

# MESA Network Defender: Robot Controller Guide

---



## Overview

The MESA Network Defender Robot is a high school cyber programming challenge that requires students to apply both programming and cryptography skills. One of the key components of the challenge is programming an autonomous robotic controller that guides the robot around a network. This document provides the background information required to implement this controller.

## Background

### The Networks

The networks are represented by mazes, and each network is made up of cells of equal sizes that are aligned with walls. This greatly simplifies the navigation task. However, keep in mind that a maze can have an arbitrary number of cells and is not limited to a square or rectangular shape. See Figure 1 for an example of the network.



**Figure 1: Network**

During the challenge, the network will only be presented from the first-person perspective of the robot, so there is no top-down view of the network as shown. However, for development, there is an option available to view the maze from a top view to assist algorithm development.

Keep in mind that at the competition date, a different set of mazes will be provided. Therefore, it is crucial to create controller software that is flexible enough to navigate an unknown network.

### The Viruses

Each network has been infected with viruses. The mission of the robot is to move around the network to neutralize as many of these viruses and collect points. There are two types of viruses:

#### Benign Bugs

To defeat a benign bug, a robot simply has to make contact with the virus. Figure 3 depicts a Benign Bug. 1,000 points are awarded for defeating each Benign Bug.

## Vile Viruses

Vile Viruses are protected by a lockbox that must first be disarmed by solving a cryptography puzzle. See Figure 4 for a picture of a lockbox. Once the secured box is disarmed, the virus inside is exposed and the robot can proceed on to eliminate the virus (by making contact with it) and collect points. Each defeated Vile Virus is worth 5,000 points.



**Figure 3: Benign Bug**



**Figure 4: Locked Box Protecting a Vile Virus**

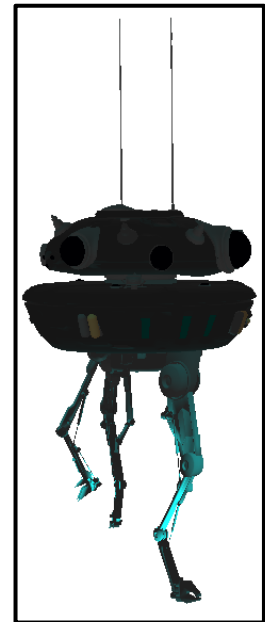
## The Robot

The virtual robot is represented as a 3D model of a droid, depicted in Figure 5. Even though the robot is visualized by the 3D model, the robot is represented as a sphere with a diameter of half a cell size for collision purposes.

The robot is capable of following basic movement commands, including commands to move forward and backward and to turn left and right. The robot also has three obstacle sensors for detecting distances to the walls in front of it and to its left and right. Lastly, the robot can sense all viruses within a predefined radius.

A list of available commands is provided in the next section.

For this challenge, the students will be required to create a robot controller, which is an intelligent program that tells the robot how to navigate through the network mazes in search of the viruses.



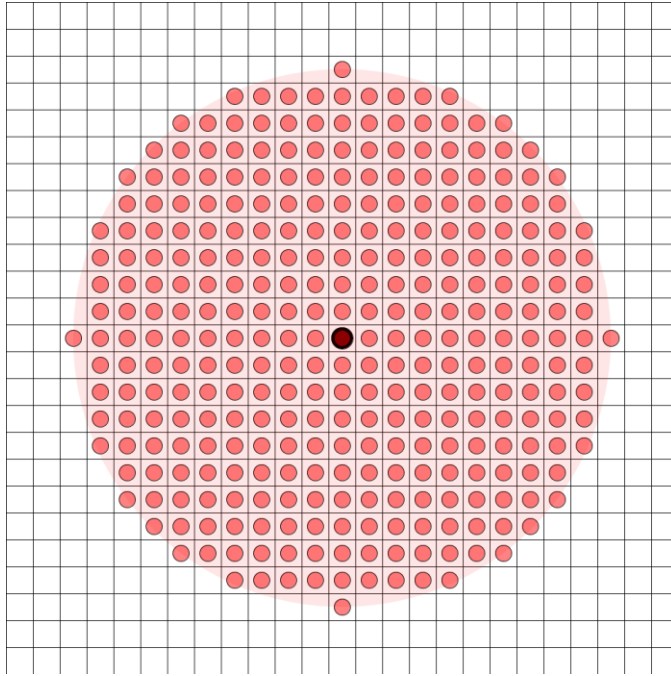
**Figure 5: The Robot**

## Robot Commands

The following commands are available to use in robot controllers.

