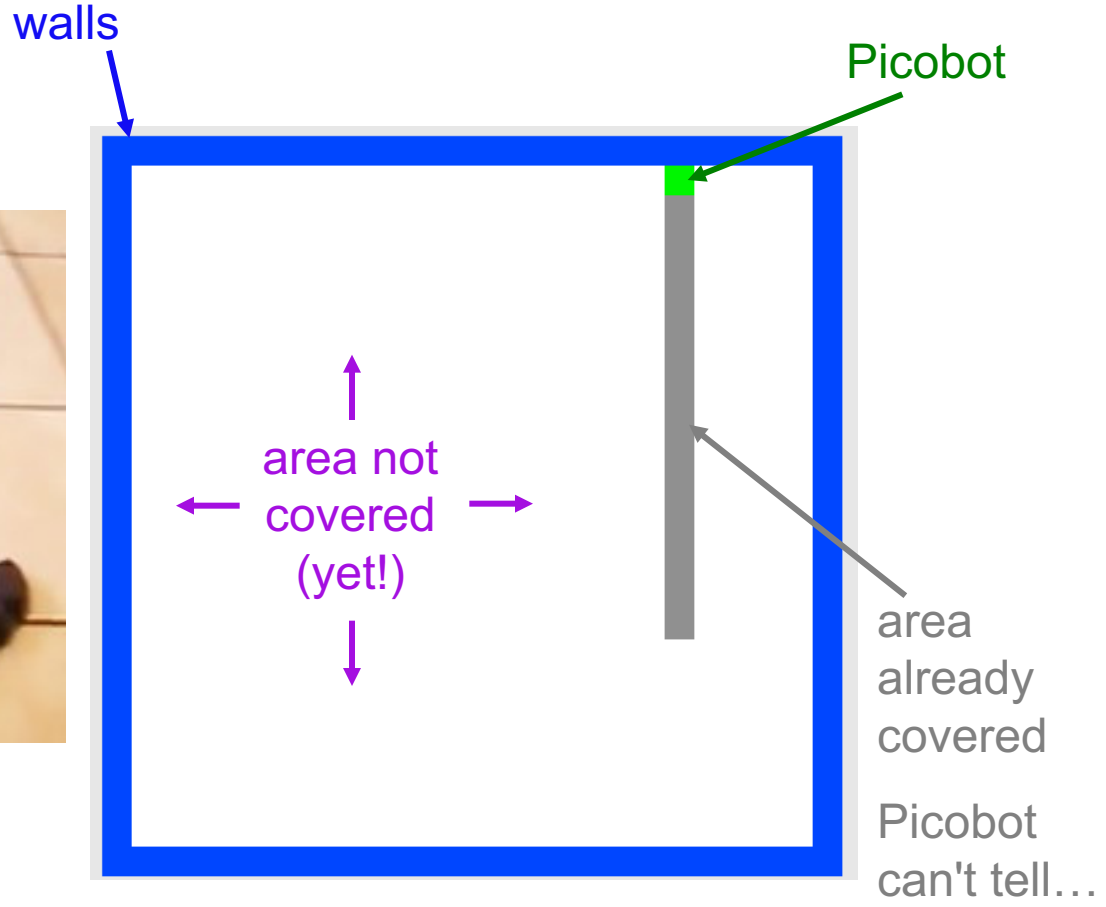
Picobot!

Goal: full-room coverage with only ***local sensing***…



The Roomba!

can't tell "vacuumed" from "unvacuumed" area



Let's see it!

Another language *already*?

Python

General-purpose language

you might see 50% by the end of camp

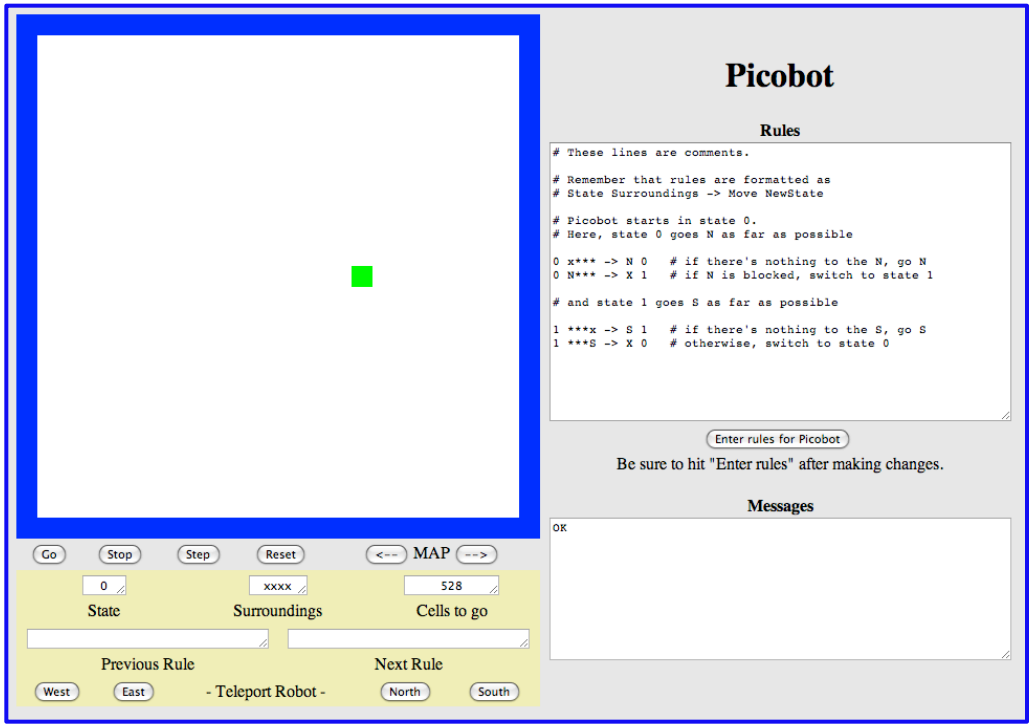
even then, <1% of its libraries!

Picobot

Special-purpose language

you'll see 100% in the next 15 minutes

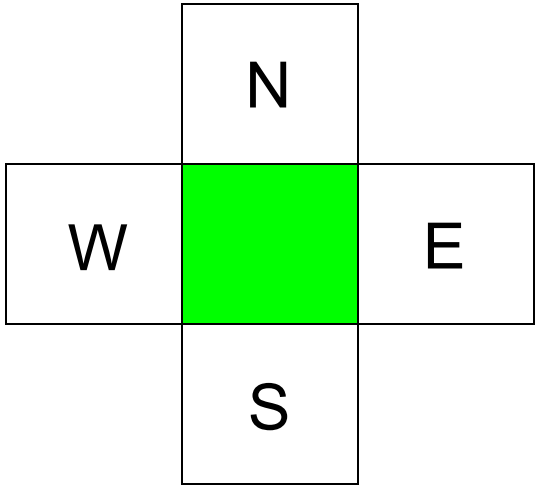
Picobot!



