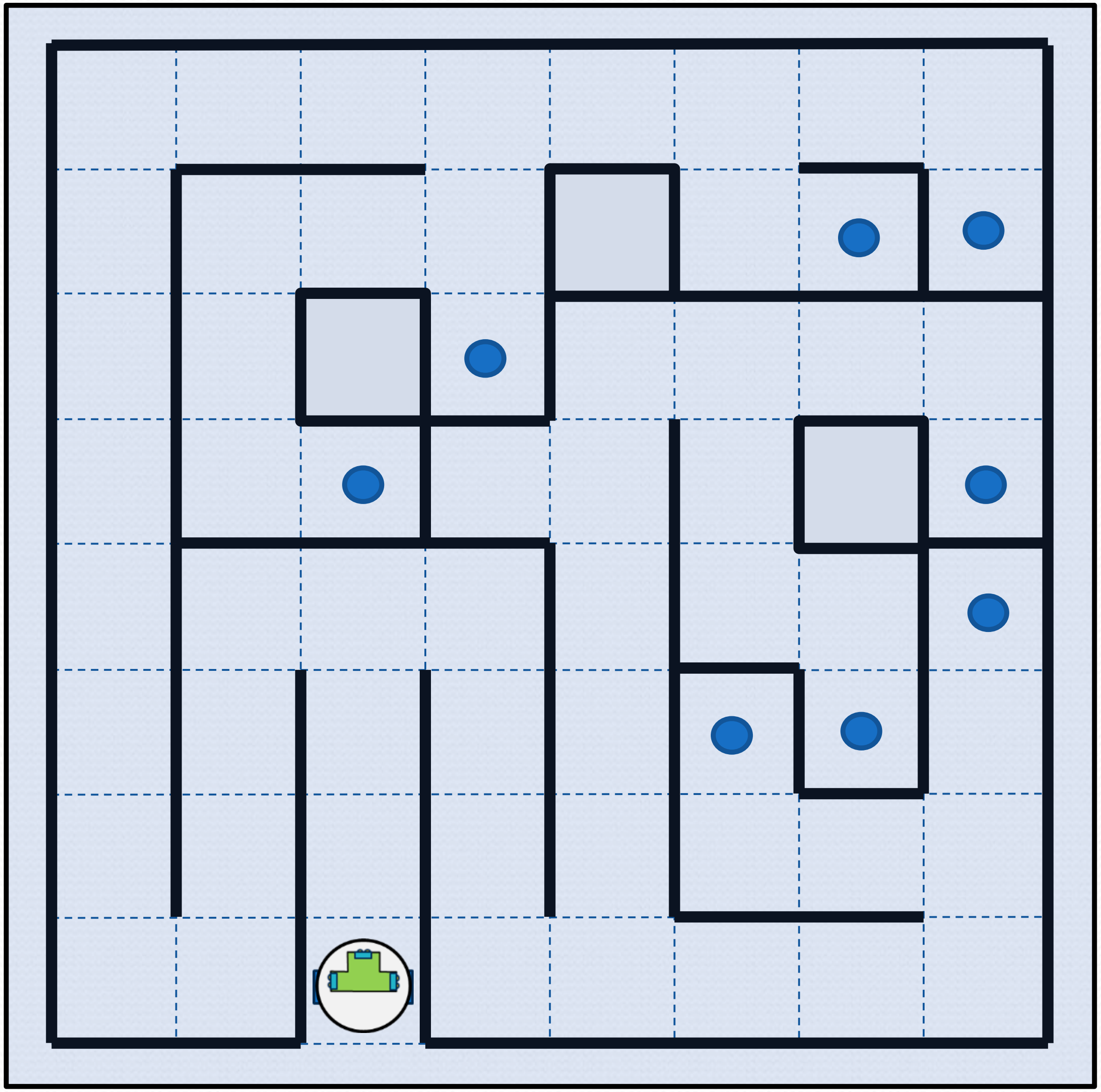
Exercise 23: Split Your Team (maze_solve/ MtrDrv)

- Reccommended Split
  - 2 people on ***maze_solve*** implementation & test
  - 2 people on ***MtrDrv*** implementation & test
- For this exercise you will split your team and either work on ***maze_solve*** & testing or on ***MtrDrv*** which is responsible for final motor drive compensating for battery voltage droop.
- Decide how you are splitting roles and jump to the appropriate section of this document. ***maze_solve*** is covered first. ***MtrDrv*** is slightly simpler than ***maze_solve***.

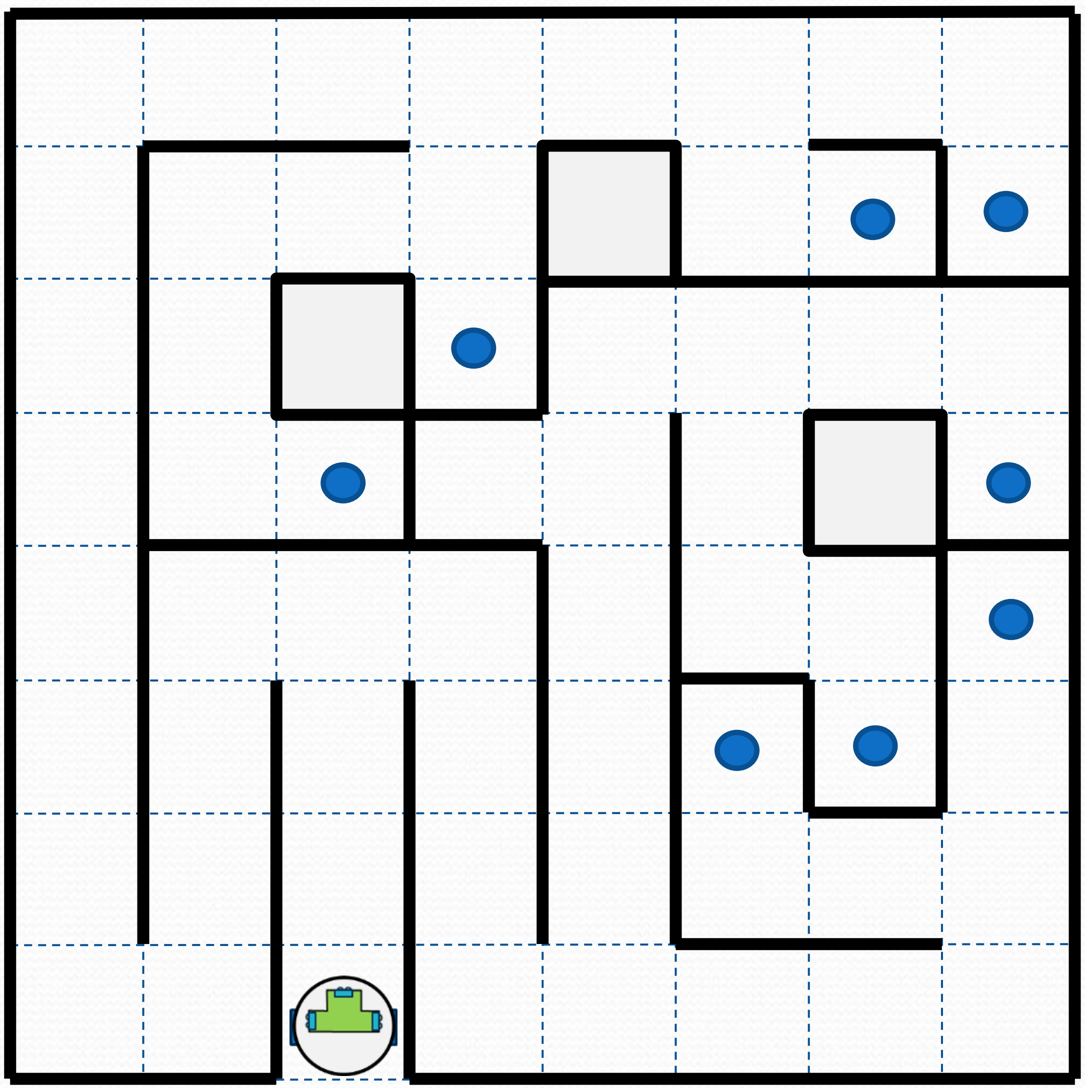
Exercise 23: maze_solve



A magnet could be placed on any of the  spaces. The mazeRunner's job is to traverse the maze till it finds it.

Think about the algorithm you would use.

Exercise 23: maze_solve



It turns out a maze traversing algorithm is not as complex as you might think.

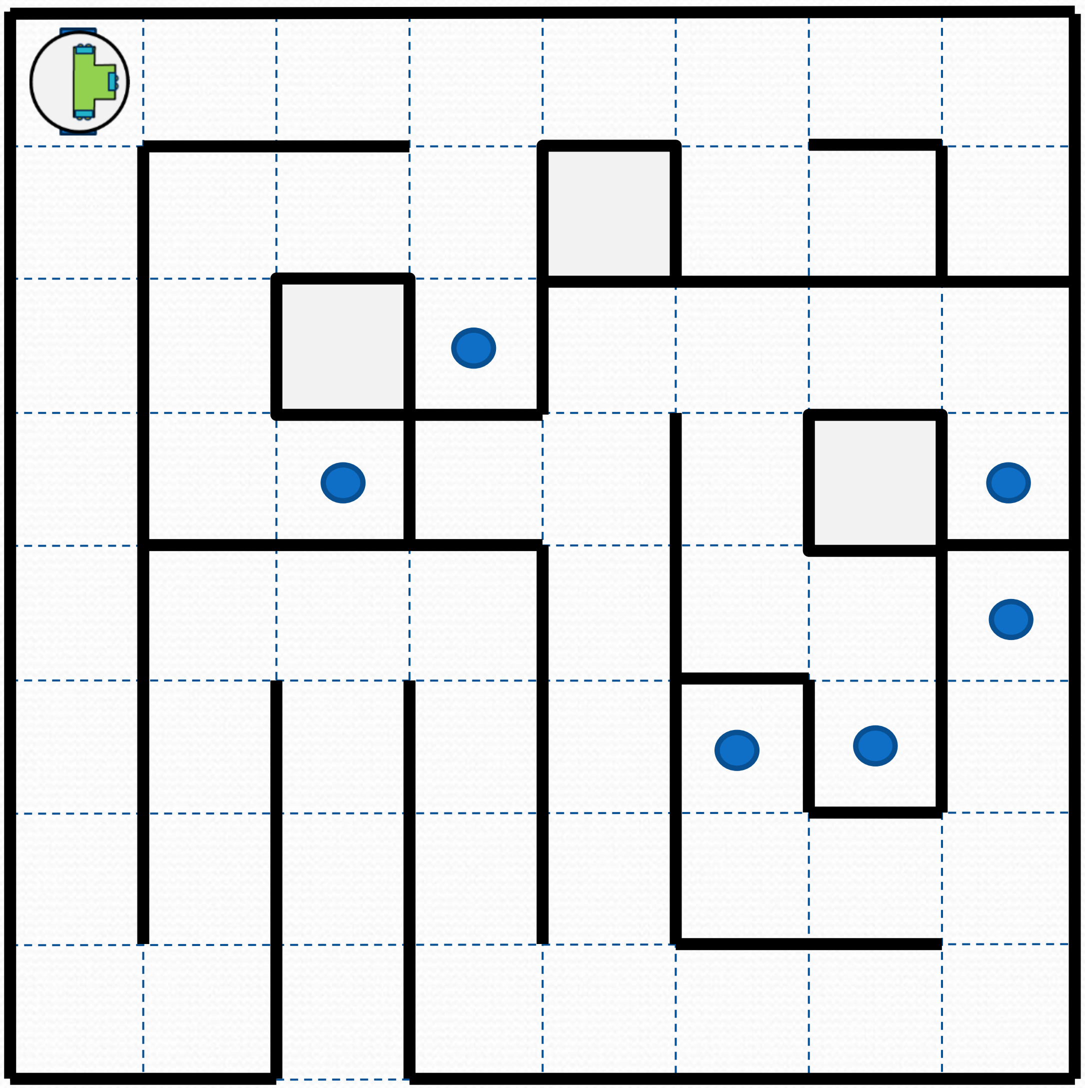
It can be solved with a left affinity or right affinity algorithm.

