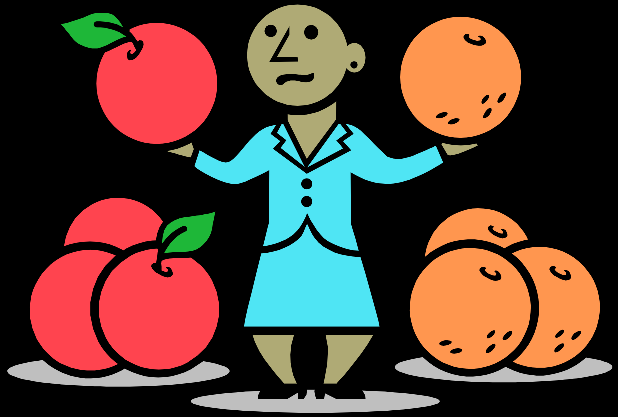
Webots Basics

Computer vs. Robotic Programs

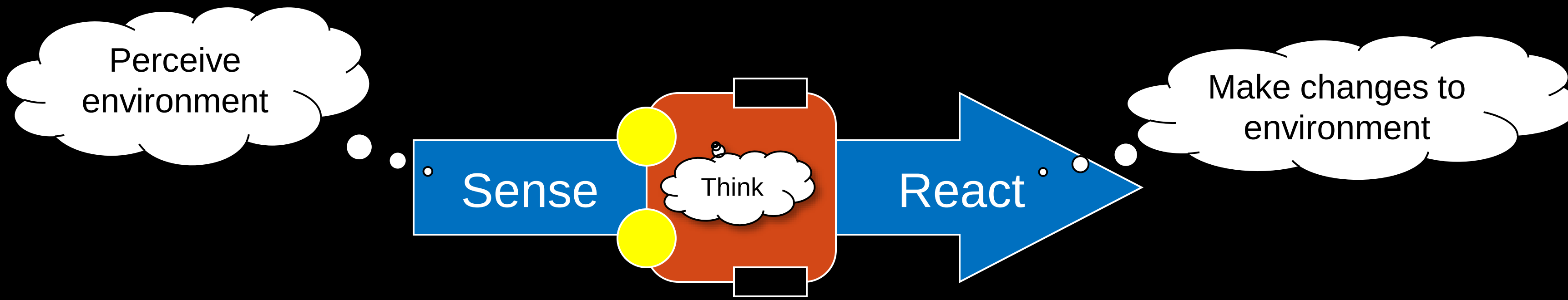
- **Computer Programs:**
  - designed to compute an answer
  - data usually valid when available
  - predictable program flow
  - foreseen errors easily handled

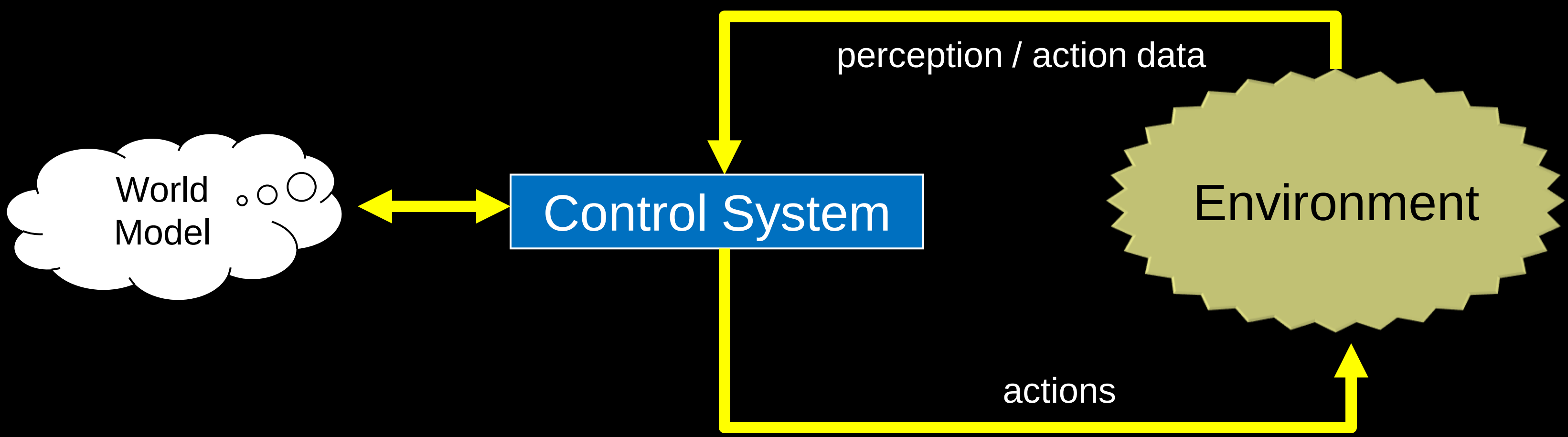


- **Robotic Programs:**
  - designed to react to achieve goals, not "an answer"
  - sensors often produce invalid data (or data missing)
  - unpredictable situations due to dynamic environment (e.g., unforeseen obstacles, wrong or missing sensor data, communication outages, hardware failure, etc..)
  - 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)

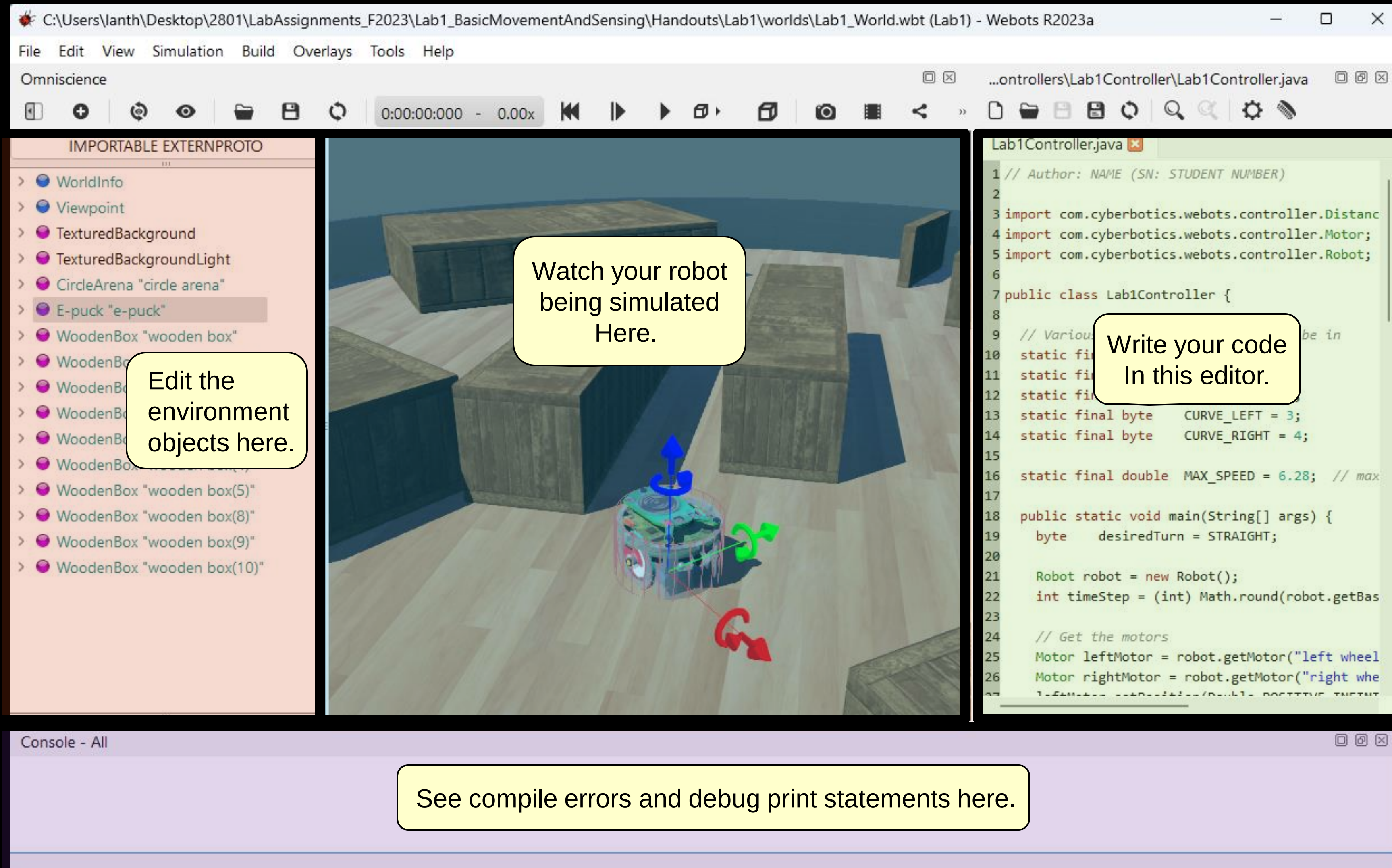
```
import com.cyberbotics.webots.controller.Robot;

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.