The Picobot simulator

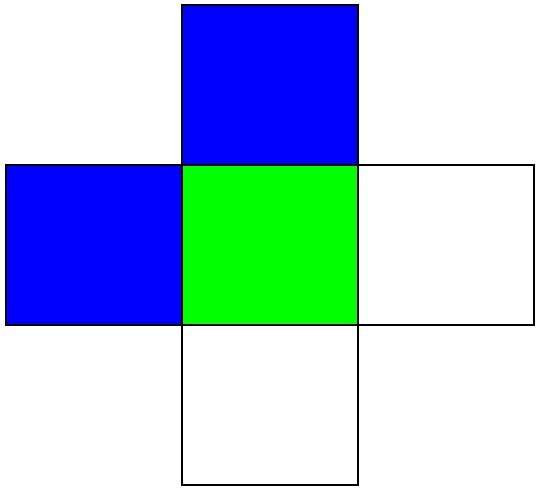
`cscamp.cs.vassar.edu/picobot`

Surroundings



Picobot can only sense things directly to the N, E, W, and S

For example, here its surroundings are



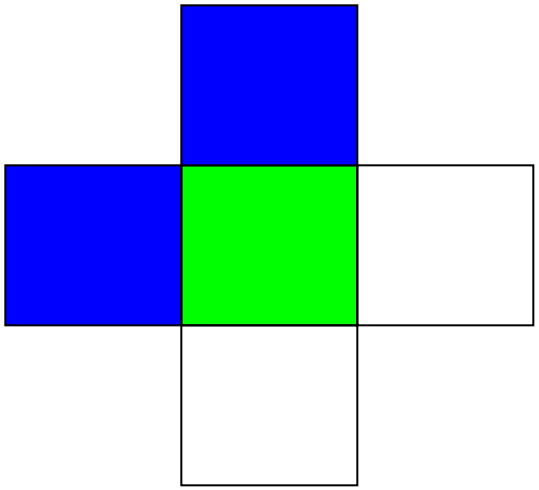
NxWx

N E W S

Surroundings are always in NEWS order.

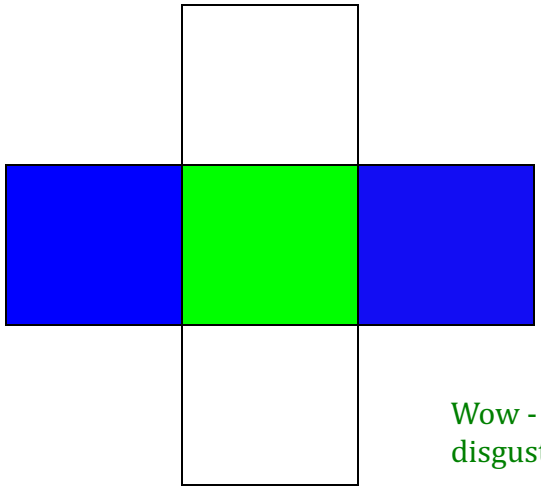
What are these surroundings?

Surroundings are always in NEWS order.



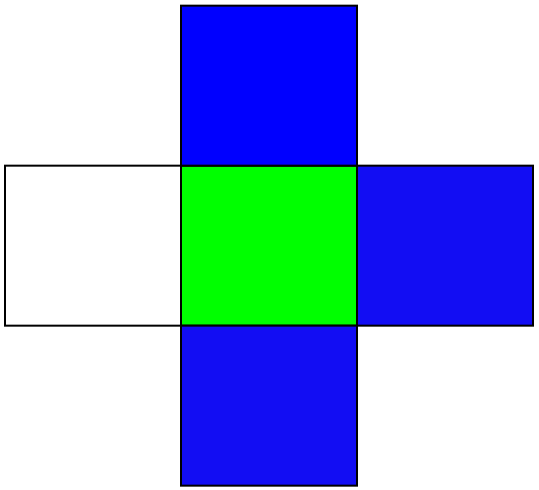
N E W S

NxWx

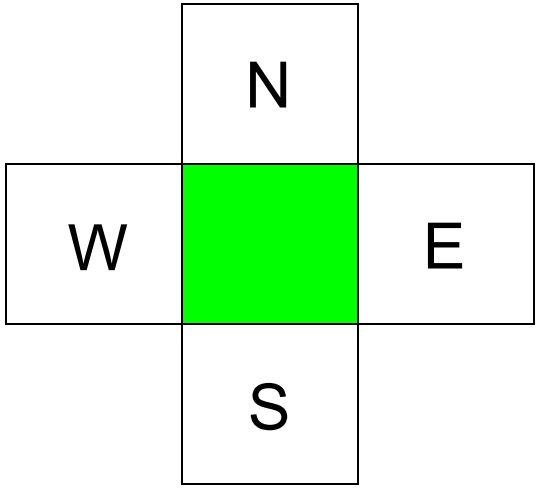


Wow - this one is disgusting!



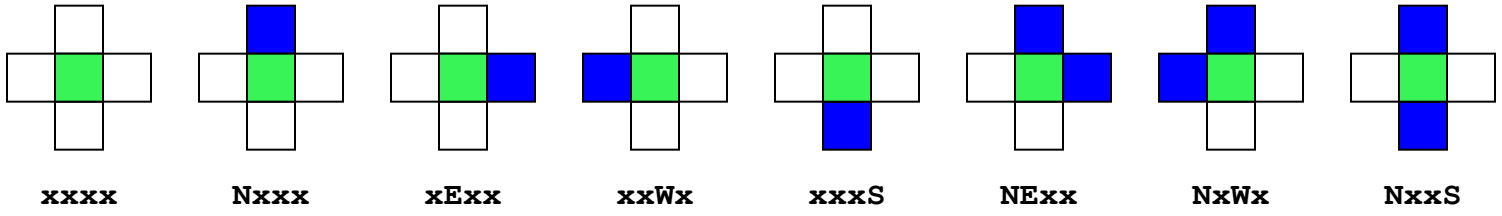


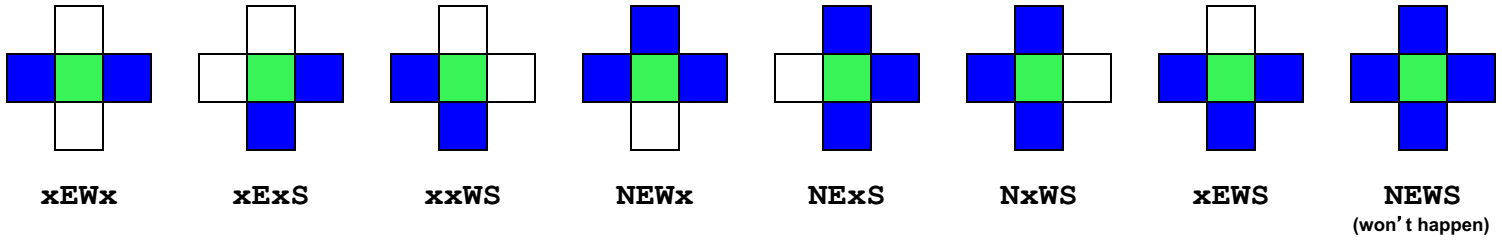
Surroundings



How many distinct surroundings are there?

