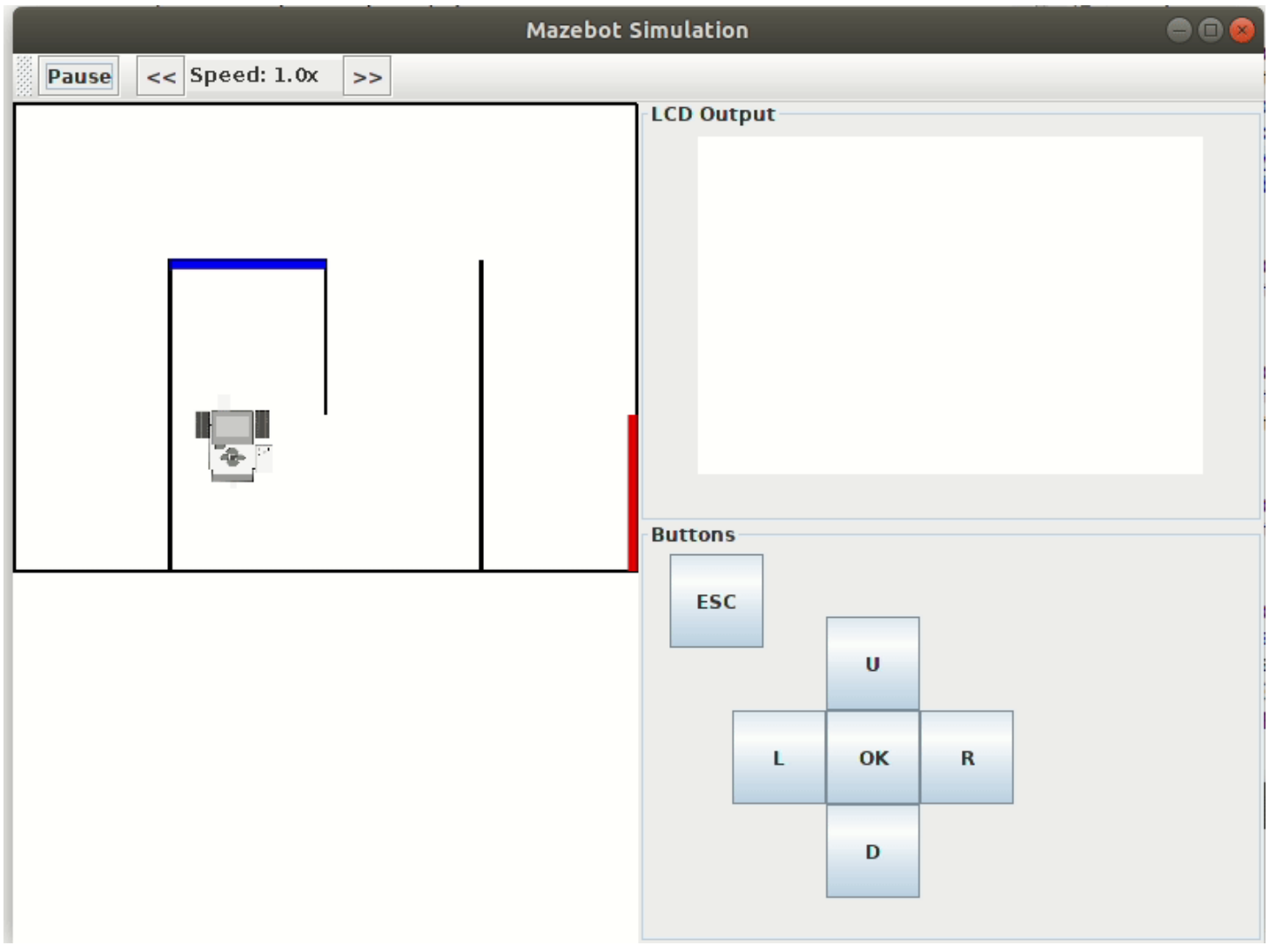
Last edited by **Peter Oppermann** 2 months ago

Home

The Mazebot simulator aims to provide a simulation environment that can execute programs written for the Lego EV3 robot with the leJOS operating system. It should not require any modification of the source code of the EV3 programs to make it executable on your local computer.

