Solving Problems in CS

CS 100: Introduction to the Profession
Michael Saelee; saele@iit.edu
Course website:

http://moss.cs.iit.edu/cs100
Day 2 takeaway: CS is about *solving problems*
- What problems?
- How to represent these problems?
- How efficiently can we solve them?
- Is it possible to solve them?
What problems?

- *theoretical CS*: mathematical/abstract
  - e.g., if we can verify the solution to a given problem quickly, can we also solve it quickly? (aka P=NP?)

- *applied CS*: real-world problems
  - e.g., can we get a robot to reliably explore every corner of a room?
How to *represent* these problems?

- i.e., so that a computer can be used to can automate their solutions

- typically many possible representations

- choice may be limited by real-world constraints!
Picobot implementation by, and figures adapted from Zach Dodds and Wynn Vonnegut, Harvey Mudd College
What does the bot need to sense/remember in order to fully explore the room?
So many options!

- our goal: only *local sensing*
- i.e., immediate surroundings only
- minimizes hardware cost
Picobot can only sense things directly to the N, E, W, and S

E.g., N and W are obstructed, while E and S are clear
How many possible surroundings?

Answer: $2^4 = 16$
In addition to its sensors, Picobot also has:

1. knowledge of its current state

2. a list of rules (its program) dictating movement and state transitions based on its current state and surroundings
State

Picobot's memory is a single number, called its state.

State is the *internal context* of computation.

Picobot always starts in state 0.

I am in state 0. My surroundings are xxWS.
Picobot moves according to a set of rules:

<table>
<thead>
<tr>
<th>state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xxWS</td>
<td>N</td>
<td>0</td>
</tr>
</tbody>
</table>

If I'm in state 0 seeing xxWS, then I move North, and change to state 0.
Wildcards

Asterisks * are wild cards. They match walls or empty space:

I am in state 0. My surroundings are xxWS.

Aha! This matches x***

<table>
<thead>
<tr>
<th>state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x***</td>
<td>N</td>
<td>0</td>
</tr>
</tbody>
</table>

here, EWS may be wall or empty space
What will this set of rules do to Picobot?

<table>
<thead>
<tr>
<th>state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x***</td>
<td>-&gt;</td>
<td>N</td>
</tr>
<tr>
<td>0</td>
<td>N***</td>
<td>-&gt;</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>***x</td>
<td>-&gt;</td>
<td>S</td>
</tr>
<tr>
<td>1</td>
<td>***S</td>
<td>-&gt;</td>
<td>X</td>
</tr>
</tbody>
</table>

Picobot checks its rules from the top each time.

When it finds a matching rule, that rule runs.

Only one rule is allowed per state and surroundings.
How efficiently can we solve them?

- how long does it take a computer to solve increasingly harder versions of the same problem?
- What is the *minimum number* of states and rules to traverse each room?

*our best: 3 states, 7 rules*  
*our best: 4 states, 8 rules*
Is it possible to solve them?

- given a problem and its chosen representation, can we automatically solve every instance of it in finite time?
- Can you write a set of rules to traverse every possible room?

- How would you measure the complexity of a given room?
Have fun!

When you’ve solved the first two rooms, create a “secret gist” containing your rules and submit it via the form on the website.

Try solving the harder rooms, if you want!