

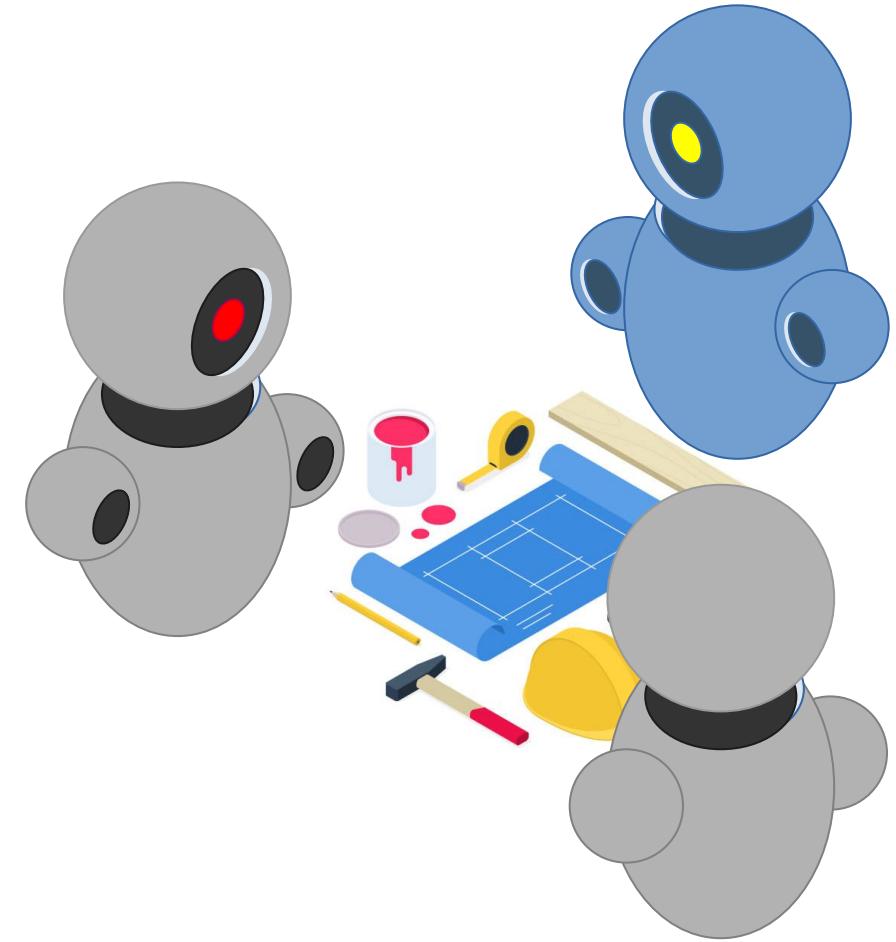


Inteligência Artificial das Coisas com Sistemas Multiagente Distribuídos e Embarcados

Prof. Me. Nilson Mori Lazarin
<http://lattes.cnpq.br/4304954704049203>

Prof. Me. Bruno Policarpo Toledo Freitas
<http://lattes.cnpq.br/1007295696070905>

INTRODUCTION





Cognitive Hardware on Networks (Chon) Research Group

<https://chon.group>

Where we have been?

This workshop is part of:

- **Undergraduate course**
 - CEFET/RJ Maria da Graça and Nova Friburgo
- **Graduate course**
 - M.Sc. and Ph.D. at UFF
 -

It was **presented at**:

IFSP, IFC, UFPel, UFRJ, UFSC, UnB, UTFPR, UDESC and Estácio;



