

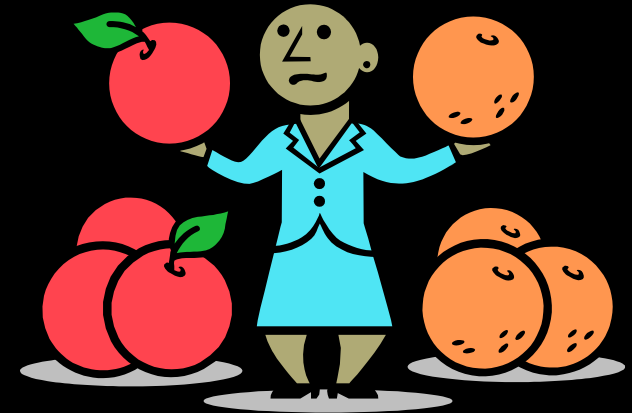
# Webots Basics



# Computer vs. Robotic Programs

## ■ Computer Programs:

- designed to **compute an answer**
- **data usually valid** when available
- **predictable** program flow
- foreseen **errors easily handled**

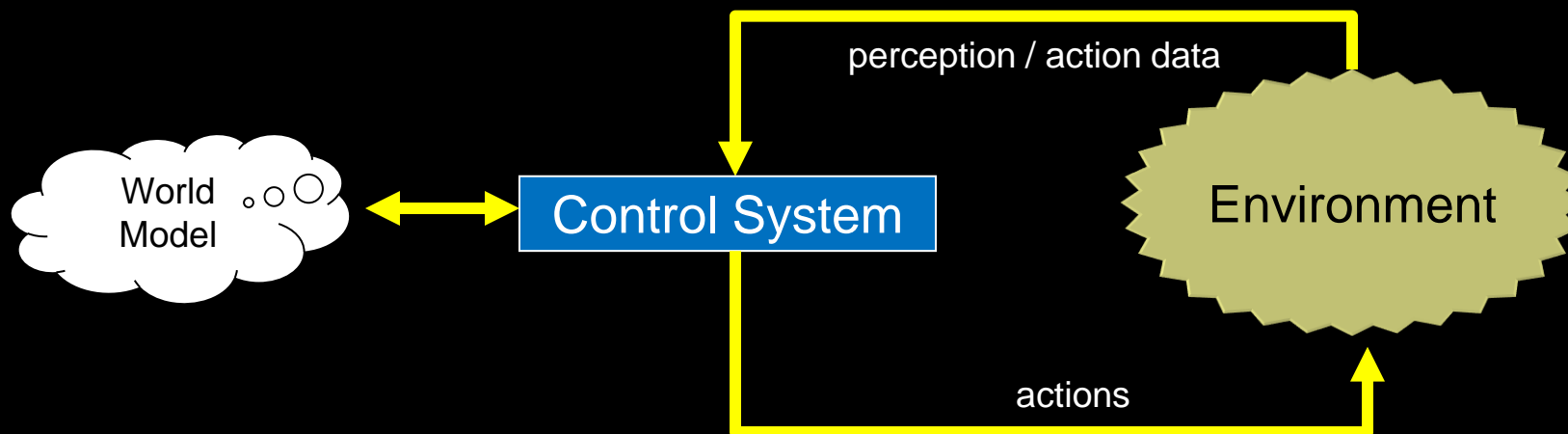
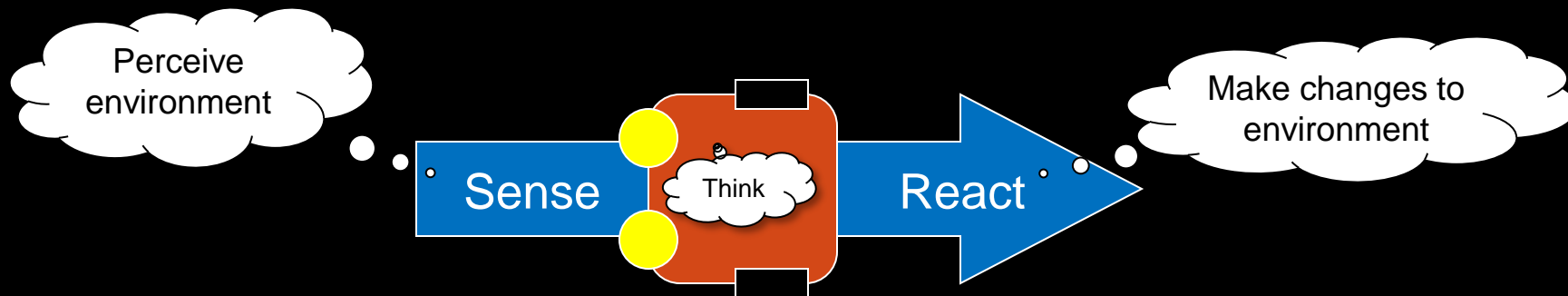


## ■ Robotic Programs:

- designed to **react to achieve goals**, not “an answer”
- sensors often produce **invalid data** (or data missing)
- **unpredictable situations** due to dynamic environment (e.g., unforeseen obstacles, wrong or missing sensor data, communication outages, hardware failure, etc..)
- program must degrade gracefully in **difficult situations**

# Robot Processing

- Here is the basic “flow of control” commonly used:



# Robot Control - Pseudocode

```
MyRobotController() {
```

```
    Set up sensors and motors
```

```
    WHILE (TRUE) {
```

```
        detectLeft = read left sensor;  
        detectRight = read right sensor;
```

```
        IF (detectLeft) THEN  
            turn = right;  
        ELSE IF (detectRight) THEN  
            turn = left;  
        ELSE  
            turn = none;
```

```
        IF (turn == left) THEN  
            turn leftMotor on backwards  
            turn rightMotor on forward  
        ELSE IF (turn == right) THEN  
            turn rightMotor on backwards  
            turn leftMotor on forward  
        ELSE  
            turn rightMotor on forward  
            turn leftMotor on forward
```

```
    }
```

```
}
```

Each robot program runs in an infinite loop.

