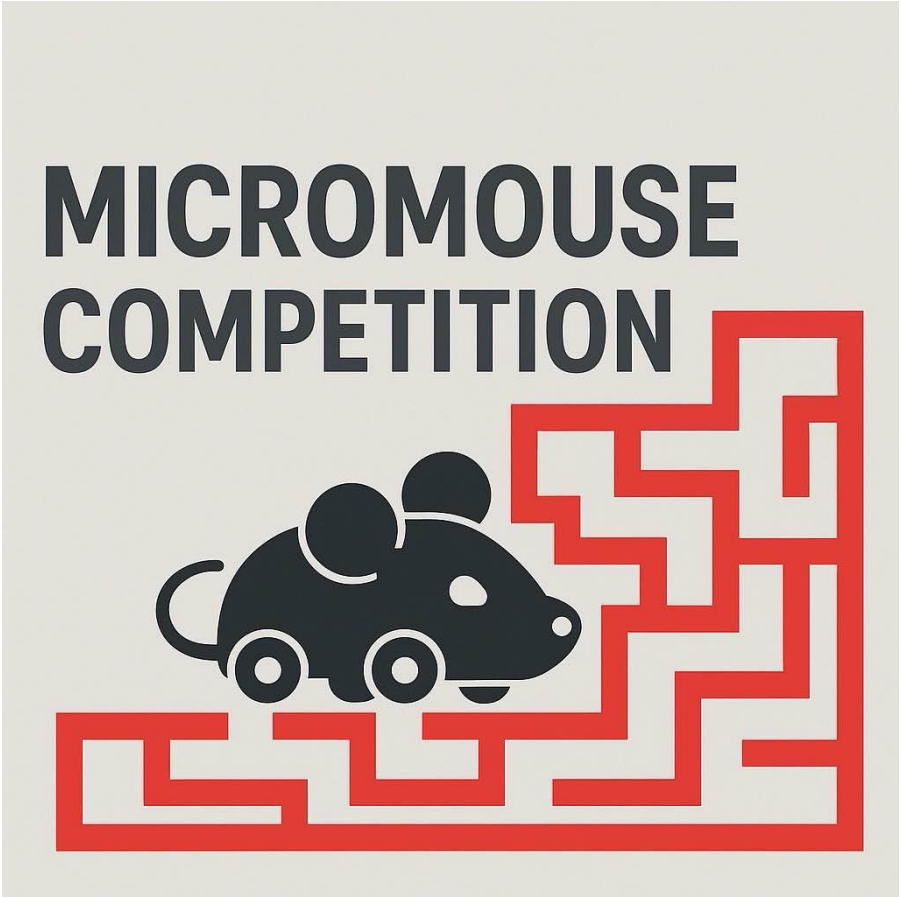
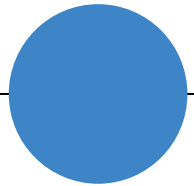
Micromouse

7/17/2025
Jing Jia

MAYFLOWER
A.S.T.E.M.
ACADEMY



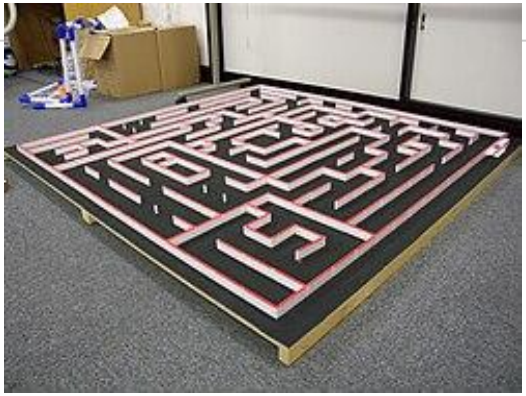
Micromouse

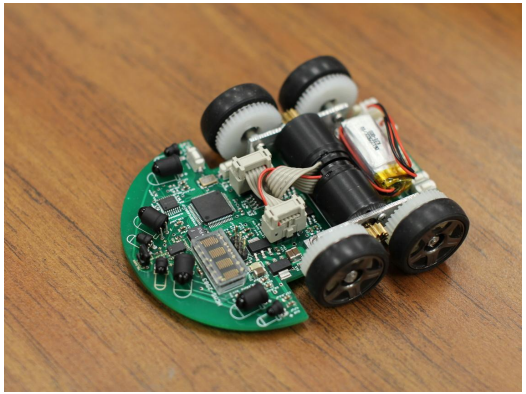


Micromouse

Micromouse is an event where small robotic mice compete to solve a 16×16 maze. It began in the late 1970s.

The maze is made up of a 16×16 grid of cells, each 180 square mm with walls 50 mm high.

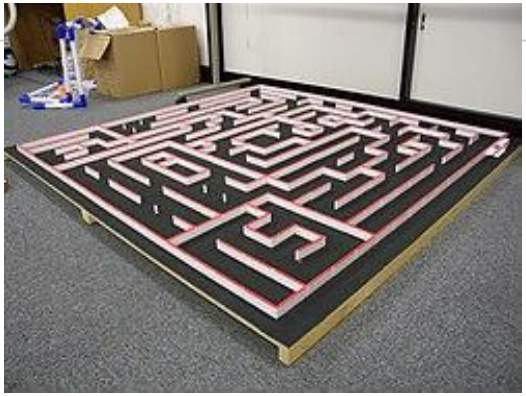


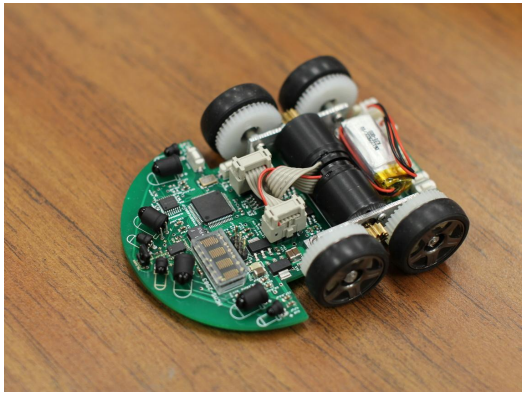


Micromouse

The mice are completely autonomous robots that must find their way from a predetermined starting position to the central area of the maze.

The mouse needs to keep track of where it is, discover walls as it explores, map out the maze and detect when it has reached the goal.

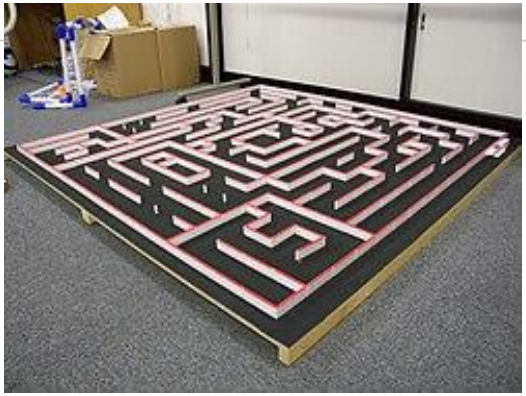


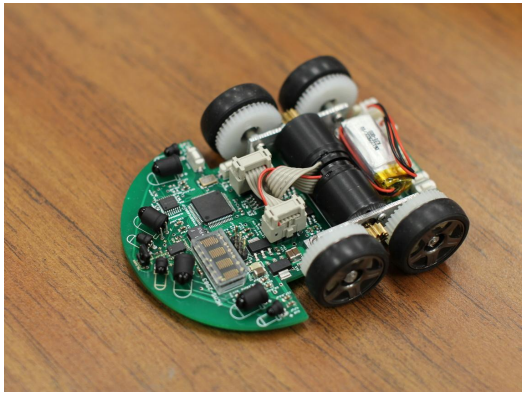


Micromouse

Having reached the goal, the mouse will typically perform additional searches of the maze until it has found an optimal route from the start to the finish.

Once the optimal route has been found, the mouse will traverse that route in the shortest achievable time.





Competition

https://urtc.mit.edu/

All IEEE Student Members and Graduate Student Members are eligible to participate. High School Students (non-IEEE Member) are also welcome to participate, but they need to attend a 30 minutes introduction session about IEEE and how IEEE can help them when they are an Undergraduate Student at a college.
From https://events.vtools.ieee.org/m/434766

Rules for the Micromouse

1. A Micromouse shall be self-contained (no remote controls). A Micromouse shall not use an energy source employing a combustion process.

2. A Micromouse shall not leave any part of its body behind while negotiating the maze.

3. A Micromouse shall not jump over, fly over, climb, scratch, cut, burn, mark, damage, or destroy the walls of the maze.

Rules for the Micromouse

4. A Micromouse shall not be larger either in length or in width, than 25 centimeters. The dimensions of a Micromouse that changes its geometry during a run shall not be greater than 25 cm x 25 cm. There are no restrictions on the height of a Micromouse.

5. Any violation of these rules will constitute immediate disqualification.

Rules for the Maze

1. The maze is composed of multiples of an 18 cm x 18 cm unit square. The maze comprises 16 x 16 unit squares. The walls of the maze are 5 cm high and 1.2 cm thick (assume 5% tolerance for mazes). The outside wall encloses the entire maze.

2. The sides of the maze walls are white, the tops of the walls are red, and the floor is black. The maze is made of wood, finished with non-gloss paint.

Rules for the Maze

WARNING: Do not assume the walls are consistently white, or that the tops of the walls are consistently red, or that the floor is consistently black. Fading may occur; parts from different mazes may be used. Do not assume the floor provides a given amount of friction. It is simply painted plywood and may be quite slick. The maze floor may be constructed using multiple sheets of plywood. Therefore, there may be a seam between the two sheets on which any low-hanging parts of a mouse may snag.

Rules for the Maze

3. The start of the maze is located at one of the four corners. The start square is bounded on three sides by walls. The start line is located between the first and second squares. That is, as the mouse exits the corner square, the time starts. The destination goal is the four cells at the center of the maze. At the center of this zone is a post, 20 cm high and each side 2.5 cm. (This post may be removed if requested.) The destination square has only one entrance.

Rules for the Maze

4. Small square zones (posts), each 1.2 cm x 1.2 cm, at the four corners of each unit square are called lattice points. The maze is so constituted that there is at least one wall at each lattice point.

5. Multiple paths to the destination square are allowed and are to be expected. The destination square will be positioned so that a wall-hugging mouse will NOT be able to find it.

Rules for the Contest

1. Each contesting Micromouse is allocated a total of 10 minutes of access to the maze from the moment the contest administrator acknowledges the contestant(s) and grants access to the maze. Any time used to adjust a mouse between runs is included in the 10 minutes. Each run (from the start cell to the center zone) in which a mouse successfully reaches the destination square is given a run time. The minimum run time shall be the mouse's official time. First prize goes to the mouse with the shortest official time. Second prize to the next shortest, and so on.

Rules for the Contest

2, Each run shall be made from the starting square. The operator may abort a run at any time. If an operator touches the Micromouse during a run, it is deemed aborted, and the mouse must be removed from the maze. If a mouse has already crossed the finish line, it may be removed at any time without affecting the run time of that run. If a mouse is placed back in the maze for another run, a one-time penalty of 30 seconds will be added to the mouse's next run time.

Rules for the Contest

3. After the maze is disclosed, the operator shall not feed information on the maze into the Micromouse however, switch positions may be changed.

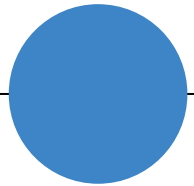
4. The illumination, temperature, and humidity of the room shall be those of an ambient environment. (40 to 120 degrees F, 0% to 95% humidity, non-condensing).

BEWARE: Do not make any assumptions about the amount of sunlight, incandescent light, or fluorescent light that may be present at the contest site.

Rules for the Contest

5. The run time will start when front edge of the mouse crosses the start line and stops when the front edge of the mouse crosses the finish line. The start line is at the boundary between the starting unit square and the next unit square clockwise. The finish line is at the entrance to the destination square.

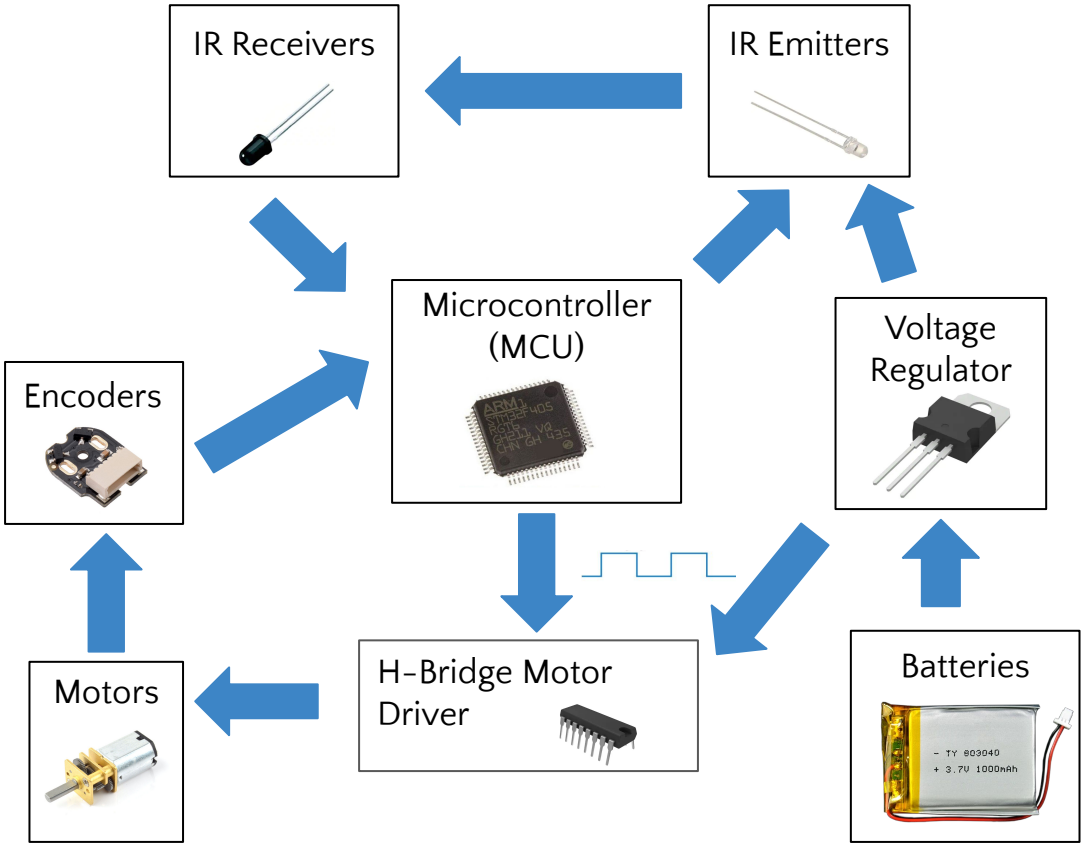
Hardware



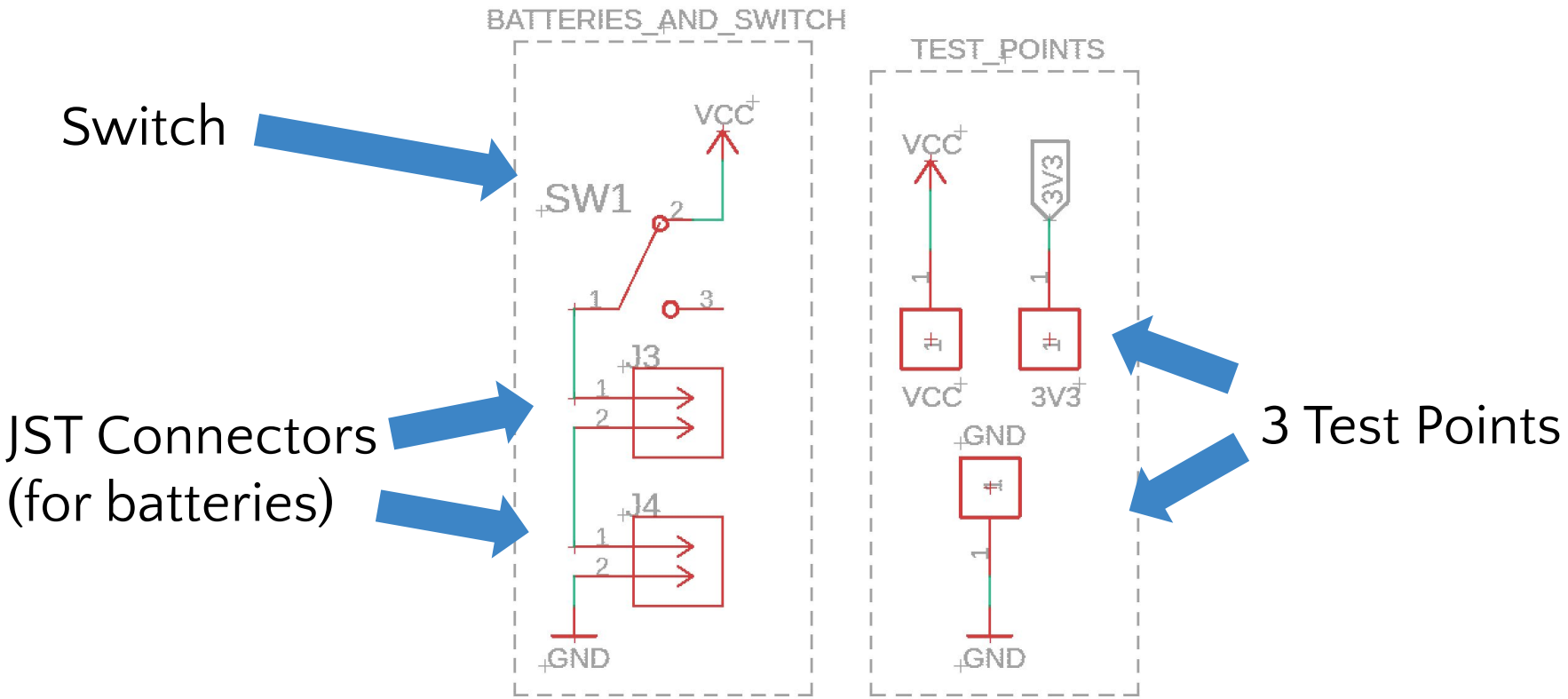
1 Power Circuit

Overview

- Power Circuit
- MCU
- Motors
- IRs
- Other components
- Advanced Topics:
  - Diagonal LEDs
  - Curve Turn
  - Saving maze

