BeeperBot Manual

© Joseph Culberson Janelle Harms
Herb Yang
Edition 1.0 date August 23, 2010

August 23, 2010

There is more discussion of programming BeeperBot in the full CMPUT 101 course notes available at `http://www.cs.ualberta.ca/resources-services/teaching-resources`. This manual is a subset of Chapter 1 in those notes. The focus of this manual is on the tool, if you want to learn how to program BeeperBot we suggest you read Chapters 1 and 2 of the full CMPUT 101 notes.

1 What is BeeperBot?

BeeperBot, or BB for short, is a programming environment to control a primitive graphical robot that exists in a two dimensional grid world. We often refer to the robot as BB itself. The robot can only interact with walls and beepers. For walls, BB can only detect them, and if she tries to move through a wall, an error occurs.

BB can create, destroy, collect and distribute beepers. She has no built-in way to count, and so BB can only distinguish between having one or more beepers, or none.

Nevertheless, BB can carry out powerful computations, given enough time. Many elementary but fundamental concepts of computing science can be illustrated easily with BB. And that is of course the goal of this software.

2 Obtaining BeeperBot (BB)

Before proceeding further, you need to obtain the program BeeperBot from the website, if you have not already done so. (We assume you have access to a computer for this course. If you do not, then the following exercises will have to be carried out in one of the computing science labs. We suggest you be prepared to spend quite a bit of time in the labs in this case.)

1. Download the file `beeper.jar` from this site.

   `http://www.cs.ualberta.ca/resources-services/teaching-resources`

2. Put the file `beeper.jar` in the folder where you keep applications.

To uninstall BeeperBot, simply remove the file `beeperbot.jar`. In addition, if you have changed your settings, in your home directory there may be a file `.gvrsettings` which you may also remove.

3 Getting Started with BB

On a MAC or PC you should be able to start BB by clicking or double clicking the file `beeper.jar`.

On Unix or Linux enter the command

```
java -jar /Applications/beeperbot.jar
```

where of course you must replace `Applications` with the path to the location of `beeperbot.jar` on your system. You can also start it this way in a terminal in MAC OS X if you prefer that to clicking.

On MAC OS X you can also drag `beeperbot.jar` to the dock (at the document end) and start it from there.

After starting BB, you should have a window that looks like Figure 1. The graphics may vary slightly depending on the system you are using. This may also differ a bit if you have used BB before and saved a different set of preferences. The following assumes this is the first time you have run BB, and that BB is using the default settings.

The window consists of five panes which we now examine. You have BB running so you can follow along.

Program Pane: This is found on the left side of the window, and is an editable text area. When started it contains a default program consisting of only the lines shown in Figure 1.

Try it now: Click in the window and enter the text " *move*" at line 2 followed by the line " *pick_beeper*" at line 3. Your program should now look like

```
define main {
  move
  pick_beeper
}
```

Initial World Pane: Located on the top right of the window, this is another editable text window. Click on the initial world pane and enter the lines below exactly as shown (note the capital 'N', all other letters are lower case).

```
robot 2 3 N
beepers 2 4 1
```