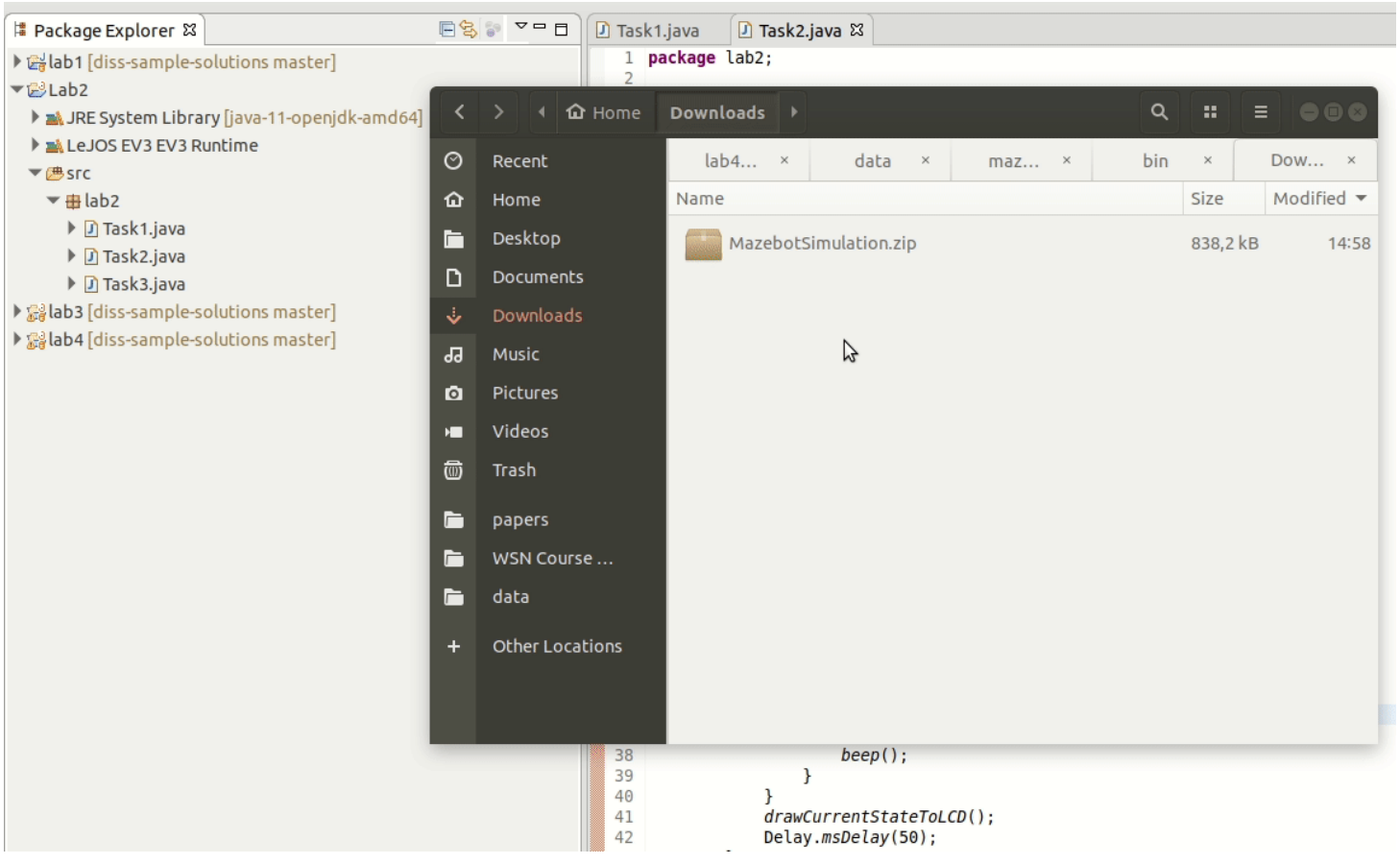


Installation and Setup

The installation requires you to do the following three steps. The steps are carried out and shown exemplarily for an existing leJOS project.

