Ciência da Computação - CC7711

Introdução ao Simulador Webots®

Inteligência Artificial e Robótica - CC 7711

Simulador Webots®

Webots™
robot simulation

Olivier Michel
Founder & CEO

CYBERBOTICS
professional mobile robot simulation

Webots®

rápida prototipação e simulação de
robôs móveis

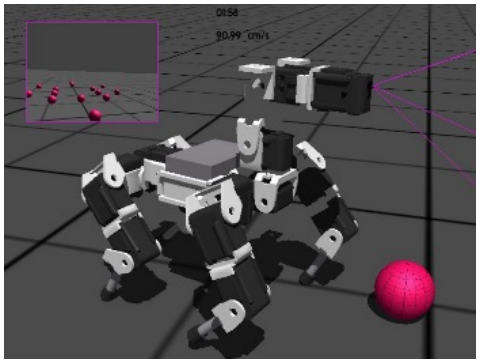
A partir de 2019 ele passou a ser
uma versão *free* (Open Source) para uso

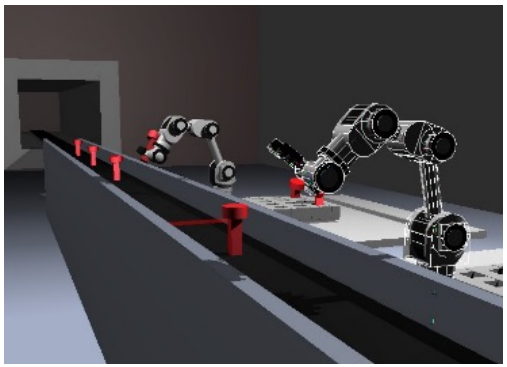
CYBERBOTICS
professional mobile robot simulation

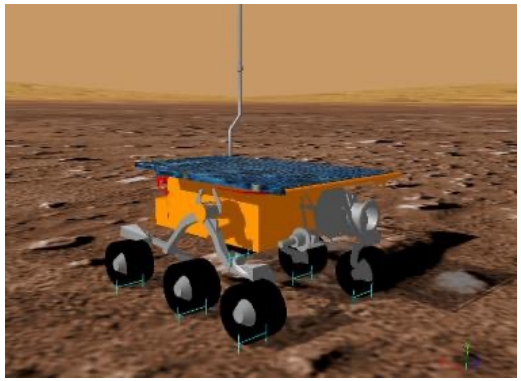
Webots R2019a.
Now Open Source.

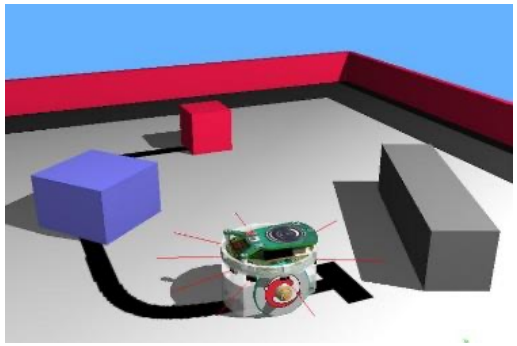
Webots®

Portfolio









Webots®

Alguns usuários

SAMSUNG
Panasonic
ideas for life
SONY
Pioneer
NTT
BAE SYSTEMS
NASA
Tanner RESEARCH
VORWERK
General Electric
SRI International
ALDEBARAN
SoftBank Group
EDF
cea
AREVA
HONDA
TOYOTA
RENAULT NISSAN
JOHN DEERE

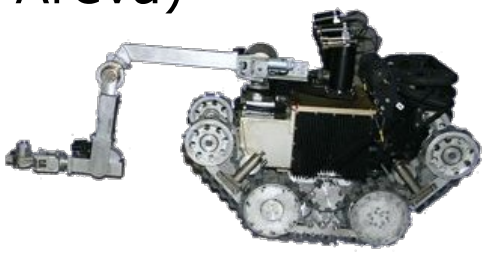
Simulador Webots®

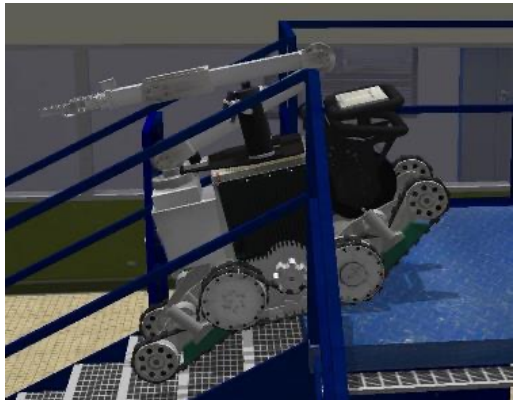
Webots®
Colaborações recentes

- 2012: NAO, ROMEO and PEPER robot
- simulations (Aldebaran robotics)

- 2014-2016:
  Intelligent Vehicle simulation
  (PSA – EPFL/DISAL - CTI)

- 2015-2017: Nuclear disaster
  robot simulation (EDF / CEA / Areva)





Como usar o Simulador ?

Simulador Webots®

Webots®

- Ele possui diversos MUNDOS já modelados.
- Possui diversos sensores, atuadores e robôs completos
- Pode-se criar seu próprio robô como usar os mais de 40 tipos de robôs comerciais já modelados

Ambiente de Simulação

C:\Users\flavi\Documents\tutorials\worlds\FIAP\newFIAP.wbt (No Project) - Webots R2019a

File Edit View Simulation Build Overlays Tools Wizards Help

Simulation View

0:01:04:032 - 0.95x

- WorldInfo
- Viewpoint
- Background
- RectangleArena
- DirectionalLight
- Pioneer3at
- AdvertisingBoard

Selection: WorldInfo

Node

DEF:

C:\Users\flavi\Documents\tutorials\worlds\controllers\pioneer1\pioneer1.c

pioneer1.c | 4_wheels_collision_avoidance.c

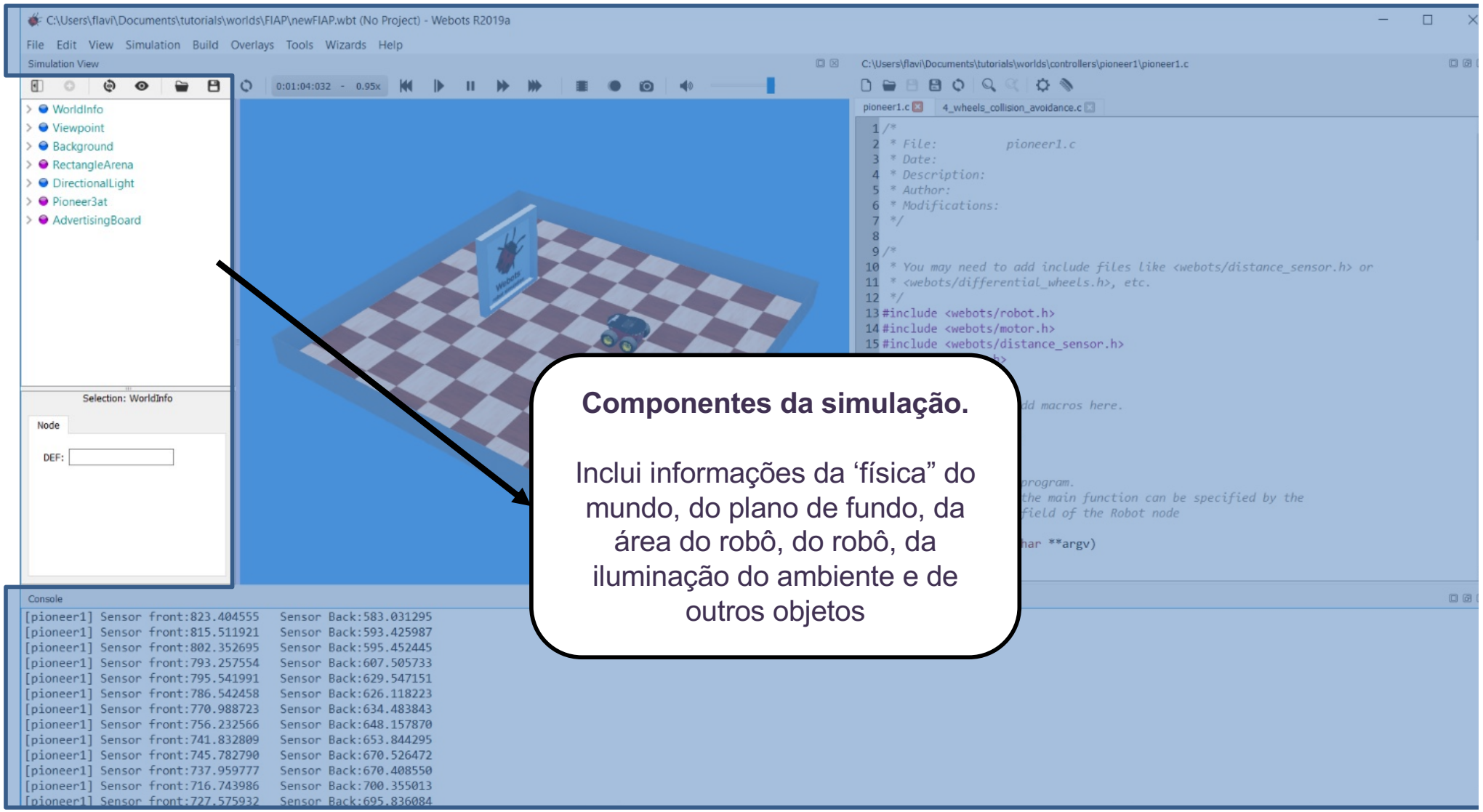
```
/*
 * File:          pioneer1.c
 * Date:
 * Description:
 * Author:
 * Modifications:
 */

/*
 * You may need to add include files like <webots/distance_sensor.h> or
 * <webots/differential_wheels.h>, etc.
 */
#include <webots/robot.h>
#include <webots/motor.h>
#include <webots/distance_sensor.h>
#include <stdio.h>

/*
 * You may want to add macros here.
 */
#define TIME_STEP 64

/*
 * This is the main program.
 * The arguments of the main function can be specified by the
 * "controllerArgs" field of the Robot node
 */
int main(int argc, char **argv)
{

```

Console

```
[pioneer1] Sensor front:823.404555   Sensor Back:583.031295
[pioneer1] Sensor front:815.511921   Sensor Back:593.425987
[pioneer1] Sensor front:802.352695   Sensor Back:595.452445
[pioneer1] Sensor front:793.257554   Sensor Back:607.505733
[pioneer1] Sensor front:795.541991   Sensor Back:629.547151
[pioneer1] Sensor front:786.542458   Sensor Back:626.118223
[pioneer1] Sensor front:770.988723   Sensor Back:634.483843
[pioneer1] Sensor front:756.232566   Sensor Back:648.157870
[pioneer1] Sensor front:741.832809   Sensor Back:653.844295
[pioneer1] Sensor front:745.782790   Sensor Back:670.526472
[pioneer1] Sensor front:737.959777   Sensor Back:670.408550
[pioneer1] Sensor front:716.743986   Sensor Back:700.355013
[pioneer1] Sensor front:727.575932   Sensor Back:695.836084
```