2^4 == 16 possible

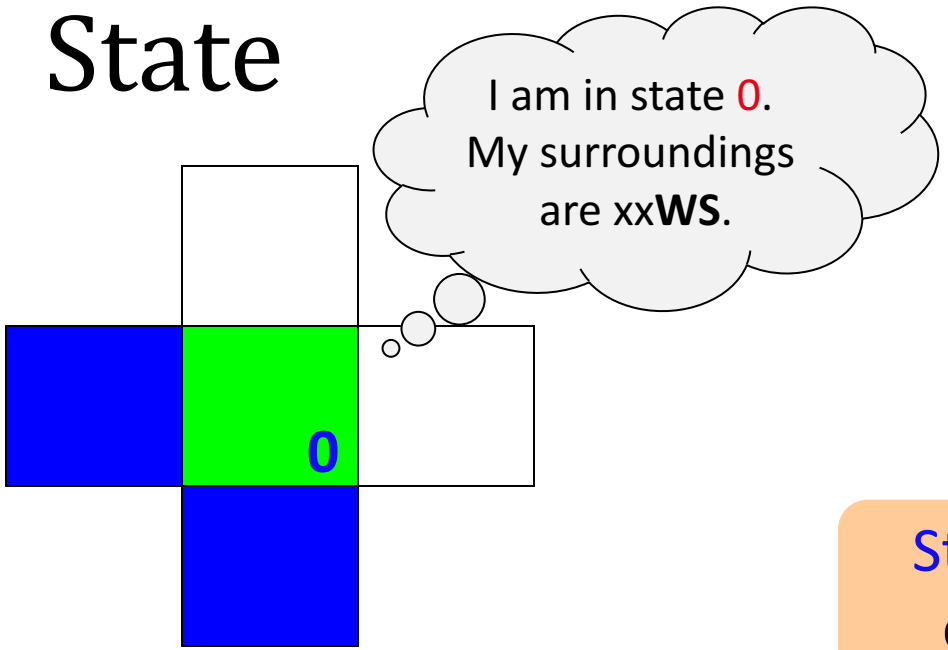




Aargh!



State



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

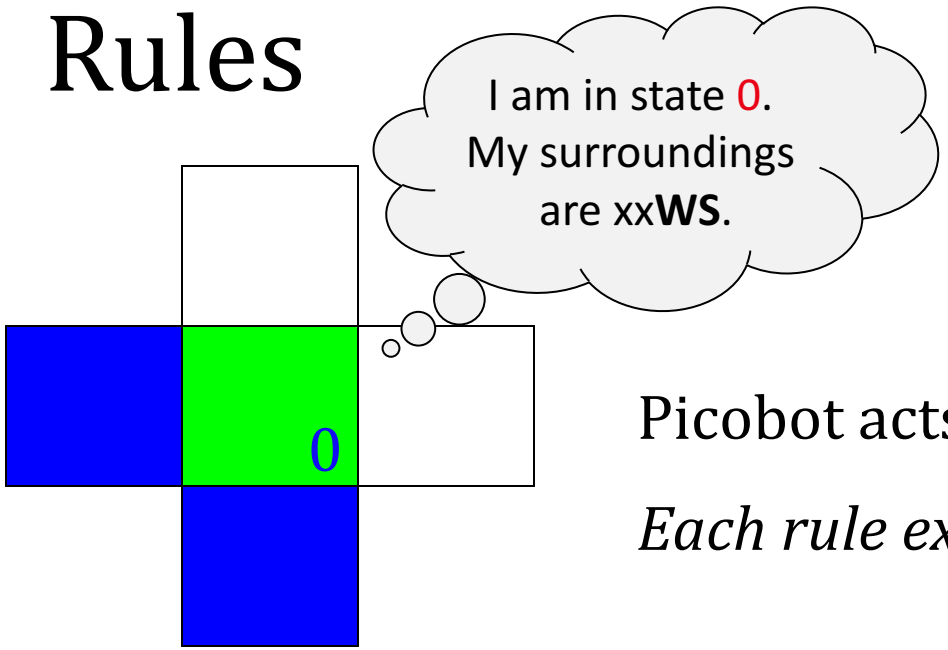
State is the *internal context* of a computation, i.e., its *subtask*.

Picobot always starts in state 0.

State and **`surroundings`** represent everything Picobot knows about the world

self-contained
but not simplistic

Rules



Picobot acts through a **set of rules**

*Each rule expresses **your intent** for Picobot!*

current state	surroundings		direction	new state
0	**xxWS**	→	**N**	**0**

syntax

If Picobot's in state **0** *seeing* **xxWS**,

Then move **N***orth, and "change" to state* **0**.

semantics

Picobot programming ~ ***rules***

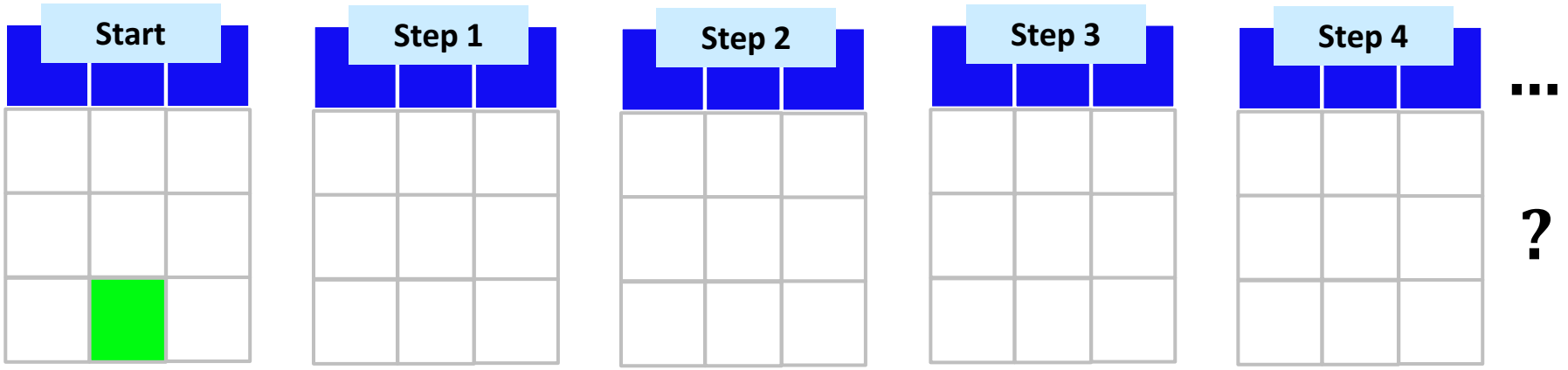
	current state	surroundings		direction	new state
rule (A)	0	Nxxx	→	S	0
rule (B)	0	xxxx	→	N	0

these two rules are a complete Picobot program

Notes

Picobot checks its rules from the top each time.

