Design a Line Maze Solving Robot

Teaching a Robot to Solve a Line Maze

By

Richard T. Vannoy II

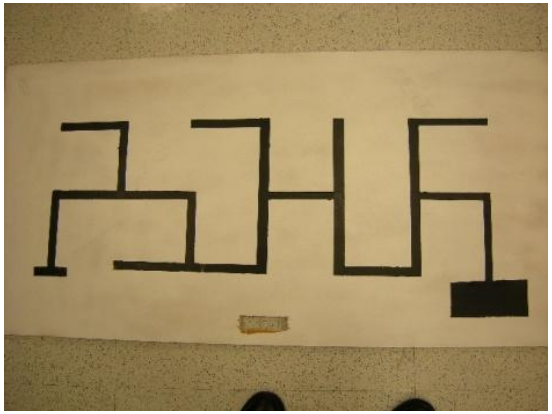
April 2009

RoboticsProfessor@gmail.com

Please email me at the address above if you have questions or comments.

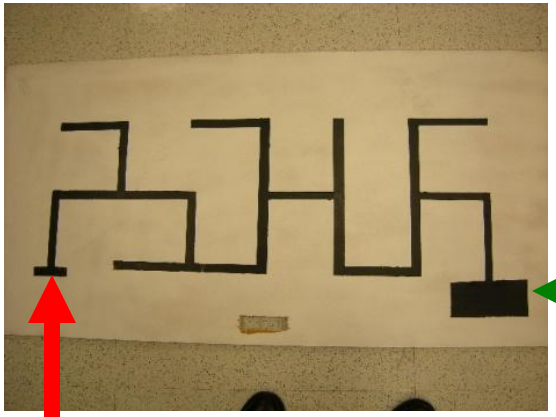
What is a Line Maze?

A line maze is usually a black line on a white background. It could also be a white line on a black background, but for this presentation black lines on a white background will be used.



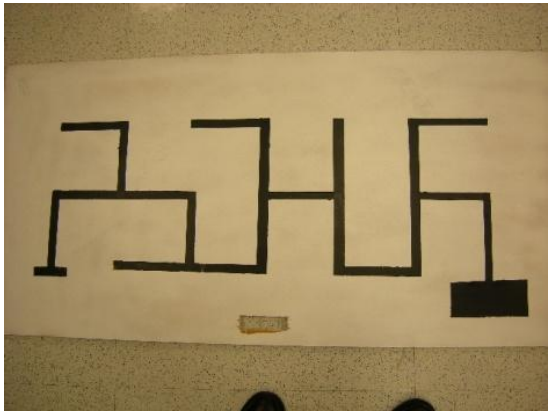
What is a Line Maze?

Each line maze has a Start point and a Finish point. The robot is expected to follow the lines and find it's way from Start to Finish in the fastest time possible.



What is a Line Maze?

The actual course can be fairly simple, as the maze on the left, or it can get very complicated.

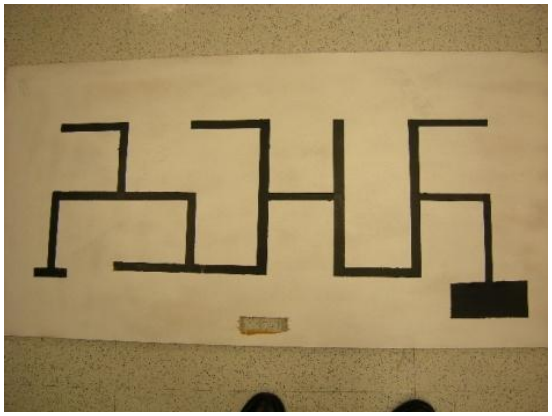


Finish

Start

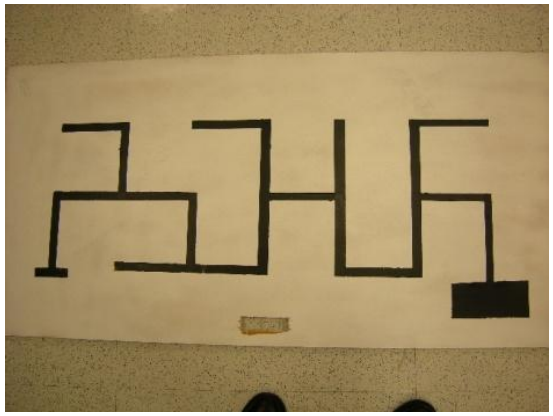
What is a Line Maze?

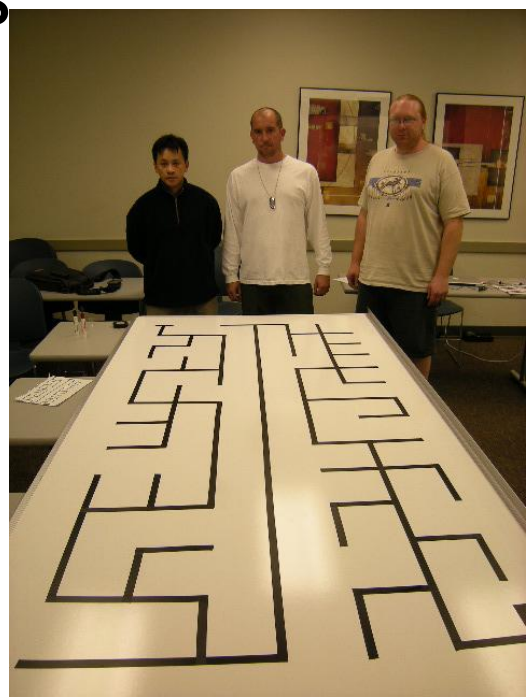
The course on the right was designed with several long straight paths to give an advantage to a robot that knows when it can increase speed.



What is a Line Maze?

Notice that there are a number of dead-end paths in the maze. The robot typically cannot traverse the maze without first taking a number of wrong turns





Solving a Line Maze

This slide show will walk a robot hobbyist through the logic and procedure a robot needs to solve a line maze such as the one shown here.

Left Hand Rule

For this presentation, the robot will always use the left hand rule, which means:

1. Always prefer a left turn over going straight ahead or taking a right turn.
2. Always prefer going straight over going right.

If the maze has no loops, this will always get you to the end of the maze.

Right Hand Rule

The right hand rule is just the opposite:

1. Always prefer a right turn over going straight ahead or taking a left turn.
2. Always prefer going straight over going left.

If the maze has no loops, this will always get you to the end of the maze.

Which Rule Do I Use???

It really doesn't matter.

Both the left hand and the right hand rules will get you to the end of a simple maze.

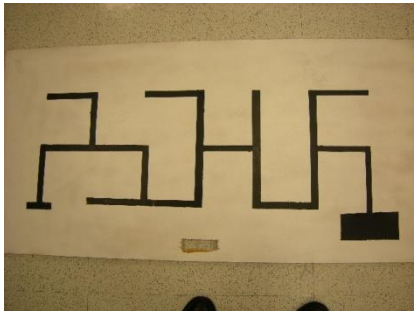
Which you select is purely a matter of personal preference.

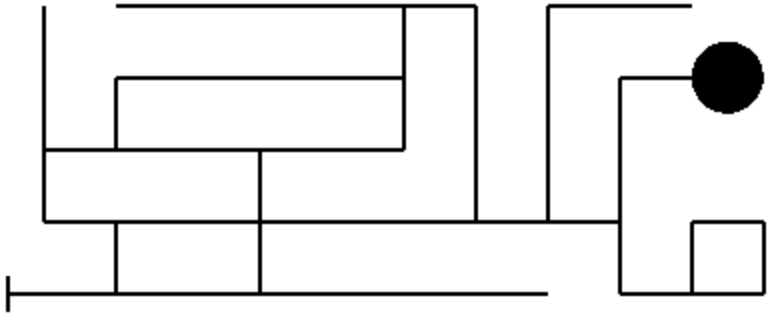
Just pick one and be consistant.

Simple Maze

In the two mazes below, notice that:

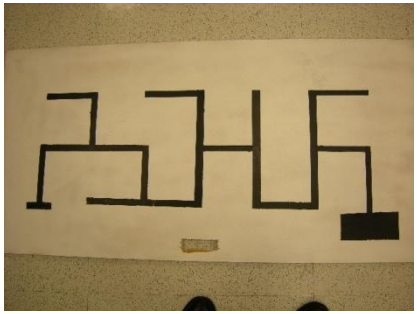
1. The left hand maze has no loops. Using the left hand (or right hand) rule will always get you to the end of the maze.
2. The right hand maze has several loops.

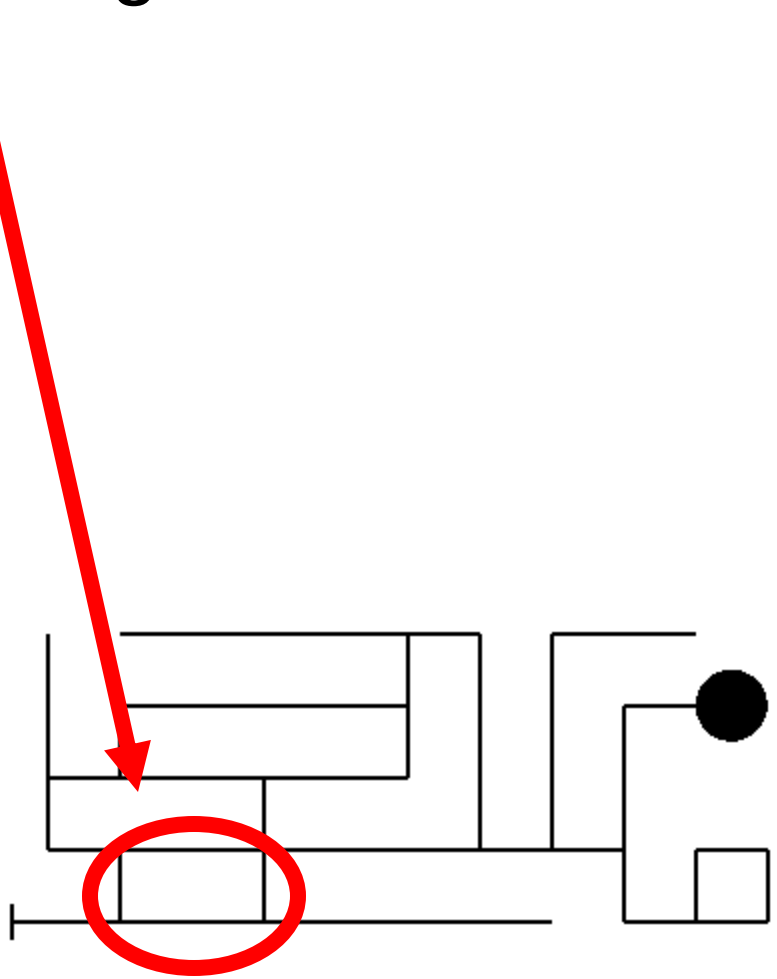




Simple Maze

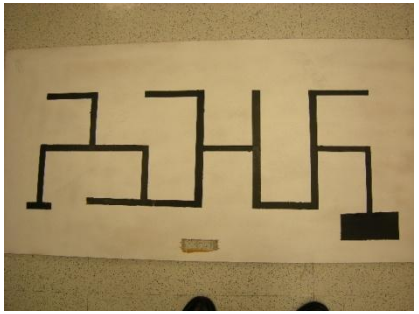
Notice the loops in the right hand maze.

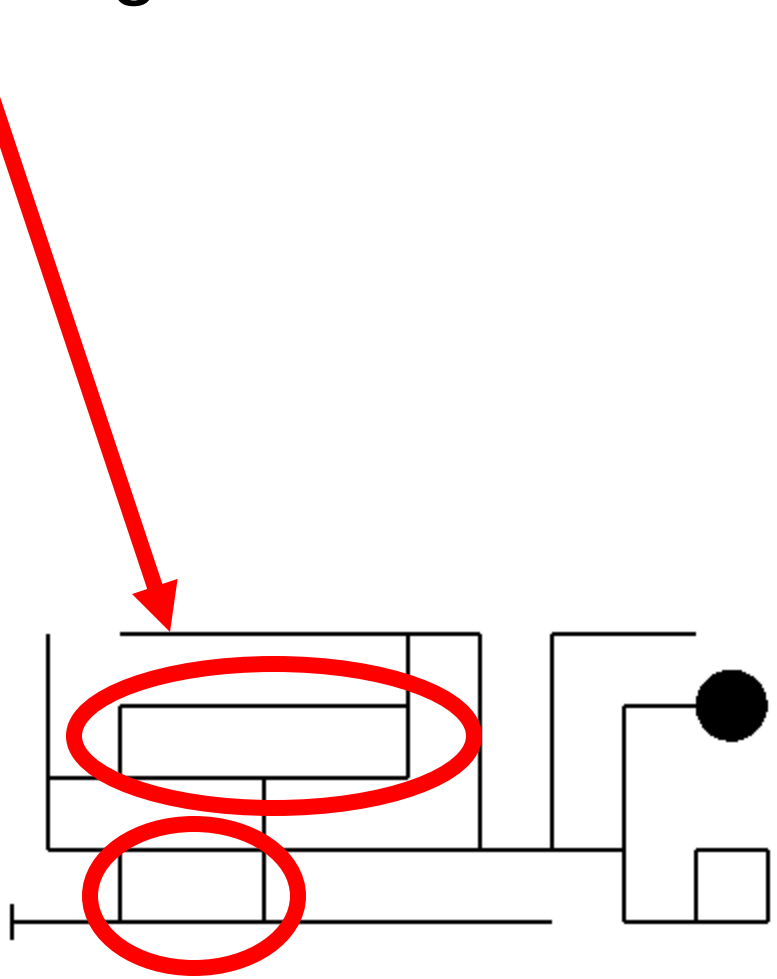




Simple Maze

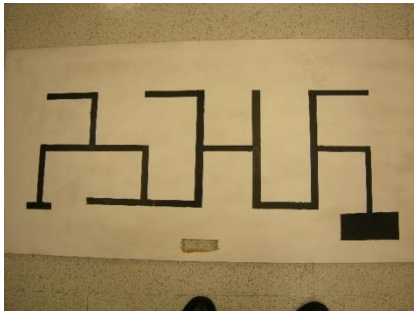
Notice the loops in the right hand maze.

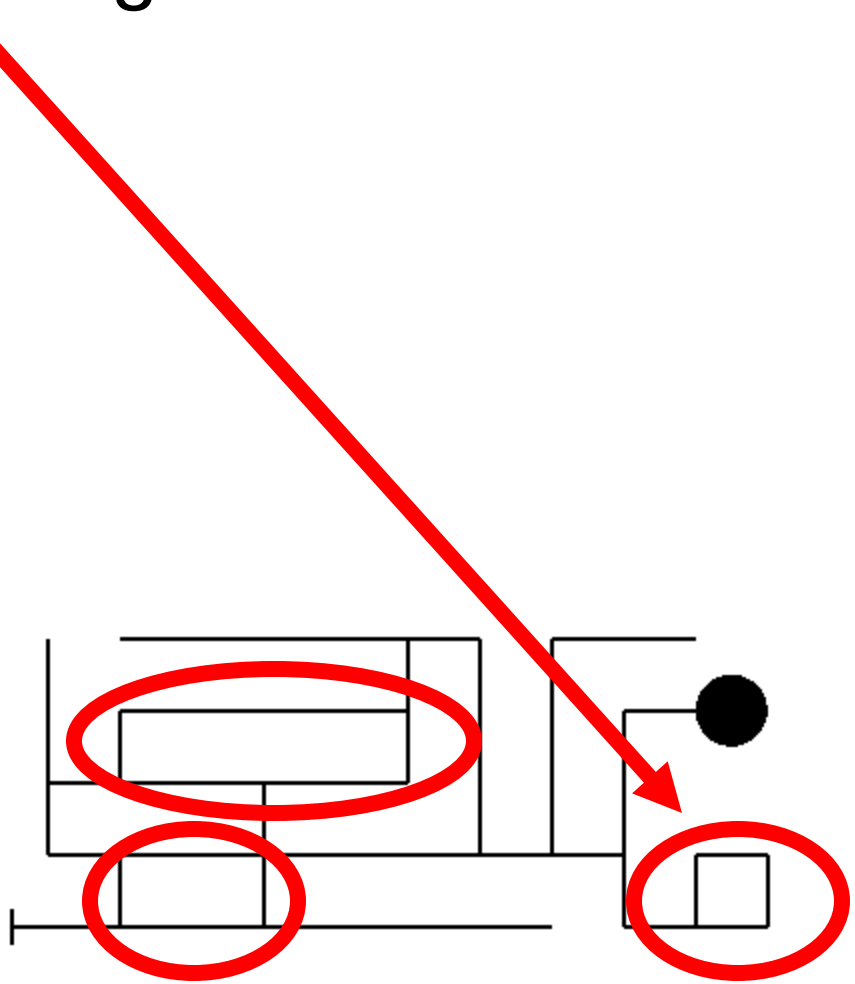




Simple Maze

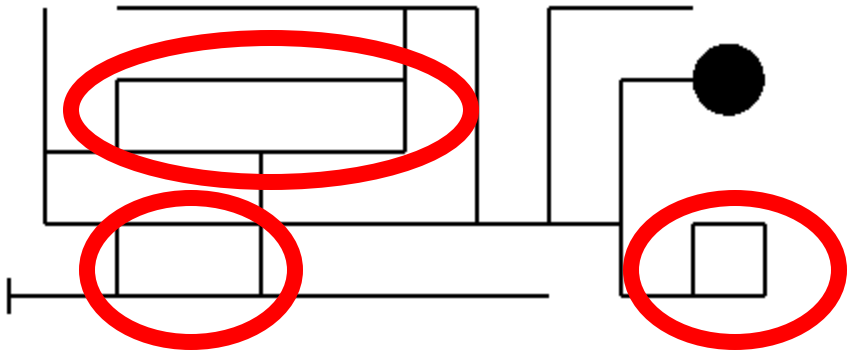
Notice the loops in the right hand maze.





Simple Mazes Only

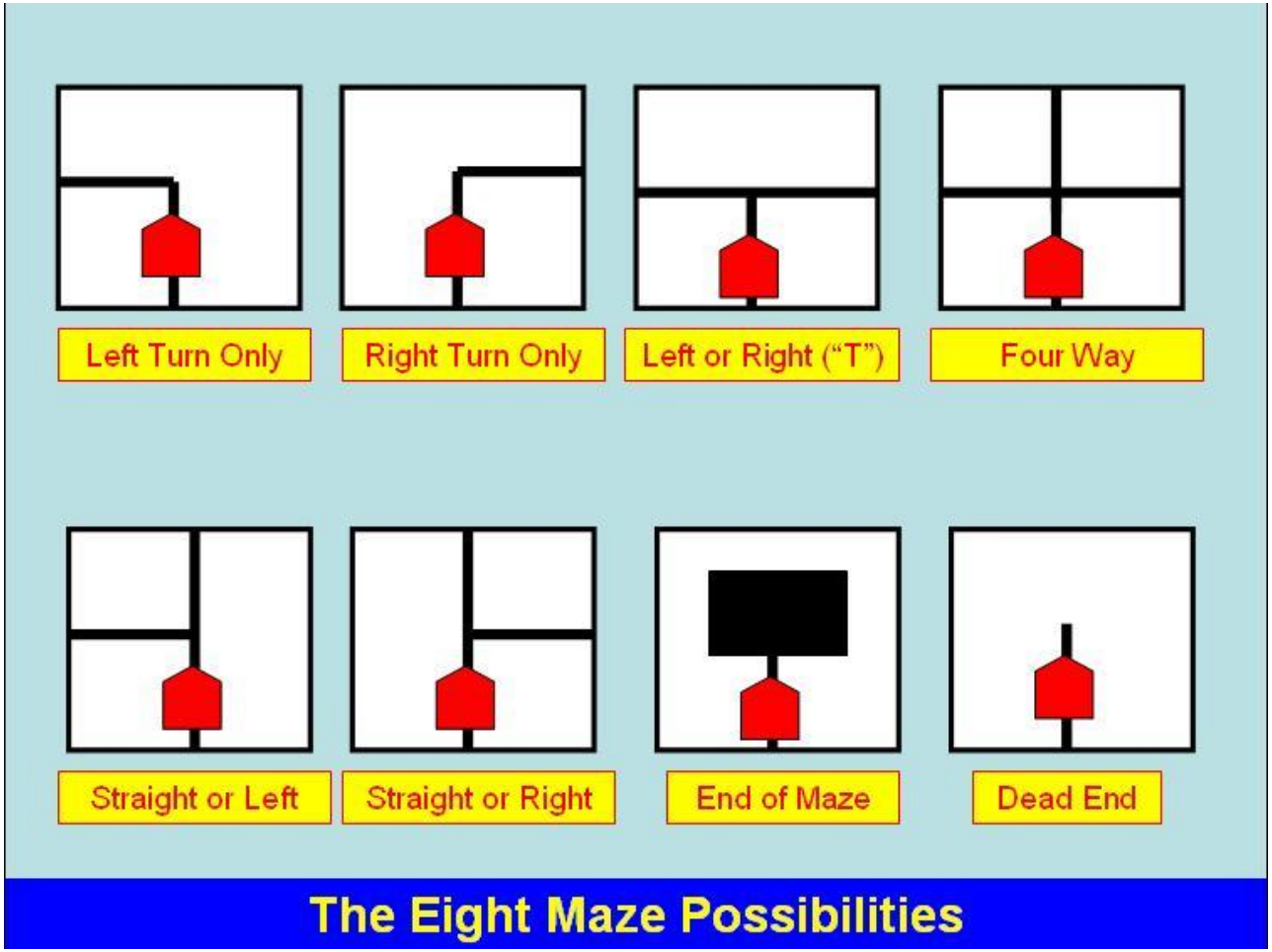
At the present time, this presentation does not address how to solve the maze below. This algorithm may be added at a future date.



The 8 Possiblities

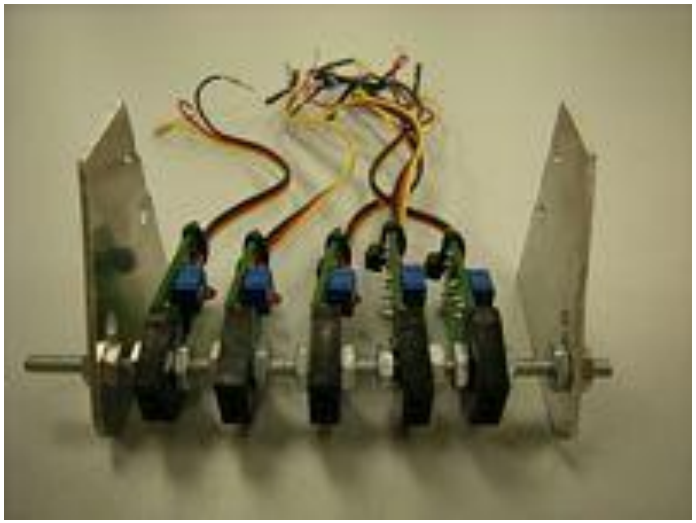
Given a maze with no loops, there are only 8 possible situations that the robot can encounter.

We'll come back to this in future slides.



Line Sensors

- Line sensors can have a number of configurations.
- This link Parallax.com has a great article using four sensors to solve a line maze.
- This presentation will assume a robot with five infrared sensors like those shown here.



Line Sensors Use

Line sensors shine visible or infrared light down at the floor and the measure the reflection

Using a 1 to mean “Sensor sees black.” and 0 to mean “Sensor sees white.”, a robot travelling along the black line to the right might produce several patterns:

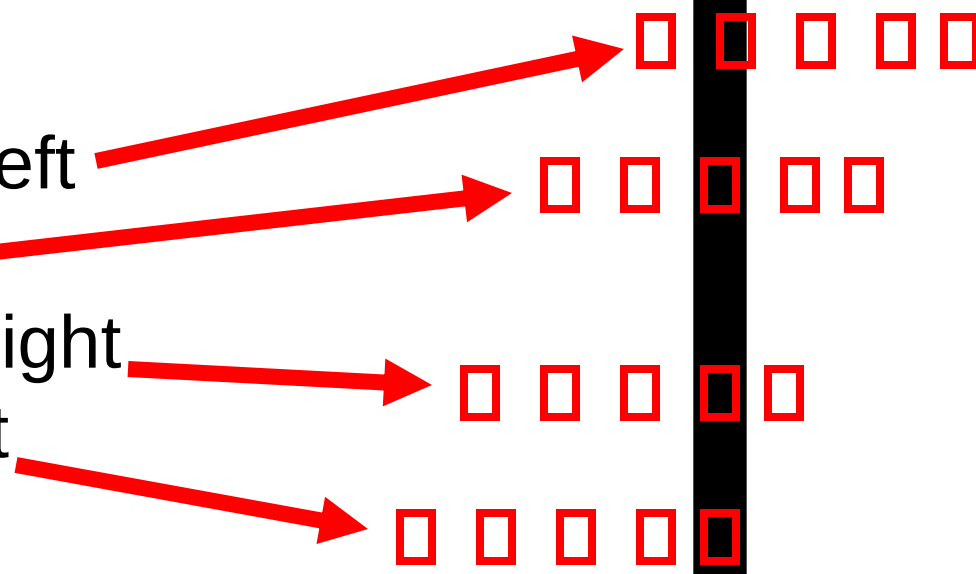
1 0 0 0 0 = Line off to left

0 1 0 0 0 = Line a little to left

0 0 1 0 0 = Dead center!

