CSCI 1320 Computer Science I: Engineering Applications
Instructor: Zagrodzki
Assignment 0
Due Sunday, September 2nd, by 6:00 pm

Objectives: this assignment is meant to give you a first glimpse into programming

- Break a problem down into specific sub-problems
- Write an algorithm to solve a specific problem, and then translate that algorithm into a program in a specific programming language (in this case, the Picobot language)
- Write clear, concise documentation for every step of your algorithm
- Develop test cases that reveal programming bugs. Fix the bugs then test again

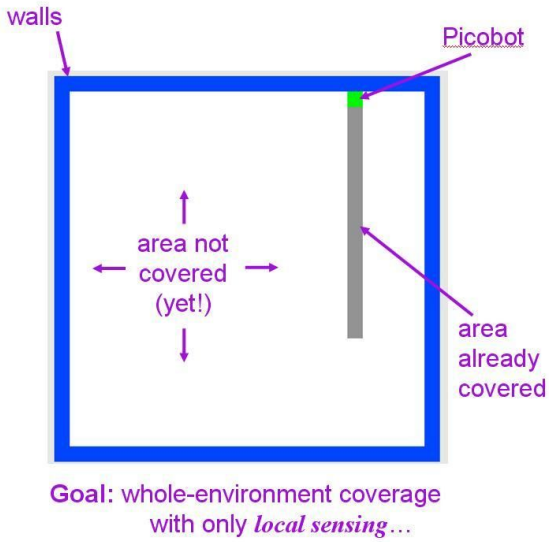
Picobot

Introduction to Picobot

Here is a link to the Picobot page. (http://www.cs.hmc.edu/picobot/)
This problem explores a simple "robot," named "picobot," whose goal is to completely traverse its environment. An analogy in the real world today would be the vacuuming robot Roomba, who needs to navigate around obstacles and/or furniture in a room.

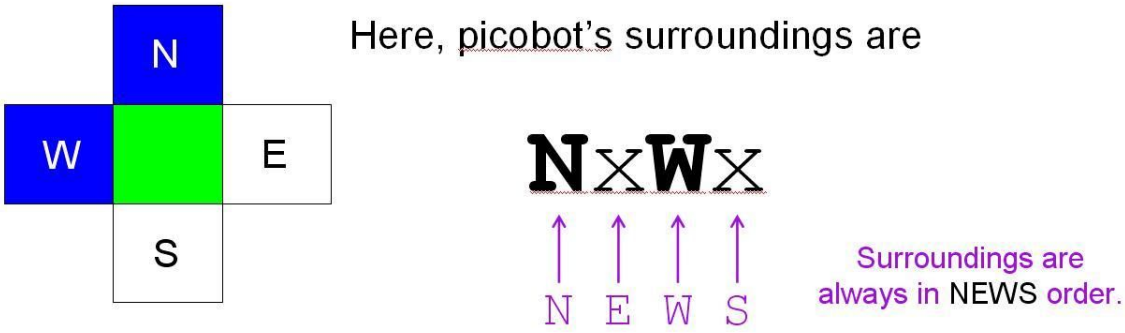
Picobot starts at a *random* location in a room -- you don't have control over Picobot's initial location. The walls of the room are blue; picobot is green, and the empty area is white. Each time picobot takes a step, it leaves a grey trail behind it. When Picobot has completely explored its environment, it stops automatically.

Picobot overview:



Surroundings
Not surprisingly, picobot has limited sensing power. It can only sense its surroundings immediately to the north, east, west, and south of it.

