


# Algorithms 3: Implementation



# Team Assessment!

Team member...

- ...is always too busy to join meetings
- ...doesn't show up to class and expect us to catch them up
- ...doesn't communicate, and things get done twice/wrong
- Why?
  - Lazy \$#%\*
  - 
  - Busy...
    - Prioritize what is more critical

8am	8 – 9 Breakfast w. Colloquium speaker
9am	9 – 10 Grade Lab 3 HW
10am	10 – 11 Prep ECE3400 class
11am	11 – 12p ECE3400 lecture
12pm	12p – 1:30p Faculty meeting
1pm	
2pm	1:30p – 2:30p ECE3400 Lab
3pm	2:30p – 3:30p Meeting w. Owen/Yawen
	3:30p - Meeting w Vaidehi
4pm	4p - Meeting w. Tim/Asena
5pm	4:30p – 6p ECE Colloquium
6pm	6p - Meeting w. Steven
7pm	6:30p – 7:30p Johnson&Johnson deadline!!
8pm	7:30p – 9p ECE3400 Lab

# Team Assessment!

Team member...

- ...is always too busy to join meetings
- ...doesn't show up to class and expect us to fill catch t
- ...doesn't communicate, and things get done twice/w
- *Follow through with consequences!*
  - Talk, pizza jar, public shaming, talk to TAs and Prof
  - Team Assessment
  - What most of you did:
    - 1: 22, 2: 20, 3: 18, 4: 19.5, 5: 20.5
  - Zero sum game! (Team of 6 have 45 points)
  - What you should do:
    - 1: 25, 2: 25, 3: 5, 4: 20, 5: 25

*Let's do the math!*

Each team assessment is 7.5pts

A team of 6 people have 45pts

$0.5\% * 45 = 0.225\text{pts}$

200 points total in class

Consequence is: 0.1125%



# Algorithms 3: Implementation



**Implementation**

*Realistic*

**Simulation**

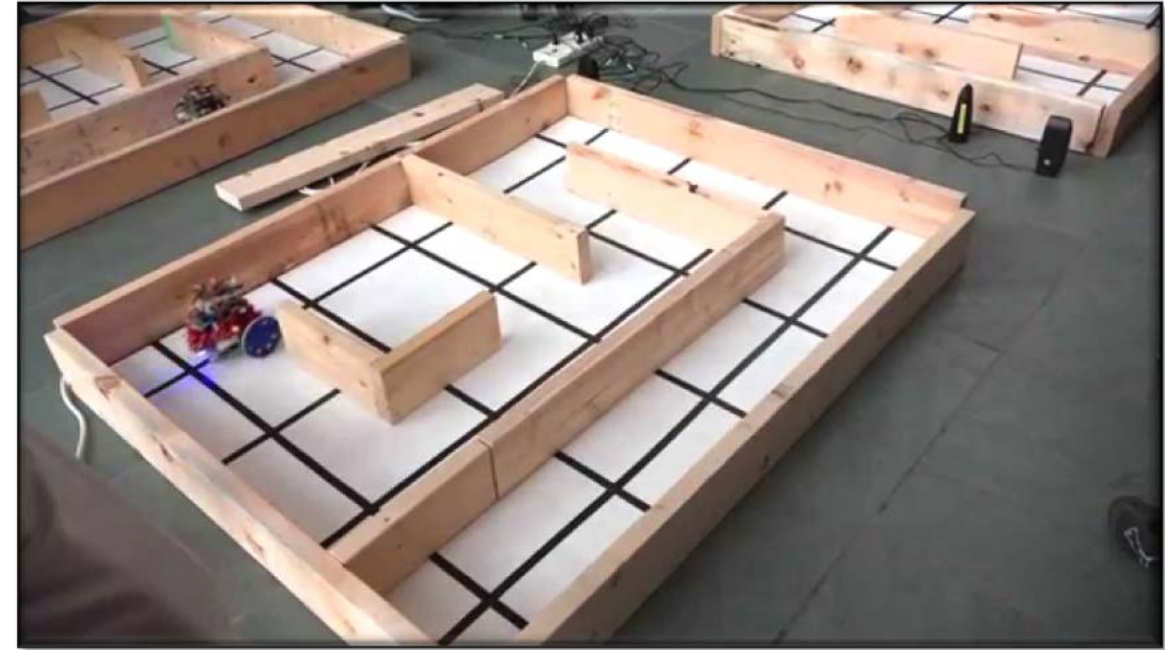
*Fast  
iterations*

**Theory**

*Formal  
guarantees*

# How Accurate Does the Simulation Have to Be?

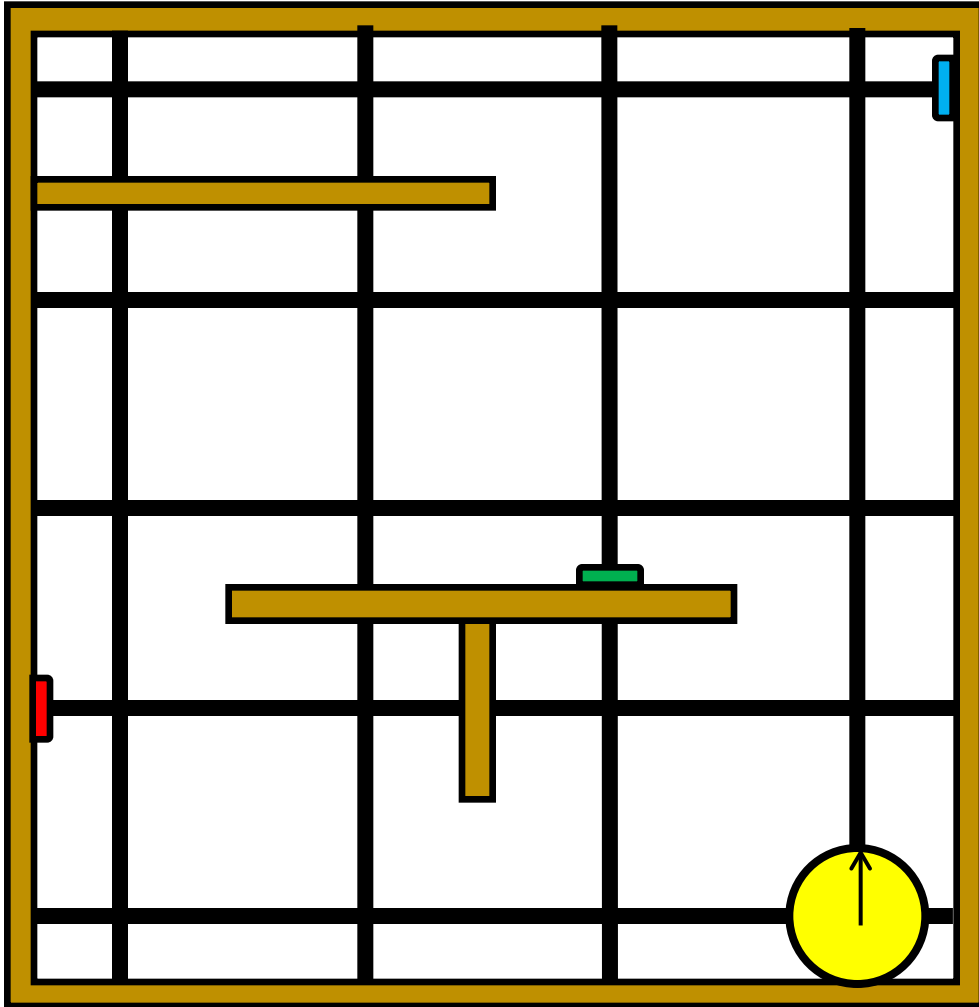
- World representation
  - Black tape grid
  - Walls
  - Treasure
  - Start signal
  - FPGA interface
- Robot representation
  - Motion model (heading, differential steering, speed, etc.)
  - Sensors
  - Battery
  - Memory
  - Reliability