Simulador Webots®

Ambiente de Simulação

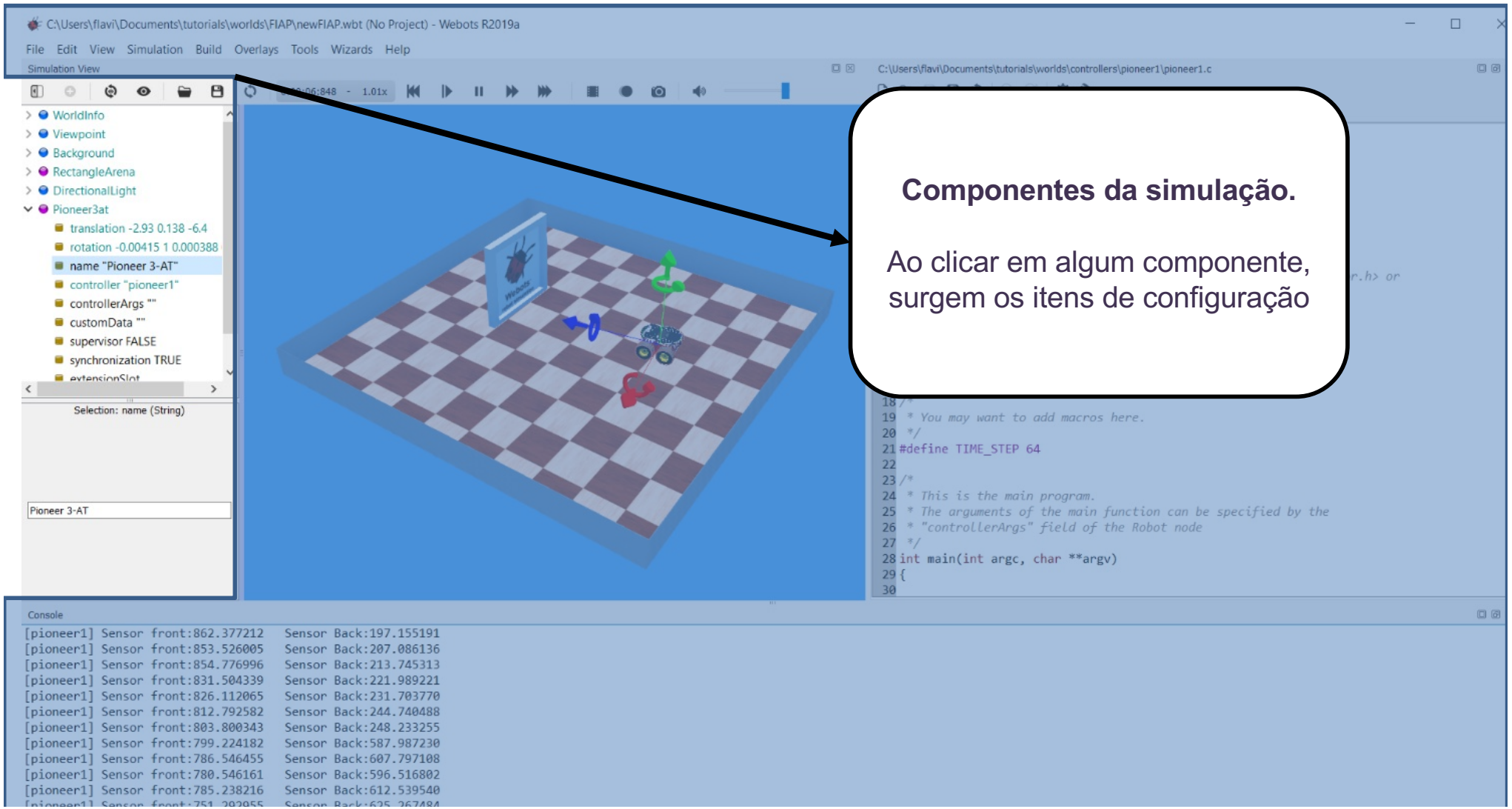


Componentes da simulação.

Inclui informações da ‘física” do mundo, do plano de fundo, da área do robô, do robô, da iluminação do ambiente e de outros objetos

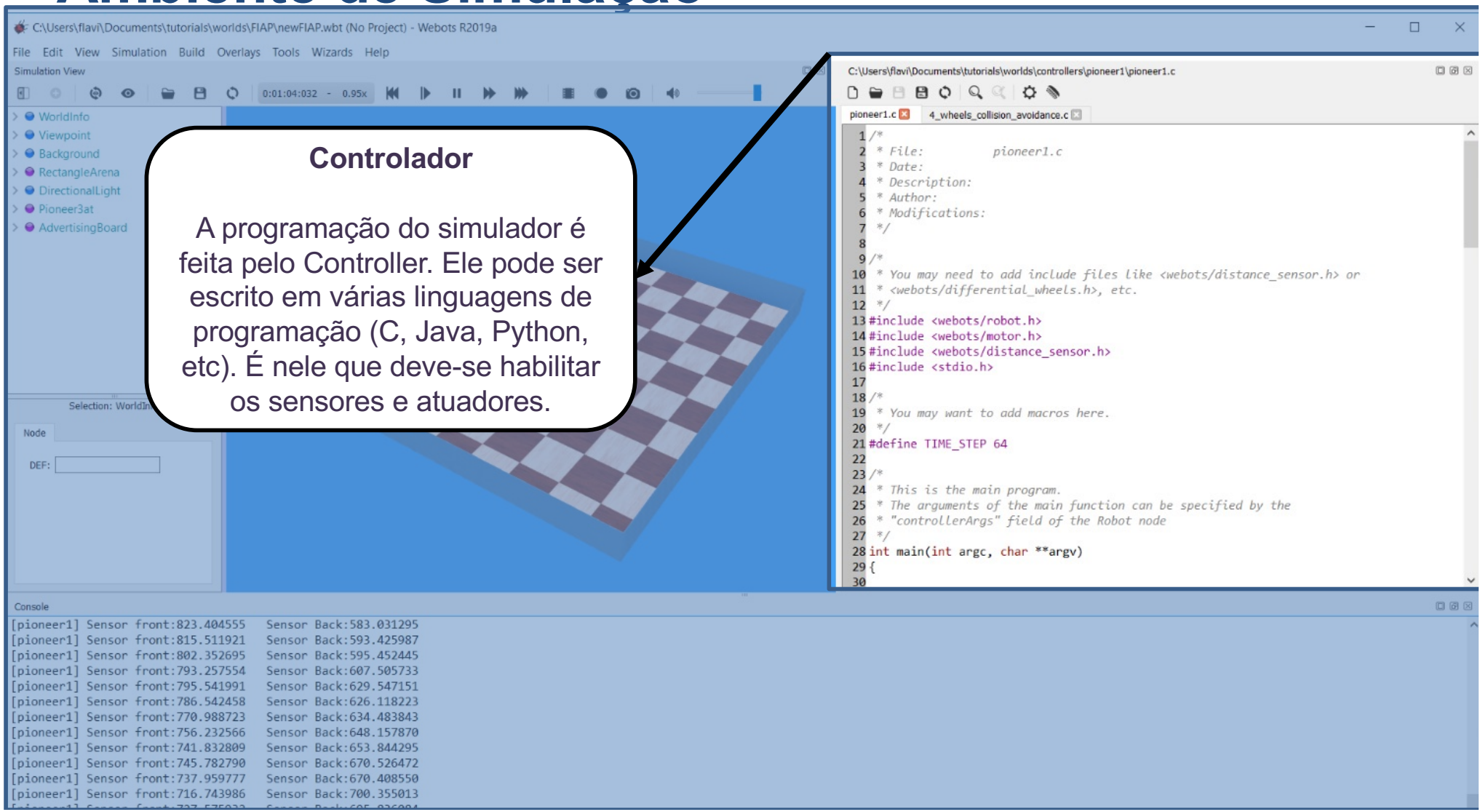
Simulador Webots®

Ambiente de Simulação



Simulador Webots®

Ambiente de Simulação

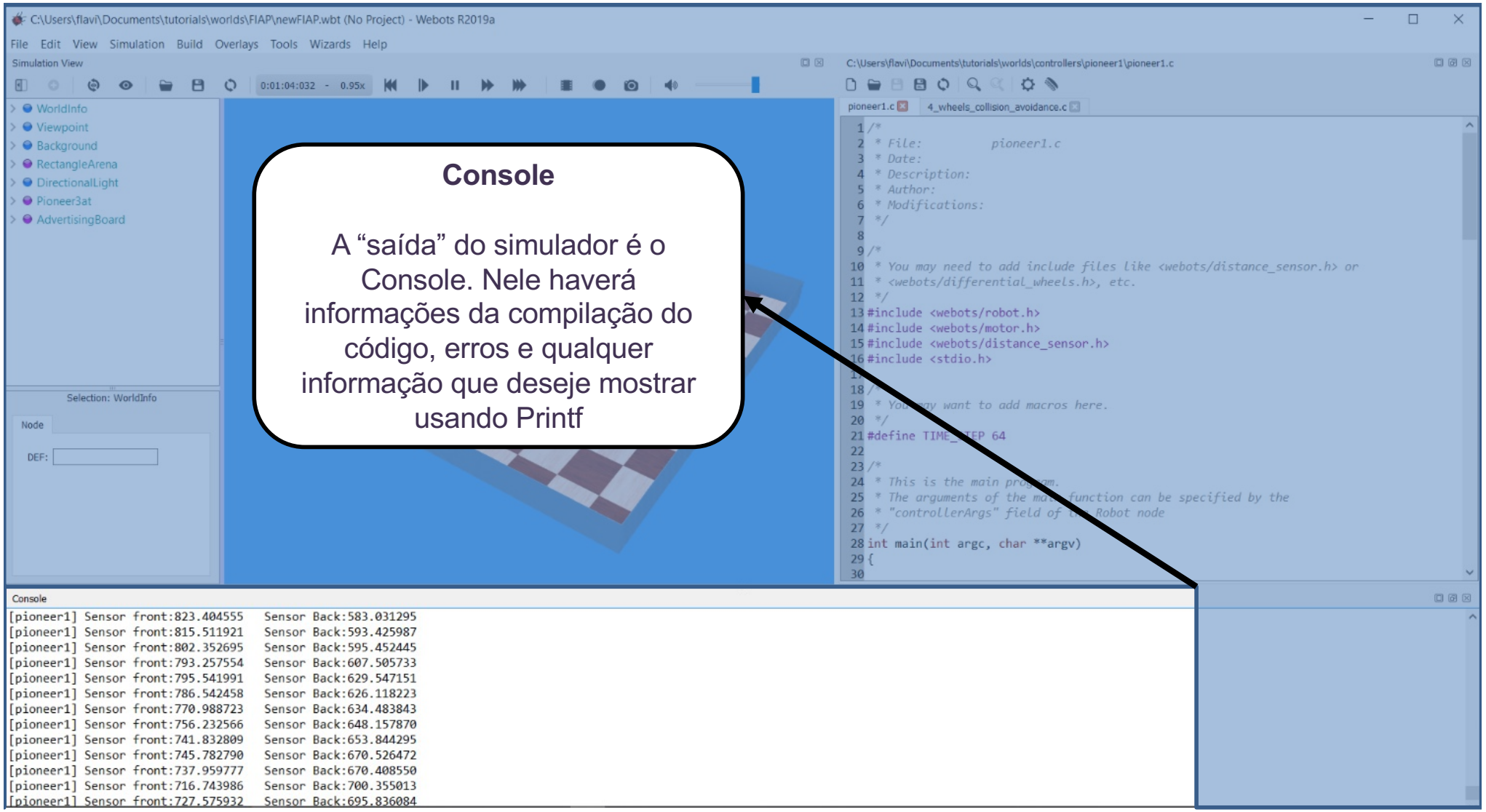


Controlador

A programação do simulador é feita pelo Controller. Ele pode ser escrito em várias linguagens de programação (C, Java, Python, etc). É nele que deve-se habilitar os sensores e atuadores.

