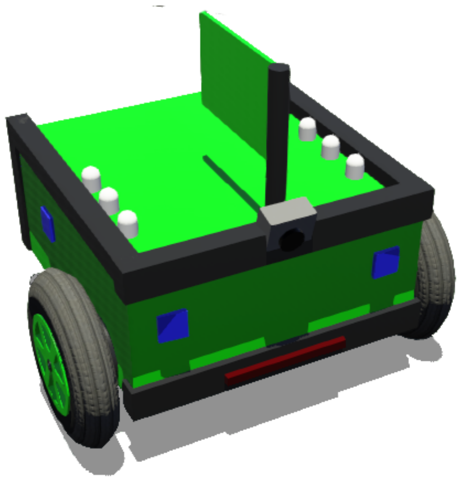
SR2021 Microgames



Welcome to SR2021, these tasks will help you learn how to program your robot and make best use of all of its sensors.

We recommend you work through this sheet as a team so you can learn from each other and get through it more quickly, don't be afraid to ask questions if you get stuck! Also if you're a seasoned robot programmer

0. Join the Discord chat!

For your first task, we'd recommend you join our discord chat server, we'll be using it as our forum to provide you mentoring and support, both for these tasks and onwards throughout the entire competition.

Your team leader should have received the link, along with a password to get into your specialized team section, so you need to ask them for a copy!

1. Getting Started

1. Get the simulator working! Follow the instructions at studentrobotics.org/docs/simulator/ (download **both** Webots and our simulation files)
   If you attended last year and still have Webots installed, you should upgrade to the latest version as there have been a number of bug fixes since then.

2. Run the test robot code.
   Once you've downloaded both files, install Webots, then extract the 'competition simulator' zip. This is going to be your main workspace for the simulator. Double click on the `arena.wbt` file (in the '`worlds`' folder) to run the simulation. It should launch webots (if not, ask for help!), and then start running a simulation of our sample robot.

3. After running the simulation, you can check your folders, there should be a file named '`robot.py`' file just outside the 'competition-simulator' folder.
   This is our sample robot code (if a file ends in .py, it means it is a python file). Open it in whatever code text editor you're using.

   There are many code editors available, such as Visual Studio Code.

Once you've found the `robot.py` file and managed to open it up to see, you can move on to the next task.

2. Hello World 👋🌍

A tradition all programmers do when they're learning to program is to get their code to write the words "Hello World". In this challenge, we're making things a bit more complicated, instead of "Hello World" we're going to run some code to get a secret password.

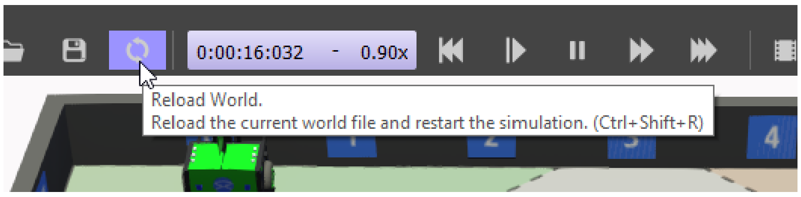
In python you can use the '`print()`' function to write out text for you to read later, for example, in the sample robot there's a line that says `"I can see {} things"`. The '`{}`' is automatically replaced by the number of things it can see, that's what '`.format()`' does. (you can read more about `.format()` at docs.python.org/3.7/library/stdtypes.html#str.format if you're interested)

Now, copy and paste the following code into the top of your `robot.py`, which should be just outside the 'competition-simulator' folder:

```
print(f"Secret Code: H3"+str(35%12)+f"0 W{0}{chr(82)}LD")
```

(Don't worry if you don't understand it, we wrote it to be deliberately hard to understand, so you can't figure out what it does before you run it)

Then, after saving, you can run the code by either double-clicking on the Webots `arena.wbt` file (in the 'worlds' folder) , or by clicking 'Reload World' at the top of the screen.