*Can you make the  
code portable?*

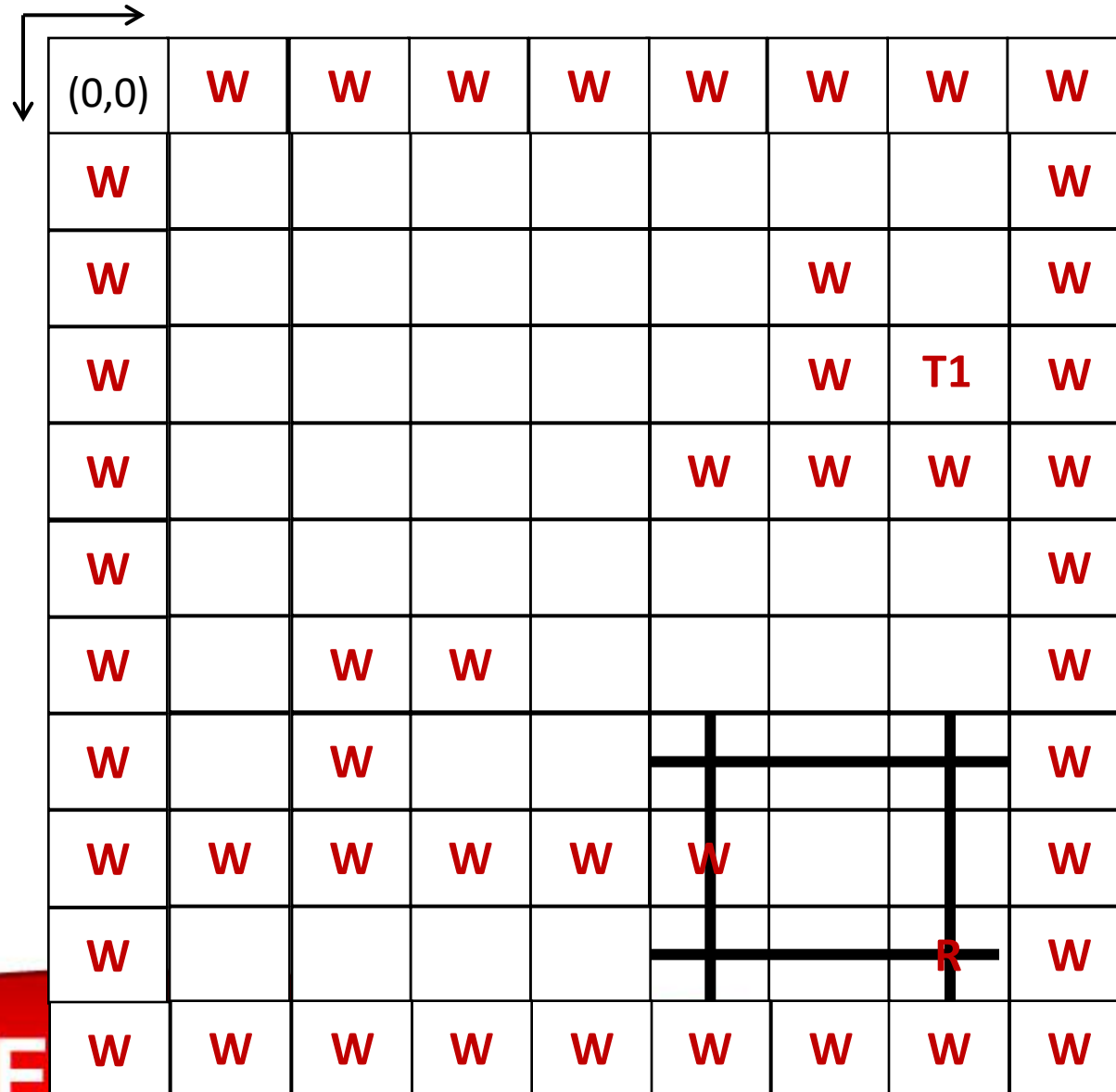


# World Representation



- The junctions are where your robot/treasures can be!
- The walls are in between the junctions

# World Representation



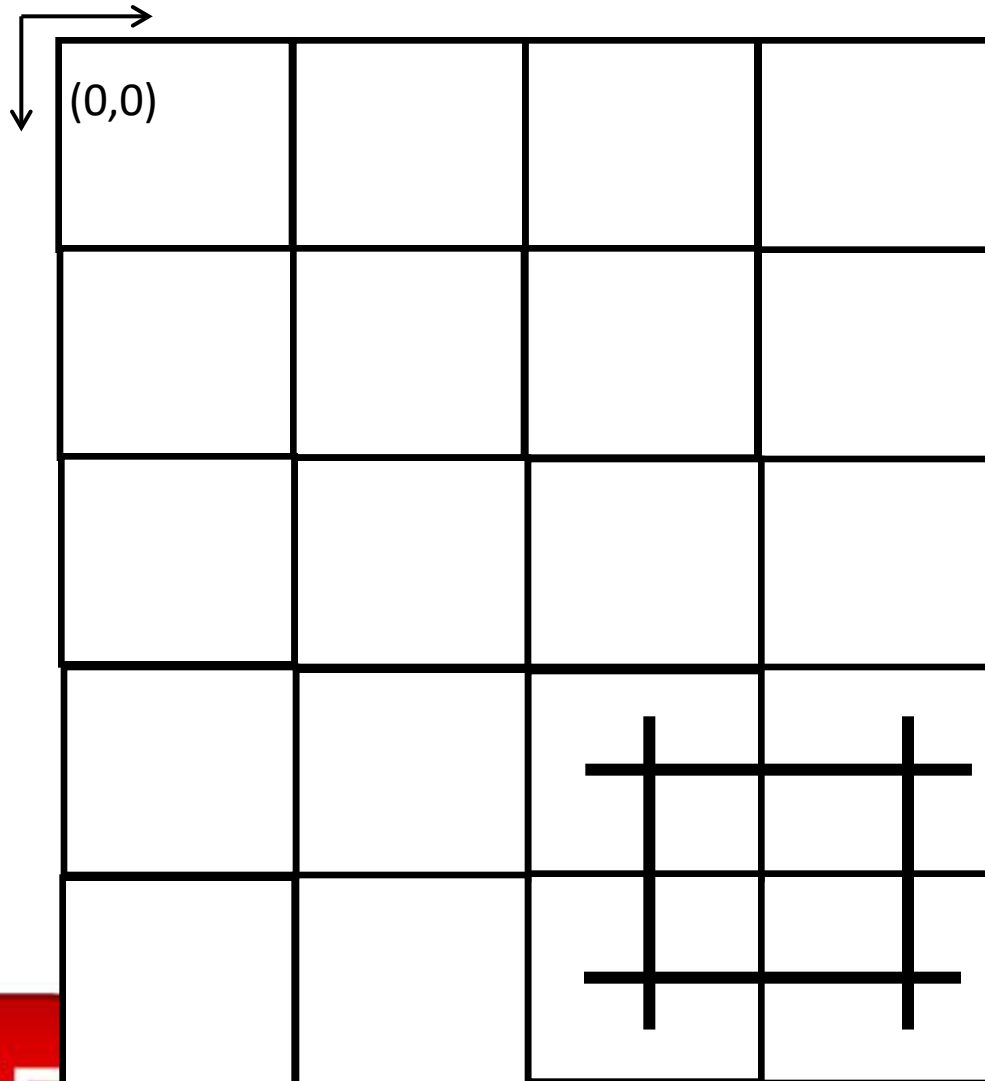
(0,0)	W	W	W	W	W	W	W	W	
W									W
W						W			W
W						W	T1		W
W					W	W	W		W
W									W
W		W	W						W
W		W							W
W	W	W	W	W	W				W
W							R		W
W	W	W	W	W	W	W	W	W	W