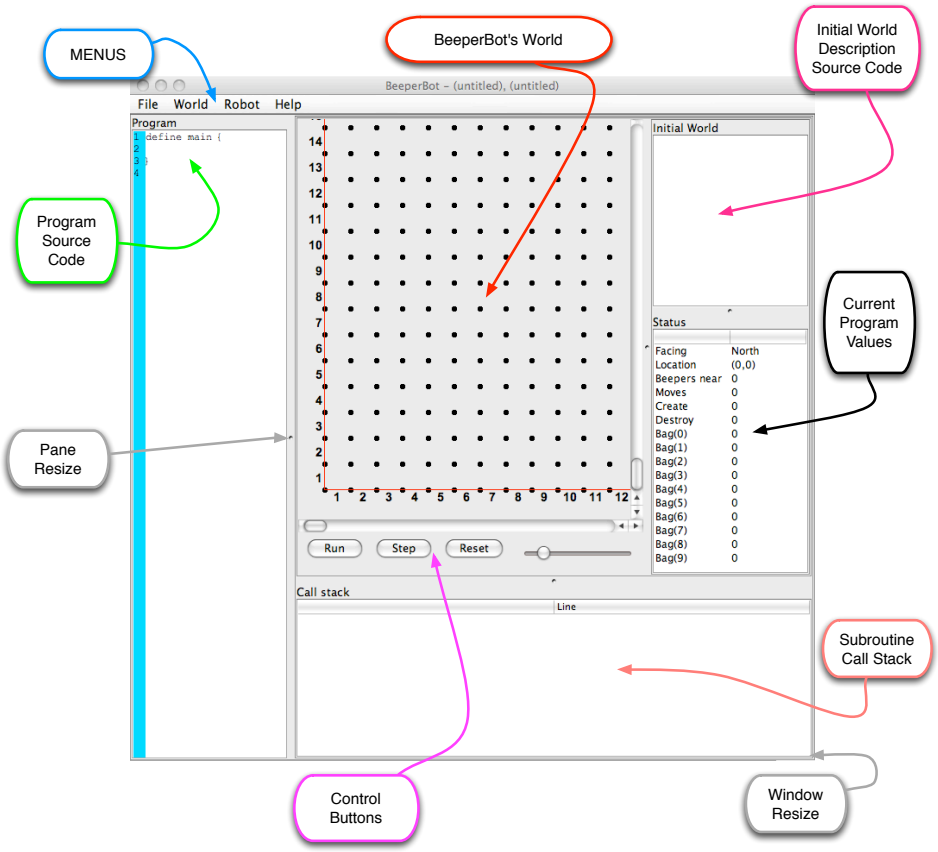
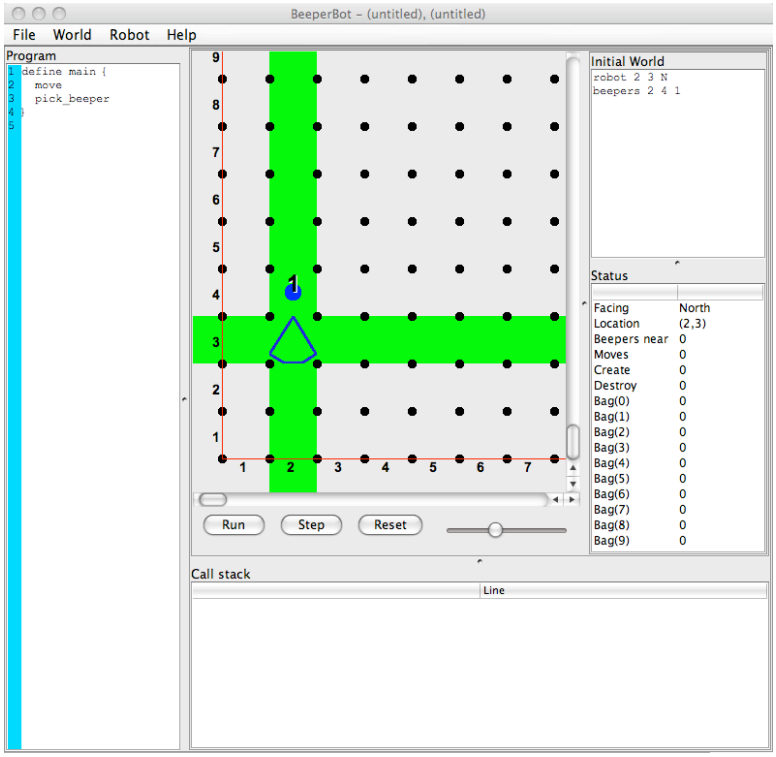


Figure 1: On initial start up, BeeperBot will look like this, without the colorful labels of course. Your display may differ slightly depending on the system you are using and the options you are using.

World Pane: BeeperBot's World is located in the center top of the window. At the bottom of this pane there are three buttons that control execution of the program. There is also a slider knob that changes the scale of the world display.

- **Reset** Initializes the world. **Try it now**. Click the **Reset** button. You will see that the world changes to display the location of the robot, with a blue shape representing the robot, and a green cross indicating the coordinates. (The green coordinate display can be turned off in the menu **Robot** if desired.) After clicking **Reset** the Window should look like this.



