



# El entorno de simulación Webots

---

Robótica

Alberto Díaz y Raúl Lara

Curso 2022/2023

Departamento de Sistemas Informáticos

License CC BY-NC-SA 4.0

# ¿Qué es Webots<sup>1</sup>?

---

Plataforma para la simulación de robots en entornos físicos:

- *Open Software*, publicado bajo los términos de la licencia [Apache License 2.0](#).
- Muy utilizado por la industria, la educación, la investigación y otros campos.

Entorno completo de modelado, programación y simulación para prototipado:

- Creación de mundos [realistas](#) con características físicas como masa o fricción.
- [Objetos pasivos simples](#) u objetos activos llamados [robots](#).
- Ofrece diferentes esquemas de movilidad (e.g. ruedas, patas o voladores)
- Múltiples [sensores](#) (e.g. cámaras) y [actuadores](#) (e.g. motores).

---

<sup>1</sup> Sitio web oficial: <https://www.cyberbotics.com/>.

<sup>2</sup> Concretamente desde diciembre de 2018, desde la publicación de la versión [R2019a](#).

# Algunos términos comunes

---

**Mundo** : Fichero que contiene las descripciones de los robots y su entorno.

**Controlador** : Programa con el código que controla cualquier robot del mundo.

**Controlador supervisor** : Aquel que permite funciones de administración.

**Nodo** : Cada objeto existente en el escenario o mundo.

**Campo** , propiedad o característica: campo variable en el nodo.

# Instalación del simulador

---

En Windows, basta con descargar el instalador de la última versión y lanzarlo.

En macOS, dos opciones:

- Descargar el fichero de instalación `.dmg` de la aplicación e instalar.
- Instalar a través de *homebrew*.

En GNU/Linux, varias opciones:

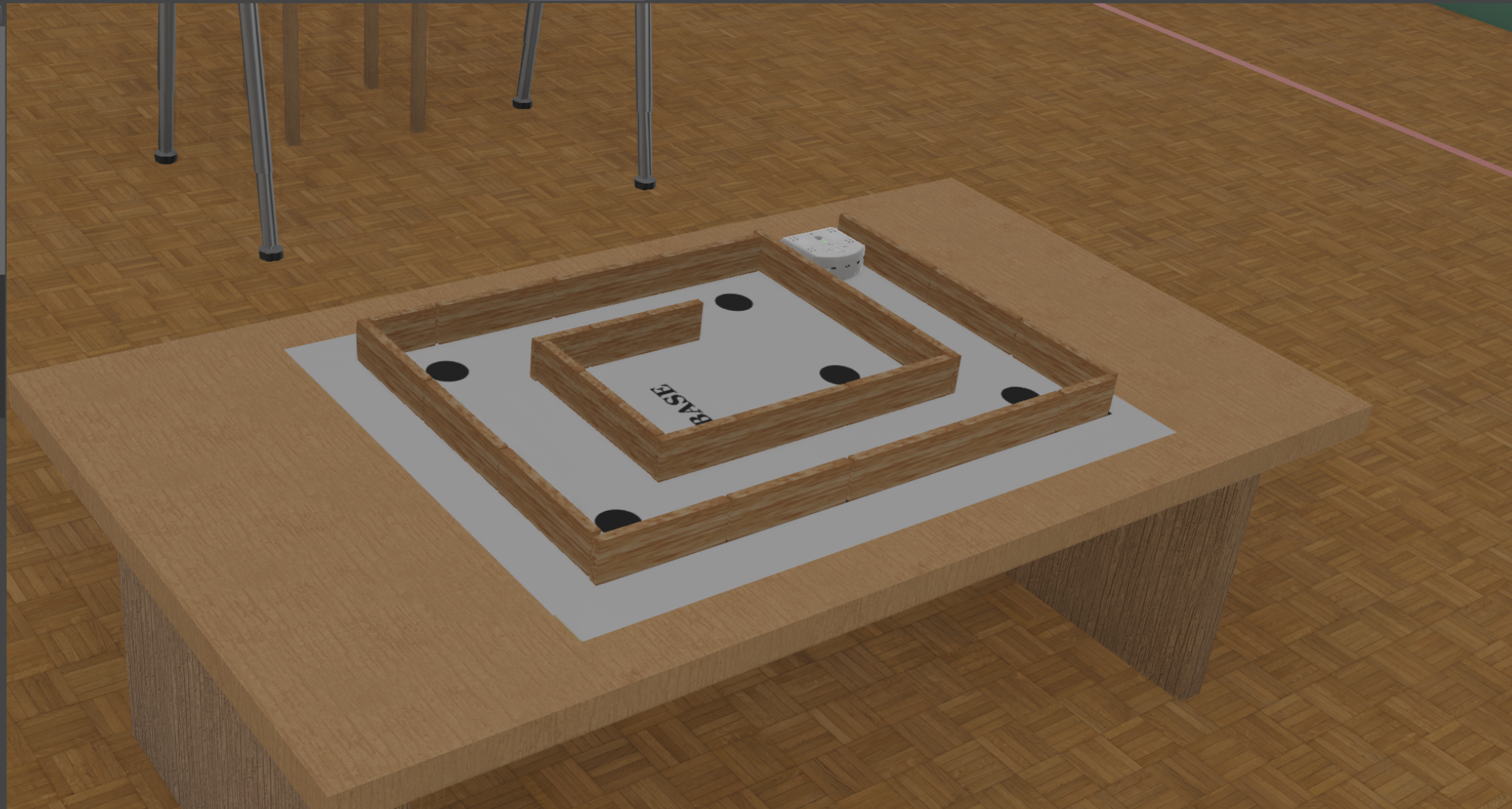
- **Añadiendo el repositorio como fuente adicional del APT (recomendado).**
- Desde un *tarball* ( `.tar.bz2` ) o un paquete `.deb` en el caso de Debian.
- Instalando el paquete disponible en `snap` .