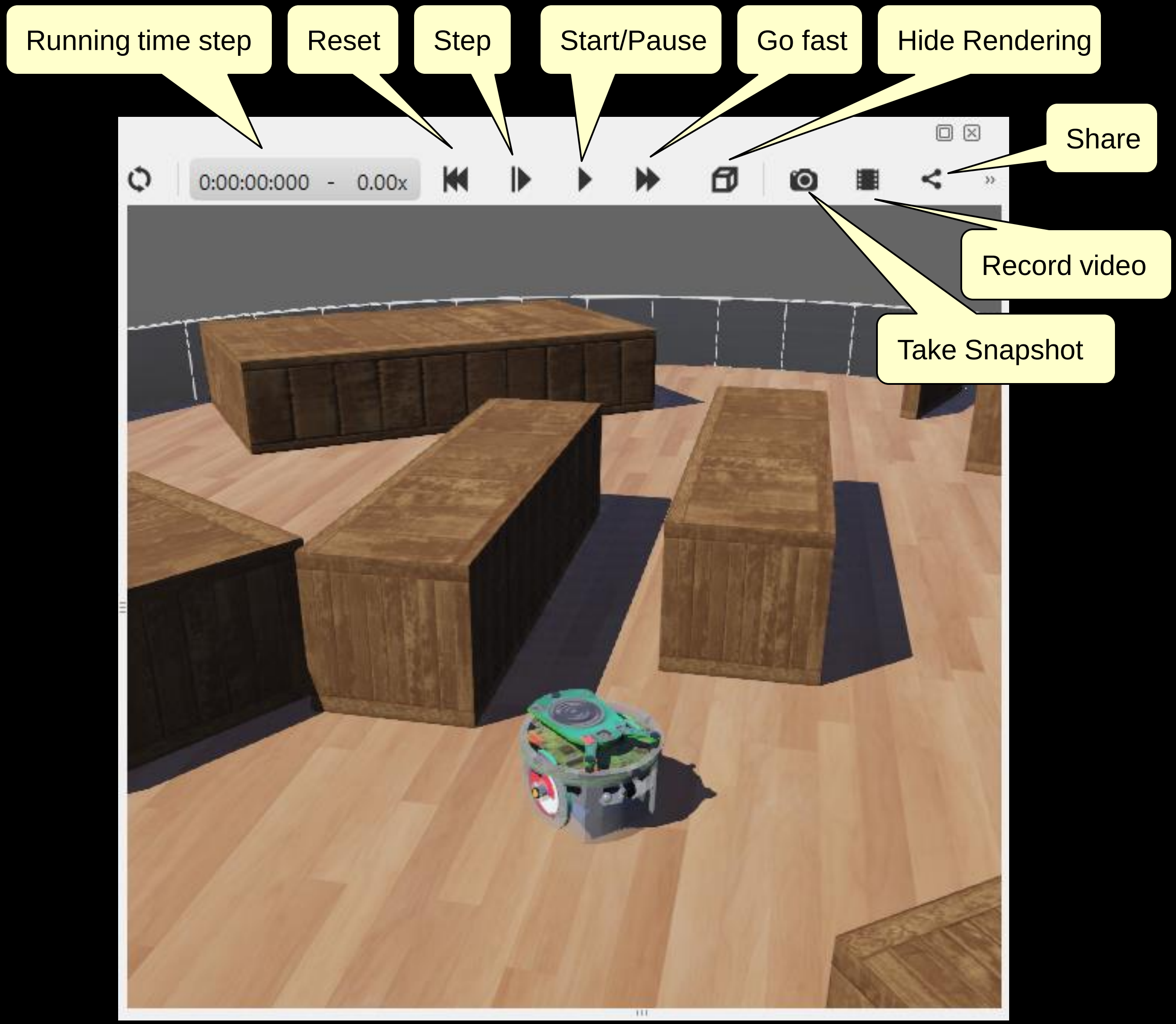
Simulation time-step in (ms). You won't change this.