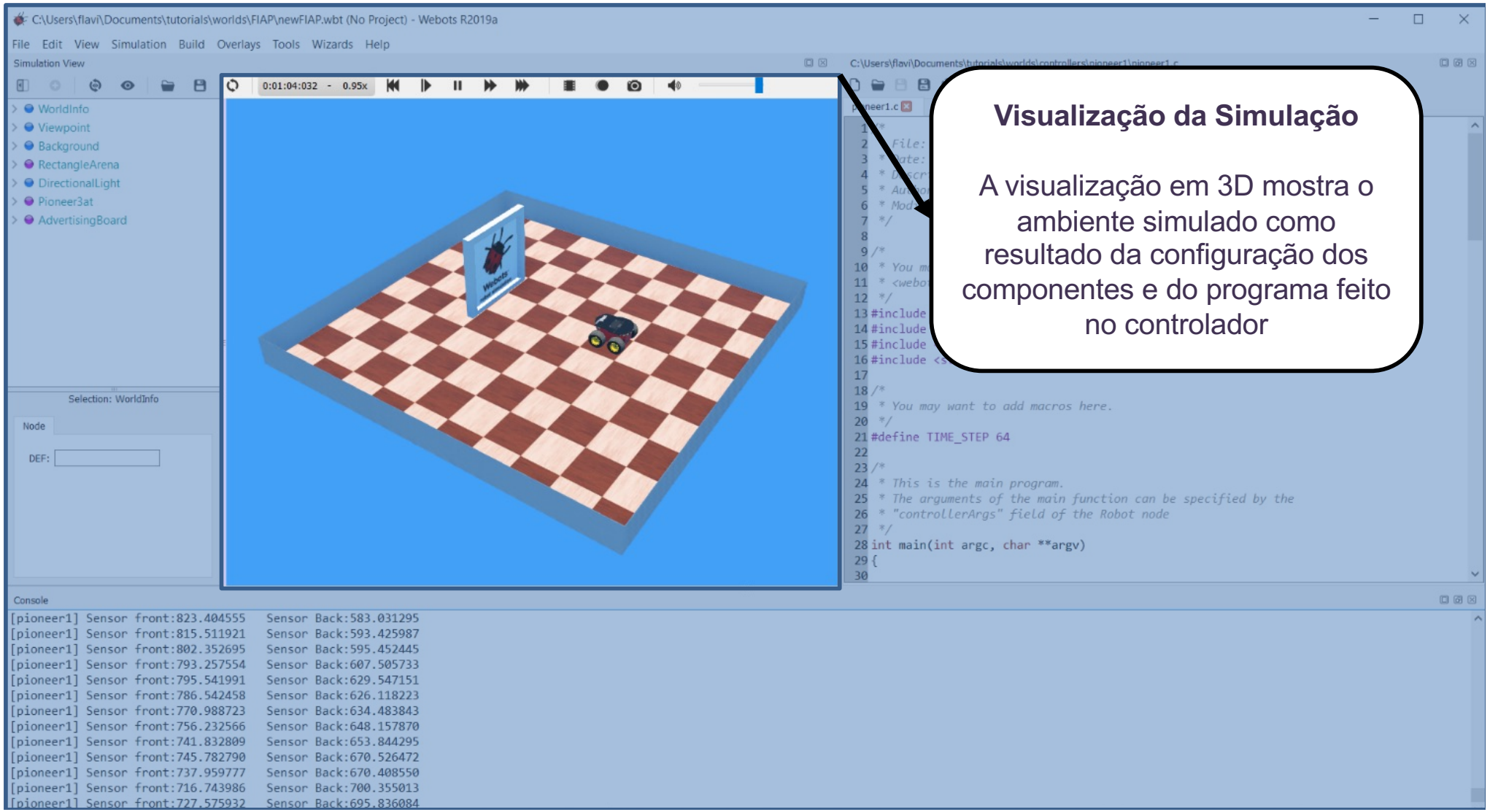
Inteligência Artificial e Robótica - CC 7711

Simulador Webots®



Simulador Webots®

Ambiente de Simulação

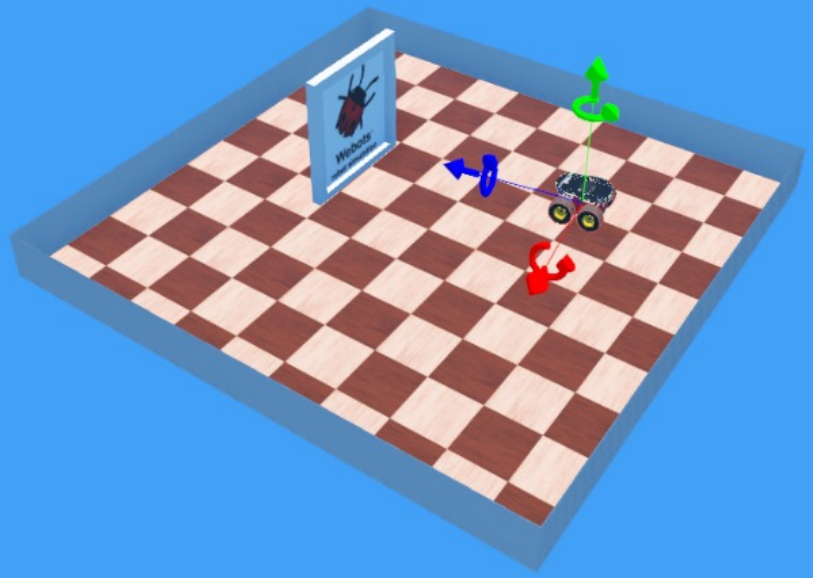


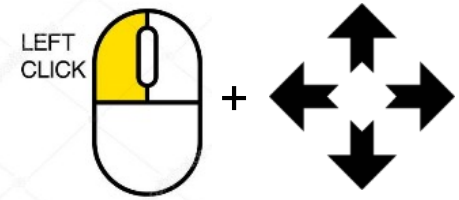
Mexendo nos objetos e no ambiente

Simulador Webots®

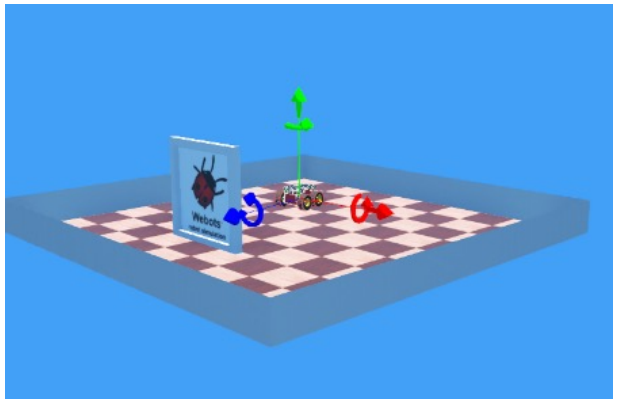
Interagindo com os objetos

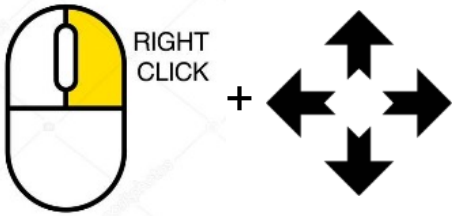
Movendo a Arena da simulação
Clique na área e segure