- **Step** Executes one step of the program each time it is clicked. **Try it now**. Note that each time you click, a line of text is highlighted in the program pane. The highlighted line is the next step of the program to be executed.

  Note that on the first click, the line *define main*{ is highlighted, the second click highlights the line " *move*". On the third click the line " *pick_beeper*" is highlighted, and at the same time the robot in the world pane moves one step forward.

  If you keep clicking the program will start over again, after first resetting to the initial world description.

- **Run** Executes the program. While the program is running this button turns to **Pause**, although for this short program it may be too quick to notice. If the program is paused, then clicking **Run** again continues the program from where it was paused. Clicking **Reset** at any time stops the program and resets it to the initial world conditions.

Status Pane: This is located on the right center. It is a display only text window. It indicates the following:

- **Facing** The direction the robot is currently facing, with "North" being "up".
- **Location** The column and row BB is currently in.
- **Beepers Near** The number of beepers in the same location as BB.
- **Moves** The total number of *move* and *turn_left* statements that have been executed so far in the current run.
- **Create** How many beepers BB has created so far in the current run.
- **Destroy** How many beepers BB has destroyed so far in the current run.
- **Bag(0)...Bag(9)** In Auxiliary mode BB has 10 bags, labeled 0 through 9. Each bag will be listed on a separate line. The number on the right indicates the number of beepers currently in the bag. Note: in primitive mode BB has no bags, and in standard only one bag. The display will change according to which mode BB is in.

Call Stack Pane This is a text display window that shows the current subroutine calls.

3.0.1 Tutorial Exercise

Using the program and world defined above, step through the program one step at a time, and observe precisely when the robot moves and when it picks up a beeper putting it into *bag(0)*. After each click of the **Step** button, carefully check both the World display, and the information in the **Status** pane. Notice in particular that the code line is highlighted *before* the action takes place. Also note that some code lines do not change either the world or status file.

4 A brief overview of BB's menus

As indicated in Figure 1 there are 4 menu items in BeeperBot.

File For loading, saving and creating new programs, as displayed and edited in the **Program** pane. Also contains the item **Exit** which exits BeeperBot.

World For loading, saving and creating new initial world descriptions, as displayed and edited in the **Initial World** pane.

Robot This item has has several functionalities discussed separately below. Broadly, it has controls for running the program, and sets various preferences.

Help Links to a built-in manual describing the interface, the program syntax and listing the commands and conditionals available in the different operating modes.

The manual under the **Help** menu might be sufficient to learn BeeperBot. Note however, as with any computer language, there is a huge gap between learning the language elements and learning to effectively program in that language.

For the **File** and **World** menus, all files are saved as plain text, and can be viewed using most editors. However, you should not edit program or world files using editors such as Microsoft Word as these introduce formatting text that may cause BeeperBot to fail. Be sure to choose distinct names for your World and Program files: e.g. *exercise_1_world.txt* and *exercise_1_program.txt* so that they will not overwrite each other.

4.0.2 The Robot Menu

As stated above, the **Robot** menu has several purposes. Here we list the various items with a brief description of each.

Run

Step

Reset These have the same functionality as the corresponding buttons in the **World Display** pane described above; namely they run the current program.

0 - Slow

1 - Medium

2 - Fast

3 - Full Speed These buttons control the speed at which the code executes. **Slow** allows the user to follow the execution of most programs, but is very tedious for longer running pieces of code. **Fast** is probably too fast to follow what is happening with both the code and the robot in real time, but can give a view of what bits of code are being used the most. **Full Speed** does not update the display and does not trace the execution of the code, but instead runs the code until the program halts, and then displays the result.

Note that execution speed can also be modified by statements in the source program. This feature is handy for working with some of the larger programs in the notes.

Highlight coordinates This toggles on or off the green cross bars indicating the row and column where the robot is located in the **World Display** pane.

Show beeper dots This toggles whether or not a dot is displayed in the cells containing one or more beepers. If it is toggled off, then only the number of beepers is displayed in the cell.

Use large fonts This toggle affects the size of the font used for the text in the **Program**, **Initial World** and **Status** panes. The actual size used is set in the pop-up window activated by the **Settings** item in the **Robot** menu. This may be useful when displaying the program in a classroom setting for example.