**Table 1: Robot Controller Commands**

Command	Command Description	Example
<b>step_forward</b>	Move forward 1 cell Move forward N cells	<code>robot.step_forward()</code> <code>robot.step_forward(3)</code>
<b>step_backward</b>	Move backward 1 cell Move backward N cells	<code>robot.step_backward()</code> <code>robot.step_backward(2)</code>
<b>turn_left</b>	Turn 90° left U-Turn	<code>robot.turn_left()</code> <code>robot.turn_left(2)</code>
<b>turn_right</b>	Turn 90° right U-Turn	<code>robot.turn_right()</code> <code>robot.turn_right(2)</code>
<b>sense_steps</b>	Returns the number of free cells between the robot and a wall for a specified sensor. Can use predefined constants to specify the sensor:  <b>SENSOR_FORWARD [DEFAULT]</b> <b>SENSOR_LEFT</b> <b>SENSOR_RIGHT</b>	<code>robot.sense_steps()</code> <code>robot.sense_steps(</code> <code>robot.SENSOR_FORWARD)</code> <code>robot.sense_steps(</code> <code>robot.SENSOR_LEFT)</code> <code>robot.sense_steps(</code> <code>robot.SENSOR_RIGHT)</code>
<b>sense_viruses</b>	Allows the robot to send out a pulse to detect all viruses within a radius of 10 cells. The method returns a list of virus. For example:  <code>[(3,1), (-2,-4)]</code>  means there is a virus three cells to the left and one cell in the front, and a virus two cell to the left and four cells behind. For a virus to be detected, its center must lie within a 10 cell radius of the robot. For example, Figure 6 illustrates which viruses (light red dots) could be detected by a robot positioned at the dark red dot.	<code>robot.sense_viruses()</code>



**Figure 6: Circle shows the range of the virus sensor**

## Controller Files

### Robot Controller Development

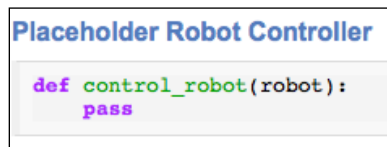
One of the primary challenge components for which the students are responsible is the development of a series of robot controllers. Each network requires its own robot controller as there may be a desire to use a slightly different algorithm based on what is presented in each maze. To encourage code reuse, the students will be able to submit a code base that may include some core robot controller functions in a Python module that can be referenced from each of their robot controllers.

The challenge software engine will automatically import the designated robot controller Python module for a given network and pass robot navigation responsibilities to that module. This module must define the `control_robot()` method that is called upon initialization. This is the only method that must be implemented to control the robot. This method has only one argument: the robot object itself.

The automatically imported robot controller Python module has a filename that follows the pattern `controller_<NETWORK>.py`, where `<NETWORK>` is the name of the network being attempted (which is the same as the network name displayed on the main screen). On competition day, these modules will already be provided with an empty `control_robot()` method declaration. For example, if attempting the first network (named `N1`), the module `controller_N1.py` will

automatically be imported and the `controller_N1.control_robot()` method will be invoked. If a controller cannot be found to match the current network, it will search for a default file `controller_default.py` and use it instead.

By default, each of the placeholder robot controllers will have the contents displayed in Figure 7: Placeholder Controller Contents.

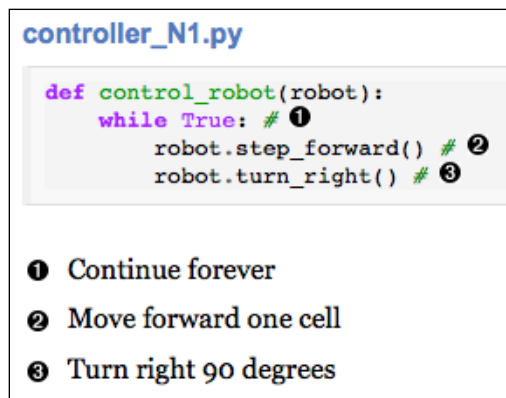
A code editor window titled "Placeholder Robot Controller" showing a Python function definition. The code is: 

```
def control_robot(robot):  
    pass
```

```
Placeholder Robot Controller  
  
def control_robot(robot):  
    pass
```

**Figure 7: Placeholder Controller Contents**

Figure 8 shows a simple robot controller that causes the robot to simply go around in a square (assuming there are no walls to interfere).