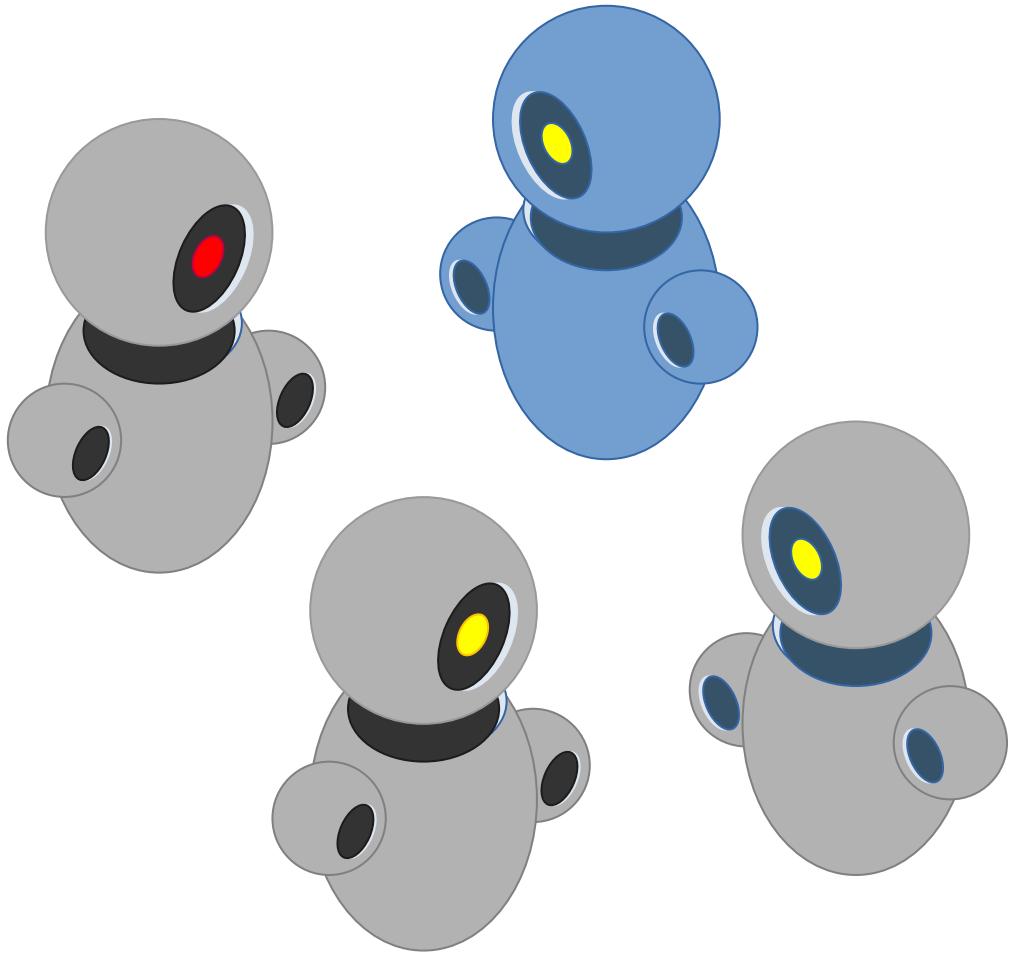
<https://workshops.chon.group>

Goals

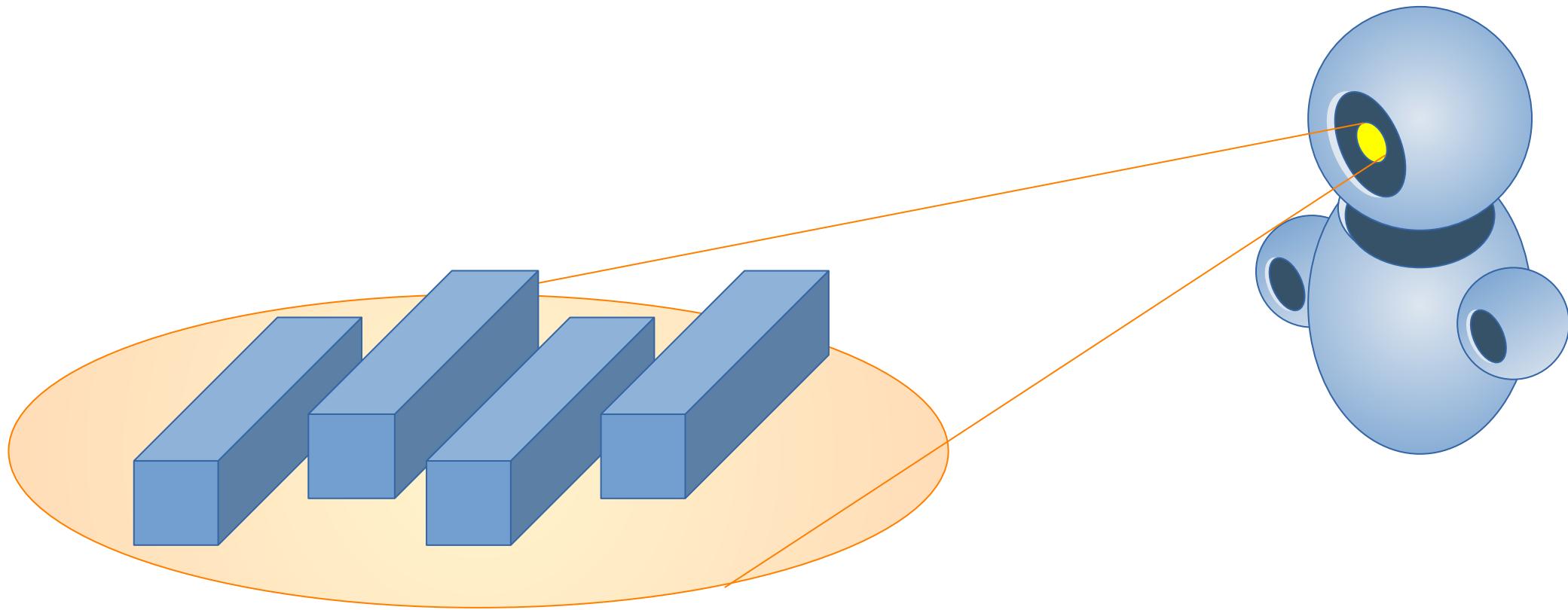
This workshop goals are:

- **To experience** the development process of **Embedded MAS** using BDI-agents and Jason.
- to learn how to **collaborate** using IoT in **distributed** scenarios.
-

THE AGENT ORIENTED APPROACH



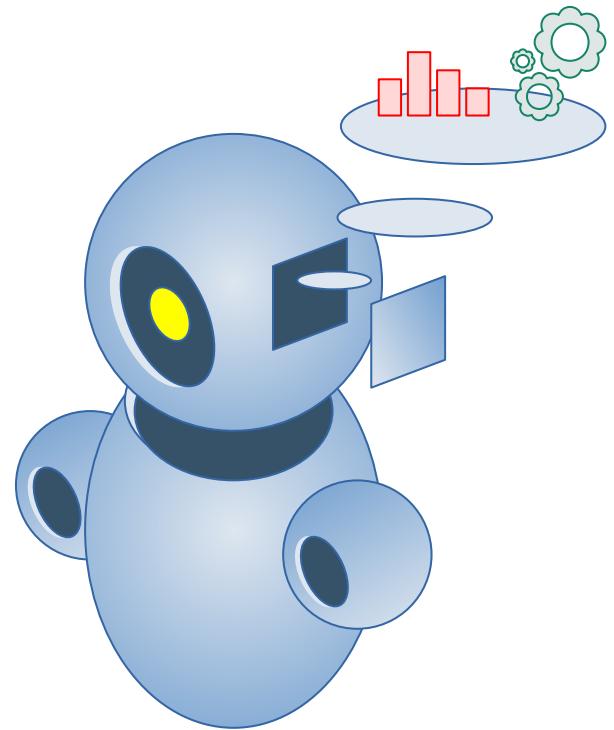
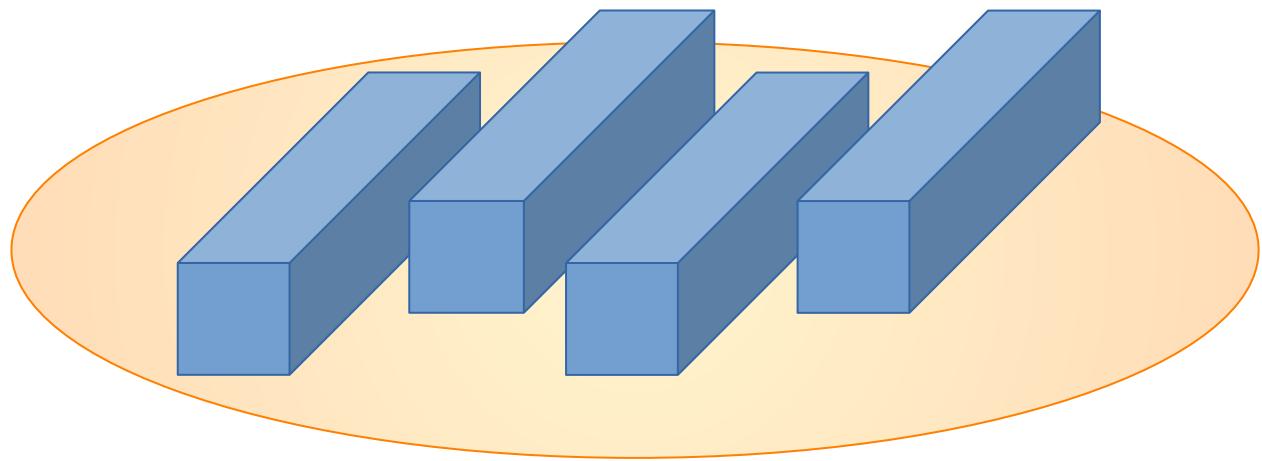
Agent



WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

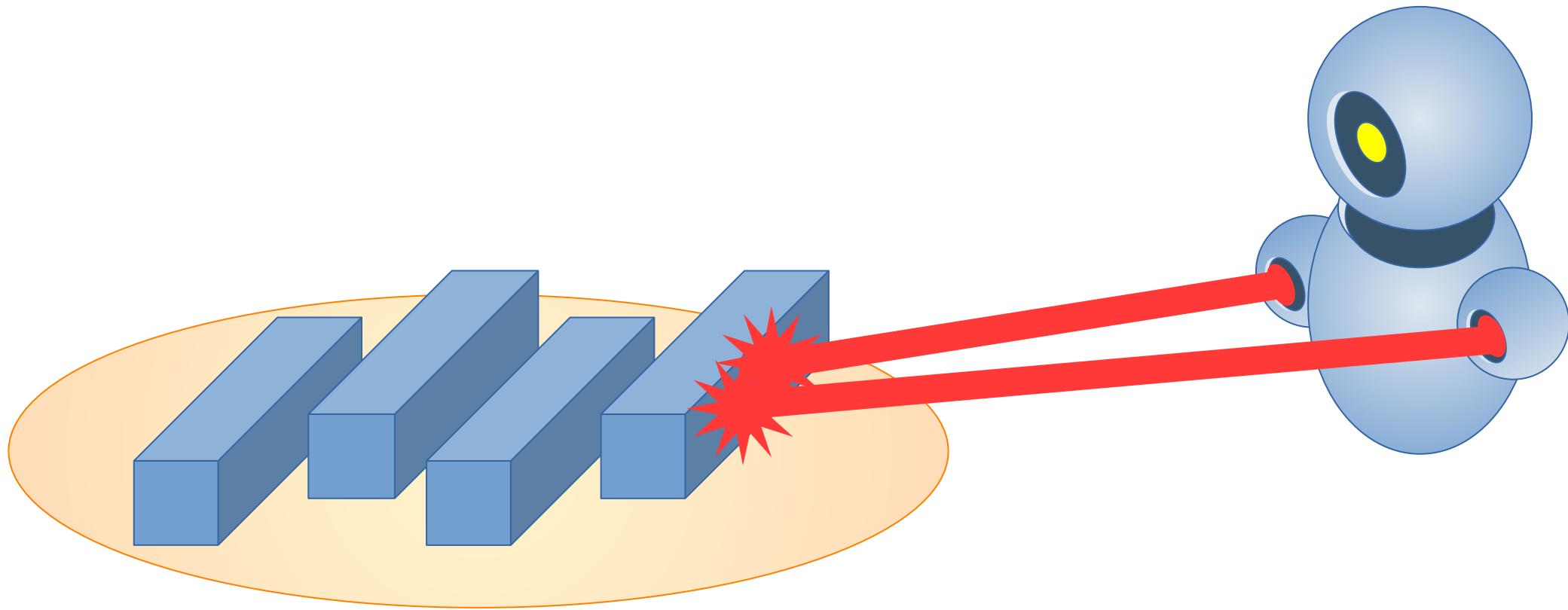
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agent



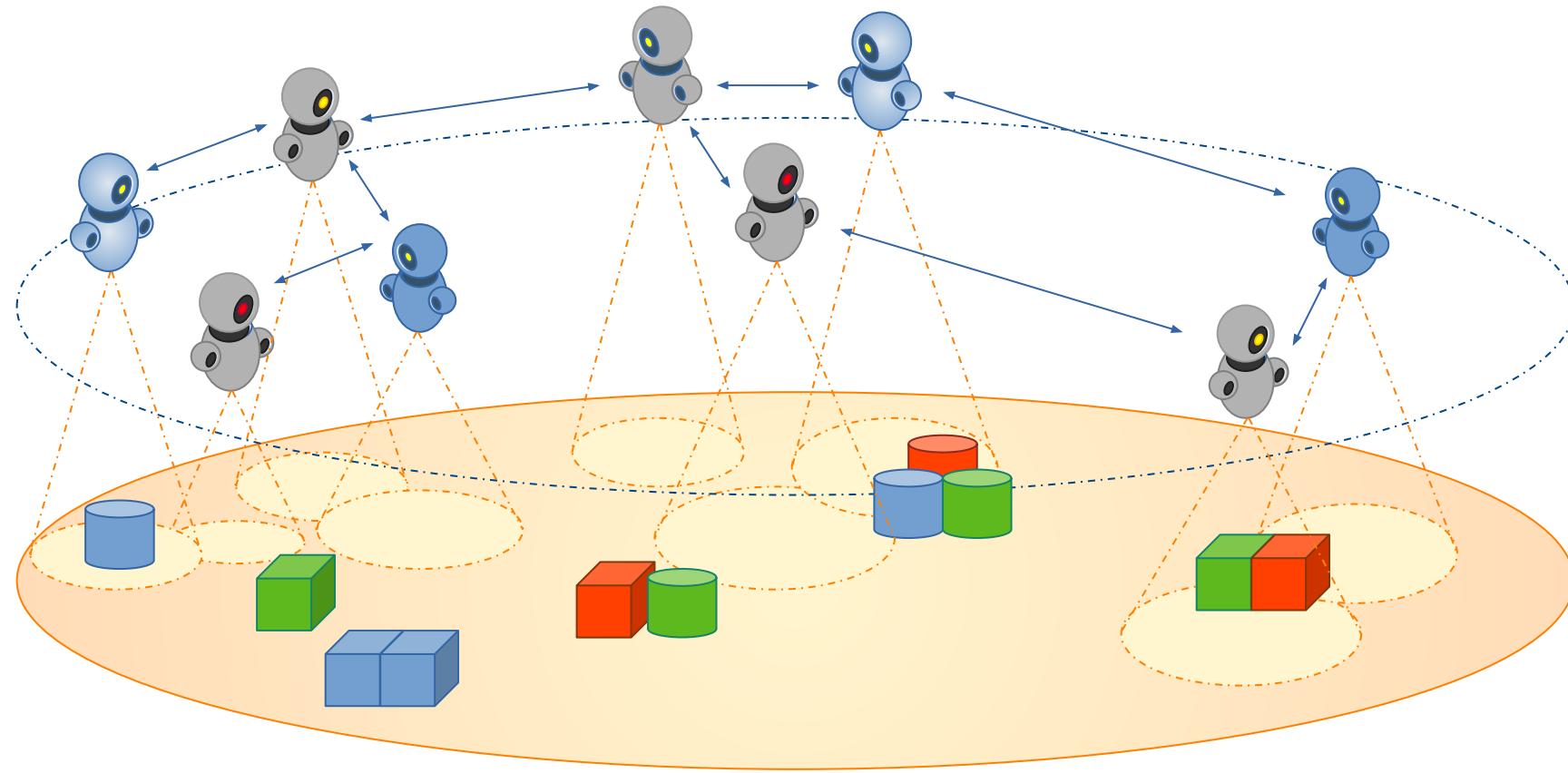
WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agent