## Initialize

Set up the motors, sensors, variables etc...

## Sense

Read sensors to determine if a collision is detected on the left or right sensor.

## Think

Decide whether robot should go straight or turn away.

## React

Turn left, turn right, or move forward accordingly.

# Robot Control – JAVA code

Informs JAVA compiler where to find libraries of various functions that you will use in your program. You will add lots more **import** statements.

All classes are given a name which must match the filename ... (**LabController.java** in this case)

```
import com.cyberbotics.webots.controllers.*;
```

```
public class LabController {
```

```
    /* Declare static constants & variables here */
```

```
    public static void main(String[] args) {
```

```
        Robot robot = new Robot();           // Creates a generic Robot object
```

```
        int timeStep = (int) Math.round(robot.getBasicTimeStep());
```

```
        /* INITIALIZE ... variables, sensors, motors, etc... */
```

```
        while(robot.step(timeStep) != -1) {
```

```
            /* SENSE ... Read sensors */
```

```
            /* THINK ... Make decision as to what to do */
```

```
            /* REACT ... Move motors, head, arms, etc... */
```

```
        }
```

```
    }
```

```
}
```

**main** function is starting point of your program.

Simulation time-step in (ms).  
You won't change this.

Loops forever (until PAUSED or STOPPED)

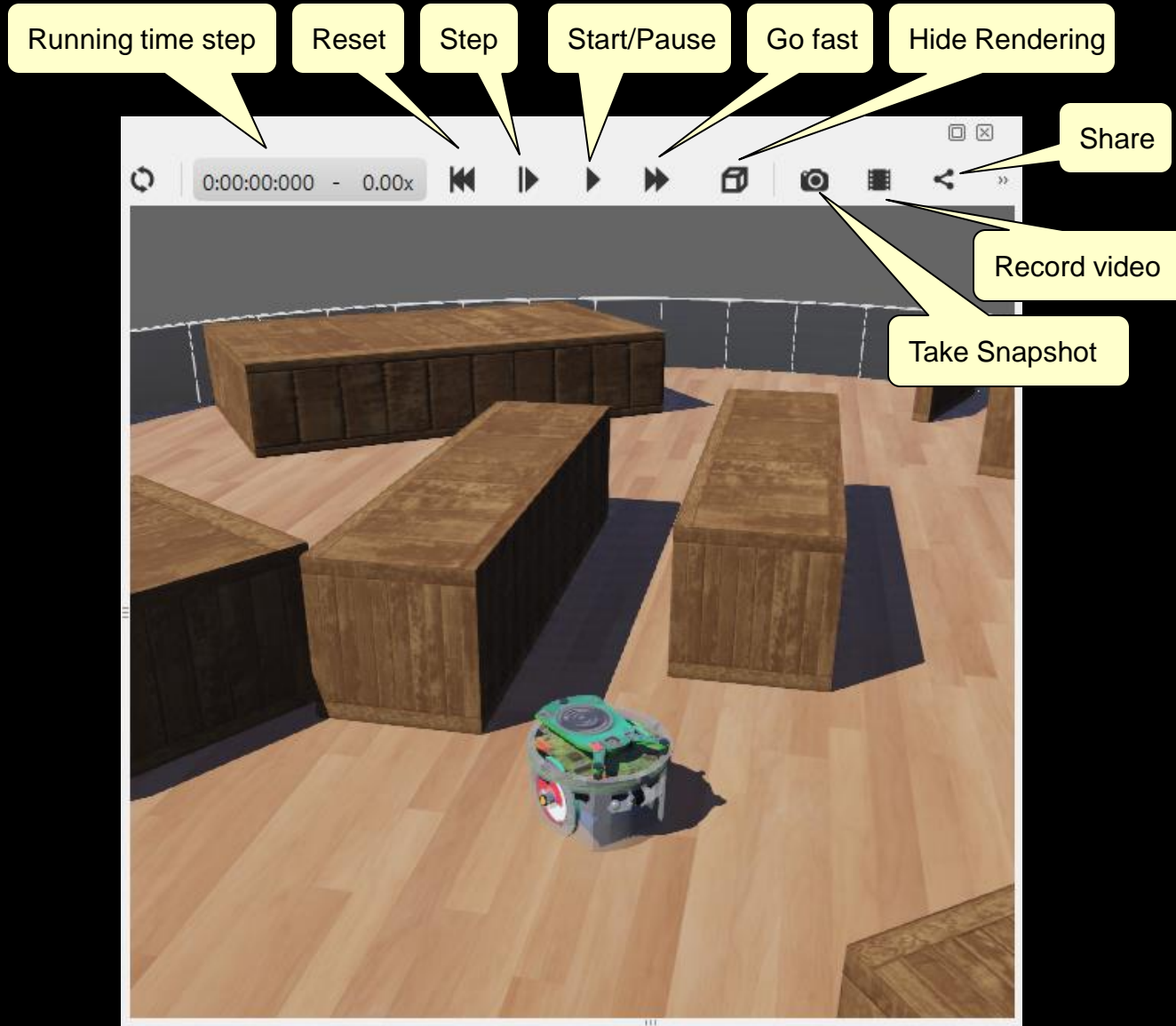
# Webots Interface – main GUI

The screenshot displays the Webots R2023a interface. The top menu bar includes File, Edit, View, Simulation, Build, Overlays, Tools, and Help. The toolbar below contains icons for various functions like opening files, saving, and running simulations. The main area is divided into three panels:

- Left Panel (IMPORTABLE EXTERNPROTO):** A list of objects available for import, including WorldInfo, Viewpoint, TexturedBackground, TexturedBackgroundLight, CircleArena "circle arena", E-puck "e-puck", and multiple instances of WoodenBox. A callout bubble says: "Edit the environment objects here."
- Central Panel (3D View):** A 3D simulation of a robot (E-puck) in a virtual environment with wooden blocks and a blue sky. A callout bubble says: "Watch your robot being simulated Here."
- Right Panel (Lab1Controller.java):** A code editor showing the Java code for the robot's controller. A callout bubble says: "Write your code In this editor."

