

```
% xrobots -f maps/maze.map scheme/beep1.scm
GNU Robots, version 1.0
Copyright (C) 1998,1999,2000 Jim Hall <jhall1@isd.net>
GNU Robots comes with ABSOLUTELY NO WARRANTY
This is free software, and you are welcome to redistribute it
under certain conditions; see the file ‘COPYING’ for details.
Loading Guile ... Please wait
Map file: ../maps/maze.map
Robot program: ../scheme/beep1.scm

----------------------STATISTICS----------------------
                         ROBOT ONE
------------------------------------------------------------
Shields: 100
Energy: 988
Score: 0
```

The `beep1.scm` file is an example program we have supplied: it turns the robot to face each direction (north, south, east, west), and beeps if it can feel a prize.

Controlling a Robot in Batch Mode

If you look inside the `beep1.scm` file, you'll notice one important feature: the code never explicitly calls `robot-turn`. Instead, there is a function called `controller`. The way that the batch mode works is that, once your file is loaded, the Scheme interpreter makes repeated calls to your `controller` function, and expects a list in return. This list should contain one of two Scheme expressions to be evaluated: `robot-turn` or `robot-move`. The interpreter then takes the list returned by `controller`, and evaluates it.

The GNU Robots package will not allow you to call `robot-turn` or `robot-move` yourself. The reason for this is to enforce turn-taking, in head-to-head competition of the robots.

Running Multiple Robots

In order to make the planning problem interesting, we have provided the ability for two robots to play against each other. You can call GNU Robots with an *opponent* controller file, using the `-o` option:

```
% xrobots -f maps/maze.map -o scheme/beep2.scm scheme/beep1.scm
GNU Robots, version 1.0
Copyright (C) 1998,1999,2000 Jim Hall <jhall1@isd.net>
GNU Robots comes with ABSOLUTELY NO WARRANTY
This is free software, and you are welcome to redistribute it
under certain conditions; see the file ‘COPYING’ for details.
Loading Guile ... Please wait
Map file: ../maps/maze.map
Robot program: ../scheme/beep1.scm

----------------------STATISTICS----------------------
                        ROBOT ONE
------------------------------------------------------------
Shields: 100
Energy: 988
Score: 0

                        ROBOT TWO
------------------------------------------------------------
Shields: 100
Energy: 988
Score: 0
```

Notice that as the code executes, each robot turns at a time.

(One caveat: the `-o` option must come before your own scheme file on the command line.

```
% xrobots -f maps/maze.map scheme/beep1.scm -o scheme/beep2.scm
```

will not do what you want.)

FAQ

1. What is the starting shield strength? What is the shield damage for collisions with various objects? Is there a way to replenish shields?

The starting shield strength is 1000.

If you collide with another robot or a baddie, then your shields are depleted by -10. If you collide with a wall, your shields are depleted by -2. In all other cases, your shields are depleted by -1.

There is no way to replenish shields.

2. What happens to non-baddie, non-wall objects when you zap them?

   If you zap space, wall, or another robot, nothing happens.

   If you zap a baddie, food or a prize, then they disappear. Zapping depletes your energy by -10.

3. What are the limits of what can be done on a single turn? (sensing, moving, turning, grabbing)

   During interactive play, (i.e., using interactive-robots or xinteractive-robots), there aren't really "turns", and all actions are limited only by your available energy.

   During competition play (i.e., using robots or xrobots), when your implementation of `controller` is called, you are able to call the following actions as often as you like, and you are limited only by their impact on your energy.

   ```
   robot-smell
   robot-feel
   robot-look
   robot-is-wall (Phase I only)
   robot-get-food (Phase I only)
   robot-get-prizes (Phase I only)
   robot-get-shields
   robot-get-energy
   robot-get-position
   robot-get-map-size
   robot-success
   ```

   Your implementation of `controller` cannot, however, call the following actions:

   ```
   robot-turn
   robot-move
   robot-grab
   robot-zap
   ```

   You must return a scheme expression containing one of these actions, which will then be called for you. If you call `robot-turn` from within controller, nothing will happen.

4. What happens if you collide with a non-wall object on a move? Damage, final position, persistence of object?

   The object persists. You don't move, and your shields are depleted in strength (see question 1).

5. How are the starting points selected, particularly for multiple robots?

   This is undecided at the moment.

6. What happens if two robots collide?

   See question 4.

7. Do smelling, feeling, looking include walls?

   Yep.

8. What is the starting energy level?

   The starting energy is 100.

9. Will competitions be run using multiple maze trials? (affects strategic risk aversity)

   This is currently undecided.

10. If the robot tries to move and hits something, they lose shields but do they lose energy for that particular move? What if someone commanded robot-move 10 and ended up not moving at all? Would they lose 20 energy (for the 10 moves commanded), 2 energy (for the one attempted move) or 0 (due to not moving at all)? And in competition play, if you command a move, hit something immediately and end up not moving, does this count as a turn?

    You lose shields and energy for unsuccessful moves. but, if you try to move 10 and can only move 2, you'll only lose energy for the 1 successful move and the one unsuccessful move. and yes, if you command a move and hit something immediately, this counts as a turn.

11. Do we have knowledge of where the robot starts within the maze? (ie. top left) Or does the robot also have to perform localization?

    For Phase I, you can query `robot-get-position`. For Phase II, you don't have knowledge of the map, so the robot's position is not helpful, but you can still query it.

12. In terms of the size of the map will this be given as the total number of rows and columns? Additionally, is it correct to assume that every space around the perimeter of the map will be filled with a wall object? If not, what is the consequence of falling off the edge of the map? What happens at the edge of the universe (aside from traveling at a velocity near the speed of light)?

    You can assume that the edge of the maze consists of walls. like columbus, you can't fall off the edge.