Go straight till blocked or preferred opening

Take preferred opening if possible, otherwise take available direction

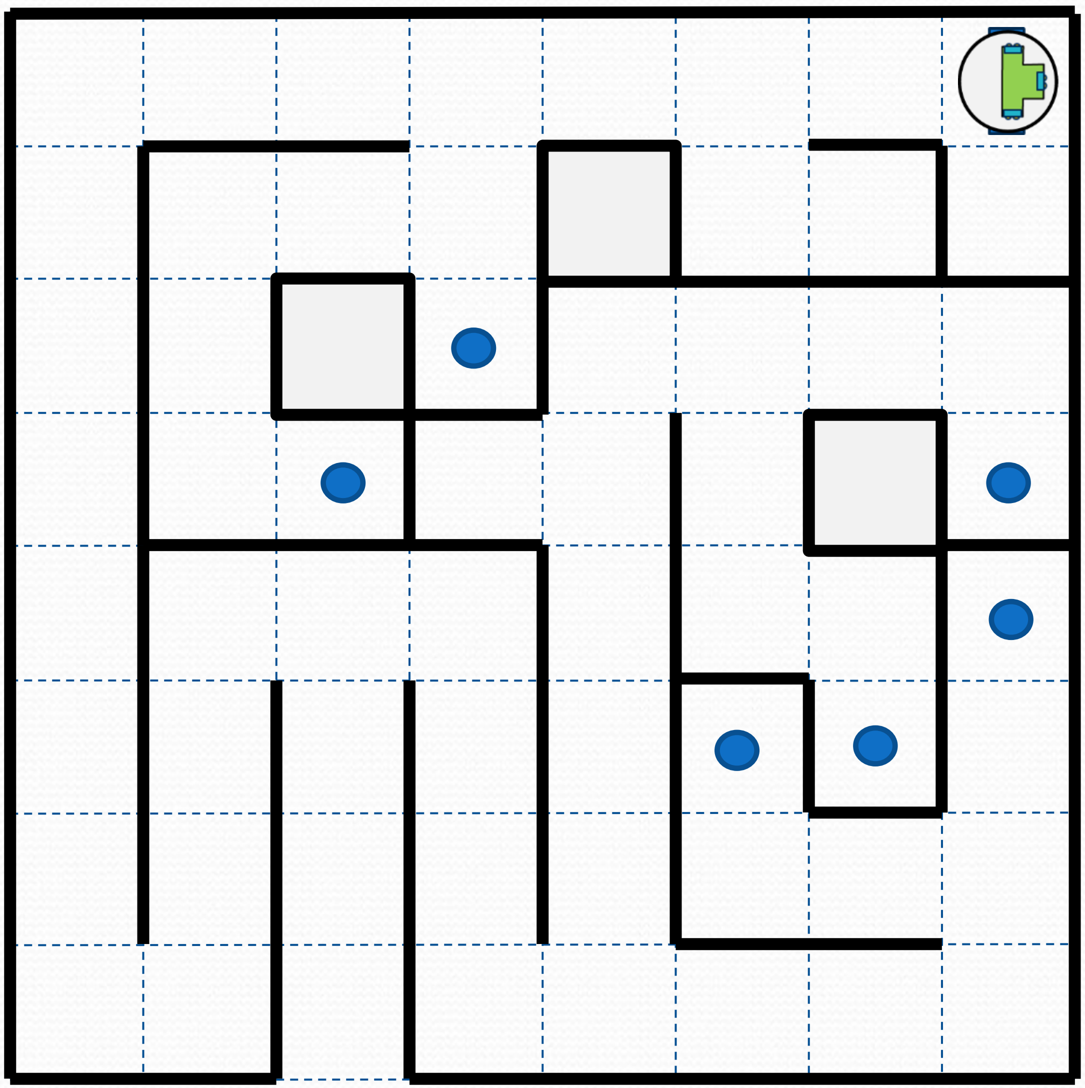
If no available direction spin a 180

Exercise 23: maze_solve



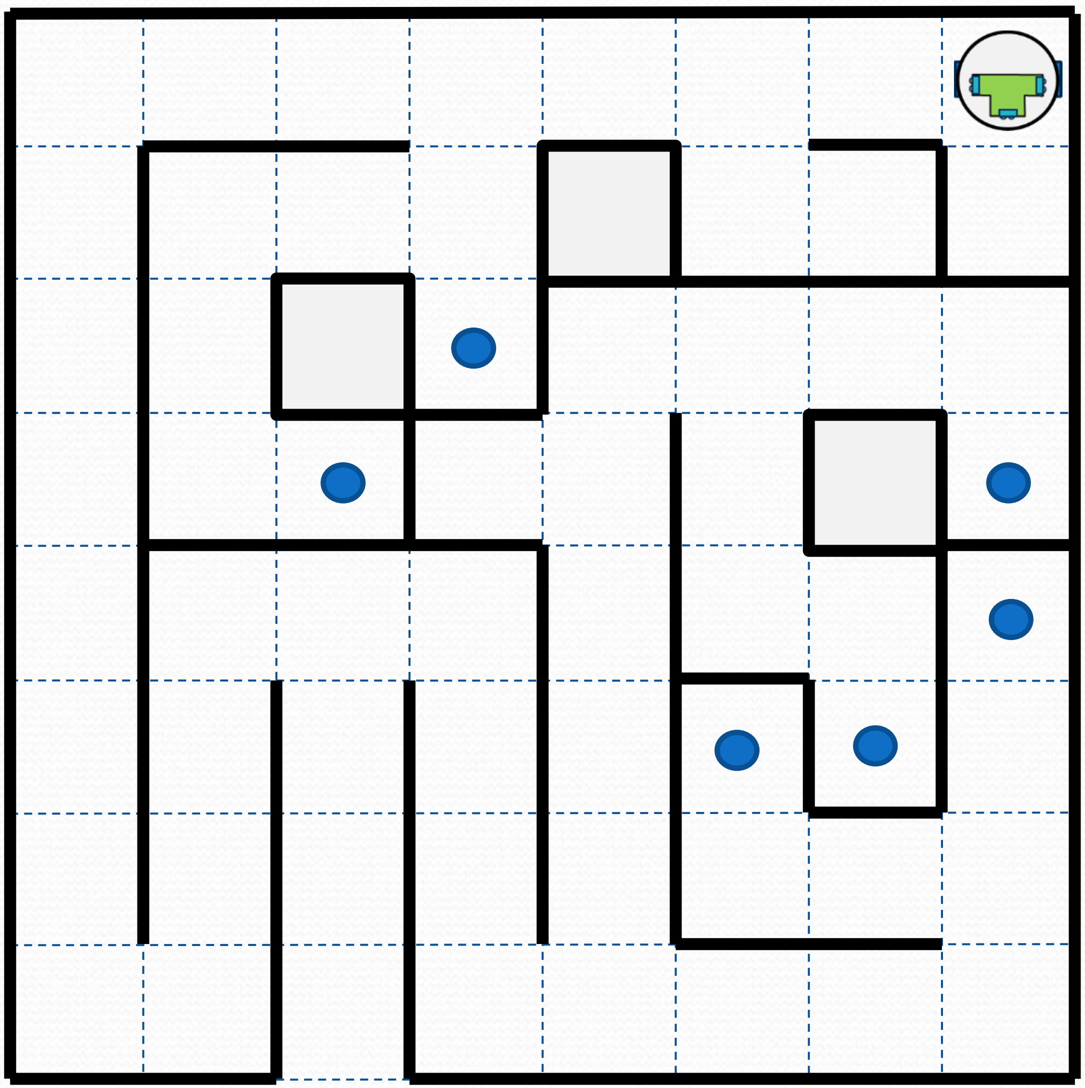
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



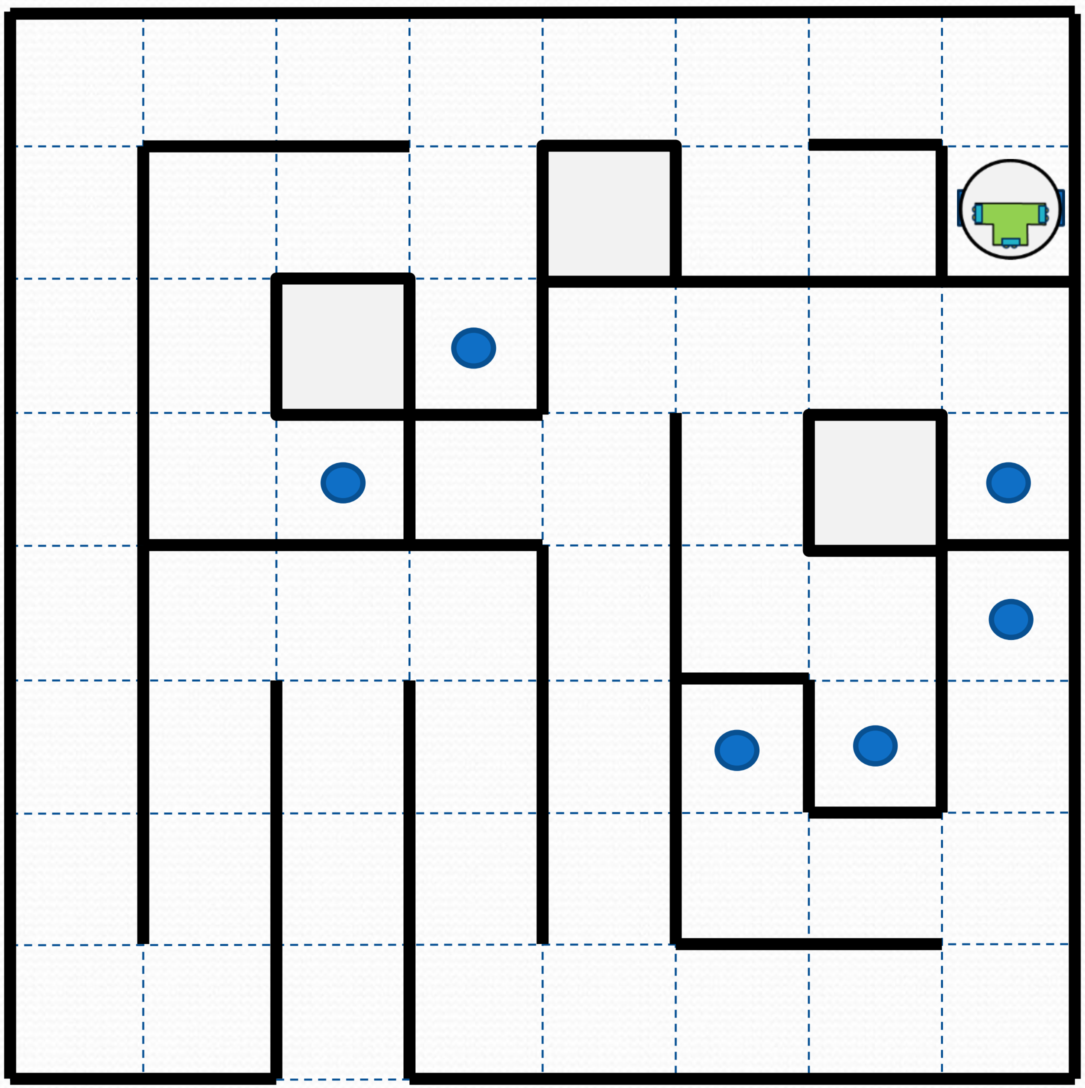
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



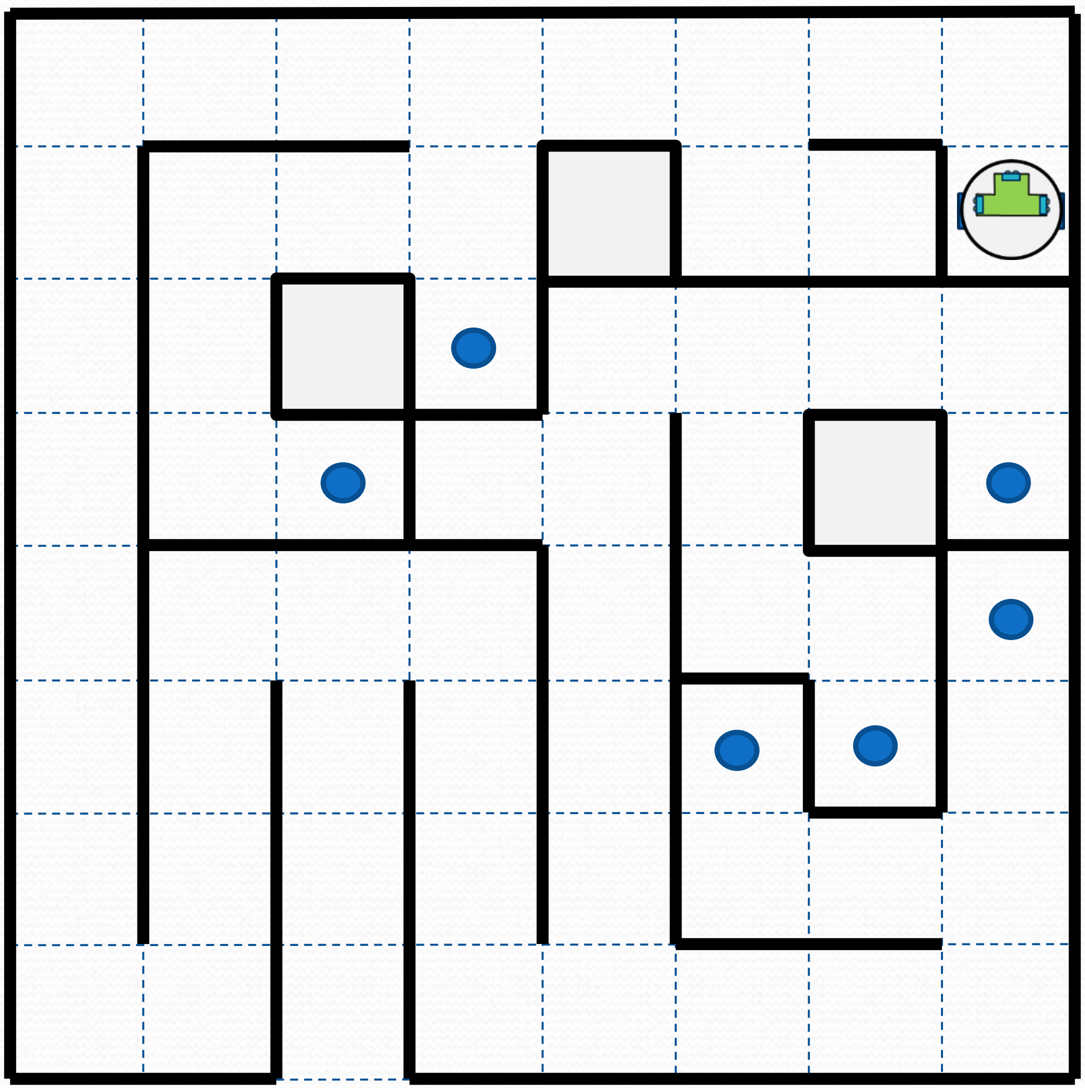
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