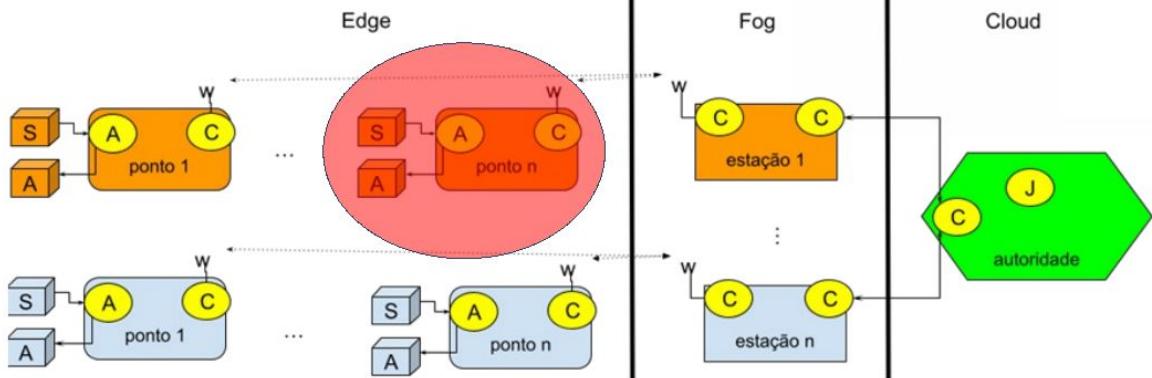
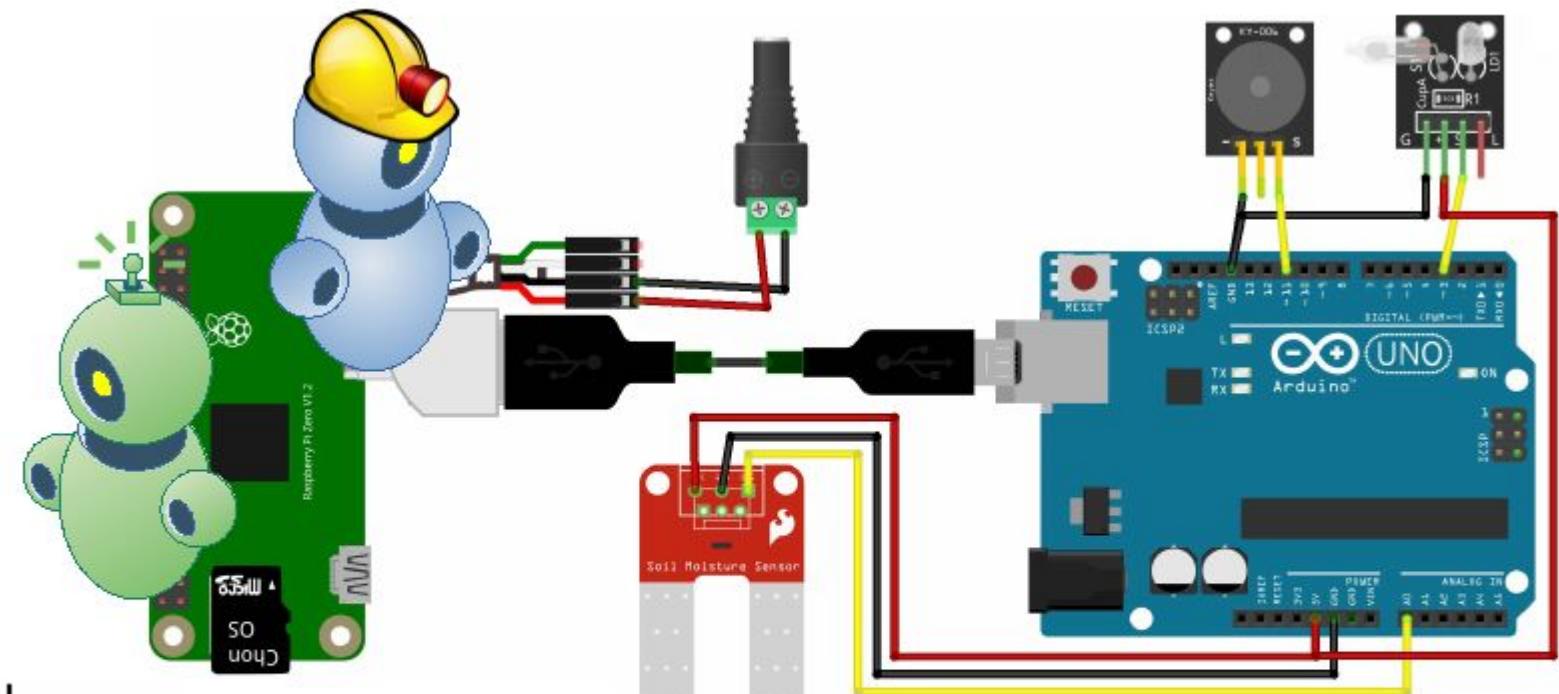
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Multi-agent System