Loops forever (until PAUSED or STOPPED)

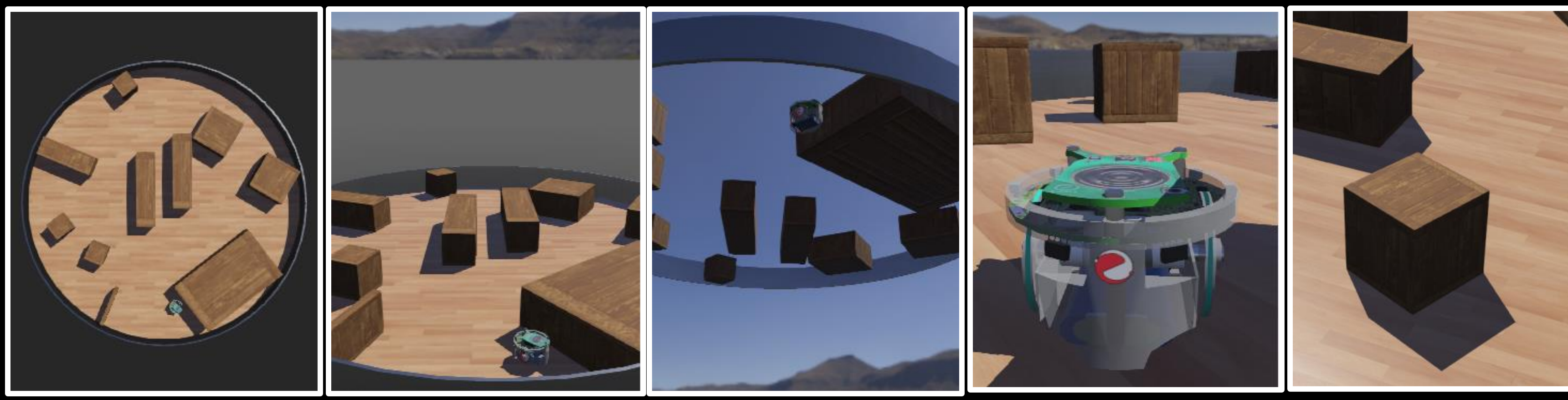
Webots Interface – main GUI



Webots Interface – Simulation Controls



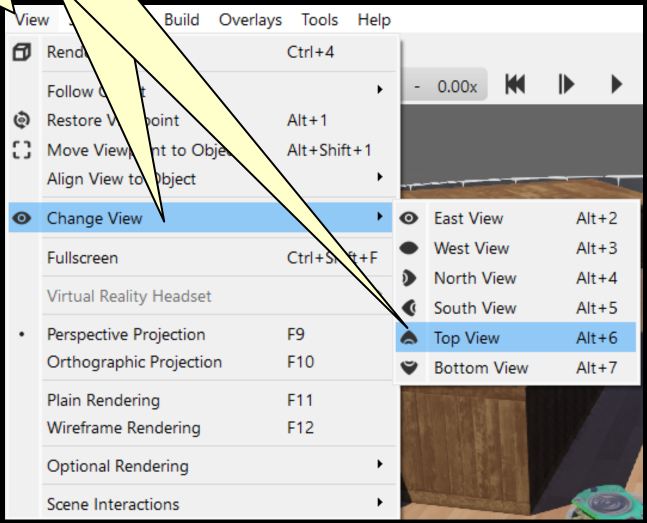
Webots Interface – Changing Viewpoint



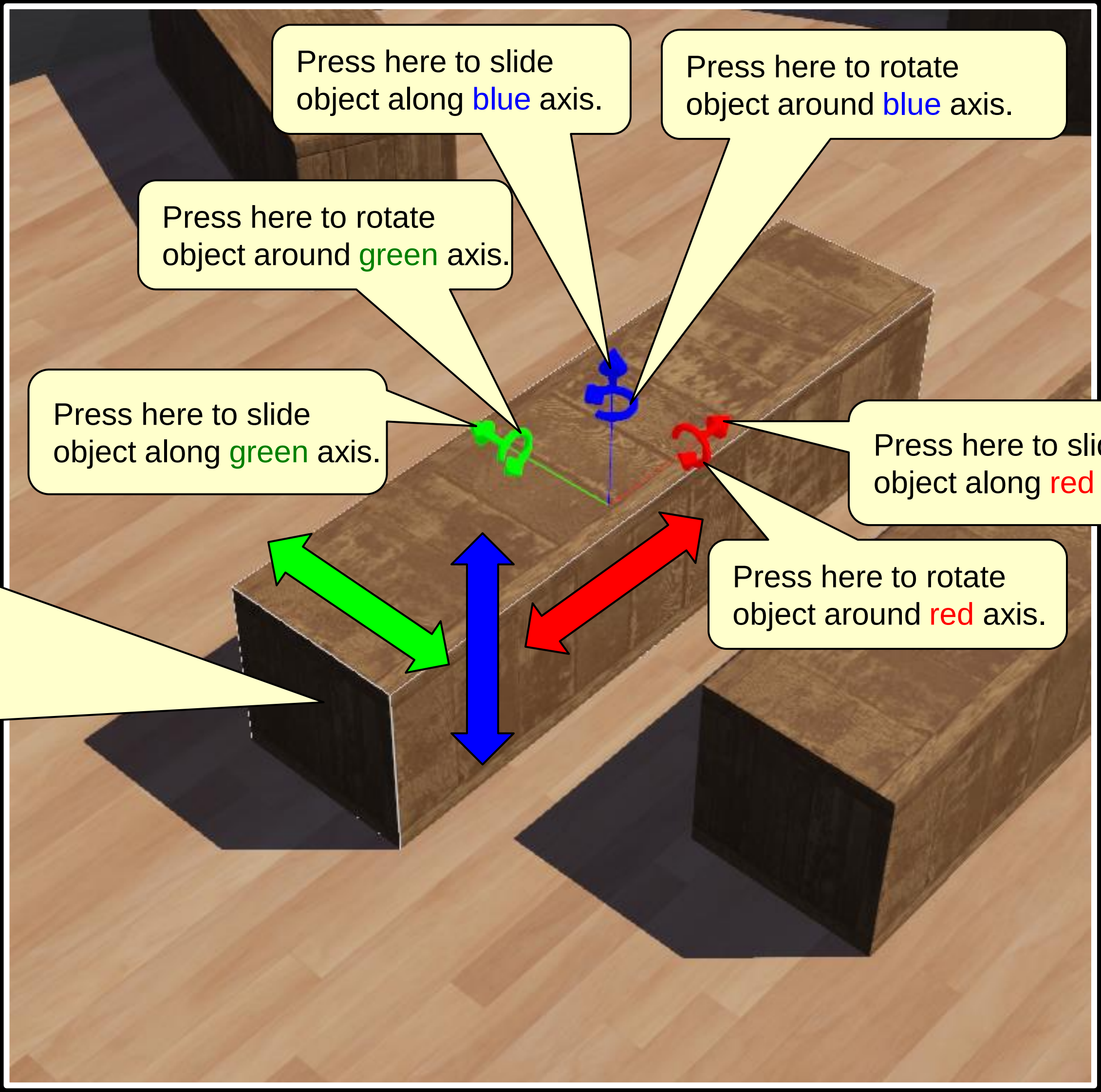
Use the mouse to change viewpoint:

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