Get the code and the mazes

First, download the **jar archive** and the **mazes.zip** latest release from our release page and copy it into your project folder. You will need to add this archive to your build path so that the java compiler knows it has to look for available classes in there. Extract the content of the mazes folder into your project directory as well.

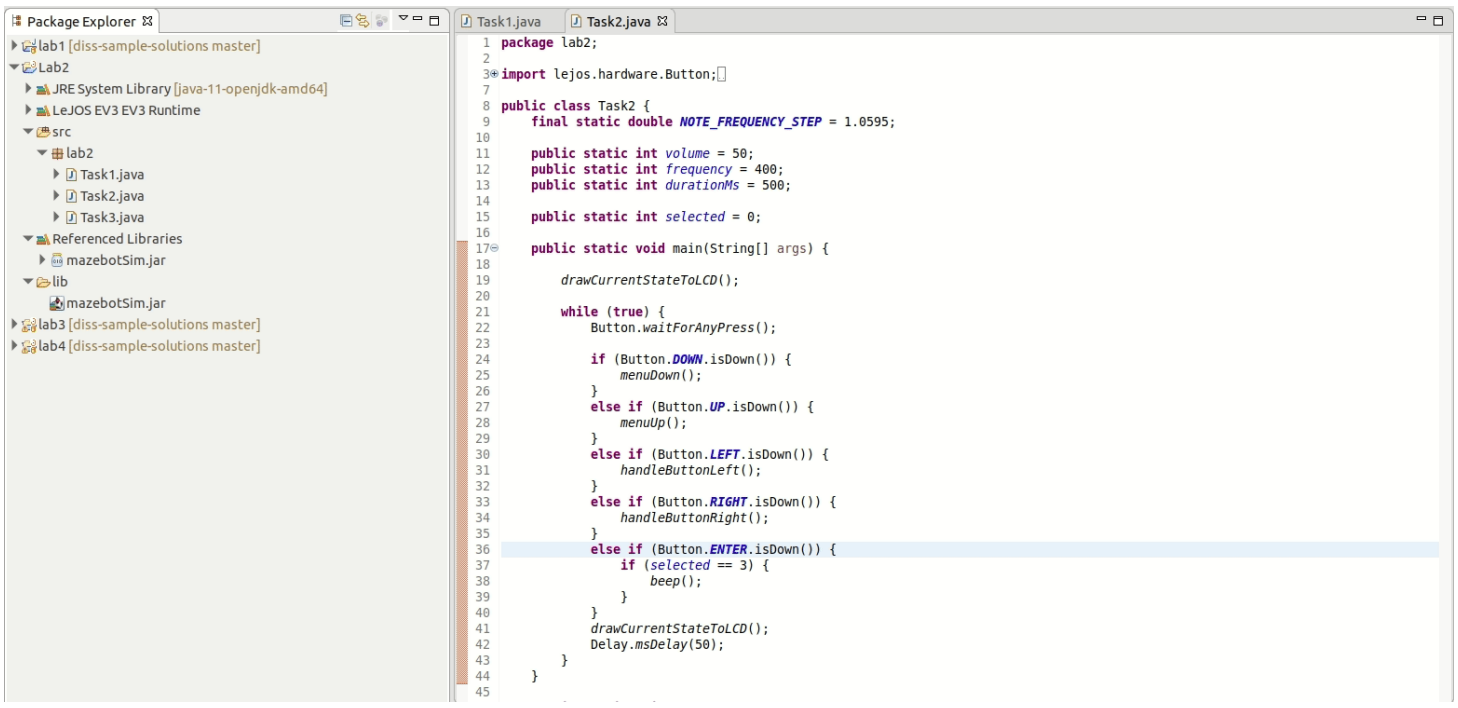


Create an alternative main class.

You need just a couple of lines of source code to set up the simulation. But since you still want to be able to execute your program on the real robot, we will create a new Main class with a new Main method that we will call `SimulatedMain.java`. In this simple Main function, we will set up the simulation and call your old main method, which is also runnable on the robot.