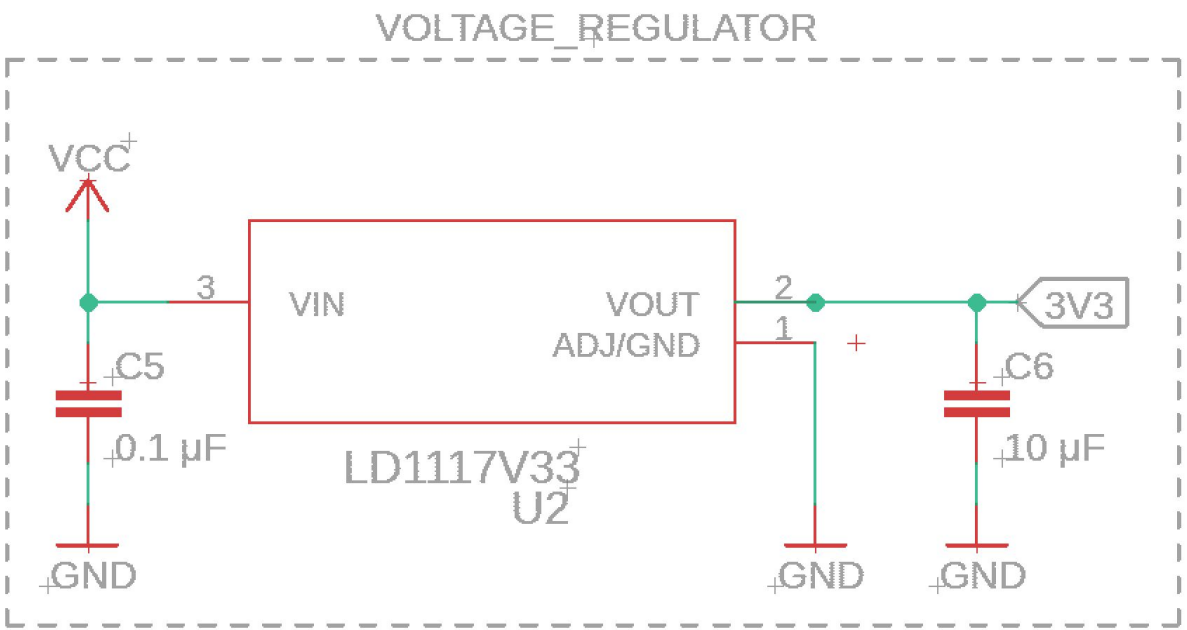


Batteries and Test Points



Voltage Regulator

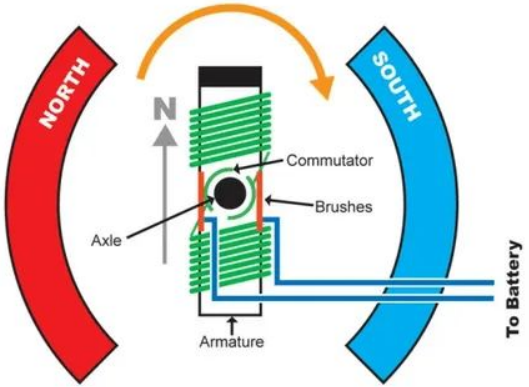
- Set up as indicated in the data sheet



2 Motors

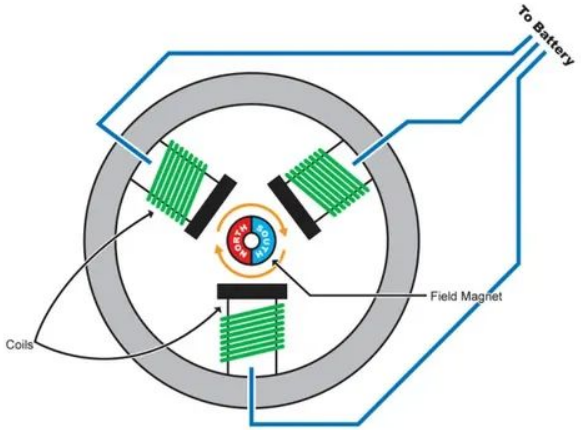
Motor Types

Brushed



- Cheap and Simple
- Easy to work with
- Less precision
- Less efficiency

Brushless



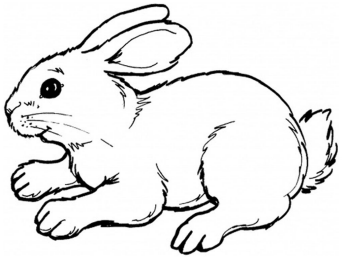
- Expensive and Complex
- Harder to work with
- More precision
- More efficiency

Motors used on the Rat Kits

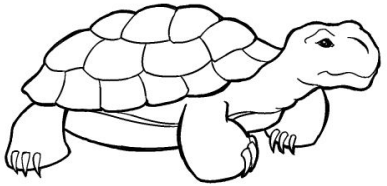
- Brushed
- 30:1 Gear Ratio
- 6 V Ideal Operating Voltage
- Maximum current draw of 0.67 A

Gear Ratio Speed vs Precision

15:1 -> More options for speed, but encoder counts half as precise



50:1 -> Lower max speed, but more precise encoder counts



30:1 -> A nice balance, good for beginners

Checklist for Motor Control

- ❑ Motor On/Off State
- ❑ Motor Direction
- ❑ Motor Speed

Motor On/Off State

We can't power the motor directly from the MCU!!

Why? Not enough voltage or current from the MCU

Big boi battery power

MCU power

Motors require at least 6V and can draw up to 0.67A

MCU can only supply 3.3V and relatively little current

The Problem: Motor Speed

Since we are directly connecting the motors to the battery voltage, we only have two speeds -> 0% and 100%

How do we tell the motors to go 50% speed? 75%?

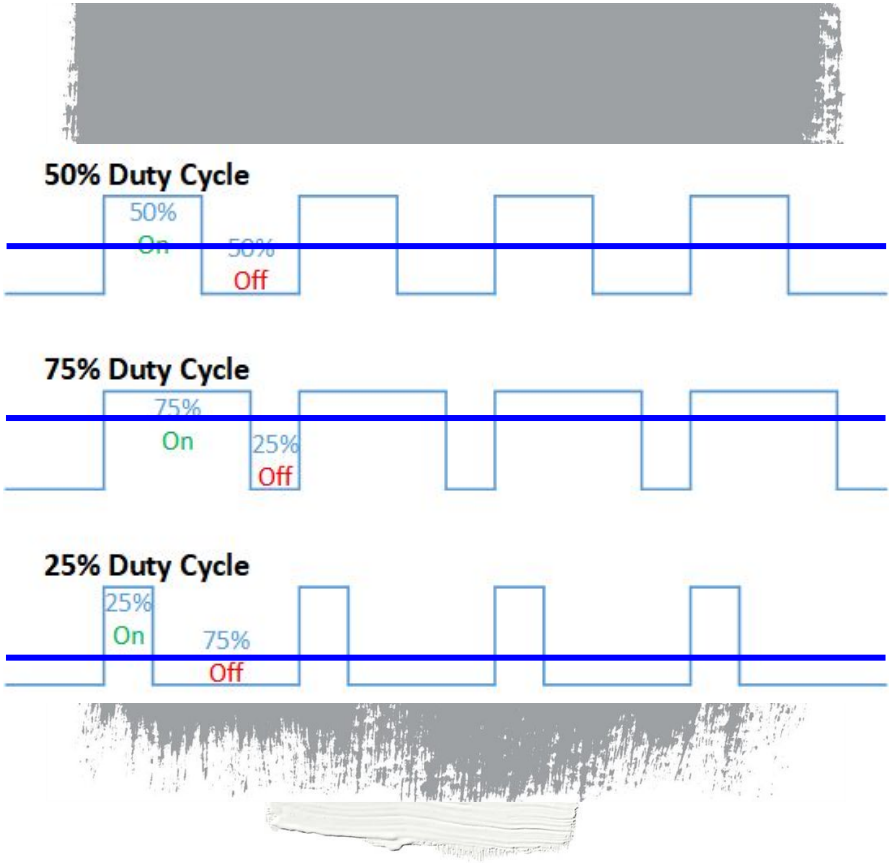
PWM

We can rapidly mix on and off inputs to create a range of voltages

Motors have inertia, so they respond slowly to rapid voltage changes

Quick Vocab Terms:

- PWM: Pulse-Width Modulation
- Duty Cycle: % of time in the ON state
- Frequency: # of cycles per second



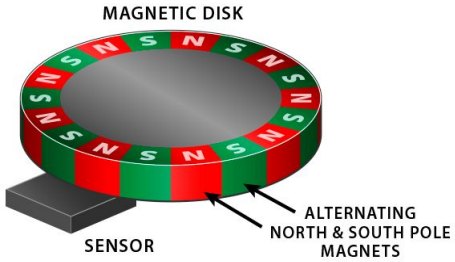
One Final Piece

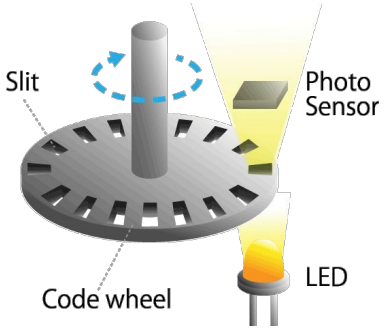
Encoders

Encoders continually measure the rotation of the motor

Gives us a way to tell how **far** and **fast** we are traveling!

Magnetic vs Optical Encoders





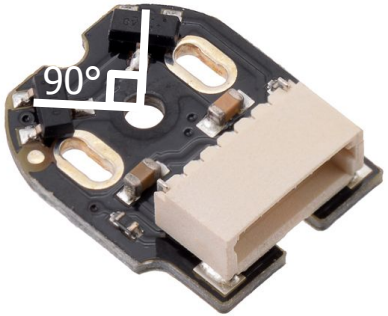
Magnetic Encoders

Senses change in rotational position as motor spins

6 pole magnetic disk (60° between each pole)

2 Hall Effect Sensors (90° between each sensor

- Detects the presence of a magnetic field



Hall Effect: Production of a voltage difference from a current and a magnetic field

Encoders

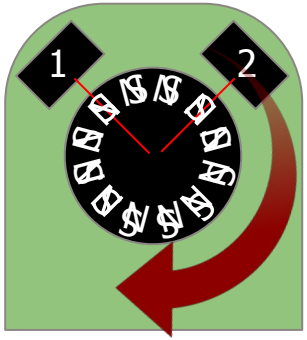
Magnetic Poles are 60° apart

Hall Effect Sensors are 90° apart

⇒ 30° per encoder count

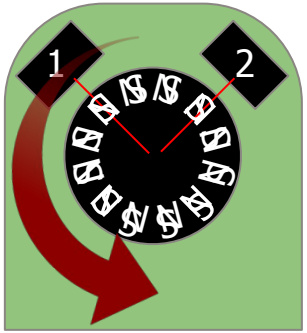
Spin direction: determined from phase difference between the two sensors

Clockwise



CW	Sensor 1	Sensor 2
0°	N	-
30°	-	S
60°	S	-
90°	-	N
120°	N	-

Counterclockwise



CCW	Sensor 1	Sensor 2
0°	N	-
30°	-	N
60°	S	-
90°	-	S
120°	N	-

How Precise are Encoders?

Let's do some math :)

Encoders Count every **30°**

(360°/rotation) / (30°/count) = **12 counts/full motor rotation**

12 Counts/motor rotation * 30:1 Gear ratio = **360 counts/wheel rotation**

Wheel Dimension: 3.2 cm Diameter ⇒ 10 cm Circumference

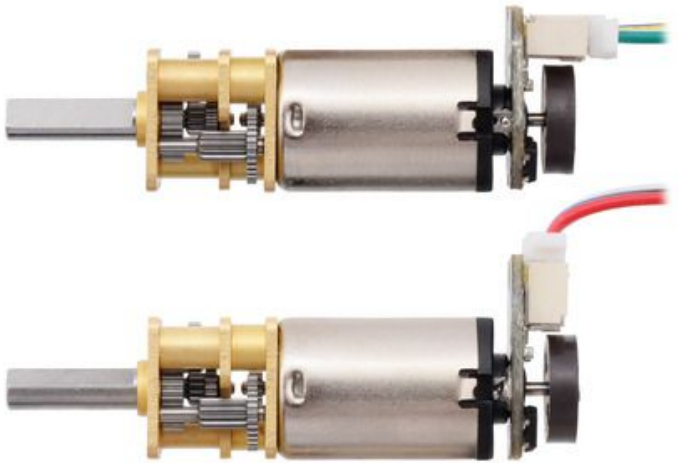
360 counts/rotation / (10 cm / rotation) = **36 counts/cm**

Encoder Options - Top vs Side Entry

Affects how close motors can be placed

Also may affect battery placement

Top Entry

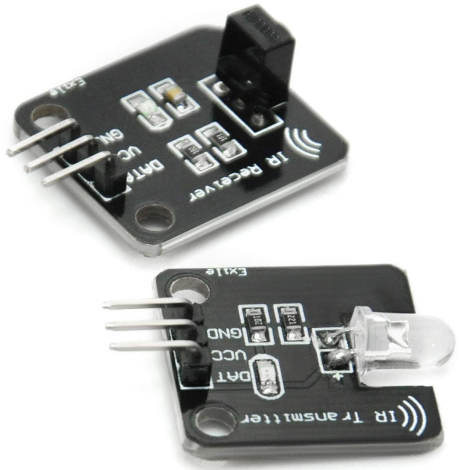


Side Entry

3 IRs

How IR Sensors Provide Distance Information

IR (Infrared) sensors used in setups like this typically consist of:



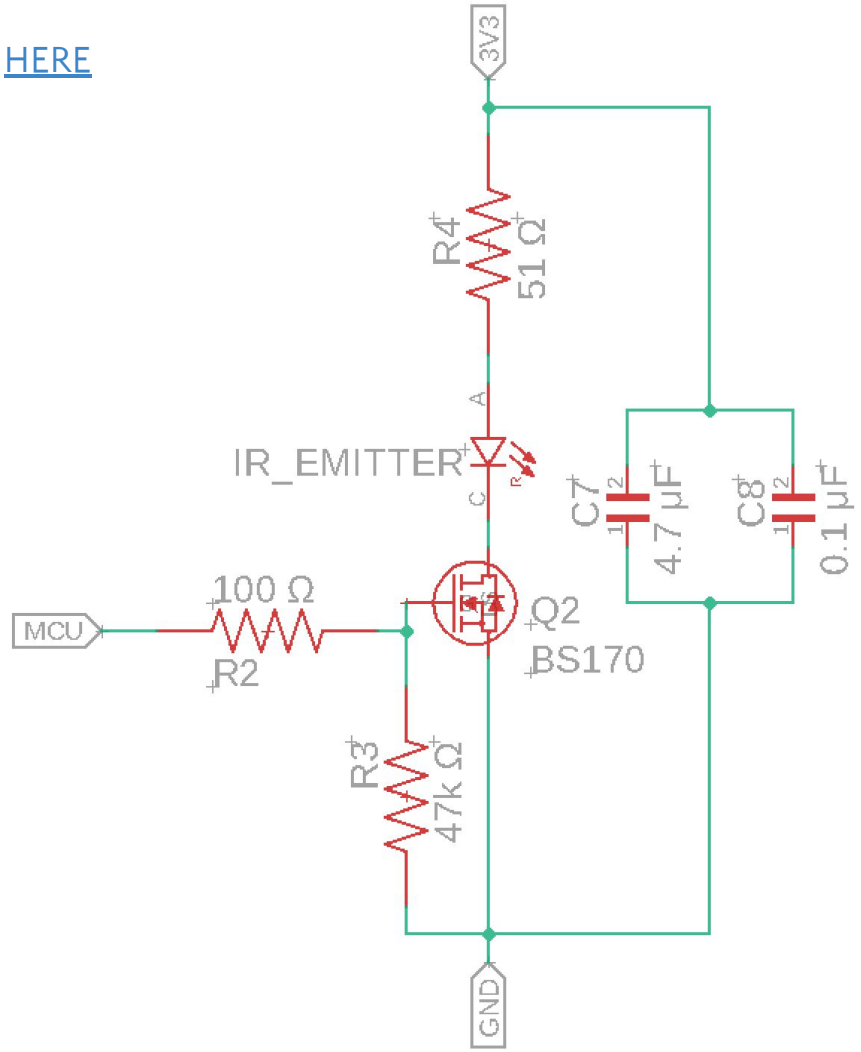
- An IR LED (Emitter): emits infrared light.
- A photodiode or phototransistor (Receiver): detects reflected IR light from nearby objects.

The system measures the amount of reflected light, not time-of-flight like LiDAR.