## Option 1:

- Twice as many cells as junctions
- Walls are just marked grid spaces
- Robot moves two steps every time
- Compute possible moves:
  - $[r.x \ r.y] = [7 \ 9]$
  - Check wall  $[r.x-2 \ r.y]$
  - Check wall  $[r.x+2 \ r.y]$
  - Check wall  $[r.x \ r.y-2]$
  - Check wall  $[r.x \ r.y+2]$



# World Representation



## Option 2:

- One cell per junction
- Walls:
  - `cell.wall = [N E S W]`
- Compute possible moves:
  - `cell.wall` negated
- Mark both current and adjacent cell!

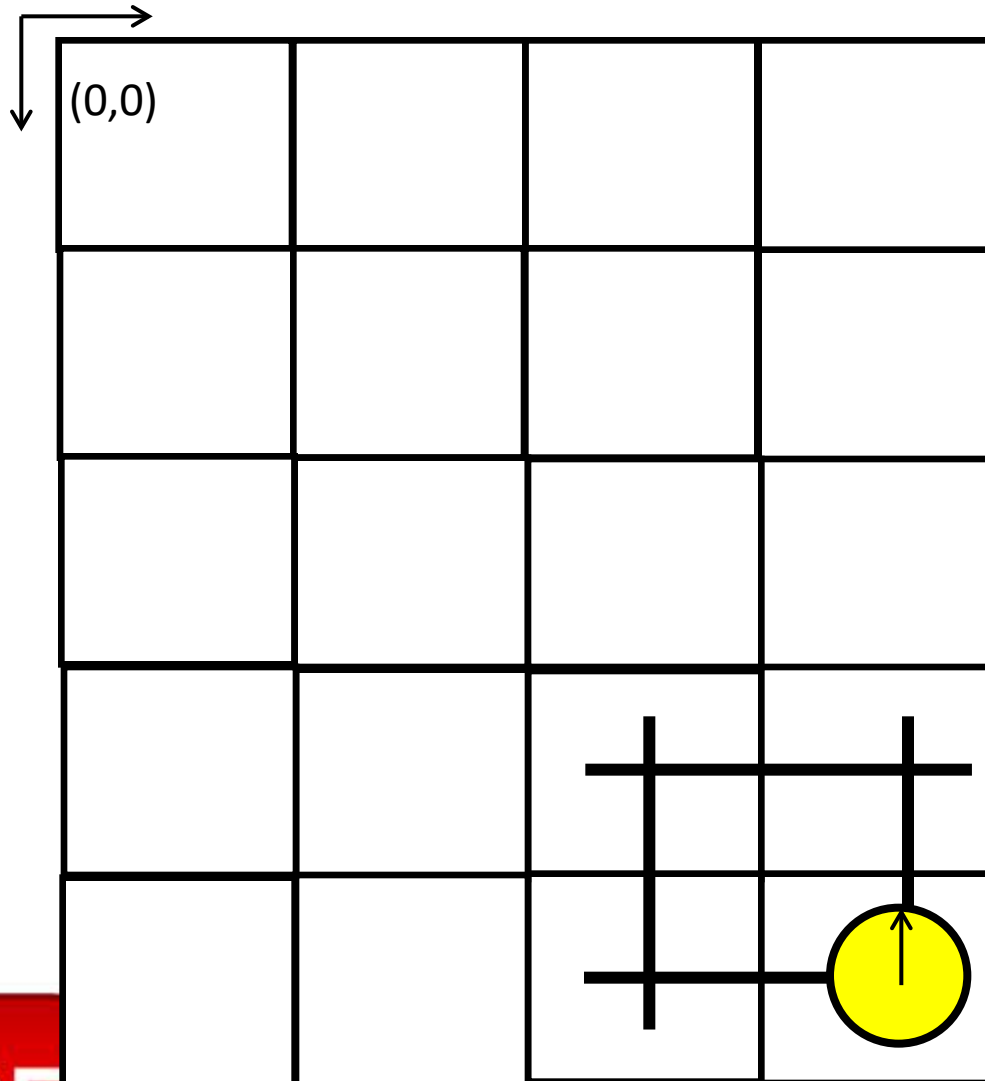
# World Representation

16	17	18	19
12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

Option 3:

- Linear array
- Slightly faster computation

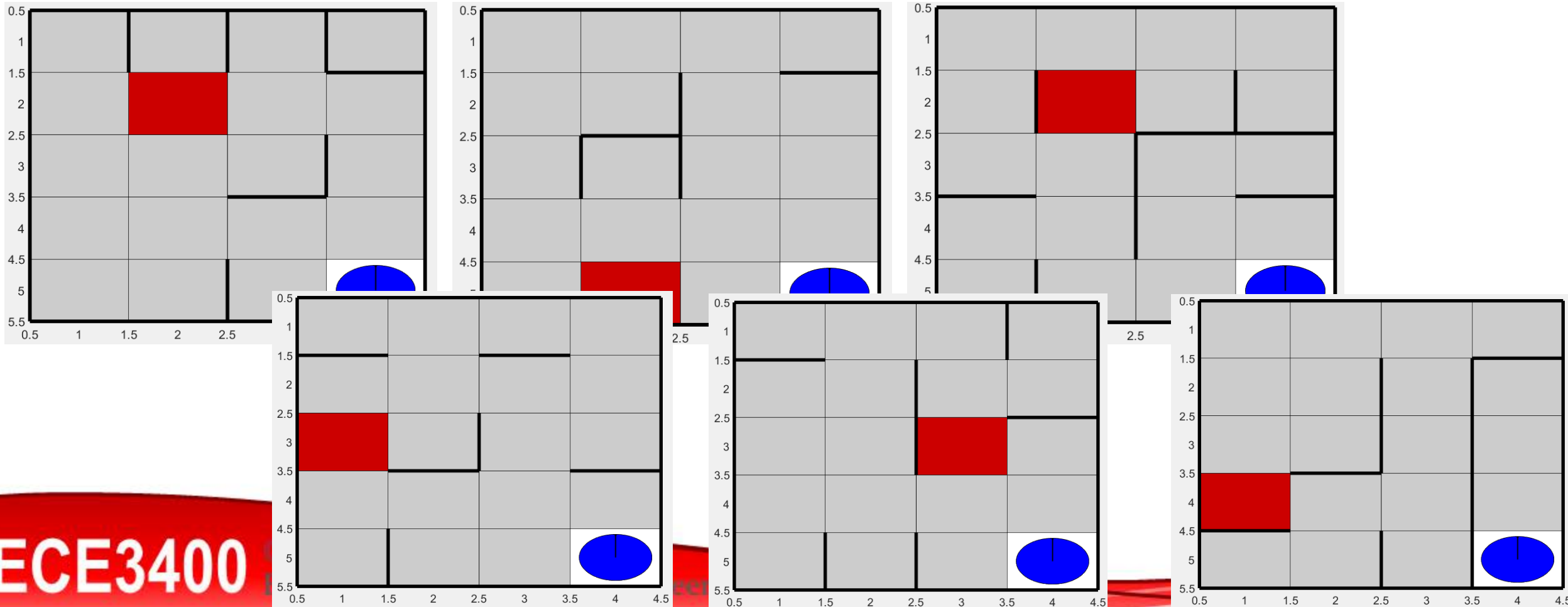
# Robot Representation



- $\text{robot.pos} = [x \ y]$
- $\text{robot.heading} = \text{N/E/S/W}$
- Turning:
  - North =  $[0 \ -1]$
  - East =  $[1 \ 0]$
  - South =  $[0 \ 1]$
  - West =  $[-1 \ 0]$
  - Turn Sequence =  $[0 \ -1; 1 \ 0; 0 \ 1; -1 \ 0]$
- $\text{robot.visited}$
- $\text{robot.frontier} = [\text{r.pos} \ \text{r.heading}]$

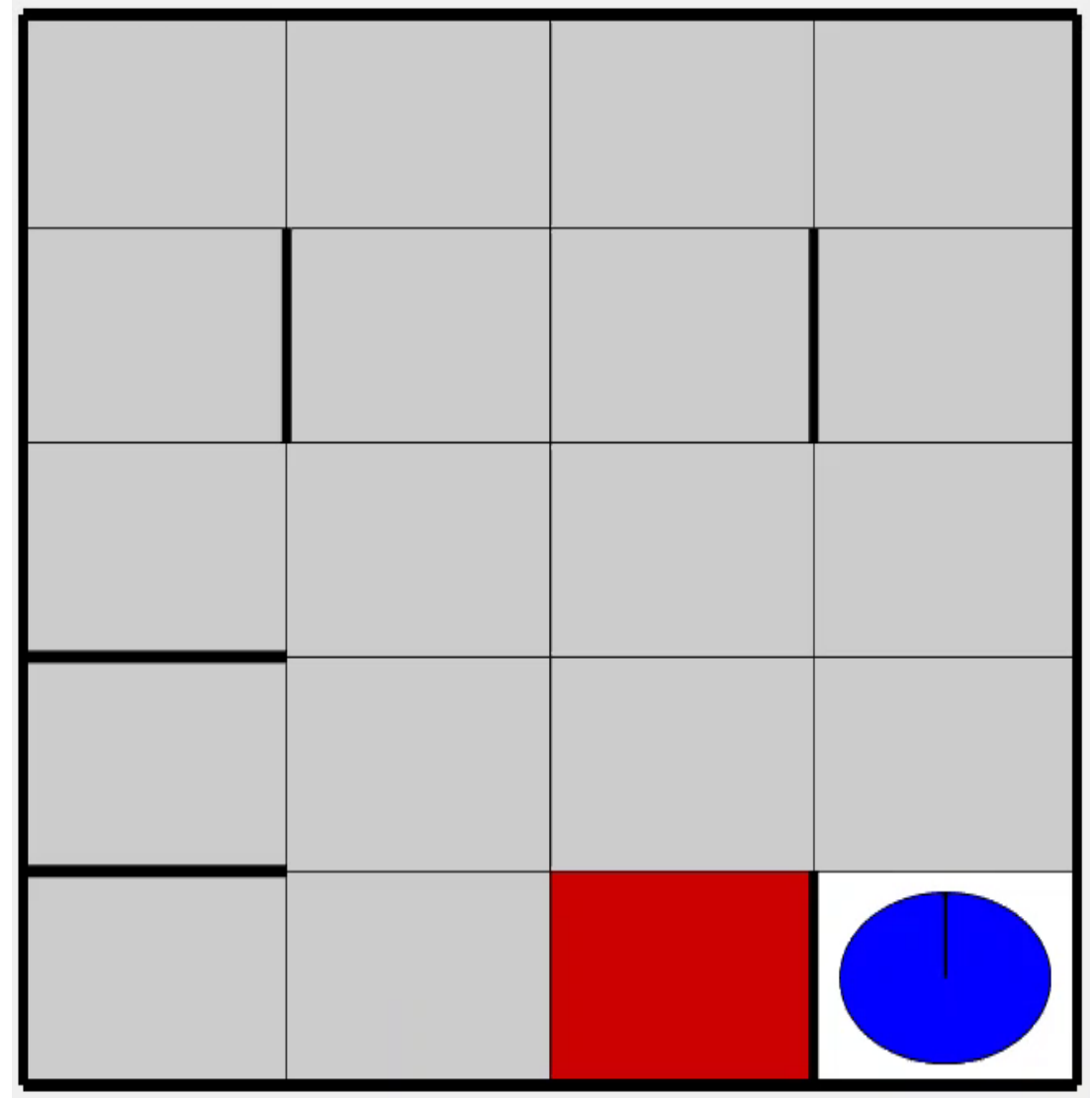
# How to make the most use of your simulation

- Generate random mazes!
- Run repeatedly (potentially w/o displays)