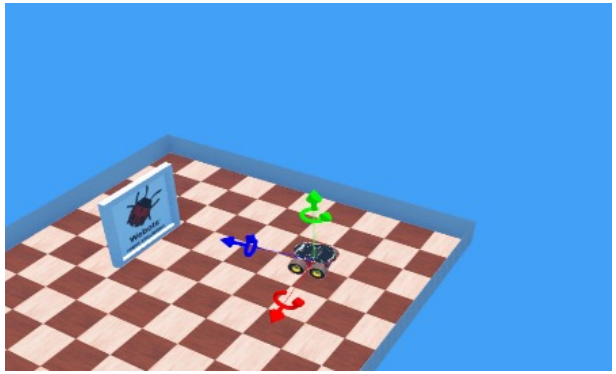


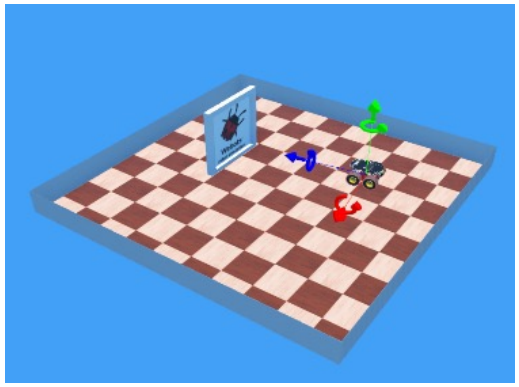
Rotação nos 3 eixos

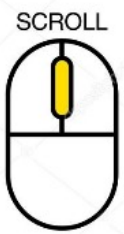




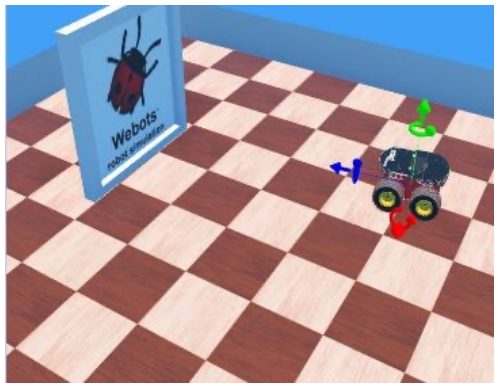
Translação nos 3 eixos



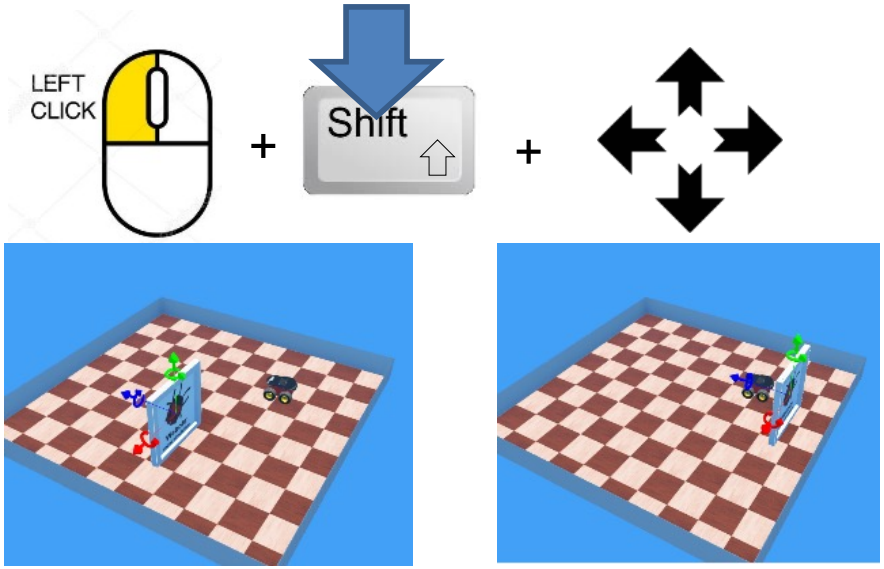




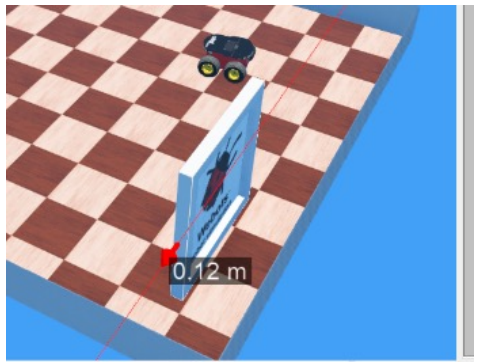
Zoom In ou Zoom out

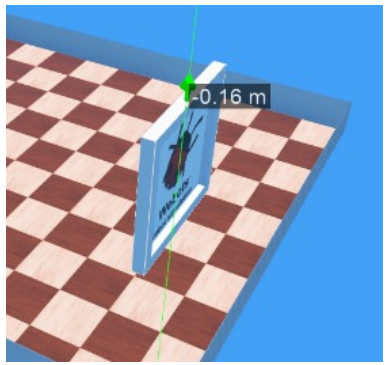


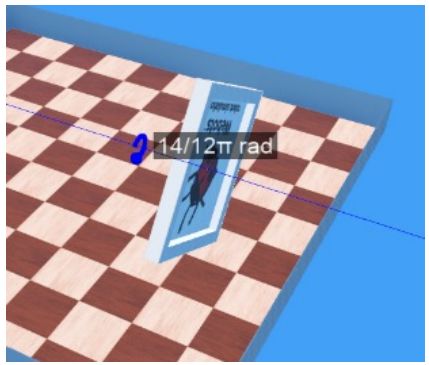
Interagindo com os objetos

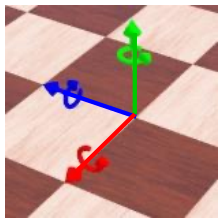


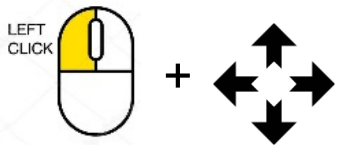
Se clicar EXATAMENTE nas setas de movimentação (verde, azul ou vermelha), é possível rotacionar e transladar os objetos no eixo clicado.











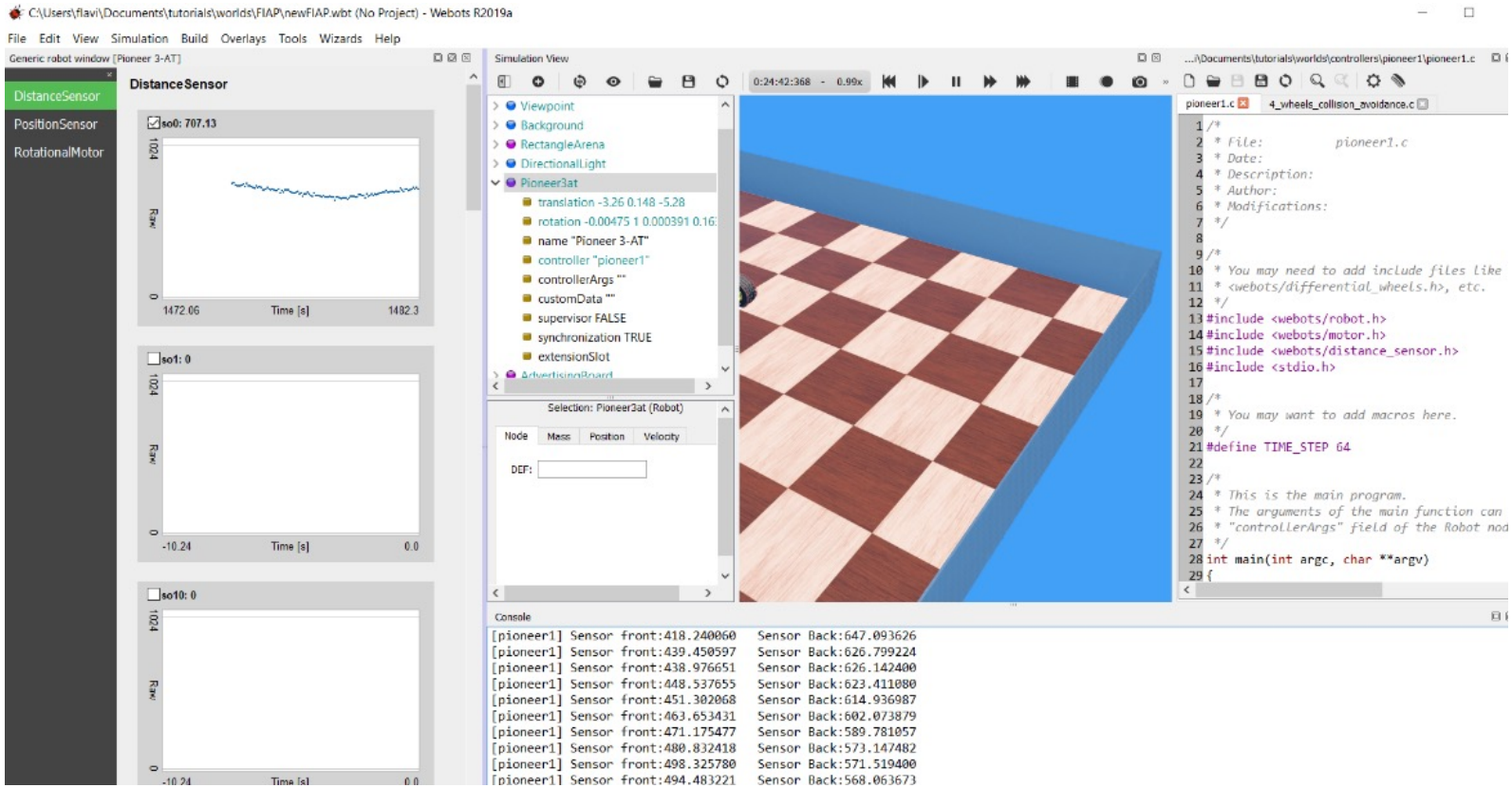
Verificando sensores e atuadores do robô

Sensores e atuadores do robô

- **Dê um duplo clique (LEFT) sobre o robô. Um novo painel a esquerda irá aparecer**

Nomes dos sensores e atuadores

É possível ver os valores de cada sensor clicado (✓) durante a simulação

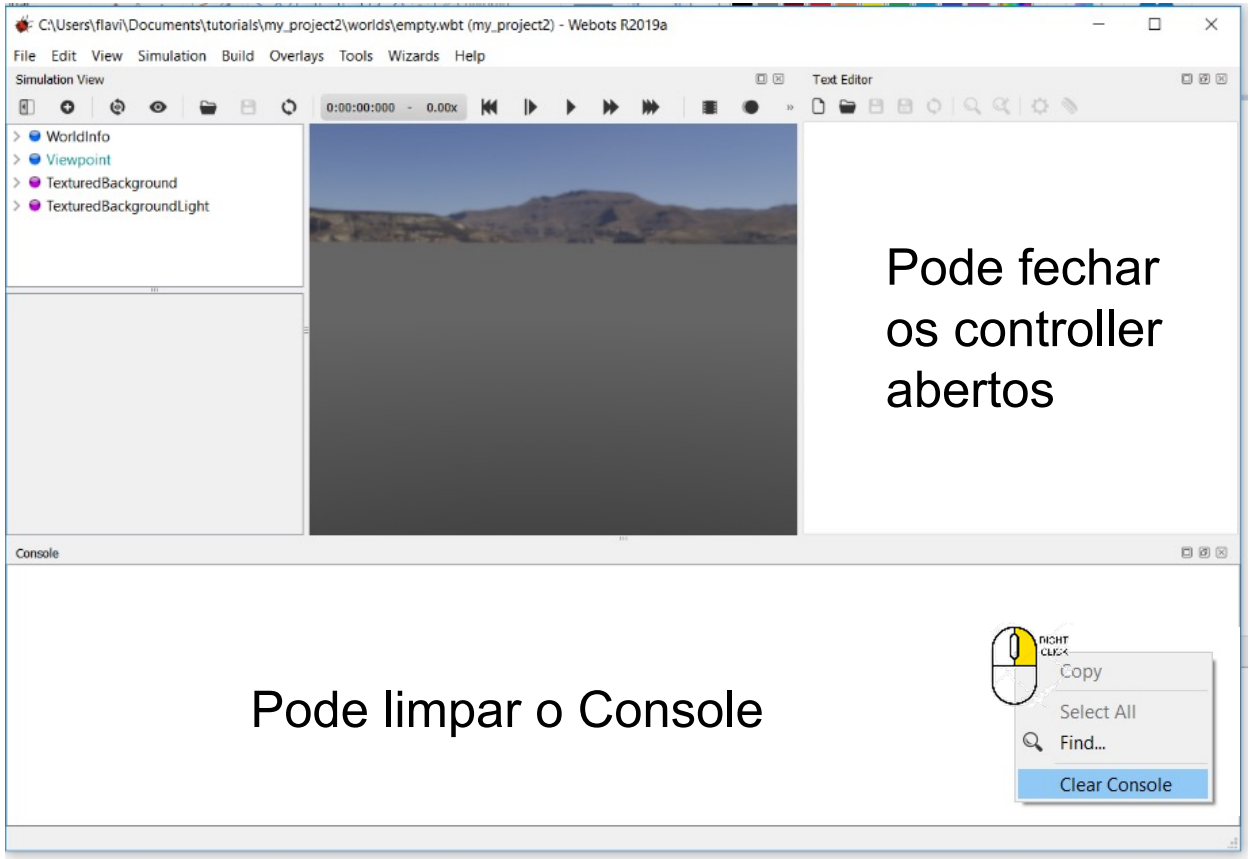


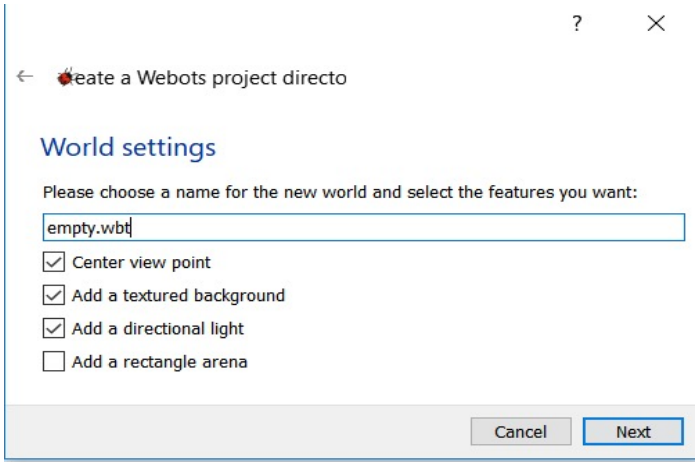
Criando um novo !
(todos)

Simulador Webots®

Criando nova Simulação

- **Wizard -> New Project Directory**
- **Ele irá criar um novo mundo (simulação)**
  - **Todos os diretórios**