0 0 0 1 0 = Line a little to right

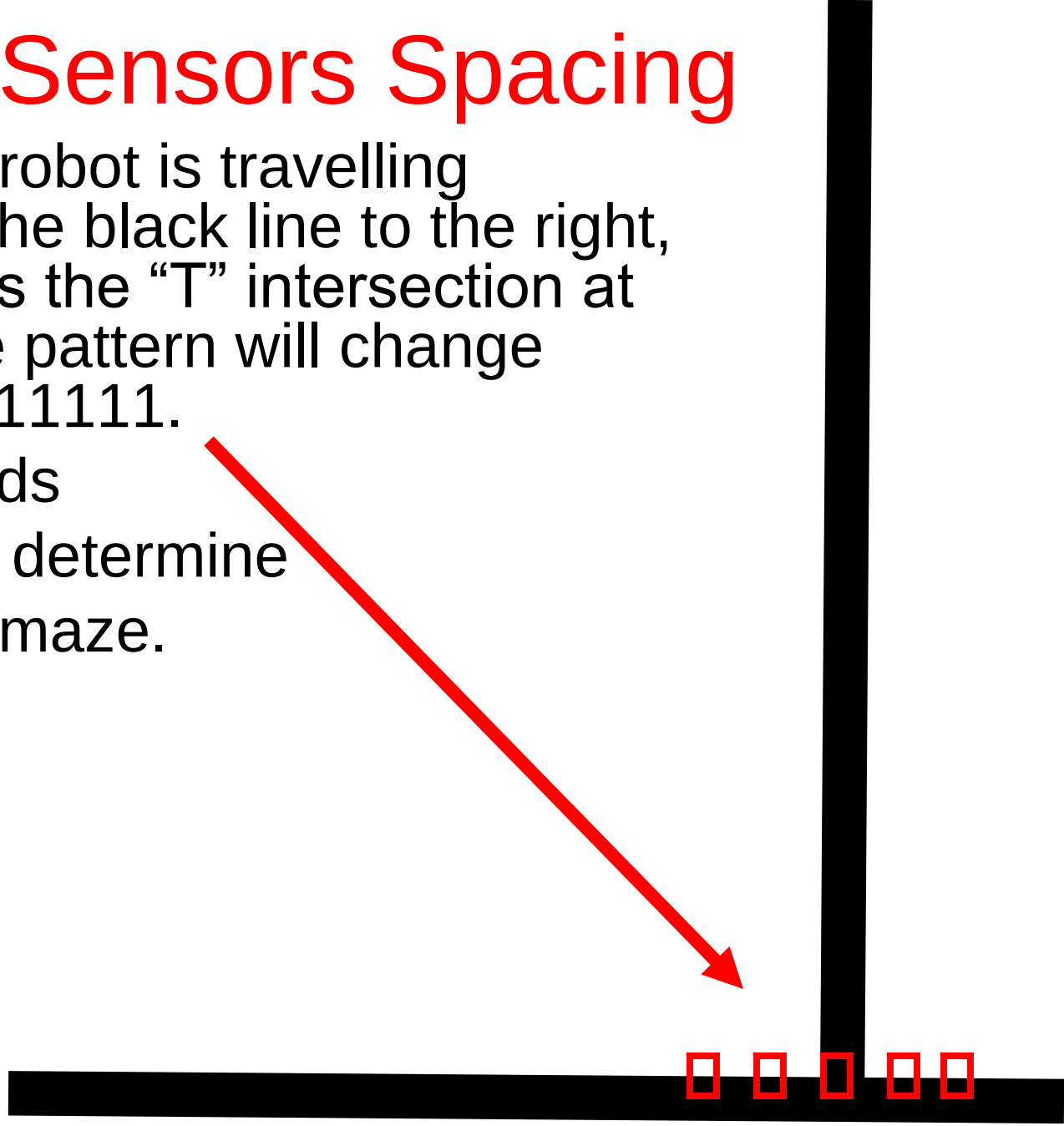
0 0 0 0 1 = Line off to right



Line Sensors Spacing

Notice that if the robot is travelling downward on the black line to the right, when it reaches the “T” intersection at the bottom, the pattern will change from 00100 to 11111.

The program reads these patterns to determine where it is in the maze.

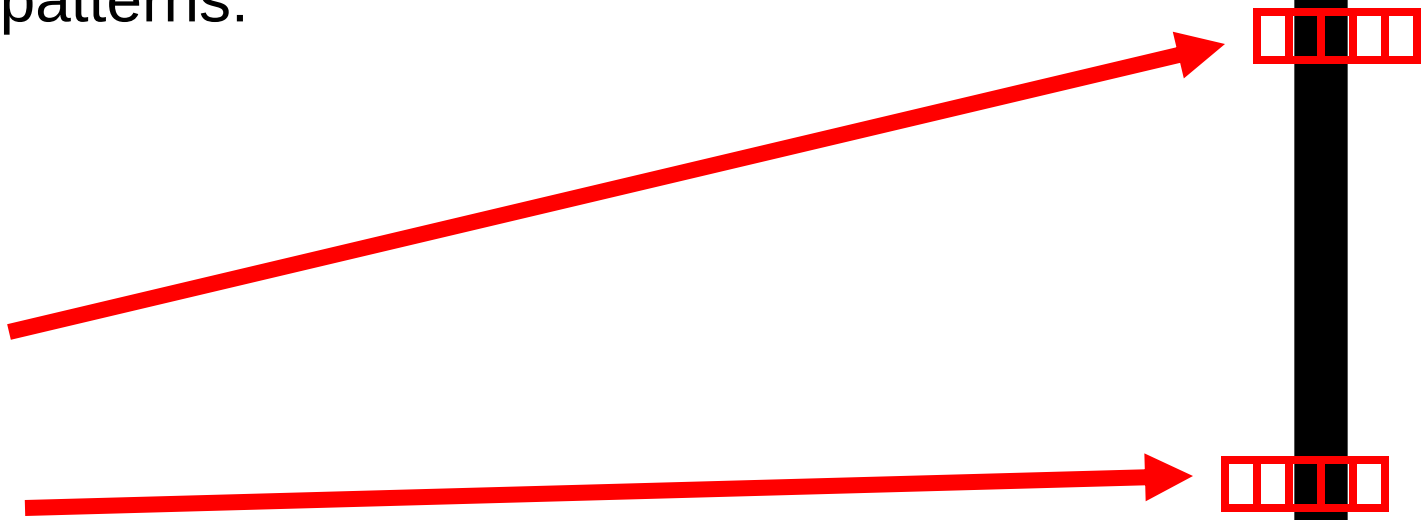


Line Sensors Spacing

If the sensors are spaced closer together so that two sensors can detect the line at the same time, there are many more combinations and you can have more precise control over your robot's path.

Possible patterns:

1 0 0 0 0
1 1 0 0 0
0 1 0 0 0
0 1 1 0 0
0 0 1 0 0
0 0 1 1 0
0 0 0 1 0
0 0 0 1 1
0 0 0 0 1



Five Sensors Closely Spaced

- This project will use five closely spaced sensors.
- These sensors will look directly down on the track and then be read by the program to determine the correct next action.

- Next, let’s see how these sensors can detect the line and intersections.

How Many Combinations?

- With five sensors that can each be a one or a zero, there are 2^5 or 32 possible combinations. We will be walking through many of these, but also be aware some of these are impossible or highly unlikely in the mazes described here. For example, you would not expect to see these particular combinations in a line maze:

  1 0 1 0 1 or 1 1 0 1 1 or 1 0 0 1 1

…and a few others.

How Many Behaviors?

The robot, most of the time, will be involved in one of the following behaviors:

1. Following the line, looking for the next intersection.
2. At an intersection, deciding what type of intersection it is.
3. At an intersection, making a turn.

These steps continue looping over and over until the robot senses the end of the maze.

We will cover these in the next section.

Follow the Line

Following the line is relatively easy. Here is some pseudocode:

```
Select Case Pattern
  Case Pattern = %00100 ‘ Full speed ahead
    leftMotor=fast; rightMotor=fast
  Case Pattern = %01100 ‘Go left a little
    leftMotor=medium; rightMotor=fast
  Case Pattern= %10000 ‘ Way off!
    leftMotor=slow; rightMotor=medium
```

…and so on

Follow the Line

In the code on the previous slide:

leftMotor=fast; rightMotor=fast

leftMotor=medium; rightMotor=fast

leftMotor=slow; rightMotor=medium

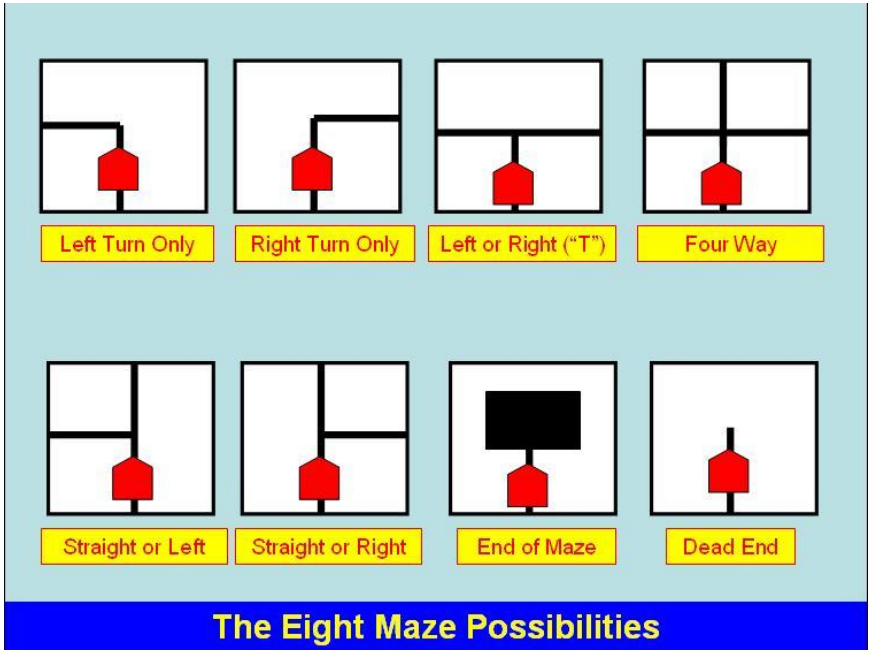
Slow, medium and fast are arbitrary speeds that should get the robot turning or moving in the correct direction when straying from the line.

Experiment with specific speeds until you get smooth line tracking.

Intersection and Turn Handling

The next few slides will walk through how the robot handles coming to turns or intersections.

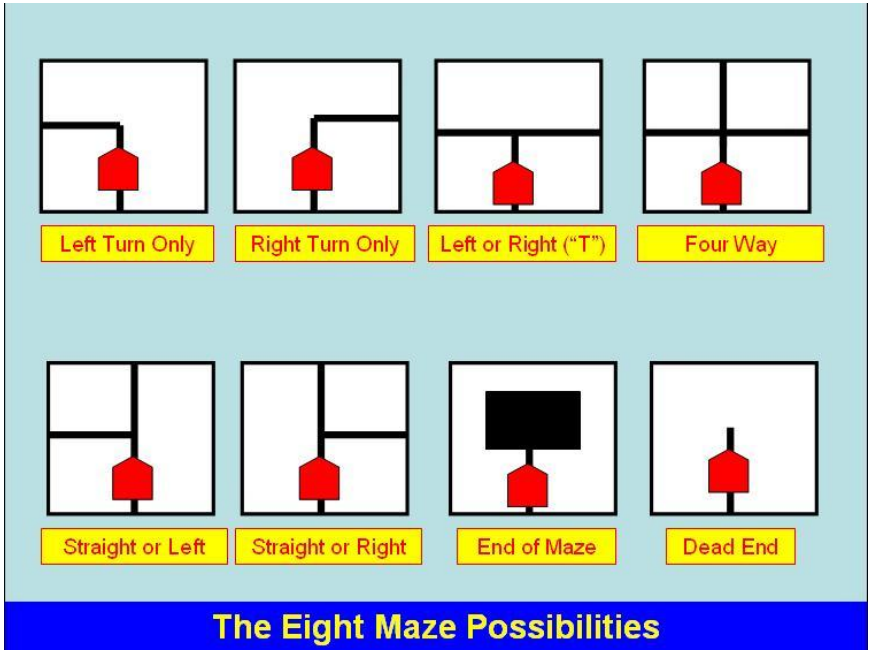
The robot needs to be taught the correct behavior depending on the type of turn or intersection it encounters.



Intersection Handling

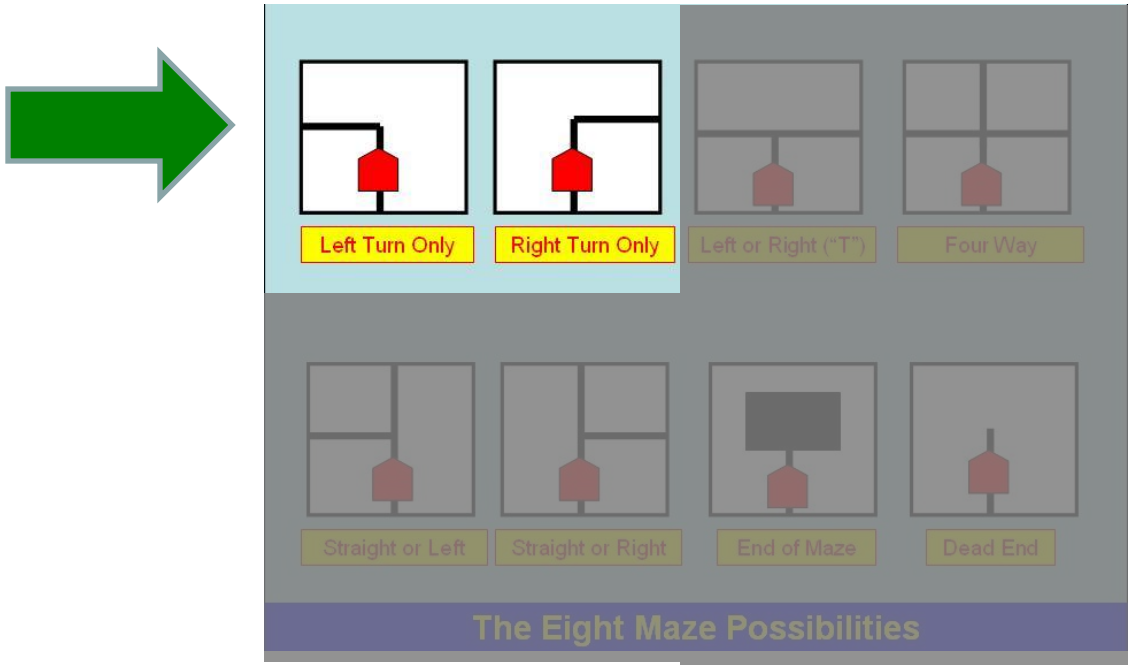
First, I need to give a more rigid definition of an intersection. I will define an intersection as "a place where the robot has more than one choice of direction."

From the illustration to the right, you can see that in some of the eight situations, there is only one possible action. This distinction needs to be made because later some of these will need to be stored by the robot and some won't.



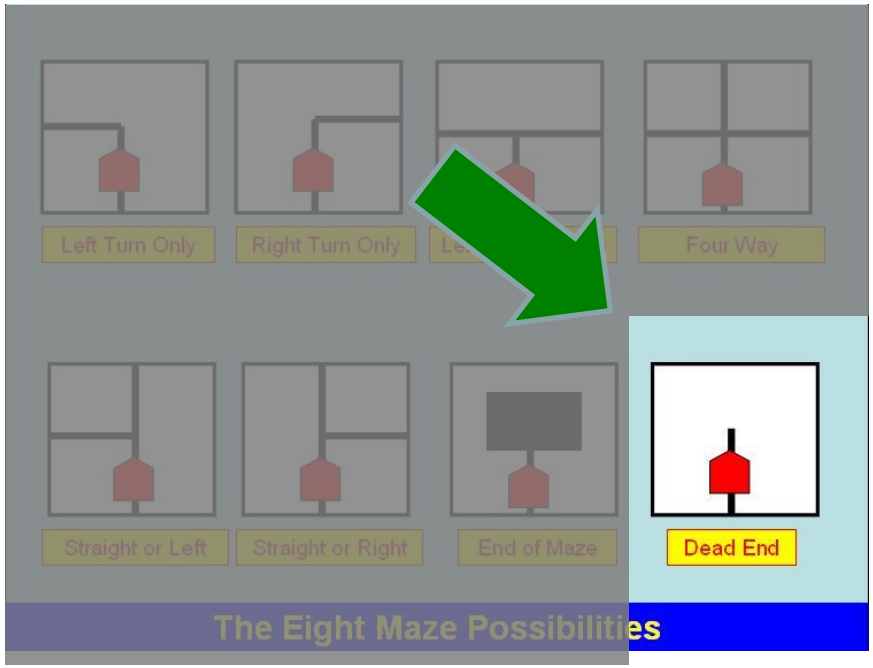
Not Intersections!

In these two cases, the robot has no choice but to make a 90 degree turn. Since the robot will always make the same 90 degree turn and it has no other option, this turn need not be stored when solving the maze.



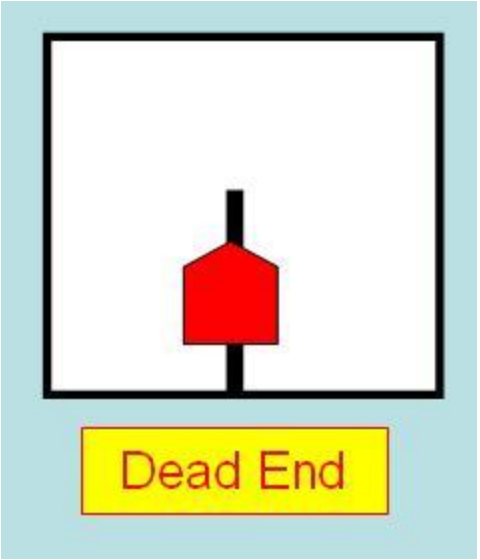
Not Intersections!

In this case, the robot has no choice but to make a 180 degree turn to exit the dead end. Since reaching a dead-end means that the robot has recently made a bad turn, we need to store this fact so that a previous turn can be corrected. The robot should not visit this dead-end on the next run.



The Dead-End

- The Dead-End is the easiest intersection.
- Sensors will go from “00100” to “00000”.
- This is the only normal situation where the robot should see all zeros.

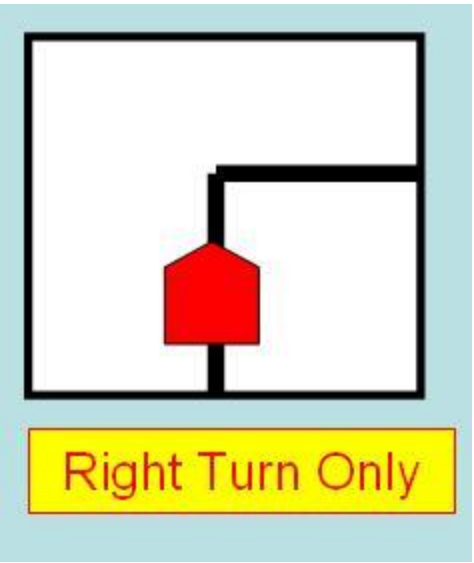


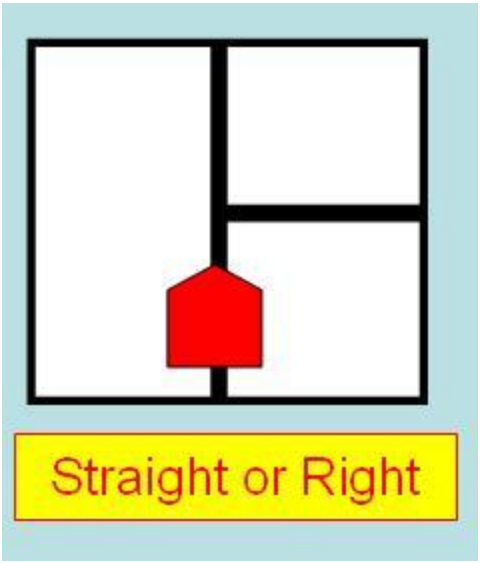
The correct behavior at a dead-end is to make a U-Turn. The pseudocode for this would look something like this:

```
If Pattern = %00000 then gosub U-Turn
```

“Right Only” or “Straight or Right”

- Here is a situation where the same pattern will initially appear for two different situations.



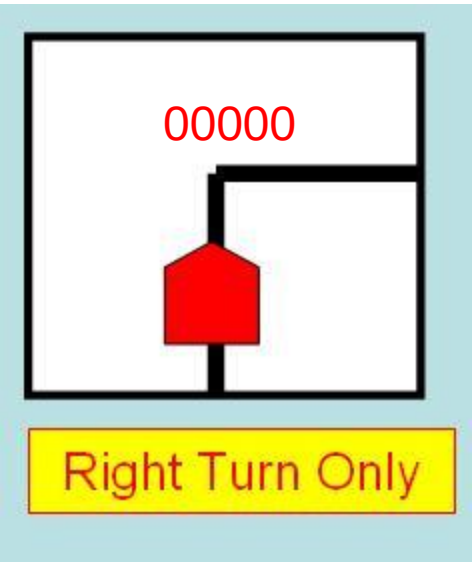


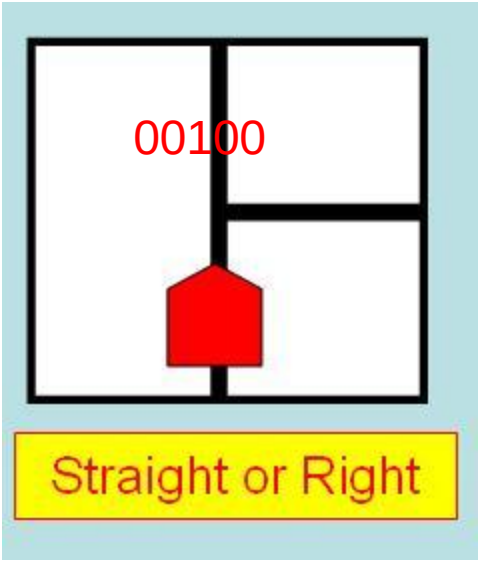
Both show the pattern “00111” when initially detected.

How do we know which is which?

Move Forward One Inch

- Create a subroutine called inch() that moves the robot forward one inch.
- Now read the sensors again!

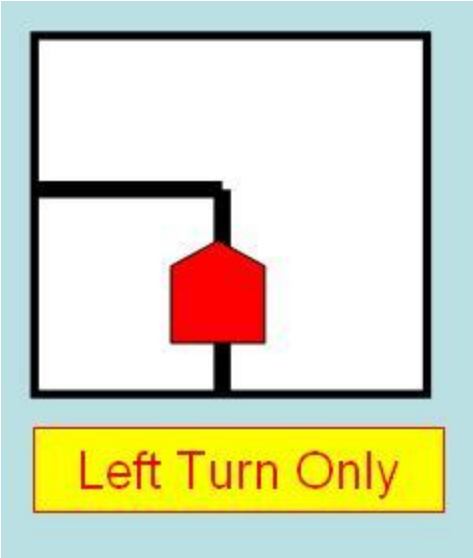


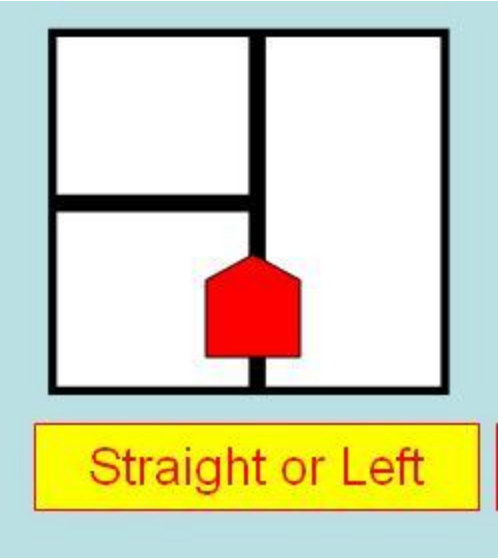


If the sensor pattern is now “00000”, then the robot is at a “Right Turn Only”. With any other reading, the robot is at the “Straight or Right” intersection.

“Left Only” or “Straight or Left”

- Here is a similar situation where the same pattern will initially appear for two different situations.



