CS4 Homework 0

Contents

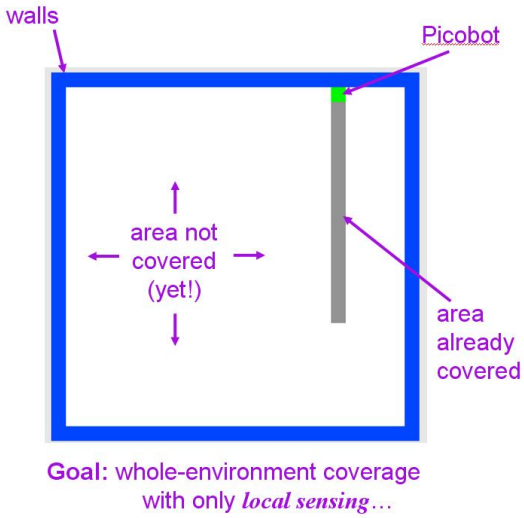
0 Overview

Due **Noon, Friday, Feb 3, 2017**. You should begin the first part of this homework assignment, Picobot, now. The remaining portions of the assignment require you to come to a Setup Section, which you can sign up for here **https://goo.gl/forms/ejsfntnNGJWryOV22** If you complete the Picobot problems before your Setup Section, you will be able to turn-in this entire homework by the end of the Setup Section.

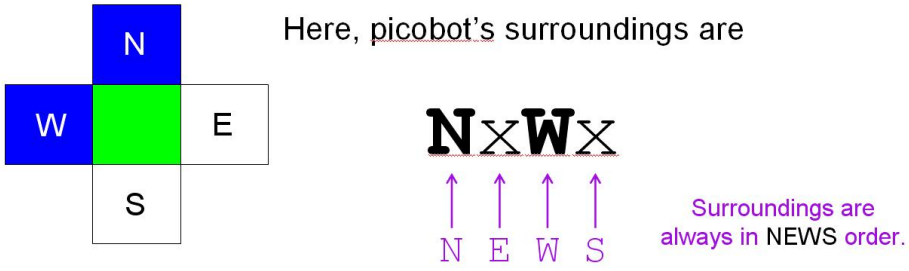
1 Picobot

The Picobot simulator **http://cs.brown.edu/courses/cs004/picobot/** allows you create and test Picobot programs.

Recall that picobot can only sense its immediate surroundings. It starts at a random location in a selected room and you must write rules to tell it which direction it should go in certain circumstances.



Picobot senses its surroundings and represents them like this:



An "N", "E", "W", or "S" means that Picobot is blocked in that direction, whereas an "x" means that Picobot could move in that direction.

Picobot also has an attribute called a state. It is a single value from 0 to 99 available to it as memory. In general, "state" refers to the relevant context in which computation takes place. Here, you might think of each "state" as one piece or behavior that the robot uses to achieve its overall goal. Picobot always begins in state 0. The state and the surroundings are all the information that Picobot has available to make its decisions!

So what does a rule look like?

```
CurrentState   Surroundings   ->   MoveDirection   NewState
```

For example,

```
0 xxxS   ->   N   0
```

is a rule that says "if picobot starts in state 0 and sees the surroundings xxxS, it should move North and stay in state 0."

The asterisk (*) can be used inside surroundings to mean "I don't care whether there is a wall or not in that position." For example, `xE**` means "there is no wall North, there is a wall to the East, and there may or may not be a wall to the West or South."
As an example, the rule

```
0   x***   ->   N   0
```

is a rule that says "if picobot starts in state 0 and sees any surroundings without a wall to the North, it should move North and stay in state 0."

1.1 Problem 1A

Develop a set of rules that cause Picobot to completely cover an empty square room, *regardless* of where it starts within the room. (The empty square room is the one that comes up when you first open the Picobot simulator page.) You can use the Reset and Teleport buttons to try different starting positions. Be sure to test special starting positions (e.g., some place along each edge, and in each corner).

You may find it helpful to review the material on Picobot in Chapter 1 of the textbook and in the lecture notes. Be sure to review this courses Collaboration Policy before you talk to any other students about this problem, or try googling a solution (hint: the former is encouraged, up to a point, while the later is strictly forbidden).

http://cs.brown.edu/courses/csci0040/CS4_Collaboration_Policy.pdf

Open the Picobot simulator, and modify/replace the existing rules with your own rules until you can get it to work. Whenever you modify the rules, be sure to click the Enter rules for Picobot button before you click Go.