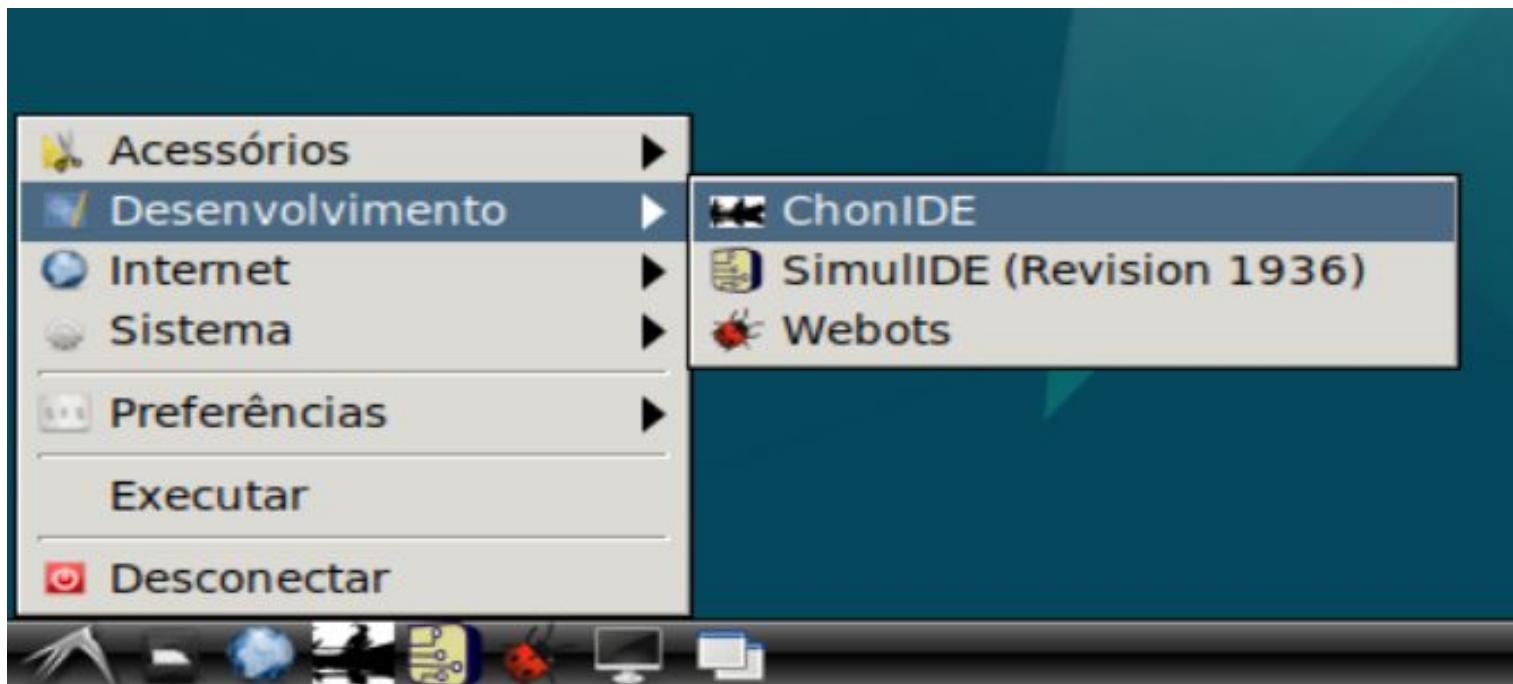
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

Embedded Multi-agent System



THE DEVELOPMENT TOOLS

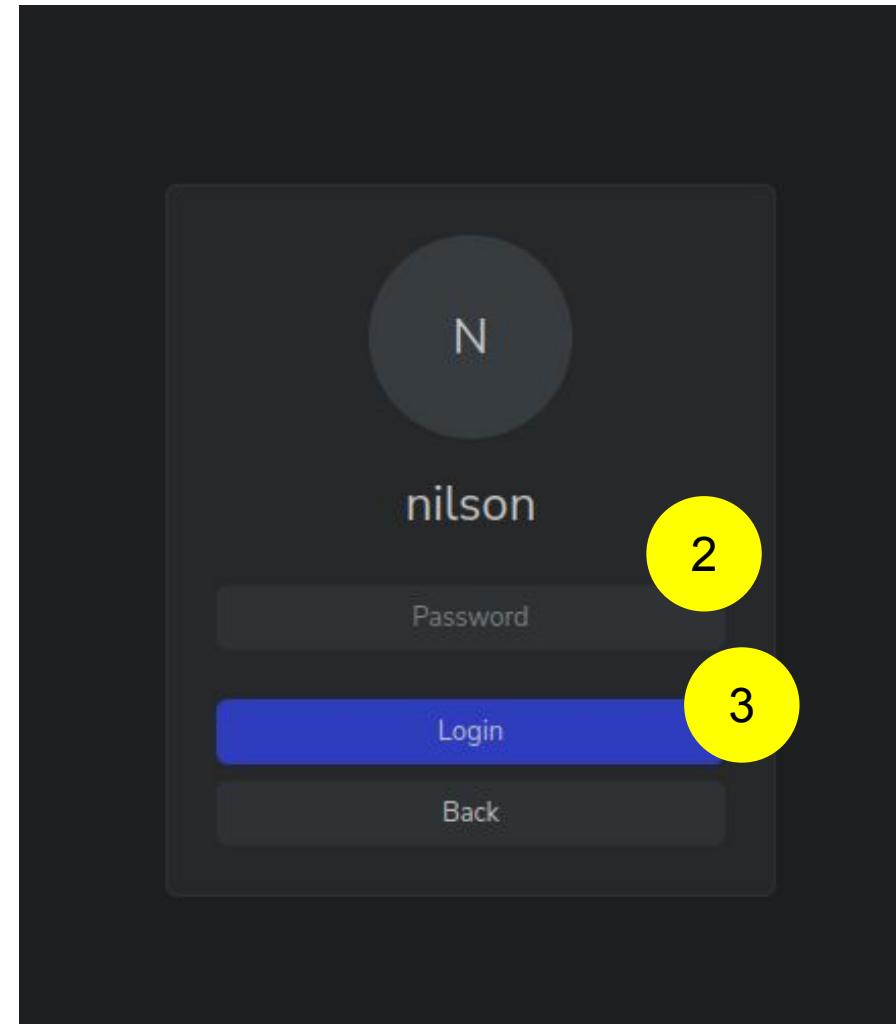
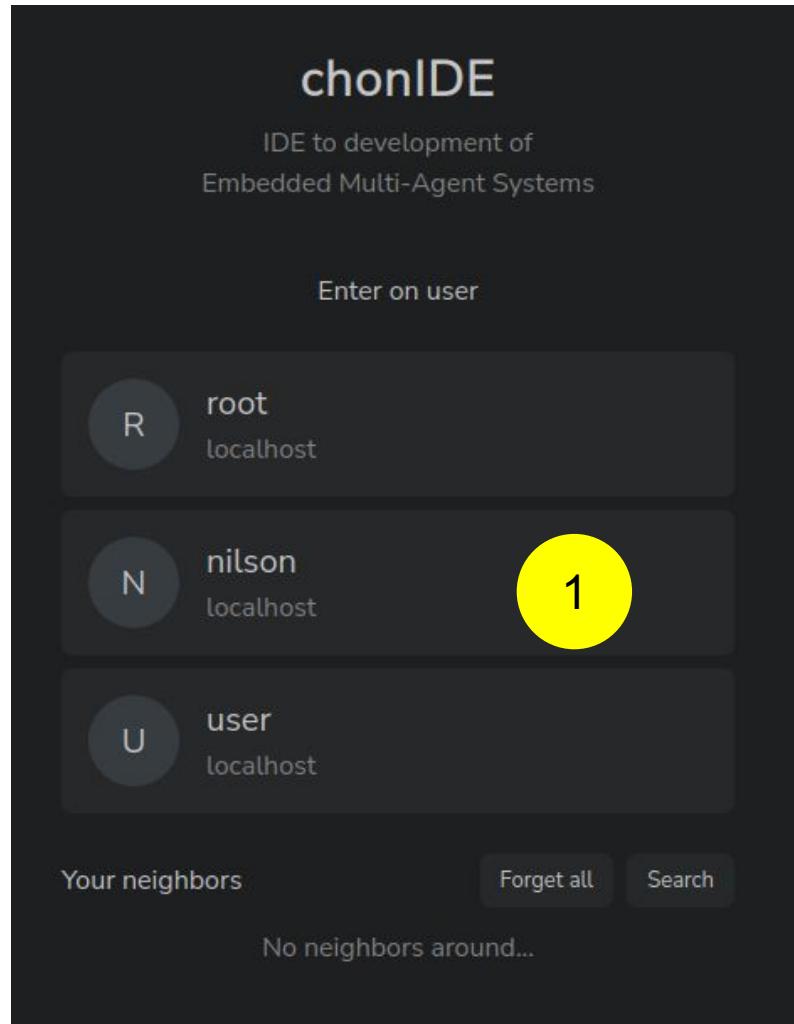