---

Existen también contenedores de docker y servidor para, por ejemplo, el lanzamiento de simulaciones *headless*.

# Un vistazo a la interfaz

- IMPORTABLE EXTERNPROTO
- WorldInfo
  - Viewpoint
  - TexturedBackground
  - CeilingLight "ceiling light(1)"
  - SpotLight
  - Table "table"
  - DEF MAZE\_WALLS Group
  - DEF THYMIO2 Thymio2
  - MazeRunnerBenchmark "robot"
  - Floor "floor"
  - Table "table(2)"
  - WoodenChair "wooden chair"
  - WoodenChair "wooden chair(1)"
  - WoodenChair "wooden chair(2)"
  - WoodenChair "wooden chair(3)"
  - WoodenChair "wooden chair(4)"
  - WoodenChair "wooden chair(5)"
  - WoodenChair "wooden chair(6)"
  - WoodenChair "wooden chair(7)"
  - Door "door"
  - DEF CEILING Solid
  - DEF DOOR\_WALL Solid
  - DEF BACK\_WALL Solid
  - DEF SHORT\_WINDOW\_WALL Solid
  - DEF SHORT\_WINDOW\_WALL Solid
  - DEF LONG\_WINDOW\_WALL Solid
  - DEF LONG\_WINDOW\_WALL Solid



```
move_in_maze.py x
1 """Naive maze runner controller."""
2
3 from controller import Robot
4
5 # Get reference to the robot.
6 robot = Robot()
7
8 # Get simulation step length.
9 timeStep = int(robot.getBasicTimeStep)
10
11 # Constants of the Thymio II motors
12 maxMotorVelocity = 6
13
14 # Get left and right wheel motors.
15 leftMotor = robot.getDevice("motorL")
16 rightMotor = robot.getDevice("motorR")
17
18 # Frontal distance sensors that can detect black circles
19 outerLeftSensor = robot.getDevice("sensorL4")
20 centralLeftSensor = robot.getDevice("sensorL3")
21 centralSensor = robot.getDevice("sensorL2")
22 centralRightSensor = robot.getDevice("sensorR2")
23 outerRightSensor = robot.getDevice("sensorR3")
24
25 # Enable sensors.
26 outerLeftSensor.enable(timeStep)
27 centralLeftSensor.enable(timeStep)
28 centralSensor.enable(timeStep)
29 centralRightSensor.enable(timeStep)
30 outerRightSensor.enable(timeStep)
31
32 # Get and enable ground sensors to detect black circles
33 groundLeftSensor = robot.getDevice("sensorL1")
34 groundRightSensor = robot.getDevice("sensorR1")
35 groundLeftSensor.enable(timeStep)
36 groundRightSensor.enable(timeStep)
37
38 # Disable motor PID control mode.
39 leftMotor.setPosition(float('inf'))
40 rightMotor.setPosition(float('inf'))
41
42 # Set ideal motor velocity.
43 velocity = 0.7 * maxMotorVelocity
44
45 isRotating = False
46 while robot.step(timeStep) != -1:
47     # Read values from four distance sensors
48     if not isRotating and centralSensor.getValue() < 0.1:
49         # Black circle detected.
50         isRotating = True
51     elif isRotating and outerLeftSensor.getValue() < 0.1:
52         isRotating = False
53
54     leftMotor.setVelocity(velocity)
55     if isRotating:
56         rightMotor.setVelocity(-velocity)
57     else:
58         rightMotor.setVelocity(velocity)
59
```