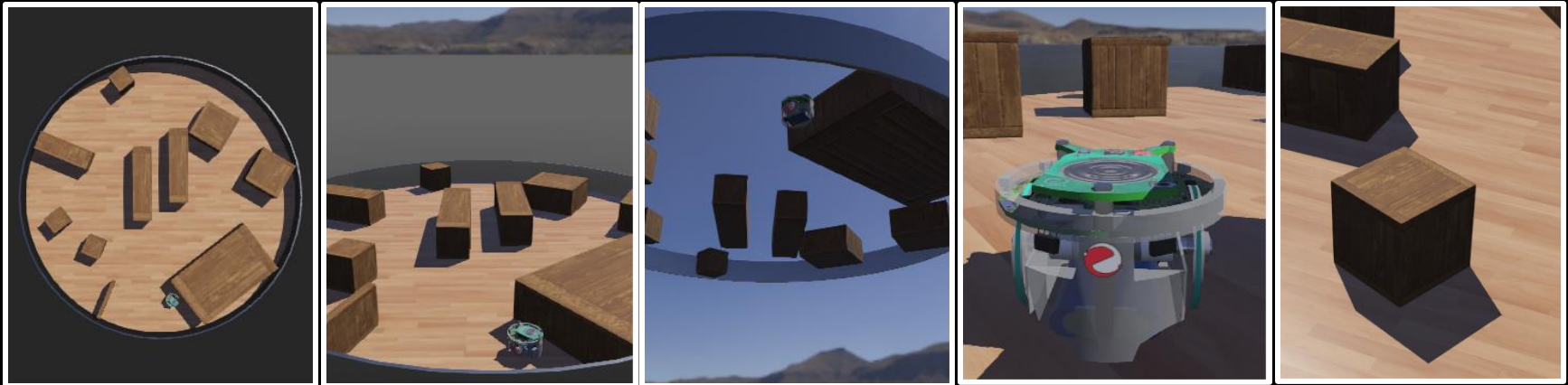
At the bottom, there is a Console panel labeled "Console - All" with a callout bubble saying: "See compile errors and debug print statements here."

# Webots Interface — Simulation Controls





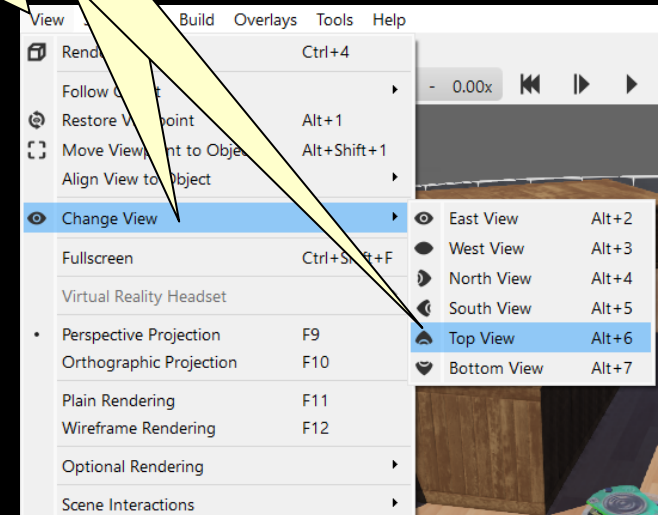
# Webots Interface — Changing Viewpoint



Use the mouse to change viewpoint:

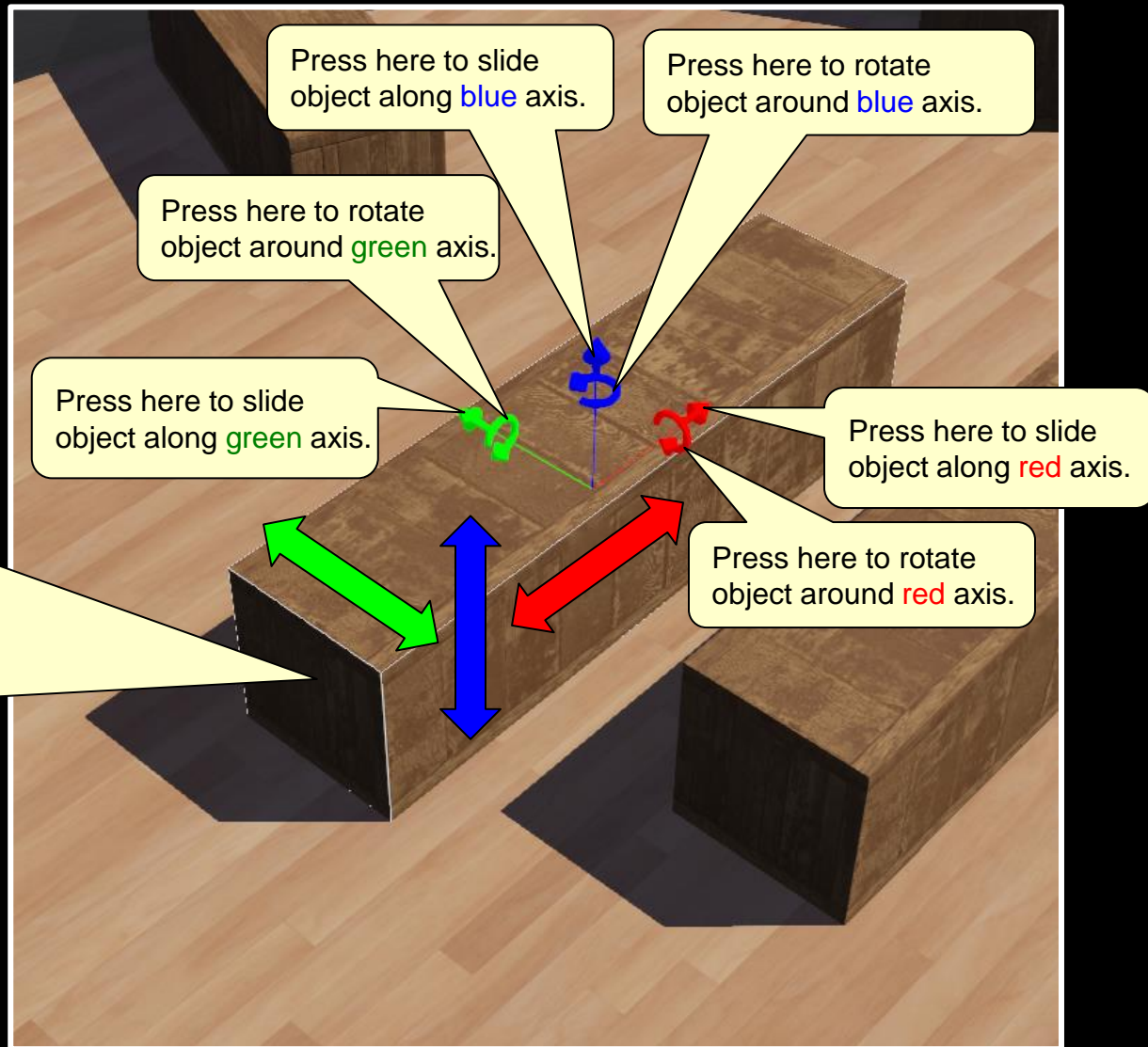
**Roll:** left press + up/down  
**Spin:** left press + left/right  
**Translate:** right press + up/down/left/right  
**Zoom:** scroll up/down  
**Select:** left click  
**Menu:** right click

View menu has more options





# Webots Interface — Moving World Objects

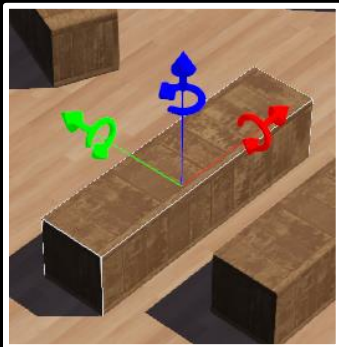
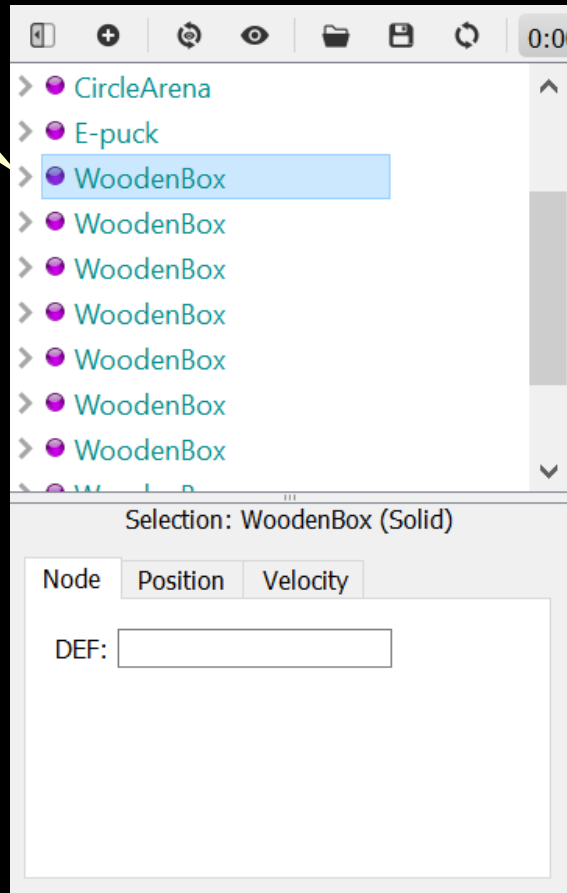


Click anywhere on object to select it, then press and hold left mouse button on one of 6 places to change object.

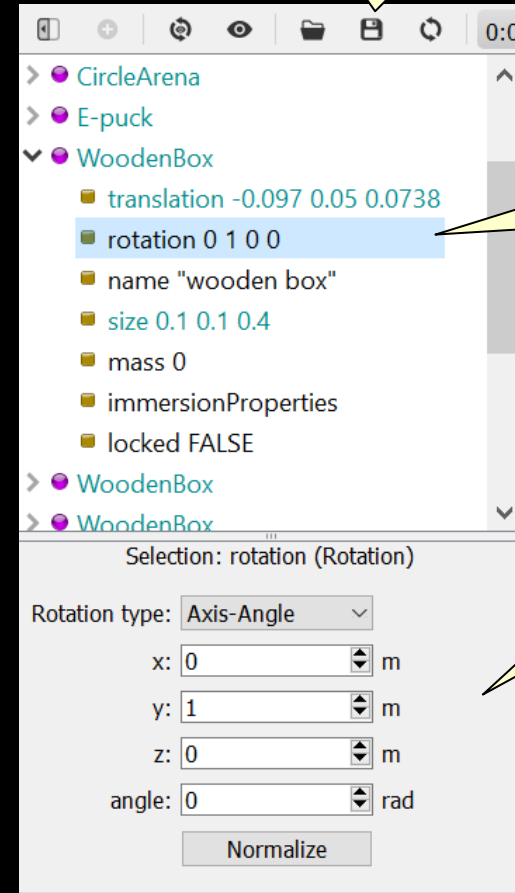
# Webots Interface — Editing World Objects

1. Click on the > to expand or collapse one object's attribute view.

Objects are all listed here in the **Scene Tree**.



