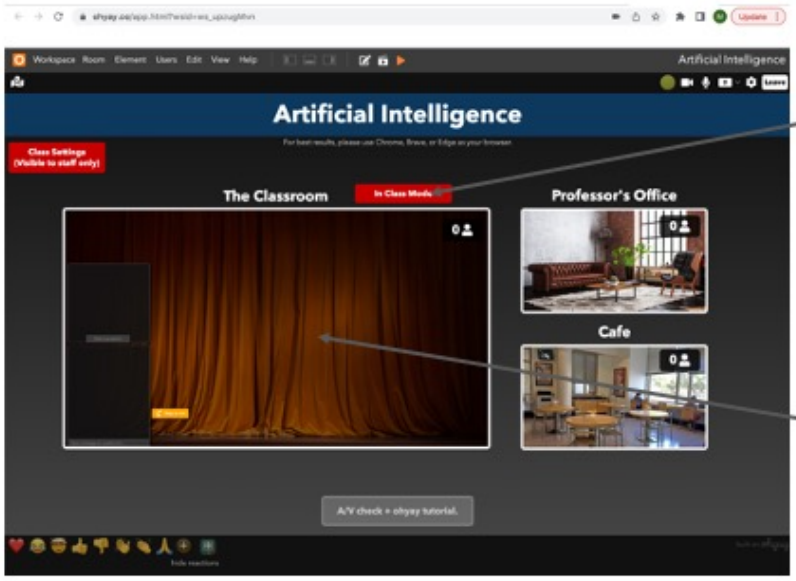
Getting started

- Course web page (syllabus, assignments, etc.): go/cs150
- Enrollment: go/cs150-attend
- OhYay video platform: go/cs150-ohyay

OhYay “hybrid” video/in-class platform (go/cs150-ohyay)

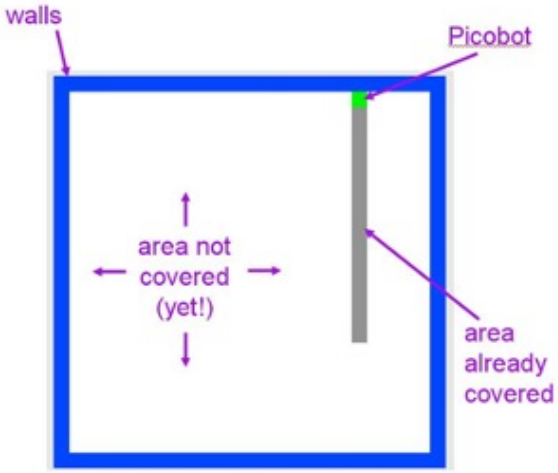


“In-class” interface with polling, chat, questions and reactions but without video

Full video interface if you are unable to attend class due to COVID-19

- In video mode you are muted by default. Use “step to mic” to ask a question.
- Chat, questions, reaction emojis for live feedback in person or remote…
- Polls (peer instruction questions)

Introduction to Picobot

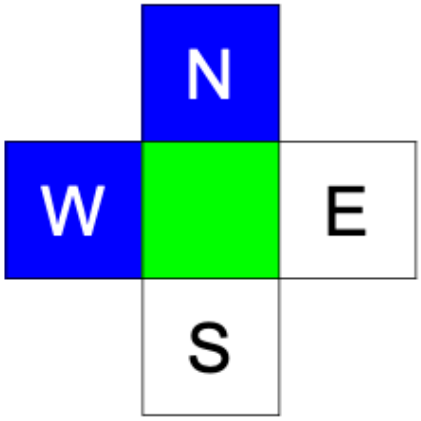


Goal: whole-environment coverage with only *local sensing*…

https://www.cs.hmc.edu/twiki/bin/view/CSforAll/IntrotoPicobot

Picobot starts at a random location in the room and stops automatically if it has explored the entire room (the goal).

Picobot surroundings



NxWx

North East West South

Surroundings are always in NEWS order

https://www.cs.hmc.edu/twiki/bin/view/CSforAll/IntrotoPicobot

This example corresponds to a wall to the “north” and “west” sides and nothing (i.e. no wall) to the “south” and “east”.

Picobot rules

Picobot's "memory", the context in which the rules is applied

```
StateNow   Surroundings   ->   MoveDirection   NewState
```

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

"If Picobot is in state 0 and senses xxxS (only a wall to the south) it should move north and stay in state 0"

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

"If Picobot is in state 0 and senses nothing to the north and anything (* wildcard) to the east, west, and south (wall or not) it should move north and stay in state 0"

https://www.cs.hmc.edu/twiki/bin/view/CSforAll/IntrotoPicobot

Note that in each step the applicable rules for the current state are checked in order from top to bottom and selects the first rule that applies and then repeats that process in the next step.

One of the trickier aspects of Picobot can be the "state". State is Picobot's "memory", how it knows, for example, that it is supposed to be going "north". Recall that Picobot can only sense its local surroundings and doesn't have any overarching sense of direction (other than those local observations). The state can provide that context. More generally the state of the computer is its current condition or alternately its current internal configuration. For Picobot the state is a number from 0 - 99. That number doesn't have any intrinsic or built-in meaning. The meaning comes from you. For example, you could effectively define state 0 as the "going north till it hits a wall" state and 1 as the "going south until it can't anymore" state (or whatever numbers you choose). That is I recommended associating a goal with each state, e.g., "go all the way to the east". And when you want to switch to a new goal, you probably want to switch states.

Will these rules fully traverse an empty room?

```
# Hashtag lines are optional comments

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

# state 0 with something to the N: go W + into st 1
0 N*** -> W 1

# state 1 with nothing to the S: go one step S
1 ***x -> S 1

# state 1 with something to the S: stay put + into state 0
1 ***S -> X 0
```

A. Yes
B. No

Answer: B.
Picobot gets stuck in the upper left (northwest) corner. Picobot always starts in state 0. In most cases it will be in the middle and so traverse north until it hits a wall (triggering the second rule). At that point Picobot goes west one space and switches to state 1. With no wall to the south, Picbot will traverse south in state 1 until it hits a wall. It will then switch back to state 0 (with no movement) and start heading north. As we noted earlier, we typically associate (define) state with a goal. In this case, state 0 could be defined as go north until a wall and then shift west one and state 1 could be defined as go south as far as it can.

Is there any starting position where these rules would successfully traverse the entire room? Yes, in the bottom right corner.

We could just test this, why do I ask you to not use the simulator? We are trying to develop a mental model of what the computer is doing. When we write our programs we start with a mental model of what the computer should do and express that in the code. An important skill is maintaining that mental model.