nsolve - All

0: move\_in\_maze: Starting controller: python3 -u move\_in\_maze.py

0: supervisor: Starting controller: python3 -u supervisor.py



# Interfaz de Webots

---

**Barra de menú** : Accesos a todos los aspectos de la aplicación.

**Barra de herramientas principal** : Trabajo sobre la simulación

**Árbol de escena** : Información jerárquica acerca del mundo, objetos y robots.

- **Worldinfo** : Parámetros como el paso de simulación y la gravedad.
- **Viewpoint** : Parámetros relacionados con la perspectiva de visualización.
- **Editor de dominio** : Modificación de características del nodo seleccionado.

**Pantalla de simulación** : Ventana para la visualización de la simulación.

**Editor de texto** : Sirve para editar los controladores de los robots.

**Consola** : Salida estándar para los controladores que estén funcionando

# **Jerarquía de archivos en un proyecto**



# Estructura base de directorios

---

<https://cyberbotics.com/doc/guide/the-standard-file-hierarchy-of-a-project>

Un proyecto es un directorio con, al menos, un directorio denominado `world/`

- Contiene ficheros de descripción de mundo ( `.wbt` ) y archivos del proyecto
- **Deberá incluir al menos** un fichero con extensión `.wbt`
- Puede incluir un directorio `textures\` con las texturas a utilizar

Ahora bien, normalmente son necesarios más directorios; estos son:

- `controllers/` : Fuentes para el control de robots.
- `libraries/` : Posibles bibliotecas externas en el proyecto.
- `plugins/` : Plugins para alterar el comportamiento típico de la simulación
- `protos/` : Prototipos disponibles para todos los ficheros del proyecto.

# Ficheros asociados a un mundo

---

Cada mundo (e.g. `world.wbt` ) lleva asociados los siguientes ficheros ocultos:

- `.world.wbproj` : Información sobre la UI del usuario (e.g. perspectiva).
- `.world.jpg` : Imagen de carga de 768x432 en simulaciones o animaciones.

Si no existen o se eliminan, se crean al guardar correctamente el mundo.

# El directorio `controllers/`

---

Contiene un **directorio por cada posible controlador** de la simulación:

- El `.wbt` contiene el nombre del controlador a iniciarse para cada robot.
- Ese nombre hace referencia al **directorio del controlador**
- Es un campo independiente de plataforma y lenguaje (sólo es una cadena)

Cuando Webots intenta inicializar un controlador sigue el siguiente proceso:

1. Busca en `controllers/` un directorio que coincida con el nombre indicado
2. Busca en el subdirectorio un fichero que coincida con el nombre indicado
3. Si hay varios, selecciona uno de ellos siguiendo el siguiente orden:

```
[.exe] > .class > .jar > .bsg > .py > .m
```

4. Si no encuentra ninguno, lanzará un error y iniciará un controlador vacío

# Nodos

Los diferentes objetos que pueblan un mundo se denominan **nodos**

Un Mundo es un archivo que contiene información como dónde están los objetos, qué aspecto tienen, cómo interactúan entre sí, cuál es el color del cielo, cómo se define la gravedad, la fricción, las masas de los objetos, etc. Define el estado inicial de una simulación. Los diferentes objetos se denominan Nodos y se organizan jerárquicamente en un Árbol de Escena. Por lo tanto, un nodo puede contener subnodos. Un mundo se almacena en un archivo con la extensión .wbt. El formato del archivo se deriva del lenguaje VRML97, y es legible para el ser humano. Los archivos de mundo deben ser almacenados directamente en un directorio llamado worlds.

**¡GRACIAS!**