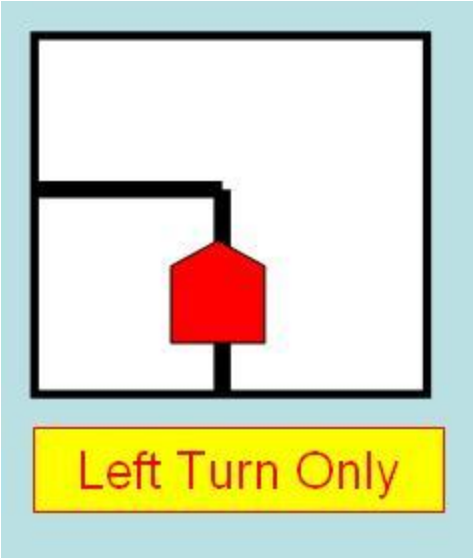


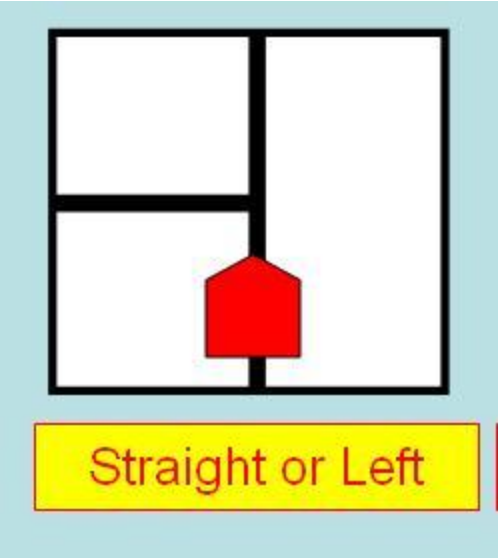
Both show the pattern “11100” when initially detected.

How do we know which is which?

Move Forward One Inch

- Call the subroutine inch() that moves the robot forward one inch.
- Now read the sensors again!

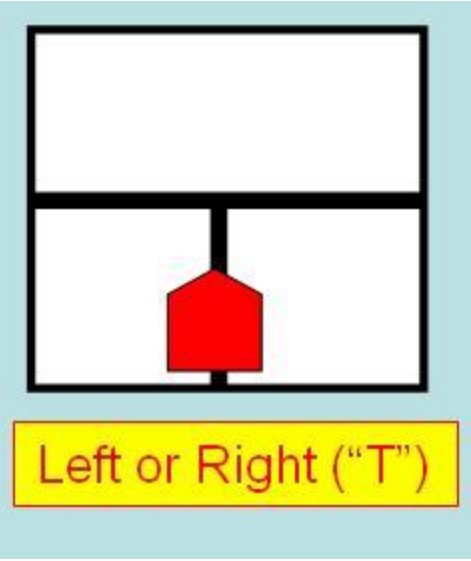


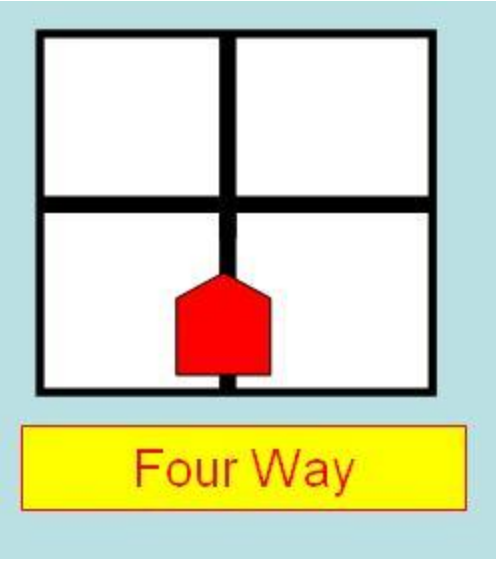


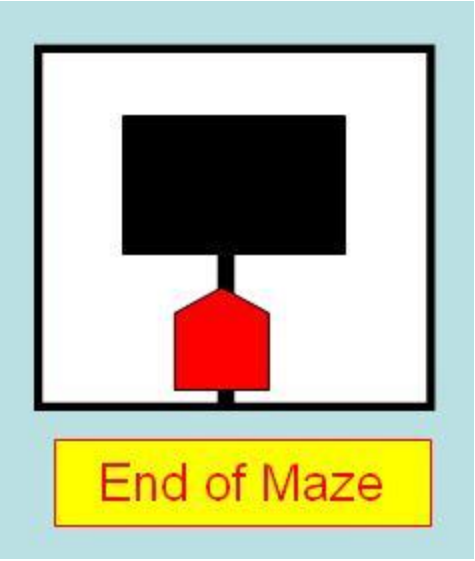
If the sensor pattern is now “00000”, then the robot is at a “Left Turn Only”. With any other reading, the robot is at the “Straight or Left” intersection.

What About These??

- And ALL of the last three intersection types will present an initial pattern of “11111” to the robot.
- Once again, “inch()” will help us determine the correct intersection.

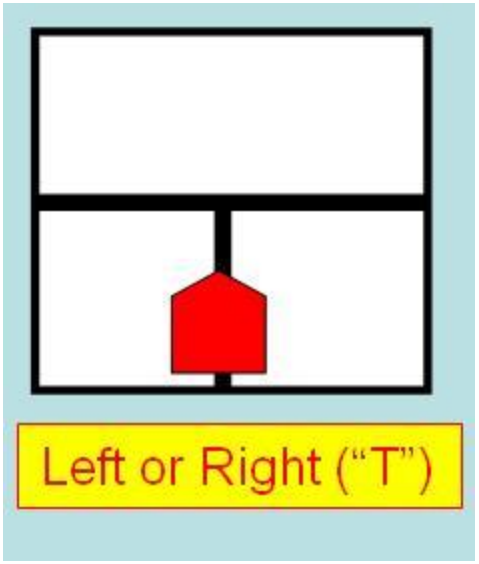


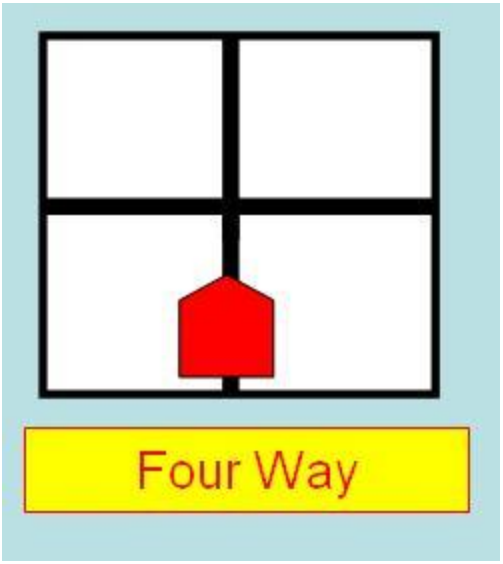


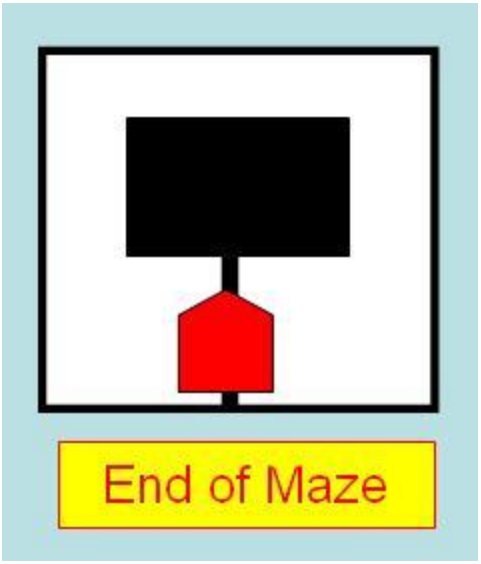


What About These??

```
Gosub inch()
ReadSensors()
If Pattern = "00000" then
   'Found the T intersection
Elseif Pattern = "11111" then
   'Found the end of the maze - STOP!
Else
   'At a four-way intersection
Endif
```



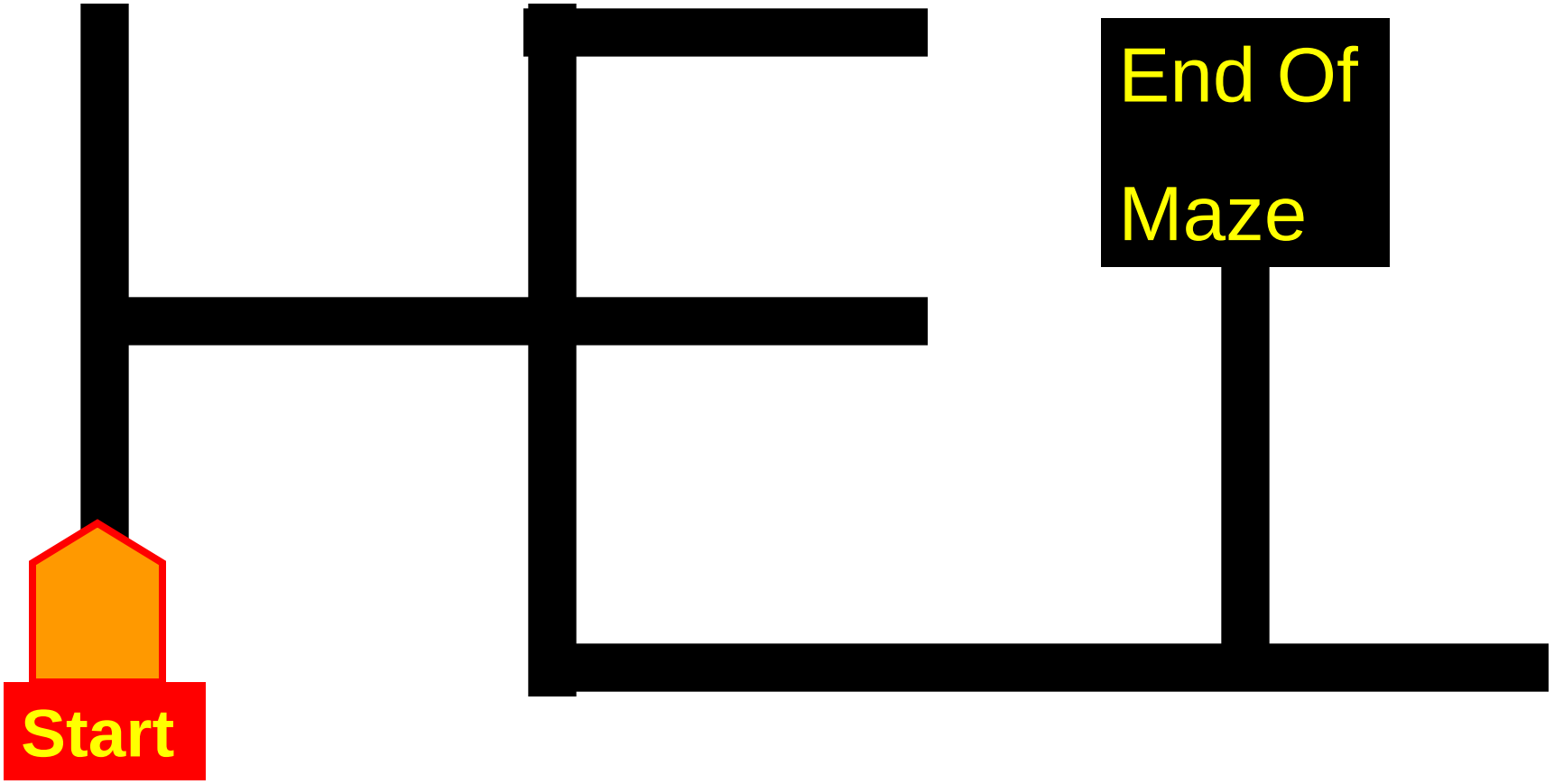




The Main Algorithm

- In order to solve the maze, the robot needs to traverse the maze twice.
- In the first run, it goes down some number of dead-ends, but records these as “bad” paths so that they can be avoided on the second run.
- Next, we will learn how this algorithm works.

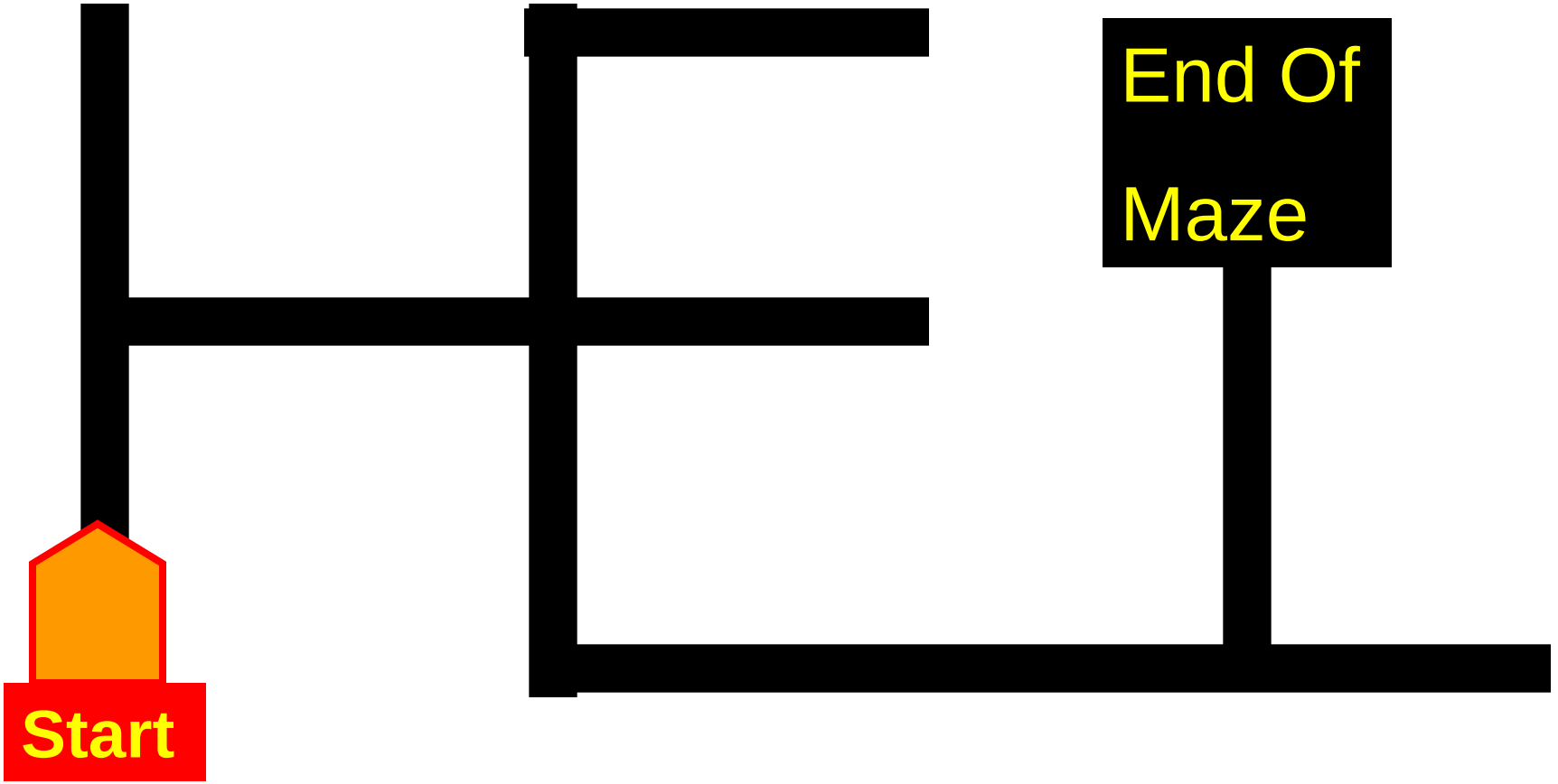
Path Stored in Memory: None



First, we start with a very simple maze.

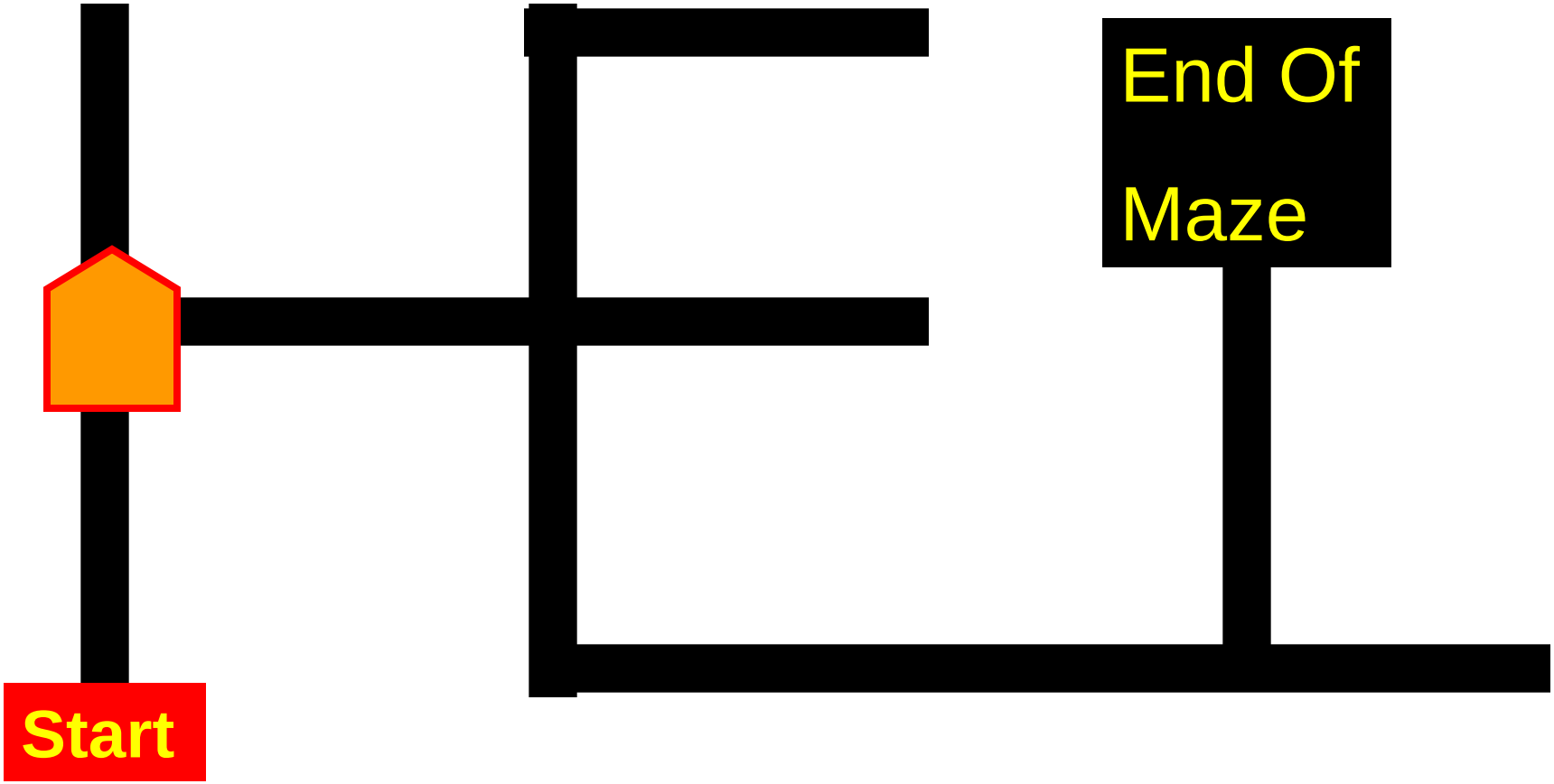
We will walk through this maze step-by-step to explain the maze solving algorithm.

Path Stored in Memory: None



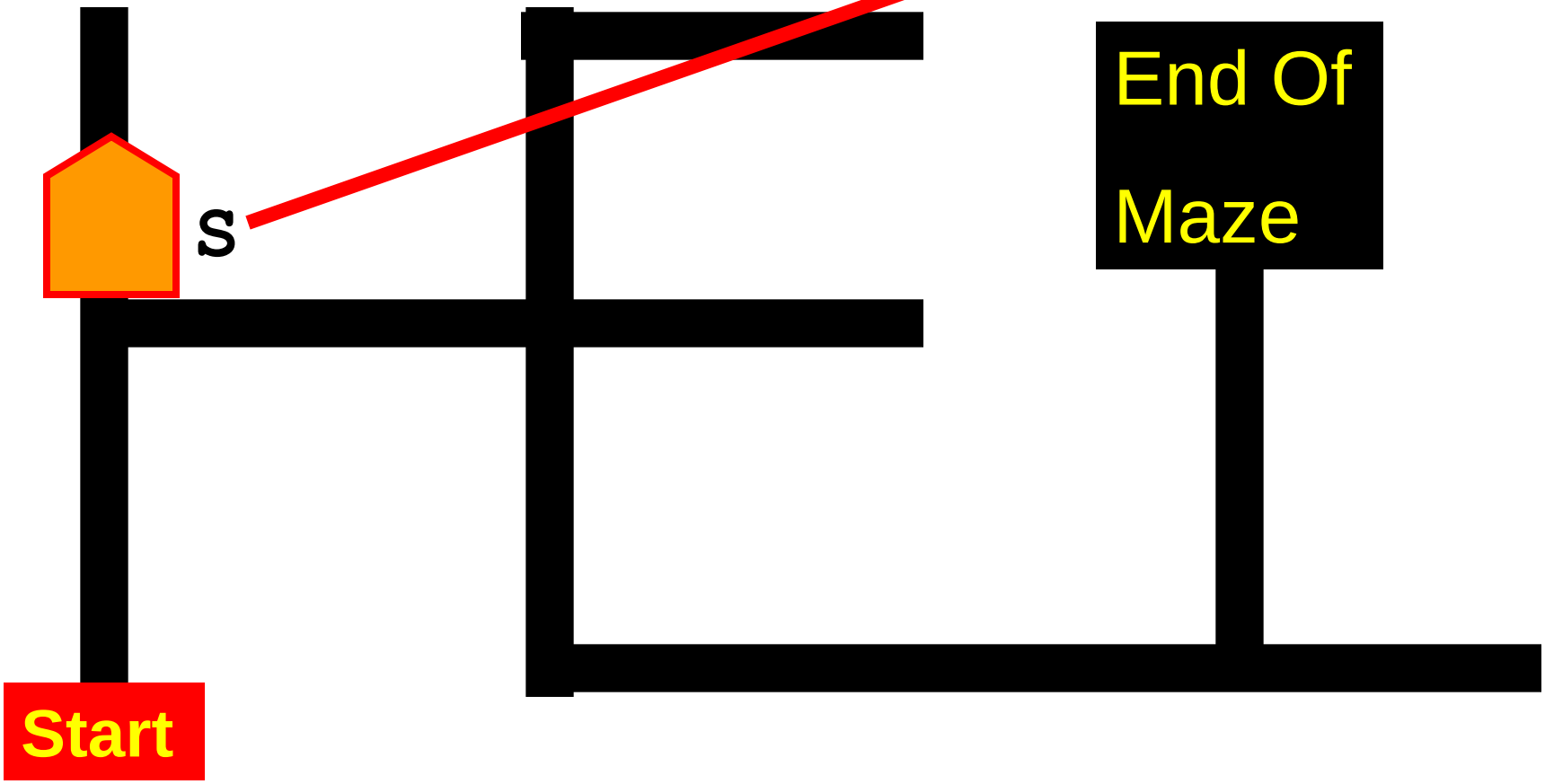
So, we will use the left hand rule for the first pass through the maze. There will be several dead-ends encountered and we need to store what we did so the correct path can be computed.

Path Stored in Memory: None



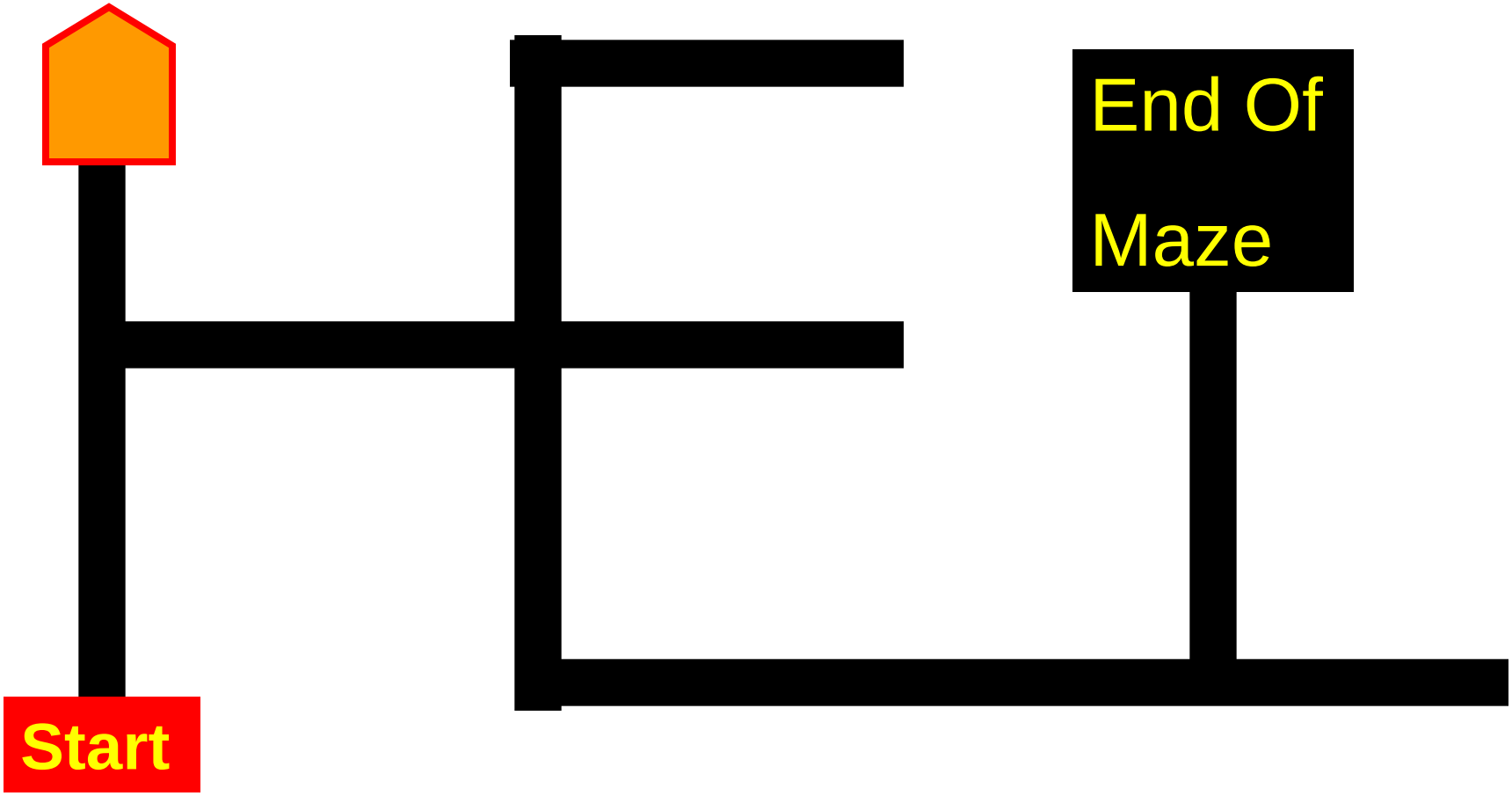
The robot takes off! At the first intersection, the robot senses a “Straight or Right” intersection. The left hand rule requires that the robot go straight.