For more info about these circuits, click HERE

IR Emitter Circuit

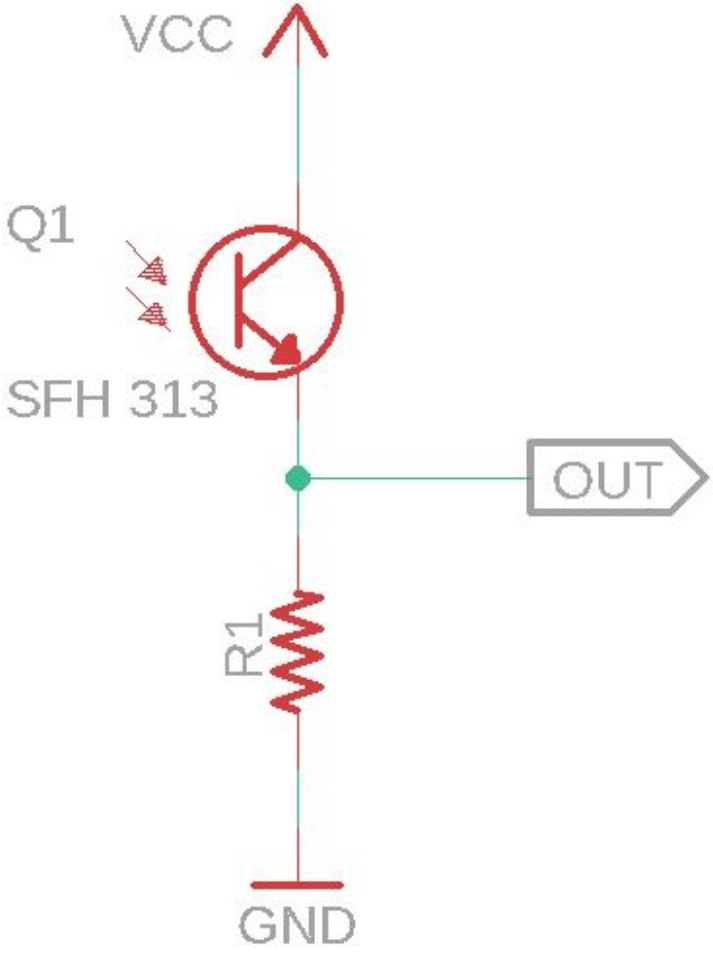
- Uses an IR LED to emit IR light
- Need a transistor to switch on / off
  - This is because the MCU does not supply enough current by itself
- MCU pin → high
  Transistor → on
  LED → high



For more info about these circuits, click HERE

IR Receiver Circuit

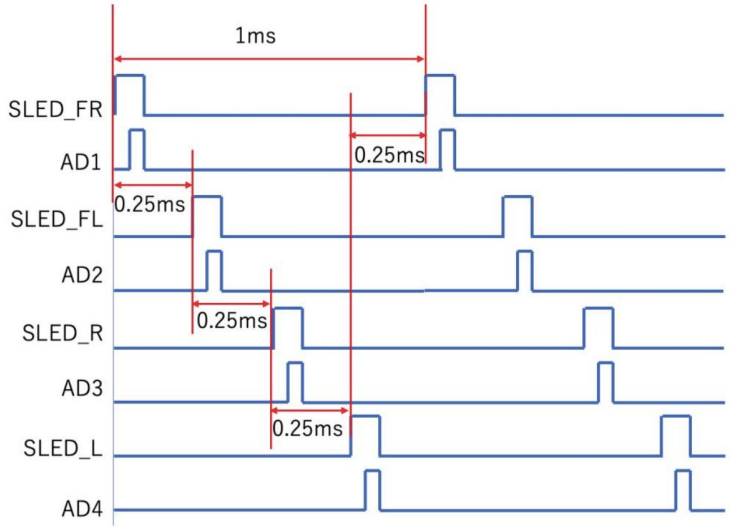
- Phototransistor to measure reflected IR light
- Resistor to create voltage divider
- Voltage between is measured by the MCU's ADC (Analog to Digital Converter)



Example

The sensor emits light using a timer. It emits light in the order of front right, front left, right, and left at 0.25 ms intervals. It completes one cycle in 1 ms.

The timing of the light emission and the A/D timing chart are shown.



Example

1. The IR LED is turned on momentarily.
2. If there's an object nearby, IR light reflects off the object back to the sensor.
3. The photodiode detects this reflected light.
4. The analog value increases with the intensity of reflected IR, which decreases as distance increases.

Closer object → more IR reflection → higher analog reading.

Farther object → less IR reflection → lower analog reading.

4 MCU

MCU

A microcontroller (MC, uC, or µC) or microcontroller unit (MCU) is a small computer on a single integrated circuit. A microcontroller contains one or more CPUs (processor cores) along with memory and programmable input/output peripherals.

- Arduino
- ESP32
- STM32
- Teensy
- PIC
- MSP430
- NXP
- BeagleBone
- Renesas RX Family
- Infineon XMC Family
- ELAN Microelectronics

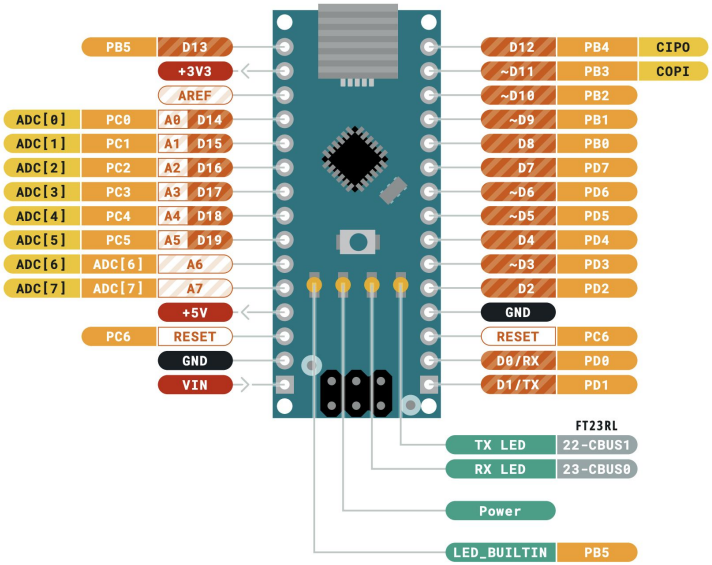
https://github.com/Slayingripper/MicroControllers

Arduino Nano

https://docs.arduino.cc/hardware/nano/

Arduino Nano is the first embedded microcontroller in the Nano series with minimum functionalities, designed for mini projects from the maker community.

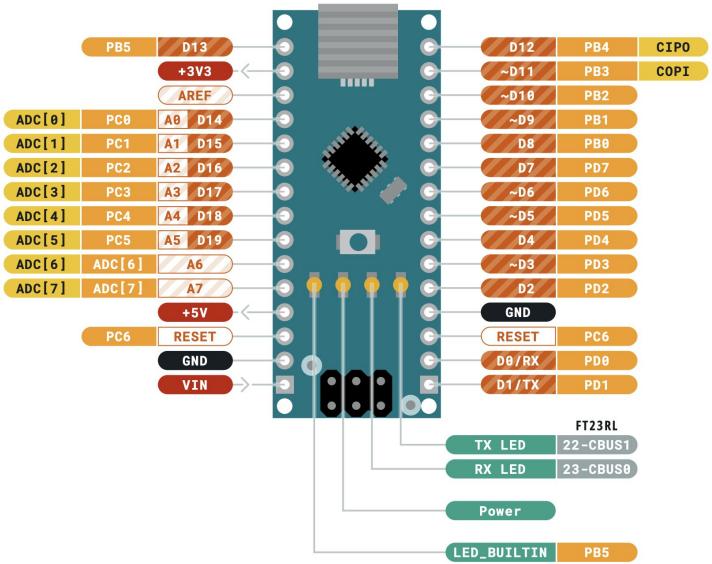
With a large number of input/output pins gives the advantage of utilizing several serial communications like UART, SPI and I2C. The hardware is compatible with Arduino IDE, Arduino CLI and Cloud Editor.



Arduino Nano

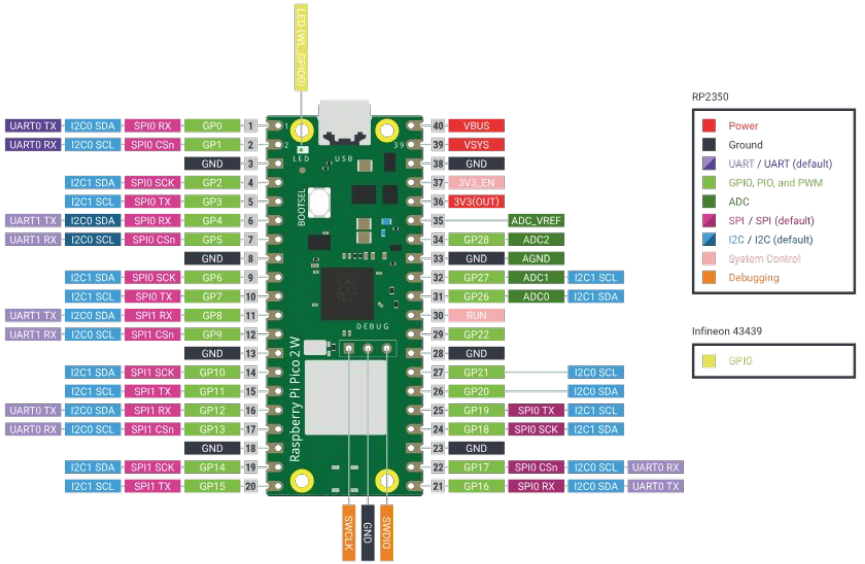
https://docs.arduino.cc/hardware/nano/

The primary processor in the Nano board is the high-performance and low-power 8-bit ATmega328 microcontroller that runs at a clock frequency of 16 MHz.



Raspberry Pi Pico 2 W

Raspberry Pi Pico 2 W adds on-board single-band 2.4GHz wireless interfaces (802.11n) using the Infineon CYW43439 to the Pico 2 hardware.



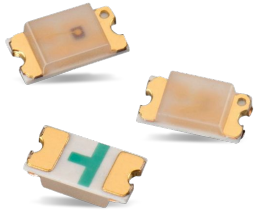
5

Other Components

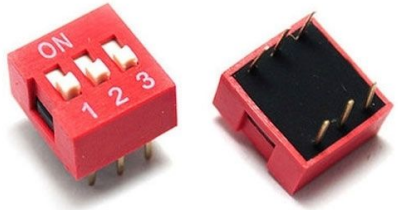
You guys probably already have some of these in mind to make your mice cool

LEDs

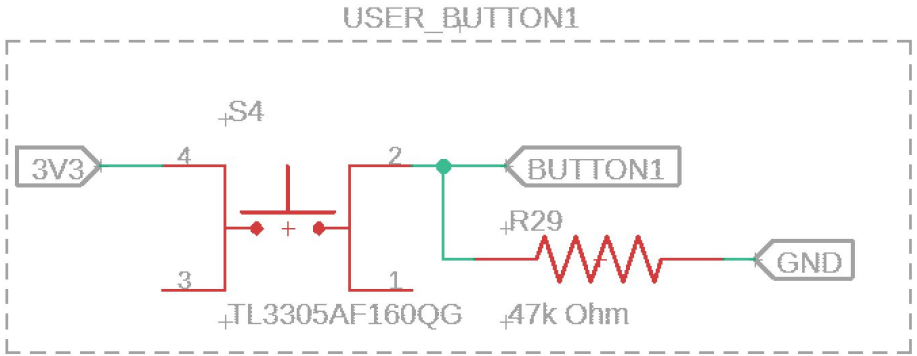
- Pay attention forward voltage and current draw
  - You can often round up for the resistors cuz we don't need that much brightness
- Neopixels are a fun choice for individually addressable rainbow LEDs



Buttons and Switches



- Allows users different modes and settings
- Make sure they are easy to reach on board
- DIP switches are a space efficient way to add more user interface
- Use a pulldown or pullup resistor



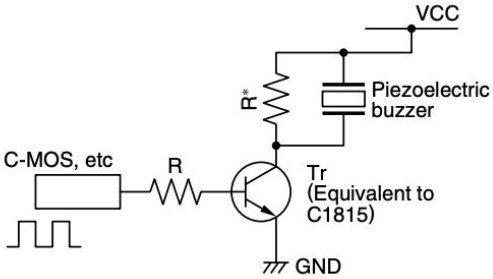
Buzzers

- Use as a status indicator, or more!
- Types:
  - Active vs Passive: active has a set frequency, passive can change frequency based off voltage PWM
  - Piezo vs Magnetic: different ways to make the buzzer make sound

Similar to our IR Emitters being turned on and off, buzzers can be set up with a MOSFET.

For passive buzzers: have the MOSFET input be a PWM signal to be able to change pitch frequency

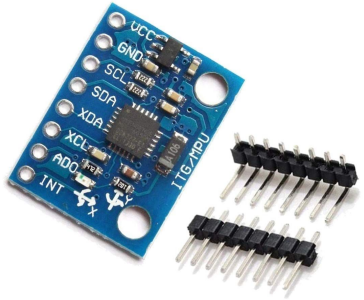
RECOMMENDED OPERATING CIRCUIT EXAMPLE



* Resistor to do charging and discharging to a piezoelectric element (Value of about 1kΩ is good efficiency).

Example circuit from TDK's data sheet

Gyroscopes/IMUs



Gives angular velocity

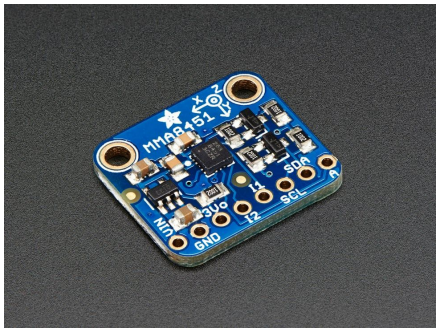
To get angular position, integrate angular velocity in code

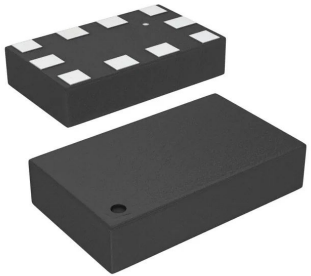
Recommended: Digital I2C gyroscope breakout board

Alternative: Analog z-axis gyroscope

- Outputs voltage between 0-100% of input power based on yaw
- Space efficient but a hassle to set up and solder

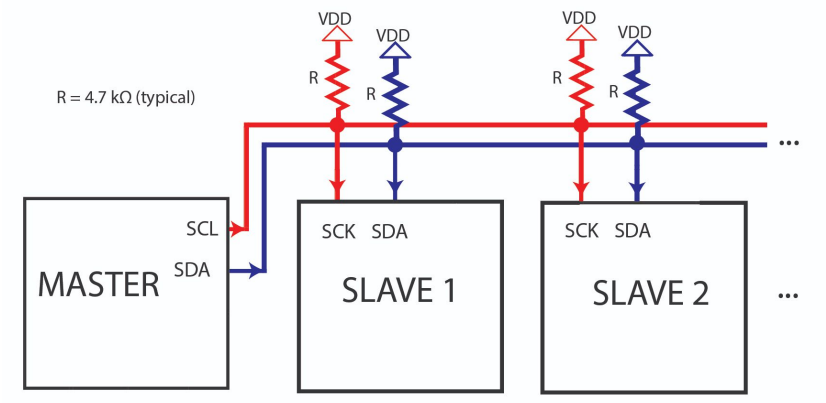
Examples: MPU-6050, MMA8451, LY3200ALHTR

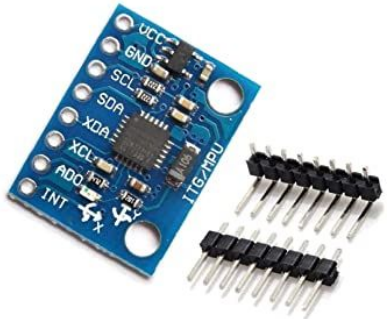




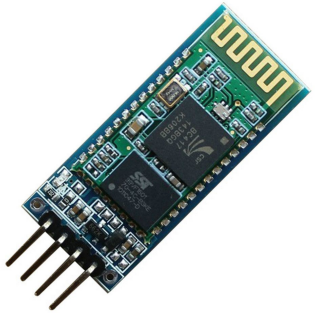
I2C Devices

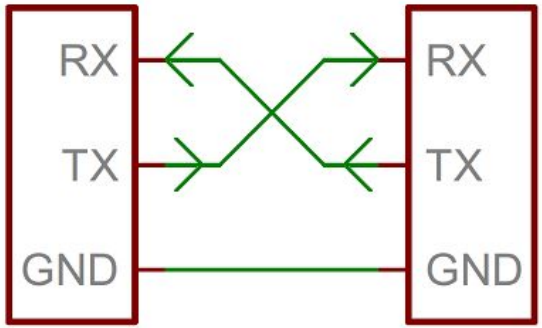
- Gyro breakout boards, OLED screens and other displays
- Can connect multiple I2C devices to the same pins of MCU
- Each I2C device has a unique address, which is how it can tell one type of device from another
- Examples: MPU6050, MMA8451