A code editor window titled "controller\_N1.py" showing a Python function definition. The code is: 

```
def control_robot(robot):  
    while True: # ❶  
        robot.step_forward() # ❷  
        robot.turn_right() # ❸
```

Below the code, there are three numbered annotations:   
❶ Continue forever  
❷ Move forward one cell  
❸ Turn right 90 degrees  

```
controller_N1.py  
  
def control_robot(robot):  
    while True: # ❶  
        robot.step_forward() # ❷  
        robot.turn_right() # ❸  
  
❶ Continue forever  
❷ Move forward one cell  
❸ Turn right 90 degrees
```

**Figure 8: Simple Controller**

If it is desired to use the front sensor to determine how far to move (instead of just moving one cell), the example controller could be modified as shown in Figure 9.

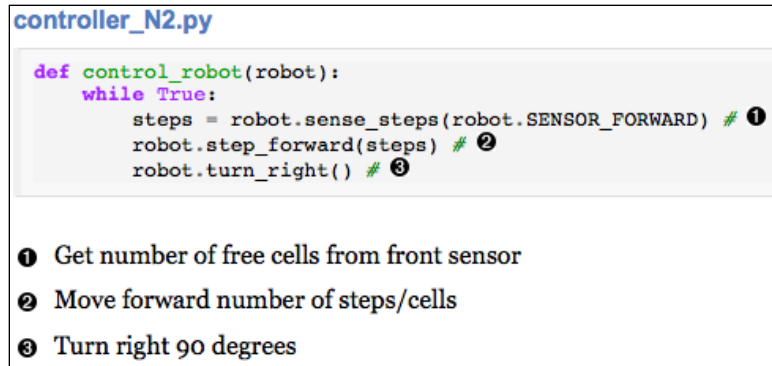


Figure 9: Simple Controller Using Sensor

### Prewritten Code

It is desired that students create their own robot controller building blocks and to reference them in their robot controllers. As mentioned in the overview, a code base may be emailed from the schools competition coordinator no later than two weeks before the competition that includes prewritten code for use on competition day. Any modifications to the code sooner than two weeks before the competition must be made manually on competition day.

The provided code should either be in one Python module file, or within one Python package directory. In addition, a default controller file can also be submitted, named **controller\_default.py**, to replace the default controller placeholders. On competition day, students will find the submitted module/package in the same directory as the robot controller placeholders, and all robot controllers for all levels will be initialized to the contents of the submitted **controller\_default.py** file. Figure 10 shows an example of a robot controller that makes use of some prewritten controller functions. How the students decide to manage control of the robot is completely up to them. However, keep in mind that on competition day there will be nine different networks, several of which may require the same types of control logic.

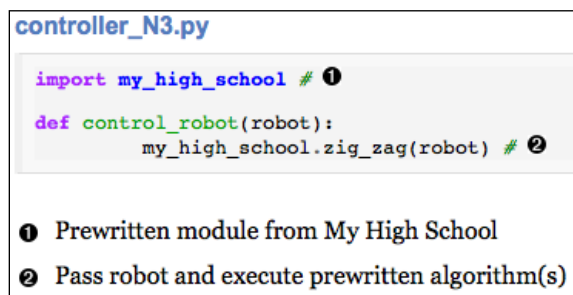


Figure 10: Prewritten Code Example

A generic robot controller might utilize the same controller logic for multiple networks, independent of what geometry the networks present. However, since a first person view of the network is displayed as each robot controller executes, it is possible to restart the network and use a different algorithm geared towards a specific problem. For example, a wall-following algorithm will not prove to be useful if there is a large room with a virus in the center. If a robot controller is generic enough, it is possible to use the same robot controller logic for multiple networks.