Path Stored in Memory: S

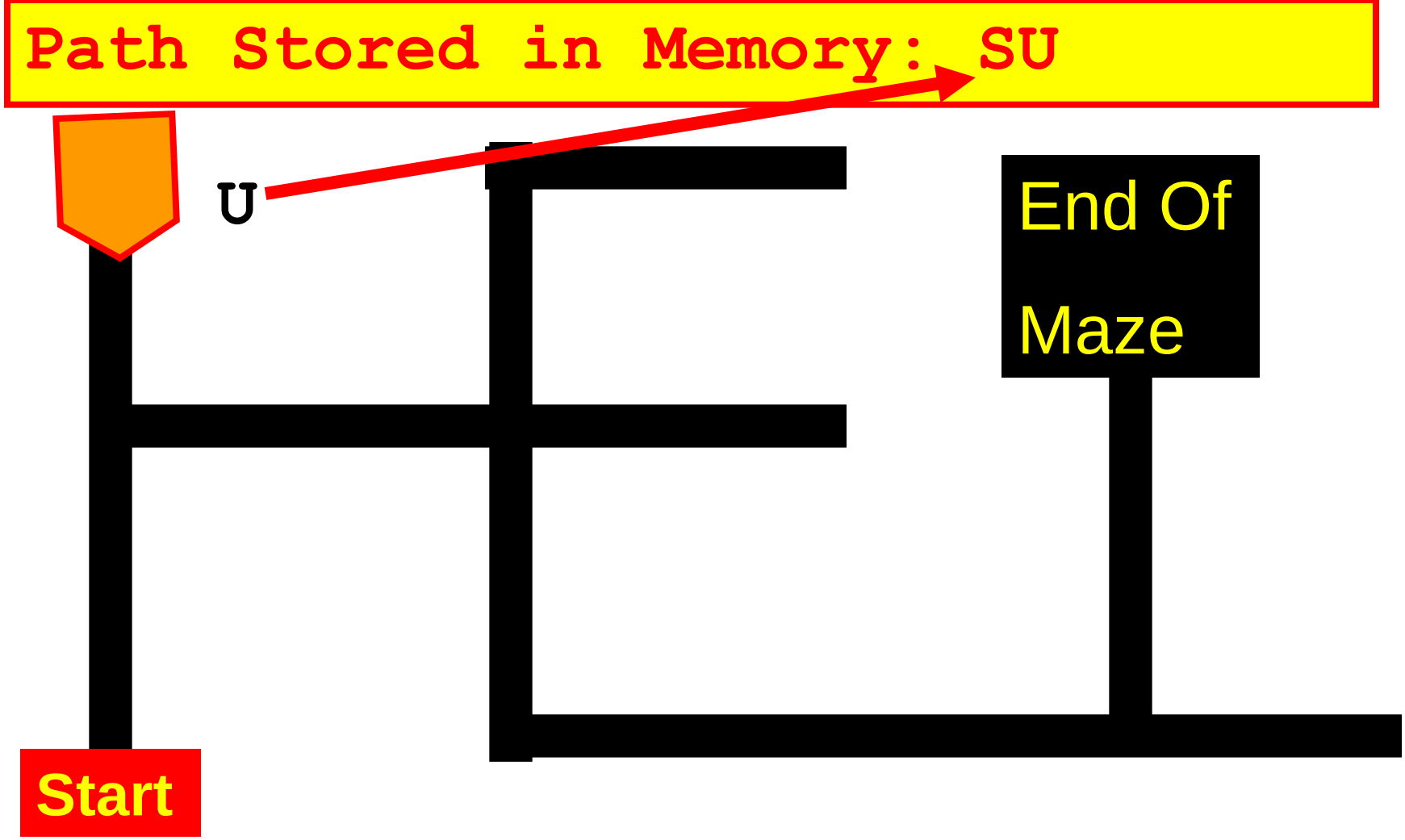


The robot continues straight an stores the turn taken "S" in memory.

Path Stored in Memory: S

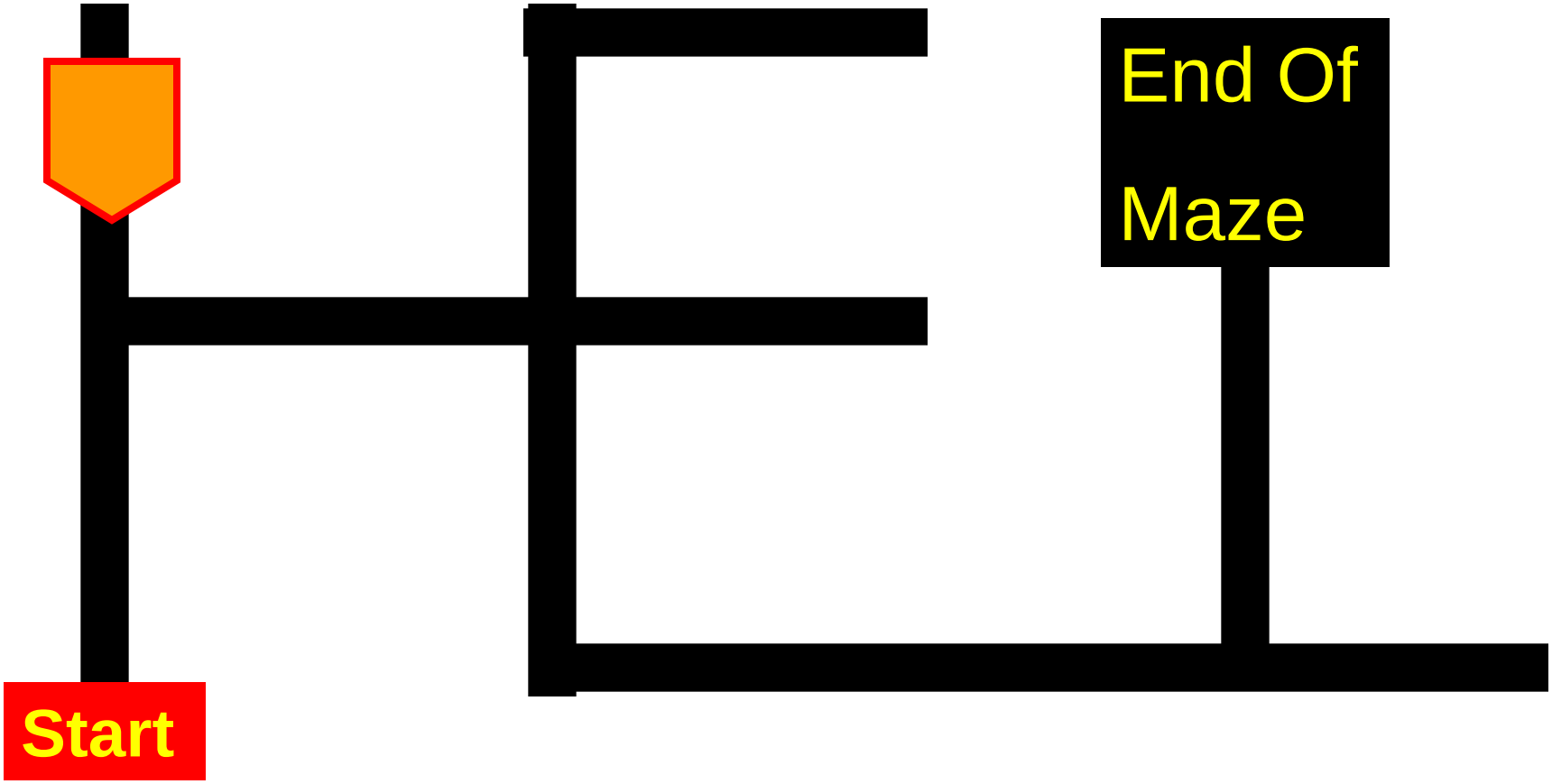


The robot runs off the path (Pattern = “00000”) and the correct behavior is to take a U-Turn.



After the U-Turn, record the turn taken “U” in memory.

Path Stored in Memory: SU



Let's stop here for a moment and discuss an important point…

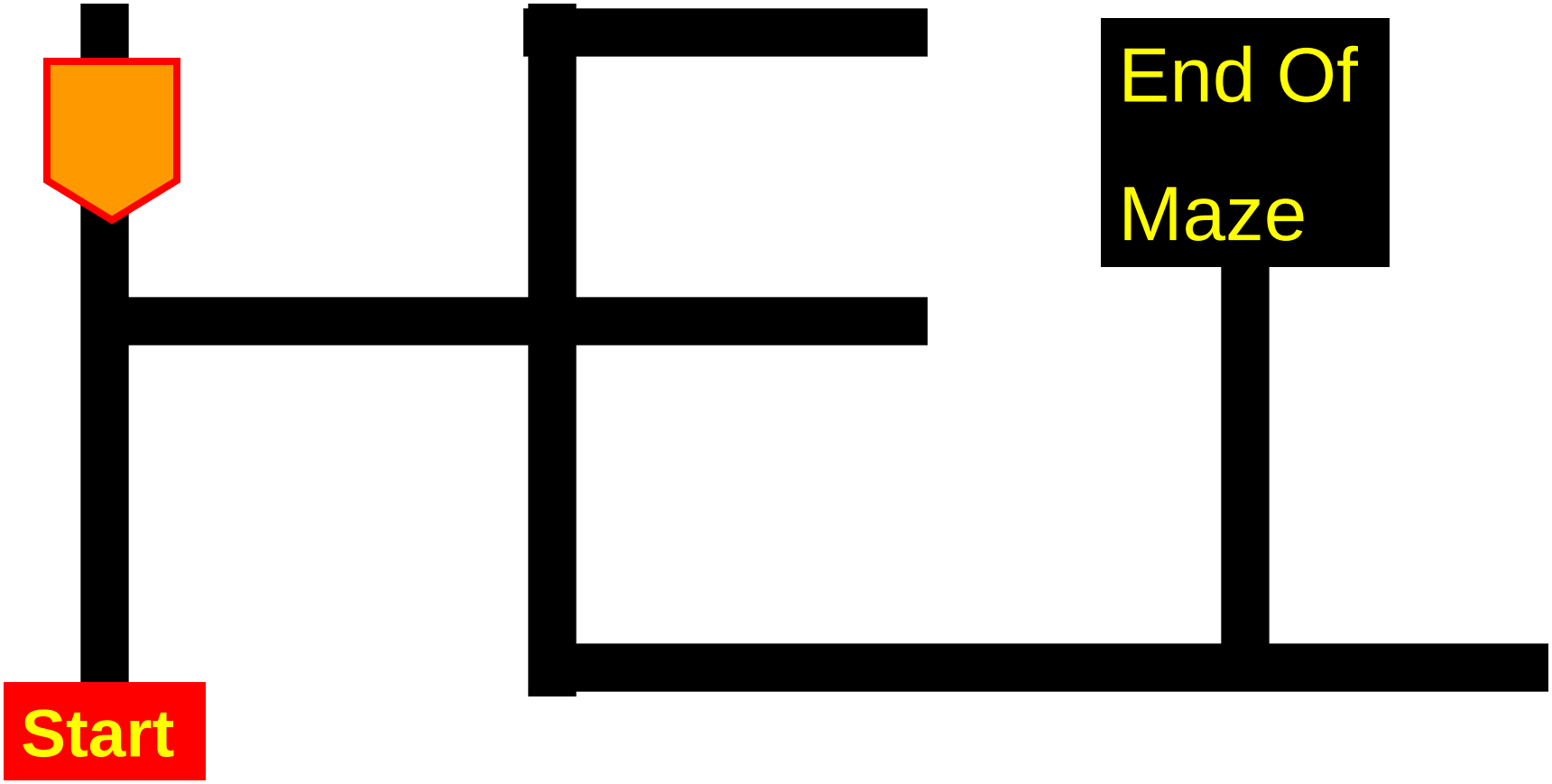
Dead Ends

- In this maze, every dead-end encountered means that the robot has taken a wrong turn!
- In any maze, the best/shortest path through it *never* includes going down a dead-end path.
- So a dead-end can always be said to tell us: The previous action/turn was not correct and needs to be changed.

Dead Ends

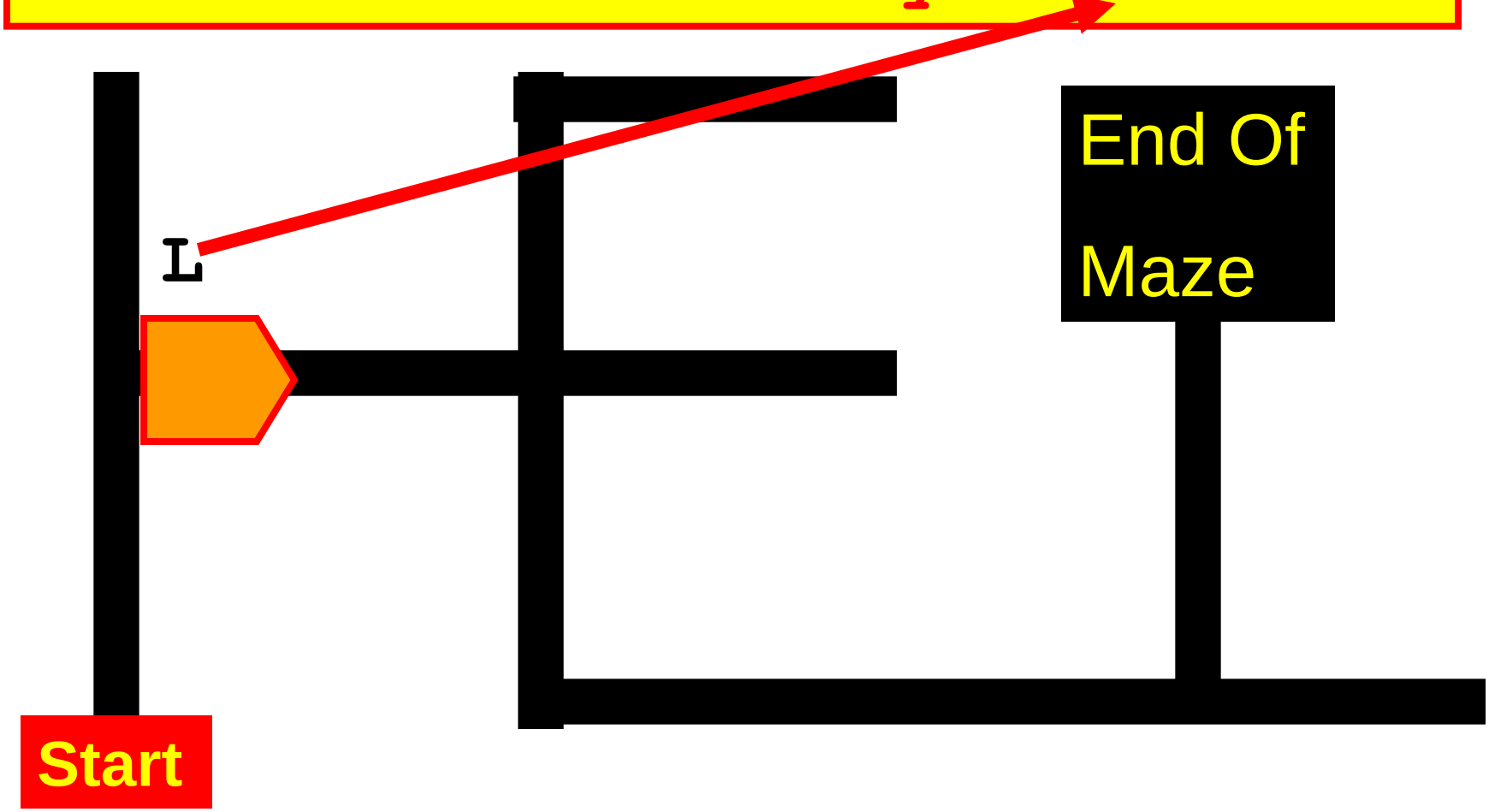
- We can’t be sure how we got here or the correct way to modify the previous turn until we return to the previous intersection.
- Let’s continue with the example to develop the algorithm…

Path Stored in Memory: SU



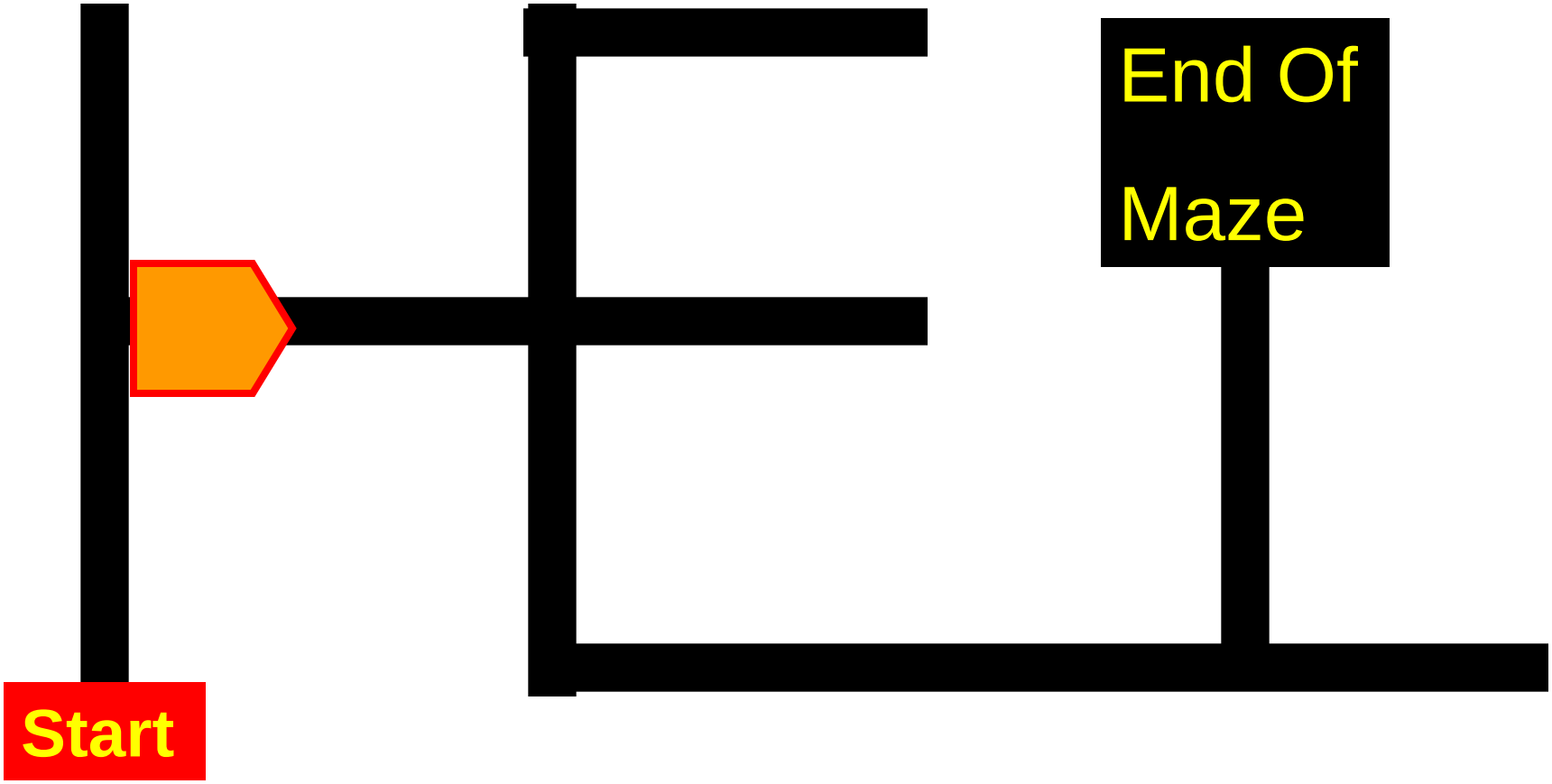
When the robot gets back to the first intersection, it will go left, so…

Path Stored in Memory: SUL



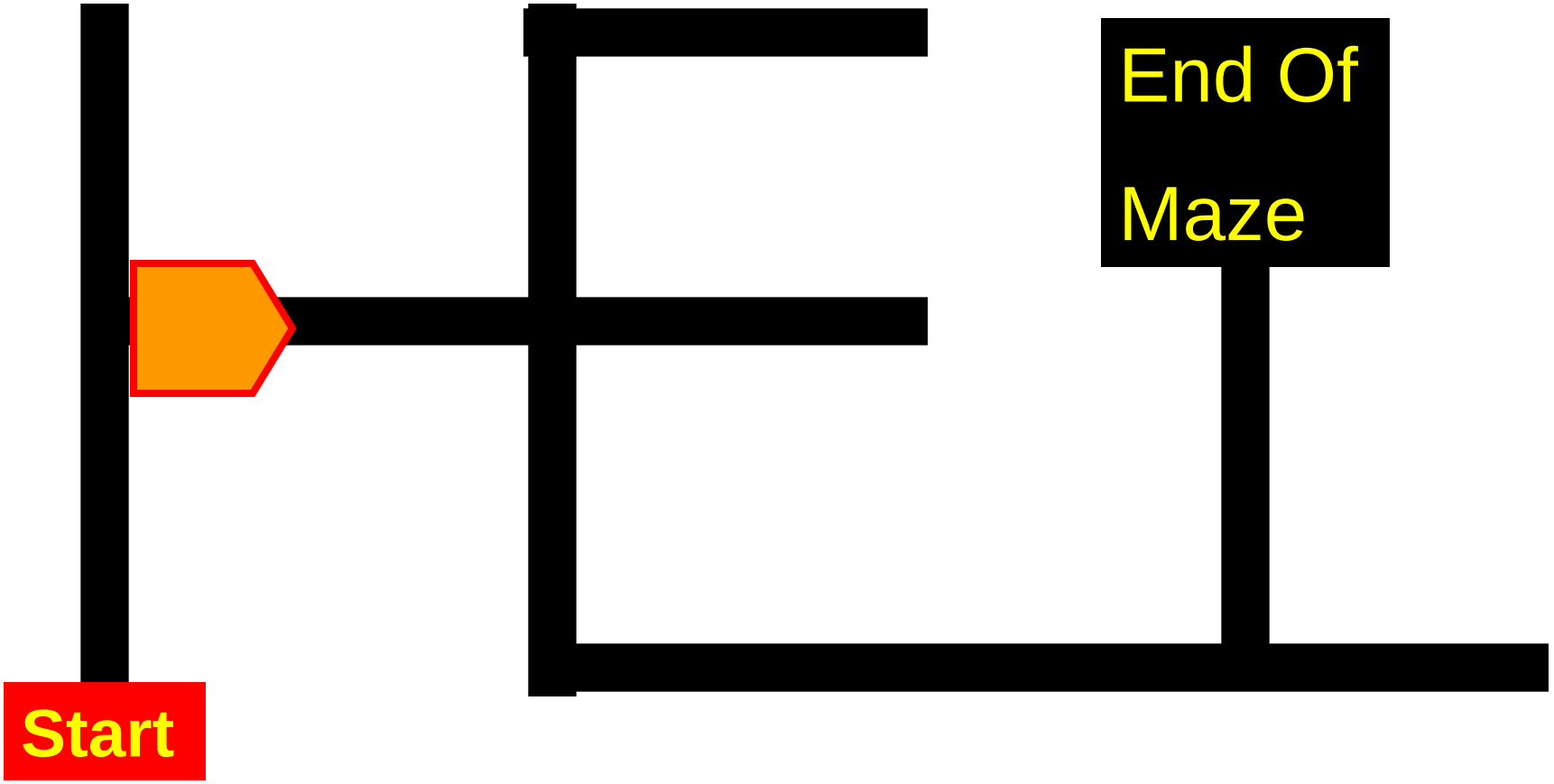
The left hand rule calls for a left turn, so take a left and record the turn.