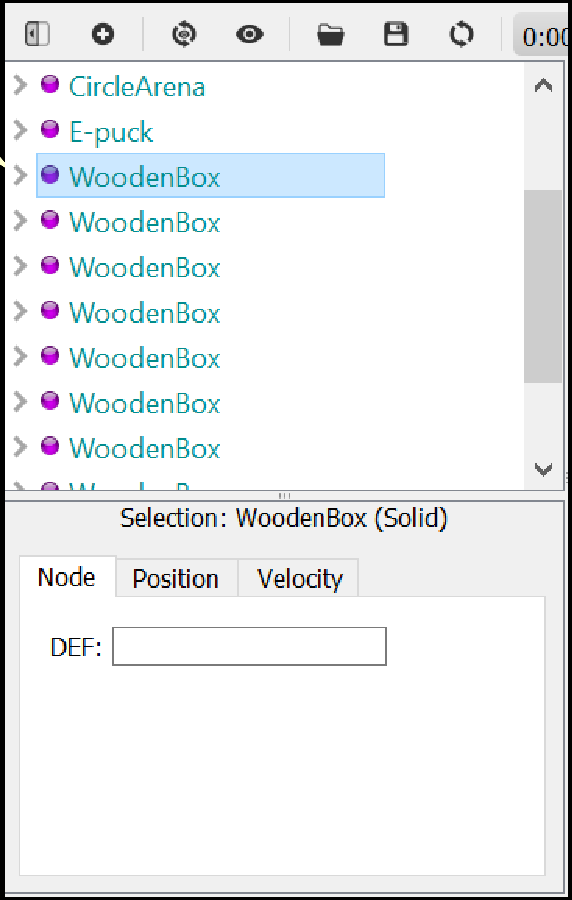


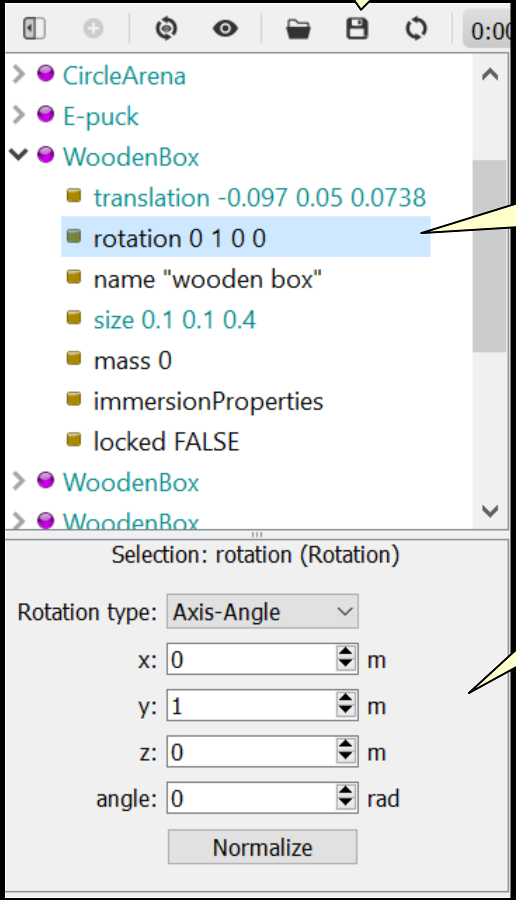
Click anywhere on object to select it, then press and hold left mouse button on one of 6 places to change object.

Webots Interface – Editing World Objects

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

4. Save your changes.

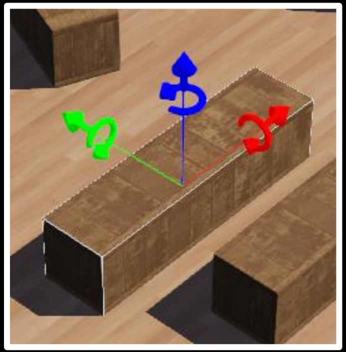