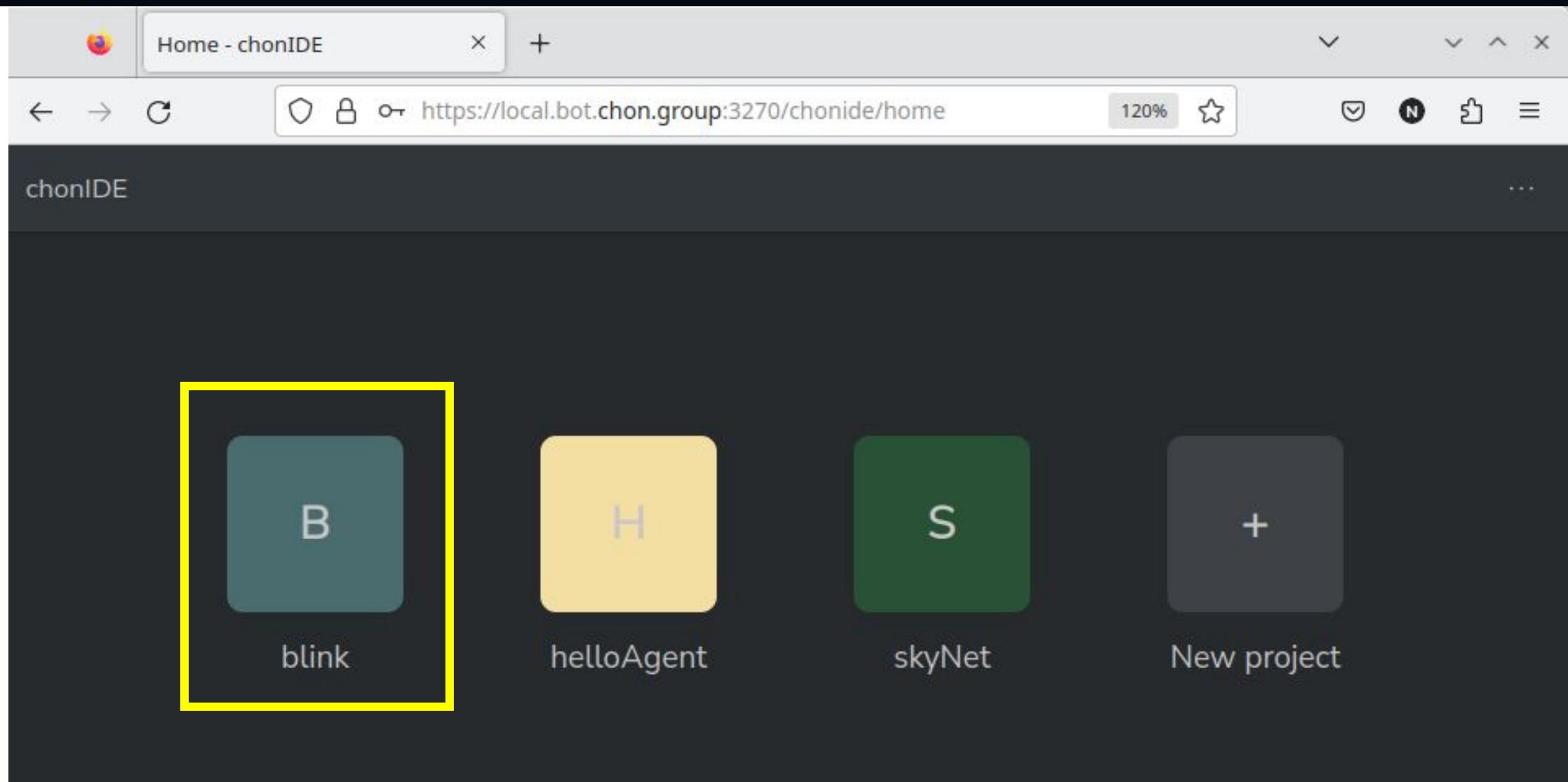
How to install?