Reload World.
Reload the current world file and restart the simulation. (Ctrl+Shift+R)

The message should appear in the 'Console' at the bottom.

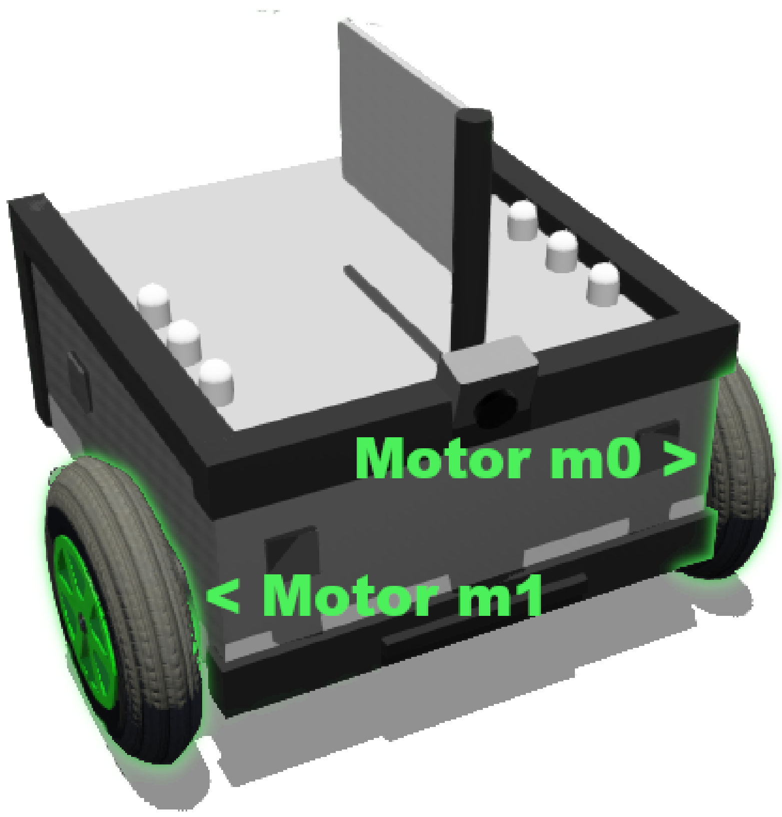
If you don't have a console at the bottom of the window, try resizing the main view of the simulation -- it may just be hidden. If that doesn't work, you can create the console by going to Tools > New Console.

If you get an error message, it might be that you have a version of python lower than our minimum, Python 3.7. You can install a later version at www.python.org/downloads/.

If you find the secret code, well done! You can move on to the next task.

3. Motor Movement

In order for your robot to move, you’re going to need to know how to move the motors. The robot has 2 motors, one on each side of the robot.



(as mentioned in studentrobotics.org/docs/simulator/programming/)

You can set the motor power to anything between -100 and 100. To find python functions that change the robot itself, you should read our documentation at studentrobotics.org/docs/. Specifically the ‘programming’ category, at studentrobotics.org/docs/programming/.

You should look for the motors section to find how to change the motor power.

Our simulation robot this year only has 1 motor board plugged in,

Now, make the robot drive forwards for 4 seconds, then drive backwards for 2 seconds. You should avoid using the normal python way to wait (`time.sleep()`), as it doesn’t guarantee it will run the same on every computer. Instead you should use `Robot.sleep()`, like this:

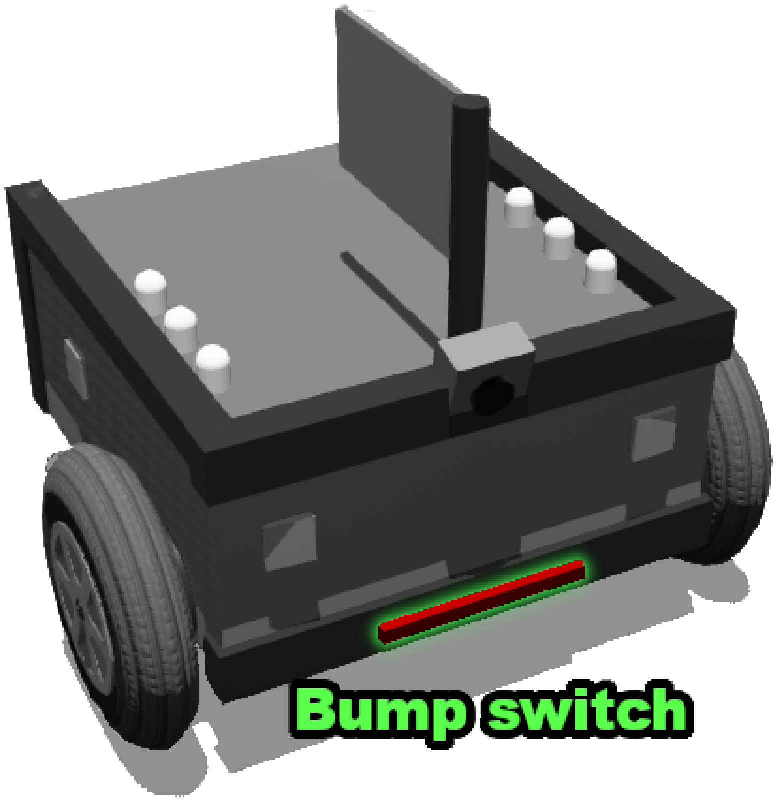
```
R = Robot()
R.sleep(seconds)
```

Use this to make the robot wait for a few seconds. While the robot waits, its motors keep moving at the same speed you last set them to beforehand.

4. Microswitches (Bump sensors) 👊

Bump sensors are great for detecting if you've, well, bumped into something. Your radar won't pick up walls, or other robots in your way, so it's an excellent way of spotting when you're stuck up against a wall.

For this task, you want your robot to drive forwards until it hits an obstacle, then go backwards and stop.



(There's also one on the back)

The simulator robot comes with 2 bump switches, they are connected to the IO pins of the 'ruggeduino', which is a board (based on the 'arduino' board) in our robotics kit that connects to various sensors.

In our physical kit, you can connect up whatever you want to these pins. Sadly this isn't how our virtual kit works, so for the meantime your robot has 2 bump switches already wired up to your Arduino, they are connected to pins 2 and 3 (as mentioned in studentrobotics.org/docs/simulator/programming/)

As always, we have extensive documentation on how to read data from the ruggeduino in our documentation: studentrobotics.org/docs/programming/sr/ruggeduinos/

As explained on the programming page, you don't need to set the pin modes, as they are already configured for you in the virtual robot. The bump switches are in INPUT mode, and as they are only connected or disconnected, you can just '`digital_read(<pin number>)`' them.