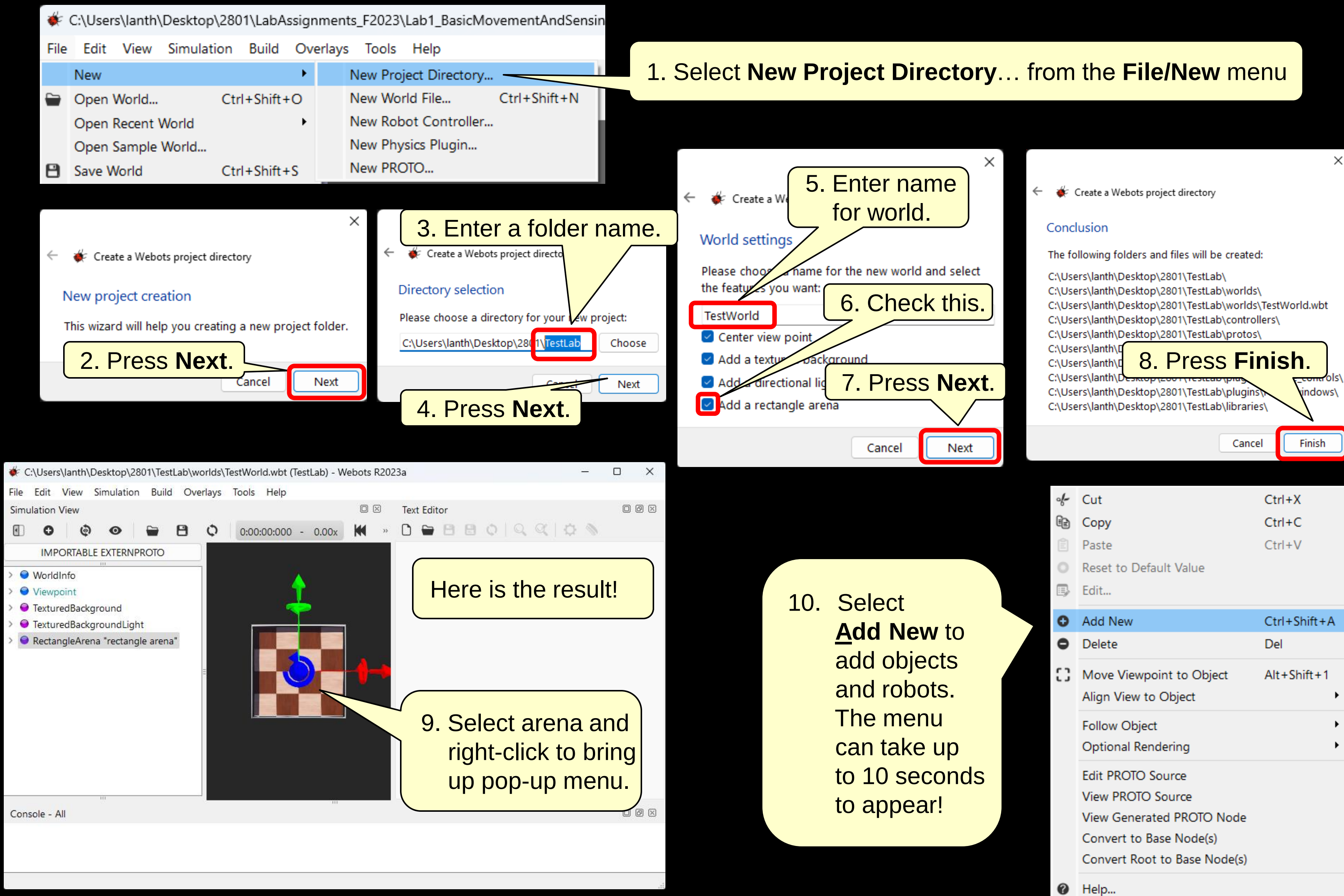
2. Select an attribute.

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

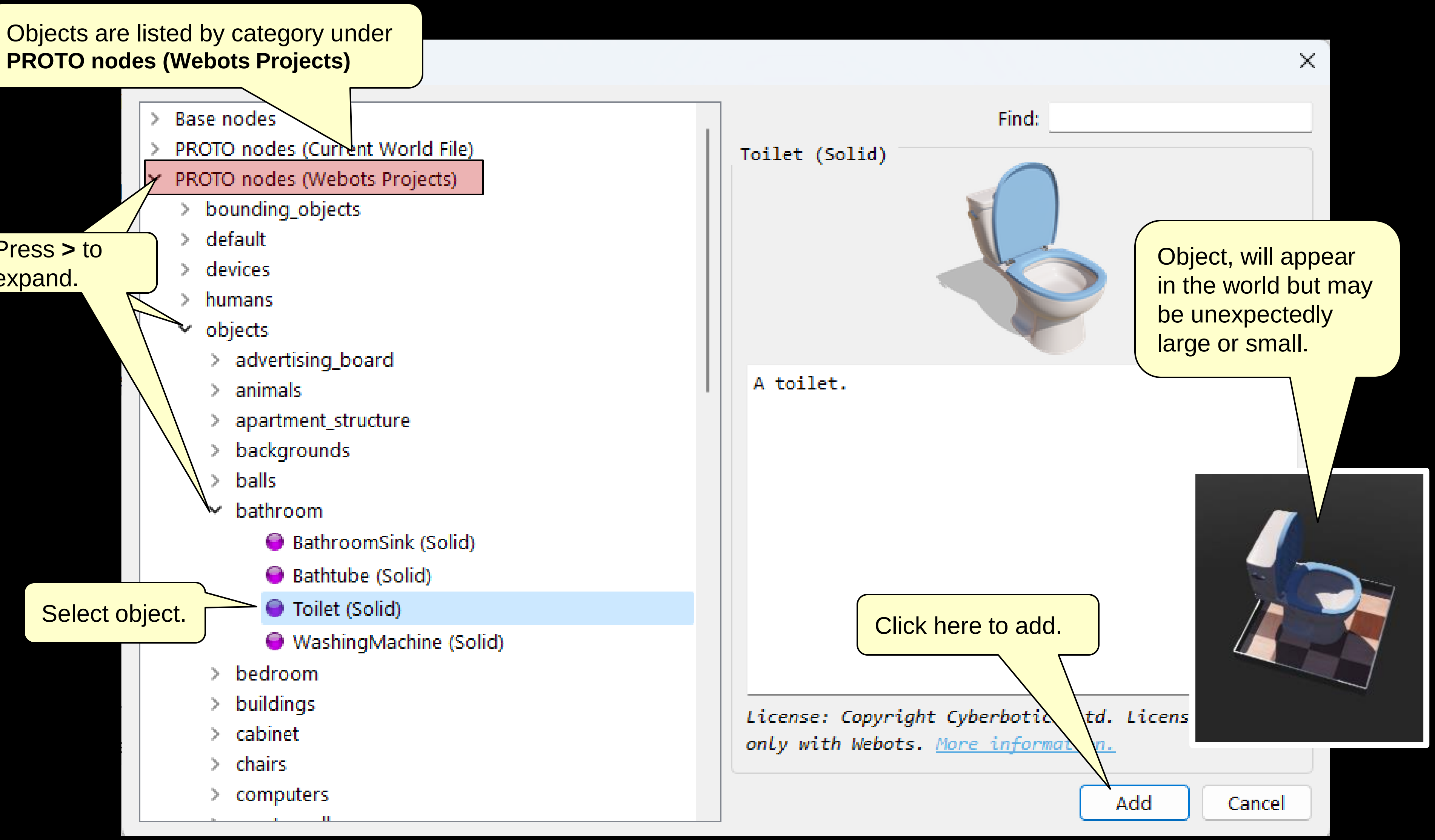
3. Change its values.



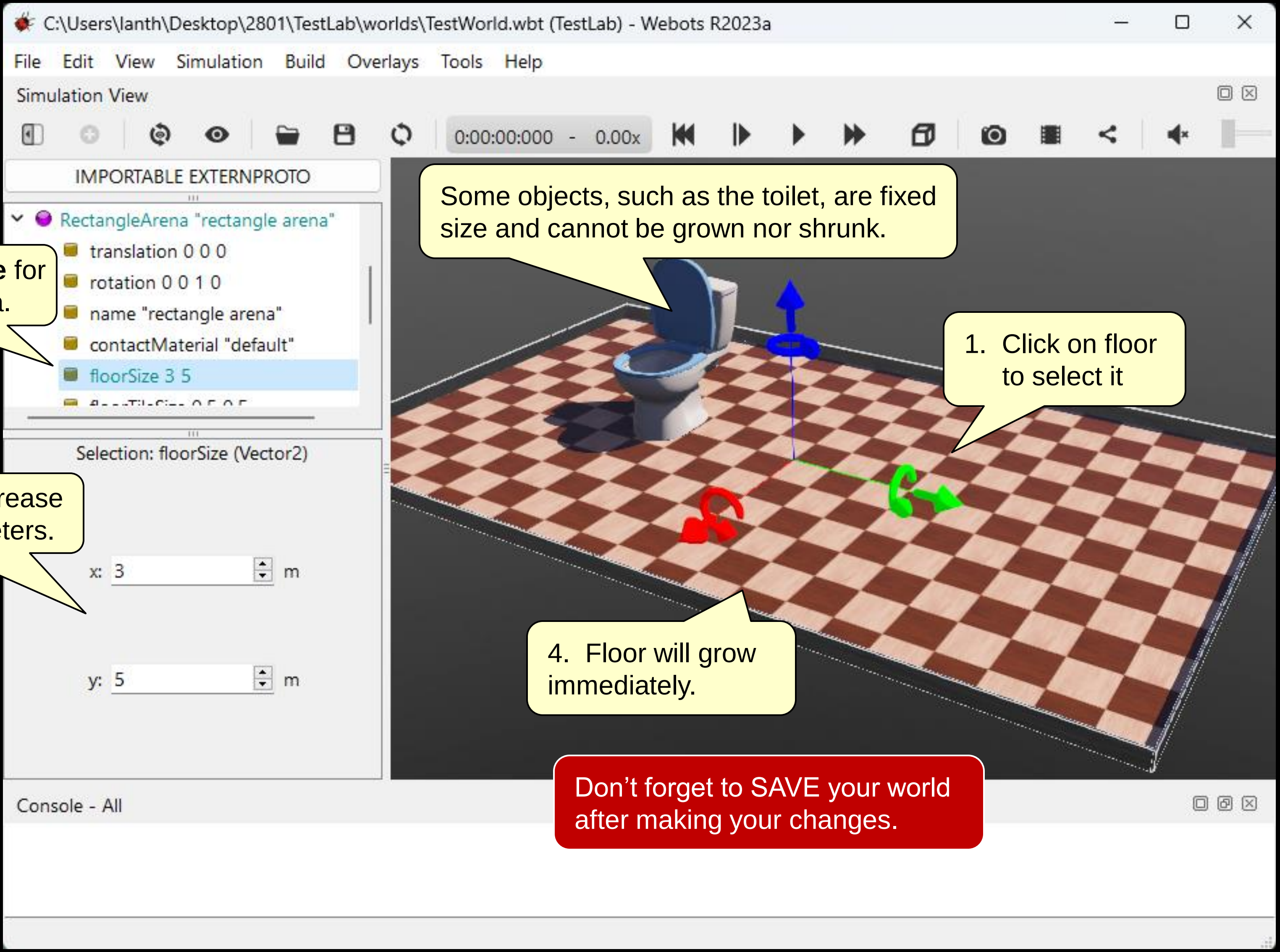
Webots Interface – Making New World



Webots Interface – Adding an object