Path Stored in Memory: SUL



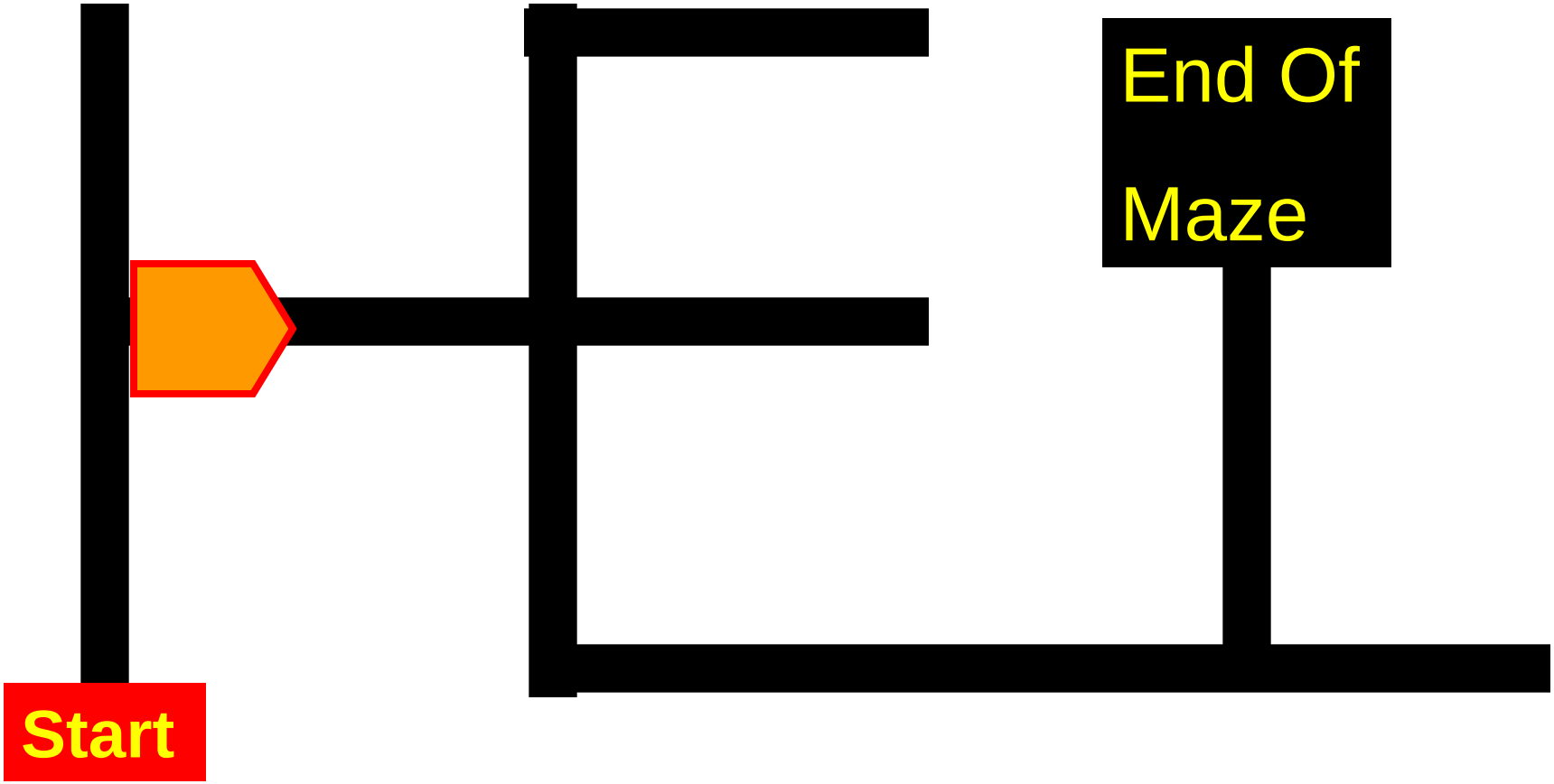
We recorded "SUL", but we know (because of the "U") that we should have not gone down that path.

Path Stored in Memory: SUL



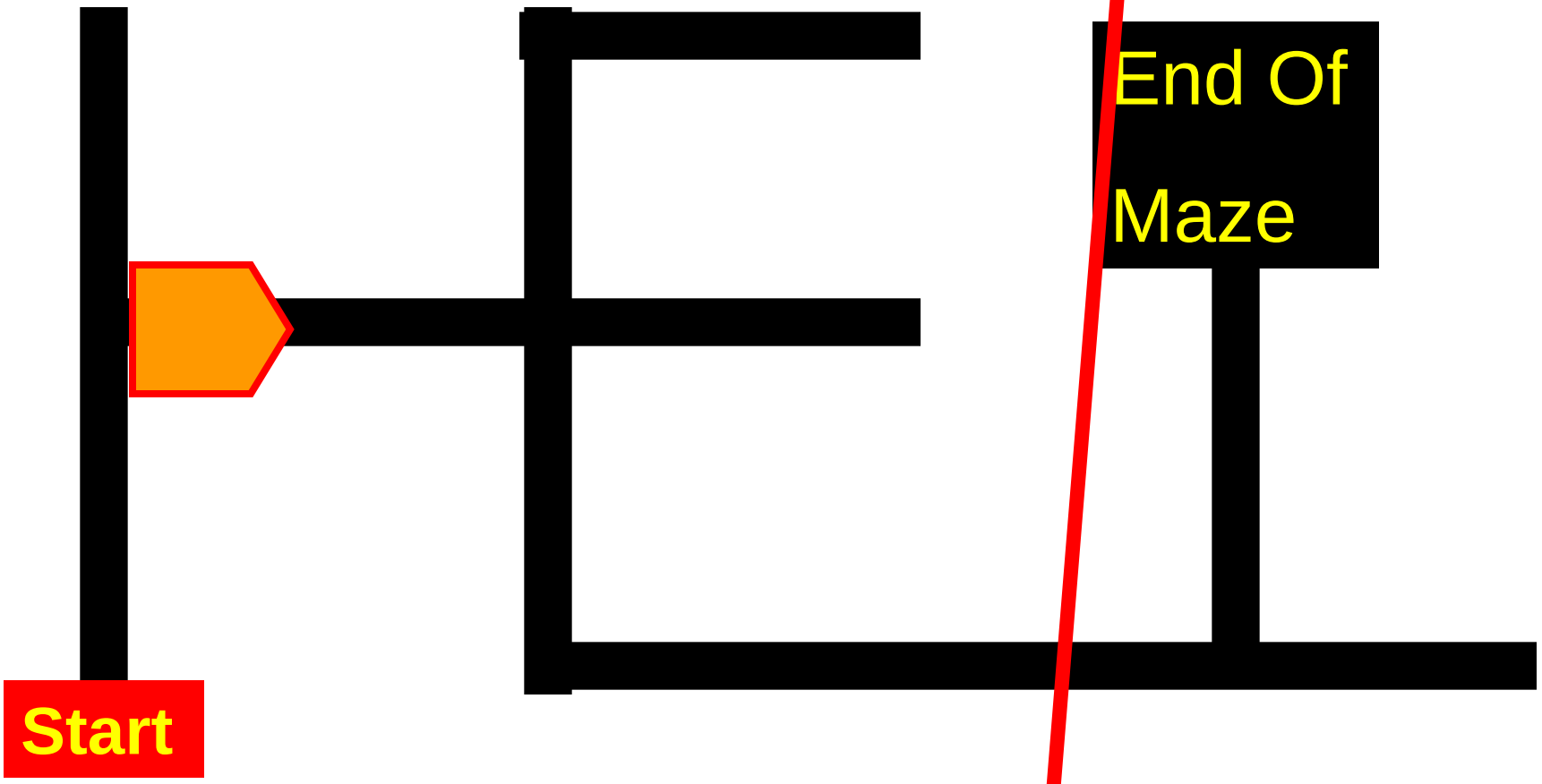
What could the robot have done to avoid going down the dead-end path? Instead of "SUL", the robot should have done "R".

Path Stored in Memory: SUL



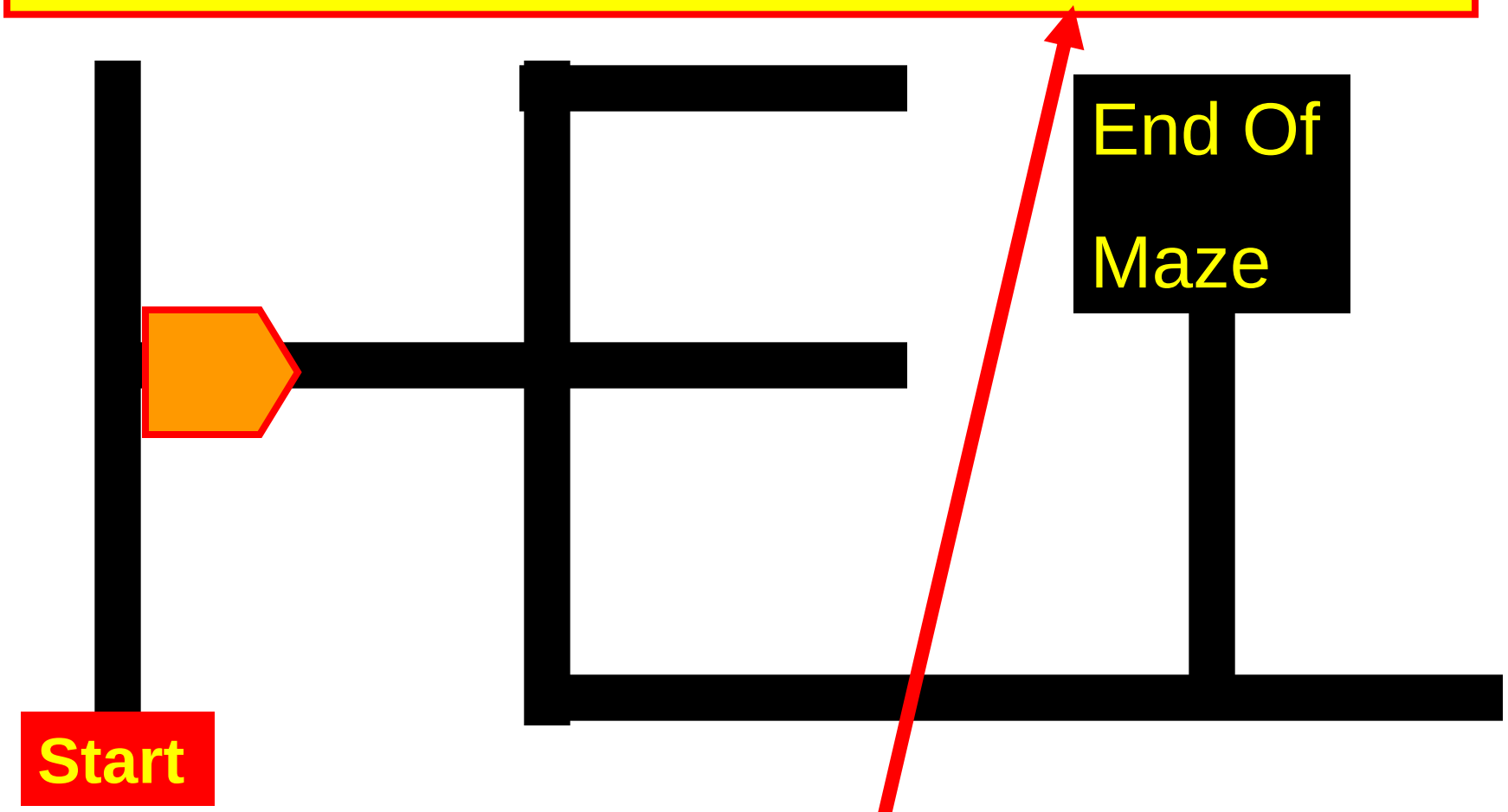
So, our first general rule is: Anywhere the stored path ends with “Straight-U-Left” (SUL), we replace it with “Right” (R).

Path Stored in Memory: SUL



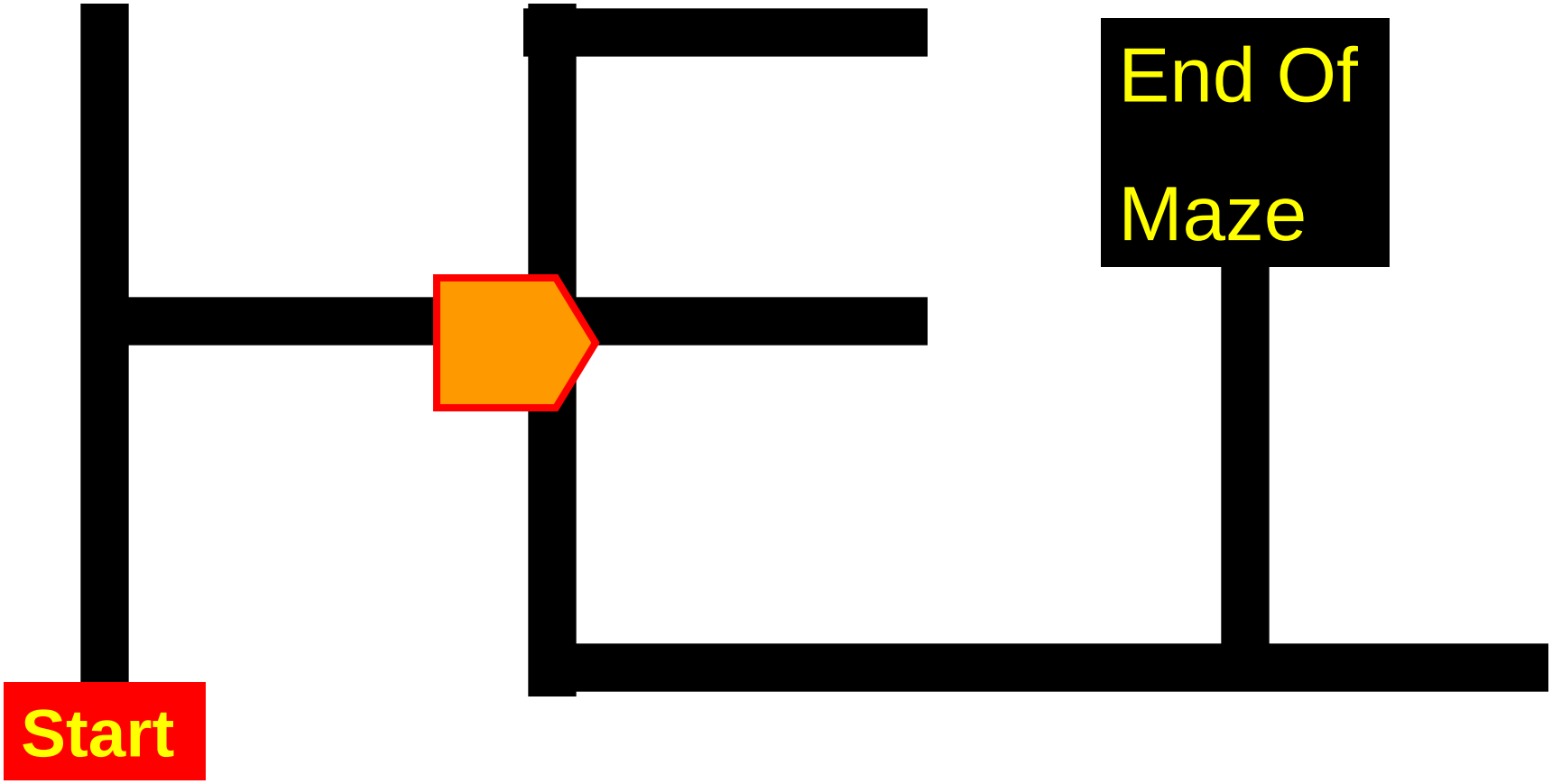
So, delete the "SUL" in memory..

Path Stored in Memory: R



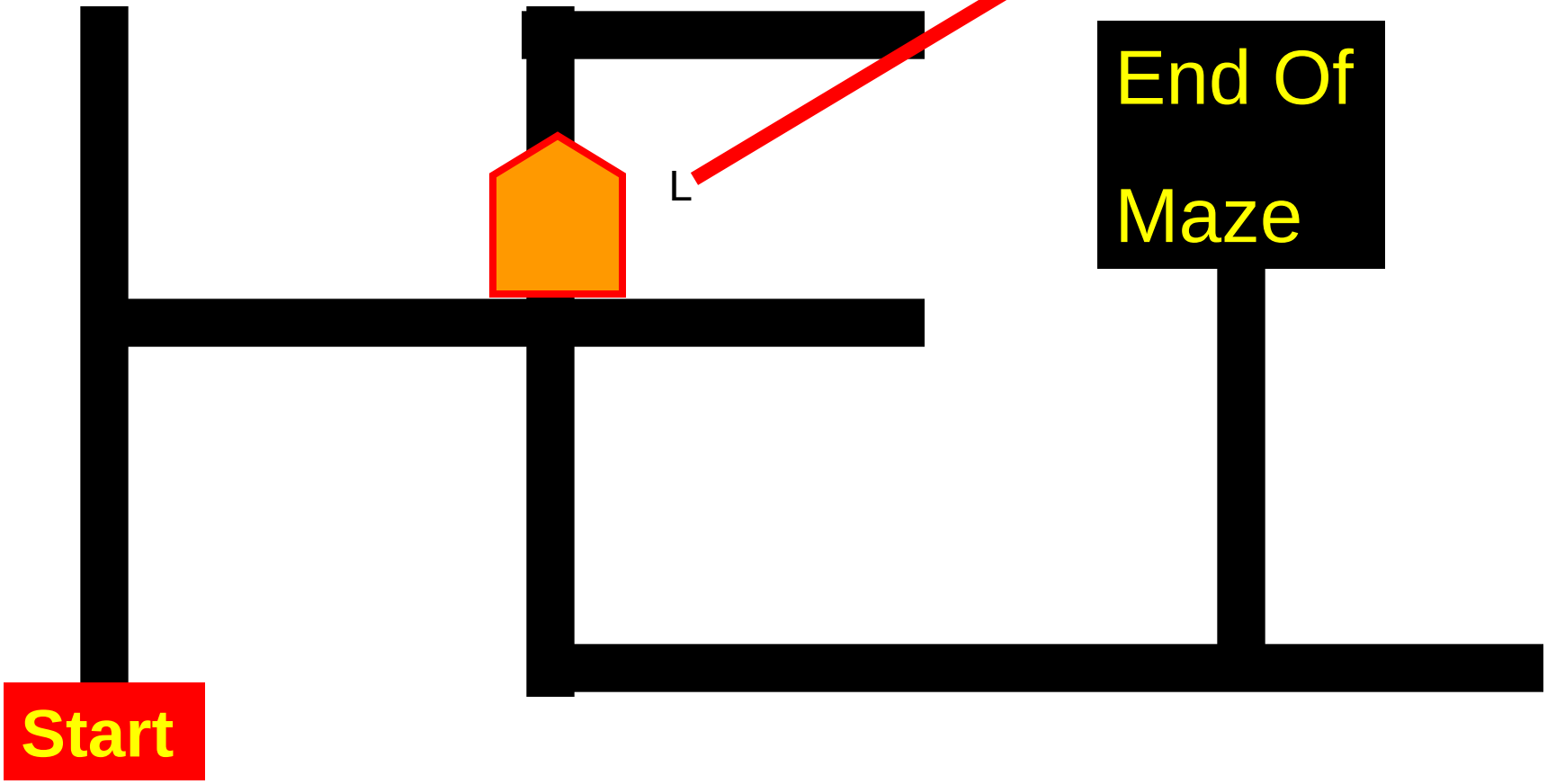
And replace "SUL" with "R". This tells the robot, next time through the maze, to take a right turn at the first intersection.

Path Stored in Memory: R



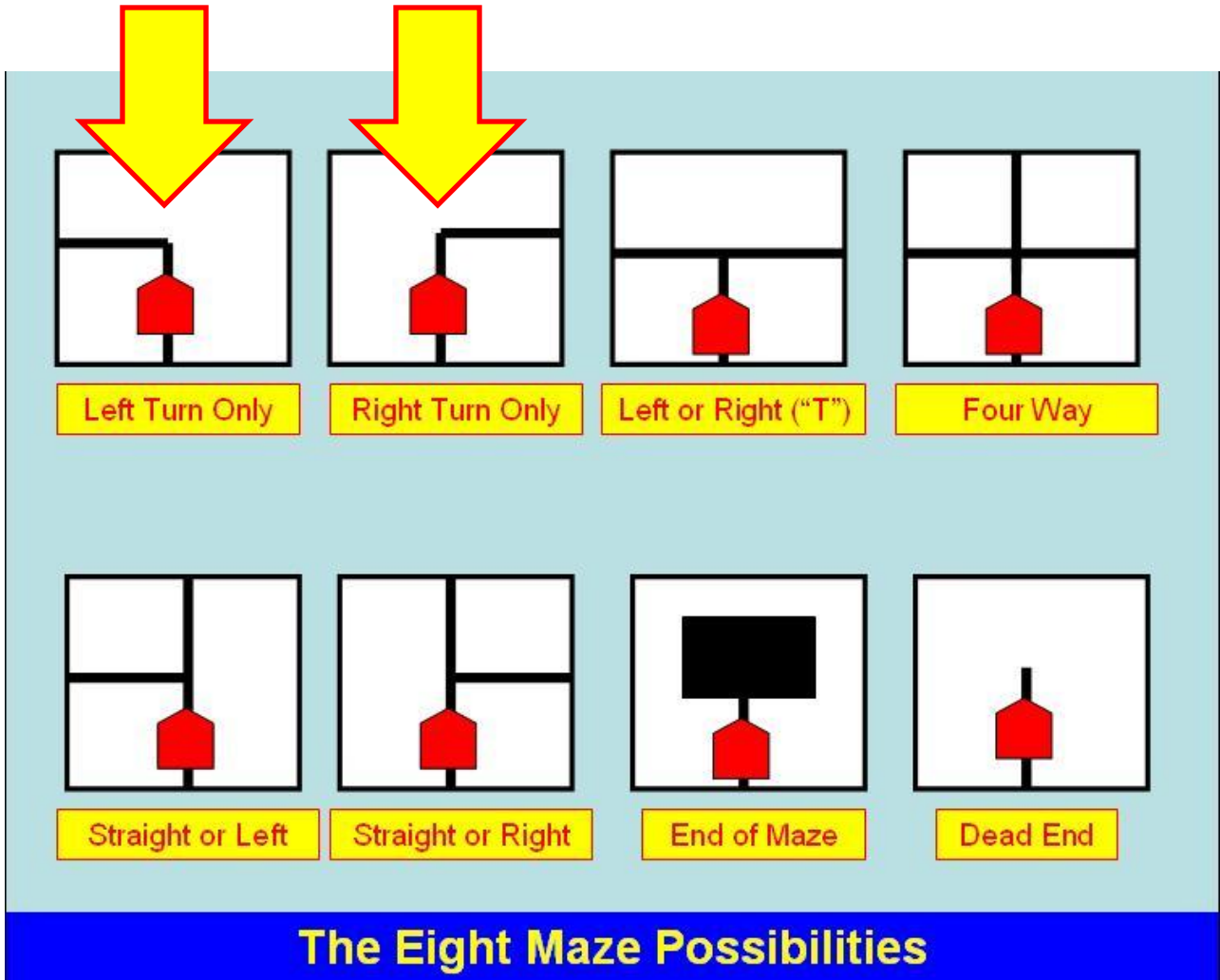
The next intersection is a “Four-Way” Intersection. The left hand rule requires a left hand turn, so turn left and record an “L”.

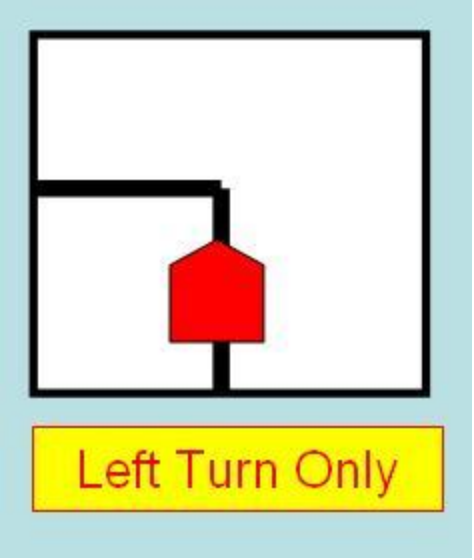
Path Stored in Memory: RL



The intersection coming up, the “Right-Only” has a special characteristic that needs explaining, so we will detour for a bit…

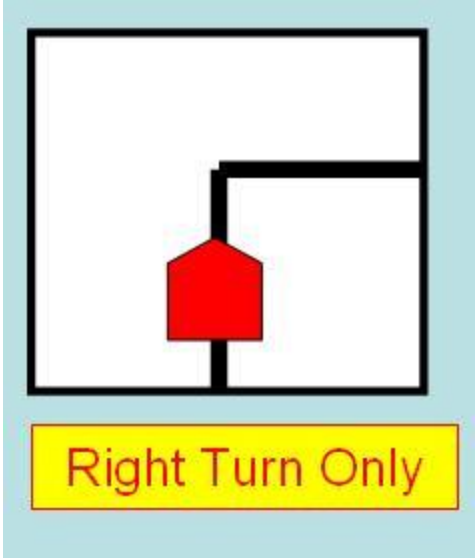
I showed this graphic earlier and I call it the “Eight Maze Possibilities”. I specifically avoided using the term “Eight Maze Intersections”, because two of them are not intersections.