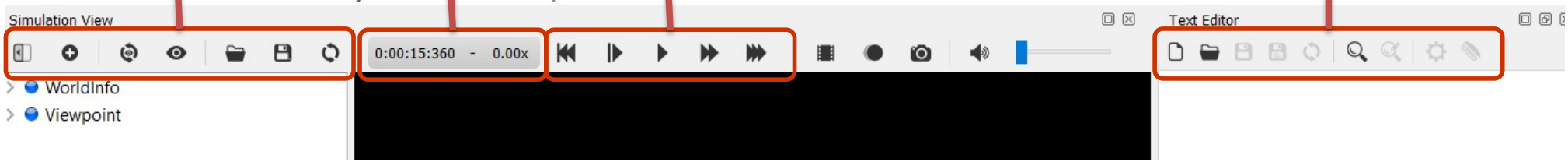
Botões de interação

Tempo e velocidade da simulação

Botões de componentes
(+) insere novo componente na simulação

Botões do Simulador
Rodar, pausar e avançar na simulação

Botões do Controller
Abrir, salvar e compilar a programação do Controller do robô



Inserindo Componentes

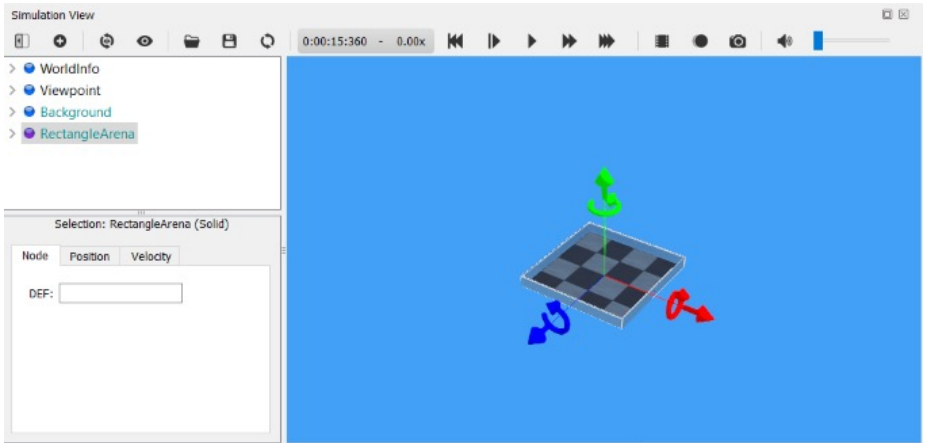
- **Inserindo a Arena Retangular**

  PROTO nodes (webots)