Webots Interface – Resizing an Object



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

1. Click on floor to select it

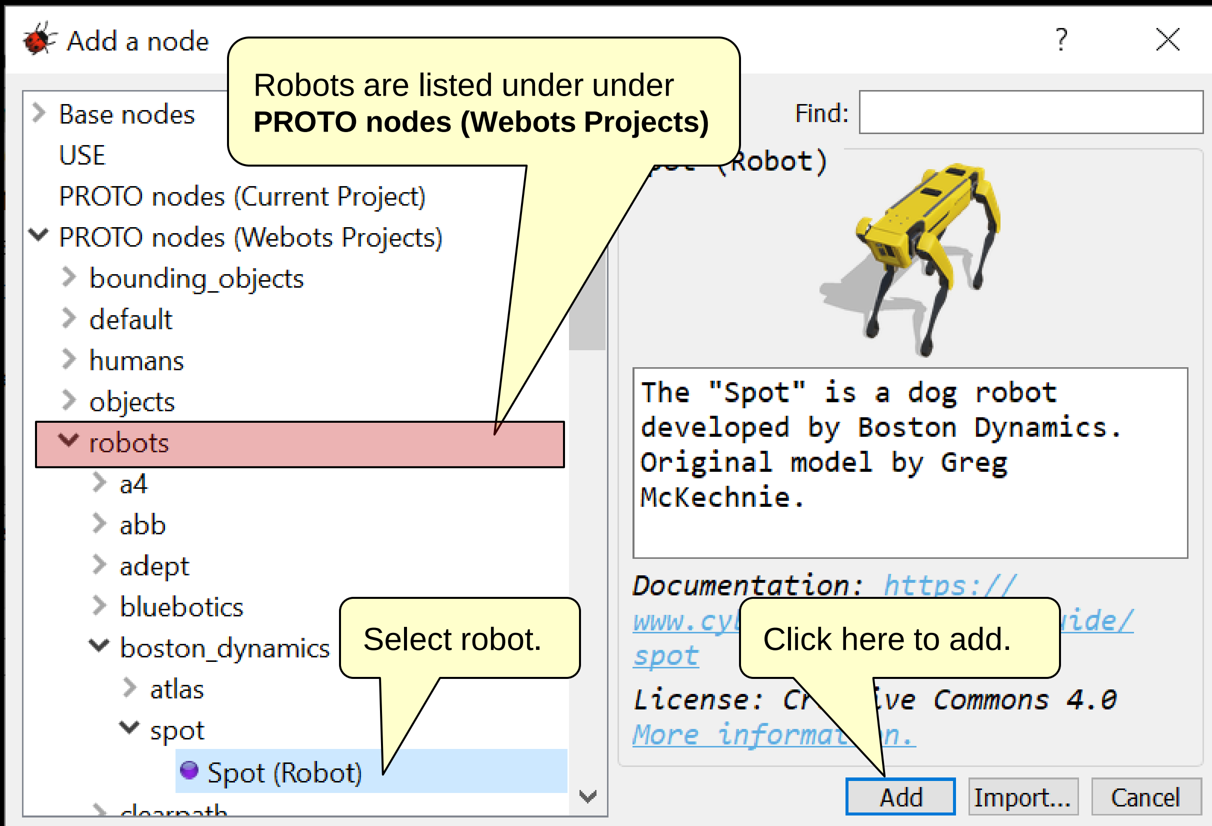
2. Select **floorSize** for RectangleArena.

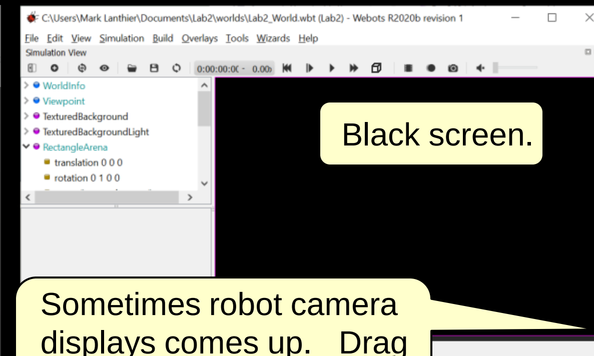
3. Increase or decrease **x** & **y** size in meters.