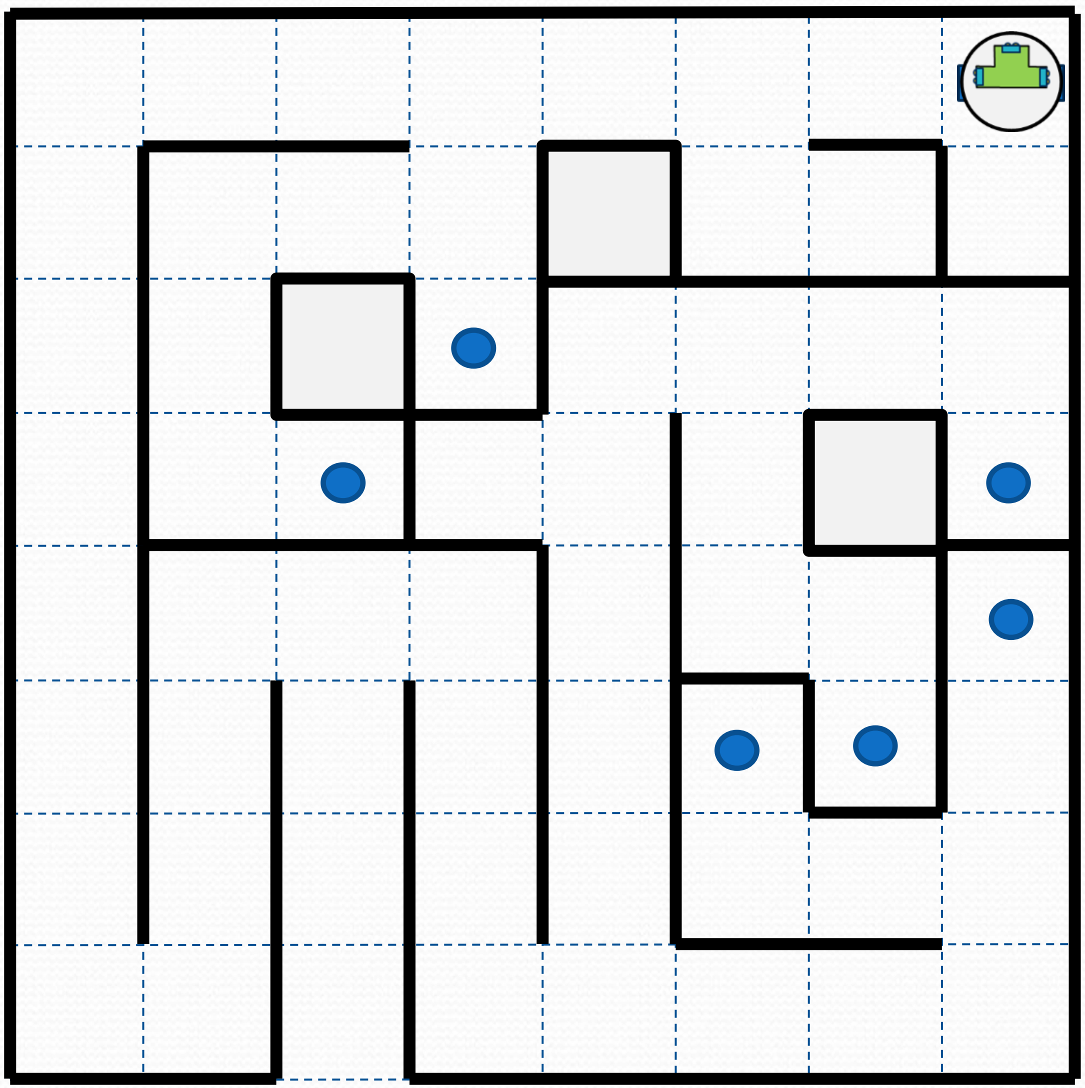
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



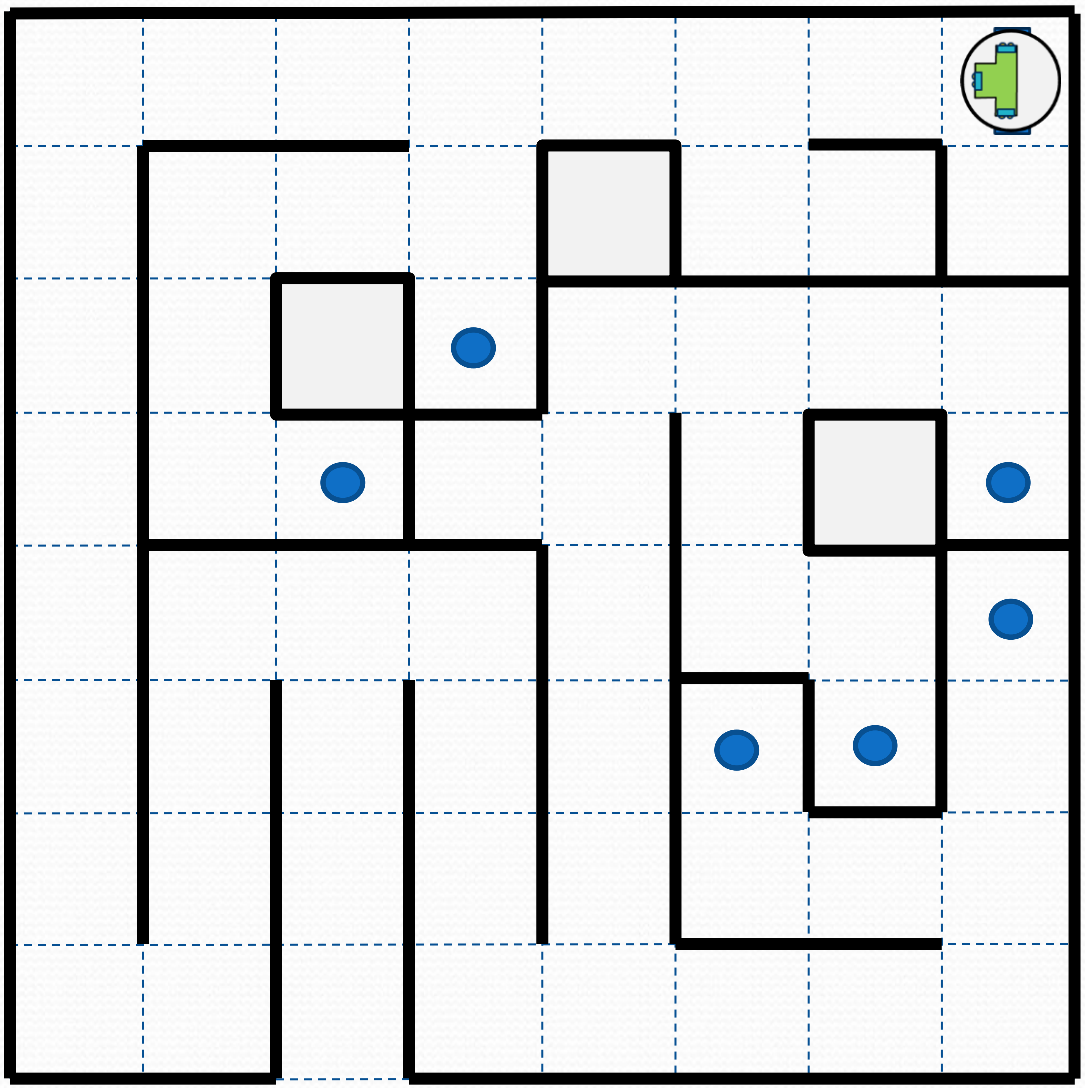
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner)*.

Exercise 23: maze_solve



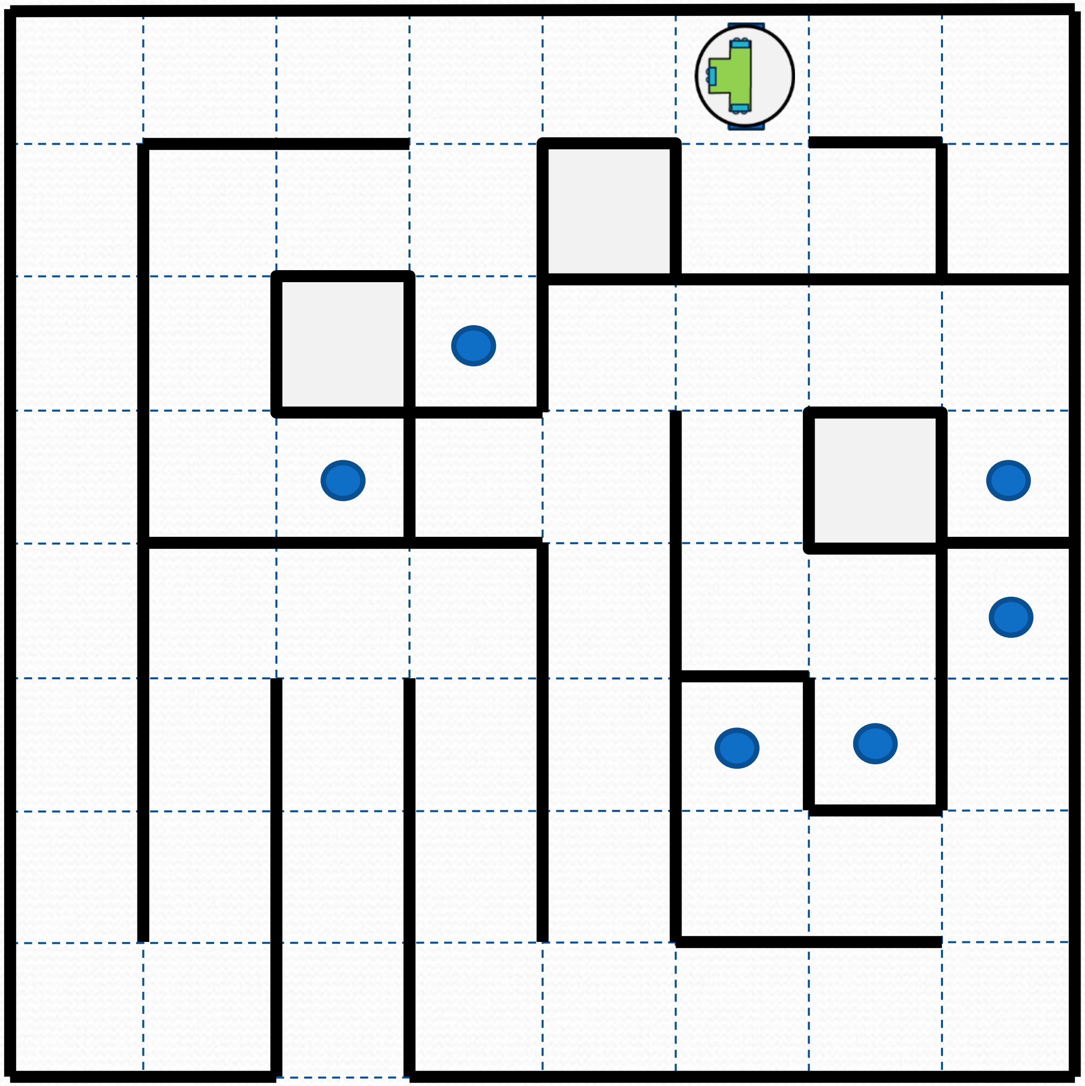
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