<https://github.com/chon-group/chonIDE/wiki>

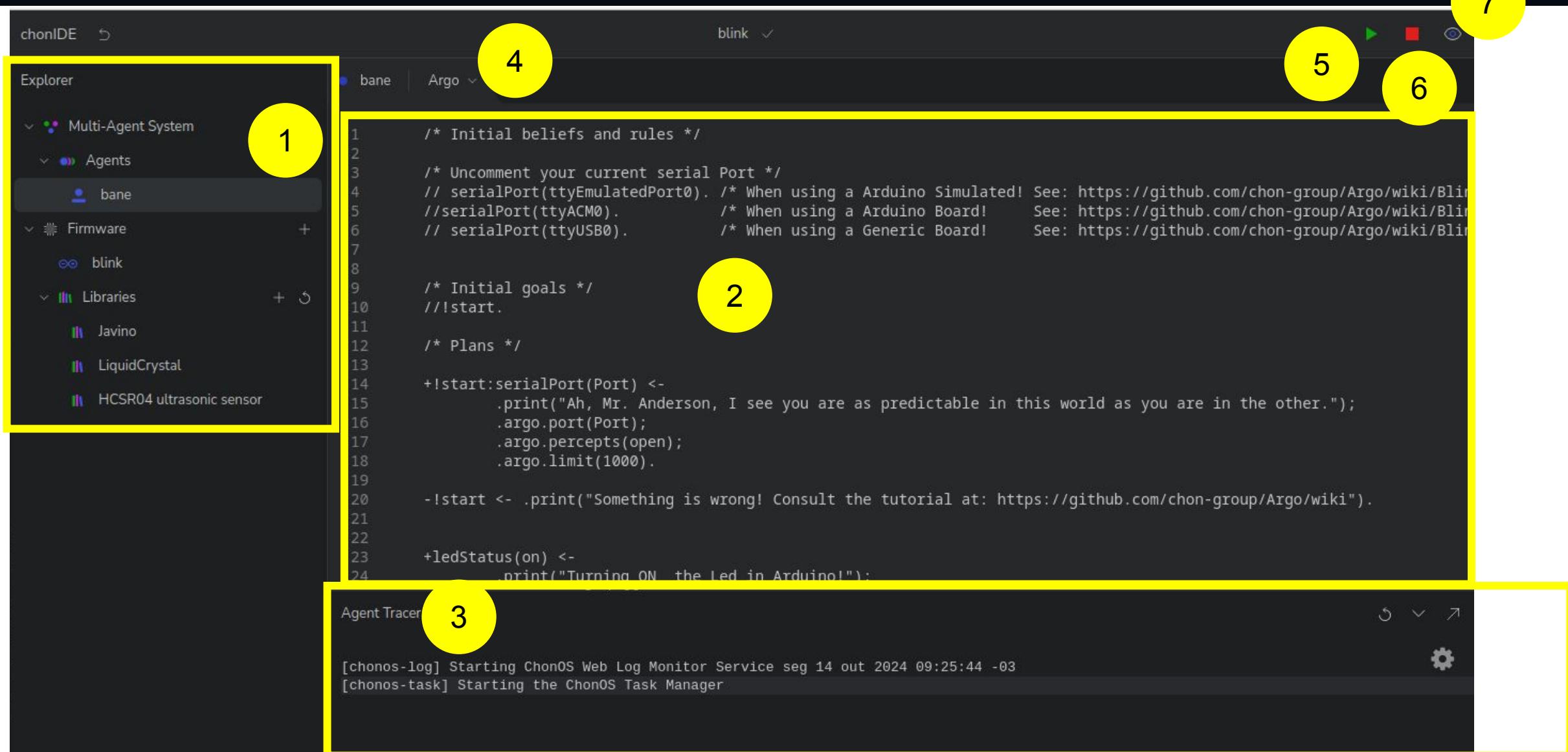
ChonIDE: login



ChonIDE: blink



ChonIDE: blink



ChonIDE: blink



ChonIDE: mindInspector

Agents

- bane
by Jason

Inspection of agent **bane** (cycle #11301)

- Beliefs

ledStatus(off)[source(percept)]
port(ttyACM0,on)[source(percept)]
resourceName(myArduino)[source(percept)]
serialPort(ttyACM0)[source(self)]

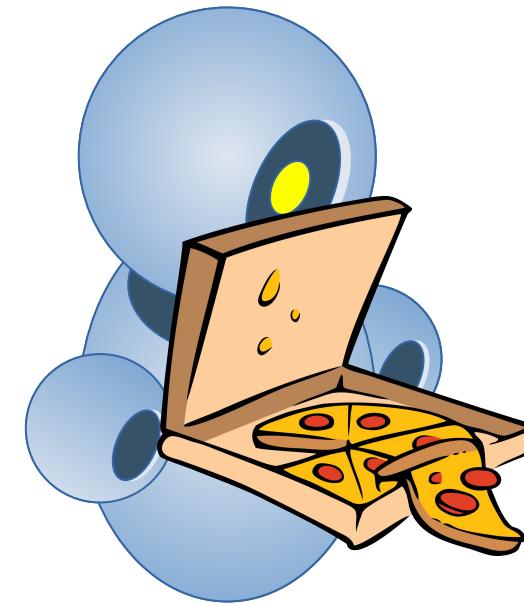
- Annotations

+ Status

[hide annotations](#) | [plan library](#)

Job! ").
INAMED --add-op
g

MY FIRST SINGLE AGENT



Arquitetura BDI

O BDI possibilita a implementação de atitudes mentais em agentes cognitivos:

- **Crença** é uma informação, ou seja, aquilo que o agente sabe sobre o ambiente, si mesmo ou outros agentes.
- **Desejo** é um propósito que o agente busca tornar realidade.
- **Intenção** é uma ação que o agente faz para alcançar um desejo.

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

AgentSpeak(L)

Um agente AgentSpeak(L) possui minimamente:

- Uma base de crenças;
- Uma lista de objetivos;
- Uma biblioteca de planos.

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

Jason Framework: Crenças

Em Jason, um agente armazena as **informações** percebidas do ambiente; as informações internas; e informações de comunicação através de crenças.

As **crenças** são armazenadas em uma **Base de Crenças** (Belief Base).

As crenças são representadas como predicados da **lógica tradicional**. Os **predicados** representam propriedades particulares.

Jason Framework: Beliefs Types

1. Mental Notes (self)

- Informações adicionadas na base de crenças pelo próprio agente.

2. Communication (agent)

- Informações obtidas pelo agente através da interação com outros agentes.

3. Environmental Perceptions (percepts)

- Informações coletadas pelo agente que são relativas ao sensoriamento constante do ambiente.

Initial Beliefs: Mental Notes

```
● smith Jason ▾  
1 sunny.
```



Inspection of agent **smith**

- Beliefs

sunny[source(self)].

- Annotations

Initial Beliefs: Mental Notes

```
● smith | Jason ▾  
1 myName(smith).  
2 salary(1000).
```



Inspection of agent **smith**

- Beliefs

myName(smith)[source(self)].
salary(1000)[source(self)].

- Annotations

Initial Beliefs: Mental Notes

```
● smith    Jason ▾  
1 myName("Smith").  
2 salary(1500.80).
```



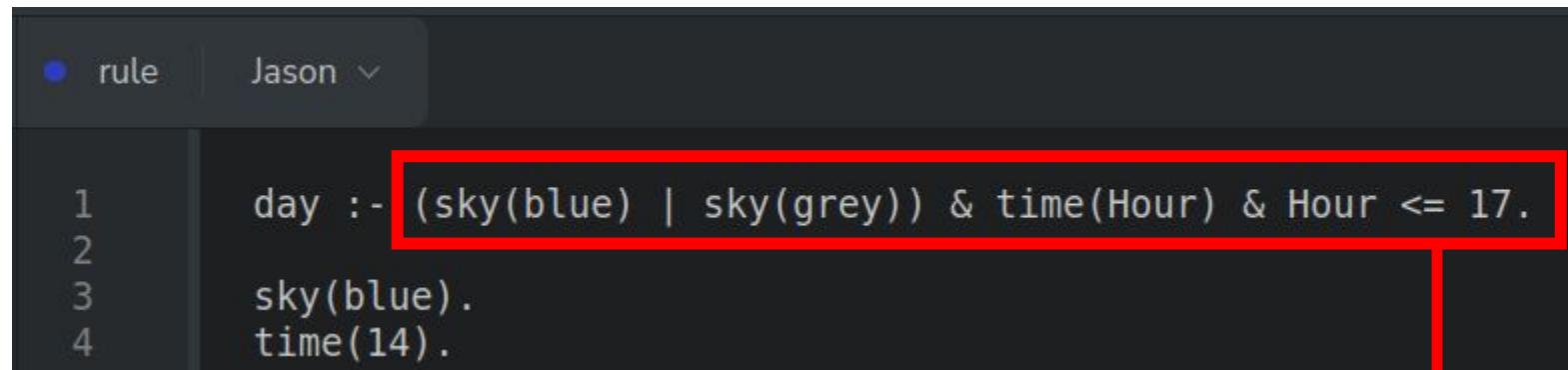
Inspection of agent smith

- Beliefs

myName("Smith")_[source(self)].
salary(1500.8)_[source(self)].