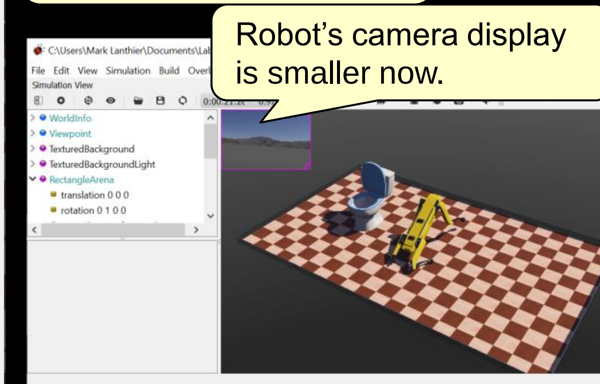
4. Floor will grow immediately.

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

Webots Interface – Adding a Robot

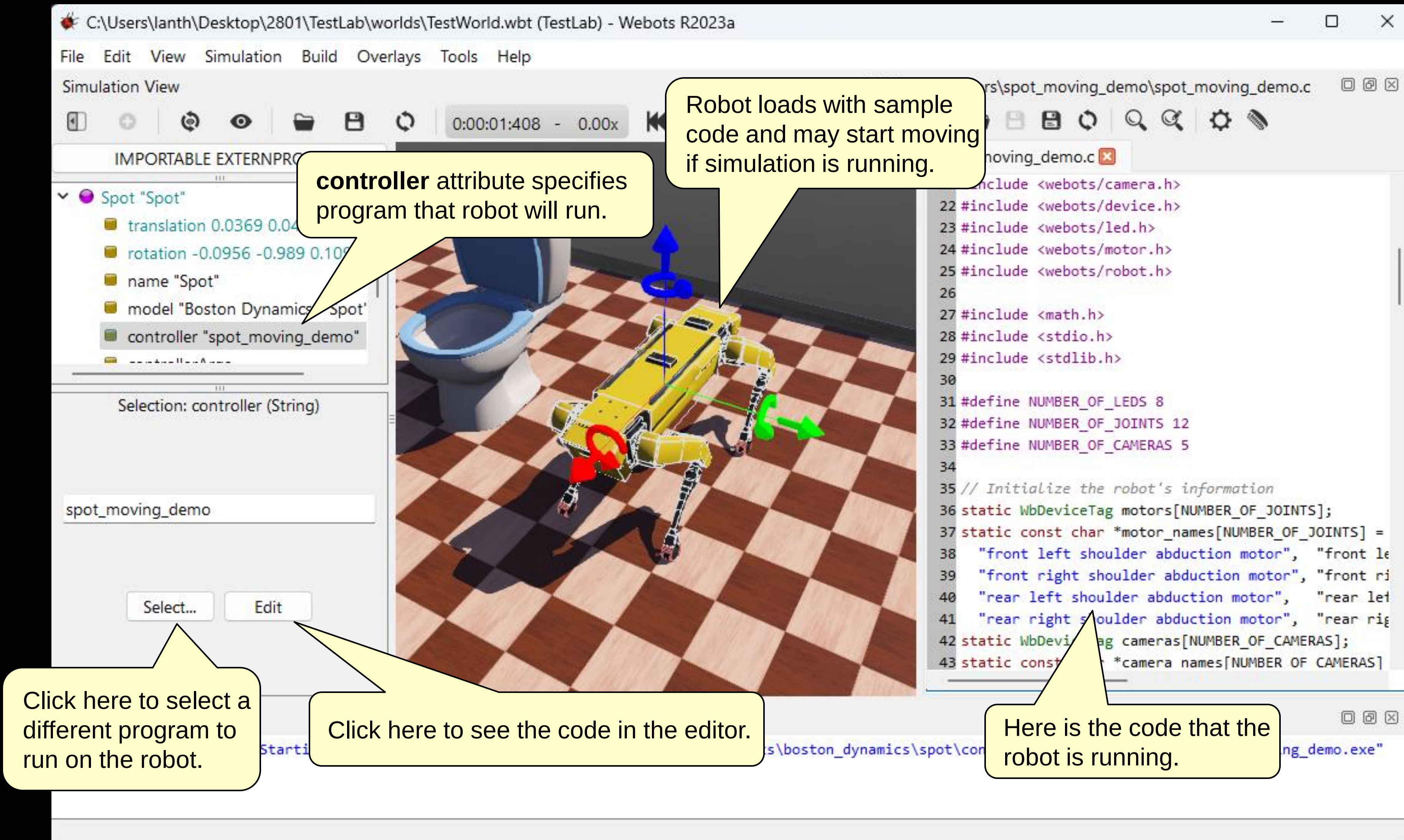




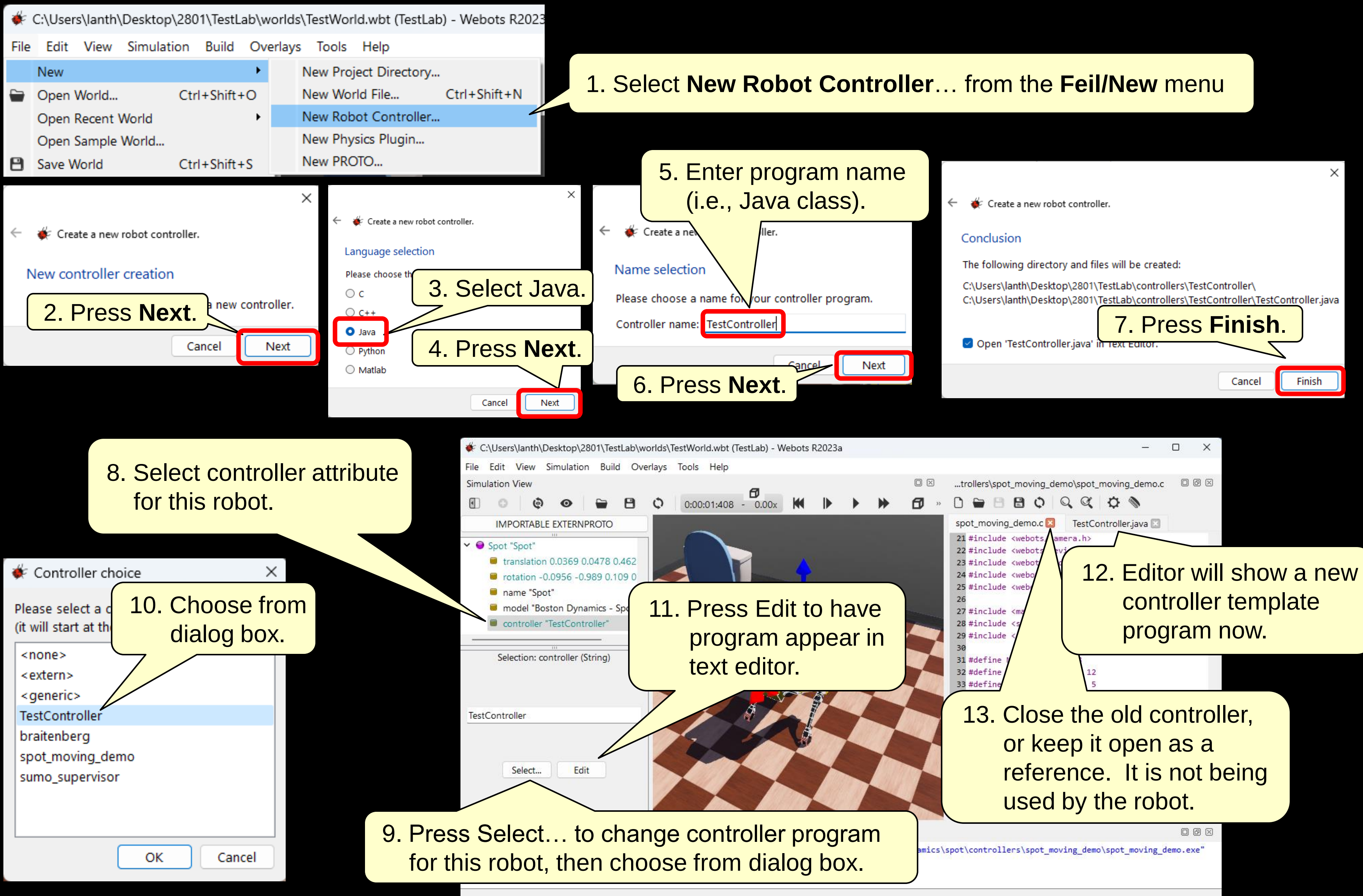


Don't forget to SAVE your world after adding your robot.