Use large beeper dots When toggled on, instead of displaying a dot, this fills the entire cell with an ugly pinkish color. This is useful when trying to use BeeperBot to create simple graphic images. This toggle has no effect if **Show beeper dots** is turned off.

Settings Pops up a window with the following contents

Calls before... A text window that requires a number. The default is 5000, and this means that for long running programs every 5000 steps an alert window will pop up, asking if you wish to continue the program. This is useful if for example you are running a program at **Full Speed** and it happens to have an infinite loop. It is expected that most BB programs will not run this long.

Operating Mode with radio buttons **Primitive Mode, Standard Mode** and **Auxiliary Mode**. Primarily, these control the number of beeper bags that BB can use. In primitive mode, BB has no bags, in standard mode she has 1 bag, and in Auxiliary mode BB has 10 bags. See the manual under the **Help** menu for more information.

Large Font Size Use the scroll to select the font size to use when **Use large fonts** is toggled on. This does not affect the default font size.

Save After making settings, click this button. Note: until you click this save button, BeeperBot will not remember recently seen files, nor the toggles that are set in the **Robot** menu.

5 Programming BeeperBot

Before proceeding further do the following.

Start BeeperBot.

If BB does not display 10 bags in the **Status** pane, then do the following.

1. Under the **Robot** menu, scroll down and click **settings**

2. In the pop-up window, click the **Auxiliary** radio button.

3. Also in the pop-up window, click the **save** button.

Throughout these notes, unless stated otherwise, we will assume you are using Auxiliary mode. BeeperBot will remember to start in Auxiliary mode on subsequent startups once you save the settings as described.

5.1 Coding an Initial World

Figure 2 illustrates how the world file constructs the initial world for BeeperBot to start in.

Notice that there is already a wall running along the west side of the world, and another along the bottom edge. There are no comparable walls on the east or north, unless you specify them. Basically, BB can go as far east or north as it wishes, until your computer runs out of memory.

To be sure you understand, you should practice making a few arbitrary worlds using BeeperBot.

5.1.1 Tutorial Exercises

1. What happens if you try to make two (or more) robots, using two robot command lines?

2. What happens if you use the line

   ```
   robot 1 6 N 55
   ```

   to define the robot location? This is a legal way to define the initial location of the robot, but your task is to find out what the number 55 at the end of the line does.

5.2 Programming Language

While reading this part, you should have BeeperBot running with the manual in the **Help** menu item **Contents**. The **Help** menu item **Contents** has a more extensive description of the commands than is found here. This section focuses as much on using the BeeperBot tool and general programming ideas as details of the language.

First let us summarize the statements available in BeeperBot.

Here we organize the command list slightly differently. There are two general types of statements, *commands* that cause some action to happen and *conditionals* which test some condition and evaluate to *true* or *false* depending on whether the condition holds or not.