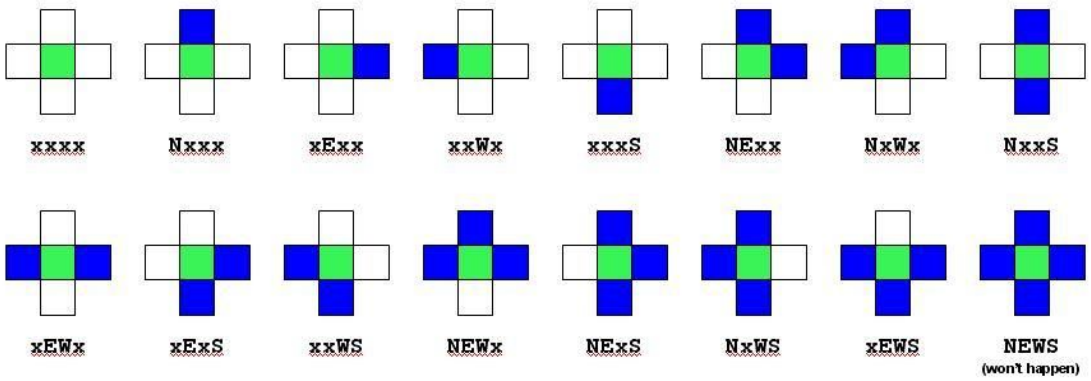
Example picobot surroundings:



In the above image, Picobot sees a wall to the north and west and it sees nothing to the west or south. This set of surroundings would be represented as follows:

NxWx

The four squares surrounding picobot are always considered in NEWS order: an x represents empty space and the appropriate direction letter (N, E, W, and S) represents a wall blocking that direction. Here are all of the possible picobot surroundings:



State
Picobot's memory is also limited. In fact, it has only a single value from 0 to 99 available to it. This number is called picobot's ***state***. 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. It state has its own separate set of rules.
Picobot always begins in state 0.
The state and the surroundings are all the information that picobot has available to make its decisions!

Rules
Picobot moves according to a set of rules of the form

StateNow 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 (so no walls anywhere except to the South), it should move North and stay in state 0."

The *Move Direction* can be N, E, W, S, or X, representing the direction to move or, in the case of X, the choice not to move at all.
If this were picobot's only rule and if picobot began (in state 0) at the bottom of an empty room, it would move up (north) one square and stay in state 0.
However, **picobot would not move any further**, because its surroundings would have changed to xxxx, which does not match the rule above.

Wildcards
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."
If this new version (with wildcard asterisks) were picobot's only rule and if picobot began (in state 0) at the bottom of an empty room, it would first see surroundings xxxS. These match the above rule, so picobot would move North and stay in state 0. Then, its surroundings would be xxxx. These ***also*** match the above rule, so picobot would again move North and stay in state 0. In fact, this process would continue until it hit the "top" of the room, when the surroundings Nxxx no longer match the above rule.

Comments
Anything after the pound sign (#) on a line is a comment. Comments are human-readable explanations of what is going on, but ignored by picobot. Blank lines are ignored as well.

An example
Consider the following set of rules:

```
# state 0 goes N as far as possible
0 x*** -> N 0   # if there's nothing to the N, go N
0 N*** -> X 1   # if N is blocked, switch to state 1

# state 1 goes S as far as possible
1 ***x -> S 1   # if there's nothing to the S, go S
1 ***S -> X 0   # otherwise, switch to state 0
```

Recall that picobot always starts in state 0. Picobot now consults the rules from top to bottom until it finds the first rule that applies. It uses that rule to make its move and enter its next state. It then starts all over again, looking at the rules and finding the first one from the top that applies.
In this case, picobot will follow the first rule up to the "top" of its environment, moving north and staying in state 0 the whole time. Eventually, it encounters a wall to its north. At this point, the topmost rule no longer applies. However, the next rule "0 N*** -> X 1" does apply now! So, picobot uses this rule, which causes it to stay put (due to the "X") and ***switch to state 1***. Now that it is in state 1, neither of the first two rules will apply. Picobot follows state 1's rules, which guide it back to the "bottom" of its environment. And so it continues... .