UART Devices



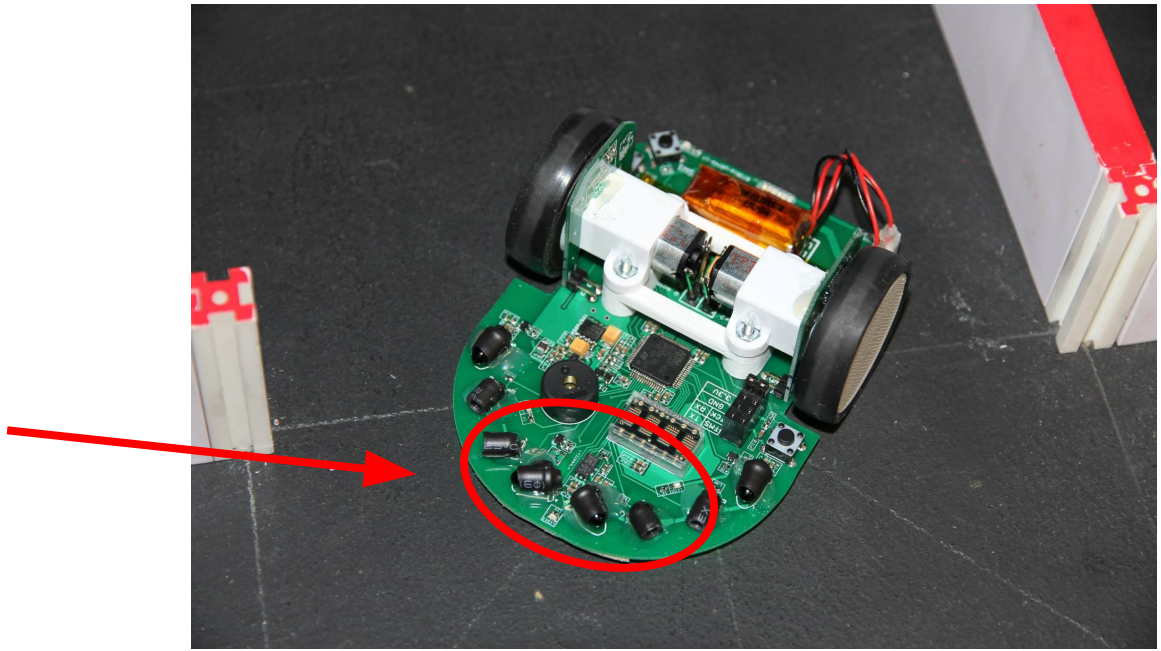


- Most common MM use is for a bluetooth module
- TX and RX are for “transmit” and “receive”
- The TX of one device connects to the RX of another, and vice versa
- Pls don’t get them confused :)
- Examples: HC-05, HC-06

6 Diagonal LEDs

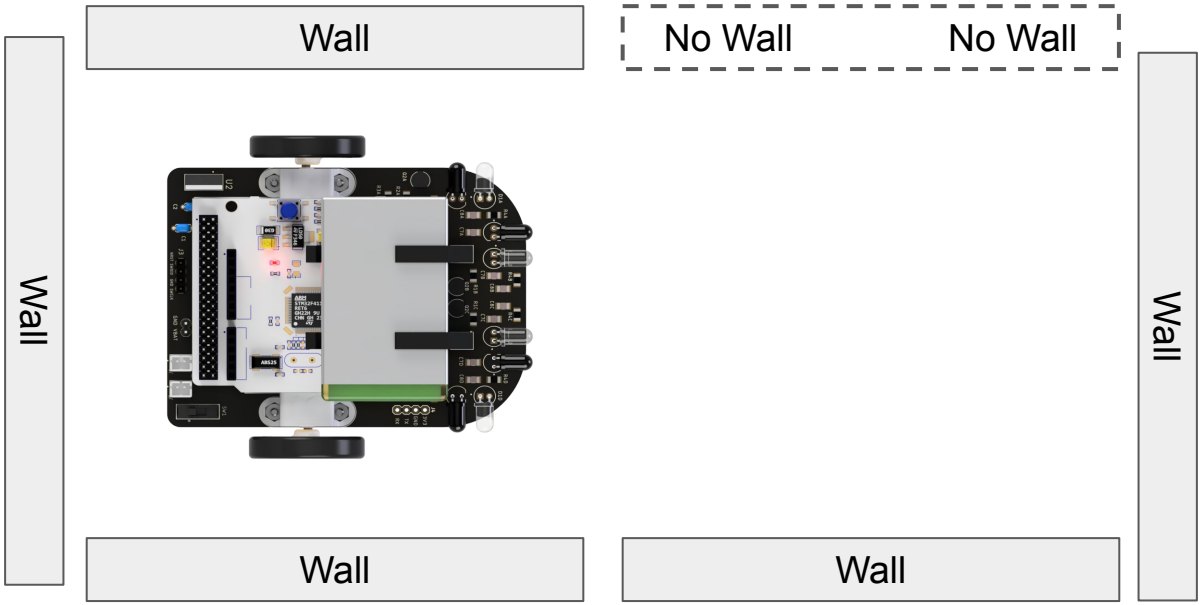
Diagonal LEDs

Allows mouse to know the walls before getting to the center of a cell



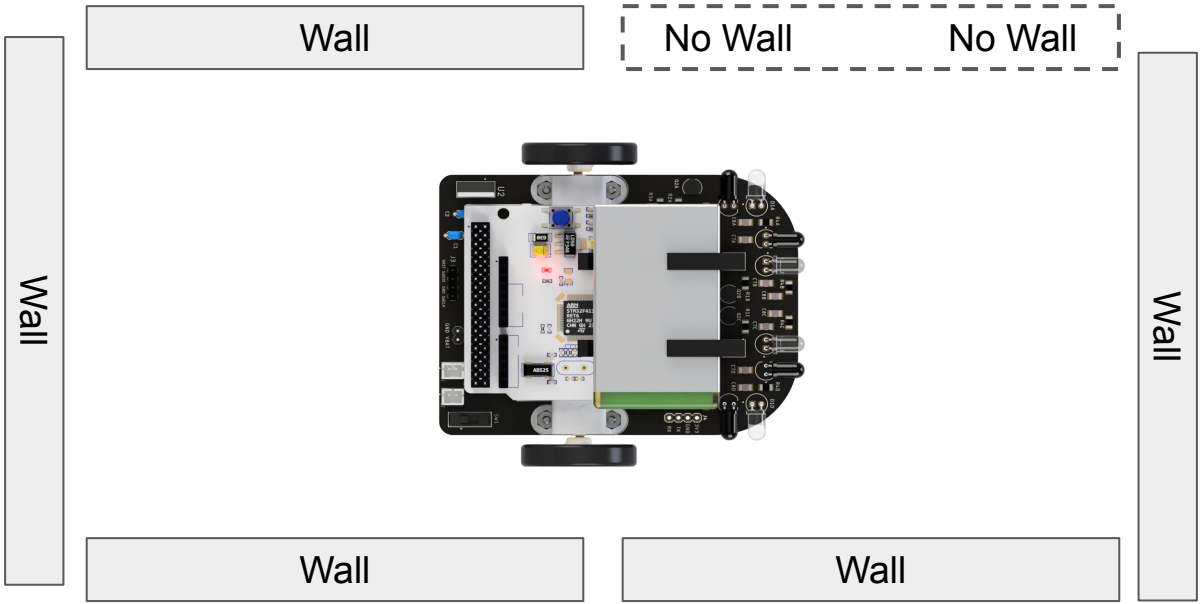
Diagonal LEDs

Allows mouse to know the walls before getting to the center of a cell



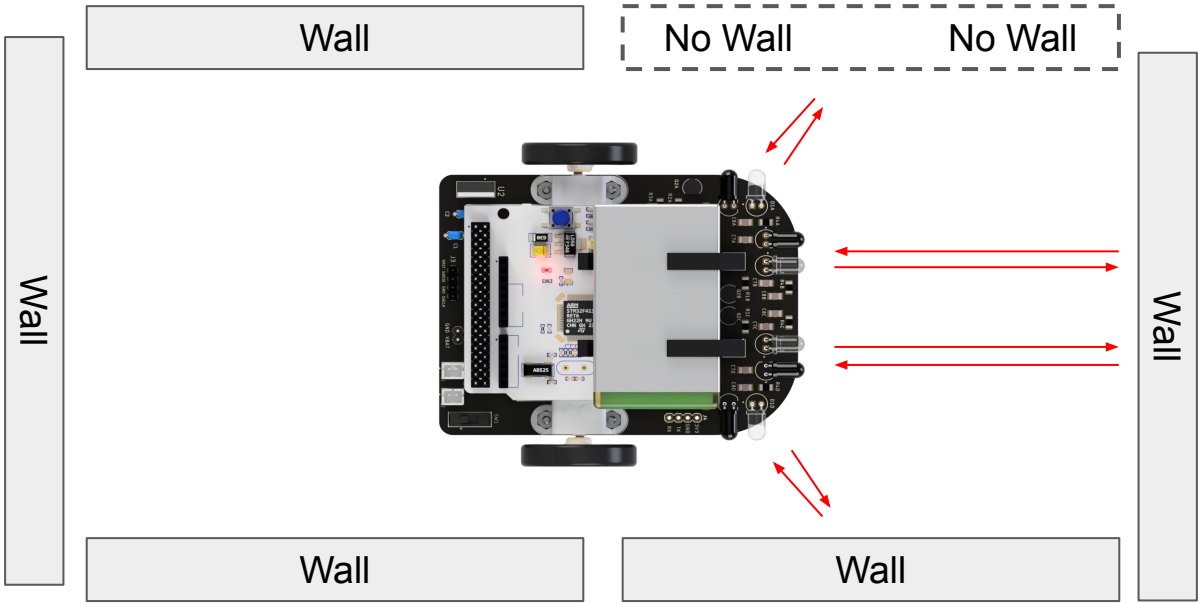
Diagonal LEDs

Allows mouse to know the walls before getting to the center of a cell



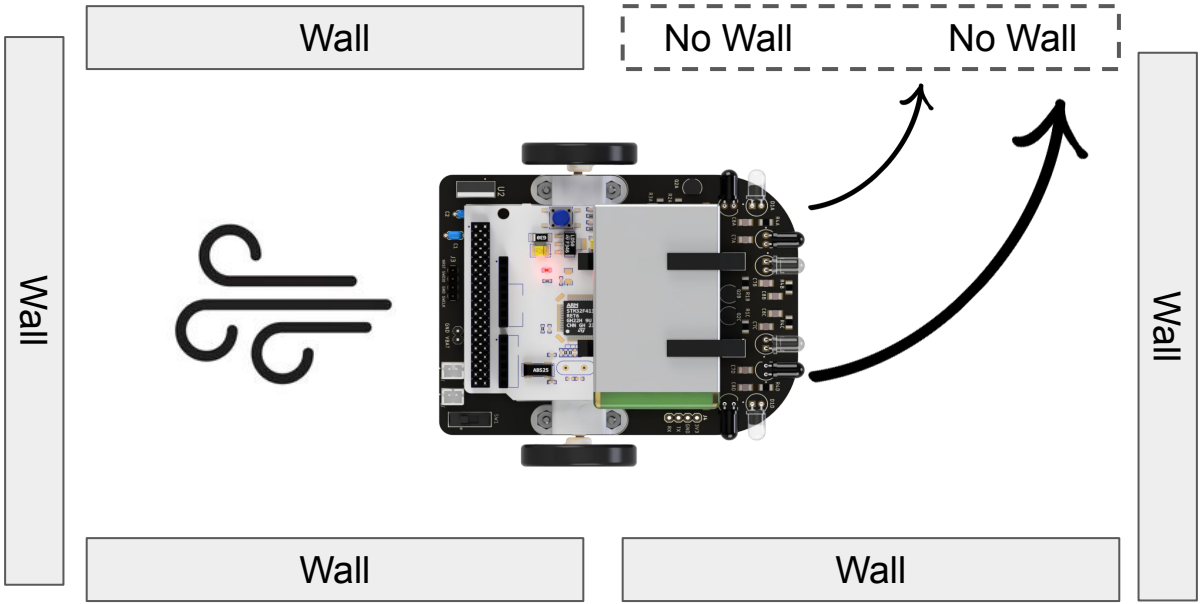
Diagonal LEDs

Allows mouse to know the walls before getting to the center of a cell



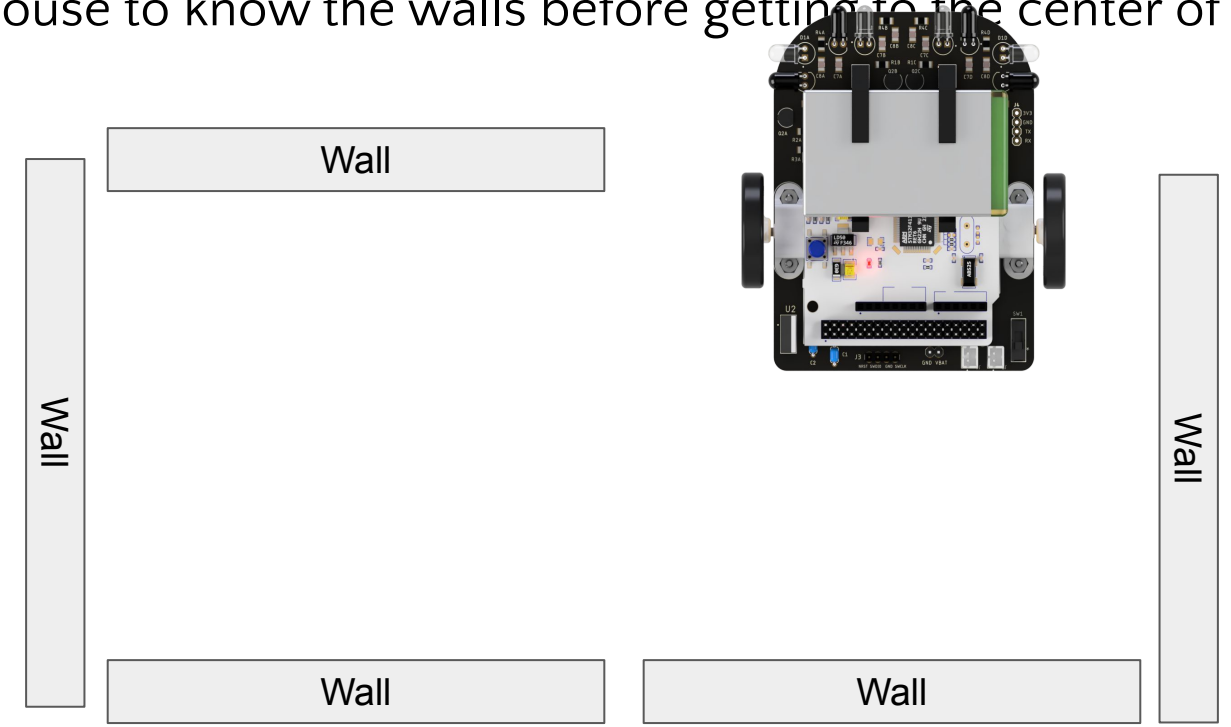
Diagonal LEDs

Allows mouse to know the walls before getting to the center of a cell



Diagonal LEDs

Allows mouse to know the walls before getting to the center of a cell

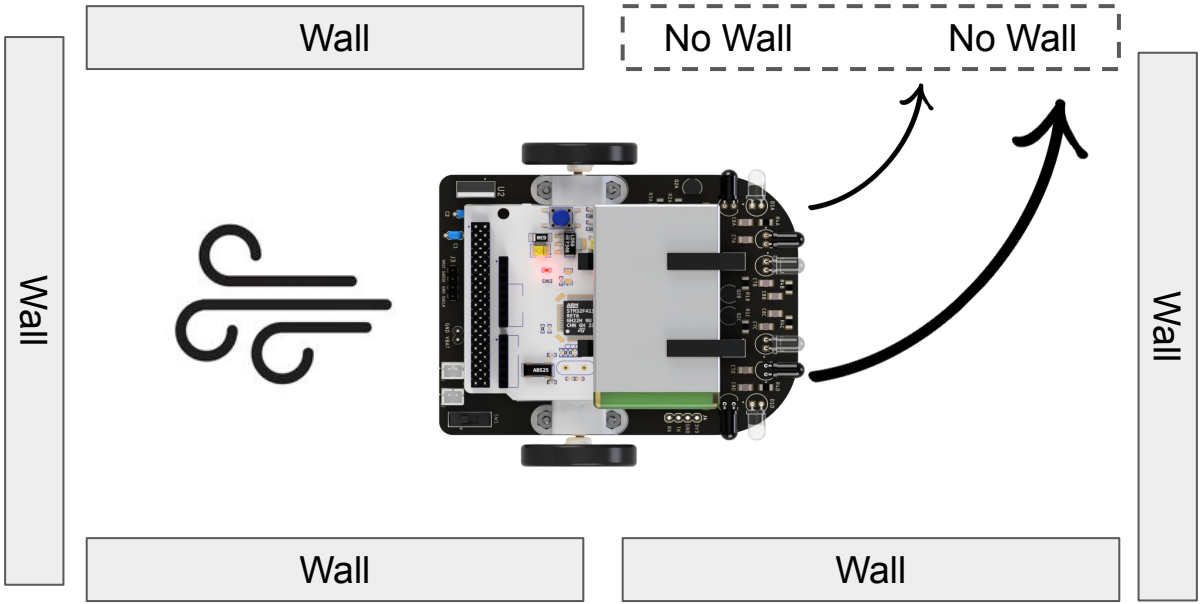


7

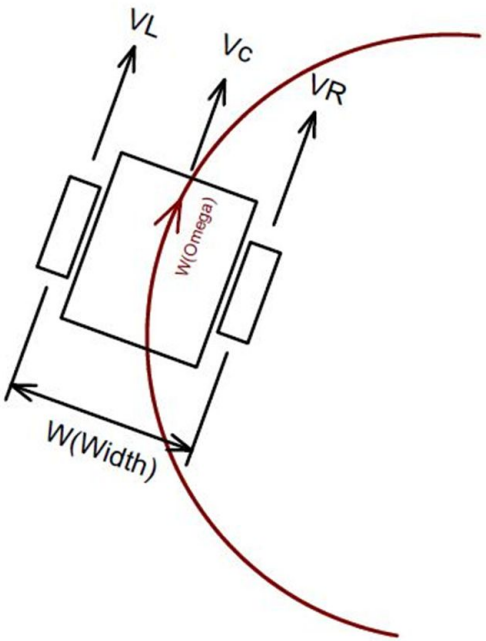
Take Turns

Curve Turn

So about that turn we were just talking about...