As discussed in lecture, you should avoid repeat rules – two or more rules that would both be triggered by a given combination of state and surroundings.

We encourage you to include comments that describe your rules. For Picobot programs, these

comments should be preceded by a hashtag # symbol. For example,

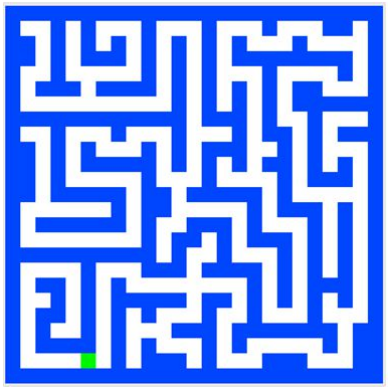
```
# state 0 with nothing N: go one step N
0 x*** -> N 0

# state 0 with something to the N: go W and into state 1
# ** This will crash if picobot has a wall to the W! **
0 N*** -> W 1
```

Optional Extra Challenge (may be difficult, and not required for full credit): At the heart of computer science are questions of complexity and efficiency: questions about how simple, fast, etc. the solution to a problem can be. For an optional challenge, see if you can create a solution for problem 1A that uses only 6 rules!

1.2 Problem 1B

Develop a set of rules that enable Picobot to completely navigate a connected maze with corridors one square wide, *regardless* of its starting point. You can obtain an example of such a maze by opening the Picobot simulator and clicking on the right-arrow (–>) button below the lower right-hand corner of the room, which will show you this maze:



Modify/replace the existing rules with your own rules until you can get it to work. Whenever you modify the rules, be sure to click the Enter rules for Picobot button before you click Go.

As discussed in lecture, you should avoid repeat rules – two or more rules that would both be triggered by a given combination of state and surroundings.

Hint: We discussed this problem in lecture (use Piazza to access the slides). In particular, we recommended:

Use a right-hand rule having four states – one for each direction in which the robot could be facing starting with at least three rules for each state: one for when the wall is on your right and you are able to move, one for when you "lose the wall," and one or two rules for hitting a dead end – i.e., for when you can no longer make progress in the current direction because there's a wall in front of you.

Optional Extra Challenge (may be difficult, and not required for full credit): See you if you can create a solution for problem 1B that uses only 8 rules. This is extremely challenging, and we don't recommend starting with only 8 rules! Rather, begin by having at least 3 rules for each state as discussed in lecture, and then see if you combine rules in some way.

1.3 Record your Answers

When you are satisfied with your rules for Problem 1A and Problem 1B, copy them into an email to yourself. When you are at the Setup Section, or sometime later, you can use this email while you are at a CS department computer to turn-in your solution to this problem. Soon you'll have the option to work remotely, but for now this is the easiest approach.

2 Setting up for Homework

If you have not yet attended a Setup Section, stop here since you will need a CS account login and an understanding of the department machines to continue. If the Setup Sections have already completed, please email the course Head TAs via cs0040headtas@lists.brown.edu in order to setup a one-on-one meeting to go over the Setup Section material and to receive a CS account login.

To setup your CS account to submit homework to CS4 for you will need to run `cs4_setup` from a terminal window on a CS department machine - this adds a course/cs0040 folder to your CS department home directory (you only have to run this once at the beginning of the course).

For each homework assignment, there may be support files that you will need. After each homework assignment is released, run the `cs4_install hw<xx>` command (again from a CS account terminal) to copy the any support files related to the assignment to it's associated folder, course/cs0040/homeworks/hw<xx>. For this homework, type:

```
cs4_install hw00
```

There should now be a `hw00` folder within your `cs0040/homeworks` directory. For the next section, you will need to navigate to the `hw00` folder with the `cd` command:

```
cd ~/course/cs0040/homeworks/hw00
```

3 Welcome to Python

Now, open a terminal and start Python by running the `python3` command. This will take you to the Python shell and the `>>>` prompt indicates that the shell is waiting for you to type your commands.

3.1 Arithmetic in Python

You can use Python as a calculator! Run the following commands and check out the results of each operator.

```
>>> 5 + 6
>>> 4.2 - 5.1
>>> 10 * 20
>>> 2 ** 4
>>> 5 / 3
>>> 5 // 3
>>> 20 % 6
```

Note the difference between / and // division.

3.2 Variables

Variables allow us to store values for later use. Type the following commands into your Python shell:

```
>>> x = 4
>>> y = 5
>>> z = x * y
```

`x`, `y` and `z` are all variables that store certain values. The value of `x` is 4 and the value of `y` is 5. Note the value of `z`.