Rule examples:

1. If no wall to the North, move North regardless of E, W and S surroundings. Remain in state 0.

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

2. If no wall to the North, but a wall to the E, move North regardless of W and S surroundings. Remain in state 0.

```
0  xE**  ->  N  0
```

3. Regardless of E, W and S surroundings, if there is a wall to the North, don't move. Change from state 0 to state 1

```
0  N***  ->  X  1
```

4. Regardless of E surroundings, if there are walls to the North, West and South, move East. Change from state 1 to state 2

```
1  N*WS  ->  E  2
```

Overlap

Sometimes, two rules you have entered overlap. That is, one rule is already covered by an already input rule. In this case, when you click "Enter rules for Picobot", the program returns comments about rule overlap. For example:

```
Oops...

On line number 22, which is this:
  3 **W* -> S 1   # go S and switch to state 1

Repeat Rule! The state: 3  surr: xxWx
  was already handled on line #19
  which reads as follows:
  3 ***X -> S 3
```

Use the comments to adjust your strategy and your rules.

The assignment

For this assignment, your task is to design a set of rules that will allow picobot to completely navigate an **empty diamond shaped room**.

Remember to click on the "Enter rules for Picobot" before you try to run picobot. Remember also to save your solution once you've got it working! You will paste the solution inside a text document and then upload that document to Moodle. Remember that your solutions must work from arbitrary starting positions within the environment.

Important: algorithm/pseudocode

Start by formulating your strategy. Write comments for each step/rule of the algorithm. If you are having trouble translating rules into code, ask for help. TAs and CAs are instructed to help you translate at the beginning, if you're having trouble with the syntax. Your submission MUST include comments for EACH rule; basically, explain every step of your strategy.

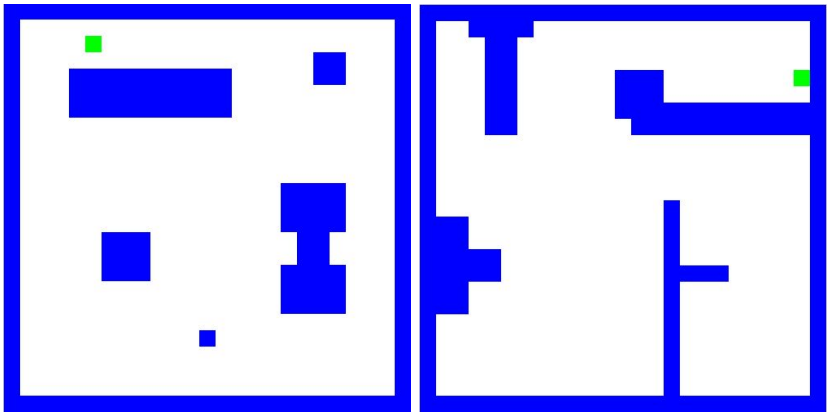
Optional extra credit

At heart, Computer Science fundamentally tries to answer questions of complexity: to show that problems are easier than initially thought -- or, sometimes, to prove that they ***can't*** be handled with fewer resources.

You might think about how *efficient* your solutions are -- both in terms of the number of states used and in terms of the number of rules. There are other ways to measure efficiency as well (e.g., speed).

1. For optional extra credit, try to create as efficient a solution as possible for the diamond room. That is, strive to use as few rules as possible in creating your solution - so long as your solution will still succeed from any starting point in the environment! 10% extra credit will be awarded for solutions that match - or beat - **11 rules** for the diamond room.

2. Additional 20% extra credit is available for figuring out rules for two other Picobot maps (10% for each room).



Grading Rubric

Criteria	Pts
Program correctness	
Diamond room	60
Does not work for every starting location	-30
Good programming practice:	
Appropriate comments for every rule	20
Can understand & explain solution/algorithm	
Diamond room	20
Totals	**100**
Extra credit	
Efficient diamond room (<= 11 rules)	10
Column Room	10
Stalactite Room	10
Final Grade (max possible)	130

Submitting the assignment:
For each problem/map solved, copy and paste your **very well commented** code into a text file (extension .txt).
Submit the solution file <firstName_LastName.txt> through Moodle as Assignment 0. Include at the top of your document your name, student ID, assignment number, and course instructor.

Note: Another useful thing to do: after you get your text file, try to see if you can copy the text and then paste it back into the Picobot page. If the copy/paste operation goes well then you're fine (because this is what you'll be doing in the interview next week.)