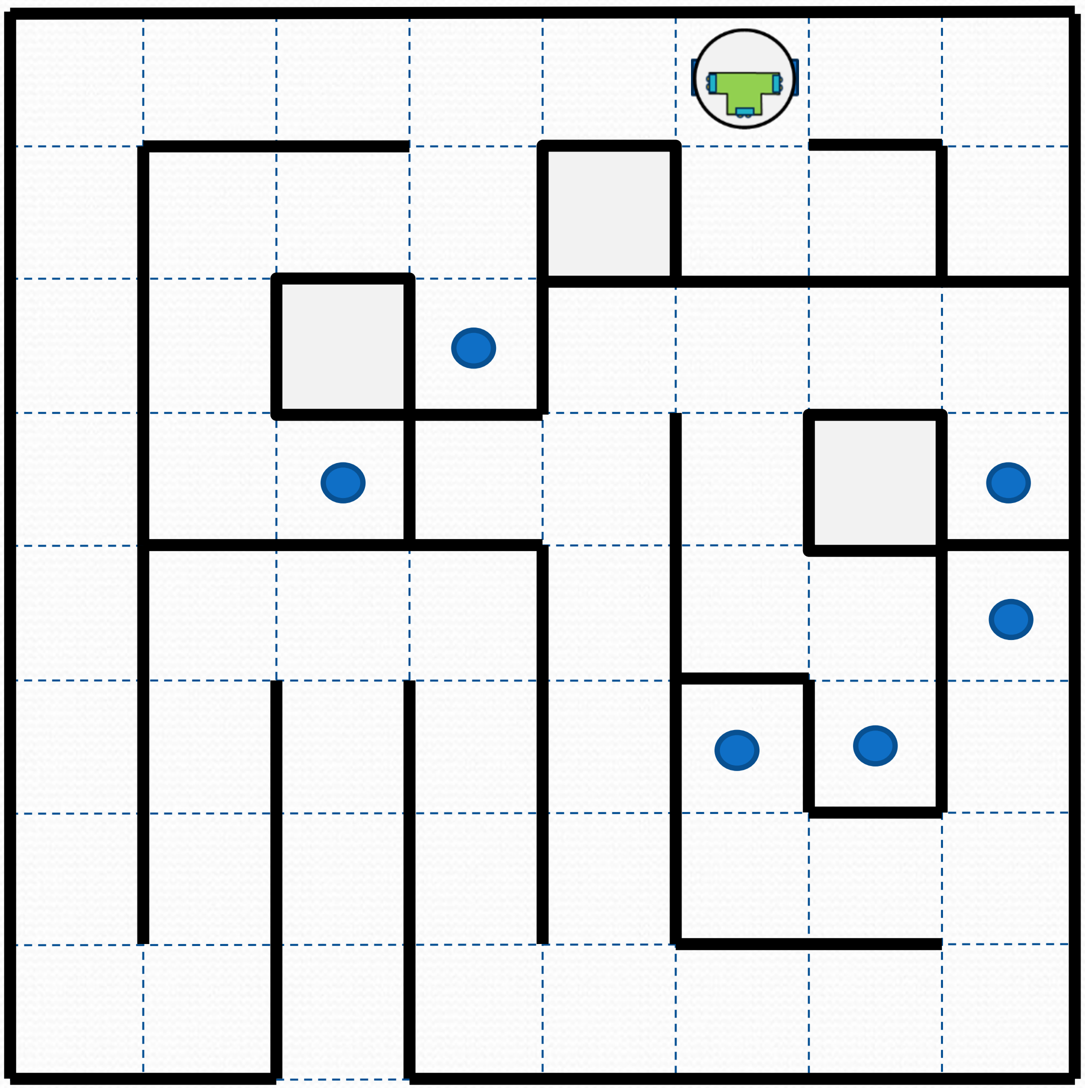
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner)*.

Exercise 23: maze_solve



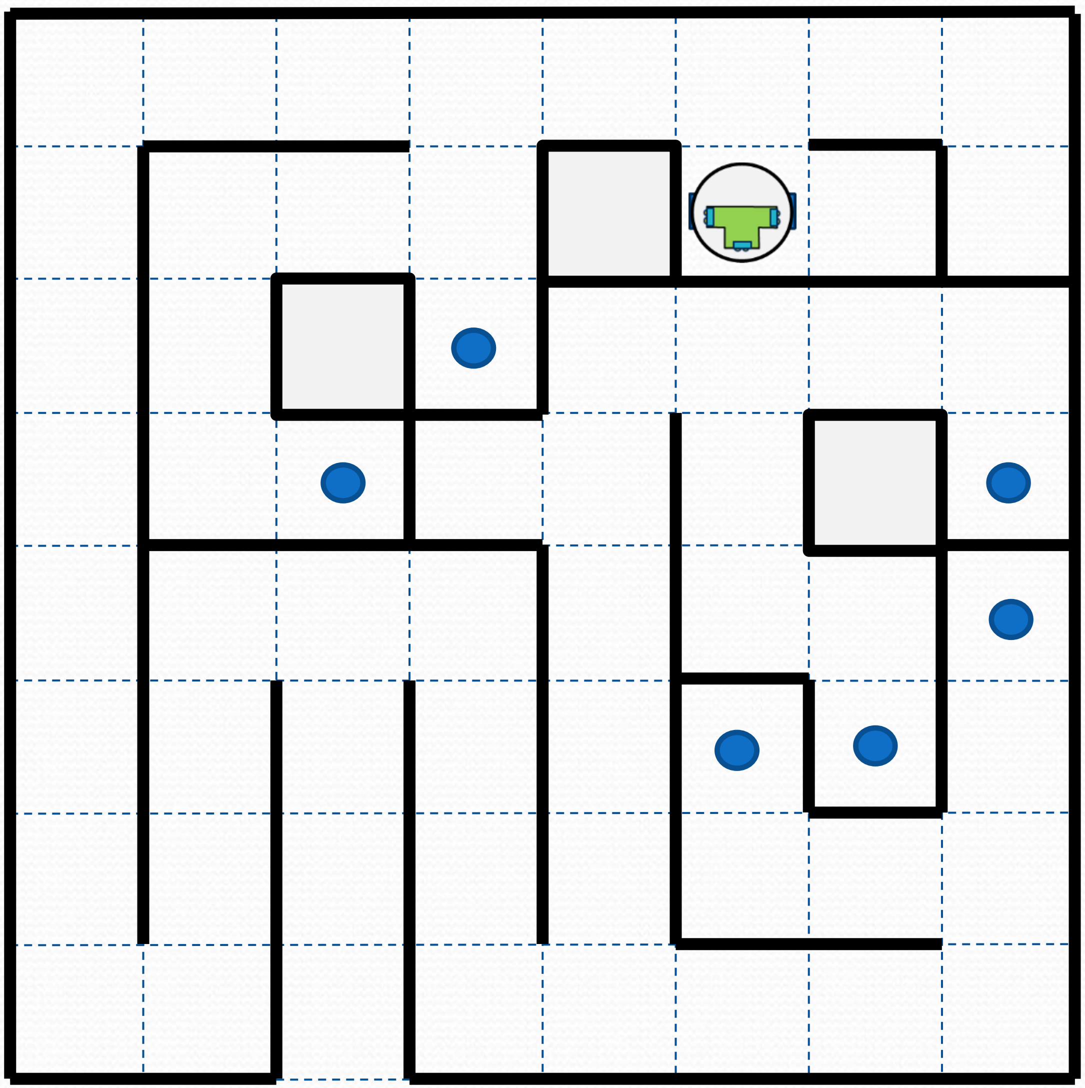
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



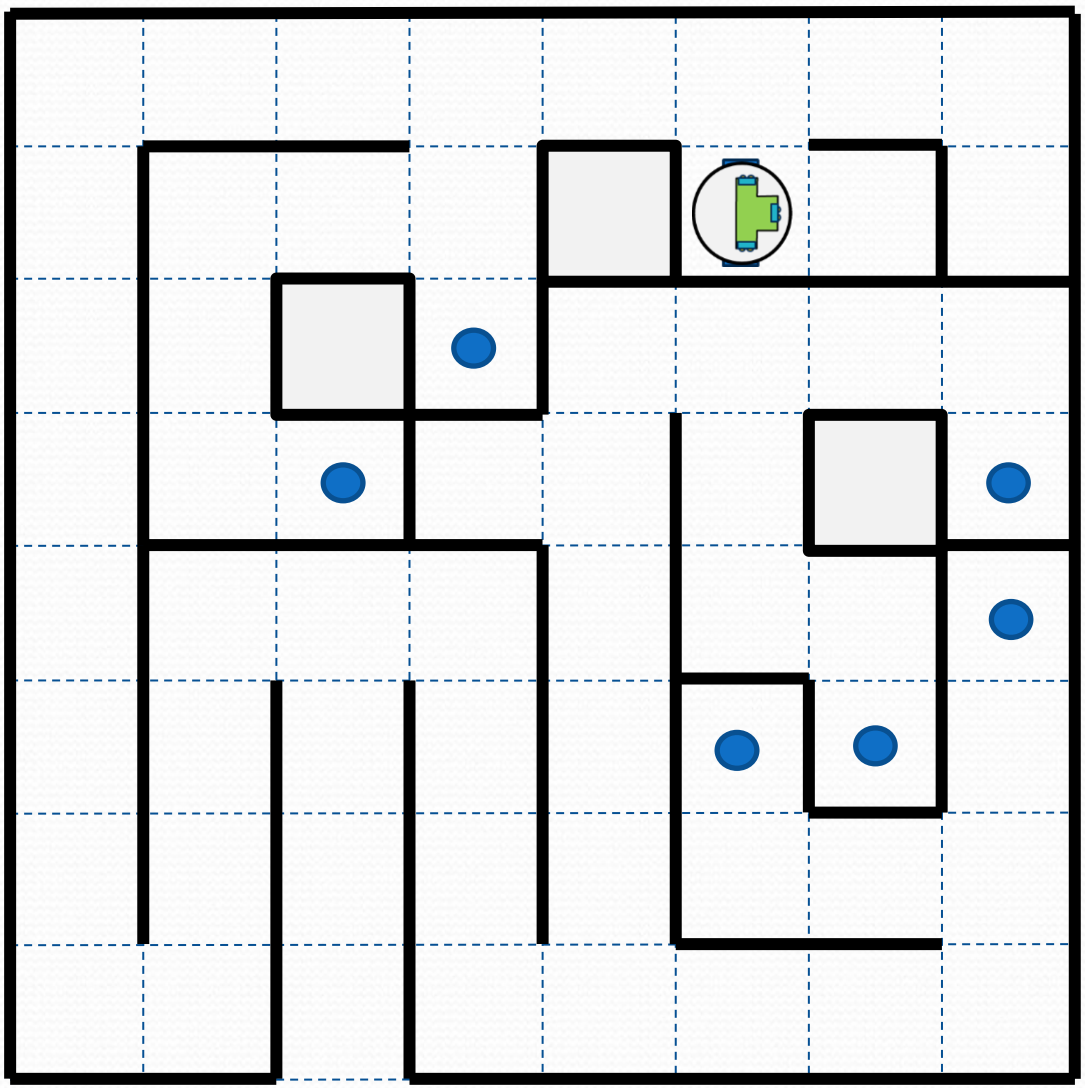
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



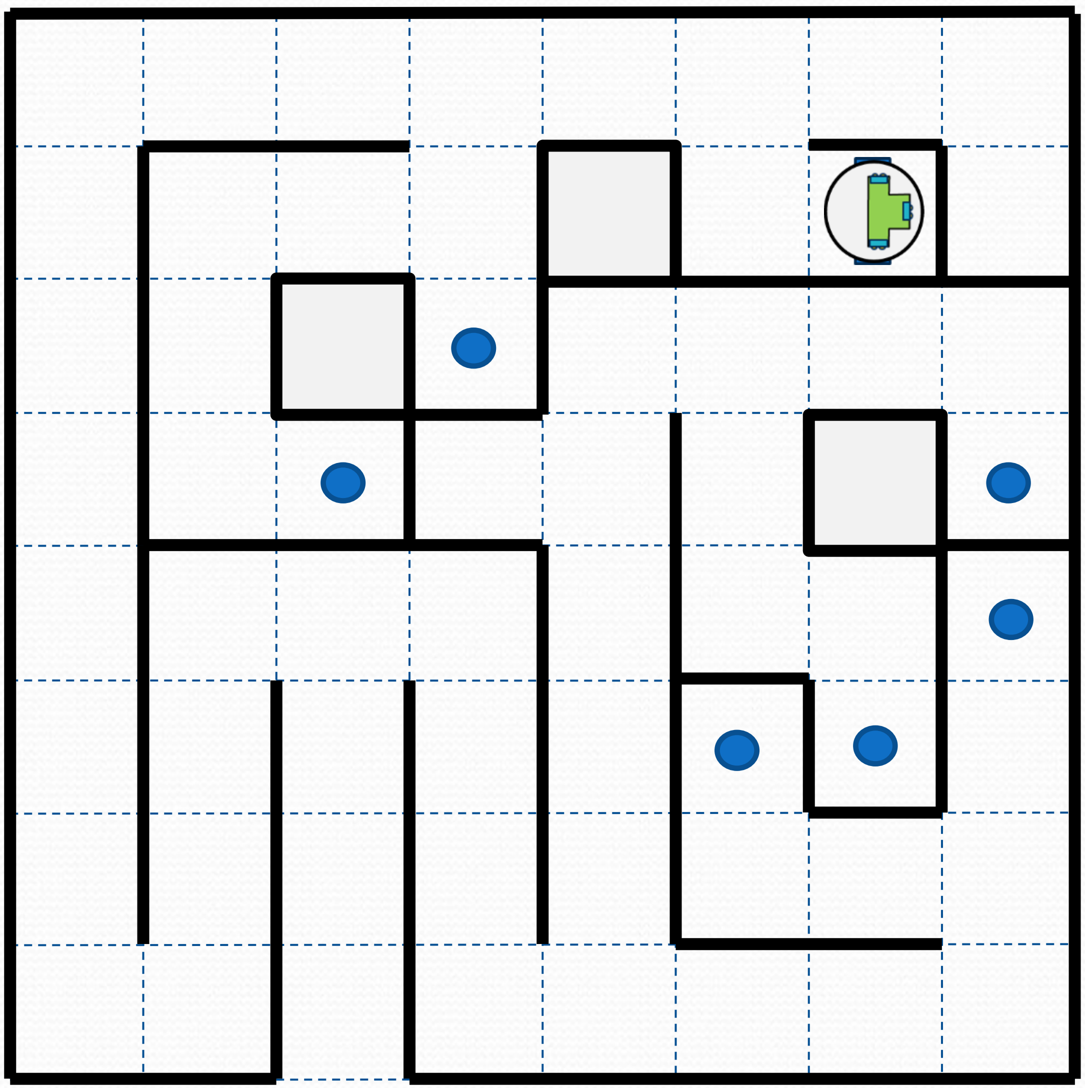
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