Add a node

floors

RetangleArena



Talvez tenha que rotacioná-lo para vê-lo e coloca-lo na posição ao lado

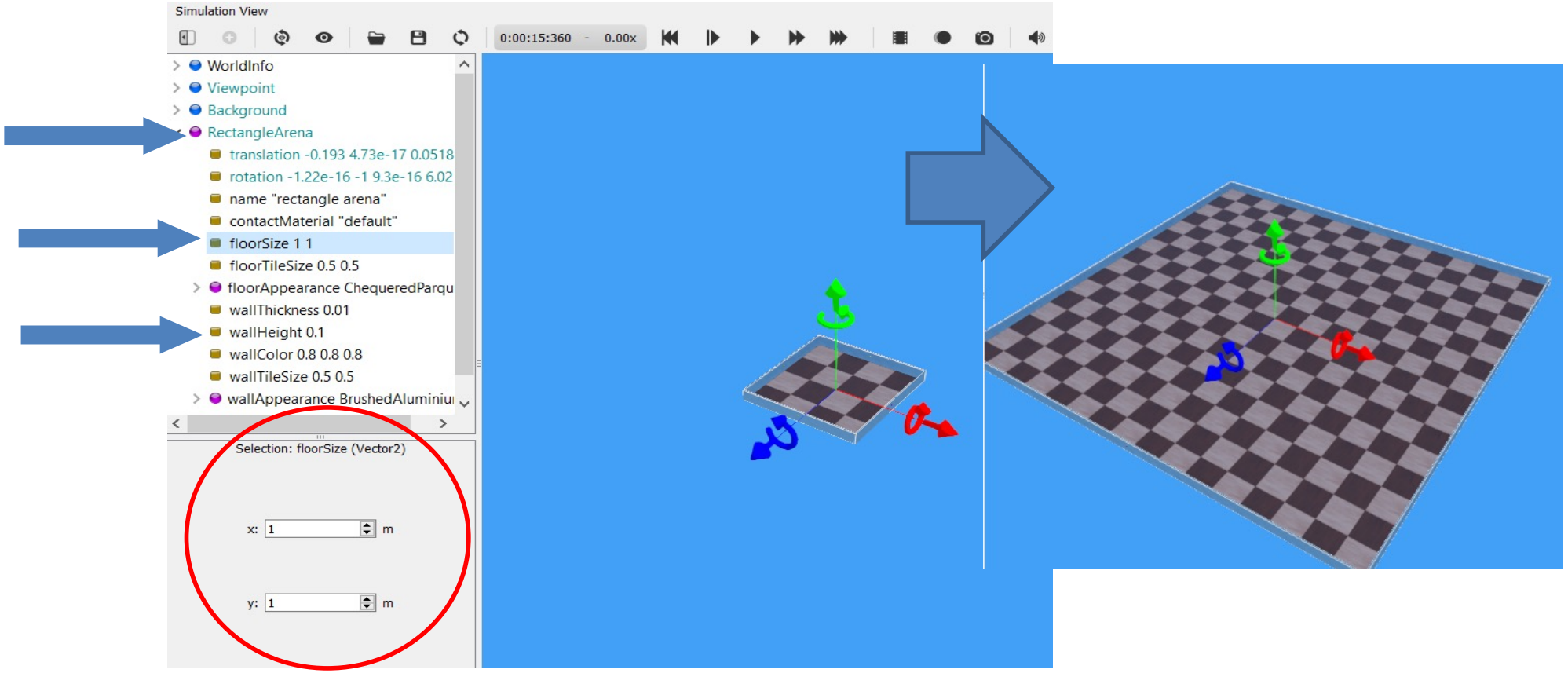
Inserindo Componentes

- **Inserindo a iluminação ambiente**

Precisamos agora configurar a Arena (tamanho) e a iluminação

Inserindo Componentes

- **Ajustando arena (tamanho e altura da parede)**

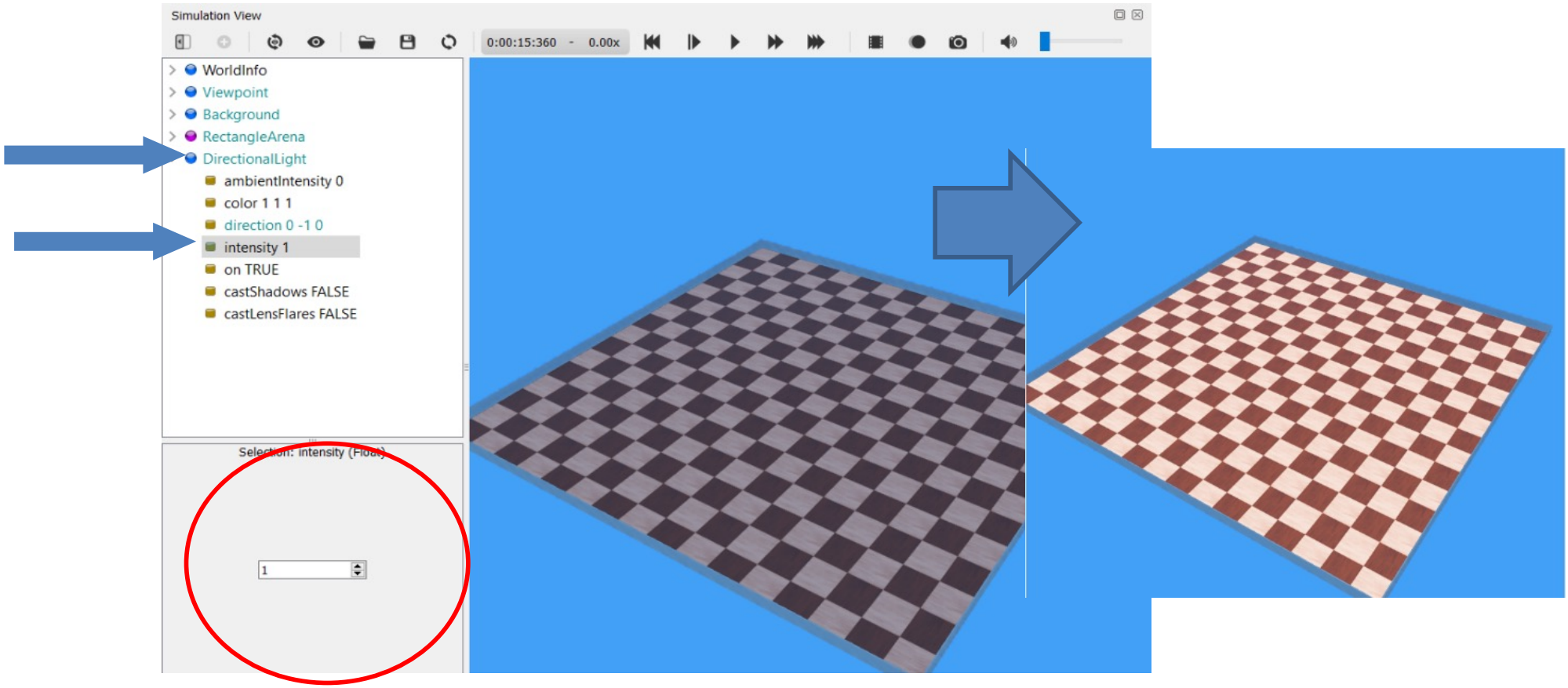


FloorSize: Colocar **4m** em cada

wallHeight: colocar **0.3m** que é o tamanho do robô que iremos usar

Inserindo Componentes

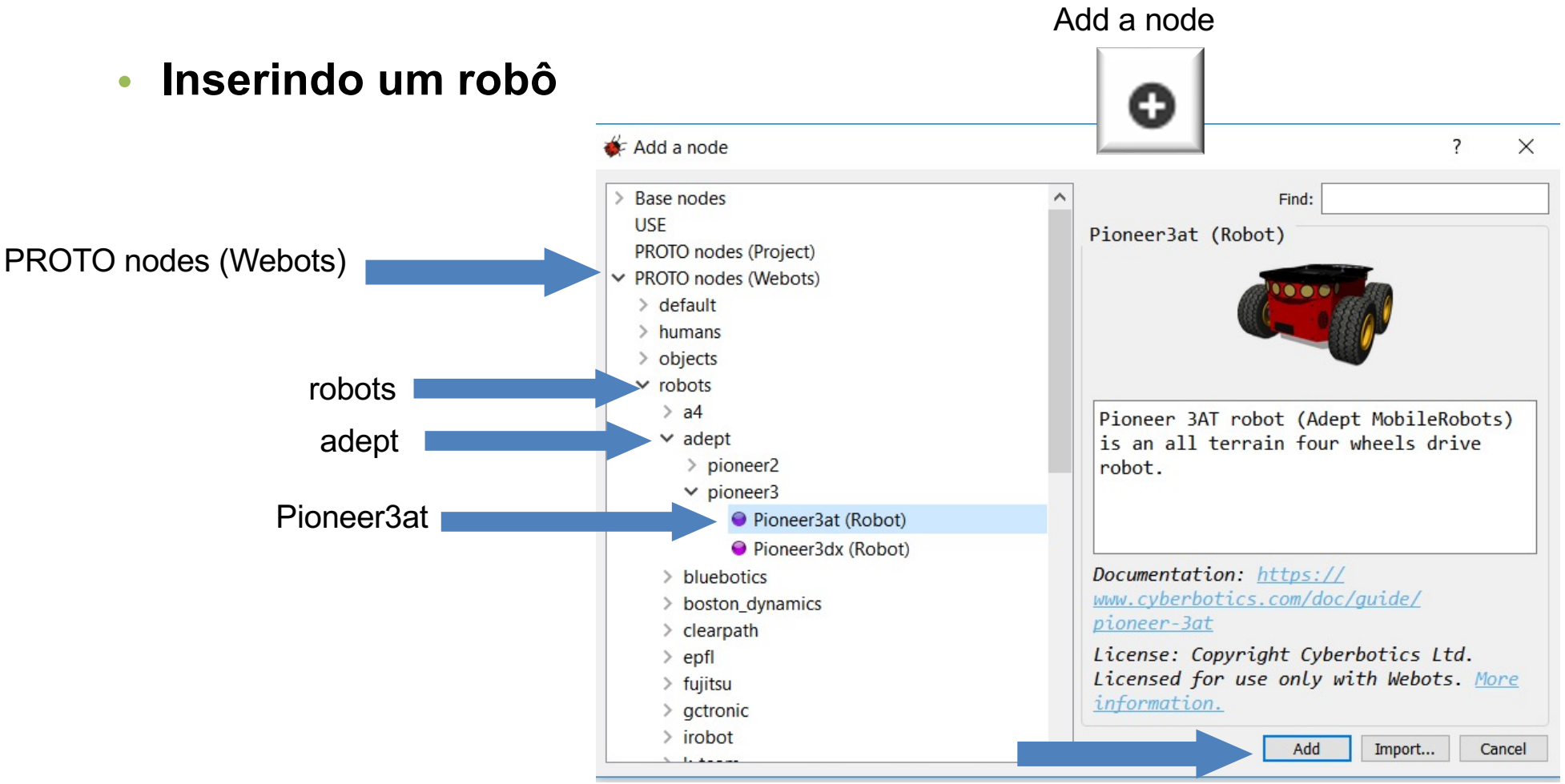
- **Ajustando arena e iluminação**



Colocar 3 de intensidade

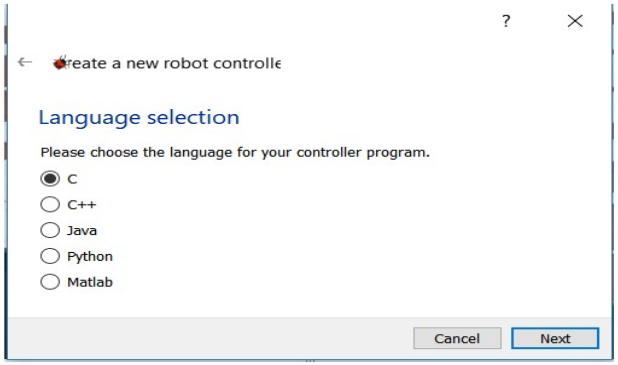
Inserindo Componentes

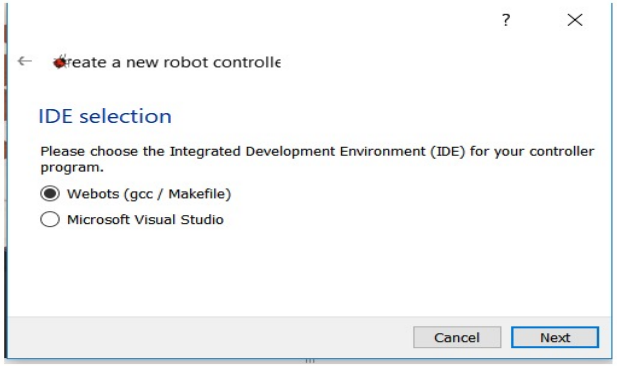
- **Inserindo um robô**

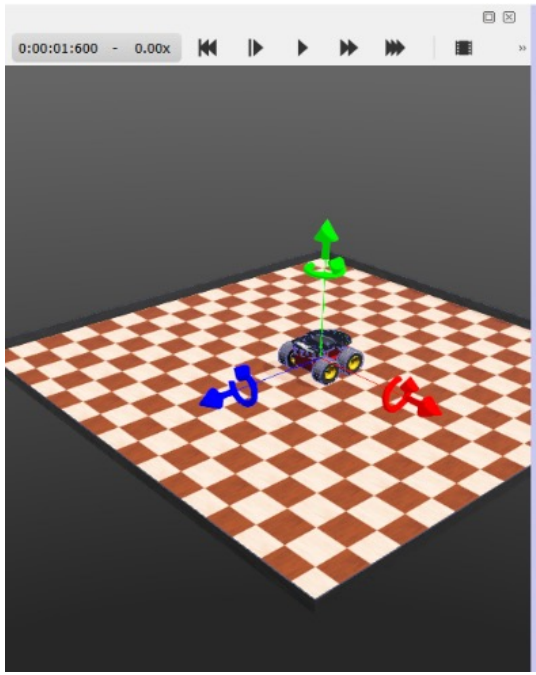


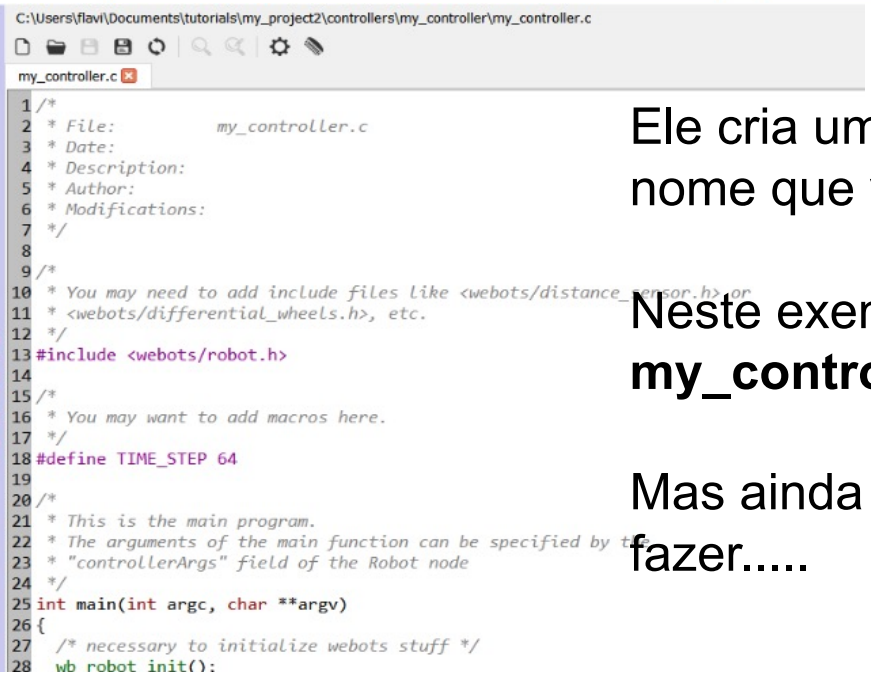
Inserindo um Controller

Wizard -> New Robot Controller









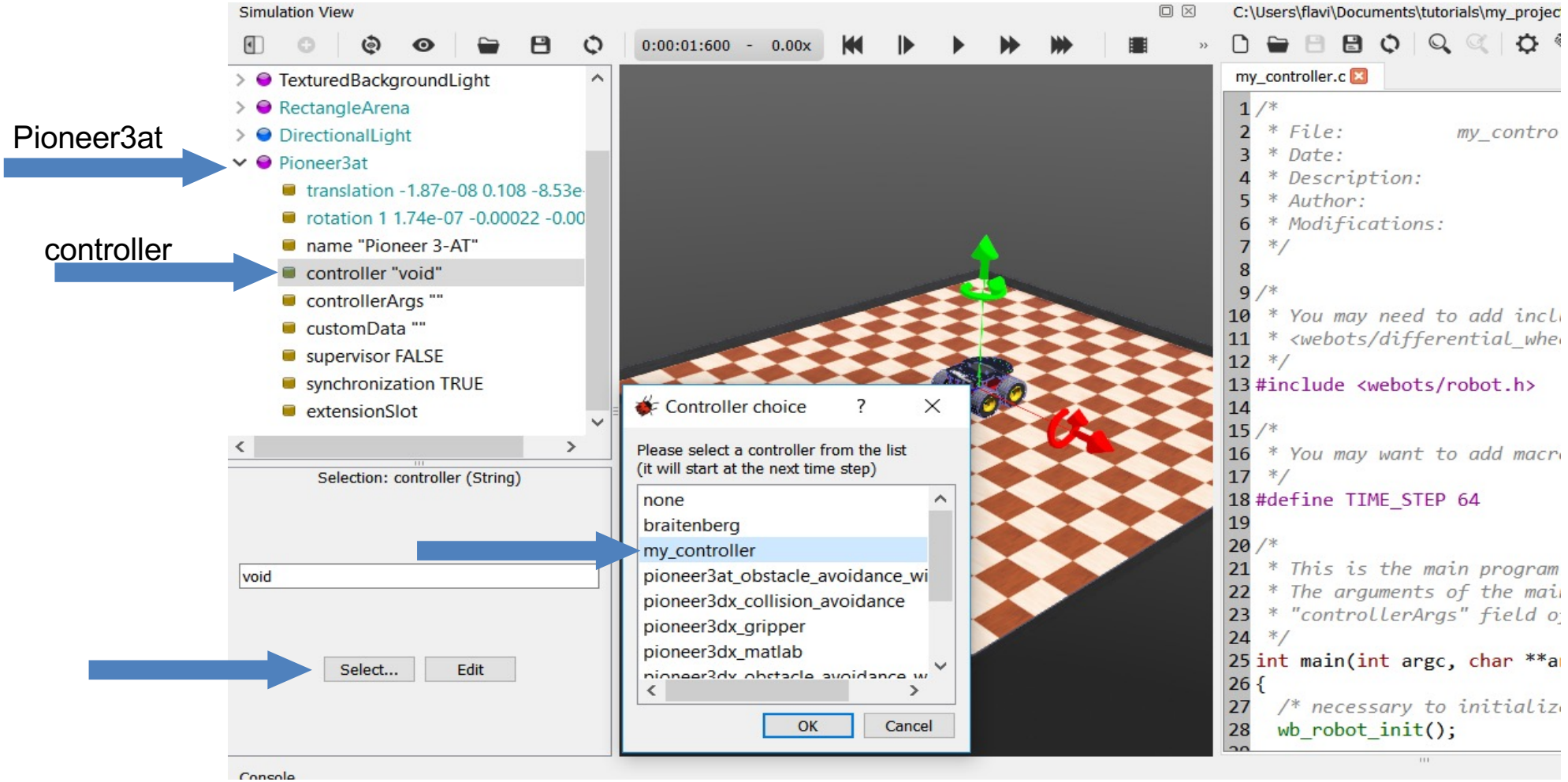
Ele cria um código padrão com o nome que você definir

Neste exemplo o nome *default* **my_controller** foi mantido.

Mas ainda temos mais 1 coisa a fazer.....