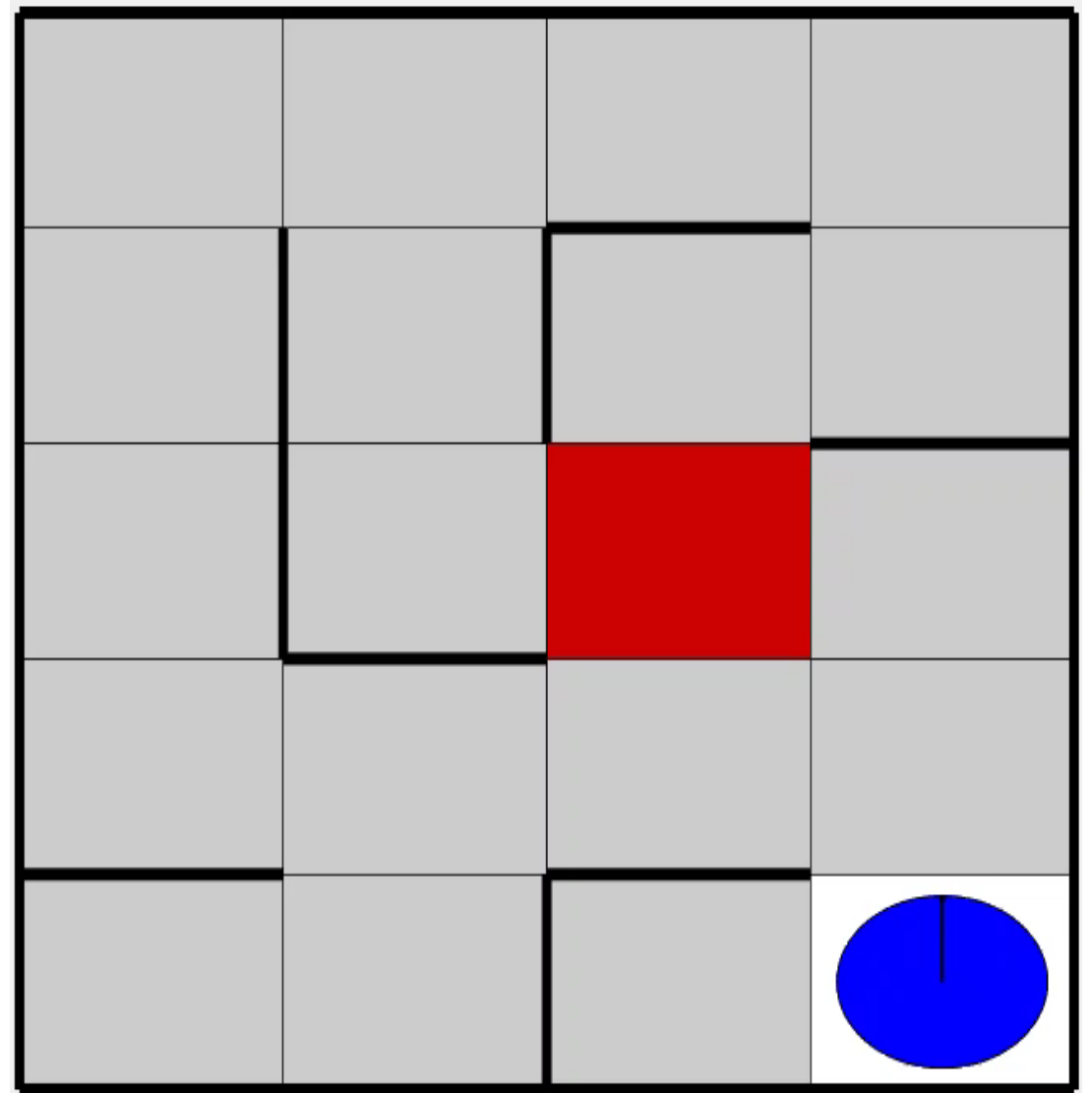
# How to Explore?

- Start simple!
- Logistics (turn sequence)
- Determined walk
  - In prioritized order:
    - Go straight
    - Turn CCW
    - Turn CW
- No. of moves:  $\infty$



# How to Explore?

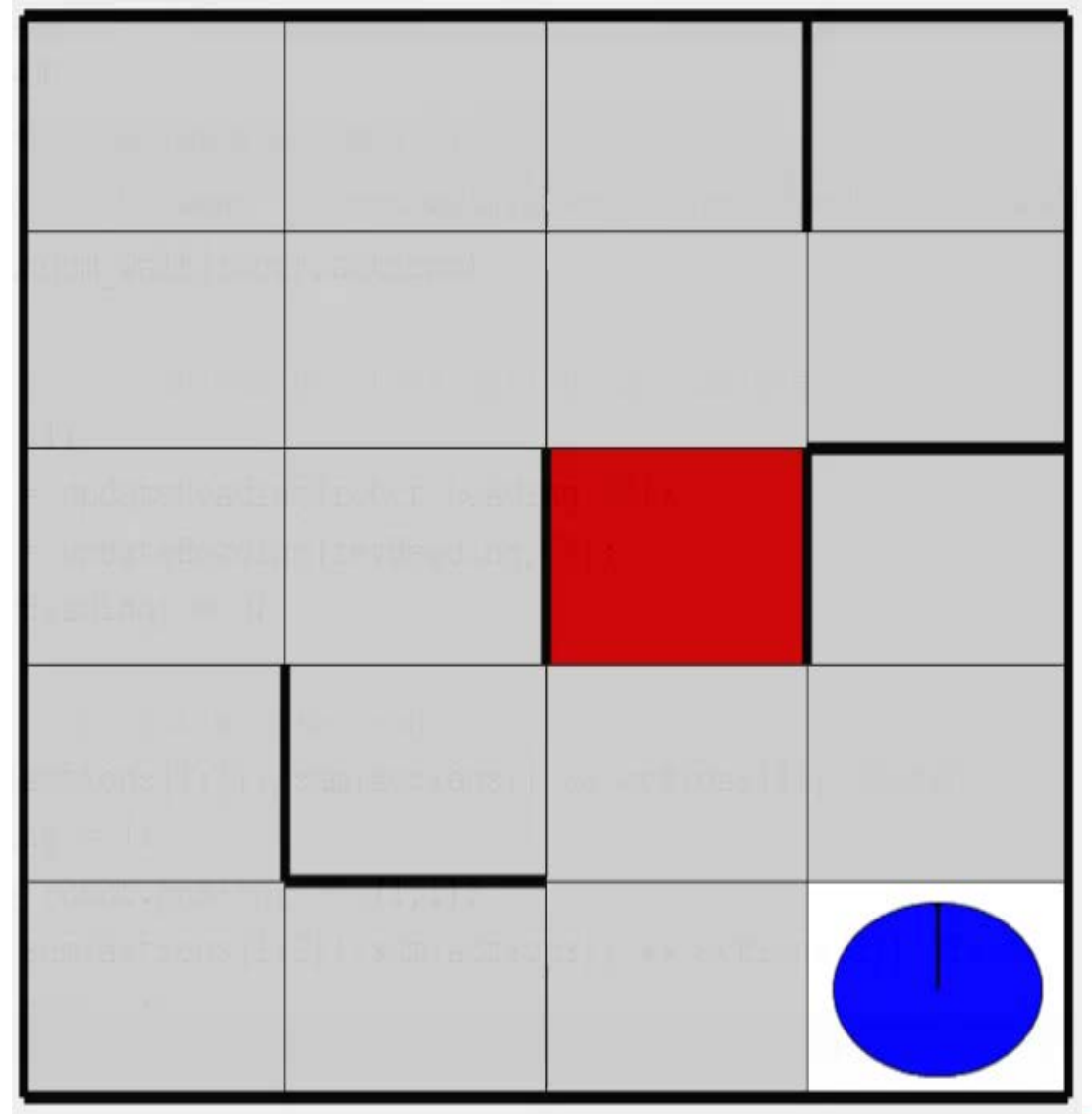
- Random Walk
- End condition?
- No. of moves: 221