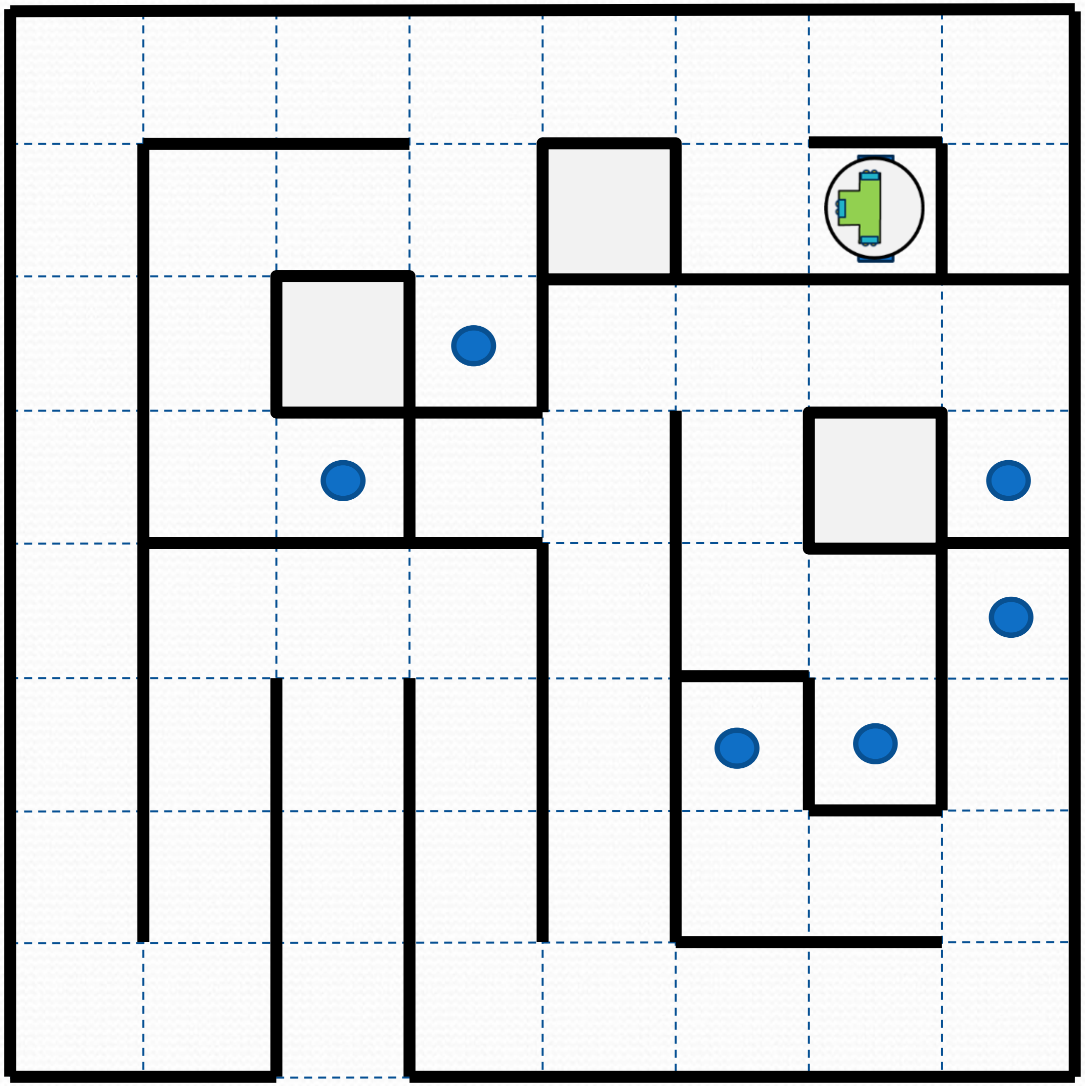
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



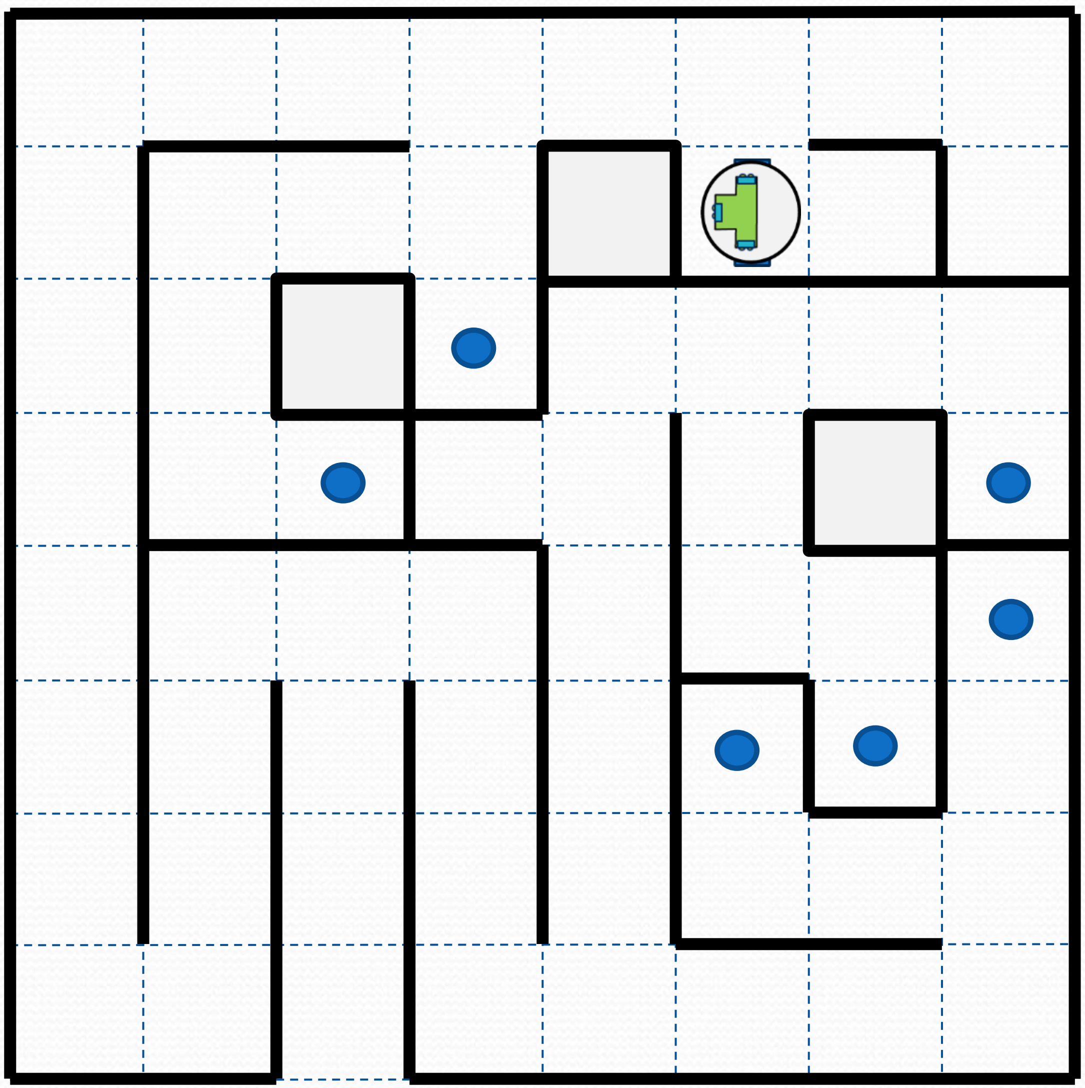
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



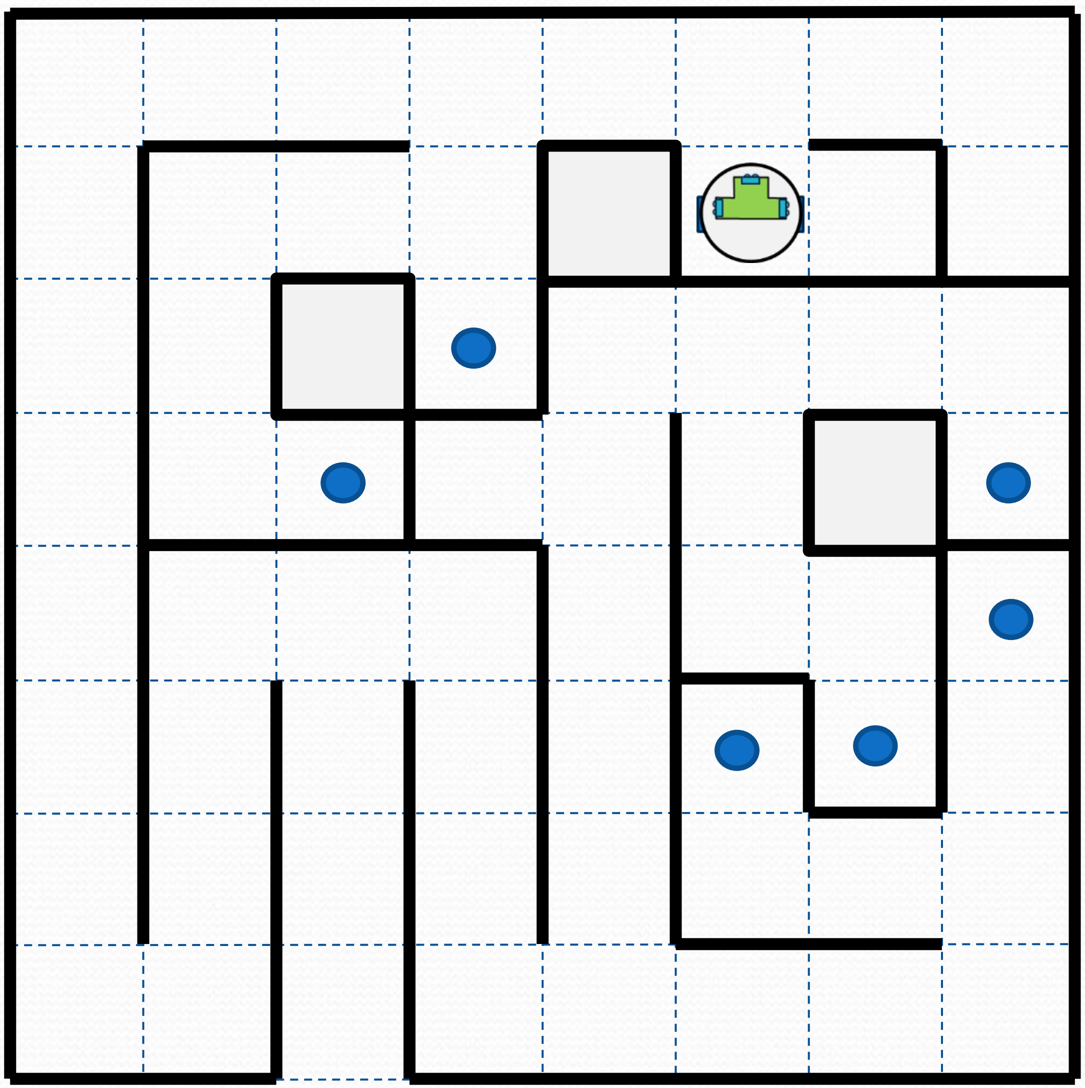
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