Curve Turn

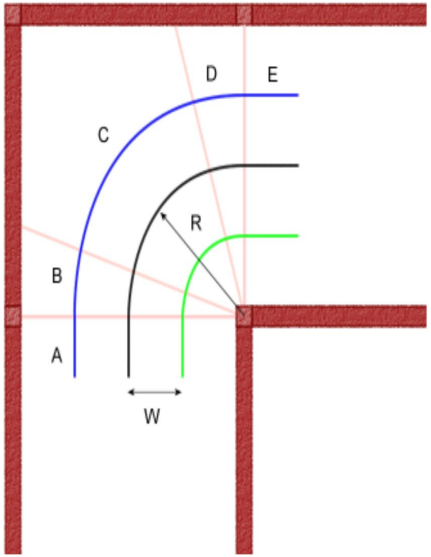


How to do this? Lots of math lol

For consistent results, must be able to set VELOCITY of left and right wheel separately

This would require a separate controller different from the distance and angular controller we've been using thus far

Need to calculate the desired speed of the inner and outer wheels to obtain desired curve



- There is 180mm from wall to wall, so R = 90mm
- Depends on size of mouse, but lets assume that both wheels are 40mm from the center, so W = 40mm

Decide an average speed/speed of center: 100mm/s

Path length/distance we travel (¼ of circumference):

- **P = 2 * π * R * ¼ = 2 * 3.14 * 90mm / 4 = 141.37mm**

Time for entire maneuver if traveling at avg speed:

- **Path/avg speed = 141.37mm / 100mm/s = 1.4137s**

Distance covered by outer wheel:

- **P = 2 * π * (R + W) * ¼ = 2 * 3.14 * (90mm + 40mm) / 4 = 204.1mm**

Required outer wheel velocity:

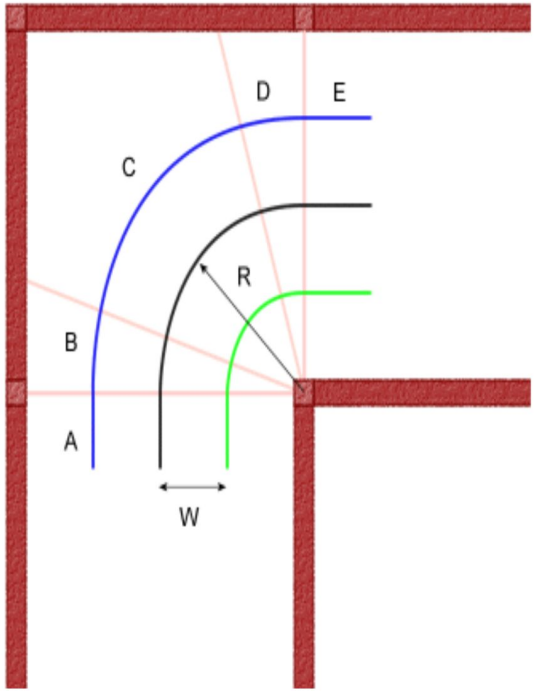
- **P = Outer wheel distance / time = 204.1mm / 1.4137s = 144.5mm/s**

Distance covered by inner wheel:

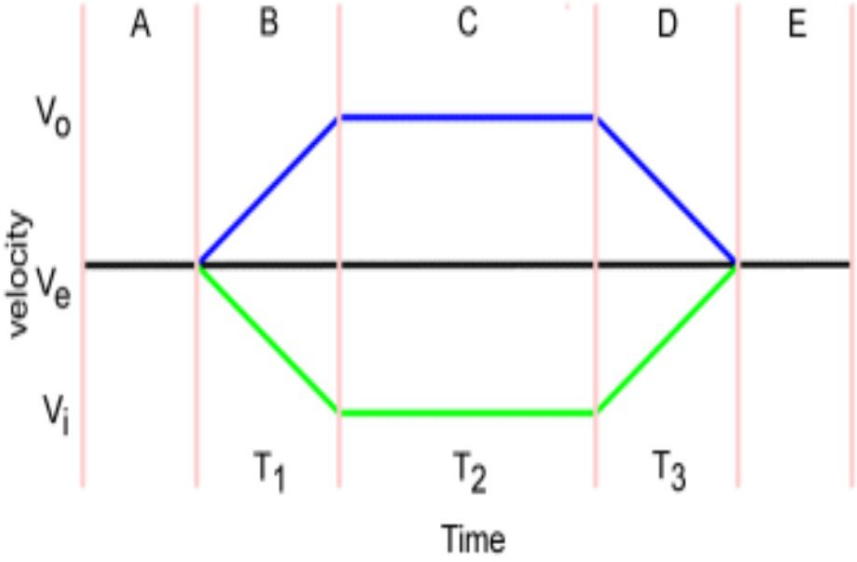
- **P = 2 * π * (R – W) * ¼ = 2 * 3.14 * (90mm – 40mm) / 4 = 78.5mm**

Required inner wheel velocity:

- **P = Inner wheel distance / time = 78.5mm / 1.4137s = 55.6mm/s**



V_e = Speed of center = 100mm/s
V_o = Speed of outside = 144.5mm/s
V_i = Speed of inside = 55.6mm/s



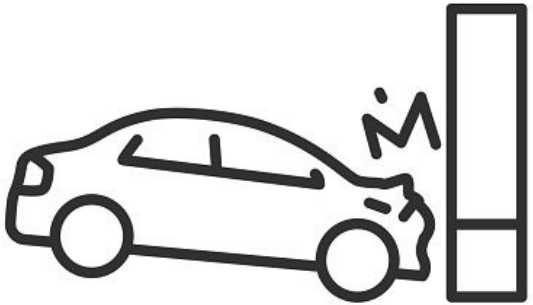
References:
https://micromouseonline.com/micromouse-book/robot-dynamics/smooth-corners/

http://micromouseusa.com/?page_id=1342

8 Saving Maze

Saving to Flash

- Backup plan in case of crash
- We don't want to lose all of our maze data!
- Solution? Save data to MCU's memory
- Now data is saved even when mouse is reset



Arduino Nano

The microcontroller on the Arduino and Genuino AVR based board has EEPROM: memory whose values are kept when the board is turned off (like a tiny hard drive). This library enables you to read and write those bytes.

Arduino Nano: 1 kB EEPROM

Maybe...

- A 16×16 maze has 256 cells. Each cell may have walls on 4 sides (N, E, S, W).
- You likely want to store for each cell:
- Which walls exist: 4 bits (1 bit per wall)
- 256 cells × 4 bits = 1024 bits = 128 bytes

https://docs.arduino.cc/learn/programming/eeprom-guide/

https://docs.arduino.cc/learn/built-in-libraries/eeprom/

Raspberry Pico W

RP2040 microcontroller with 2MB of flash memory

The main difference being that EEPROM is byte-wise erasable and can endure around 100k write cycles, while flash can only be erased in relatively large chunks and wears out considerably sooner, perhaps after ~20k cycles. But, there's a lot more of the flash, so that makes the durability issue a bit less painful.

Flash

This flash is the same device that stores the program code of your microcontroller application. That implies a few things. First and most obvious is that you don't want to corrupt your application code by overwriting it with your data. Fortunately, the application code is always programmed to the front of the flash. So you should ensure that your data is, conversely, written at the end.

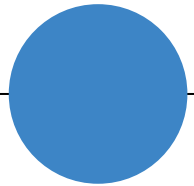
- PICO_FLASH_SIZE_BYTES # The total size of the RP2040 flash, in bytes
- FLASH_SECTOR_SIZE # The size of one sector, in bytes (the minimum amount you can erase)
- FLASH_PAGE_SIZE # The size of one page, in bytes (the mimimum amount you can write)

✓ To put some numbers to these, on the RP2040 chip, PICO_FLASH_SIZE_BYTES is 2MB or 2097152 bytes. FLASH_SECTOR_SIZE is 4K or 4096 bytes. FLASH_PAGE_SIZE is 256 bytes.

https://www.makermatrix.com/blog/read-and-write-data-with-the-pi-pico-onboard-flash/

Maze Solving

What's a maze and how do we get to goal



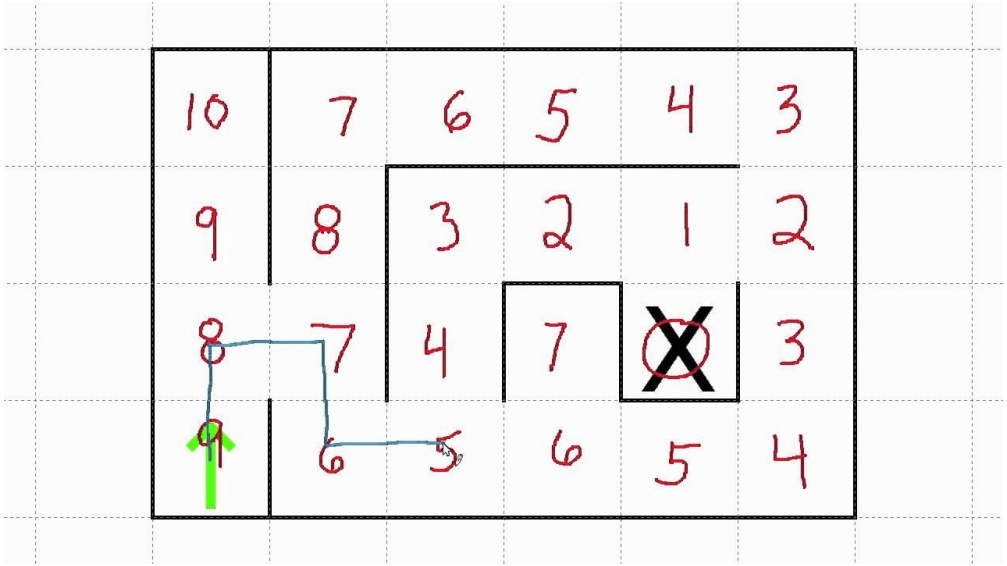
Overview

Maze Solving

- Maze Design
- Maze Solving Algorithms

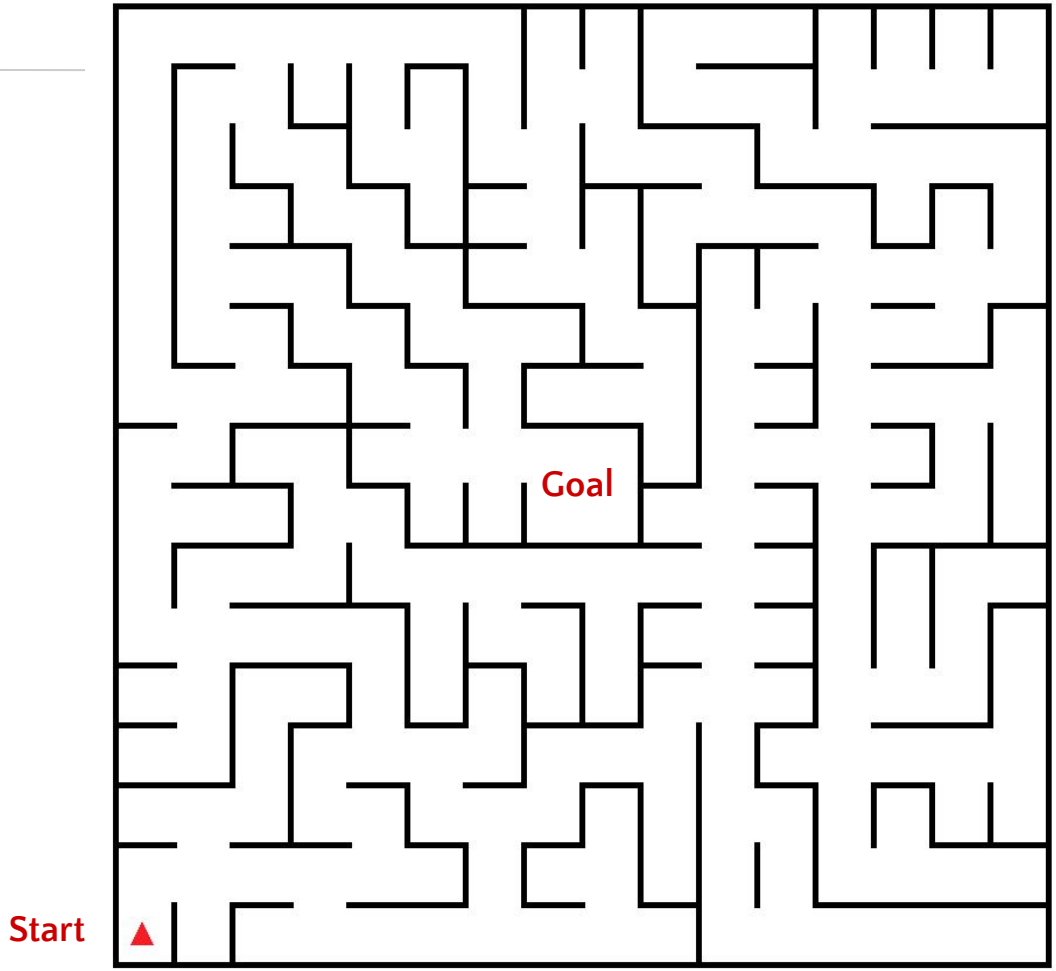
Floodfill

- High-Level Principles
- Demo
- Implementation



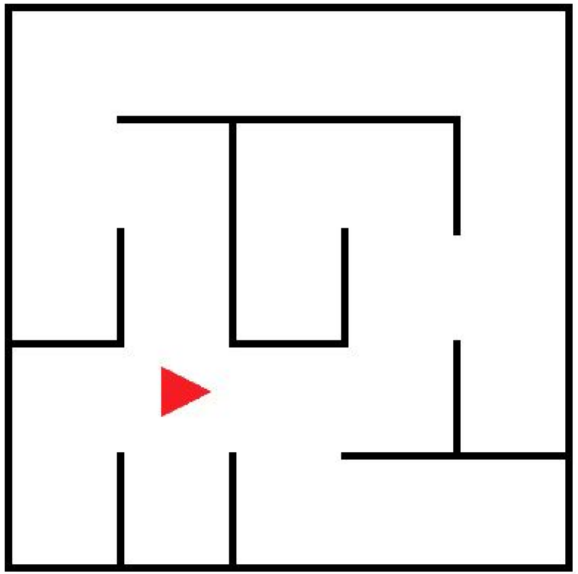
The Maze

- 16 cells x 16 cells, each cell is 18 cm x 18 cm
- Start is in the lower left corner
- Goal is the center: 4 cells total, entering any one of them is considered finishing
- We need to navigate the maze with **no prior knowledge** of the walls contained in it

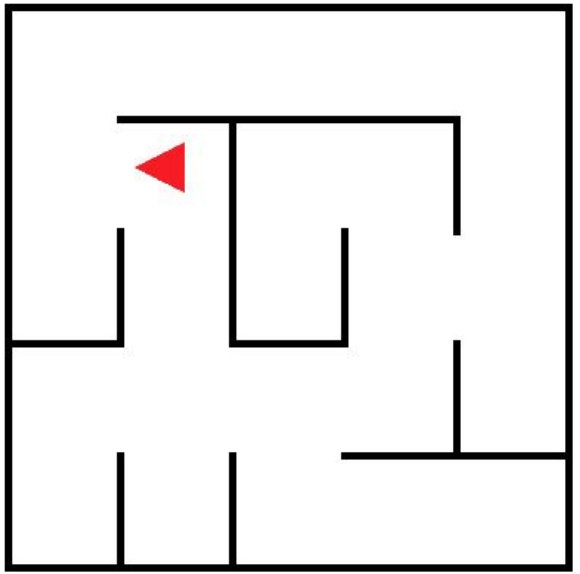


Basic Maze Solving Algorithms

Dead Reckoning



Wall Following



Maze Solving Approaches

Dead Reckoning: Only works for **paths** not **branched mazes**.

Wall Follower: Only works for **simply connected** mazes. Micromouse mazes are designed specifically so that this won't work.

Random: At each **intersection**, choose **randomly** which direction to go. Would take a very long time to find the goal.

Tremaux's Algorithm: Mark **paths** you've visited and prefer the unvisited ones. Does **not** guarantee the **shortest path**.

Floodfill: Guarantees the **shortest path** (after several runs).

1 Floodfill

Floodfill - High Level Idea

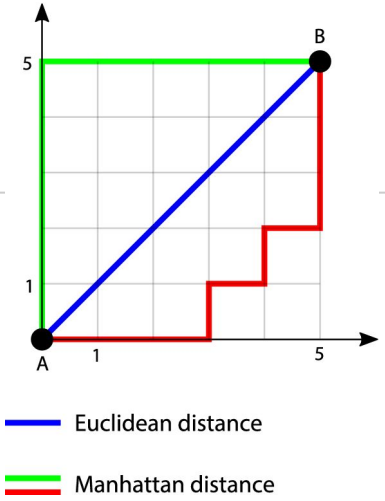
Goal: Find a path between start and end point

Working principle: Imagine pouring out a bucket of water at the goal

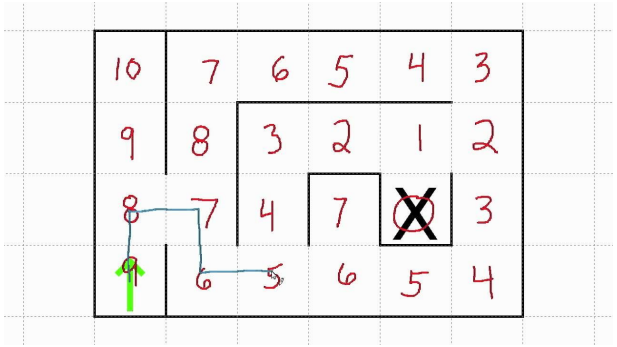
The water will flow around walls and eventually reach the starting point.