Inserindo um Controller

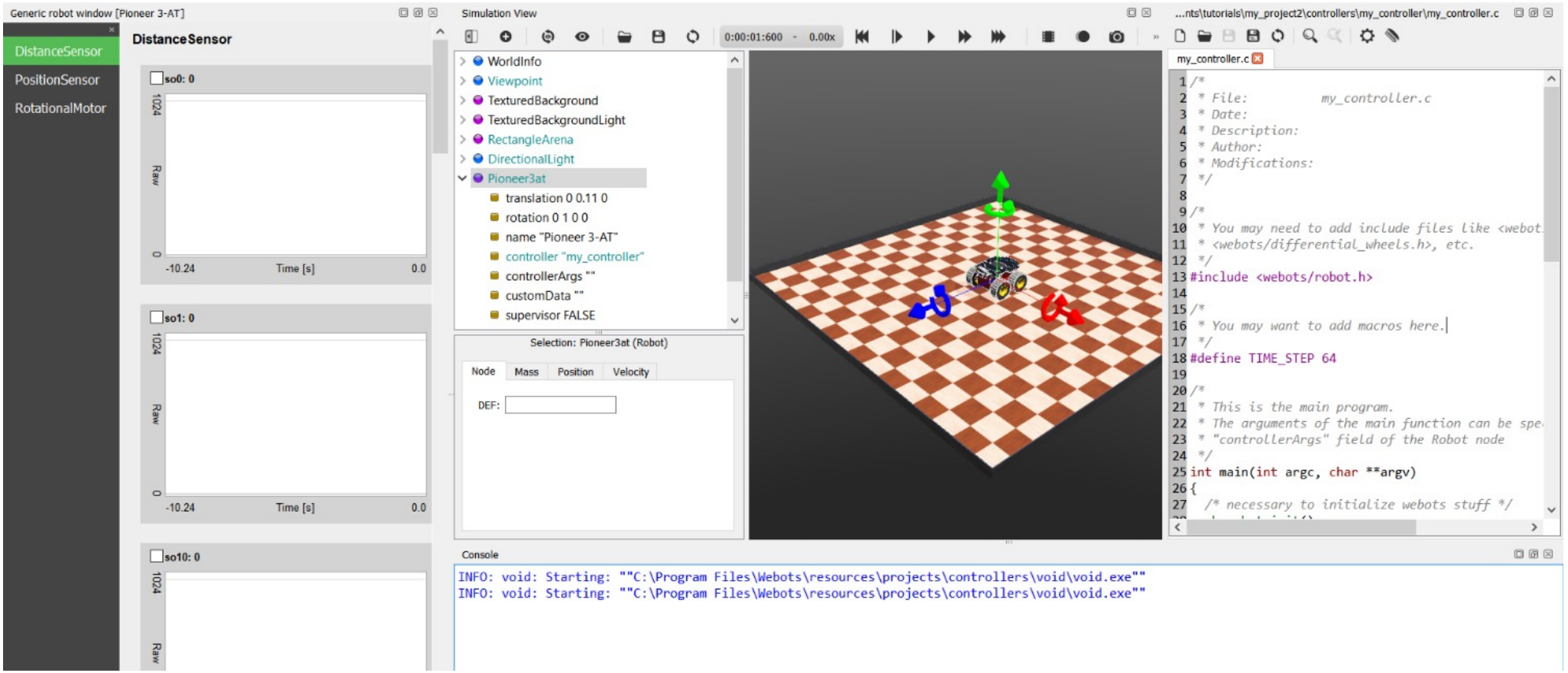
- **Relacionar o Controller com o Robô !!!!!**



Simulador Webots®

Simulando....

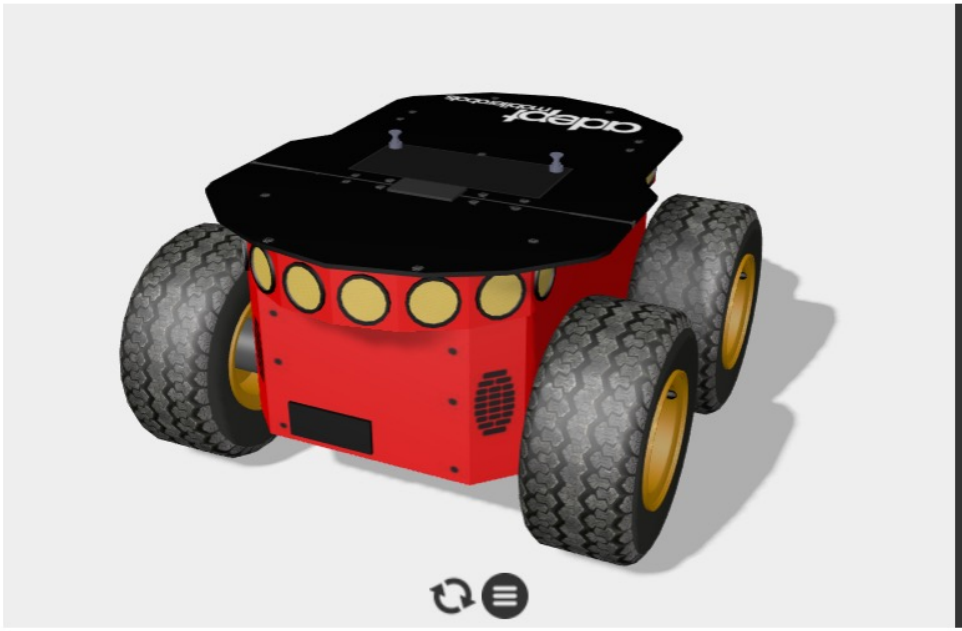
- **Tudo Pronto !!! Vamos começar a programar**
- **Dê um duplo clique em cima do robô.**
  Os sensores e atuadores irão aparecer ao lado.

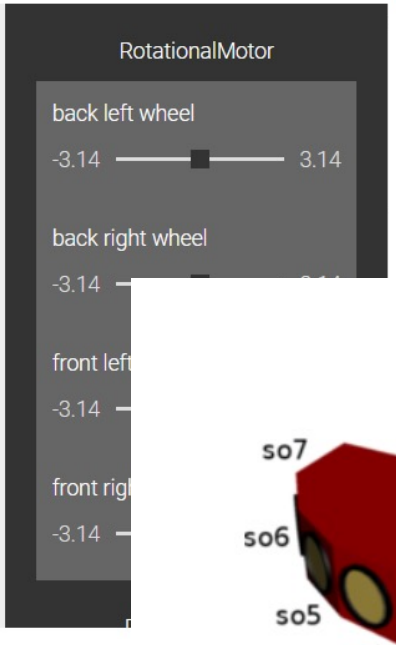


Pioneer3at definitions

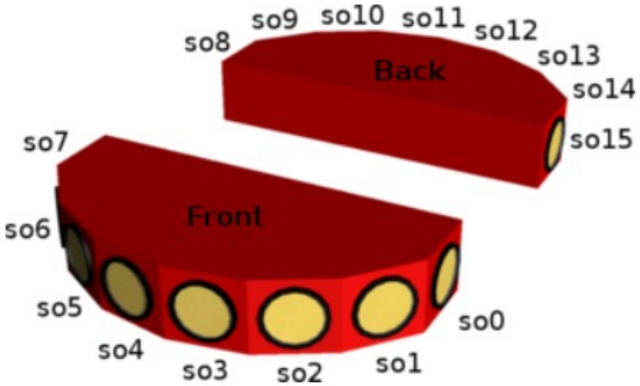
Webots User Guide R2019a

Adept's Pioneer 3-AT





Characteristics	*Values*
Length	508 mm
Width	497 mm
Height	277 mm
Weight	12 kg
Max. forward/backward speed	0.7 m/s



- **https://cyberbotics.com/doc/guide/pioneer-3at?tab=undefined**

Programando (todos)

```
/*
 * File:          my_controller.c
 * Date:
 * Description:
 * Author:
 * Modifications:
 */
```

Cabeçalho do programa

```
/*
 * You may need to add include files like <webots/distance_sensor.h> or
 * <webots/differential_wheels.h>, etc.
 */
#include <webots/robot.h>

/*
 * You may want to add macros here.
 */
#define TIME_STEP 64
```

Inserir todos os .h
<math.h> ; <stdio.h> e alguns do webots
e as macros
#define TIME_STEP 64 já é inserido automaticamente

```
/*
 * This is the main program.
 * The arguments of the main function can be specified by the
 * "controllerArgs" field of the Robot node
 */
int main(int argc, char **argv)
{
  /* necessary to initialize webots stuff */
  wb_robot_init();

  /*
   * You should declare here WbDeviceTag variables for storing
   * robot devices like this:
   *  WbDeviceTag my_sensor = wb_robot_get_device("my_sensor");
   *  WbDeviceTag my_actuator = wb_robot_get_device("my_actuator");
   */
```

O código principal função main.
Ela vem com dois argumentos que podem ser usados.
Por enquanto podemos deixar apenas **int main ()**

Aqui temos que definir os sensores, atuadores e tudo que vamos usar na programação

```
  /* main loop
   * Perform simulation steps of TIME_STEP milliseconds
   * and leave the loop when the simulation is over
   */
  while (wb_robot_step(TIME_STEP) != -1) {

    /*
     * Read the sensors :
     * Enter here functions to read sensor data, like:
     *  double val = wb_distance_sensor_get_value(my_sensor);
     */

    /* Process sensor data here */

    /*
     * Enter here functions to send actuator commands, like:
     * wb_differential_wheels_set_speed(100.0,100.0);
     */
  };

  /* Enter your cleanup code here */

  /* This is necessary to cleanup webots resources */
  wb_robot_cleanup();

  return 0;
}
```

Dentro do looping do TIME_STEP colocamos
Leituras dos sensores
Código de decisão do que fazer
Sistemas de Controle
Valores pros atuadores

Bibliotecas

Assim como todo programa em C, temos que incluir as bibliotecas que estão em arquivos com extensão .h

Temos que incluir os .h dos sensores e dos atuadores que são do próprio Webots.

```
#include <webots/robot.h>
#include <webots/distance_sensor.h>
#include <webots/motor.h>
```

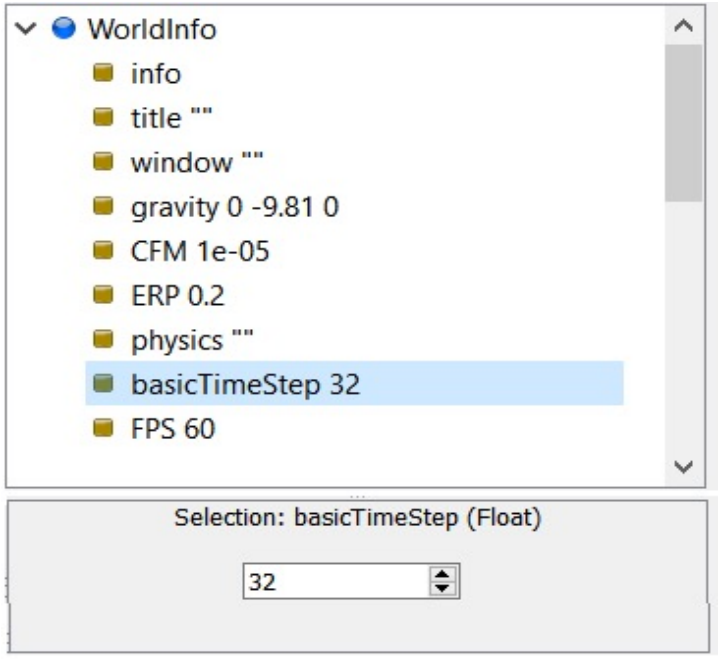
E se formos fazer operações matemáticas, imprimir informações no console, entre outras, temos que inserir as bibliotecas correspondentes:

```
#include <math.h>
#include <stdio.h>
```