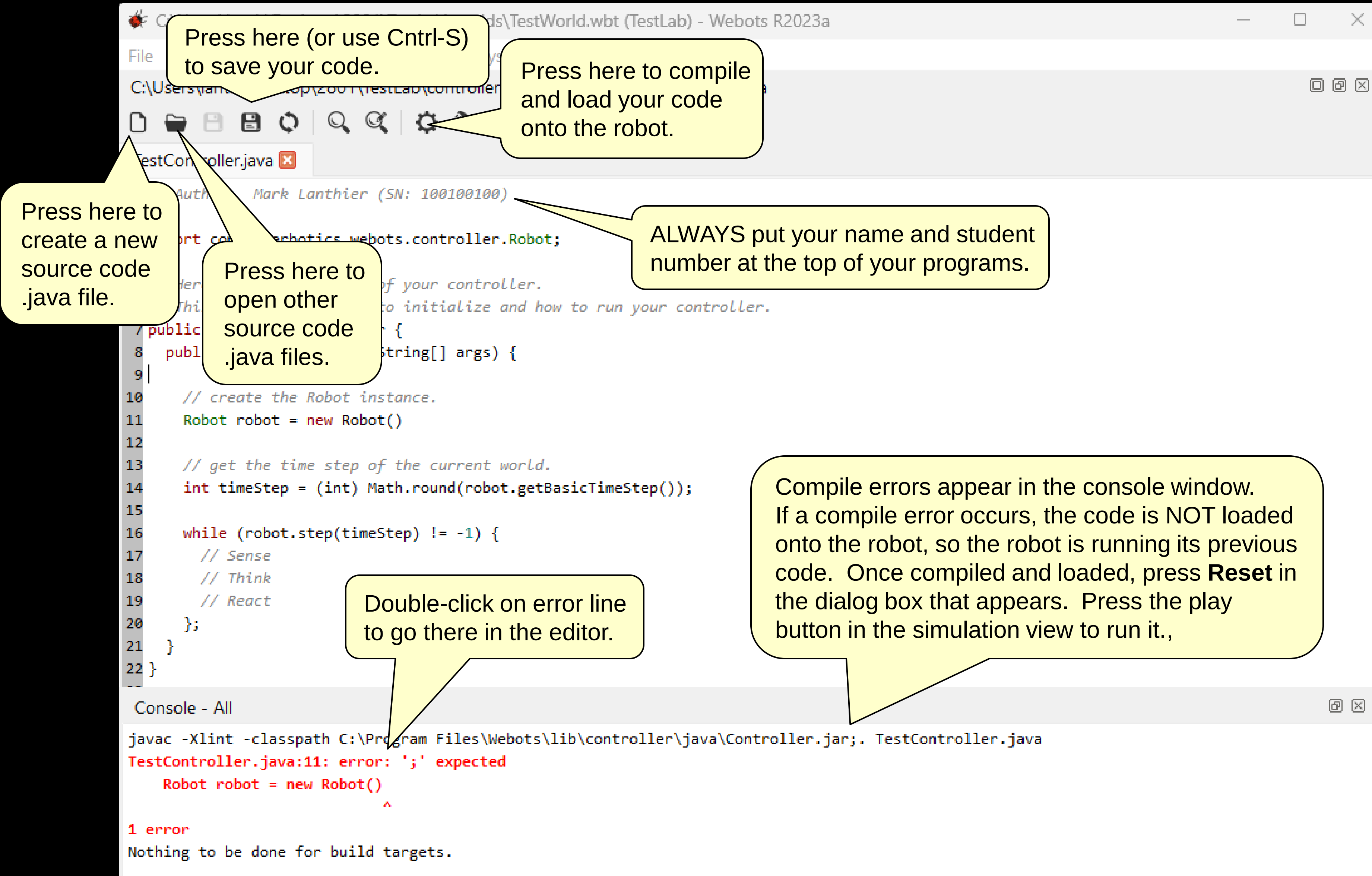
Webots Interface – Controller Code



Webots Interface – Your Controller

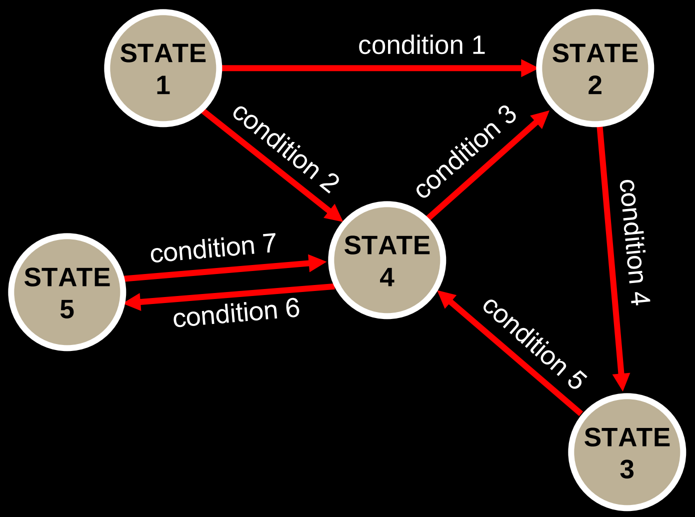


Webots Interface – Editing Code



State Machines

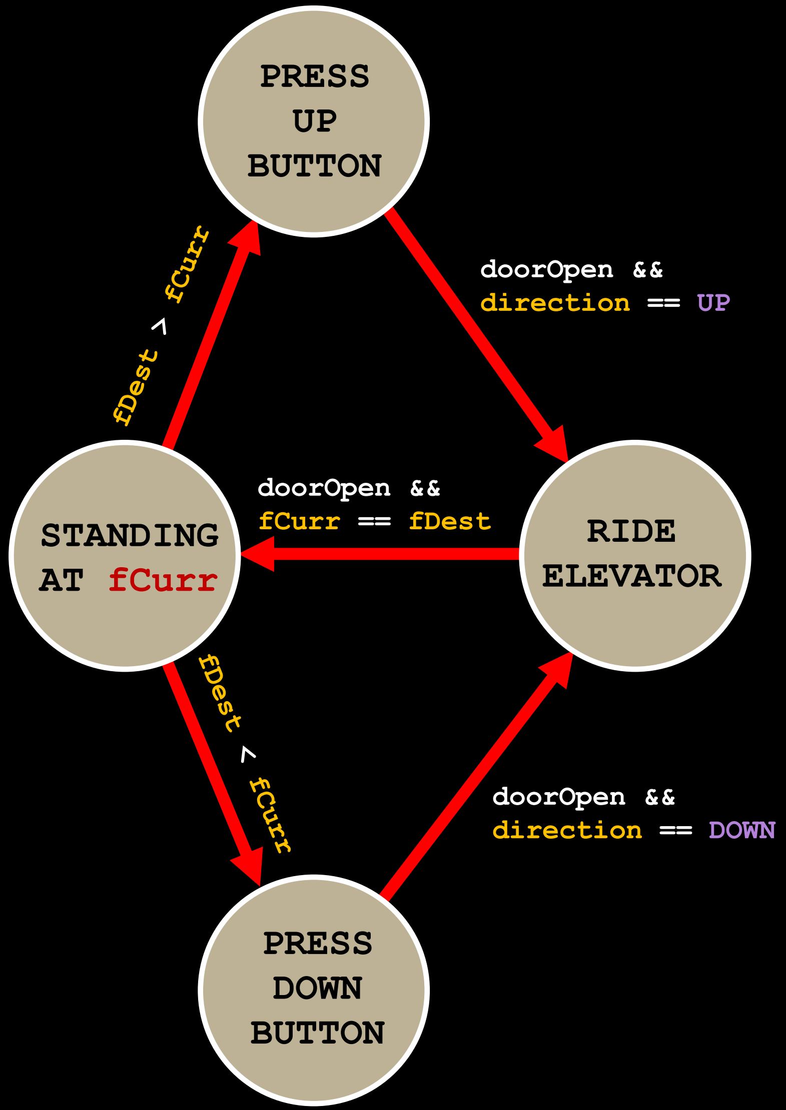
- A robot can be in various ***states*** at any time
  - e.g., STOPPED, MOVING_FORWARD, SPINNING_LEFT, etc…
- A ***state machine*** is a diagram that explains how a robot should moves from one state to another.
  - Each state indicates what the robot is doing at any moment in time.
  - A robot ***transitions*** 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_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;
```



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.

```
// 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

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..)

Start the Lab …