Intuitively, the path the water takes is the shortest valid path between the two points

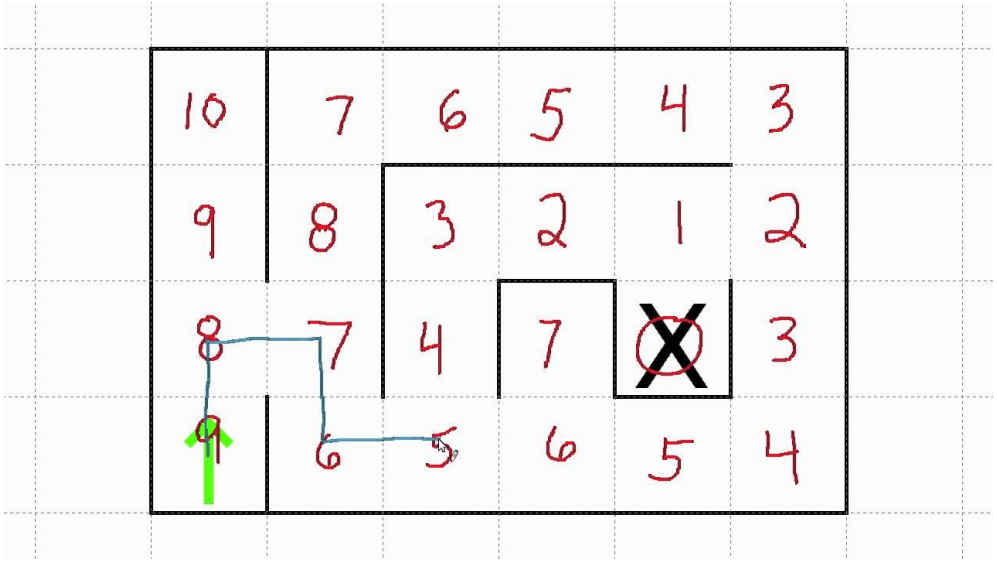
Manhattan Distance



- Your mouse can't travel through walls
- Number of cells from a given cell to the goal
- Also, the number of moves to get to the goal from a given cell
- Move towards cells that have smaller Manhattan distances from the goal



Manhattan Distance



Floodfill for Mice

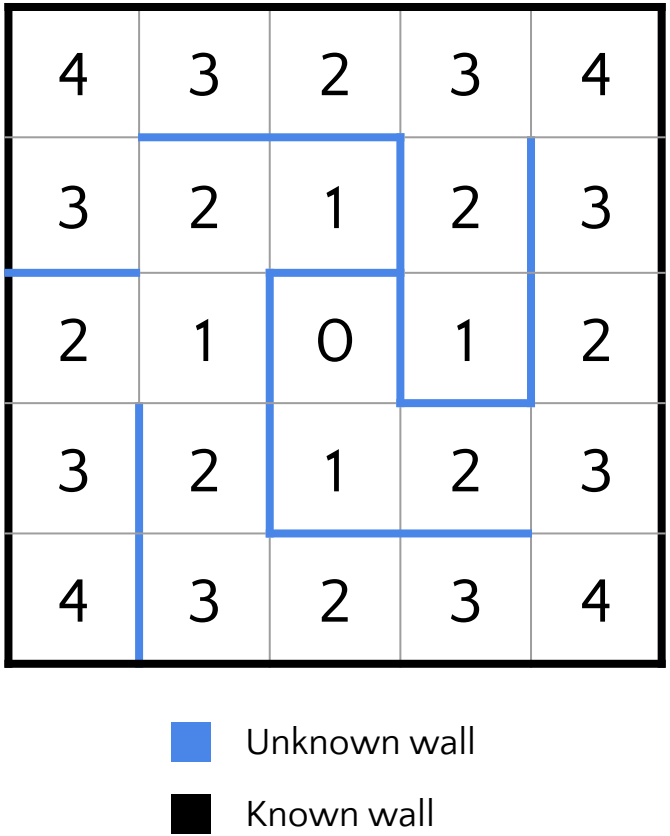
1. Assume maze has no walls to start. Use floodfill to calculate initial Manhattan distances

2. Travel towards **decreasing numbers**.

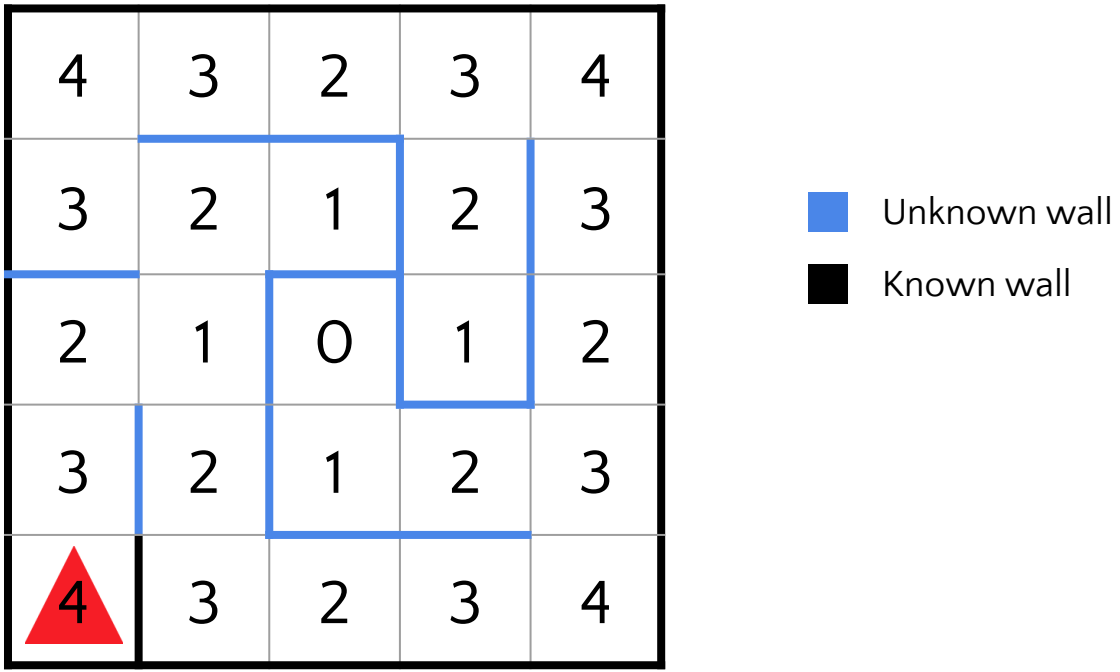
3. If you get stuck (surrounded by walls and larger numbers) then use Floodfill to **recalculate** true Manhattan distances.

As we learn about the maze, we **re-simulate** the fluid pour from the goal to see what the true Manhattan distances should be.

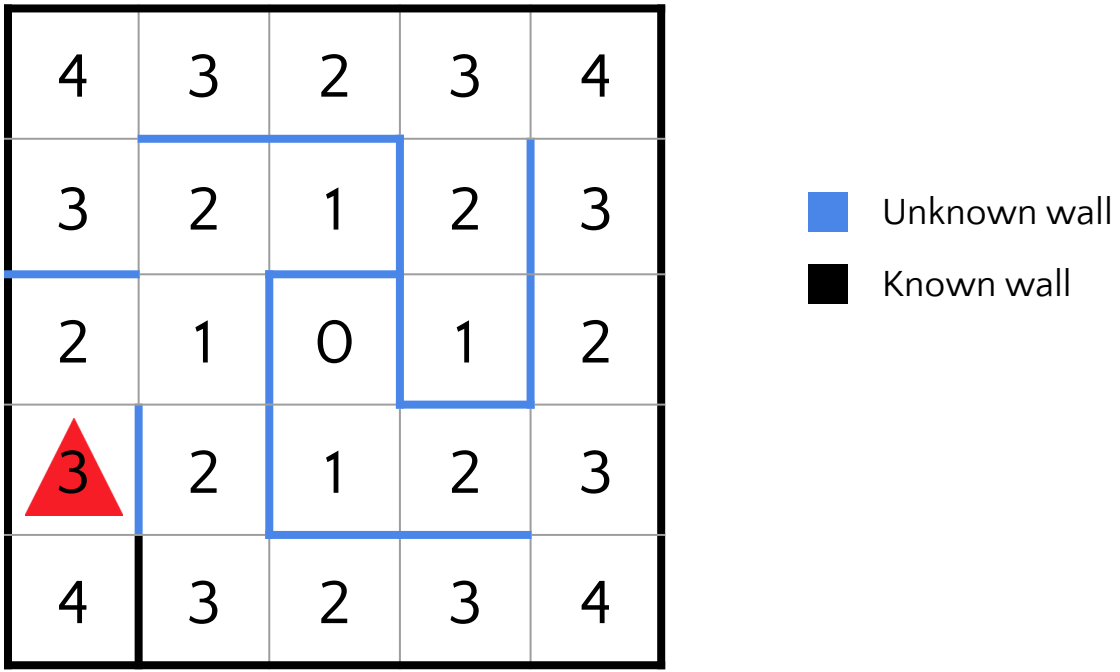
Maze Starting State



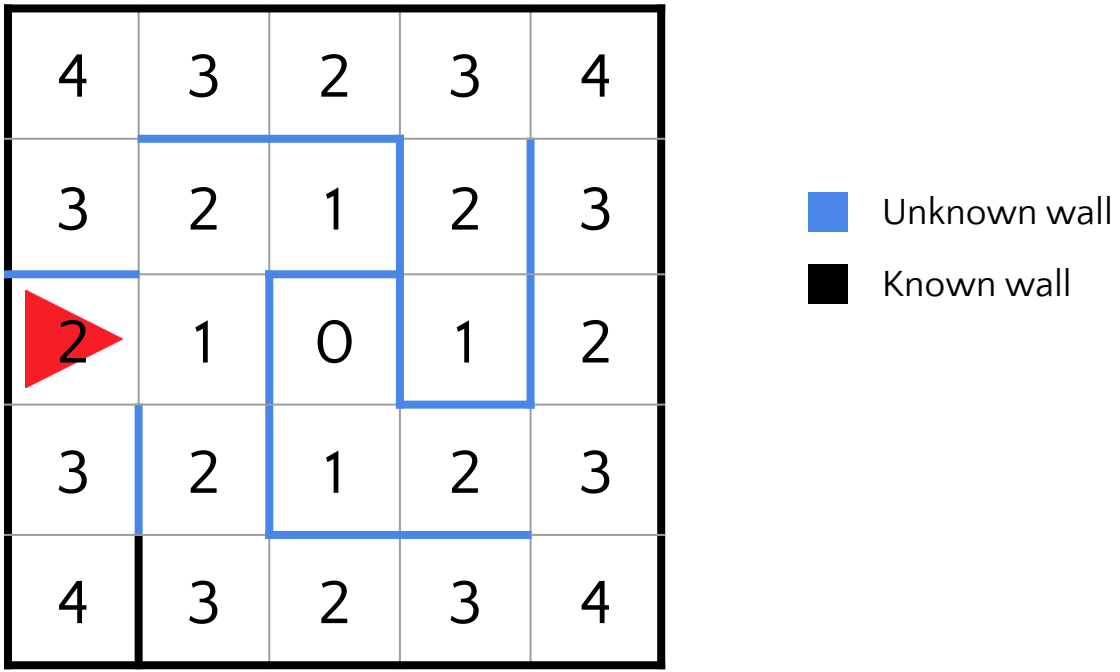
Traversing the Maze



Traversing the Maze

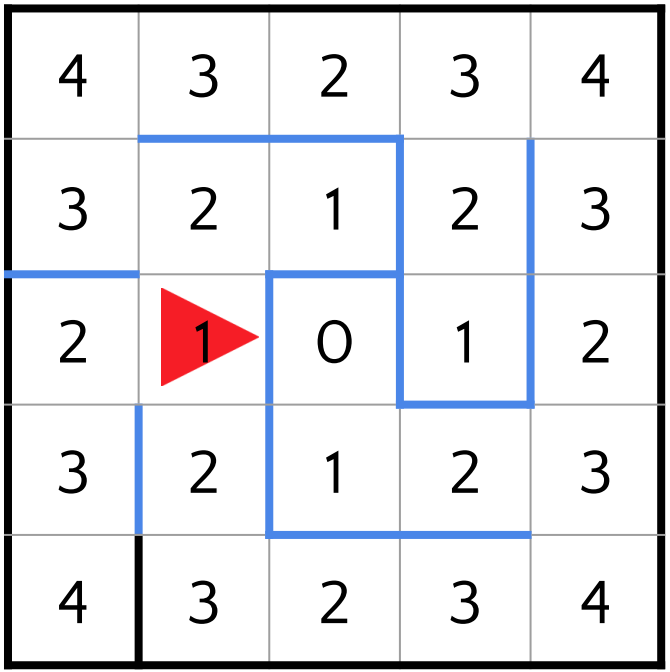


Traversing the Maze



Traversing the Maze

Mouse is Stuck!



Unknown wall

Known wall

Recalculating Manhattan Distance

We need to change the values to reflect how a hypothetical fluid would flow from the goal given the new walls we learned about

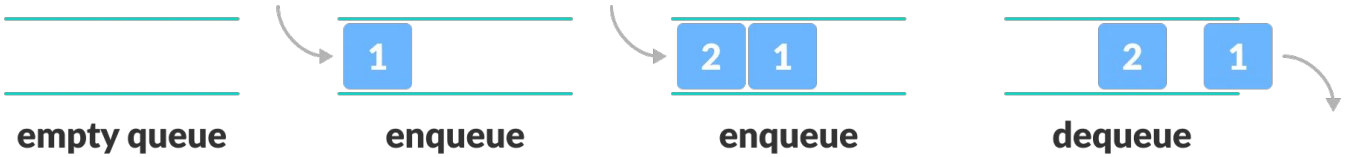
4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

Perform Floodfill

4	3	2	3	4
3	2	1	2	3
4	3	0	1	2
5	2	1	2	3
6	3	2	3	4

Floodfill
Pseudocode

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!

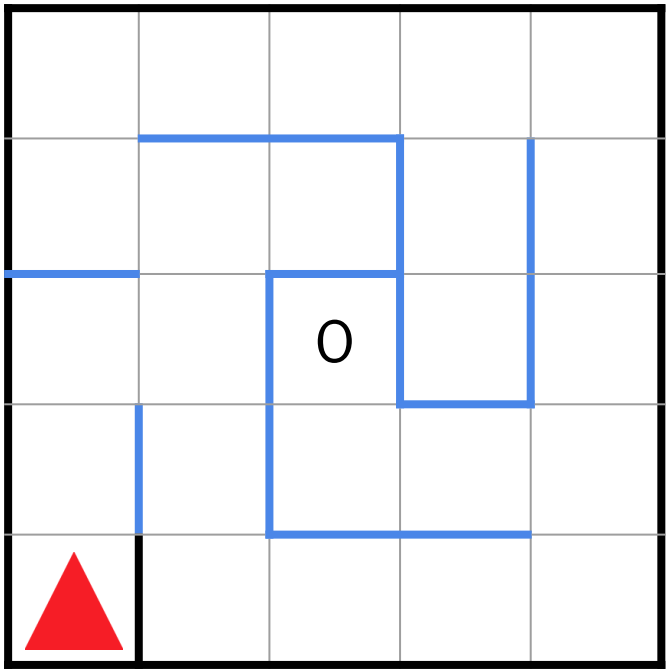


2

Floodfill Demo

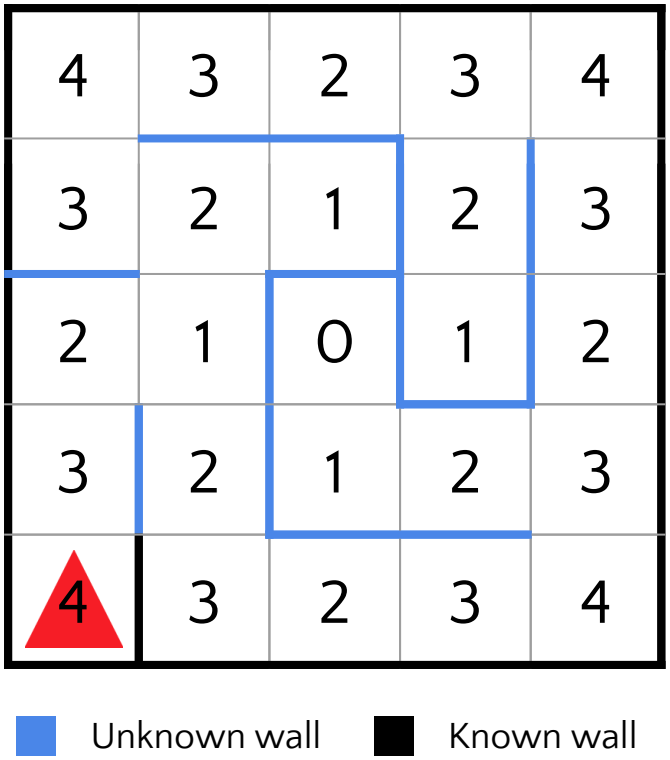
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



Floodfill Demo

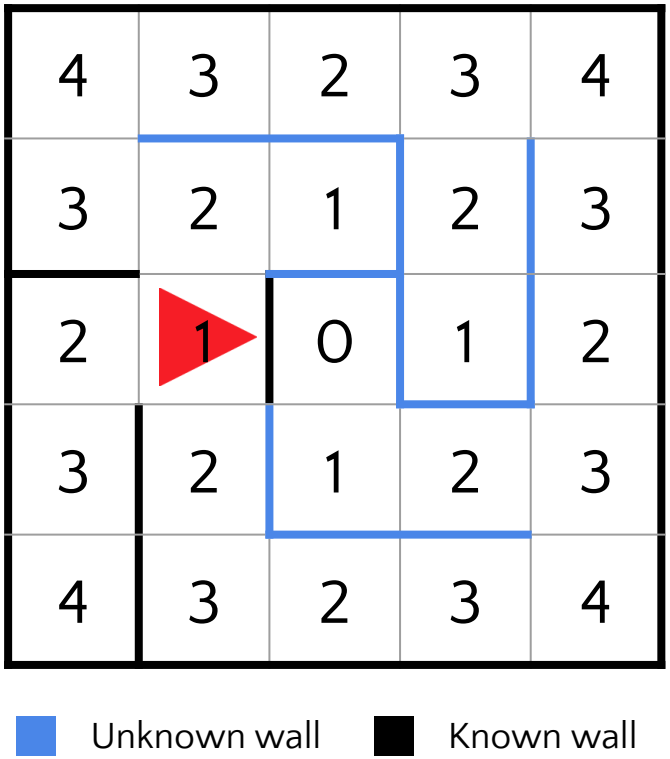
1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