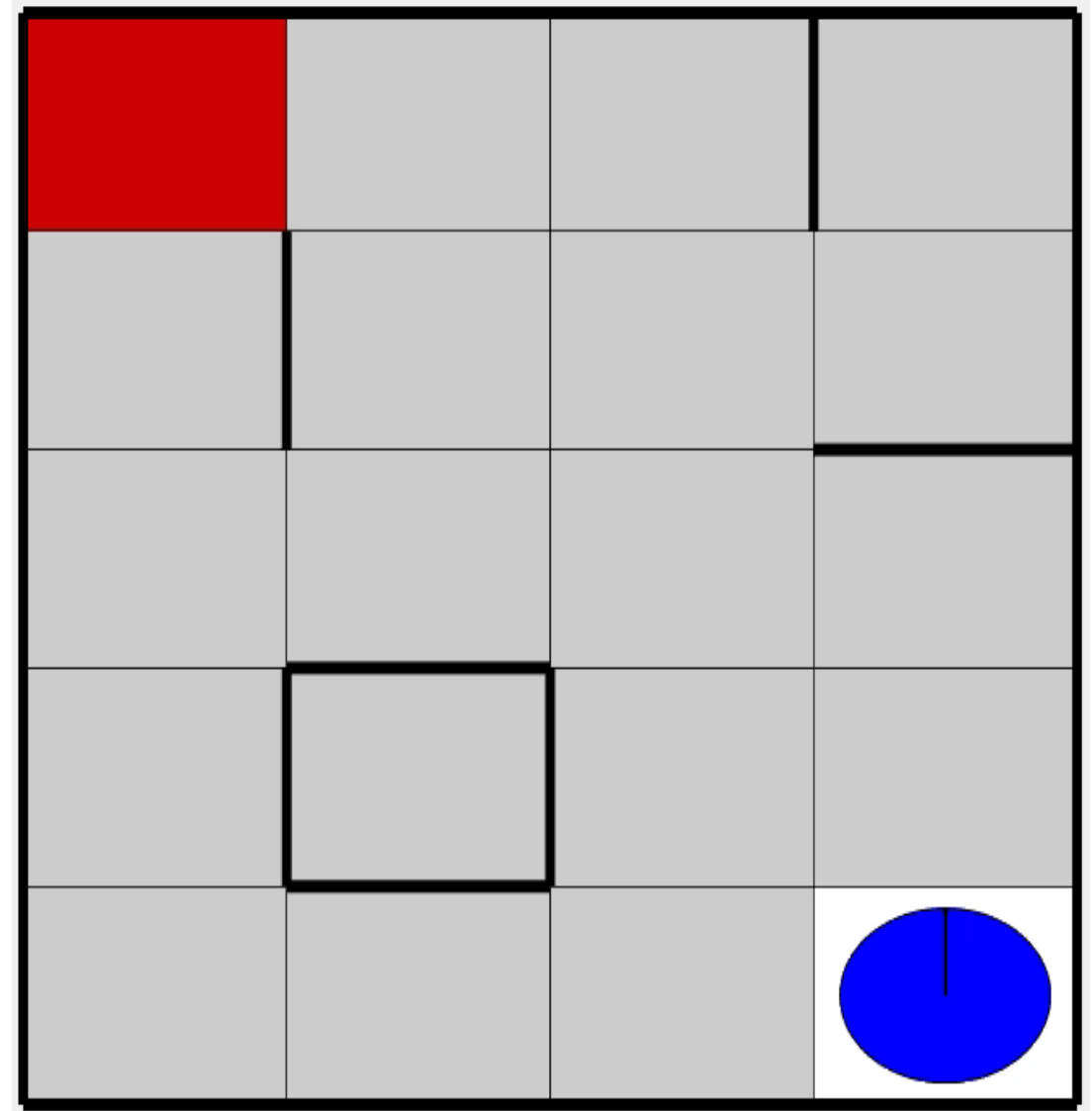
# How to Explore?

- Random Walk
  - (Don't go back)
- No. of moves: 126



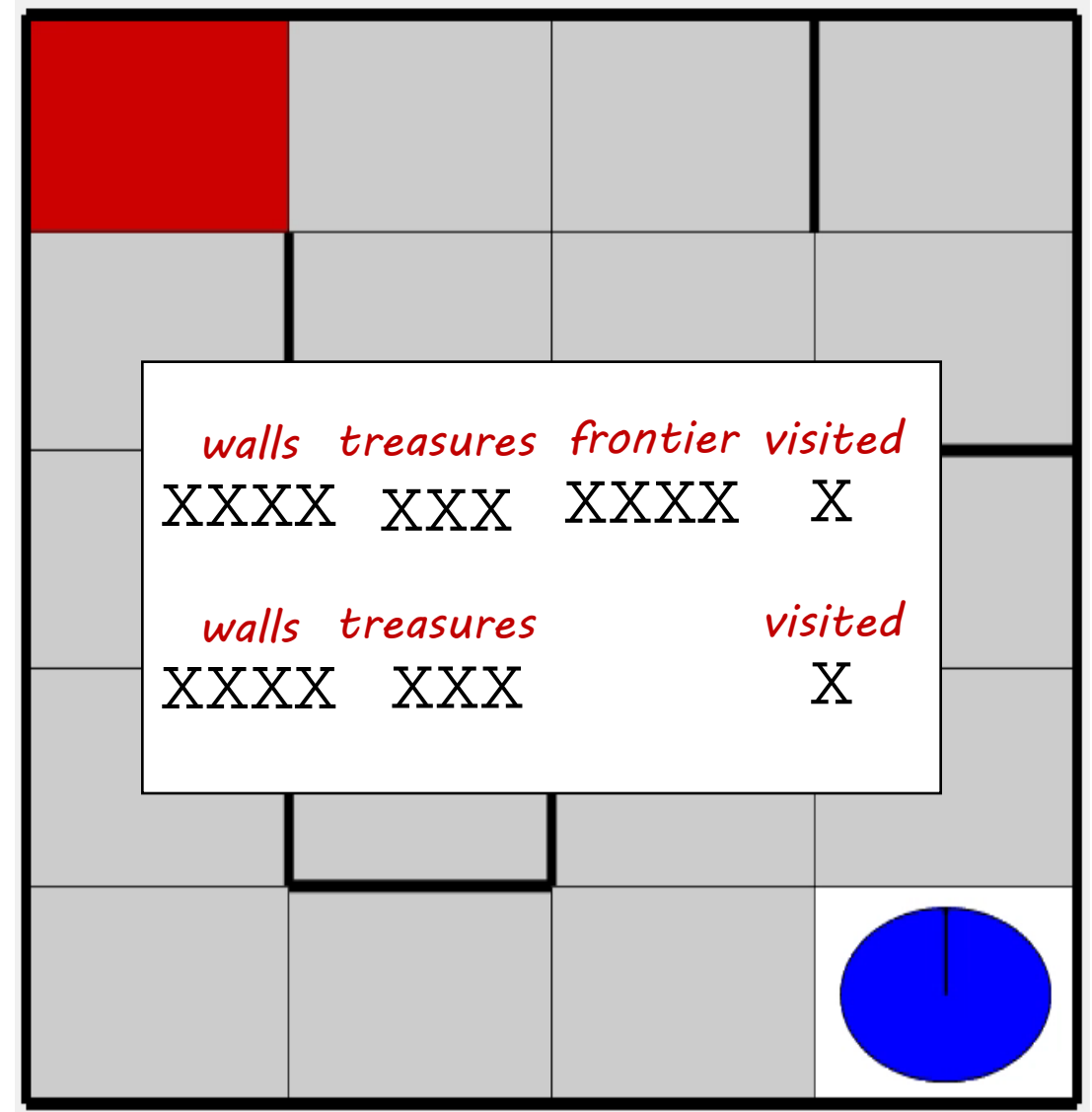
# How to Explore?