When it finds a matching rule, that rule runs.



Picobot programming ~ ***rules***

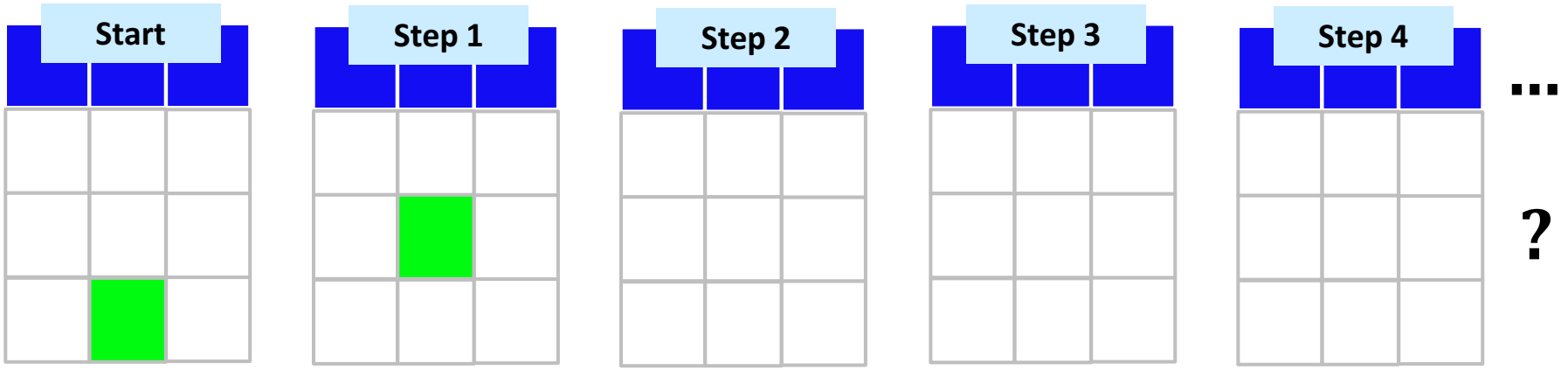
	current state	surroundings		direction	new state
rule (A)	0	Nxxx	→	S	0
rule (B)	0	xxxx	→	N	0

these two rules are a complete Picobot program

Notes

Picobot checks its rules from the top each time.

When it finds a matching rule, that rule runs.



Picobot programming ~ ***rules***

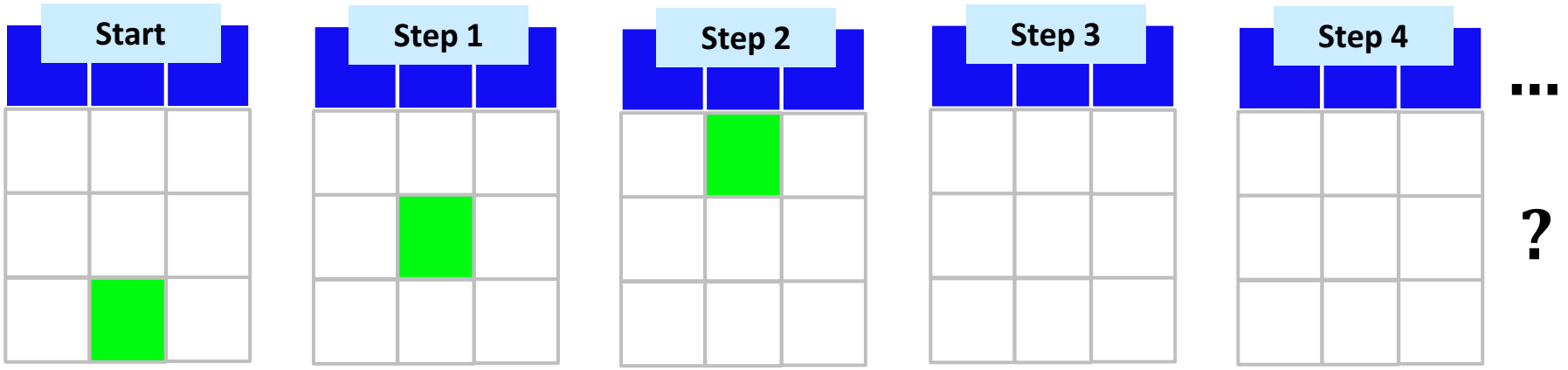
	current state	surroundings		direction	new state
rule (A)	0	Nxxx	→	S	0
rule (B)	0	xxxx	→	N	0

these two rules are a complete Picobot program

Notes

Picobot checks its rules from the top each time.

When it finds a matching rule, that rule runs.



Picobot programming ~ ***rules***

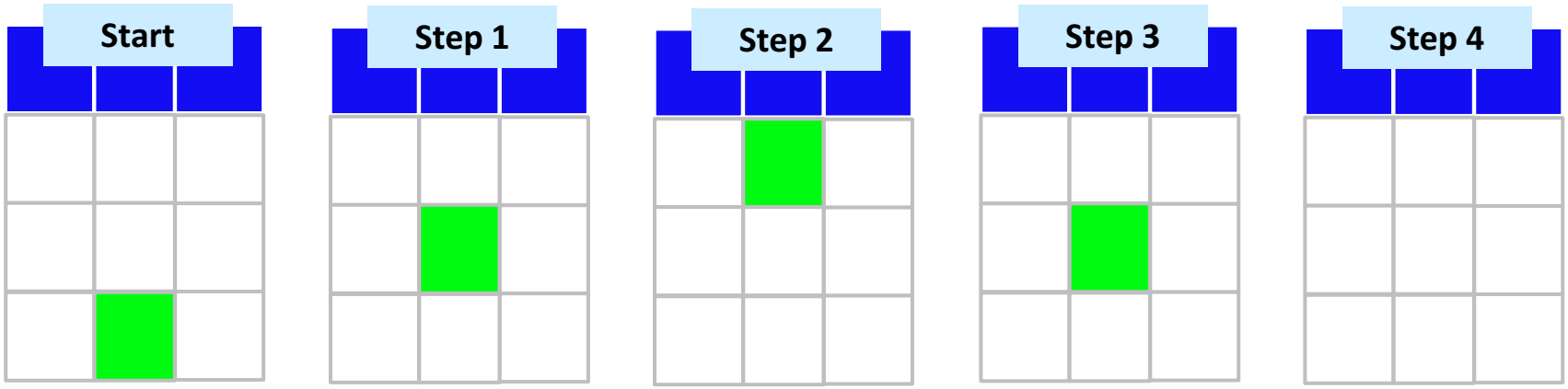
	current state	surroundings		direction	new state
rule (A)	0	Nxxx	→	S	0
rule (B)	0	xxxx	→	N	0

these two rules are a complete Picobot program

Notes

Picobot checks its rules from the top each time.