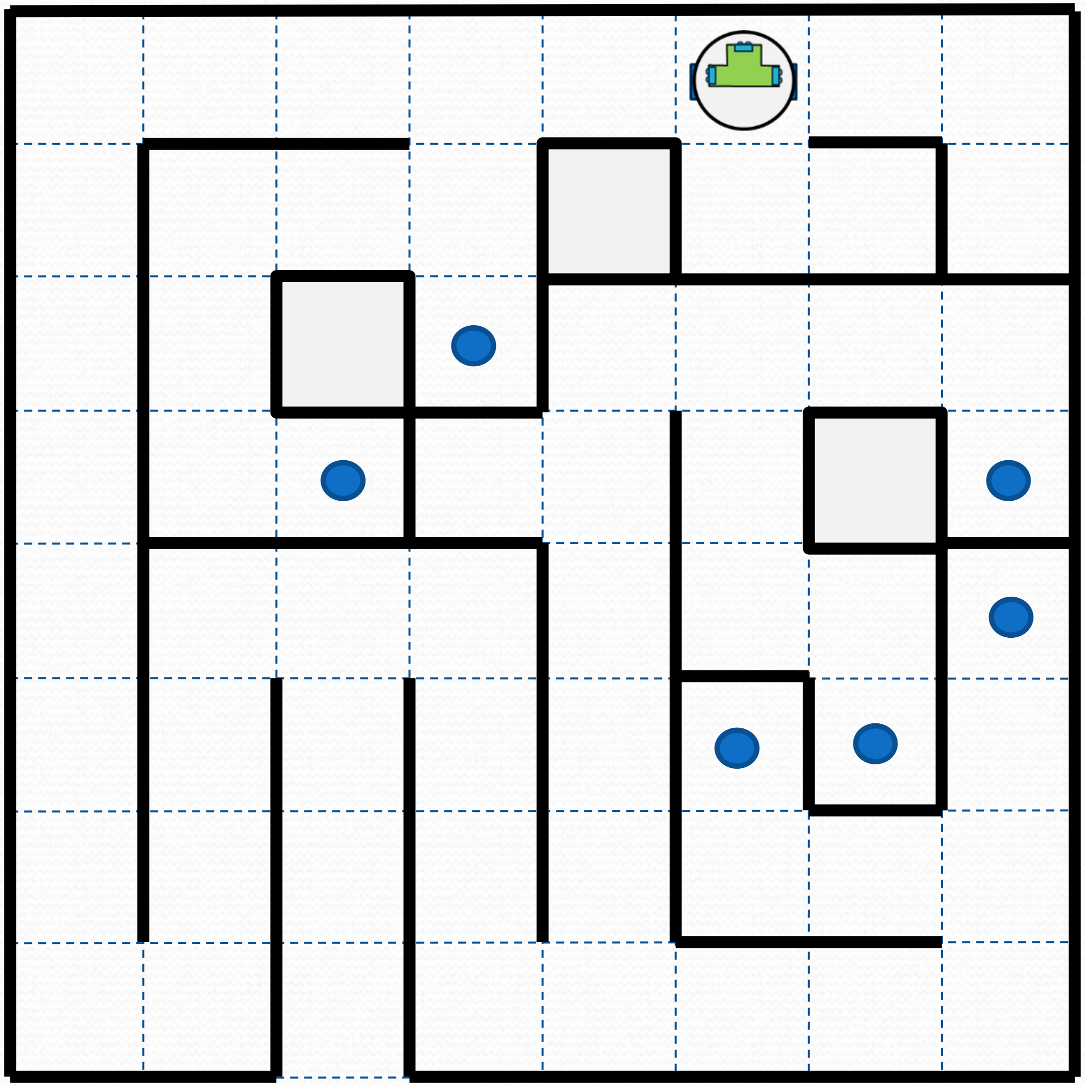
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



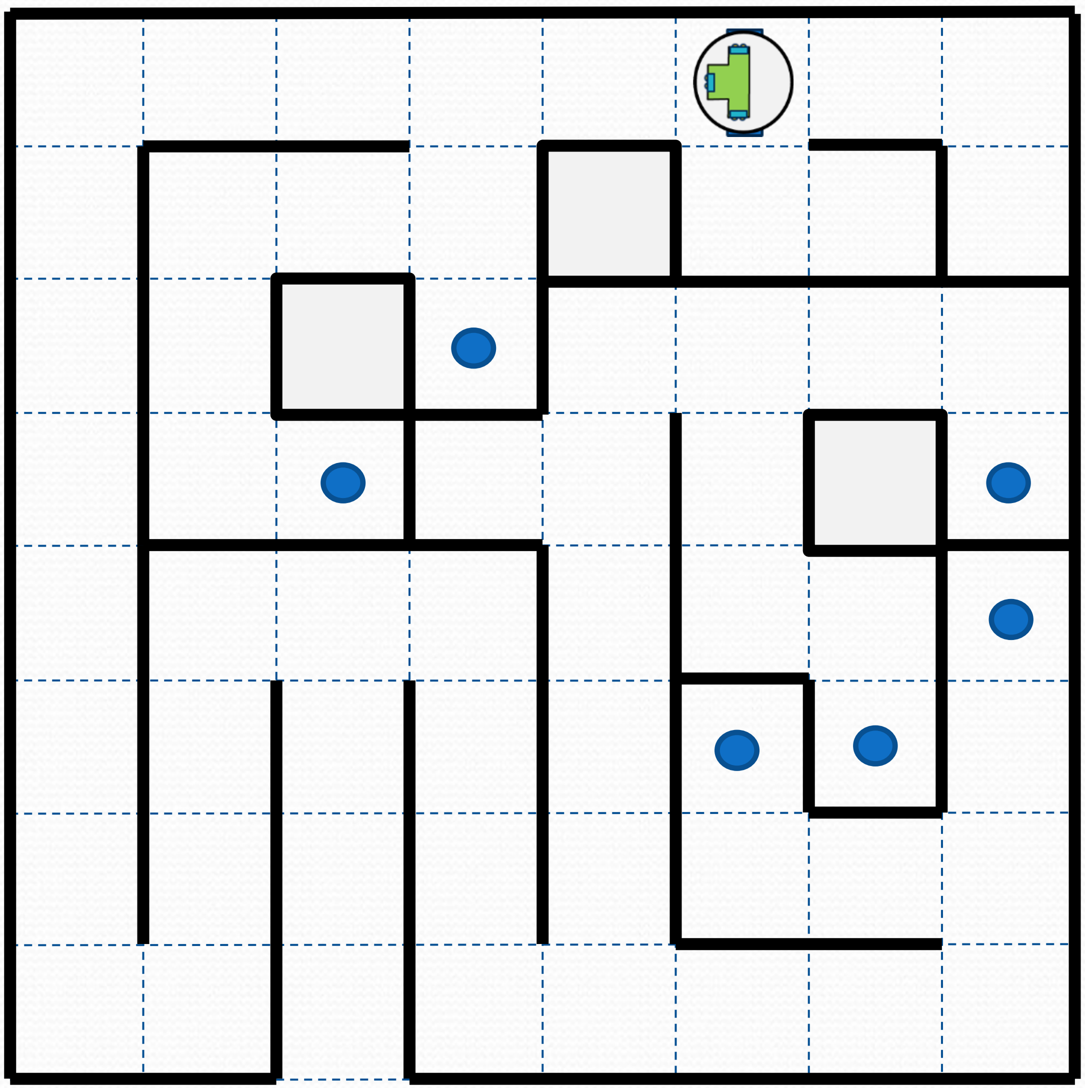
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



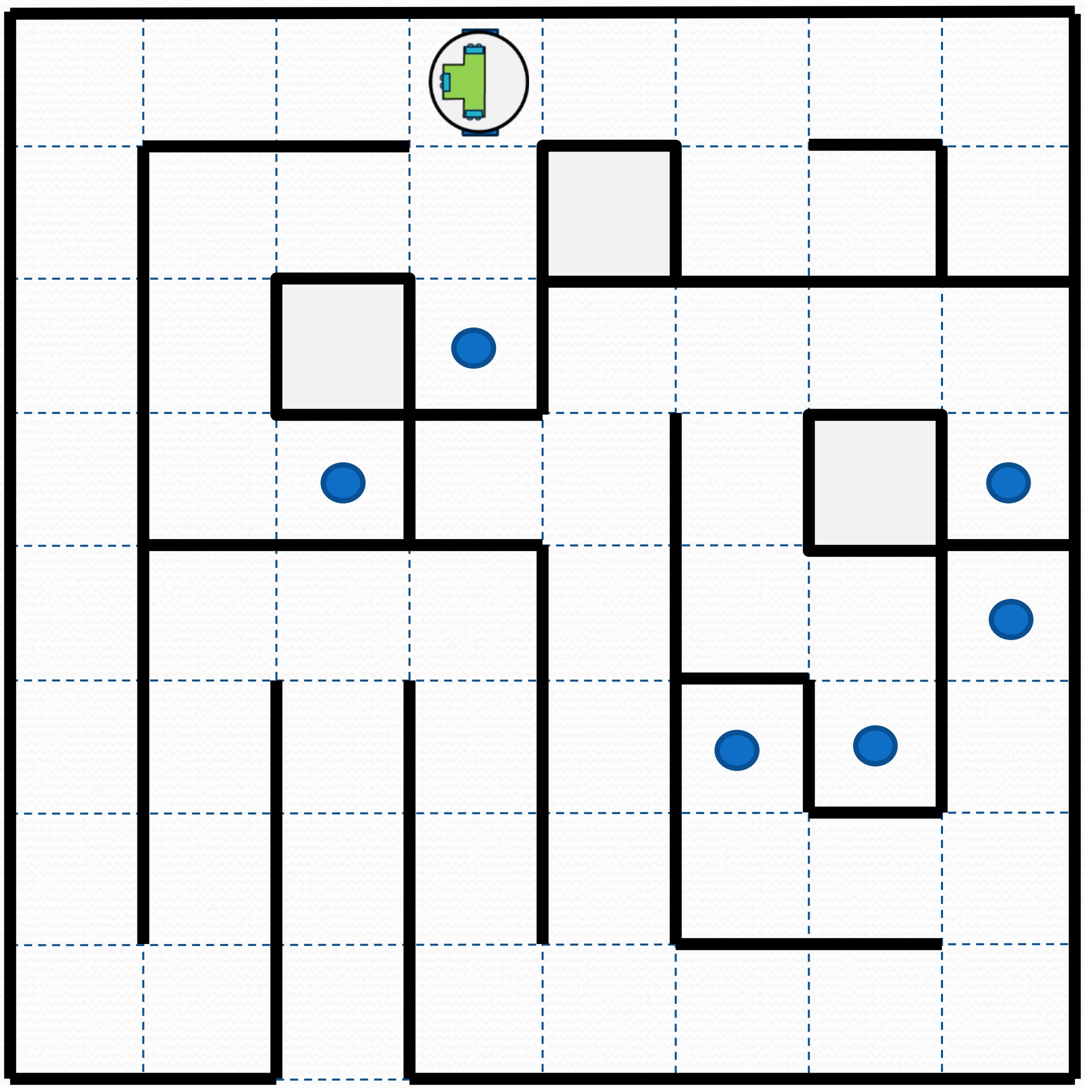
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



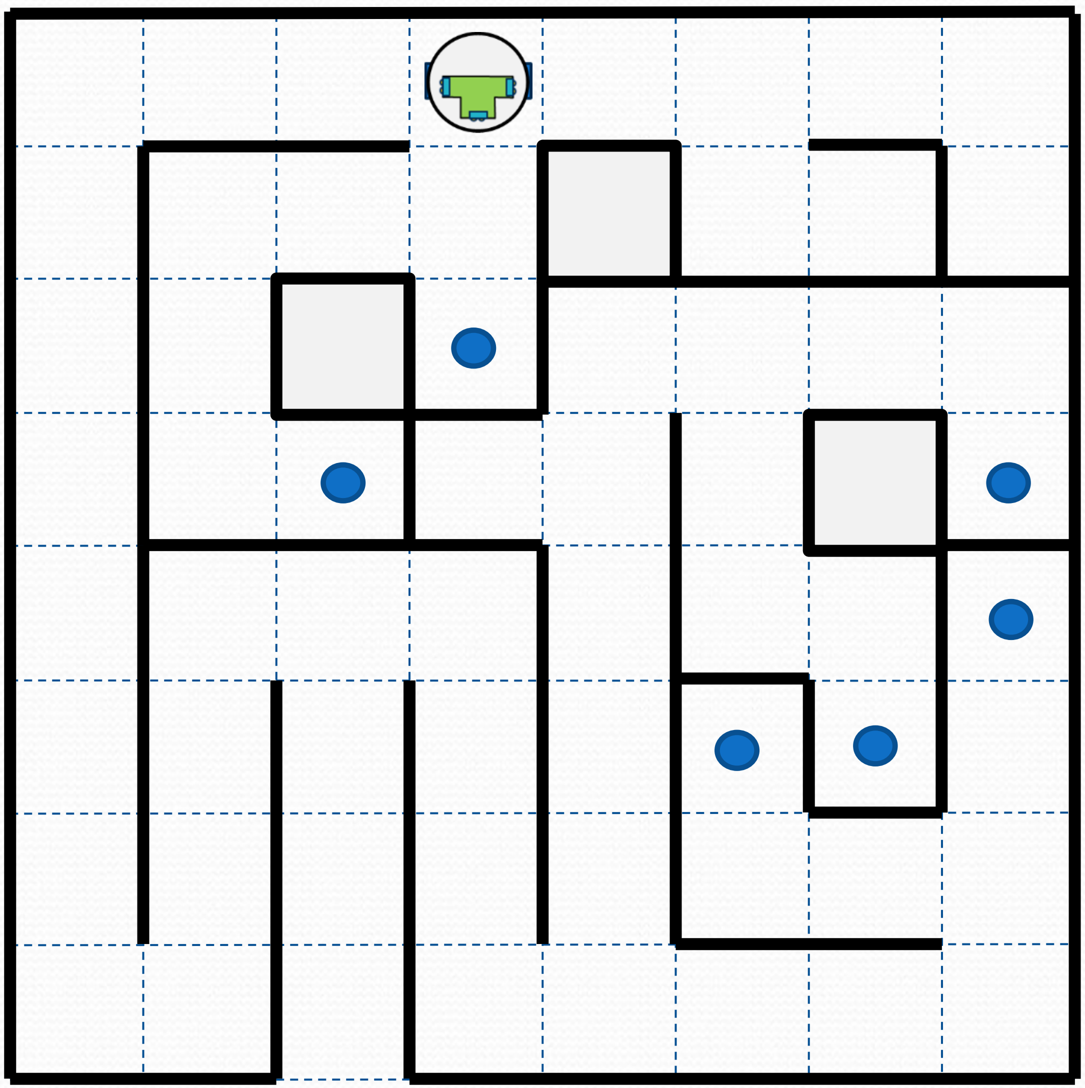
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