To do this, copy the following code snippet into the new main method. The methods are explained in detail in the Usage section.

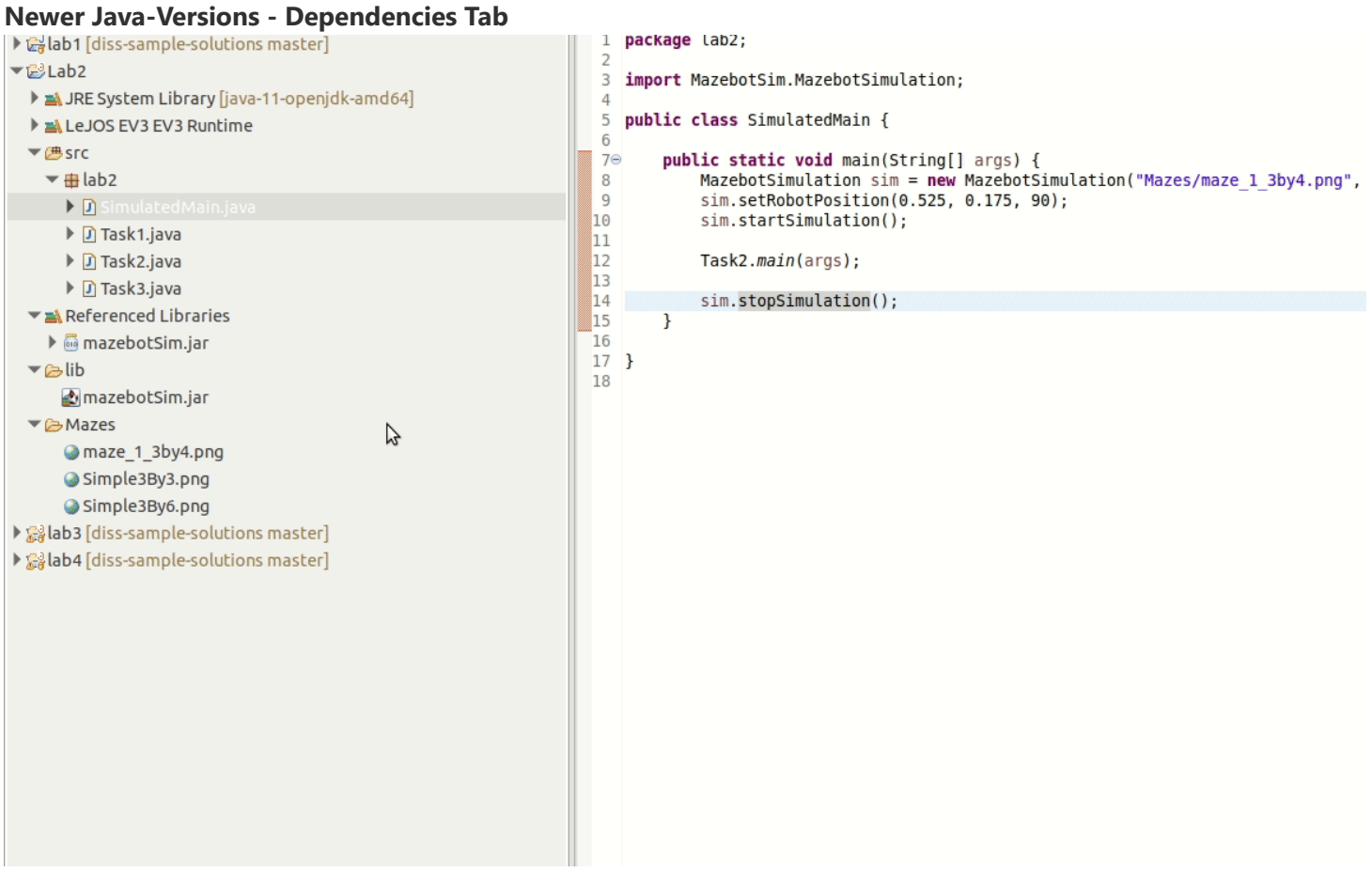
```
MazebotSimulation sim = new MazebotSimulation("Mazes/TestArea.png", 1.5,  1.5);
GuiMazeVisualization gui = new GuiMazeVisualization(1.5, sim.getStateAccessor());
sim.scaleSpeed(1);
sim.setRobotPosition(0.75, 0.75, 90);

sim.startSimulation();
gui.startVisualization();

//Call to the old main method here

sim.stopSimulation();
```