When it finds a matching rule, that rule runs.



Picobot programming ~ ***rules***

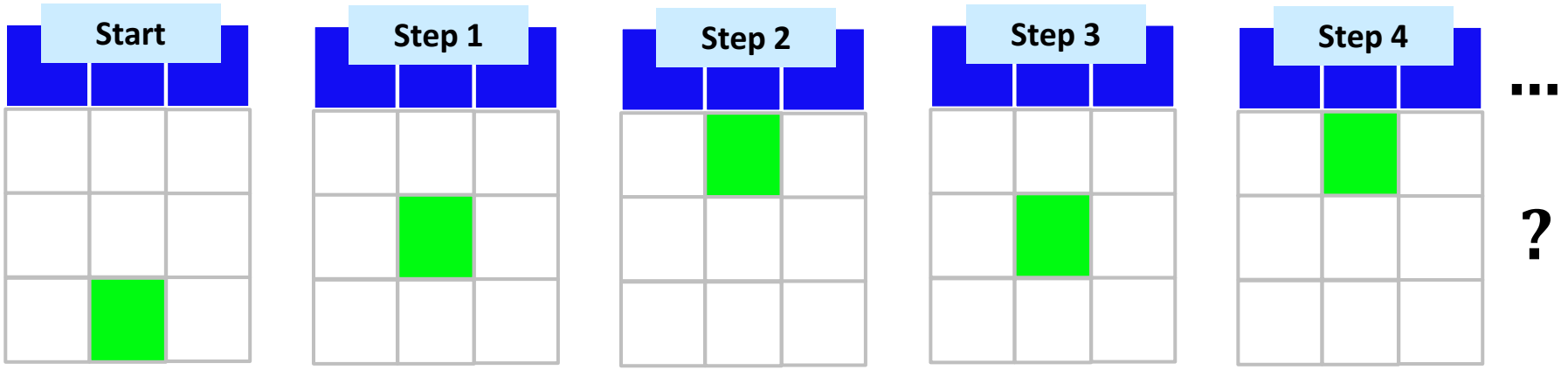
	current state	surroundings		direction	new state
rule (A)	0	Nxxx	→	S	0
rule (B)	0	xxxx	→	N	0

these two rules are a complete Picobot program

Notes

Picobot checks its rules from the top each time.

When it finds a matching rule, that rule runs.



Picobot programming ~ ***rules***

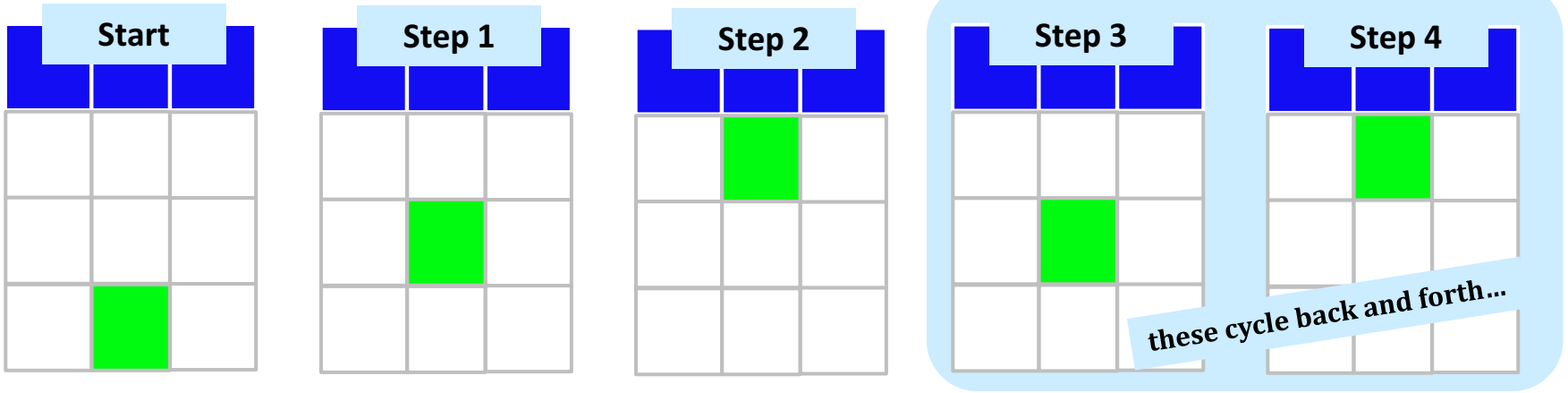
	current state	surroundings		direction	new state
rule (A)	0	Nxxx	→	S	0
rule (B)	0	xxxx	→	N	0

these two rules are a complete Picobot program

Notes

Picobot checks its rules from the top each time.

When it finds a matching rule, that rule runs.



Wildcards

I only care about **NORTH being EMPTY**

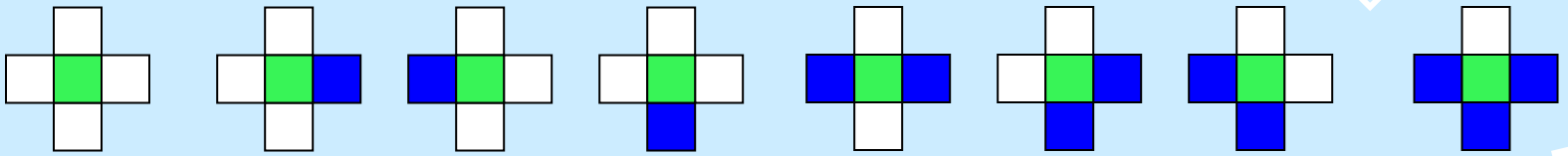
Asterisks * are wild cards.
They match walls ***or*** empty space:

current state	surroundings	direction	new state
0	x***	N	0

N must be empty

EWS may be wall ***or*** empty space

8 surroundings in one rule



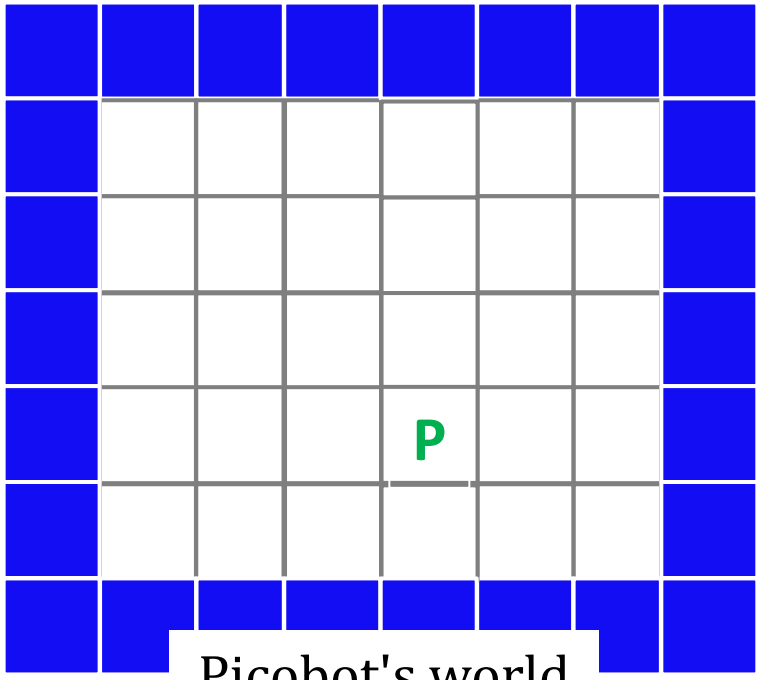
that's it!

The Rule is ***One step per rule***

One rule to rule them all?



That's *precious*!



Picobot's world

	state	surr.		move	new state
rule (A)	0	N***	->	W	1
rule (B)	0	x***	->	N	0
rule (C)	1	***x	->	S	1
more rules					

1. Run Picobot! Which rule **A, B,** or **C** runs ***first***? _____

   1a. How many times does **rule (A)** run? _____

   1b. How many times does **rule (B)** run? _____

   1c. How many times does **rule (C)** run? _____

2. Picobot stops when no rule matches. ***Where does it stop?***

3. Add a rule so that Picobot continues ***back upward!***

Extra: Rule A has a bug! What is it?

Warning! *What's wrong here?*

state	surroundings		direction	new state
0	x***	→	S	0
0	***x	→	N	0

these two rules are a broken Picobot program!

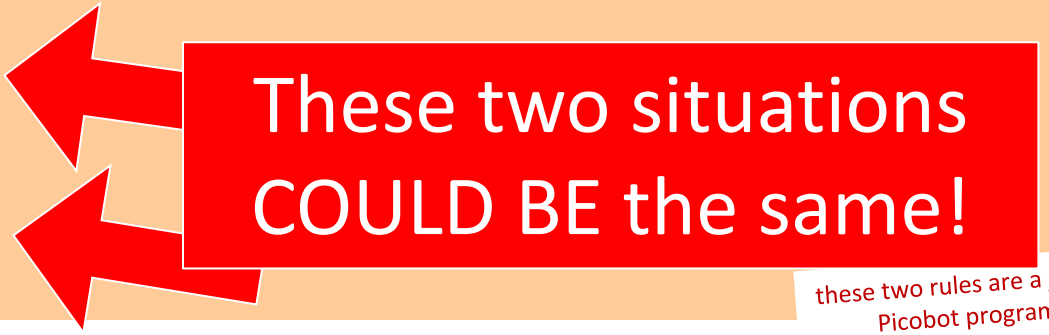
Notes

Picobot checks its rules from the top each time.

When it finds a matching rule, that rule runs.

Warning! *What's wrong here?*

state	surroundings	direction	new state
0	x***		
0	***x		



these two rules are a broken Picobot program!

Notes

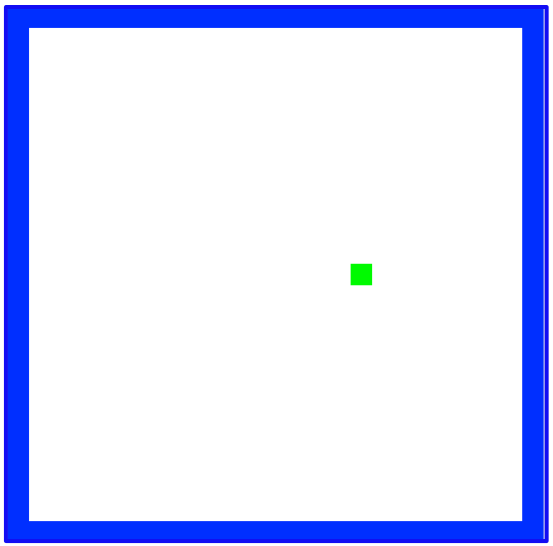
Picobot checks its rules from the top each time.

When it finds a matching rule, that rule runs.

There can only be **ONE** rule per situation!

and a "situation" is *state* and *surroundings*

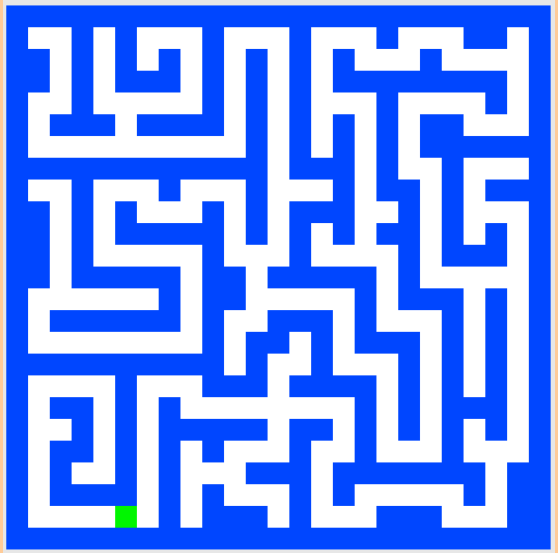
CS ~ ***C****omplexity* **S**cience



problem 3

Shortest Picobot program:

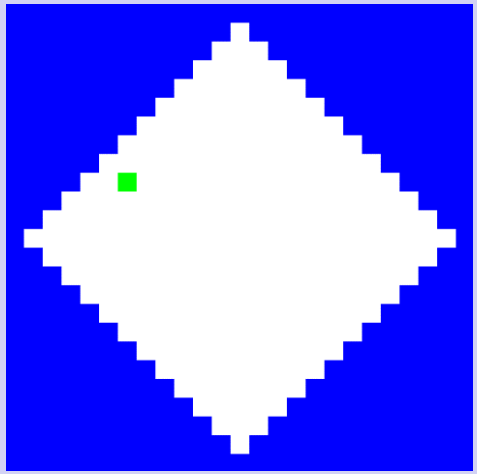
6 rules



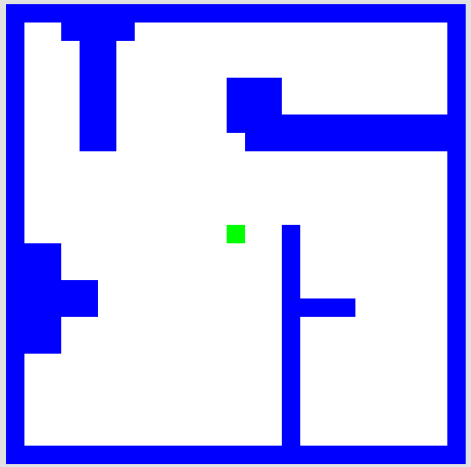
problem 4

Shortest Picobot program:

8 rules

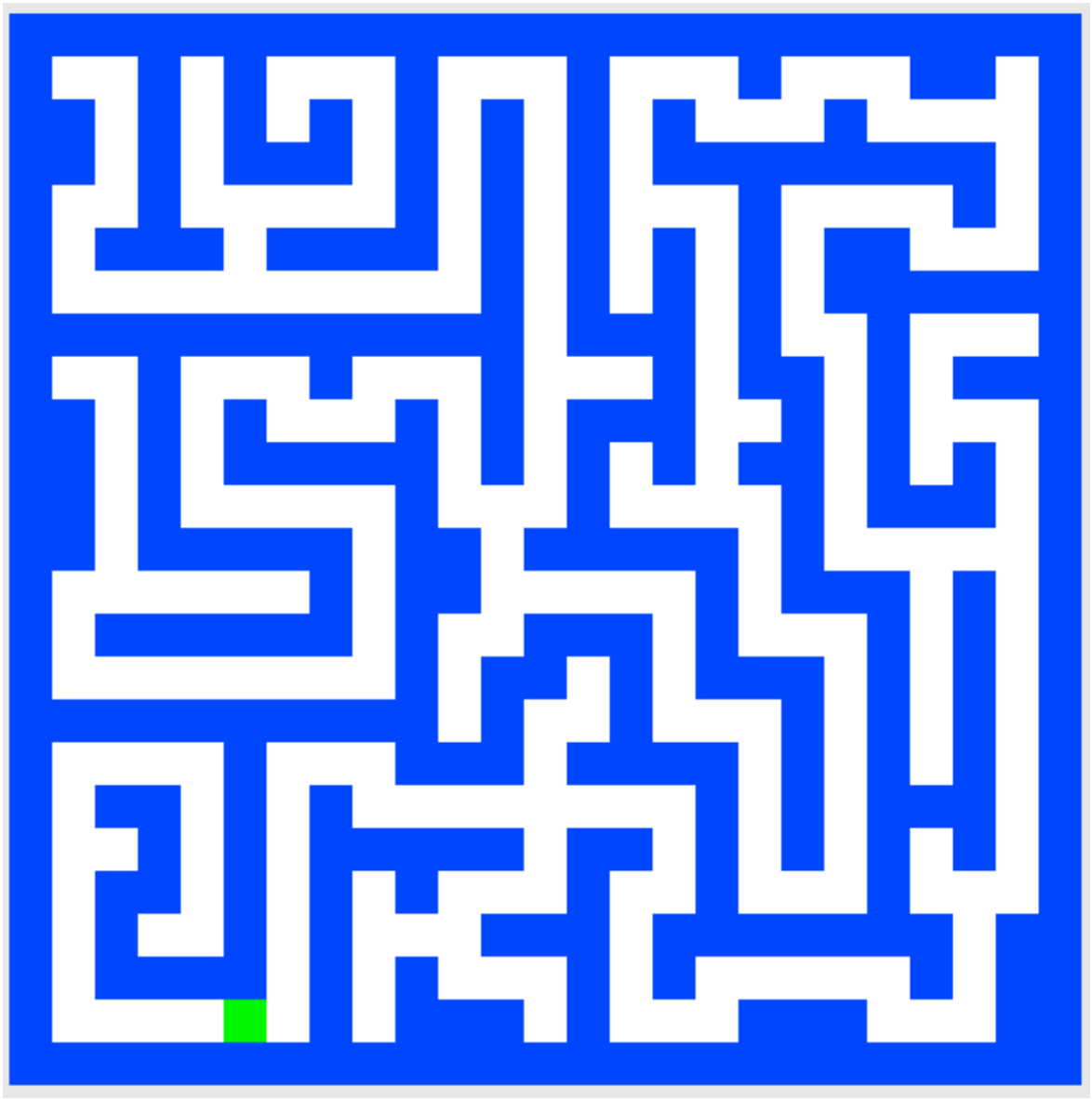


pr. 5 (extra!)



pr. 6 (extra!)

Maze strategies?



en.wikipedia.org/wiki/Maze_solving_algorithm

Create account Log in

Article Talk Read Edit View history Search

Maze solving algorithm

From Wikipedia, the free encyclopedia

There are a number of different **maze solving algorithms**, that is, automated methods for the solving of mazes. The random mouse, wall follower, Pledge, and Trémaux algorithms are designed to be used inside the maze by a traveler with no knowledge of the maze, whereas the dead-end filling and shortest path algorithms are designed to be used by a person or computer program that can see the whole maze at once.

Mazes containing no loops are known as "standard", or "perfect" mazes, and are equivalent to a *tree* in graph theory. Thus many maze solving algorithms are closely related to graph theory. Intuitively, if one pulled and stretched out the paths in the maze in the proper way, the result could be made to resemble a tree.[1]

Contents [hide]

1 Random mouse algorithm

Wikipedia
The Free Encyclopedia

- Main page
- Contents
- Featured content
- Current events
- Random article
- Donate to Wikipedia
- Wikimedia Shop