Macros

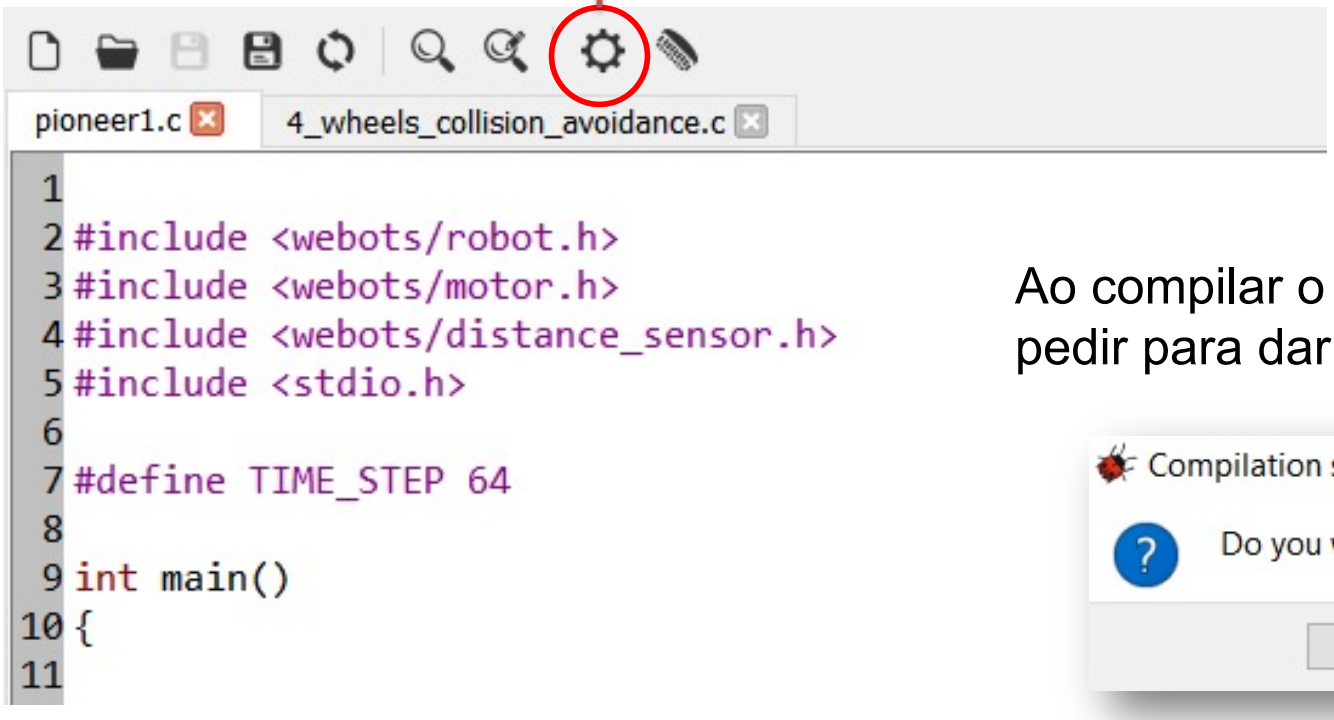
- `#define TIME_STEP 64`
- **Duração de cada passo da simulação física.**
  - **Essa macro será usada como argumento para a função wb_robot_step e também será usada para habilitar os dispositivos.**

Essa duração é especificada em milissegundos e deve ser um múltiplo do valor no campo basicTimeStep do nó WorldInfo

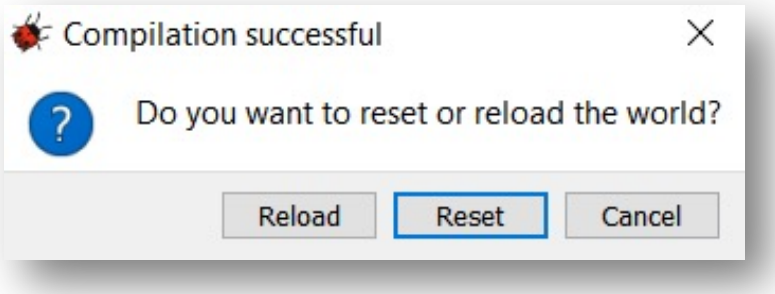


Código pronto ?

Para compilar o código !



Ao compilar o código ele vai salvá-lo e pedir para dar RESET no WORLD

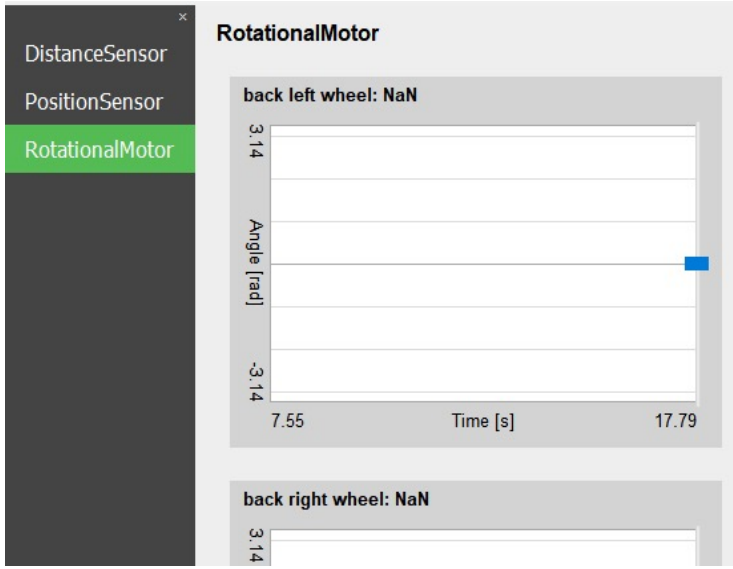


Configurando e definindo variáveis

```
int main()
{
  wb_robot_init();

  WbDeviceTag left_motor_1 = wb_robot_get_device("front left wheel");
  WbDeviceTag right_motor_1 = wb_robot_get_device("front right wheel");
  WbDeviceTag left_motor_2 = wb_robot_get_device("back left wheel");
  WbDeviceTag right_motor_2 = wb_robot_get_device("back right wheel");

  WbDeviceTag sensf = wb_robot_get_device("so3");
  WbDeviceTag sensb = wb_robot_get_device("so11");
```

Definir as variáveis (TAGs) para cada motor e cada sensor que iremos usar
wb_robot_get_device

Agora temos as variáveis
left_motor_1; left_motor_2; right_motor_1; right_motor_2;
Relacionados com cada motor do robô e as variáveis
sensf **relacionada ao sensor frontal so3**
e sensb **relacionado com o sensor traseiro so11**



Habilitando os sensores

```
int main()
{
  wb_robot_init();

  WbDeviceTag left_motor_1 = wb_robot_get_device("front left wheel");
  WbDeviceTag right_motor_1 = wb_robot_get_device("front right wheel");
  WbDeviceTag left_motor_2 = wb_robot_get_device("back left wheel");
  WbDeviceTag right_motor_2 = wb_robot_get_device("back right wheel");

  WbDeviceTag sensf = wb_robot_get_device("so3");
  WbDeviceTag sensb = wb_robot_get_device("so11");



  wb_distance_sensor_enable(sensf, TIME_STEP);
  wb_distance_sensor_enable(sensb, TIME_STEP);
```

TEMOS que habilitar cada sensor de distância que iremos usar:

wb_distance_sensor_enable

relacionando as variáveis que serão habilitadas (sensf **e** sensb**) e colocando o passo de leitura: no caso nossa macro TIME_STEP**

Habilitando os motores

```
int main()
{
  wb_robot_init();

  WbDeviceTag left_motor_1 = wb_robot_get_device("front left wheel");
  WbDeviceTag right_motor_1 = wb_robot_get_device("front right wheel");
  WbDeviceTag left_motor_2 = wb_robot_get_device("back left wheel");
  WbDeviceTag right_motor_2 = wb_robot_get_device("back right wheel");

  WbDeviceTag sensf = wb_robot_get_device("so3");
  WbDeviceTag sensb = wb_robot_get_device("so11");
```

```
  wb_distance_sensor_enable(sensf, TIME_STEP);
  wb_distance_sensor_enable(sensb, TIME_STEP);
```

```
  wb_motor_set_position(left_motor_1, INFINITY);
  wb_motor_set_position(right_motor_1, INFINITY);
  wb_motor_set_position(left_motor_2, INFINITY);
  wb_motor_set_position(right_motor_2, INFINITY);
```

TEMOS que habilitar os motores, colocando eles numa posição específica ou deixando eles livres para girar (INFINITY)

wb_motor_set_position

Fazendo robô andar

```
double left_speed = 1.0;
double right_speed = 1.0;
double sensf_value;
double sensb_value;
```

Criando variáveis para as velocidades das rodas

Criando variáveis para receber os valores dos sensores

```
while (wb_robot_step(TIME_STEP) != -1) {
```

Looping infinito com o passo definido em TIME_STEP

```
  sensf_value = wb_distance_sensor_get_value(sensf);
  sensb_value = wb_distance_sensor_get_value(sensb);
```

Lendo os Sensores

```
  printf("Sensor front:%f   Sensor Back:%f\n",sensf_value,sensb_value);
  fflush(stdout);
```

Mostrando os valores dos Sensores no console

```
  wb_motor_set_velocity(left_motor_1, left_speed);
  wb_motor_set_velocity(left_motor_2, left_speed);
  wb_motor_set_velocity(right_motor_1, right_speed);
  wb_motor_set_velocity(right_motor_2, right_speed);
```

Colocando as velocidades nos motores/rodas

```
};
  wb_robot_cleanup();
 return 0;
}
```

Função delay (milissegundos)

O Webots não aceita a função *delay* padrão do C porque senão o processo de simulação fica parado. Com isso, fiz uma função *delay*, que aceita como parâmetro um inteiro representando milissegundos, que não para a simulação.

Basta colocar essa função antes da int main() e usar.
Exemplo para delay de 2 segundos: delay(2000)

```
void delay(int time_milisec)
{
double currentTime, initTime, Timeleft;
double timeValue = (double)time_milisec/1000;
initTime = wb_robot_get_time();
Timeleft =0.00;
while (Timeleft < timeValue)
 {
  currentTime = wb_robot_get_time();
  Timeleft=currentTime-initTime;
  wb_robot_step(TIME_STEP);
  }
}
```

Webots™
robot simulation

EXPLICANDO A FUNÇÃO:

A função `wb_robot_get_time()` retorna um double com o tempo da simulação (a mesma que você vê lá no topo da simulação)

E dentro do While tem a função `wb_robot_step(TIME_STEP)` para manter a simulação rodando dentro da função delay. Caso contrário a simulação para.

Obs.: (double) é um casting de variável, foça a variável time_milisec do tipo inteiro virar double.

Hands on (todos)

Hands On

- **Criem um código para fazer o robô ir para frente e para trás invertendo a velocidade toda vez que encontrar uma parede (pode ser em dupla).**