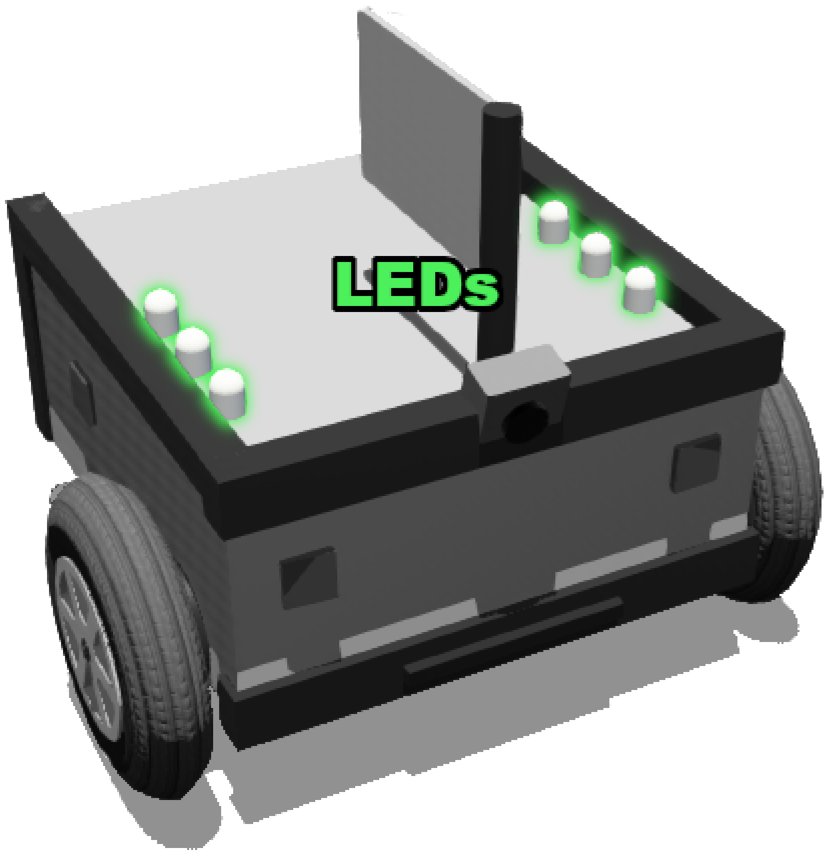
Our documentation should be your go-to place to find information about our robots, so you should learn how to read and understand it.

Now, make your robot drive forward, then stop when it hits the wall. Once it can, you can move to the next task!

Some hints with the programming:

- If you have a `R.sleep()` for more than 0.1 seconds, you're doing something wrong here.
- Look into how '`while`' loops work, if you haven't heard of them before

5. Light up the Sky 💡



Your robot has six LEDs on it. These can be handy to understand what your robot is thinking during a match, as a way to complement using the logs.

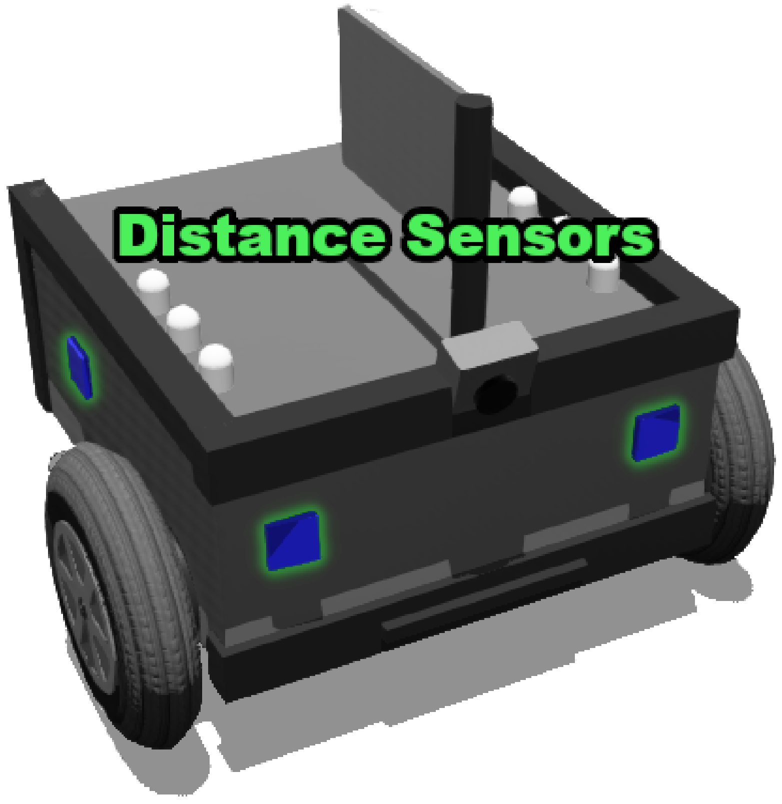
The simulator programming docs explain how to use the LEDs: studentrobotics.org/docs/simulator/programming/#leds

Modify your program from the Microswitches task so that your robot lights it's green LEDs when moving and its red LEDs when it has hit the wall.