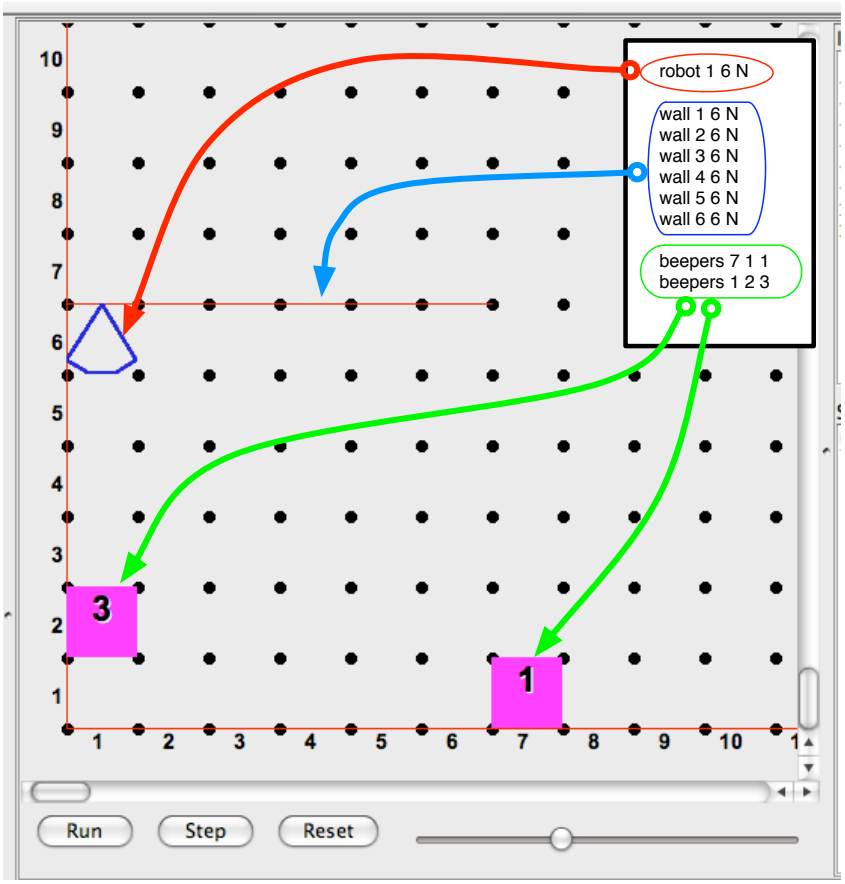


Figure 2: How the world file works: The robot command indicates the initial location of the robot, the two numbers indicating the column and row, the N indicating it should be facing north. The six wall commands place walls on the north side of row six, one in each column from 1 to 6. The other option is to place the walls on the east side of a cell using 'E'. The beepers command can place any number of beepers on any cell in the initial world.

5.2.1 Robot Control

These statements allow you to program movement of the robot, while avoiding walls.

Movement Commands *move* and *turn_left*.

Wall Checking Conditionals *front_is_clear*, *not_front_is_clear*, and similar for left and right

Compass Checking Conditionals *facing_north*, *not_facing_north*, and similar for east, south and west.

5.2.2 Working with Beepers

These statements allow you manipulate beepers, both in the world and in the beeper bags. Remember, BB has 10 beeper bags (in **auxiliary** mode) each of which can store any number of beepers. A cell in the world may also hold any number of beepers.

Manipulation Commands *create_beeper*, *destroy_beeper*, *pick_beeper*, *put_beeper*, *move_beeper(0,1)*. . . *move_beeper(9,0)*

Location Check Conditionals *next_to_a_beeper*, *not_ next_to_a_beeper*

Bag Check Conditionals *has_beeper(0)* . . . *has_beeper(9)*, and *not_has_beeper(0)* . . . *not_has_beeper(9)*. (Note: *any_beeper_in_beeper_bag* and its negation are equivalent to *has_beeper(0)* and its negation, and are included only for compatibility reasons.)

5.2.3 Program Control

These statements control the order in which the code is executed during a process. Note that *process* refers to the running of the code.

Code Control Statements *if*, *if . . . else*, *do*, *while*

Subprogram Definition *define*

Program Speed *set_speed*, *restore_speed*

Comments any line that begins with a # is treated as a comment, and ignored when the program is running.

We believe that the best way to become familiar with the concepts of running a program is to try a few examples. Check the BeeperBot **Help** menu for what these commands do.

6 Two Sample BeeperBot Programs

In order to show how BeeperBot is programmed, we present here two small programs. Text versions of these programs can be found in a Sample Programs[1], ready to be opened in BeeperBot after downloading.

6.0.4 Example One

Here is a simple program to outline a square using beepers. Note we include line numbers for reference, they are not part of the code. This part goes in the program pane, if you are typing it in.

```
 1 # Example Program CMPUT 101 Lec 1&2 Culberson
 2
 3 define main {
 4   do (4) {
 5      do (3) {
 6         create_beeper
 7         move
 8         # what happens if the next
 9         # turn_right is moved here
10      }
11      turn_right
12   }
13 }
14
15 define turn_right {
16   turn_left
17   turn_left
18   turn_left
19 }
```

For the initial world, type in

```
robot 6 6 N
```

Starting at location (6,6) will ensure that BB does not run into a wall.

You should run this code to see what it does, and trace the code by hand to be clear you understand each part.

Things to note: There are two activities going on when you run a BeeperBot program. First, the software in the BeeperBot tool is running a *process* defined by your program source code. Evidence of this activity can be observed in the sequence of yellow highlights of the source code.