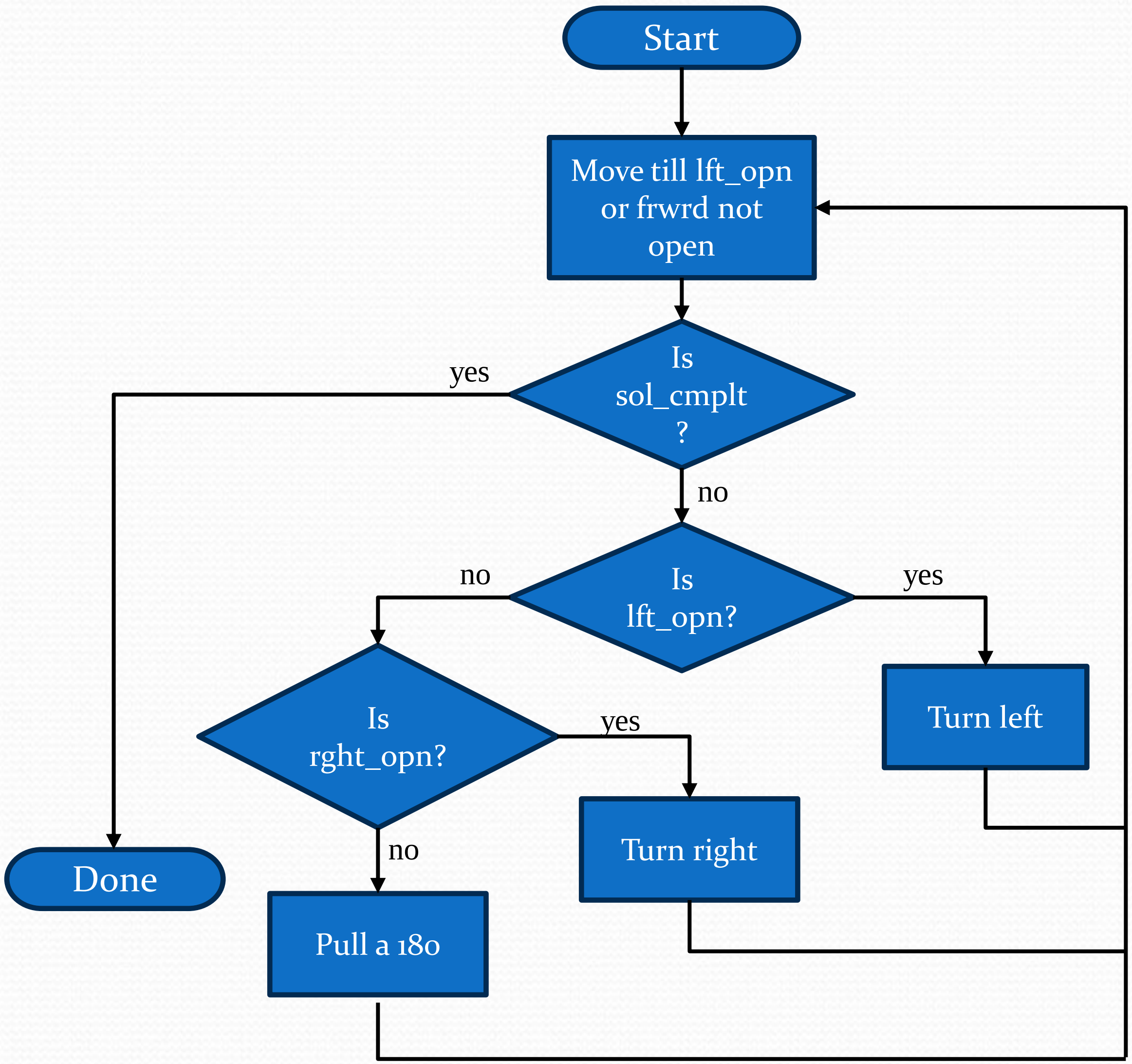
Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve



Consider motions of a left affinity algorithm that found itself in the starting position shown *(upper left corner).*

Exercise 23: maze_solve

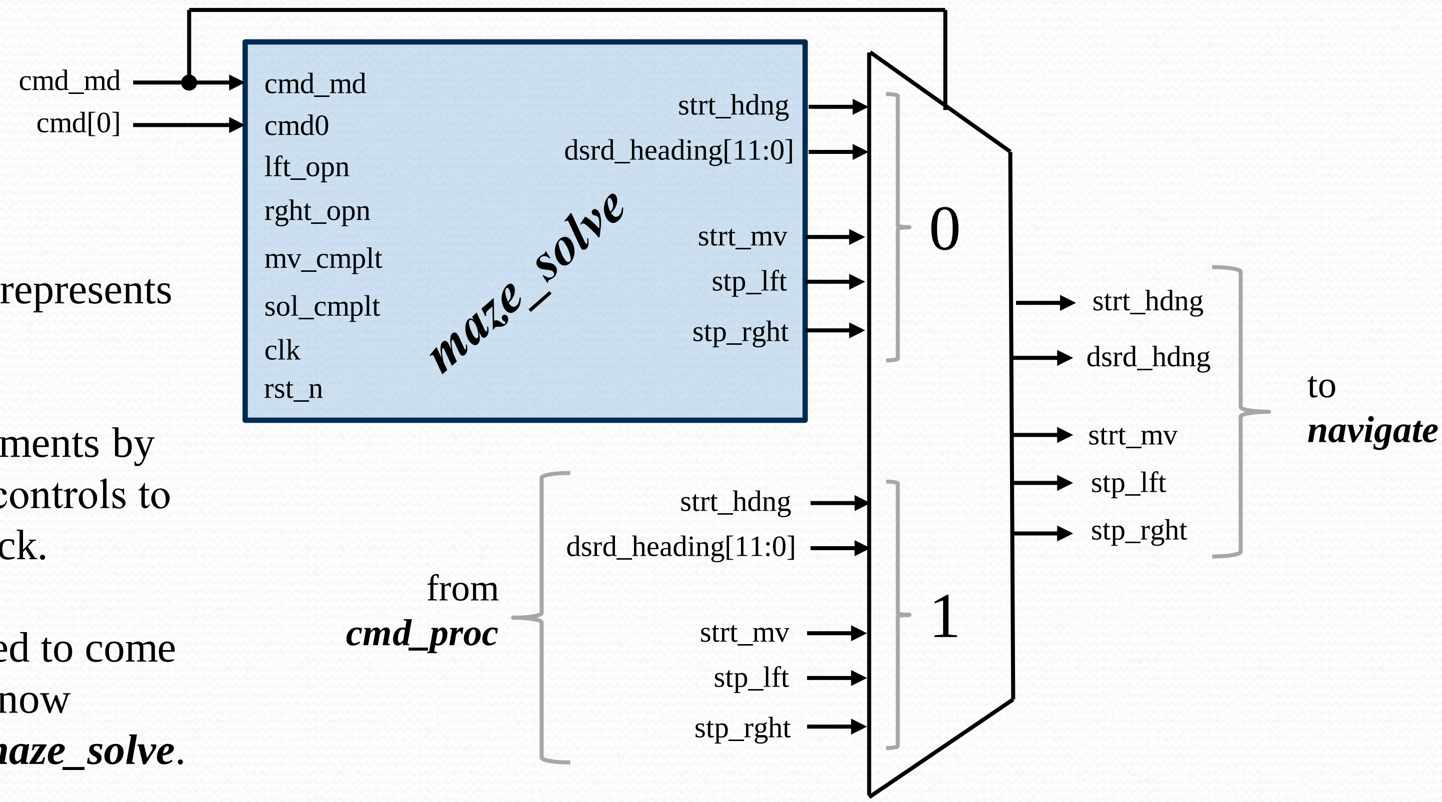


Presented here is a flowchart for a left affinity solution.

Your ***maze_solve*** block should be able to perform a left or right affinity solution depending on the state of **cmd**[0]. If **cmd**[0] is 1 then left affinity, otherwise right affinity.

In addition to a statemachine your design will also have to have a **dsrd_hdng** register to store the newest desired heading. This register can be reset to 12'h000 (north) because the solve command will always be issued right after a gyro calibration.

Exercise 23: maze_solve



- The shaded box represents ***maze_solve***.
- It controls movements by "hijacking" the controls to the ***navigate*** block.
- Controls that used to come from ***cmd_proc*** now originate from ***maze_solve***.
- The muxing of these control signals happens at the toplevel of hierarchy (***MazeRunner***), which is provided later. This is just being shown so you have the context to know the mechanics of how ***maze_solve*** can control movement.

Exercise 23: maze_solve

maze_solve interface:

Signal:	Dir:	Description:
clk,rst_n	in	50Mhz clk and asynch active low reset
cmd_md	in	**cmd_md** going low is what should kick your SM out of IDLE and have it start doing its thing.
cmd0	in	LSB of **cmd**[15:0]. If high perform a left affinity solution.
lft_opn/ rght_opn	in	When **mv_cmplt** we need to know if lft or right is available direction. Turn toward the direction of affinity if open, otherwise turn opposite if open, otherwise pull a 180
mv_cmplt	in	From ***navigate*** unit. Asserted when either hdng or mv command is complete
sol_cmplt	in	Hey…were done! Comes from hall effect sensor that detects magnet
strt_hdng	out	Instructs ***navigate*** unit to turn mazeRunner toward the new **dsrd_hdng**
dsrd_hdng	out	12-bit heading. Need a register to hold this. Can update 1 clk after **strt_hdng** asserted.
strt_mv	out	Instructs ***navigate*** unit to move forward and stop at either lft/rght opening or frwrd wall
stp_lft/stp _rght	Out	Simply derived from **cmd0**. Are we performing lft/rght affinity?

Exercise 23: maze_solve

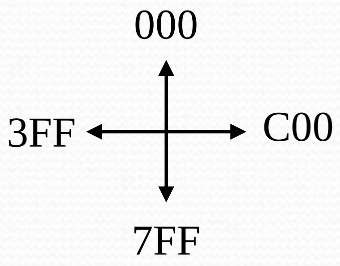
- Updating **dsrd_hdng**[11:0]

  - Your **dsrd_hdng** register should reset to 12'h000 since a maze solve command should always be issued right after a calibrate gyro command so the assumed heading of the ***mazeRunner*** would be north.

  - Updates to **dsrd_hdng** (turning left/right/180) occur through SM control signals.

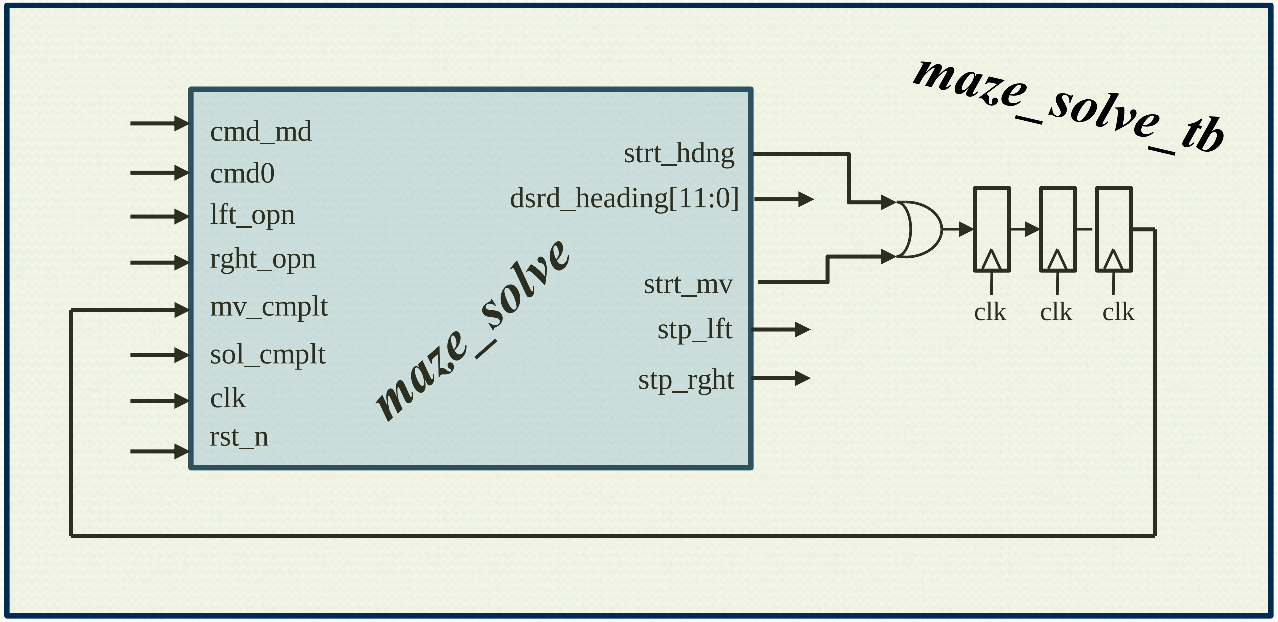
  - Keeping the headings on the orthogonal axes listed below can be a bit tricky. I used the current sign of **dsrd_hdng**, and whether it was zero or not as part of my solution.

cmd[11:0] ➔ Heading	Direction:
12'h000	North
12'h3FF	West
12'h7FF	South
12'hC00	East