Not Intersections!!

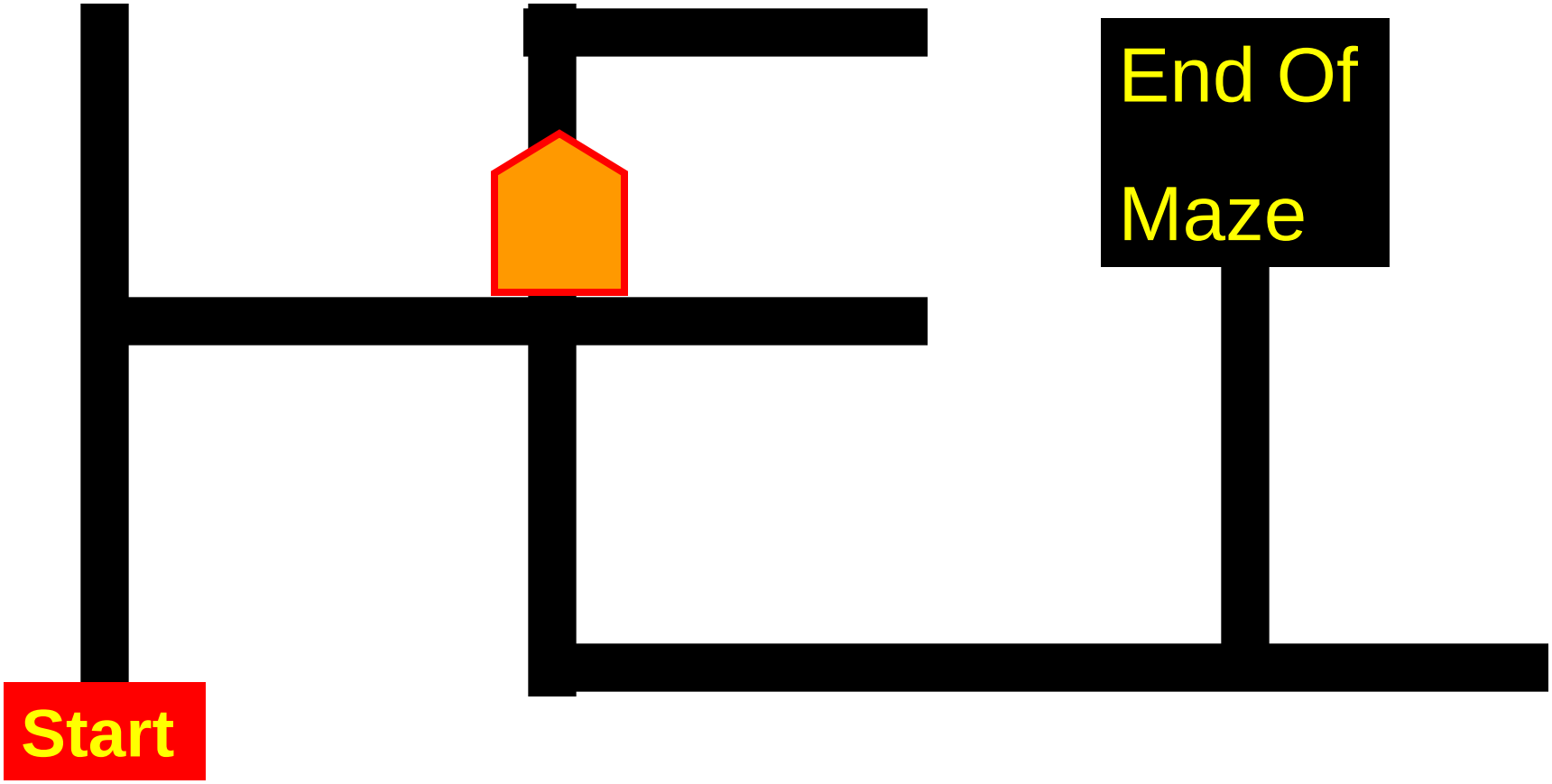
NEW DEFINITION!
Intersection: A point in the maze where the robot has more than one choice of directions to proceed.



Notice the new, restricted, definition of intersection above. What this narrower definition does is REMOVE the “Left Turn Only”, “Right Turn Only” and the “Dead-End” from the intersection list.

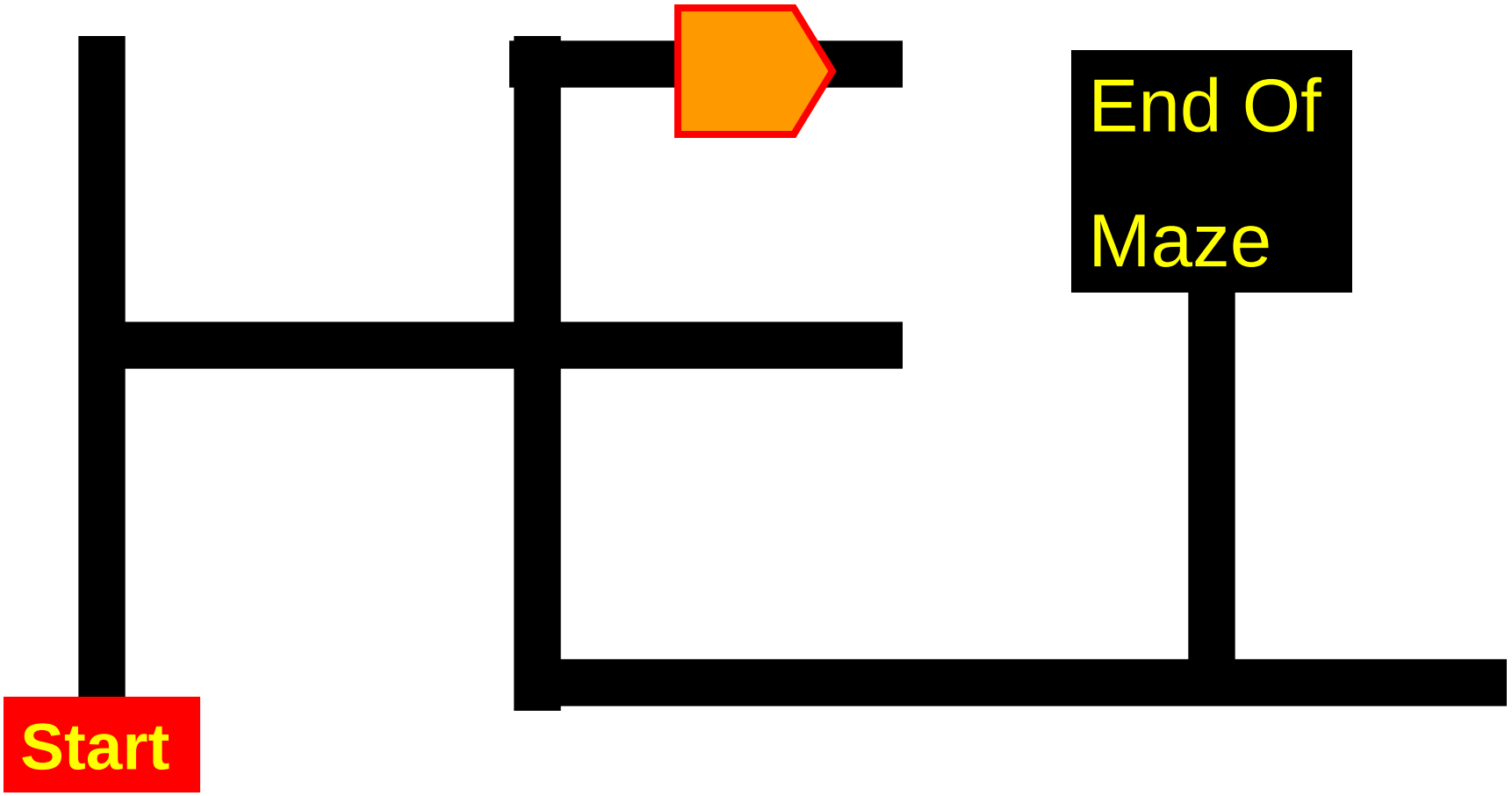
WHY?
Because the whole point of the maze algorithm is to eliminate bad turns. These three “turns” have no choice, so a “bad” turn cannot be made. Since the robot can never go any direction but one, there is not even a need to record this turn in the maze path. So, in the maze, these two will be treated as non-existent. The robot will take the correct turn as indicated by the left hand rule, but will never record either of these two turns.

Path Stored in Memory: RL



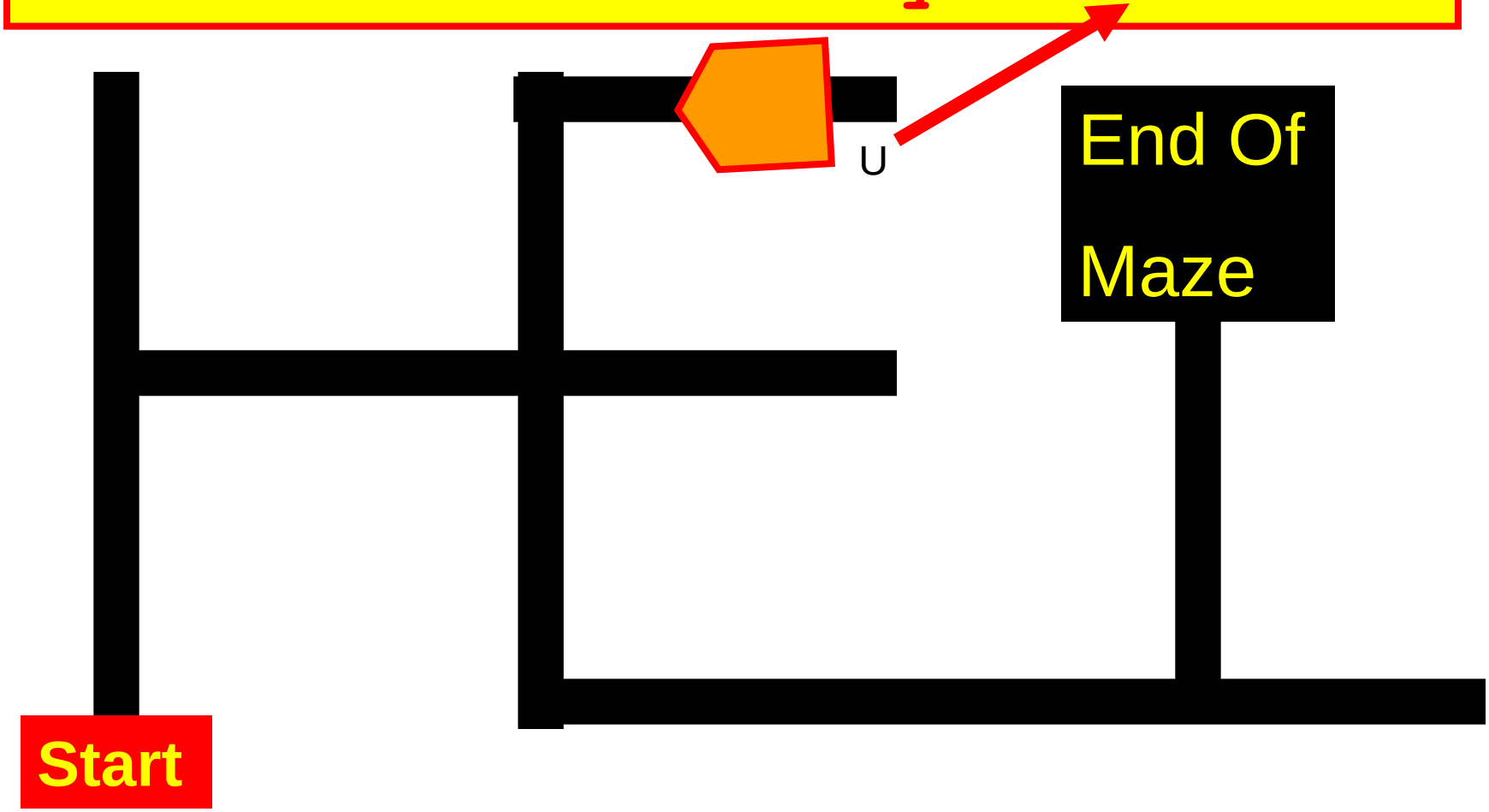
So, the robot will turn right and record nothing in memory.

Path Stored in Memory: RL



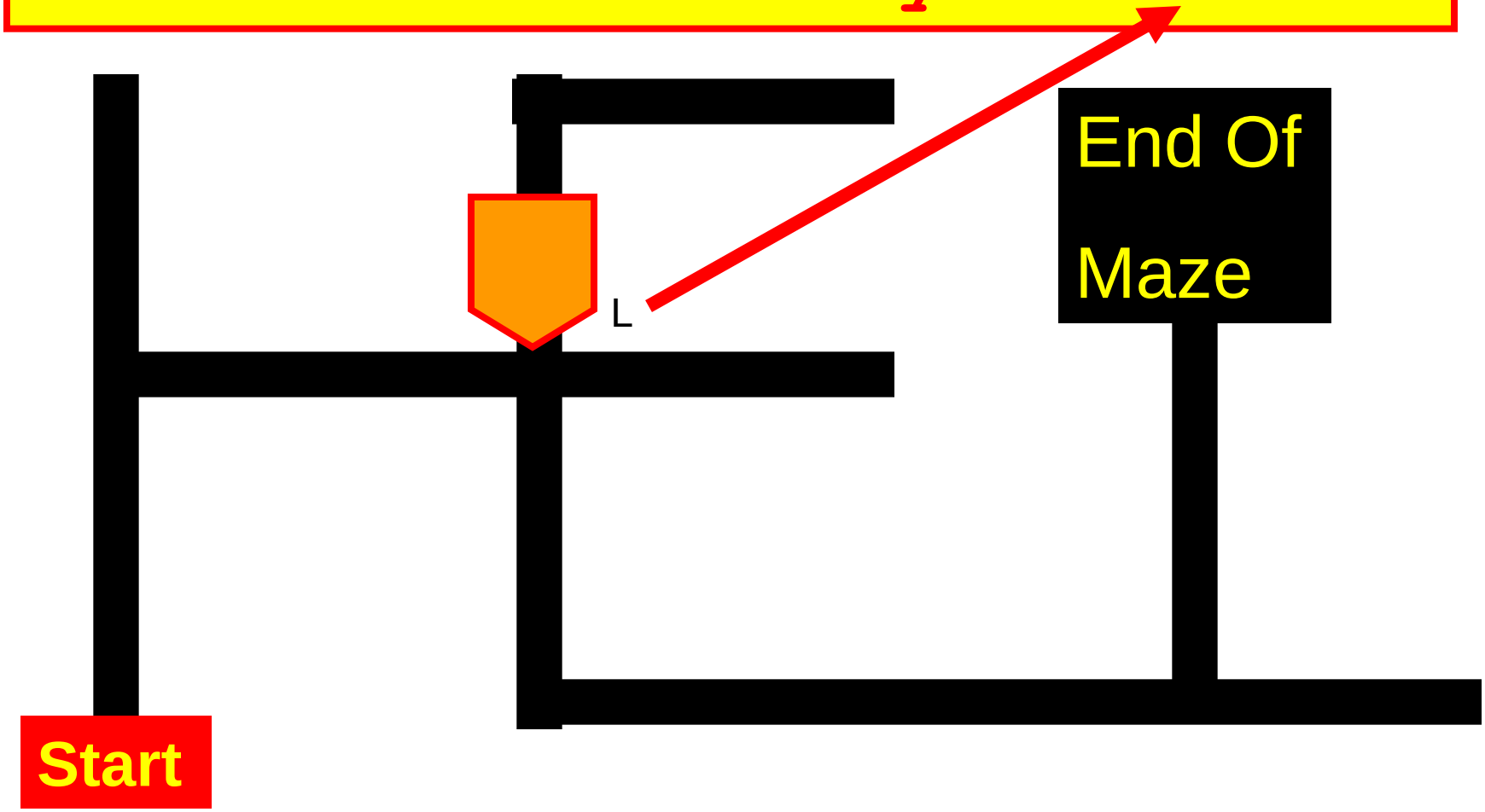
Next, the robot will encounter a dead-end, do a U-turn and record a “U”.

Path Stored in Memory: RLU



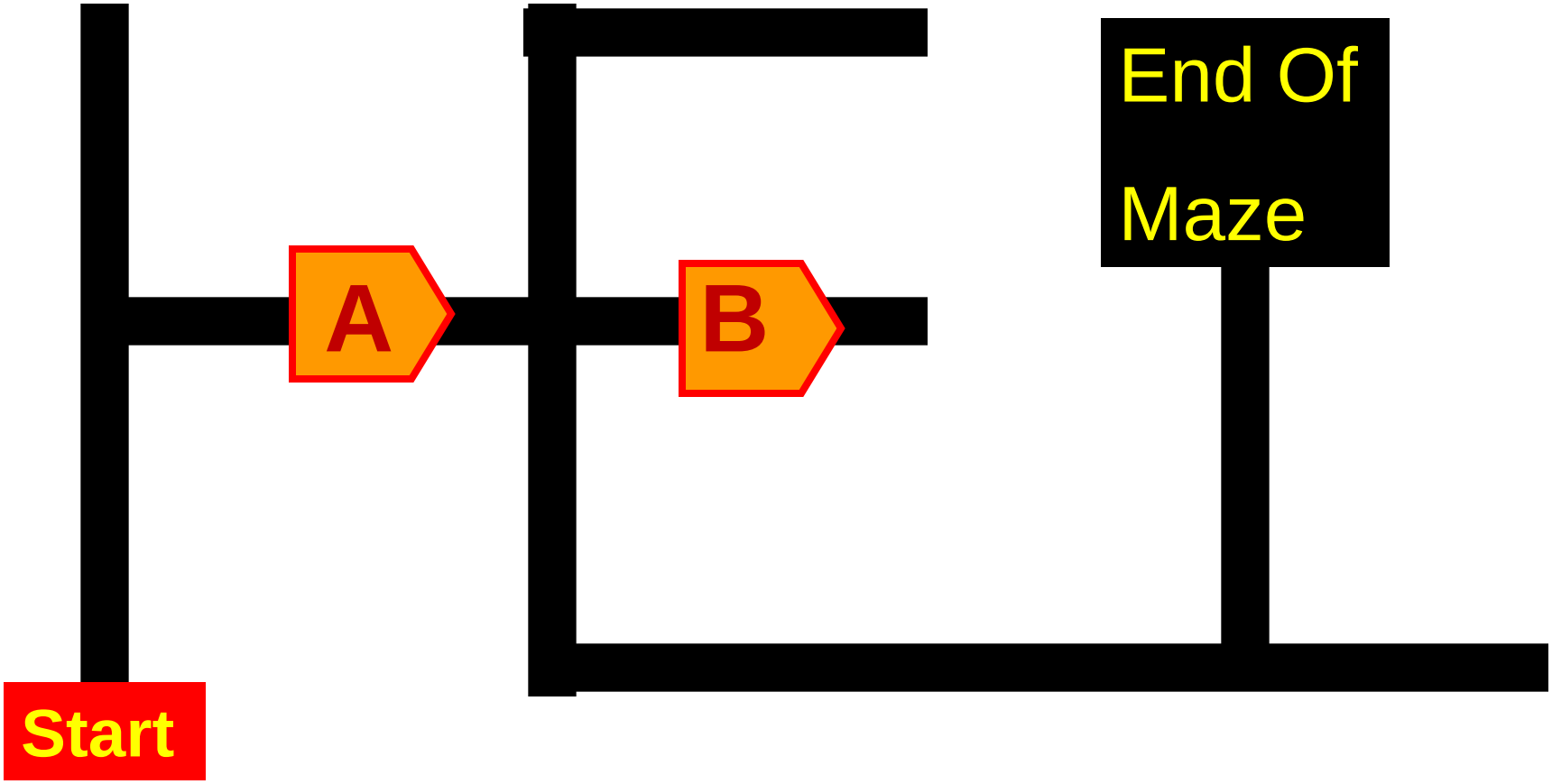
And we know from before, that the "U" tells us we just made a bad turn and need to return to the intersection we just came from.

Path Stored in Memory: RLUL



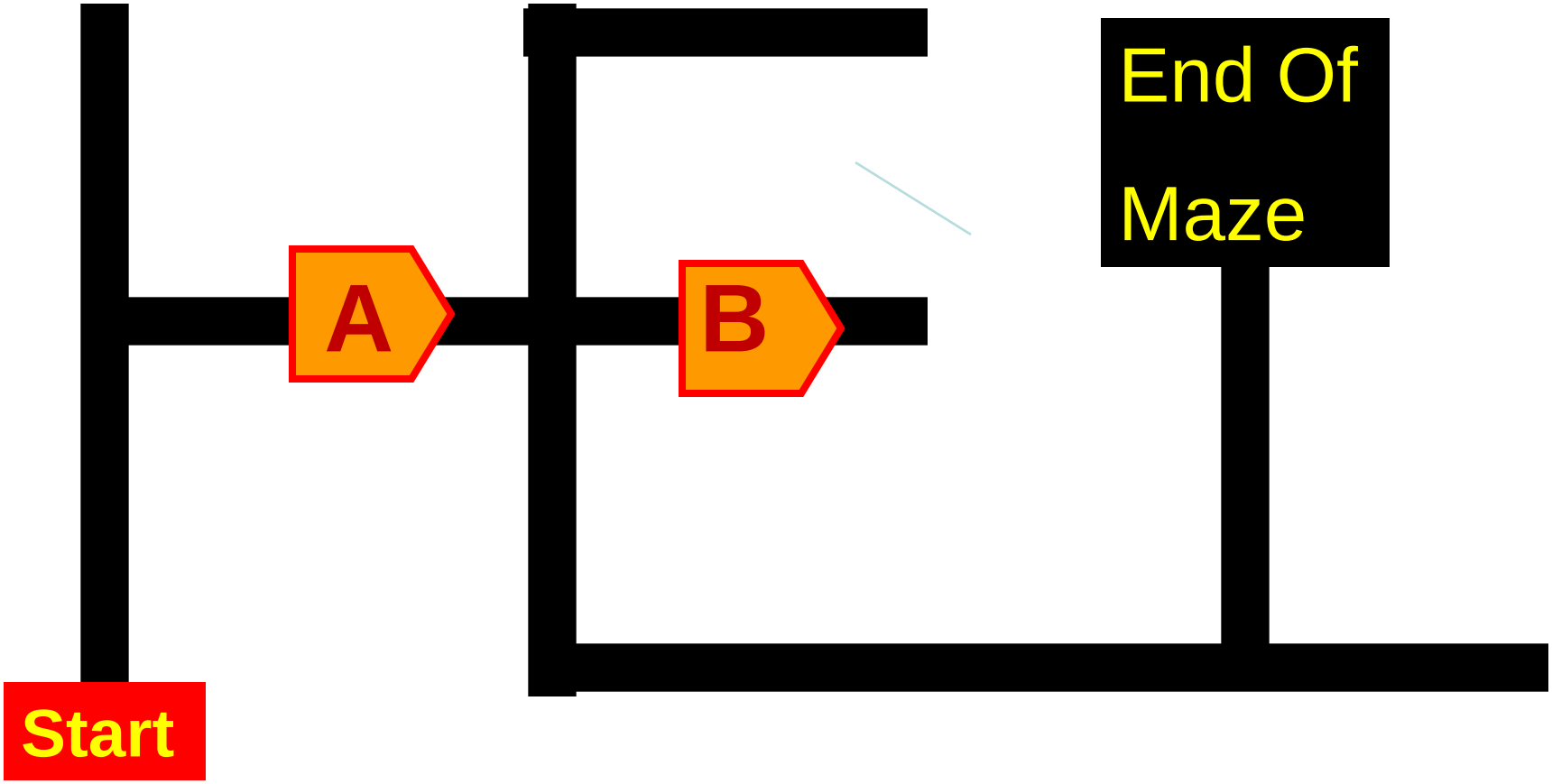
Returning to the intersection, we see left turn coming, so record the "L" and since we just came down a bad path, examine the path.