- Depth First Search
  - (No cost)
- End condition?
- No. of “moves”: 19
- Next step would be to implement BFS and do path planning....



# Embedded Programming

- Efficient storage vs. computation
- Bit masking!
  - Walls:
  - 1bit per direction (4 bits)
  - All options into 4 bits
    - None, N, E, S, W, NE, NS, NW, ES, EW, SW
  - 3 Treasures
    - 2 or 3 bits
  - Visited: 1bit
  - Frontier: ~4 bits

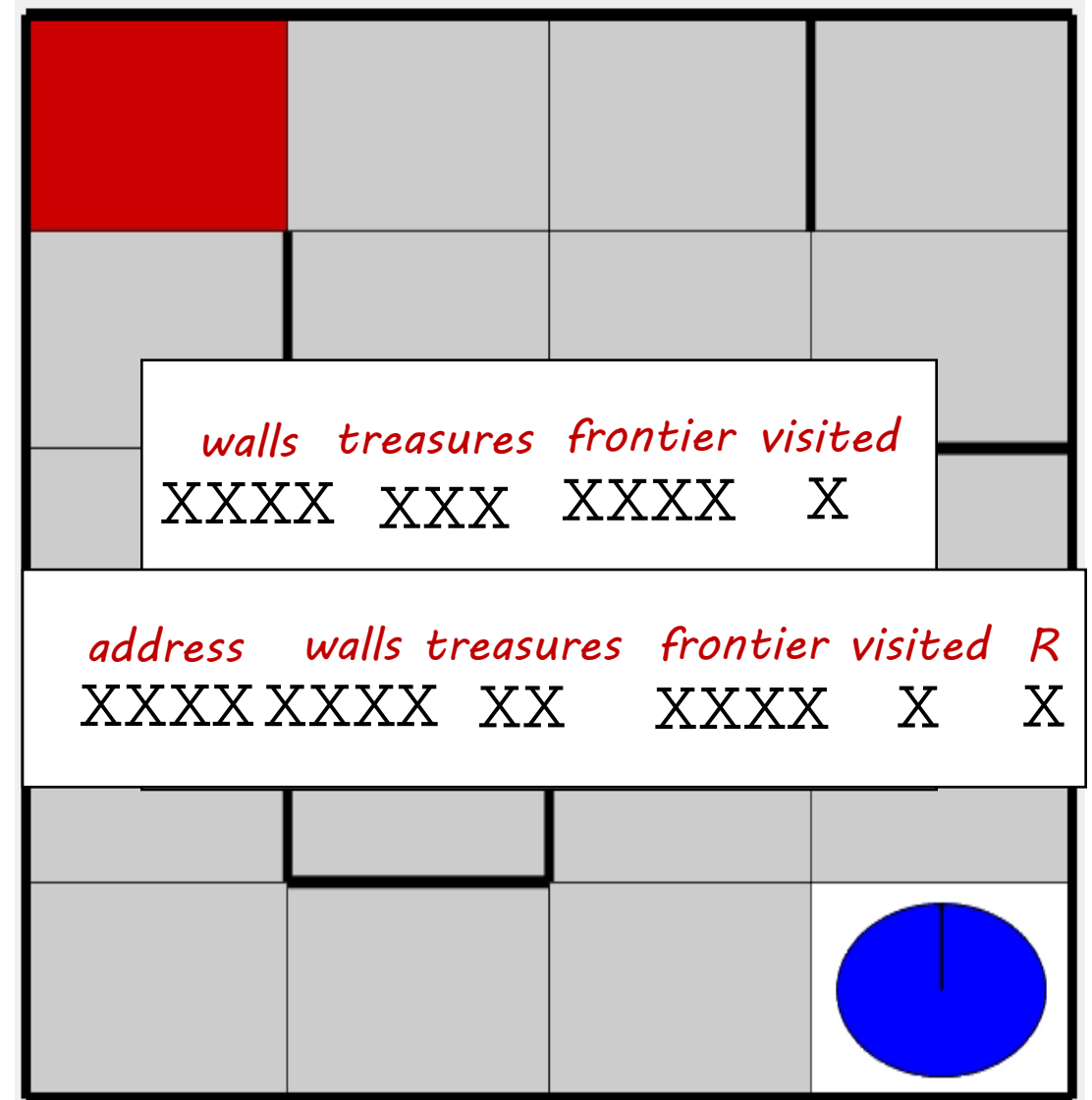


# Embedded Programming

- Efficient storage vs. computation
- Bit masking!
  - You could even include the address...
- Entire maze in 40bytes

*Separate array for  
path planning...*

16	17	18	19
12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3



# Embedded Programming

- Handy tricks
  - Bitwise NOT operator:
    - `Cell = ~0b11000000 //negate`
    - `Cell = 0b00111111`
  - Bitwise AND operator:
    - `Cell = 0b11000000 & 0b01111110 = 0b01000000`
    - `Cell &= 0b00000001; //clear everything except whatever is already in bit 0!`
  - Bitwise OR operator:
    - `Cell = 0b11000000 | 0b01111110 = 0b11111110`
    - `Cell |= 0b00000001; //make sure bit 0 is on!`
  - Bitwise XOR operator
    - `Cell = 0b11000000 ^ 0b01111110 = 0b10111110`
    - `LED ^= LED;`

# Embedded Programming

- Handy tricks
  - Bit-shift
    - `127 >> 1 = 0b01111111 >> 1 = 0b00111111 = 63`
    - `127 << 1 = 0b01111111 << 1 = 0b11111110 = 254`
    - `TCCR0 |= (1 << CS00);`
  - NB: behavior depends on the datatype!
    - `unsigned char A = 0b11111000;`
    - `A>>2 = 0b00111110;`
    - `signed char A = 0b11111000;`
    - `A>>2 = 0b11111110; //sign extension`
    - `signed char A = 0b11111000;`
    - `(unsigned char)A>>2 = 0b00111110;`

Computation time (ATmega328, 16MHz):

- Subtraction/Addition: 3896 us
- Multiplication: 3896 us
- Division: 153236 us



# Embedded Programming

- Handy tricks
  - Priority?
    - `A &= ~(1 << 6);`
    - `A = A & ~0b01000000;`
    - `A = A & 0b10111111;`
    - Clear bit 6!