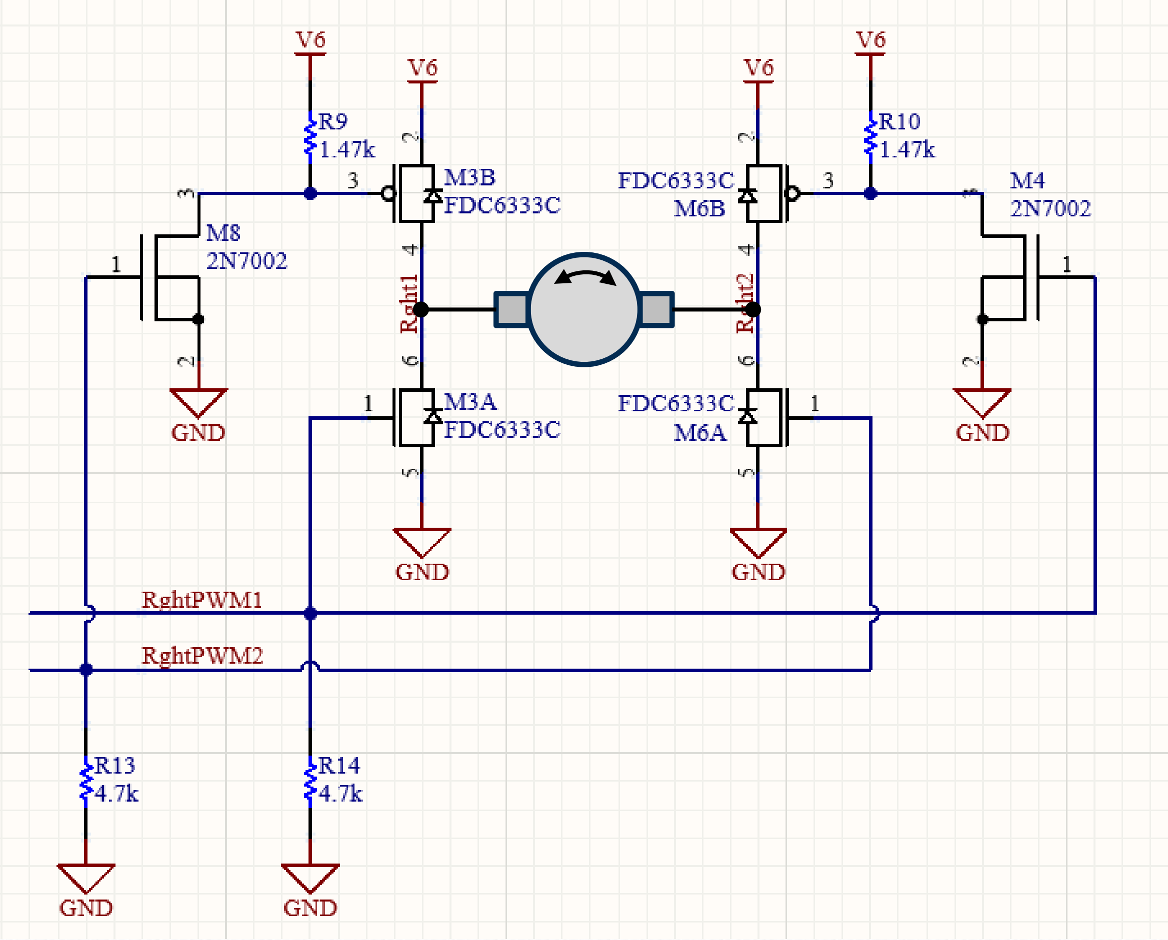
Exercise 23: maze_solve (testing)

- You are going to want to build a unit level testbench to test the basic functionality of ***maze_solve***.
- It is slightly painful to test stand alone since it has a significant number of input/outputs, and the correct outputs are not simply a function of the inputs, but of the history of moves.
- We are now done with all major blocks of the project, and testing this unit more thoroughly in the full chip environment will be easier…but, make sure it is at least reasonably wrung out with a unit level testbench.



Exercise 23: MtrDrv

- The mazeRunner's motors are driven directly off the battery voltage *(nominal 6V).*
- We drive with PWM. At 50% duty cycle roughly half the battery voltage is applied across the motor *(nominal 3V).*
- As batteries wear their voltage falls



- Proper maze navigation requires consistent acceleration/speed/braking behavior. Think about case of discovering a opening to the lft/rght. We want to stop in front of that opening so we can turn into it next.
- We need consistent acceleration/speed/braking regardless of battery level

Exercise 23: MtrDrv

- As battery level falls we want to increase our PWM drive to compensate.

- Could choose to normalize about any battery voltage in normal range of batteries. I chose to do it at 5.4V.

- The desired duty cycle compensation is *(shown only for lft, but same idea for rght)*:

  lft_scaled = (5.4/batt_voltage)*lft_spd

- Division is a high cost operation, as are floating point operations. So we want to do this math in a less expensive way.

Exercise 23: MtrDrv

lft_scaled = (5.4/batt_voltage)*lft_spd

- The ***sensor*** block *(will be provided)* measures the battery voltage through a voltage divider using the A2D converter.
- Accounting for the scaling in the measurement circuits the upper 8-bits [11:4] of the battery reading are related to the actual battery voltage as follows:

vbatt[11:4]	Batt Voltage
0xC0	4.74V *(too low)*
0xD0	5.14V *(lowest accepted)*
0xDB	5.41V (unity scaling)
0xE0	5.534V *(attenuated)*
0xF0	5.929V
0xFF	6.3V (fresh battery)

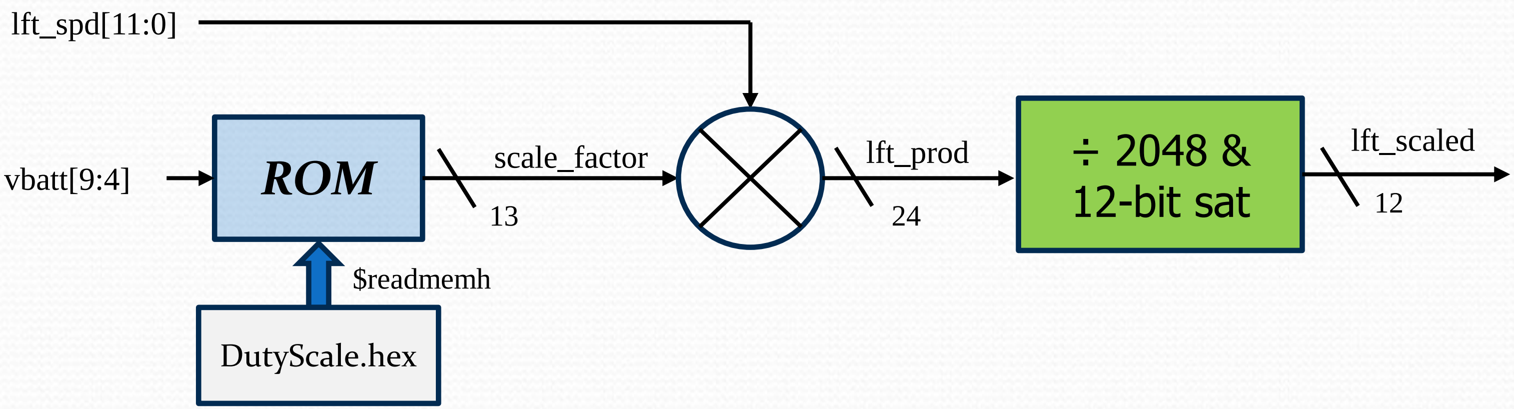
- The upper 2-bits of **vbatt** are always 11 for the range of interest, So using **vbatt**[9:4] gives us enough range and granularity to perform the PWM duty cycle scaling we desire.

Exercise 23: MtrDrv

$$\text{lft\_scaled} = (5.4/\text{batt\_voltage})*\text{lft\_spd}$$

$$2048*\text{lft\_scaled} = \underbrace{(11059/\text{batt\_voltage})}*\text{lft\_spd}$$

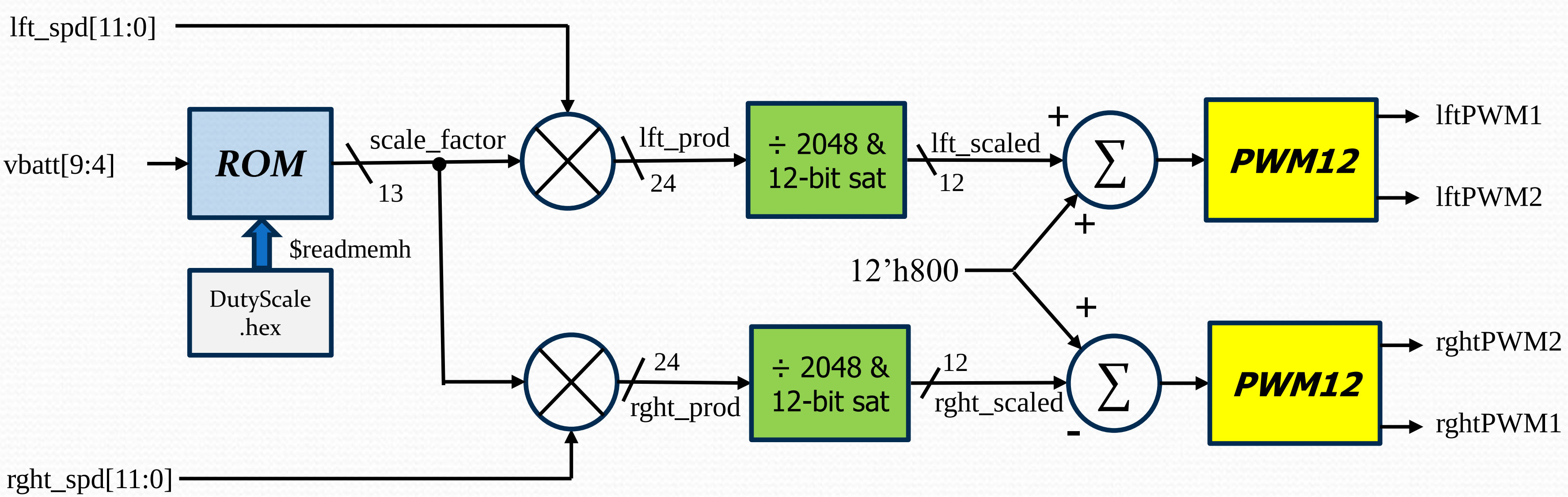
Instead of doing division *(high cost)* we will do a lookup table using **vbatt**[9:4] to represent the battery voltage. We have plenty of memory, so a 64 entry look up table *(ROM)* with 13-bit entries is nothing. The calculations to compute the scale factors is done in Excel and exported as a file for use with $**readmemh**



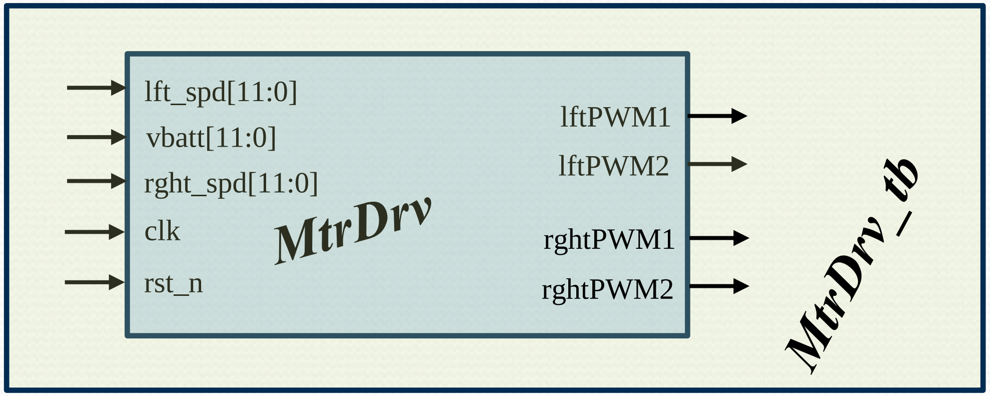
Exercise 23: MtrDrv

- A model of a ROM and its associated .hex file with scaling factors are provided. In addition to scaling ***MtrDrv*** also has to convert the 12-bit signed duty cycle to unsigned. This is simply done by adding 0x800.

- The motors in the ***MazeRunner*** are 180° in orientation. We need to invert rght motor speed to account for this. See negation when converting from signed to unsigned.



Exercise 23: MtrDrv (testing)



vbatt[11:4]	Batt Voltage
0xC0	4.74V *(too low)*
0xD0	5.14V *(lowest accepted)*
0xDB	5.41V (unity scaling)
0xE0	5.534V *(attenuated)*
0xF0	5.929V
0xFF	6.3V (fresh battery)

These test scenarios pertain to lft

- Create a simple testbench.
  - Zero in should give 50% duty cycle out regardless of **vbatt**
  - 0x3FF in should with **vbatt**[11:4]==0xDB should give approx *(slightly less)* 75% duty on PWM1 and 25% on PWM2
  - 0x3FF in with **vbatt**[11:4]==0xD0 should give approx 76.2% duty cycle on PWM1 and 23% on PWM2
  - 0xC00 in with **vbatt**[11:4]==0xFF should give approx 28.5% duty cycle on PWM1 and 71.4% on PWM2