Path Stored in Memory: RLUL



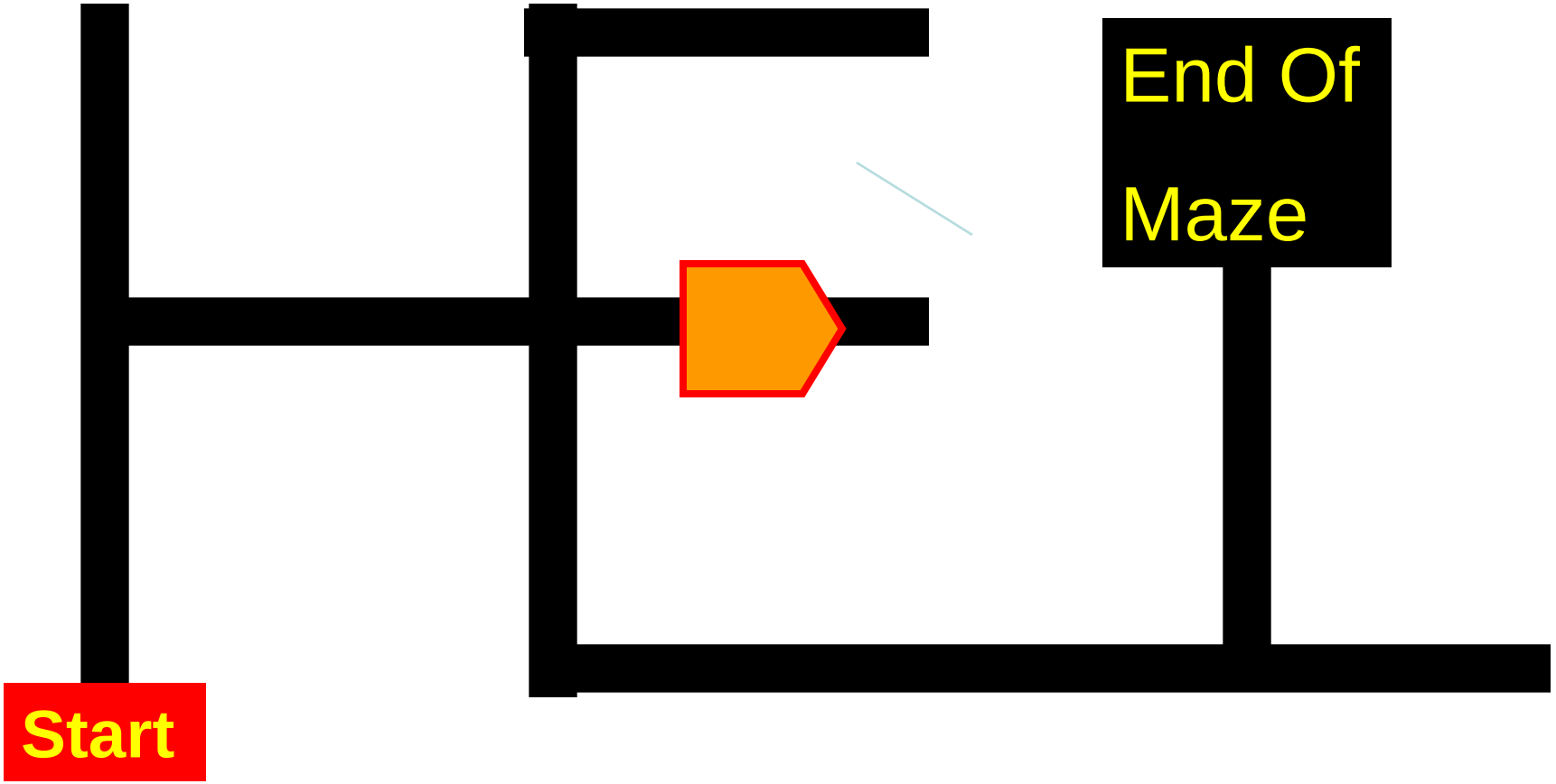
So, the robot, getting from point **A** to point **B** did "LUL" which we know to be bad. What would the robot do next time to avoid this?

Path Stored in Memory: R~~LUL~~



Next time, we want to go straight, so we can replace the "LUL" with "S" for Straight.

Path Stored in Memory: RS



We now have two “Replacement Rules”:

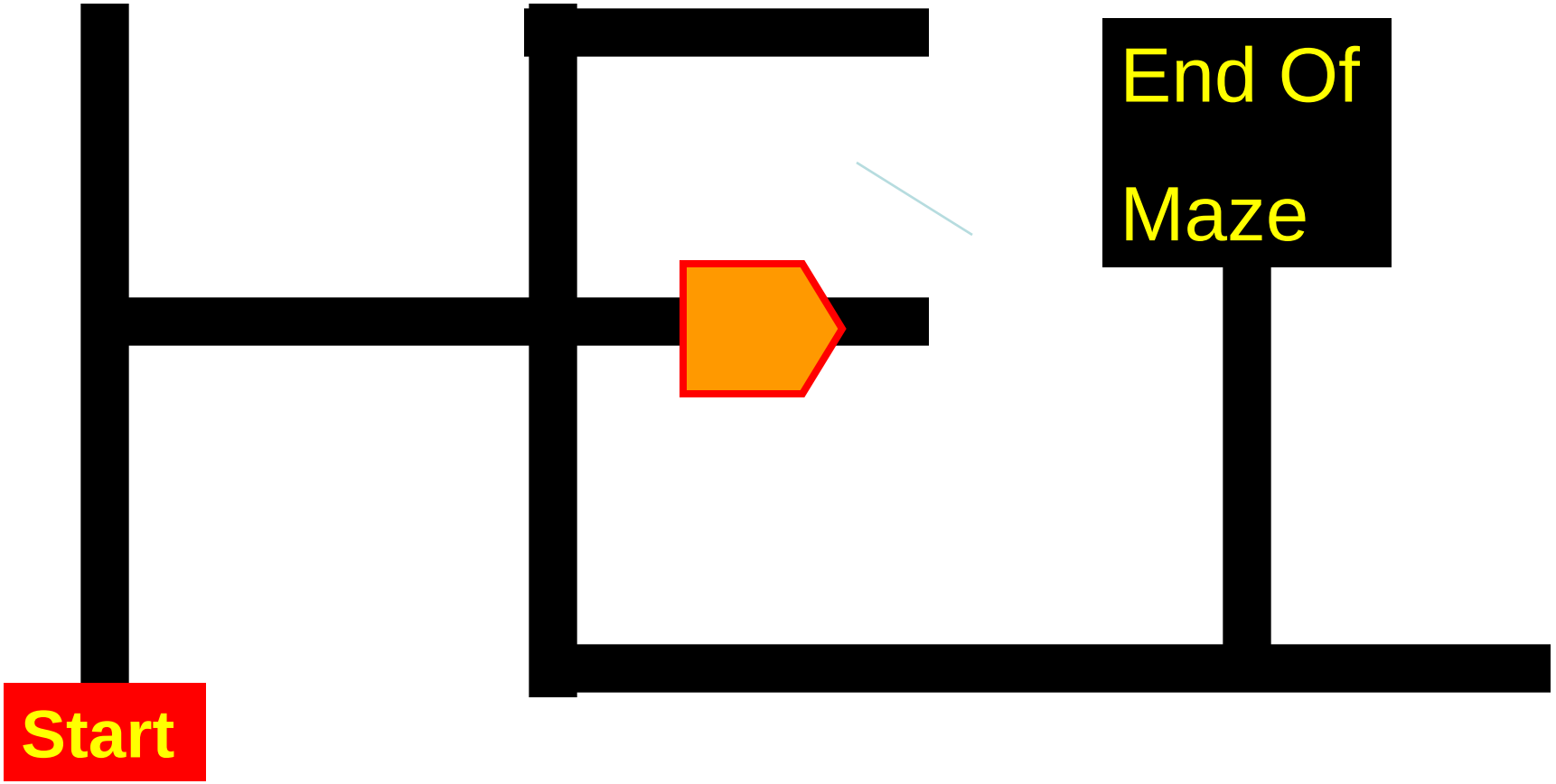
1. Replace “SUL” with “R”.

2. Replace “LUL” with “S”.

Replacement Rules

We will continue to develop these "Replacement Rules" until we have covered all of the maze possibilities we expect to encounter.

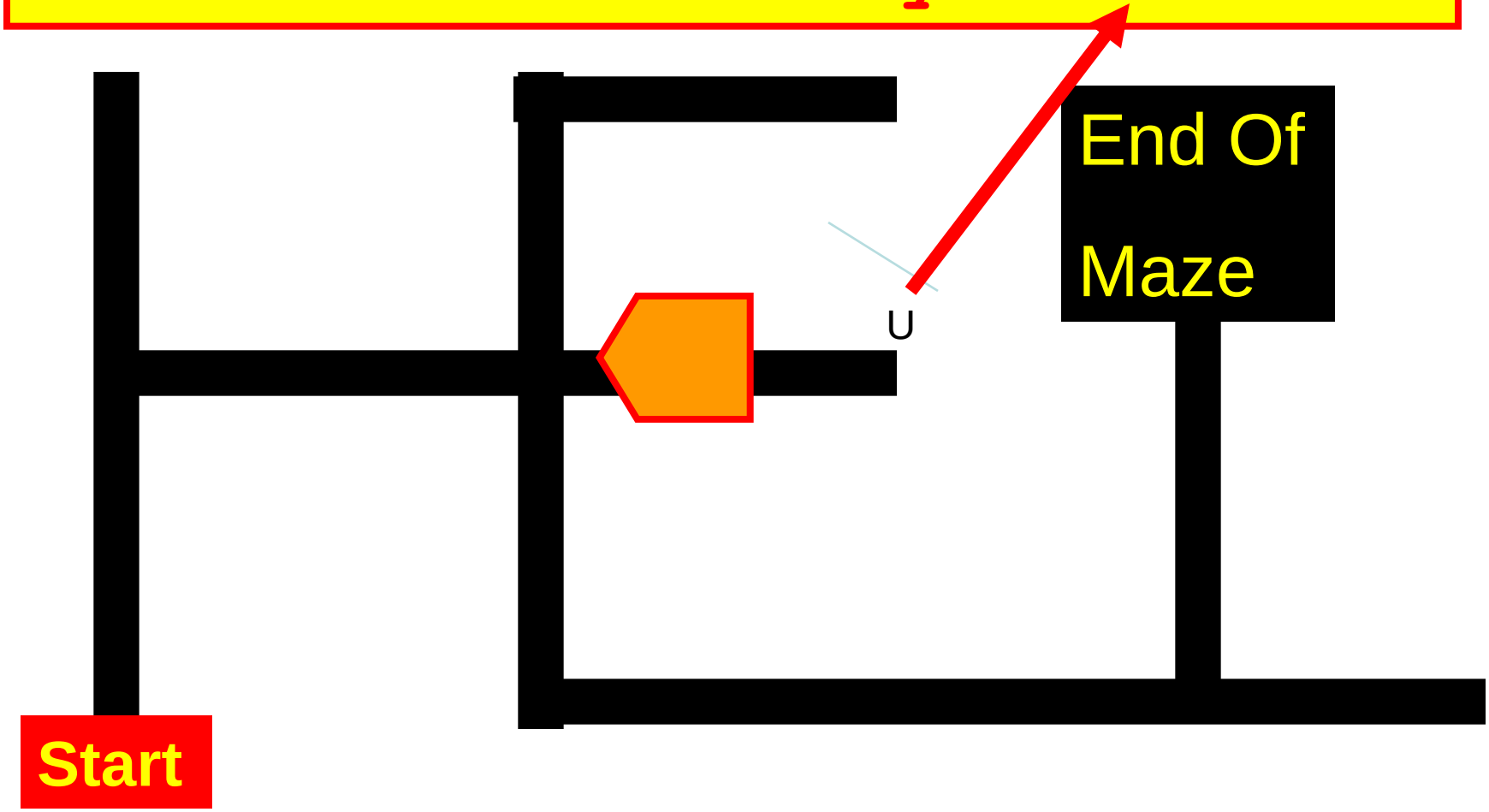
Path Stored in Memory: RS



Once again, we are coming up on a dead-end, so we know that a Replacement Rule is coming soon.

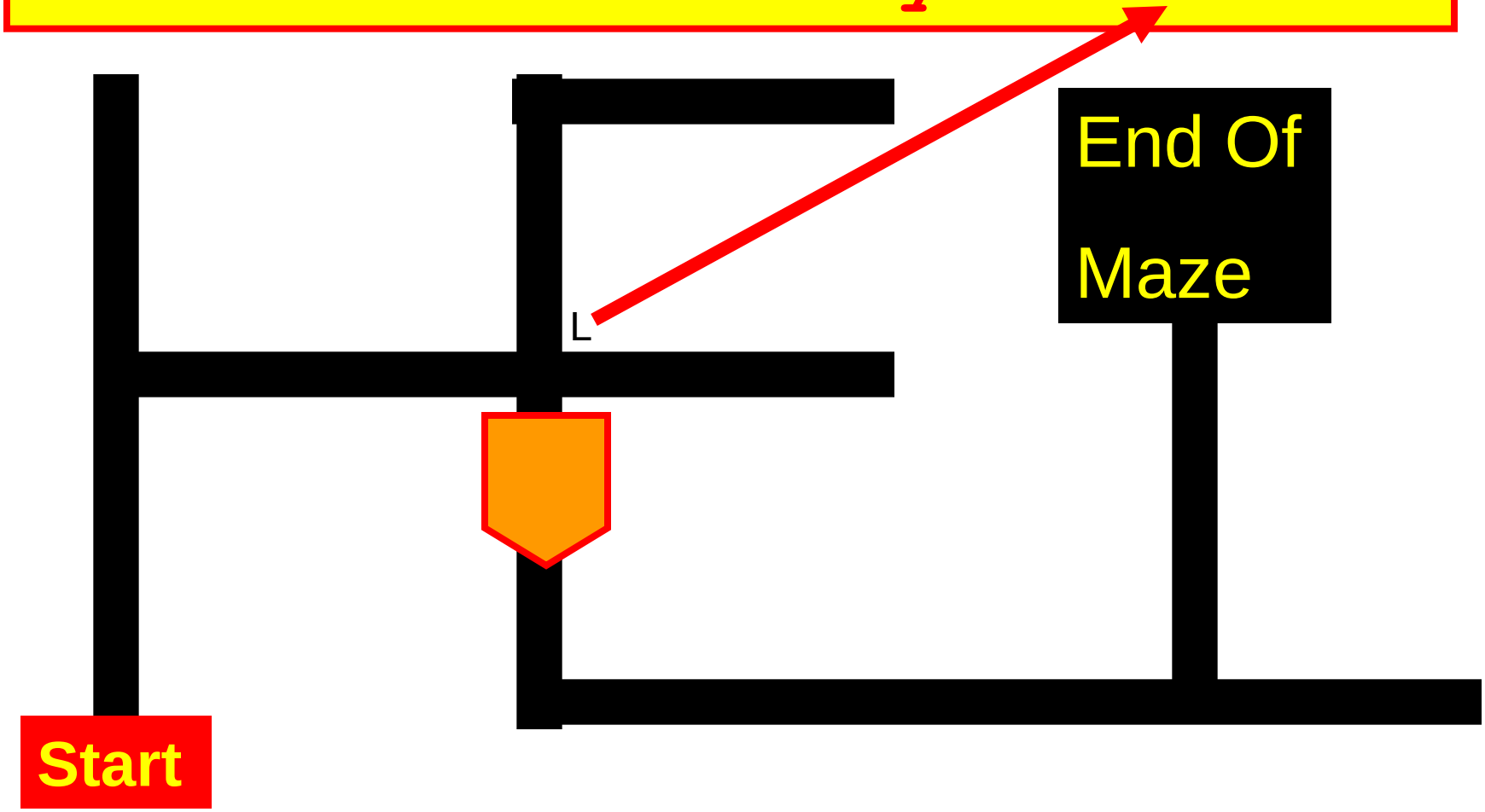
Do a 180 degree turn and record the “U”.

Path Stored in Memory: RSU



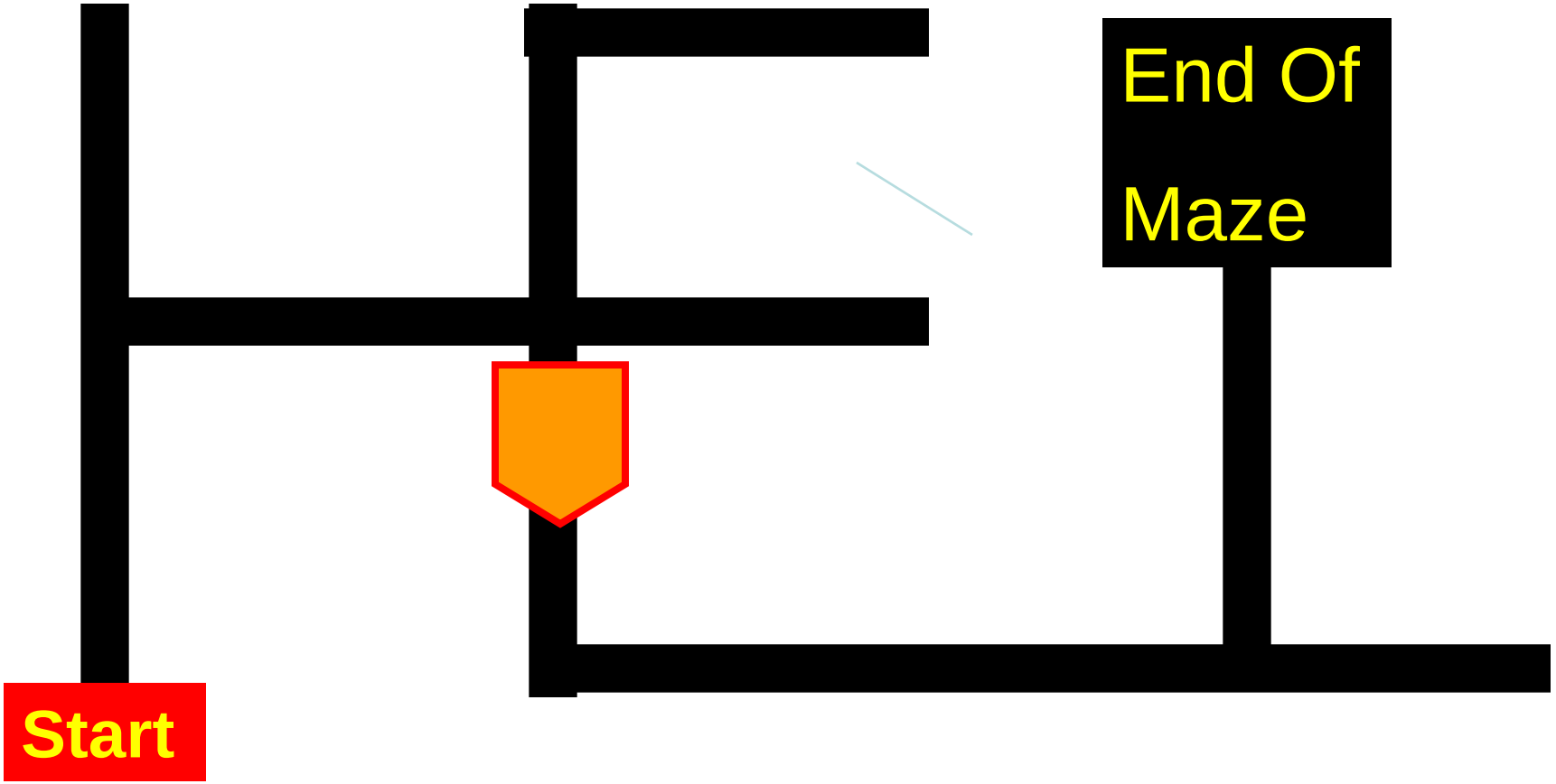
As we return to the intersection, we will take a left turn and record it.

Path Stored in Memory: RSUL



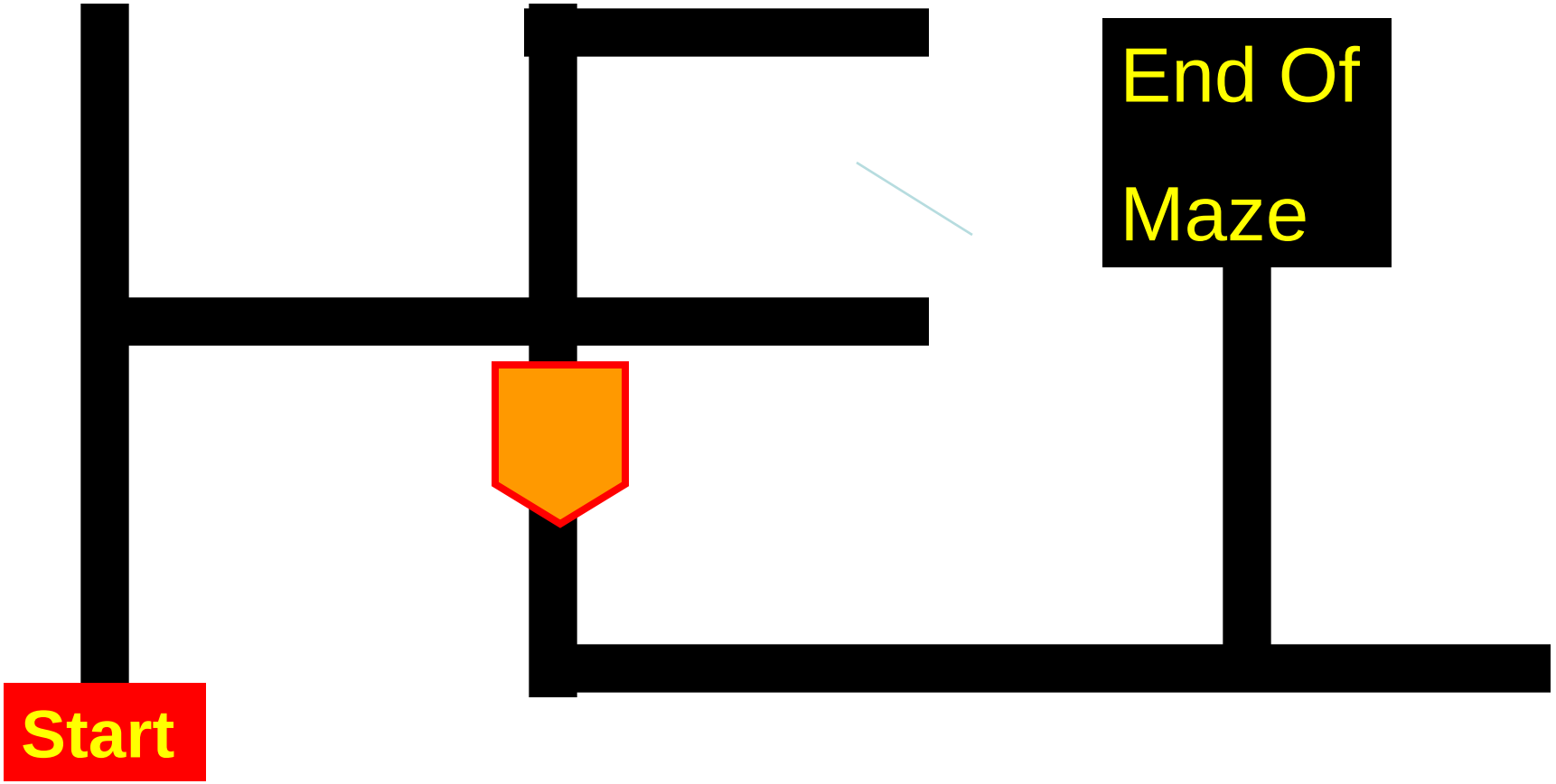
As we return to the intersection, we will take a left turn and record it.

Path Stored in Memory: RSUL



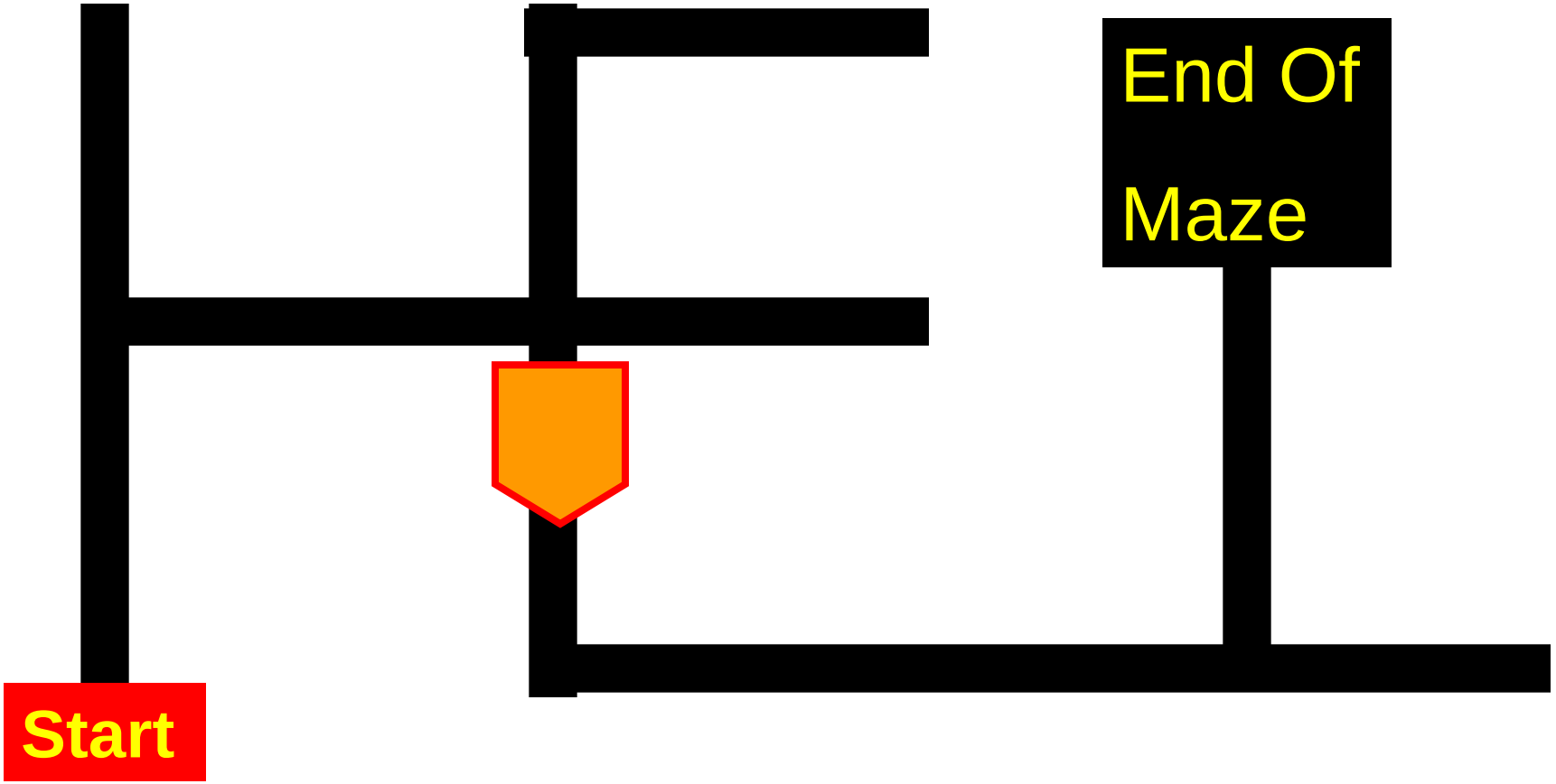
By now, you may be catching on to the answer to the question? How do we know when it is time to invoke the Replacement Rule?