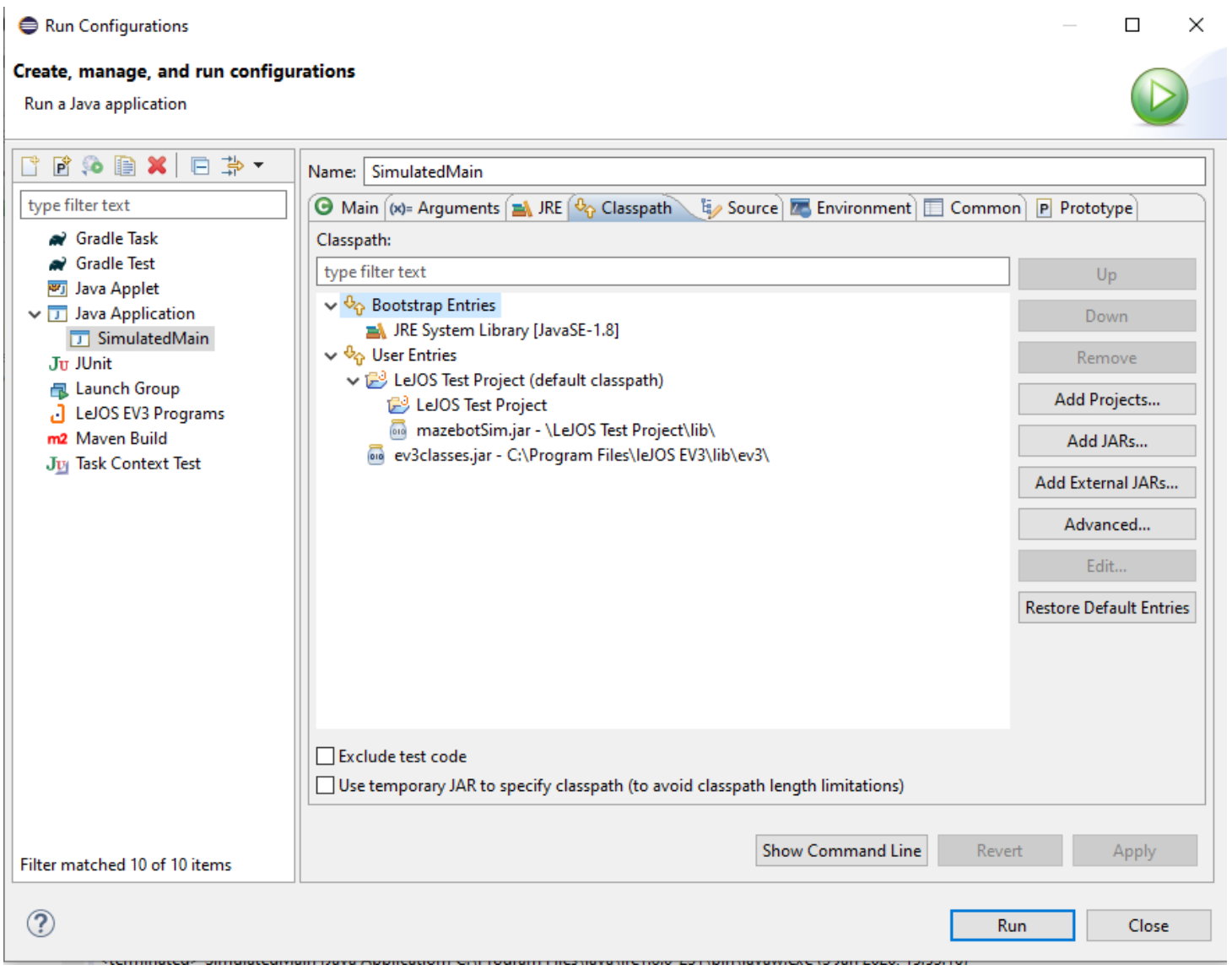
Create a run configuration

Finally, you need to create a new run configuration, that you can call whenever you want to execute the code in simulation instead of on the robot. To do so, right-click on the new SimulatedMain class in the project explorer and select 'Run As->Run Configurations'. Select Java Application. Now make sure, that in the Dependencies tab (In some older versions of Eclipse, the tab is called "Class-Path" and looks a bit different, see below) of the Run Configuration for the simulation, there is both the `MazebotSim.jar` and the original EV3 leJOS library `ev3classes.jar`. If not, add them from the directory where you have installed leJOS. Make sure that the `MazebotSim.jar` is above the `ev3classes.jar` in the list, since it overrides some classes from leJOS. You can move it up and down in the list using the Up and Down buttons on the right.

Newer Java-Versions - Dependencies Tab

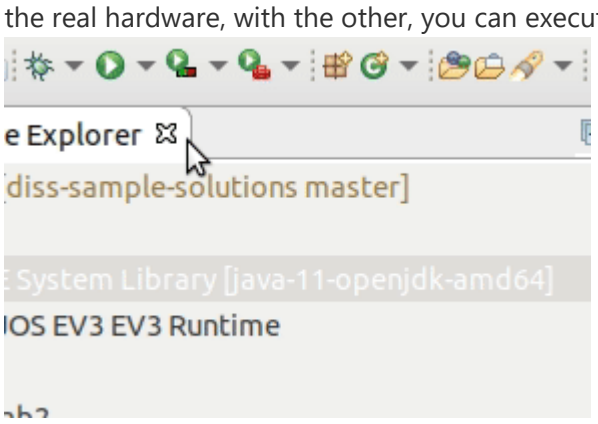


Older Java-Versions - Classpath Tab Here, please make sure, that you have the same entries in the **same order** as shown in the screenshot below. It is important that the `mazebotSim.jar` is above the `ev3classes.jar`. In case there is a `LeJOS EV3 EV3 Runtime` among the Bootstrap Entries, remove it.



Run the program

Now, you have two different run configurations. With one, you can flash your program to the robot and let it run on the real hardware, with the other, you can execute it on simulated hardware.



Usage

User Interface

The user interface provides the following functionalities:

- Visualize the position and orientation of the robot in the maze
- Show the LCD output
- Press the buttons on the robot
- Control the simulation speed and pause the simulation