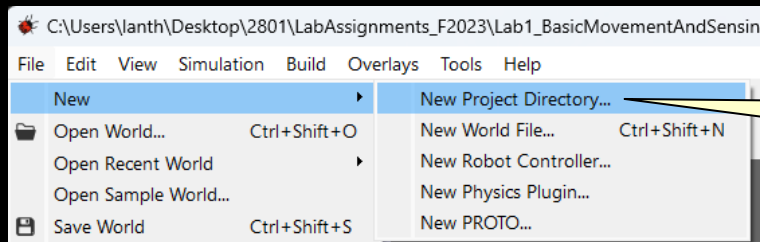
4. Save your changes.



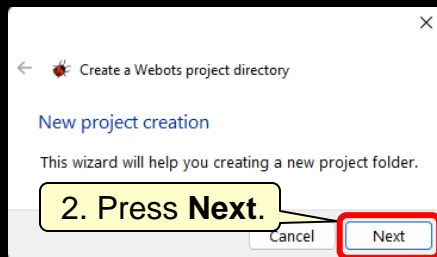
2. Select an attribute.

3. Change its values.

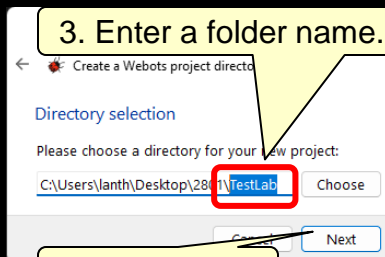
# Webots Interface — Making New World



1. Select **New Project Directory...** from the **File/New** menu

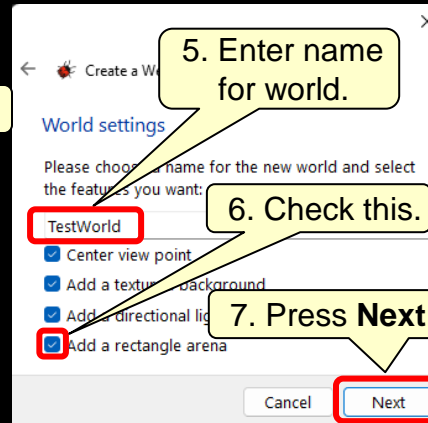


2. Press **Next**.



3. Enter a folder name.

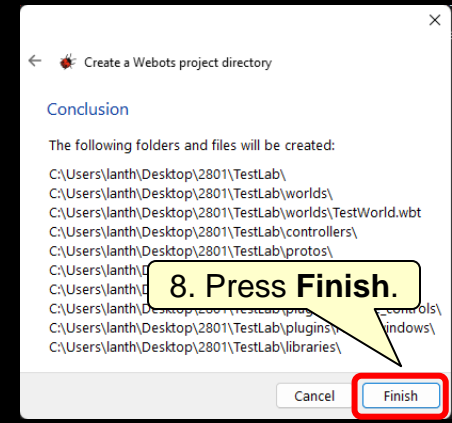
4. Press **Next**.



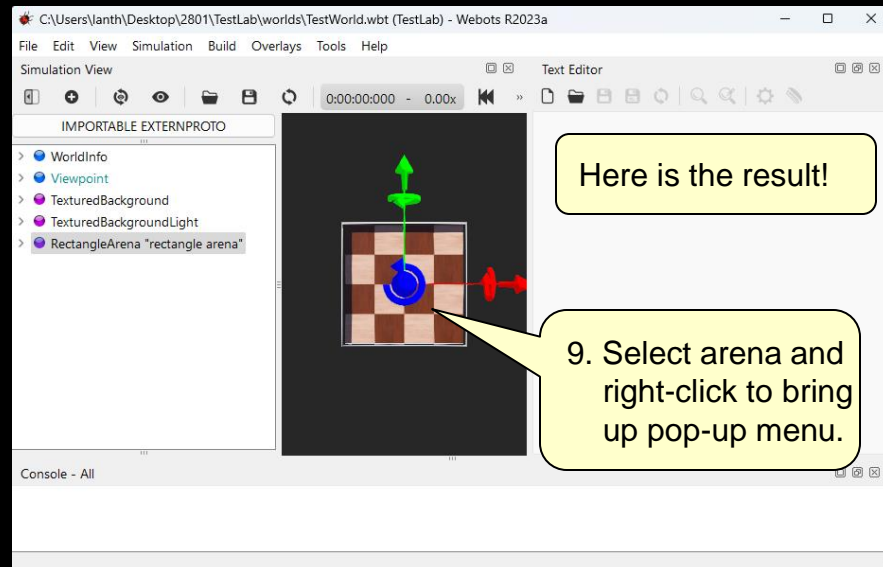
5. Enter name for world.