Second, the robot is moving around its world, manipulating beepers. This activity is controlled by the execution of your program. You can see evidence

[1] `http://www.cs.ualberta.ca/resources-services/teaching-resources`

of this activity by observing the movements of the robot, and watching the changing values in the **Status** pane.

In order to understand programming, you must come to grips with these two notions of process, and how they relate to one another, or more specifically, how the processing of your code controls the actions of the robot.

When running the program, also note the iteration counter associated with each *do* in the **Program** pane. Check carefully how the counters are tested on the completion of each iteration. In particular, follow this code until you clearly understand how the nesting works. Take the hint from the comments, and change the program to see how it changes the behavior of the robot when the *turn_right* command is moved inside the inner loop.

6.0.5 Example Two

```
# Second Example in the notes
# draws a spiral with length based on initial pile
define get_all {
    while (next_to_a_beeper) {
        pick_beeper
    }
}

define make_side {
    while (has_beeper(0)) {
        move_beeper(0,1)
        create_beeper
        move
    }
}

define restore_bag {
    while (has_beeper(1)) {
        move_beeper(1,0)
    }
    # reduce the number by one
    put_beeper
    destroy_beeper
}

define main {
    get_all
    while (has_beeper(0)){
        make_side
        restore_bag
        turn_left
    }
```

```
}
```

This program can use the following initial world description.

```
robot 7 7 E
beepers 7 7 10
```

This program draws a spiral. You may wish to turn on the beeper dots option in the **Robot** menu to get a better visual effect.

Note that in this program we have chosen to define the various subroutines before the *define main* line. Nevertheless, the program knows to start at the correct line, since it is *main*.

Note the use of beeper bags 0 and 1 to keep track of how many beepers are left. We use the create_beeper command to make copies, scattering them along one per location.

You are invited to try your hand at making other designs using BB.

7 BeeperBot History

BeeperBot is a program written as a CMPUT 401 course project at the University of Alberta in the winter of 2008. Dr. Ken Wong was the course instructor, and the team members were Matthew Johnson, Timothy Lam, Mark Nicoll, Adrienne Paton, and Matthew Whitton. Joe Culberson was the client, generating the project proposal as an expansion of the basic concepts of similar beeper manipulating robots such as Karel the Robot [2], Guido van Robot (GvR) [3], and RUR-PLE[4]. So, BeeperBot is Yet Another Beeper Robot, but we did not like YABR as an acronym.

BeeperBot has three operating modes: **primitive, standard** and **auxiliary**. As the names suggest, these are related to the standard operation of GvR, Karel etc. with standard mode being the most similar.

Even in **standard** mode, there are some differences in the syntax of the programs, and BB has two additional commands, *create_beeper* and *destroy_beeper* which operate on the set of beepers in the room in which BeeperBot is standing. These make it easier to emulate a Turing-equivalent model of computation. The syntax of BeeperBot uses braces for code blocks, and parentheses to delineate conditionals. This may make the transition to languages such as Java or C/C++ easier for students.

In **primitive** mode, BeeperBot has no beeper bag, and all commands related to the beeper bag(s) are disabled. In addition, the do command is disabled. It is not too hard to see that BeeperBot **primitive** is nevertheless still computationally complete. Code for this mode is also (except for recursion) easily represented within a simple class of structured finite state machines. Thus, this should be ideal for teaching concepts of state, both within code, and the state (configuration) of the world.

Auxiliary mode gives BeeperBot a total of 10 beeper bags, plus operators to move beepers between bags, and to test whether any bag is empty. All transfers to/from the world must go through bag 0, the default bag available in standard mode. Together with the in-code controls on program execution speed available in all modes, the hope is to illustrate concepts such as abstraction, memory organization, variables and other algorithmic concepts as typically implemented on a RAM/RASP[5] von Neumann architecture[6], without the necessity of introducing a high level language and all the attendant issues of syntax and compilation that that entails.

[2] `http://karel.sourceforge.net/`
[3] `http://gvr.sourceforge.net/`
[4] `http://rur-ple.sourceforge.net/`
[5] `http://en.wikipedia.org/wiki/Random_access_stored_program_machine`
[6] `http://en.wikipedia.org/wiki/Von_Neumann`