Interaction

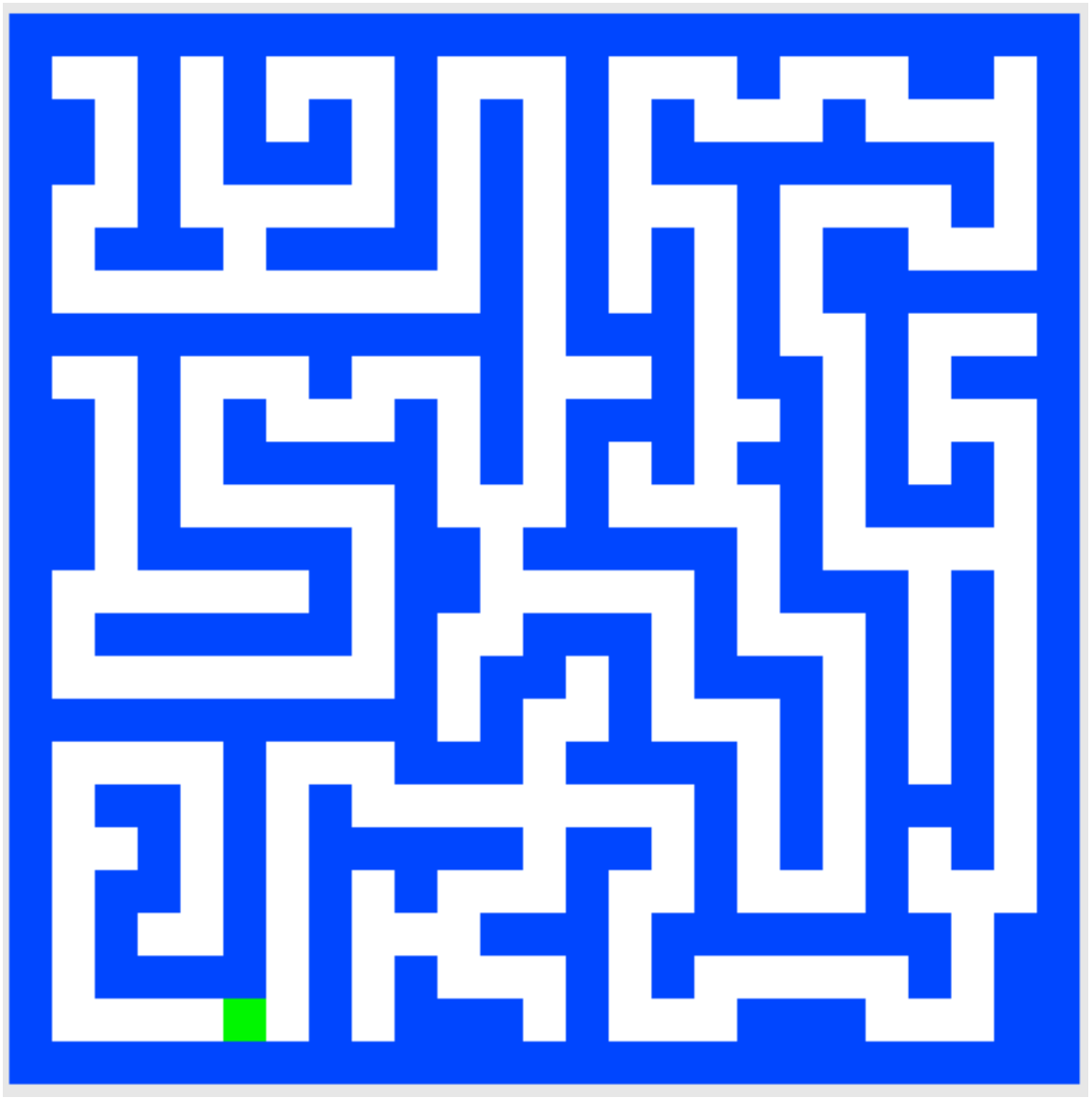
- Help
- About Wikipedia
- Community portal
- Recent changes
- Contact page

commons.wikimedia.org/wiki/File:MAZE_2000x2000_DFS.png

Right Hand Rule

Maze strategies?

Right Hand Rule



Maze strategies?

Right Hand Rule



Why might this be *difficult* for Picobot?

Maze strategies?

Right Hand Rule



Keep your "right hand" on the wall, Picobot!

State **0**
State **1**
State **2**
State **3**

facing

on the right

We'll need to use state to represent the ***direction Picobot is facing***.

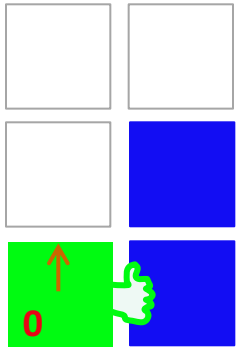
Suppose Picobot wants to traverse a maze ***with its right hand <u>always</u> on the wall...***

(A) CORRIDOR rule

<u>If you're facing N with a wall at right and space ahead</u> then <u>go forward</u>"

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

state 0 means "still facing north"

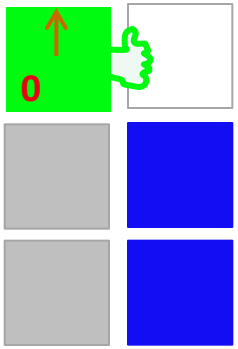


(A)

(B) INTERSECTION rule

"<u>If you're facing North and lose the wall</u>, then <u>get over to the wall now!</u>"

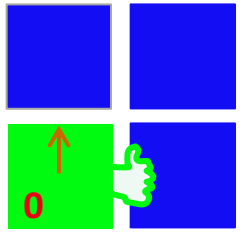
```
0    ->    E    1
```



(B)

(C) DEAD END rule

Write a rule to tell Picobot to do the right thing if it hits a dead end.



(C)

Repeat this IDEA for all four states, representing all four ***facing directions.***

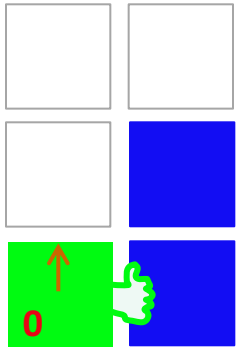
Suppose Picobot wants to traverse a maze ***with its right hand always on the wall...***

(A) CORRIDOR rule

If you're facing N with a wall at right and space ahead then go forward"

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

state 0 means "still facing north"



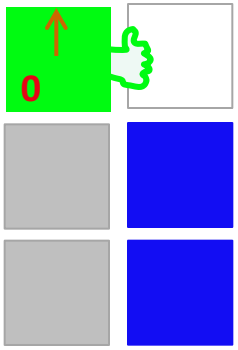
(A)

(B) INTERSECTION rule

"If you're facing North and lose the wall, then get over to the wall now!"

```
0    *x**    ->    E    1
```

state 1 means "now facing east"



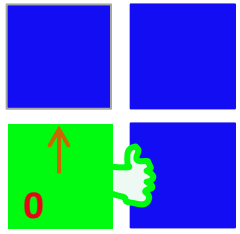
(B)

(C) DEAD END rule

Write a rule to tell Picobot to do the right thing if it hits a dead end.

```
0    NE**    ->    X    2
```

state 2 means "now facing west"



(C)

Repeat this IDEA for all four states, representing all four ***facing directions.***