6. Check this.

7. Press **Next**.



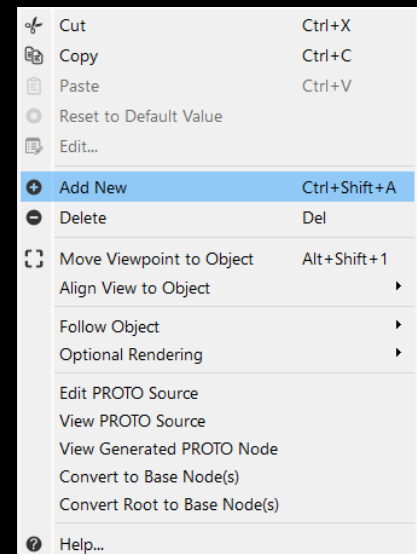
8. Press **Finish**.



Here is the result!

9. Select arena and right-click to bring up pop-up menu.

10. Select **Add New** to add objects and robots. The menu can take up to 10 seconds to appear!



# Webots Interface – Adding an object

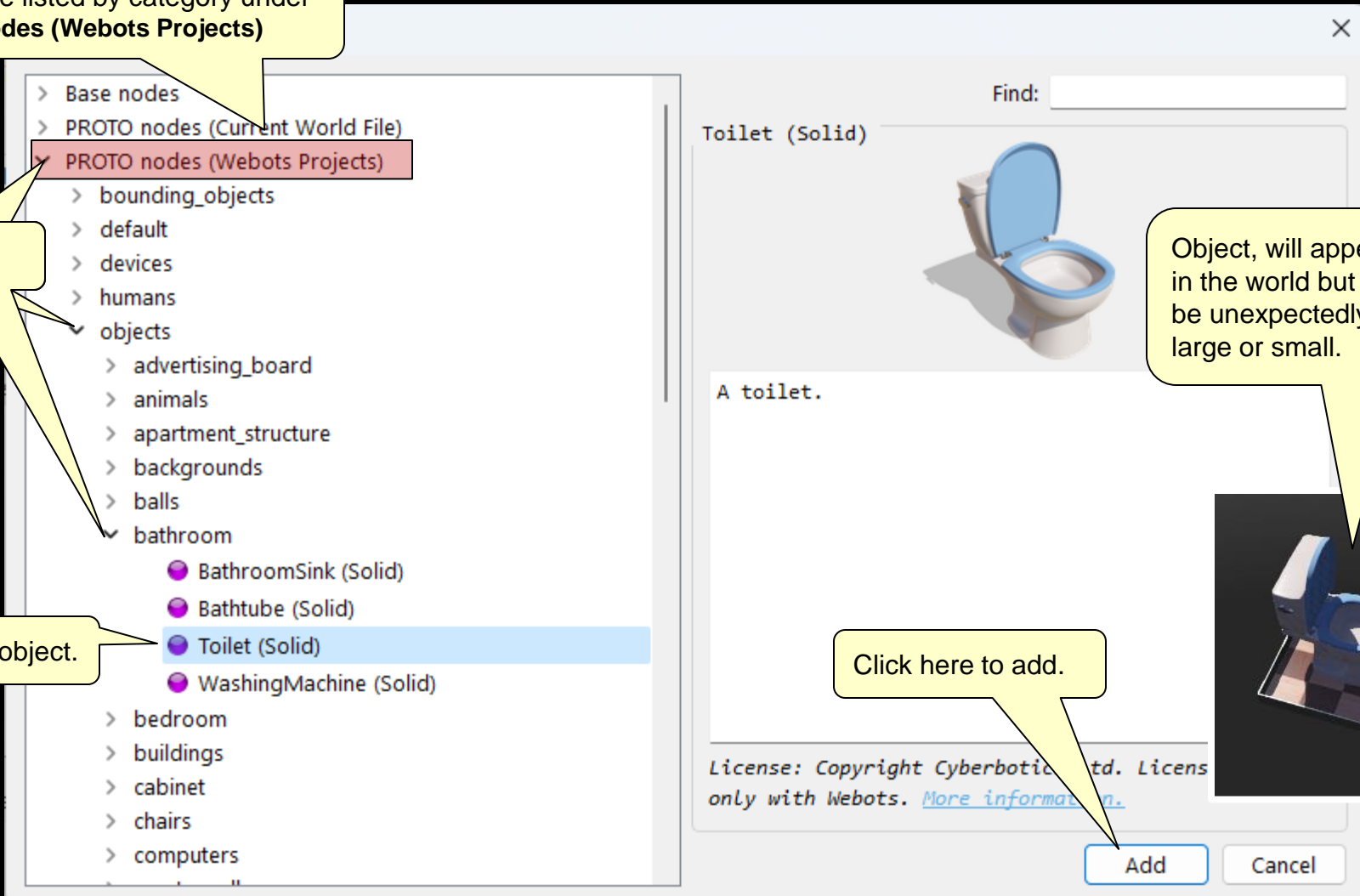
Objects are listed by category under PROTO nodes (Webots Projects)

Press > to expand.

Select object.

Click here to add.

Object, will appear in the world but may be unexpectedly large or small.



# Webots Interface – Resizing an Object

C:\Users\lanth\Desktop\2801\TestLab\worlds\TestWorld.wbt (TestLab) - Webots R2023a

File Edit View Simulation Build Overlays Tools Help

Simulation View

0:00:00:000 - 0.00x

IMPORTABLE EXTERNPROTO

RectangleArena "rectangle arena"

- translation 0 0 0
- rotation 0 0 1 0
- name "rectangle arena"
- contactMaterial "default"
- floorSize 3 5**
- floorSize 3 5

Selection: floorSize (Vector2)

x: 3 m

y: 5 m

Console - All

Some objects, such as the toilet, are fixed size and cannot be grown nor shrunk.

1. Click on floor to select it

2. Select **floorSize** for RectangleArena.

3. Increase or decrease **x & y** size in meters.

4. Floor will grow immediately.

Don't forget to **SAVE** your world after making your changes.