API

The simulation has to be initialized in the code. In the Installation guide, you were asked to copy the following code snippet into your main method:

```
MazebotSimulation sim = new MazebotSimulation("Mazes/TestArea.png", 1.5,  1.5);
GuiMazeVisualization gui = new GuiMazeVisualization(1.5, sim.getStateAccessor());
sim.scaleSpeed(1);
sim.setRobotPosition(0.75, 0.75, 90);

sim.startSimulation();
gui.startVisualization();

//Call to the old main method here

sim.stopSimulation();
```

Constructor

`MazebotSimulation(String path, double width, double height)` The constructor takes three arguments

- **Path** to the png image, that defines the maze
- **Width** of the maze in meters
- **Height** of the maze in meters

The simulation will assume the full image stretches over an area of **width**x**height**.

Set the start position

`setRobotPosition(double x, double y, double orientation)` The robot is placed in the specified position. The coordinate system has its origin in the left lower corner of the maze.

- **x** The x coordinate (horizontal) in meters
- **y** The y coordinate (vertical) in meters
- **orientation** The orientation in degrees (clockwise). At zero degrees, the robot is headed to the right.

Designing new Mazes

You can easily design new mazes yourself. All you need to do is to create a new PNG image with an image manipulation software of your choice. The MazeSimulator interprets the image in the following way:

- The background needs to be transparent. Every pixel that is not transparent is interpreted as an obstacle, that the ultrasonic sensor detects.
- You can paint the walls in any color, as long as they are not transparent. The color sensor can detect the color that you paint the wall in.