3.3 Assignment Statements

Assignment statements store a value in a variable. In the statement `temp = 4` above, 4 is stored in the variable `temp`. The sign "=" is called *assignment operator*.

Assignment statements follow these general steps:

1. evaluate the expression on the right hand side of the =
2. assign the resulting value to the variable on the left hand side of the =

Let's walk through the following example.

```
>>> a = 12
>>> b = 30
>>> a = a + b
>>> b = b * a
```

What are the values of `a` and `b` after each assignment?

Note that a variable can appear on both sides of the assignment operator.

3.4 Expressions

Expressions are combinations of values, variables, and operators. The following lines are both examples of expressions.

```
>>> temp = 4
>>> res = (temp-2) * (5/2)
```

Check out the value of `res`.

3.5 Strings

In addition to numbers, Python also has strings. A string is a sequence of characters and/or symbols. We identify strings by ' ' or " " around the sequence (in this class, we will use " ").

Here are some examples of strings:

```
>>> c = "Hello!"
>>> c
'Hello'
>>> c = "This is a string!"
>>> c
'This is a string!'
```

3.6 Print Statements

Print statements display one or more values to the terminal. The basic syntax is

```
print(expr)
print(expr_1, expr_2, expr_3)
```

where each `expr` is an expression.

Let's look at these examples:

```
>>>print("The results are:", 15 + 5, 15 - 5)
The results are: 20 10
>>>cents = 89
>>>print("You have", cents, "cents")
You have 89 cents
```

Note that print omits the quotes around the strings.

3.7 Program Flow in Python

A program in Python is simply a series of commands executed sequentially from top to bottom. This means that the first line of code will evaluate before the second, and so on.

An example program might look like:

```
total = 0
num1 = 5
num2 = 10
total = num1 + num2
```

3.8 Write your First Python Program

There are many environments to write our code in. Some popular text editors (i.e. plain-text file editors) are Atom, Gedit, and Sublime Text. In this course we recommend the use of Atom, but feel free to use whatever plain-text editor you are most comfortable with. Atom is already installed on the department machines; to set up Atom (and Python) on your personal machine, follow the instructions in this Installation Guide
http://cs.brown.edu/courses/cs004/CS4_Python_Installation_Guide.pdf

To open a file with Atom on a department machine, navigate to the directory containing the file using the `cd` command and use Atom to open the file by typing `atom <filename> &`.

Note: The `&` allows Atom to run in the background, so we are still free to use our terminal.

Open a terminal on a department machine and navigate to the `hw00` directory and use Atom to open `hw00pr02.py`. The commands to do this are shown below:

```
cd ~/course/cs0040/homeworks/hw00
atom hw00pr02.py &
```

Add the following code to your file. Make sure to replace `<your_name>` with your name.

```
name = "<your_name>"
print ("Hello", name)
```

Now we will save this file with `ctrl+S`.

Finally, let's go back to the terminal and run our first Python program! Run the following line:

```
python3 hw00pr02.py
```

Yay! You just wrote your first Python program!

4 Add your Picobot Solution

Open the `hw00pr01.txt` file that was also placed in the hw00 directory. This naming convention, (`hw##pr##`), will be used throughout the course for homework problems. The command to open this file is below:

```
atom hw00pr01.txt &
```

When the file opens, paste your solutions for the two maps in their respective sections (remember that email from earlier?) and save the file.

5 Hand in this Homework

At this point, your CS account has been set up, and your solutions should be complete. From a terminal on a CS department machine and navigate to your `hw00` directory and check it's contents by running:

```
cd ~/course/cs0040/homeworks/hw00
ls
```

Are both of your homework solution files there? i.e., you should see `hw00pr01.txt` and `hw00pr02.py` listed.

In order to turn in your homework assignments you need to run the `cs4_handin hw<xx>` command from a CS account terminal. When you are ready to hand in this assignment run

```
cs4_handin hw00
```

When this command is run the entire contents of your `~/course/cs0040/homeworks/hw00` directory is handed in.

Whenever you turn in an assignment using the `cs4_handin` command, you will receive an automatic confirmation email. Once you receive it, you know that you have successfully submitted an assignment.

If you make additional changes to your solution(s) running the `cs4_handin hw<xx>` command again will resubmit everything in hw<xx>. You can do this at anytime up to the assignment deadline. You are scored based on the date of your last hand-in, so be careful about resubmitting after an assignment deadline has passed.