# Webots Interface — Adding a Robot

Robots are listed under under **PROTO nodes (Webots Projects)**

Select robot.

The "Spot" is a dog robot developed by Boston Dynamics. Original model by Greg McKechnie.

Documentation: <https://www.cylindric.com/spot>

License: Creative Commons 4.0

[More information.](#)

Click here to add.

Add Import... Cancel

Black screen.

Sometimes robot camera displays comes up. Drag here to resize smaller.

Robot's camera display is smaller now.

Don't forget to **SAVE** your world after adding your robot.

# Webots Interface – Controller Code

**controller** attribute specifies program that robot will run.

Robot loads with sample code and may start moving if simulation is running.

Click here to select a different program to run on the robot.

Click here to see the code in the editor.

Here is the code that the robot is running.

```
1 #include <webots/motor.h>
2 #include <webots/robot.h>
3 #include <math.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #define NUMBER_OF_LEDS 8
8 #define NUMBER_OF_JOINTS 12
9 #define NUMBER_OF_CAMERAS 5
10
11 // Initialize the robot's information
12 static WbDeviceTag motors[NUMBER_OF_JOINTS];
13 static const char *motor_names[NUMBER_OF_JOINTS] =
14     "front left shoulder abduction motor", "front left
15     shoulder adduction motor", "front right shoulder abduction motor", "front right
16     shoulder adduction motor", "rear left shoulder abduction motor", "rear left
17     shoulder adduction motor", "rear right shoulder abduction motor", "rear right
18     shoulder adduction motor", "front left hip flexion motor", "front left hip
19     extension motor", "front right hip flexion motor", "front right hip extension
20     motor", "rear left hip flexion motor", "rear left hip extension motor", "rear
21     right hip flexion motor", "rear right hip extension motor", "front left knee
22     flexion motor", "front left knee extension motor", "front right knee flexion
23     motor", "front right knee extension motor", "rear left knee flexion motor",
24     "rear left knee extension motor", "rear right knee flexion motor", "rear right
25     knee extension motor";
26
27 static WbDeviceTag cameras[NUMBER_OF_CAMERAS];
28 static const char *camera_names[NUMBER_OF_CAMERAS] =
29     "front left camera", "front right camera", "rear left camera", "rear right
30     camera";
```



# Webots Interface – Your Controller

1. Select **New Robot Controller...** from the **File/New** menu

2. Press **Next.**

3. Select **Java.**

4. Press **Next.**

5. Enter program name (i.e., Java class).

6. Press **Next.**

7. Press **Finish.**

8. Select controller attribute for this robot.

9. Press **Select...** to change controller program for this robot, then choose from dialog box.

10. Choose from dialog box.

11. Press **Edit** to have program appear in text editor.

12. Editor will show a new controller template program now.

13. Close the old controller, or keep it open as a reference. It is not being used by the robot.

# Webots Interface – Editing Code

Press here (or use Cntrl-S) to save your code.

Press here to compile and load your code onto the robot.

Press here to create a new source code .java file.

Press here to open other source code .java files.

ALWAYS put your name and student number at the top of your programs.

Double-click on error line to go there in the editor.

Compile errors appear in the console window. If a compile error occurs, the code is NOT loaded onto the robot, so the robot is running its previous code. Once compiled and loaded, press **Reset** in the dialog box that appears. Press the play button in the simulation view to run it.,

```
File
C:\Users\lan\...
TestWorld.wbt (TestLab) - Webots R2023a
TestController.java
Auth: Mark Lanthier (SN: 100100100)
import com.robotics.webots.controller.Robot;
// your controller.
// to initialize and how to run your controller.
public class TestController {
    public TestController(String[] args) {
        // create the Robot instance.
        Robot robot = new Robot()

        // get the time step of the current world.
        int timeStep = (int) Math.round(robot.getBasicTimeStep());

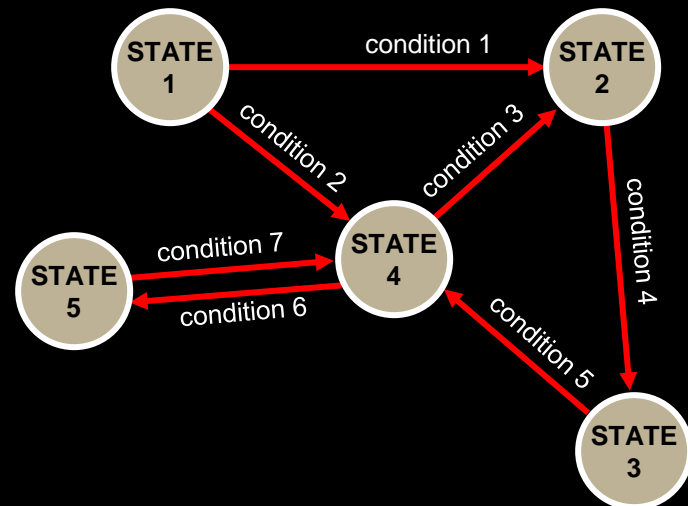
        while (robot.step(timeStep) != -1) {
            // Sense
            // Think
            // React
        }
    }
}
```

Console - All

```
javac -Xlint -classpath C:\Program Files\Webots\lib\controller\java\Controller.jar;. TestController.java
TestController.java:11: error: ';' expected
    Robot robot = new Robot()
                        ^
1 error
Nothing to be done for build targets.
```

# State Machines

- A robot can be in various **states** at any time
  - e.g., STOPPED, MOVING\_FORWARD, SPINNING\_LEFT, etc...
- A **state machine** is a diagram that explains how a robot should move from one state to another.
  - Each state indicates what the robot is doing at any moment in time.
  - A robot **transitions** from one state to another based on a condition (which is often sensor input).
- We will be using state machines for the first 3 labs so make sure that you understand them.