FRONT													BACK

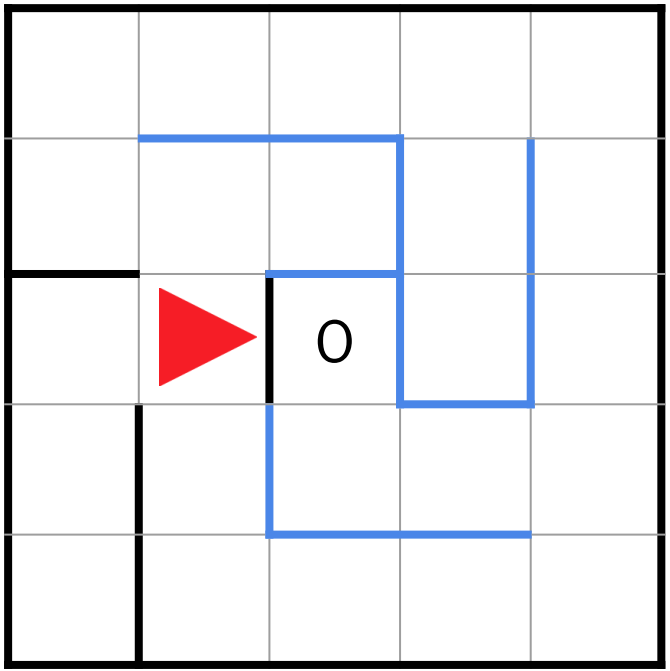
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



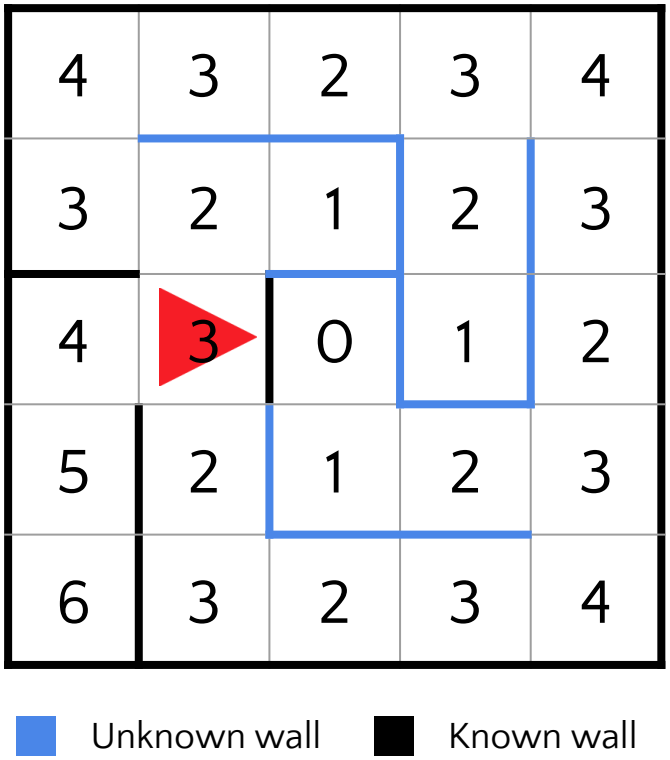
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



Floodfill Demo

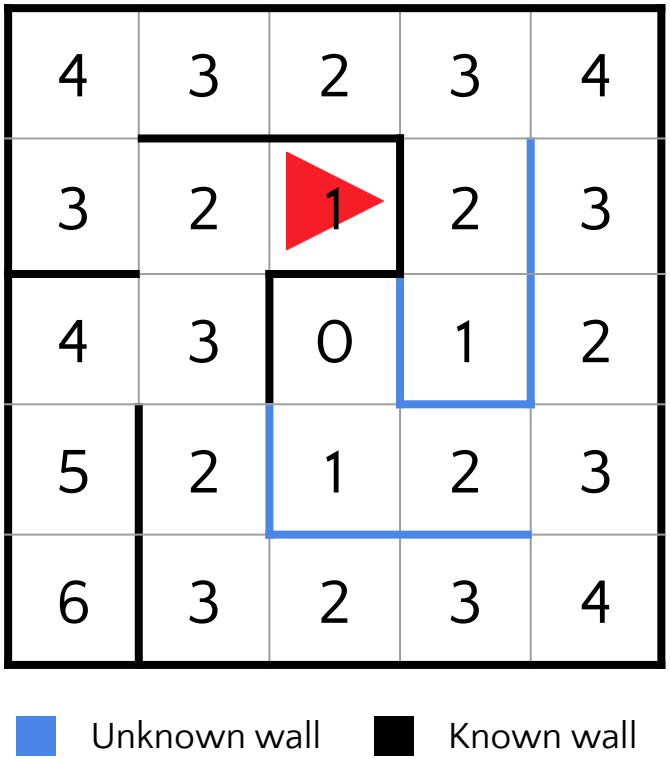
1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



FRONT													BACK

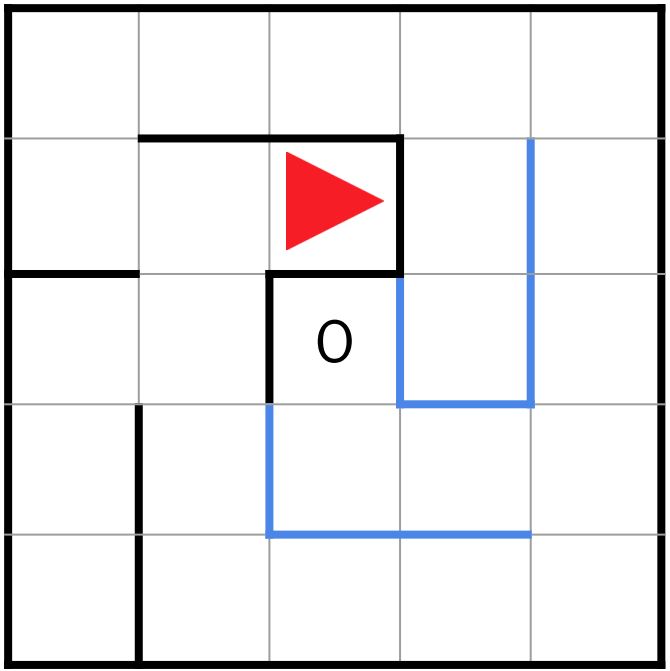
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



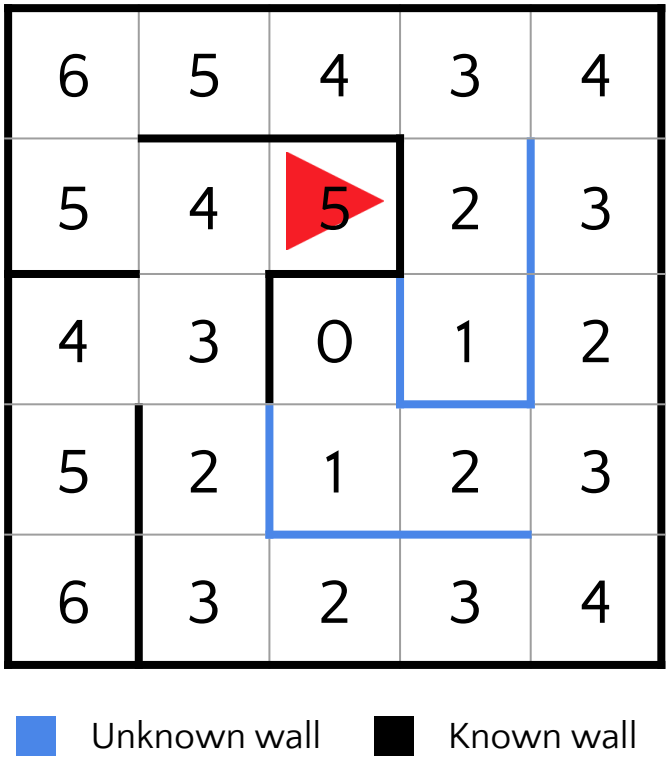
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



Floodfill Demo

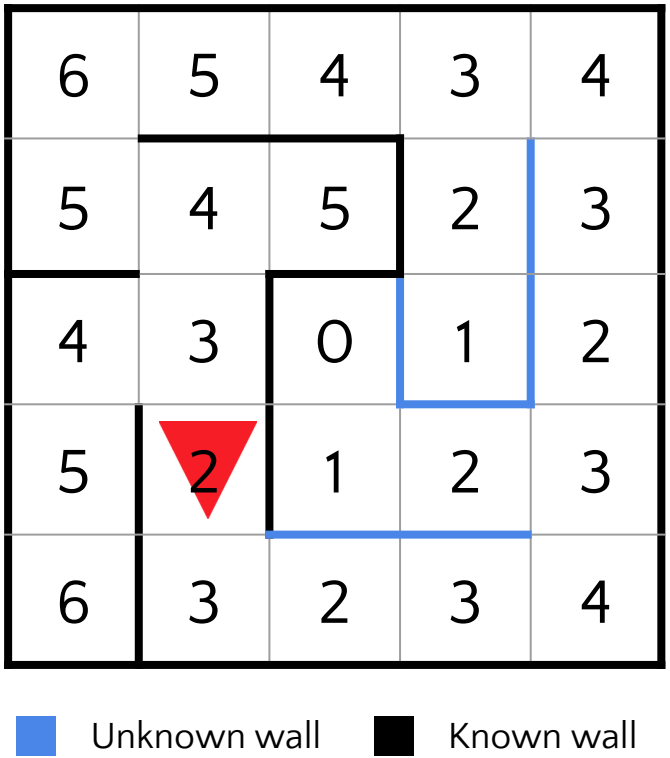
1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



FRONT													BACK

Floodfill Demo

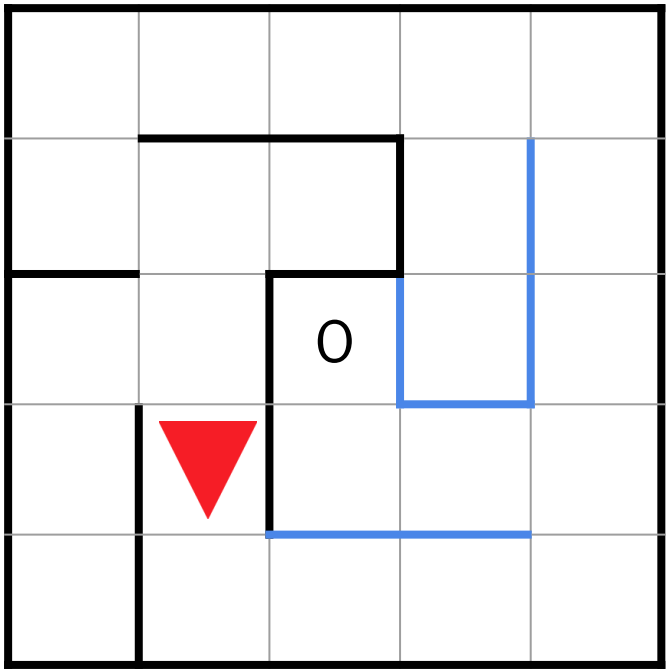
1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



FRONT													BACK

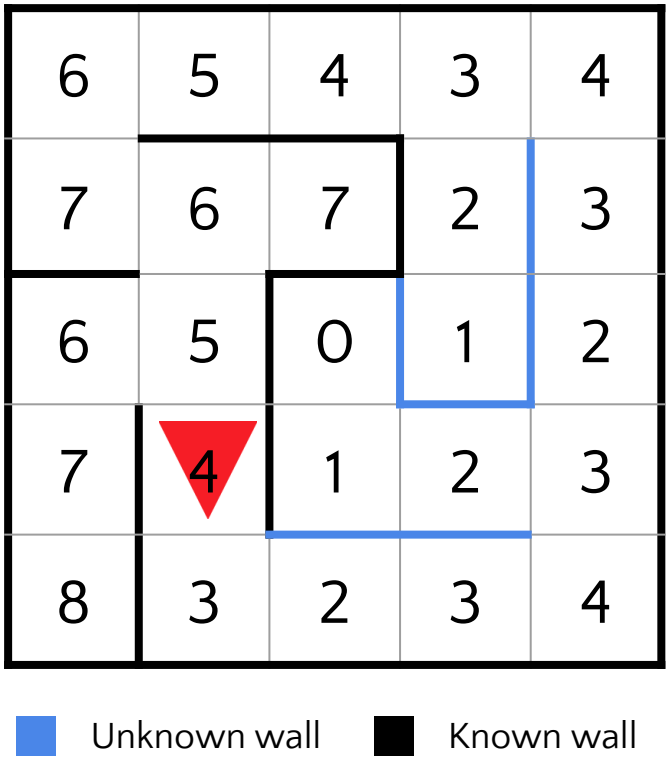
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



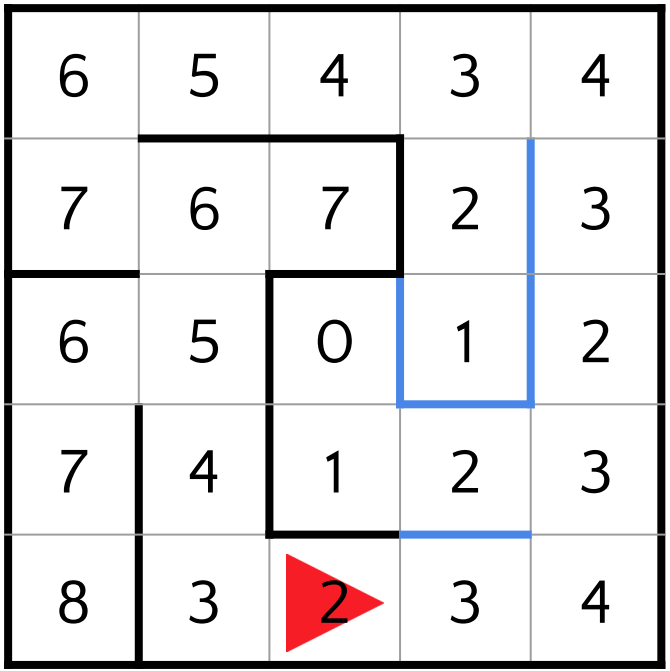
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



Floodfill Demo

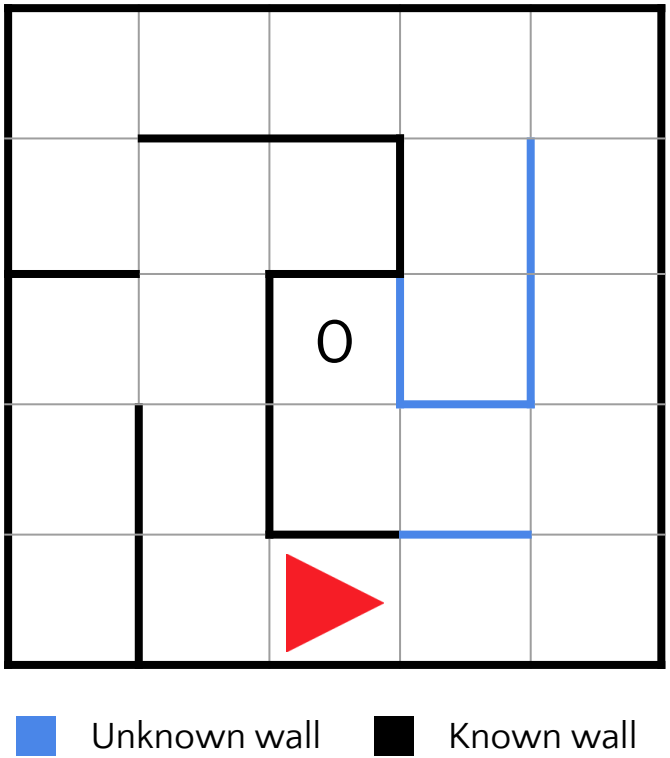
1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



FRONT													BACK

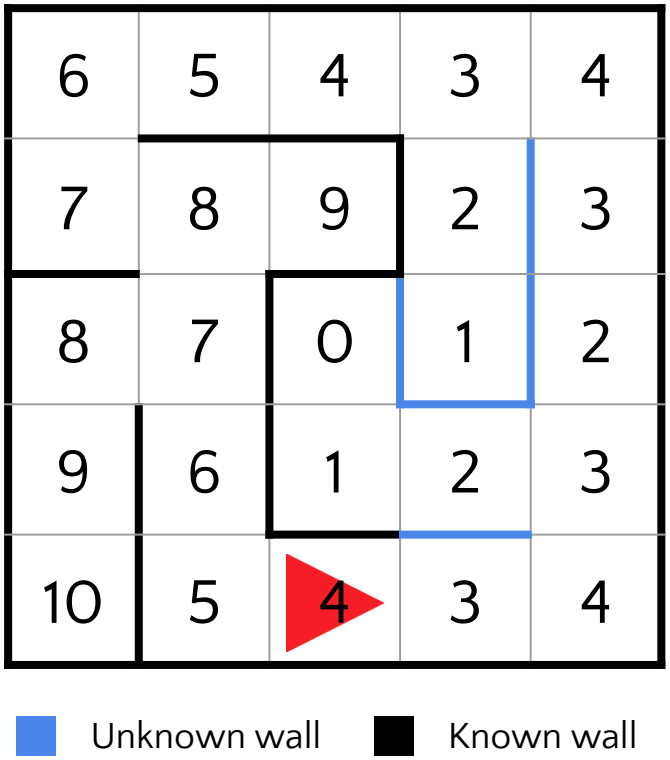
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



