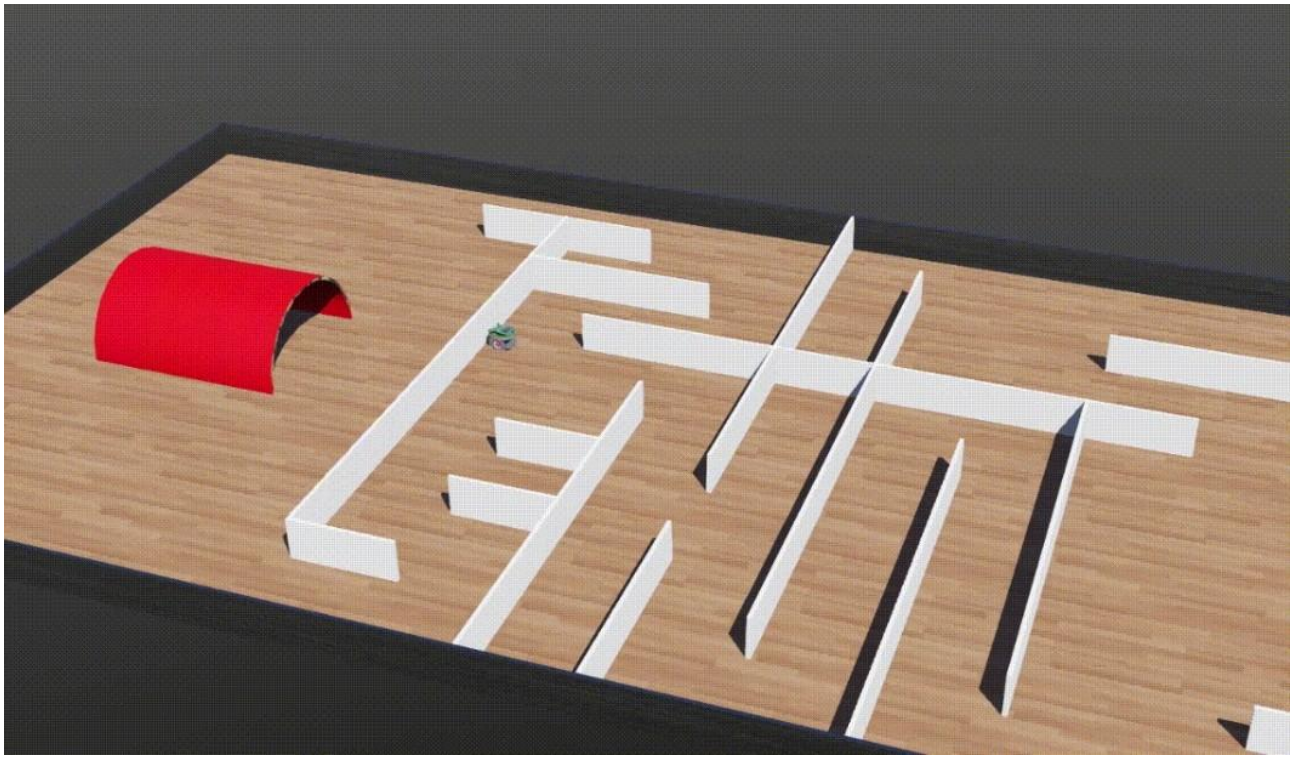
Robotics and Python. Maze Solver.

Sila, Eaaa. DMU. November 2024.

Maze solver.

In this exercise the objective is to get out of, solve, the maze,
and get to the red hangar.



A number of algorithms might be useful here.
E.g
Hand On Wall Rule, also known as either the left-hand rule or the right-hand rule. If the maze is simply connected, that is, all its walls are connected together or to the maze's outer boundary, then by keeping one hand in contact with one wall of the maze the solver is guaranteed not to get lost and will reach a different exit if there is one; otherwise, the algorithm will return to the entrance having traversed every corridor next to that connected section of walls at least once.

For more, see
https://en.wikipedia.org/wiki/Maze-solving_algorithm

"***Wall following***" (pseudo-code).
Here is the wall follower algorithm (the left-hand one) at a high level:

If left is free:
  Turn Left
Else if left is occupied and straight is free:
  Go Straight
Else if left and straight are occupied:
  Turn Right
Else if left/right/straight are occupied or you crashed: Turn 180 degrees

Template code (E-puck, Webot):
Wall_Following.py (Canvas).

"***Bug0 algorithm***" (pseudo-code).
While many planning algorithms assume global knowledge,
bug algorithms assume only local knowledge of the environment,
and a global goal position.
I.e. bug behaviors are simple:

· Follow a wall (right or left)

· Move in a straight line toward goal

Template code (E-puck, Webot):
Bug0.py (Canvas).

Exercise 1.

In groups of 2-3.

It is now you job to make the actual implementation (that works…) of an algorithm that can drive the E-puck robot from the start to the red hangar.

When you have implemented the algorithm

a) You should run the simulation at high speed in Webot.
b) While you make a recording of the simulation
   (The recording can be used as demo of your work).