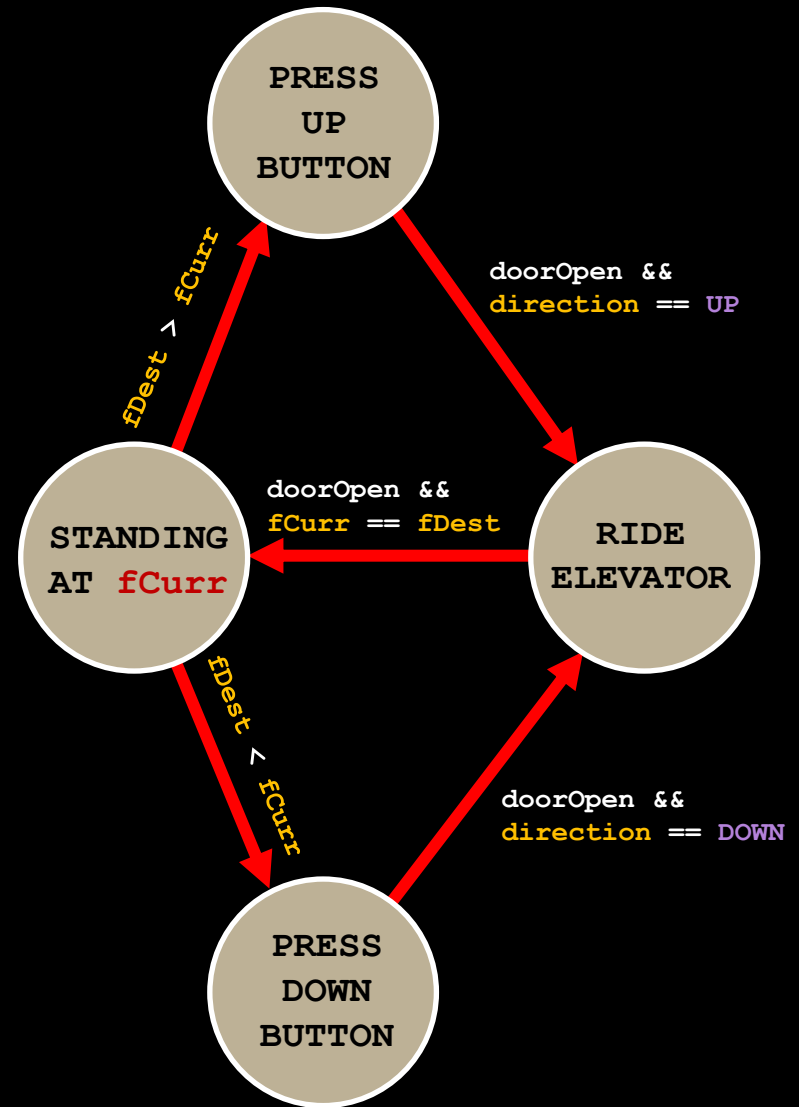
# State Machine - Elevator

- Here is an example of a state machine for using an elevator:

```
// Variables needed to transition
short    fCurr      = 0;
short    fDest      = 5;
byte     direction   = UP;
boolean  doorOpen    = false;

// Various states
static final byte STANDING_AT_FCURRE = 0;
static final byte PRESS_UP_BUTTON    = 1;
static final byte PRESS_DOWN_BUTTON  = 2;
static final byte RIDE_ELEVATOR      = 3;

// Direction constants
static final byte UP = 0;
static final byte DOWN = 1;
```



# State Machine – Elevator (Code)

```
// Variables needed to transition
short    fCurr; // current floor
short    fDest; // destination floor
byte     direction = UP; // UP or DOWN
boolean  doorOpen = false;
```

```
// Various states
static final byte STANDING_AT_FCURR = 0;
static final byte PRESS_UP_BUTTON   = 1;
static final byte PRESS_DOWN_BUTTON = 2;
static final byte RIDE_ELEVATOR     = 3;
```

```
// Direction constants
static final byte UP = 0;
static final byte DOWN = 1;
```

```
byte state = STANDING_AT_FCURR;
fCurr = 0; // any floor
fDest = 12; // any floor

while(true) {
    // SENSE
    // THINK
    // REACT
}
```

```
// SENSE
doorOpen = checkIfDoorOpen();
direction = checkElevatorDir();
```

There will be one IF statement for each arrow in the state machine.

Our SWITCH statements will be like nested IF statements.

```
// REACT
switch(state) {
    case STANDING_AT_FCURR :
        // stop moving
        break;
    case PRESS_UP_BUTTON :
        // reach out and press up button
        break;
    case PRESS_DOWN_BUTTON :
        // reach out and press down button
        break;
    case RIDE_ELEVATOR :
        // get in elevator
        // stop moving
        break;
}
```

The code in each case will be the code that does the action needed for that state (e.g., stop, walk, press, etc..)

```
// THINK
switch(state) {
    case STANDING_AT_FCURR:
        if (fDest > fCurr)
            state = PRESS_UP_BUTTON;
        if (fDest < fCurr)
            state = PRESS_DOWN_BUTTON;
        break;
    case PRESS_UP_BUTTON:
        if (doorOpen && (direction == UP))
            state = RIDE_ELEVATOR;
        break;
    case PRESS_DOWN_BUTTON:
        if (doorOpen && (direction == DOWN))
            state = RIDE_ELEVATOR;
        break;
    case RIDE_ELEVATOR:
        if (doorOpen && (fCurr == fDest))
            state = STANDING_AT_FCURR;
        break;
}
```

Decide on the new state for the next time through the WHILE loop



**Start the  
Lab ...**