Path Stored in Memory: RSUL



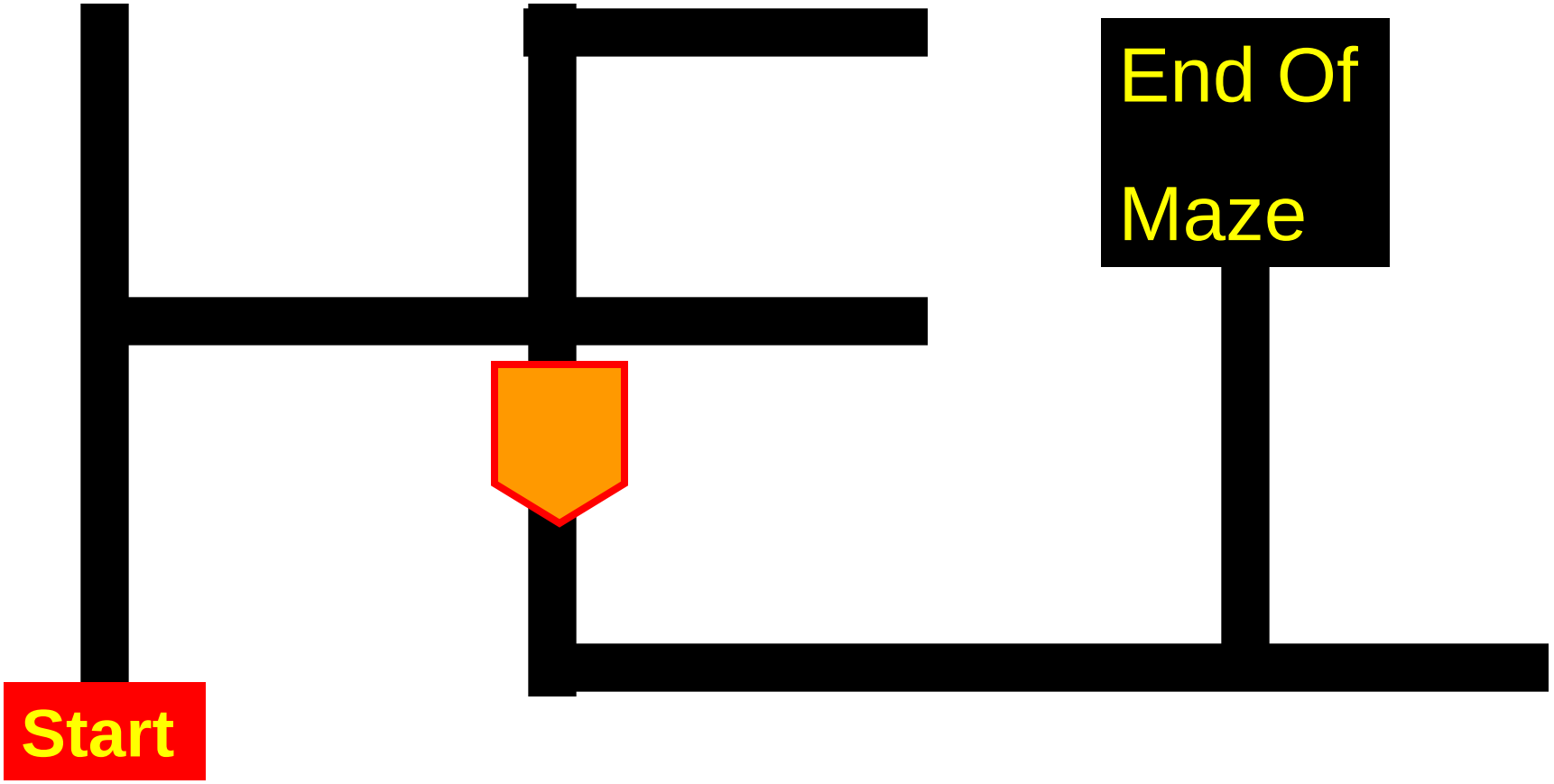
How do we know when it is time to invoke the Replacement Rule? - - - - Whenever the next to last character in the stored path is a “U”!

Path Stored in Memory: RSUL



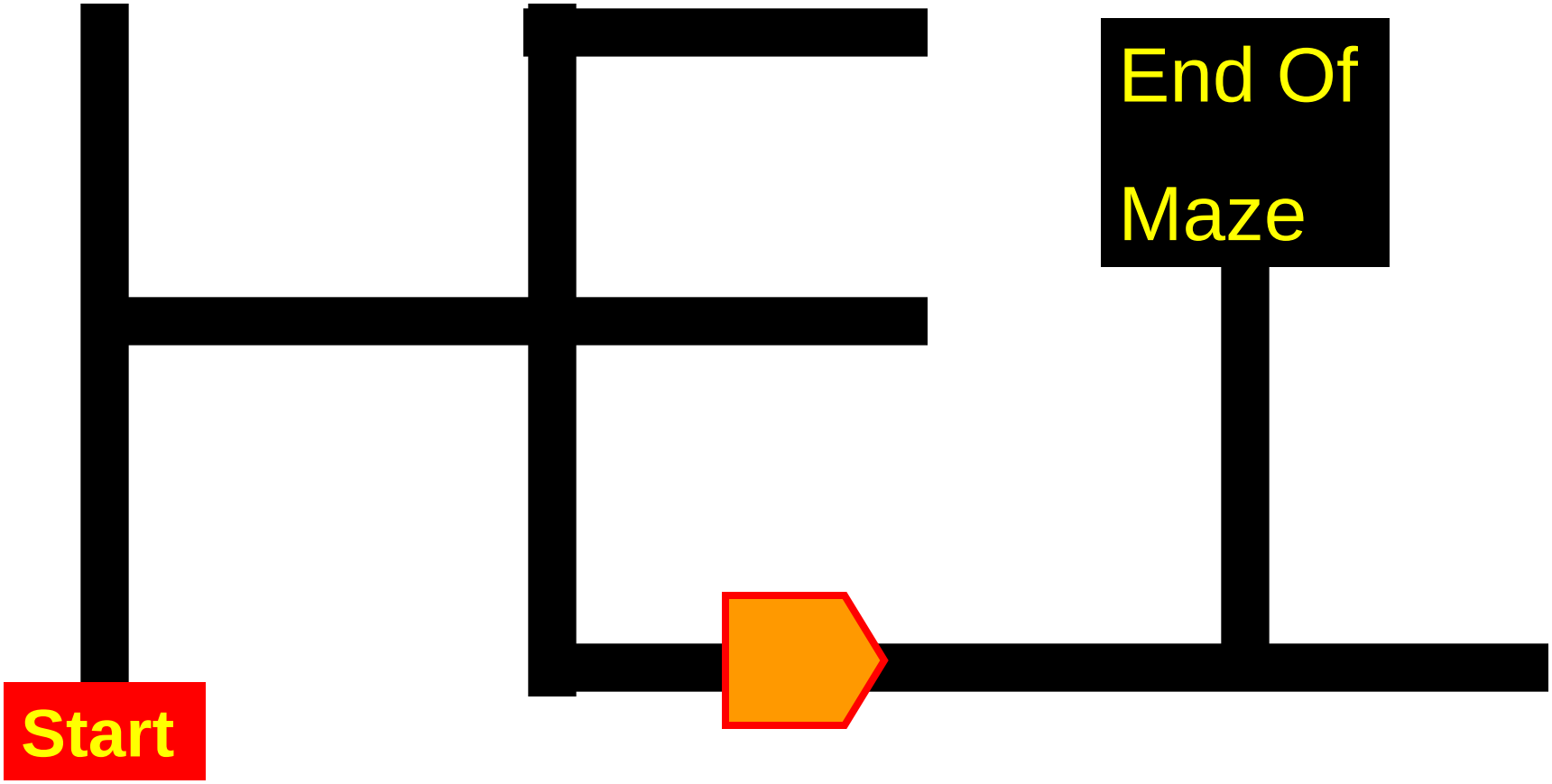
We got here with "SUL", so the correct path next time will be to take a right turn, so replace "SUL" with "R".

Path Stored in Memory: RR



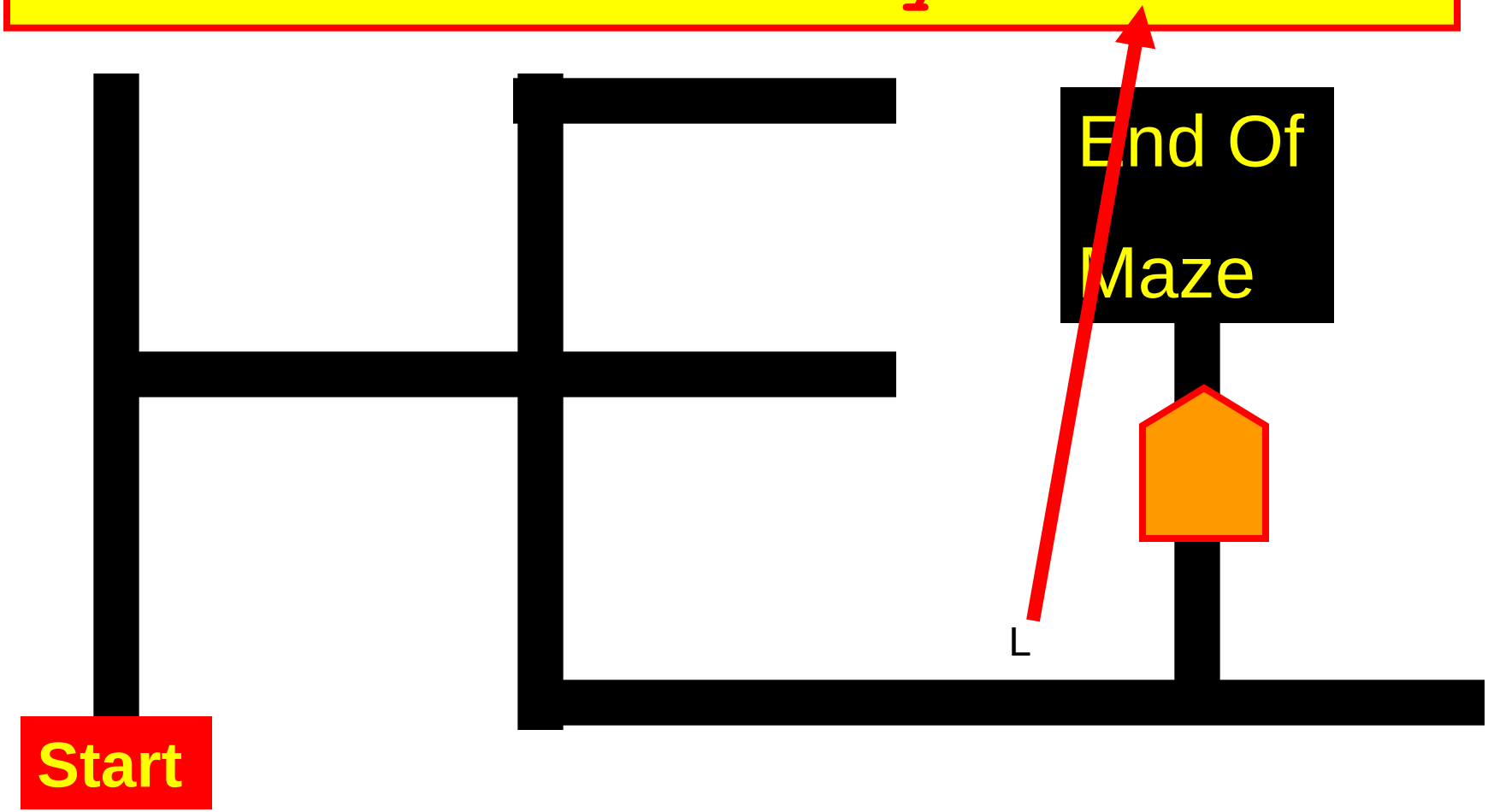
As discussed previously, the left turn coming up is not an intersection, since there is no choice, so take the left and record nothing.

Path Stored in Memory: RR



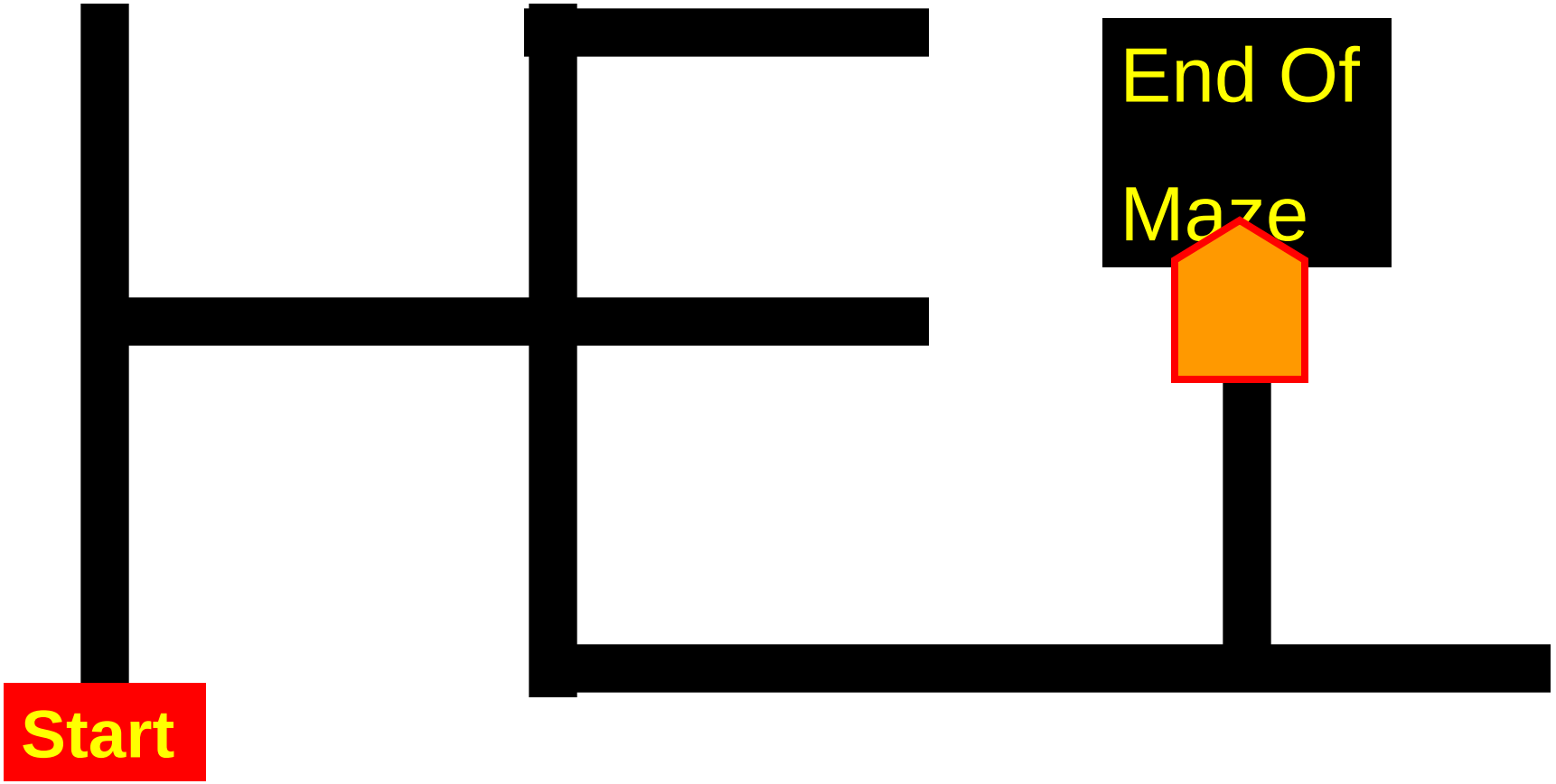
Only one more turn! The intersection coming up is a "Straight-Left" and the left hand rule will have the robot turn left and record an "L".

Path Stored in Memory: RRL



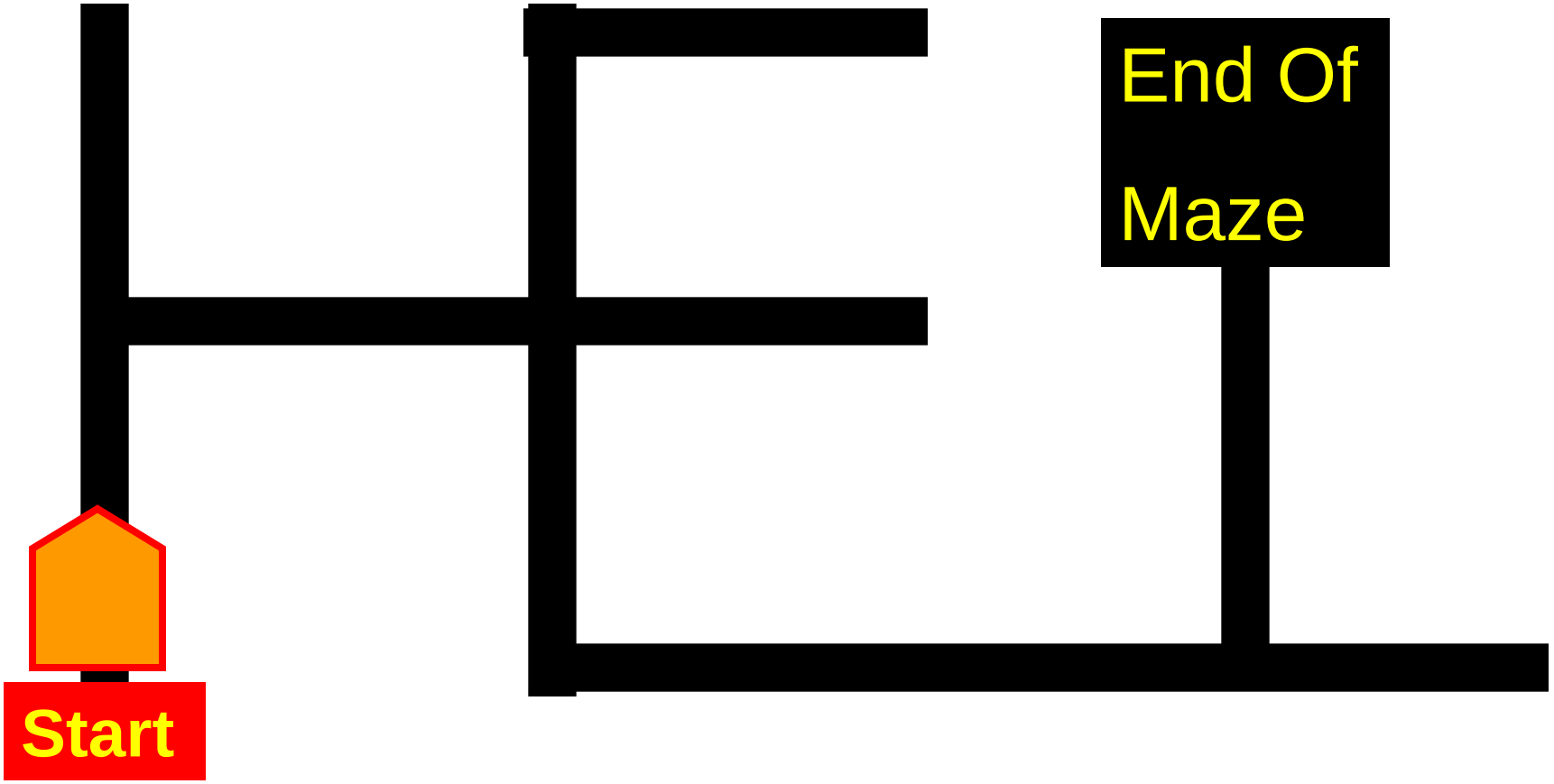
So, turn left and record an “L”.

Path Stored in Memory: RRL

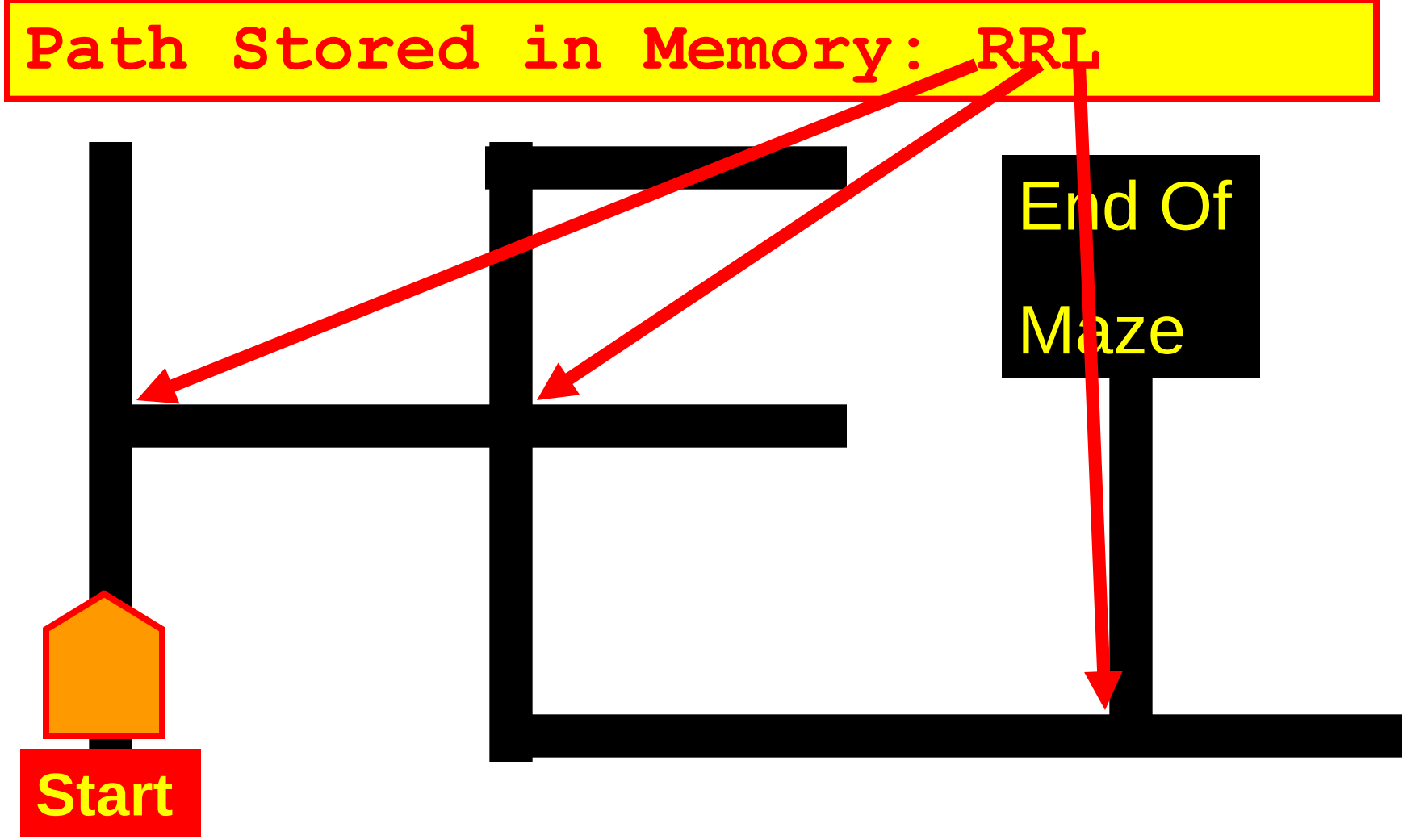


So, we reach the end of the maze with RRL in memory, ready to take the next run.

Path Stored in Memory: RRL



On the next run, the "RRL" will properly guide the robot to the finish using the shortest possible path.



TaDaaaaaa!

Now You Know

- I have not covered every possibility.

- This is a “Teach you how to fish” presentation -
- NOT a “Give you the fish” presentation.

- Have fun developing the source code for this algorithm.

- Please contact me if you have any questions.
  RoboticsProfessor@gmail.com

Line Maze Algorithm

Thank you for taking the time to review this presentation!

Richard T. Vannoy II

April 2009

RoboticsProfessor@gmail.com

Please email me at the address above if you have questions or comments.