6. Distance Sensors 👋

Distance sensors, usually ultrasonic sensors in physical robots, report the closest distance away from them in a cone perpendicular to the sensor. They can read up to 2m away, but get more noisy the further away things are. Normally they tell you the distance of the closest thing in the cone.

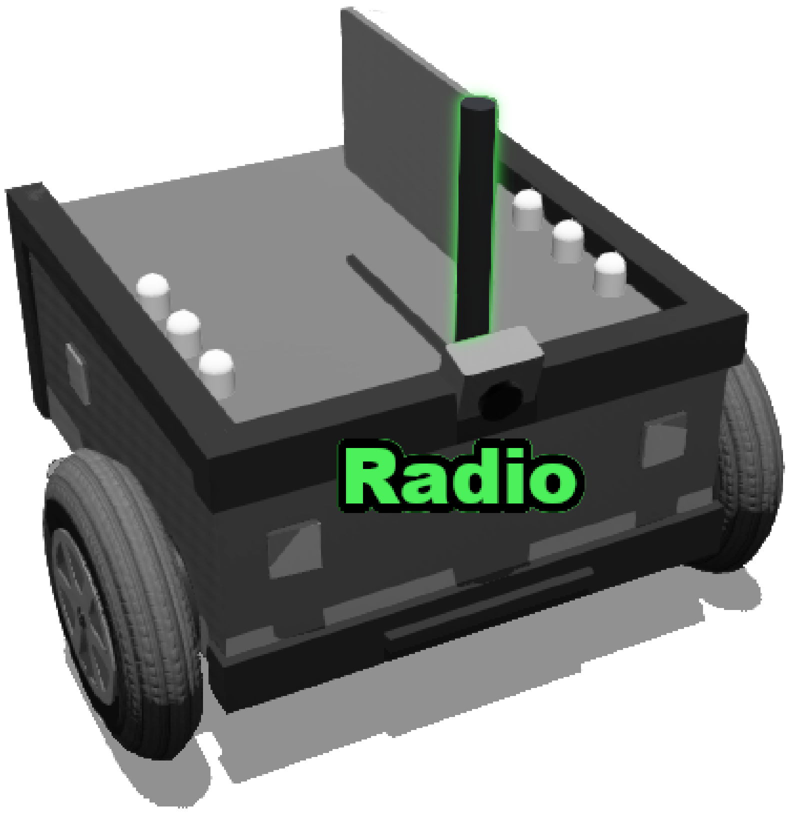
They can be used to detect when you're about to hit a wall. There are 6 in total, with 2 each on the front and back, so you can use them to get the angle of surfaces in front of you.



Our virtual robots also have the distance sensors connected up to the I/O pins of the 'ruggeduino', just like in the previous task. However these sensors give you a range of values, from 0 to 2 for distance in meters, so you will need to use `analogue_read()` instead of `digital_read()`.

To complete this task, make your robot reverse until you are 30cm away from an object (after bumping into it).

7. Radio



New this year! Your robot can get a lot of information about its surroundings by using its radio. Each central tower in the territories has a transmitter and receiver, your robot will be able to receive transmissions from a tower if it is within the tower's transmitting range. Each transmission will contain the station code, which team currently owns the territory, signal strength and relative bearing from the robot.

To capture a territory, you must be within that territory and follow the steps as outlined in the docs: studentrobotics.org/docs/programming/sr/radio/

Write some code which captures the nearest territory to your starting zone. Use the relative bearing and your previously-written microswitch code to navigate to the tower.

8. State Machine

One great way to structure your robot code is around the idea of the "states", which the code can be in. You can then express the behaviour of your robot as being in those states and the actions it takes (or things which happen to it), which causes it to move to another state.

Write a state machine so that your robot uses its LEDs as a set of traffic lights. The lights should follow the following cycle:

- Red
- Red & Yellow
- Green
- Yellow
- (back to Red)