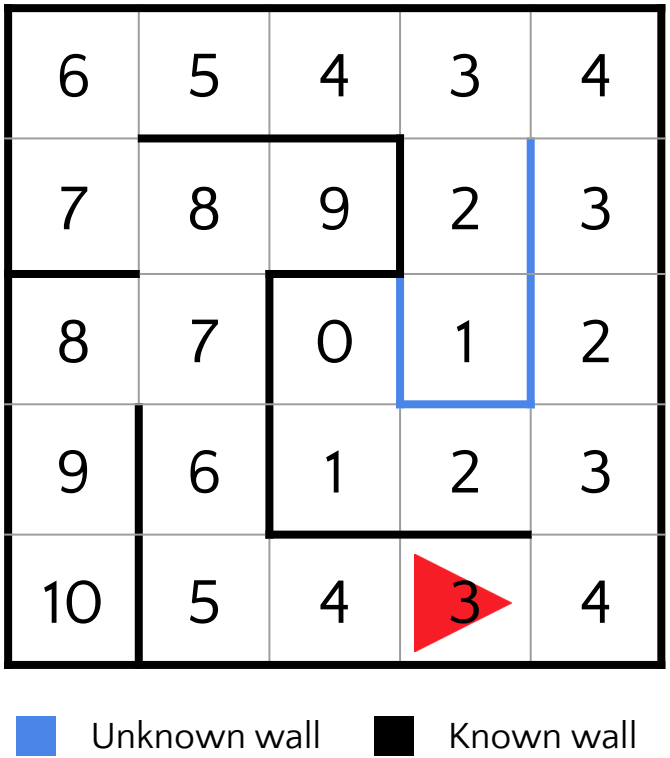
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



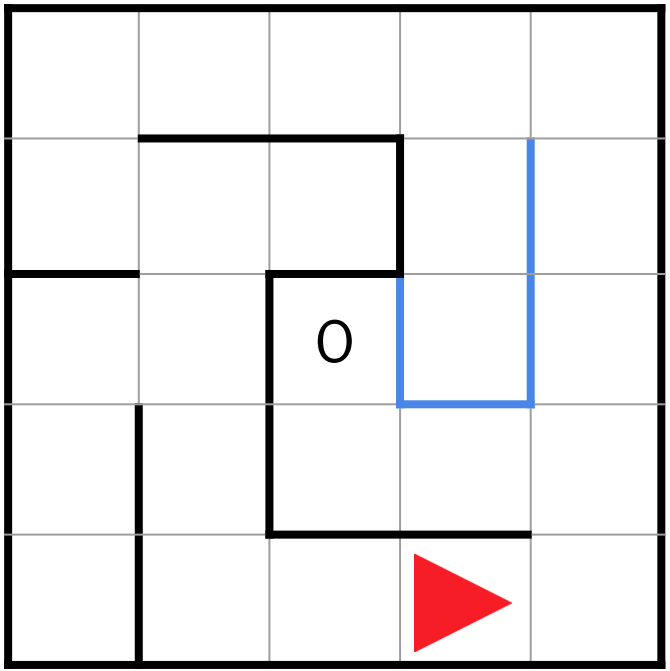
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



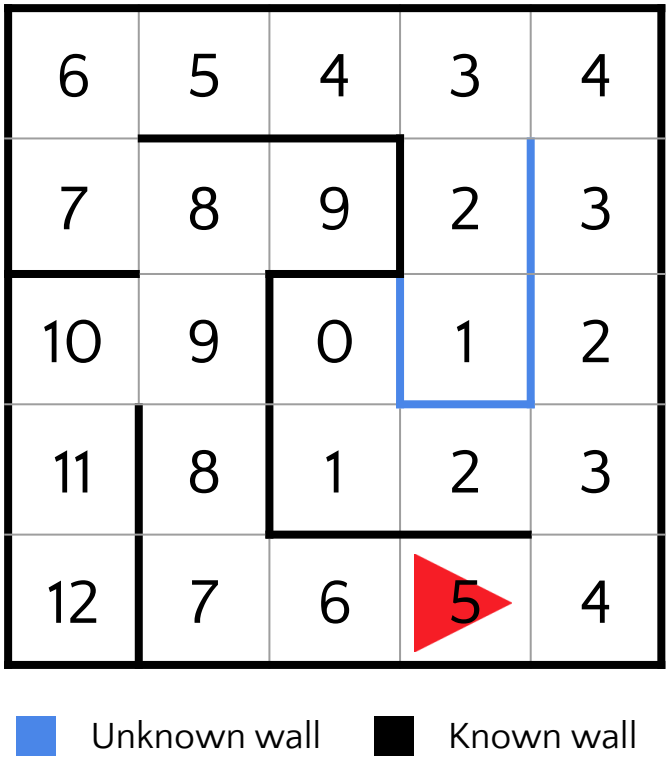
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



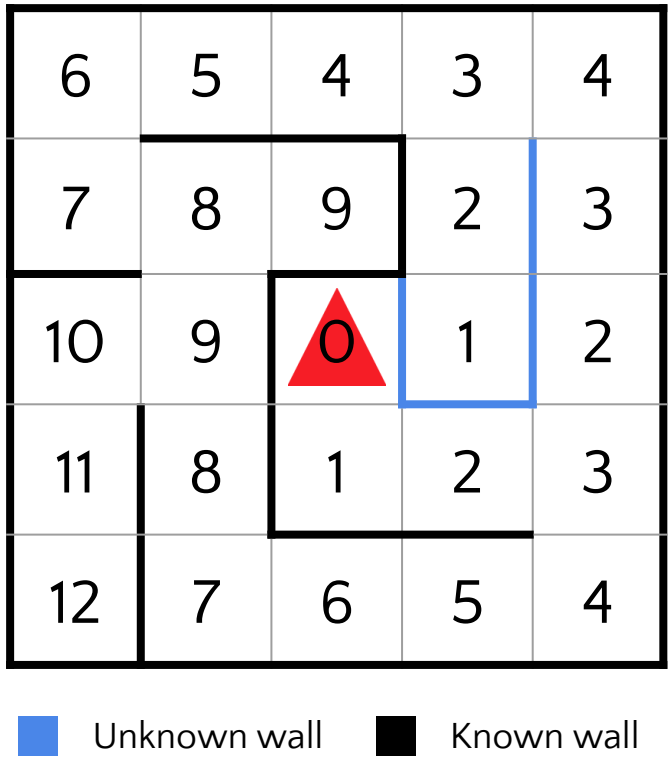
Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



Floodfill Demo

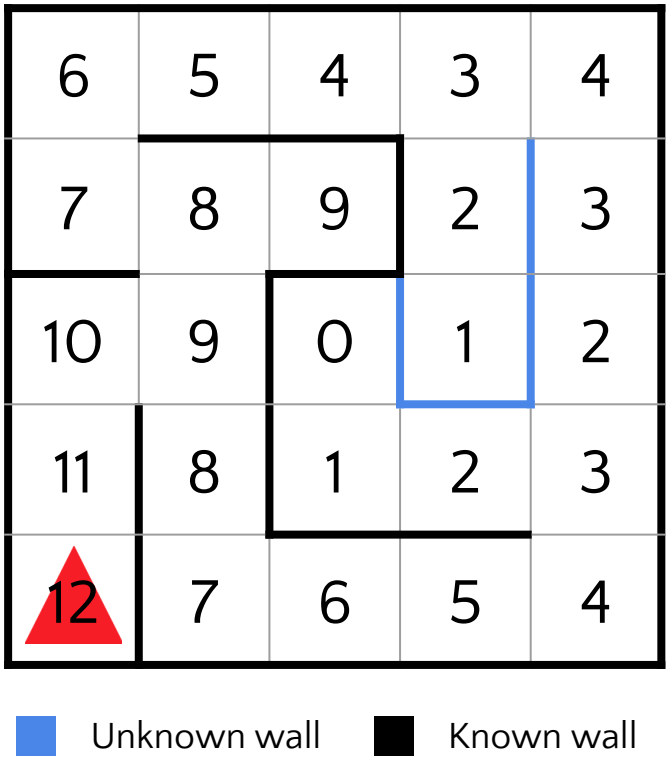
1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



FRONT													BACK

Floodfill Demo

1. Set all cells except goal to “blank state”
2. Set goal cell(s) value to 0 and add to queue
3. While queue is not empty:
   a. Take front cell in queue “out of line” for consideration
   b. Set all **blank** and **accessible** neighbors to front cell’s value + 1
   c. Add cells we just processed to the queue
   d. Else, continue!



4

Implementation Tips

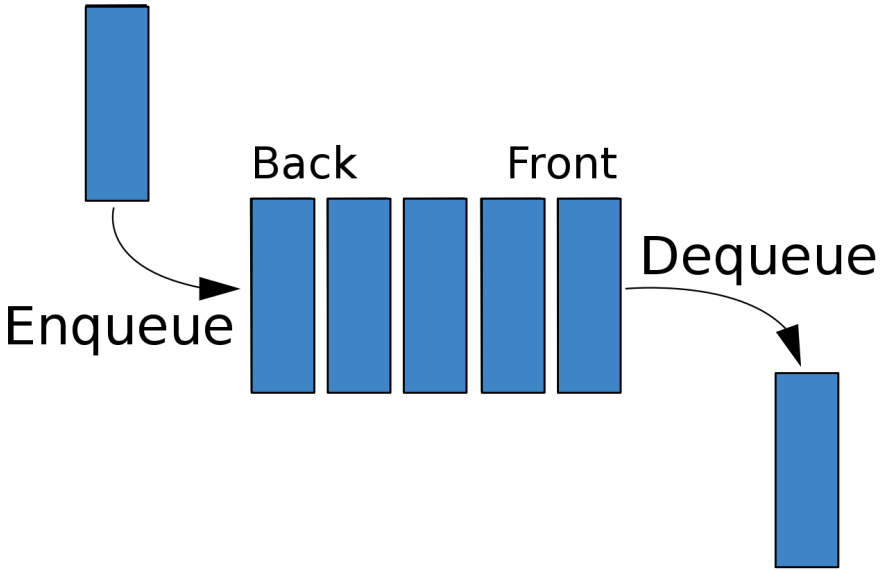
Sneak Peek into Life of an Embedded C Programmer

Data Structures: Queue

FIFO (First In, First Out) Data structure

- Add an item to the back (“Enqueue” or “Push”)
- Take an item from the front (“Dequeue” or “Pop”)

We will use a queue as a list of cells to examine



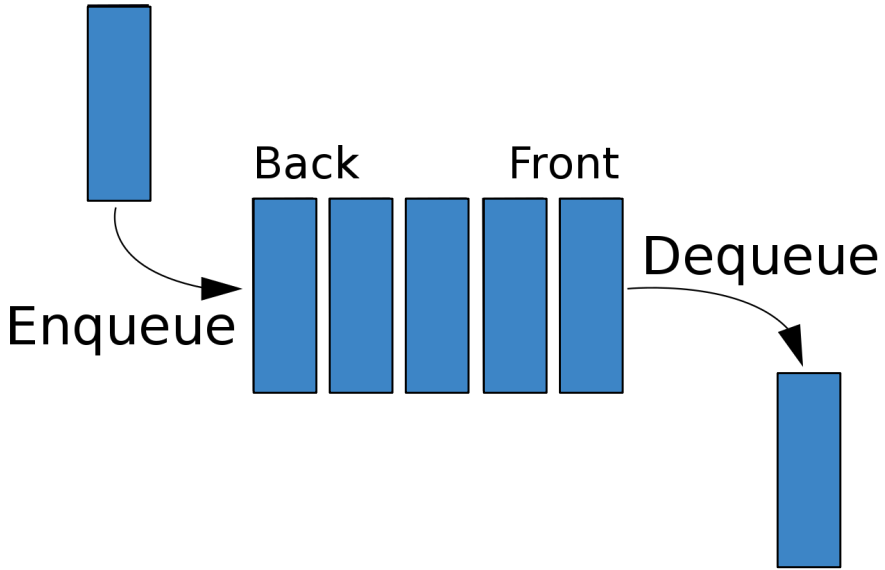
Implementing a Queue

A Queue is an abstract data type, various implementations are possible

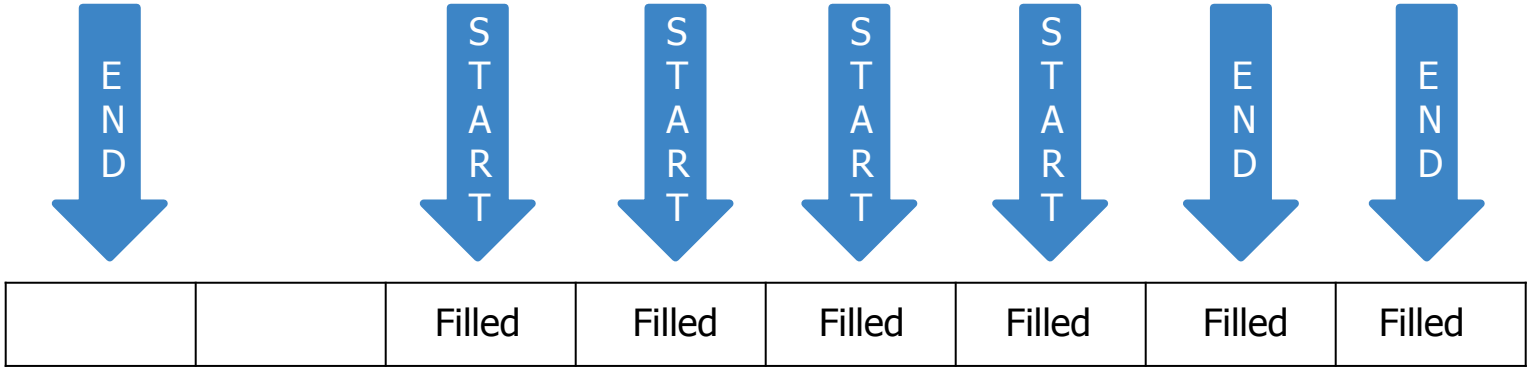
Two main ways to implement a Queue

- Circular Array
- Linked List

C does not have the standard library like C++, so cannot just import one



Queue with Circular Array



START is the index of the first item in the queue

END is the index of the array just after the last item in the queue

Other Tips

You need a Coordinate System to keep track of cells... use a **struct**!

Three 2D Arrays, to represent:

- Horizontal Walls
- Vertical Walls
- Manhattan Distances

Initializing Variables: Initialize variables assuming **no walls** in maze

Other Tips

You're probably familiar with programming in C++, so here's some differences between C and C++

- No default constructors
- No STL Library
- No Boolean variable type
- Overall just not fun

Other Tips

e.g. C++ code:

```
class A {
  public:
    A() { a = 0; }
    int a;
};

int main()
{
  A b;
  A *c = new A;
  return 0;
}
```

equivalent C code:

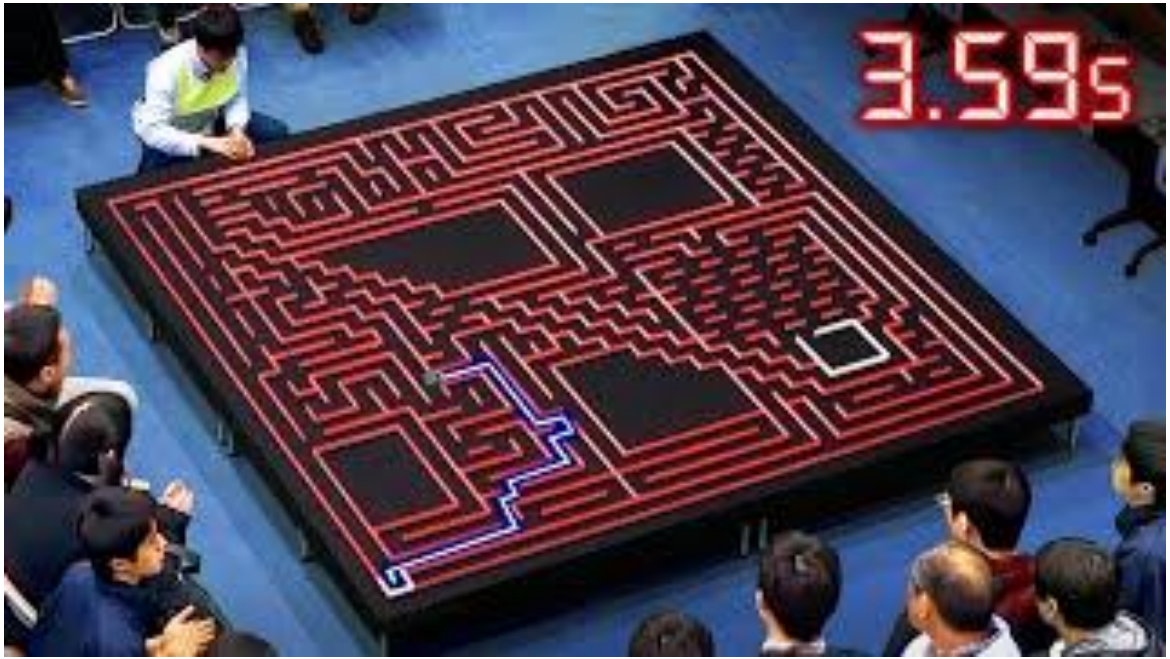
```
struct A {
  int a;
};

void init_A_types(struct A* t)
{
   t->a = 0;
}

int main()
{
   struct A b;
   struct A *c = malloc(sizeof(struct A));
   init_A_types(&b);
   init_A_types(c);
   return 0;
}
```

Reference:
https://stackoverflow.com/questions/537244/default-constructor-in-c

Video



https://www.youtube.com/watch?v=ZMQbHMgK2rw

Video

https://www.youtube.com/watch?v=-0z3v3Sw34A