# RAM

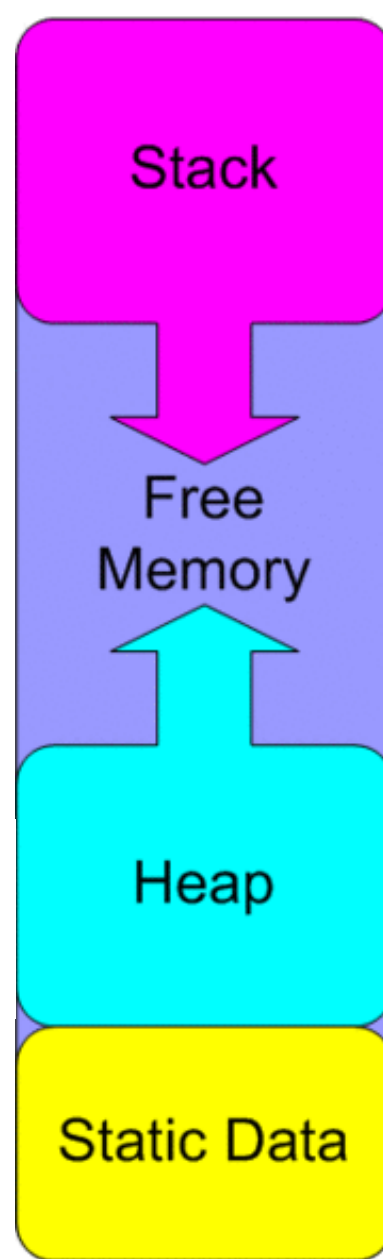
*Function calls*

- reclaimable!
- recursive fcts

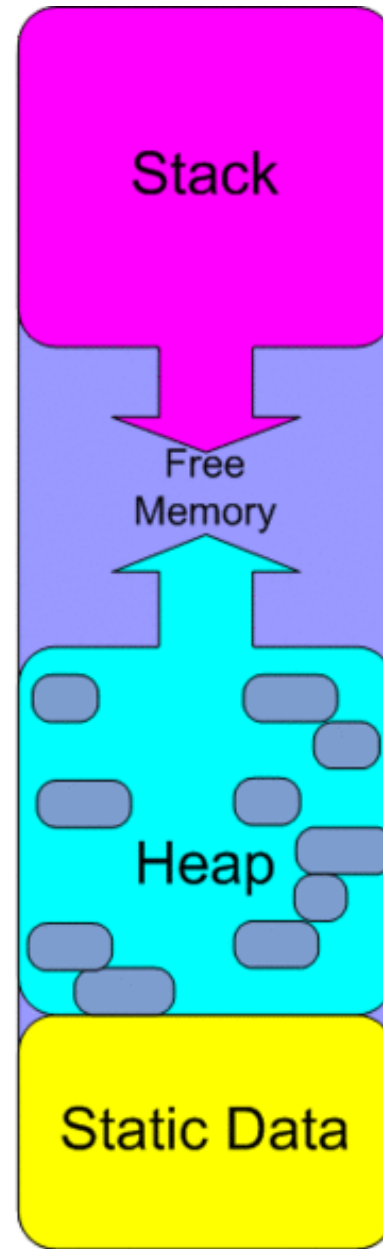
*Pick local over  
global/dynamically  
allocated variables!*

*Dynamically  
allocated objects*

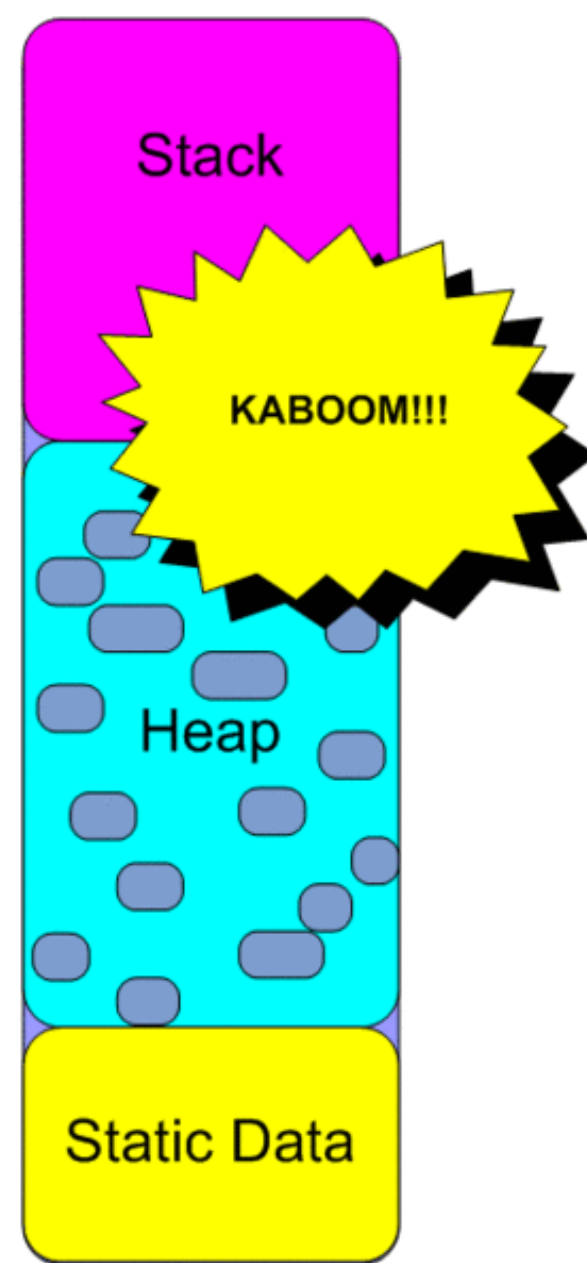
*Global/static variables*



Normal SRAM  
Operation



Fragmented Heap



Stack Crash!

# RAM

- Debugging strings
  - `Serial.print("Write something on the Serial Monitor for debugging");`
- Move constant variables to flash (instead of RAM)
  - `Serial.print(F("Write something on the Serial Monitor for debugging"));`
  - `#include <avr/pgmspace.h>`
  - `const dataType variableName[] PROGMEM = {"xxx"};     // use this form`
  - `String = pgm_read_word_near(variableName);`
- Library Buffers
- System Buffers

# ECE Colloquium: Fumin Zhang, Georgia Institute of Technology - Bio-Inspired Autonomy for Mobile Sensor Networks

Monday, Oct 30, 2017 at 4:30 PM until 6:00 PM [\[ics\]](#)  
Phillips Hall RM.233



There is an increasing trend for robots to serve as networked mobile sensing platforms that are able to collect data and interact with humans in various types of environment in unprecedented ways. The need for undisturbed operation posts higher goals for autonomy. This talk reviews recent developments in autonomous collective foraging in a complex environment that explicitly integrates insights from biology with models and provable strategies from control theory and robotics. The methods are rigorously developed and tightly integrated with experimental effort with promising results achieved.

## ECE Colloquia

[Spotlights](#)

[Faculty Awards and Honors](#)

[Connections Magazine](#)

[Video Gallery](#)

< October 2017 >

S	M	T	W	T	F	S
01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	<a href="#">23</a>	24	25	26	27	28
29	<a href="#">30</a>	31	01	02	03	04

[day](#) [week](#) [month](#)

[year](#) [today](#)

# *Go Build Robots!*



Class website: <https://cei-lab.github.io/ece3400/>

Piazza: <https://piazza.com/cornell/fall2017/ece3400/home>