- Annotations

Rules: Predicate and Variable



```
rule Jason
1 day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.
2
3 sky(blue).
4 time(14).
```



The verified conditions.

Jason Framework: Plans Format

triggering_event : context <- body.

1. triggering Event

- Um agente pode ter diversos objetivos. Os planos são ativados baseados nos eventos que podem ser ativados em determinado momento.

2. context

- São as condições para a ativação de um plano dentro vários eventos.

3. body.

- É o corpo do plano. Uma sequência de ações a ser executada pelo agente.

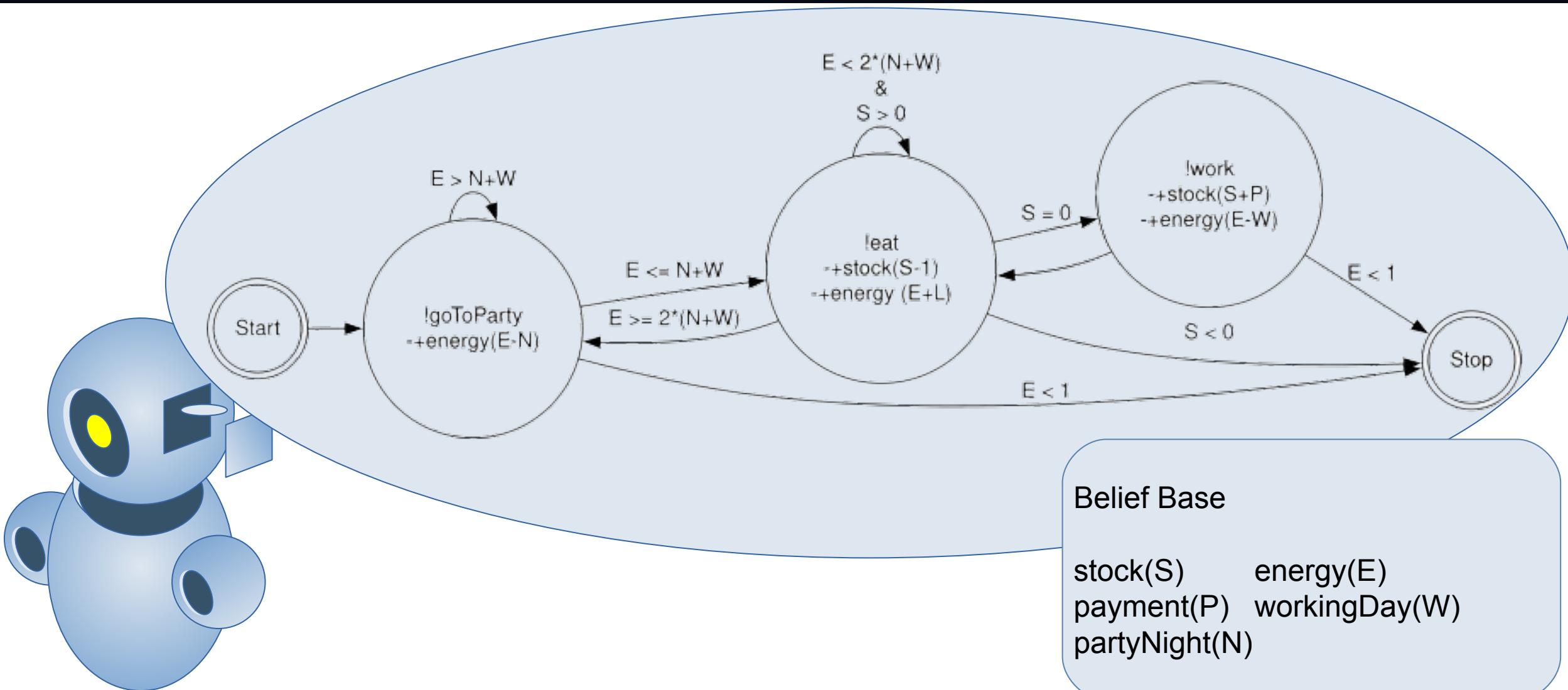
Plan: Hello World

```
bob Jason ▾  
1 !start.  
2  
3 +!start <-  
4 .print("I'm Alive!").
```



```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.  
NOTE: Picked up JDK_JAVA_OPTIONS: --add-opens=java.base/  
rmi/sun.rmi.transport=ALL-UNNAMED  
Jason Http Server running on http://127.0.1.1:3272  
[bob] I'm Alive!
```

My First Agent



My First Agent: Code

```
1  /* Initial beliefs and rules */
2  stock(0).          // This is a Belief that represents the pizza stock.
3  payment(+2).        // ... represents how many pizzas does Giacomo get by a working day?
4  energy(5).          // ... represents the Giacomo life energy.
5  workingDay(-1).     // ... represents how much energy does Giacomo lose in a working day?
6  partyNight(-2).     // ... represents how much energy does Giacomo lose on a party night?
7  lifeParameters(S,E,P,W,N) :- stock(S) & energy(E) & payment(P) & workingDay(W) & partyNight(N). //this is a RULE!
8
9  /* Initial goals */
10 !startGiacomoLife.
11
12 /* Plans */
13 +!startGiacomoLife <- !startClock; !goToParty.
14 +!goToParty: lifeParameters(S,E,P,W,N) & (E+(N+W)>1) <- -+lastParty(0); -+energy(E+N); .wait(1000); !goToParty.
15 +!goToParty: lifeParameters(S,E,P,W,N) & (E+(N+W)<=1)<- !eat.
16 +!eat: lifeParameters(S,E,P,W,N) & (E<2*(-1*(N+W))) & S>0 <- -+stock(S-1); -+energy(E+1); .wait(1000); !eat.
17 +!eat: lifeParameters(S,E,P,W,N) & (E>=2*(-1*(N+W))) <- !goToParty.
18 +!eat: lifeParameters(S,E,P,W,N) & S=0 <- !work.
19 +!work: lifeParameters(S,E,P,W,N) <- -+stock(S+P); -+energy(E+W); !eat.
20 +!startClock <- +lastParty(0); !!tickingClock.
21 +!tickingClock <- .wait(100); ?lastParty(Time); -+lastParty(Time+100); !tickingClock.
22
23 +energy(E): E < 1 <- .print("Giacomo died of hunger!"); .stopMAS.
24 +stock(S): S < 0 <- .print("Without food!"); .stopMAS.
25 +lastParty(Time): Time > 10000 <- .print("Giacomo died of sadness!"); .stopMAS.
```

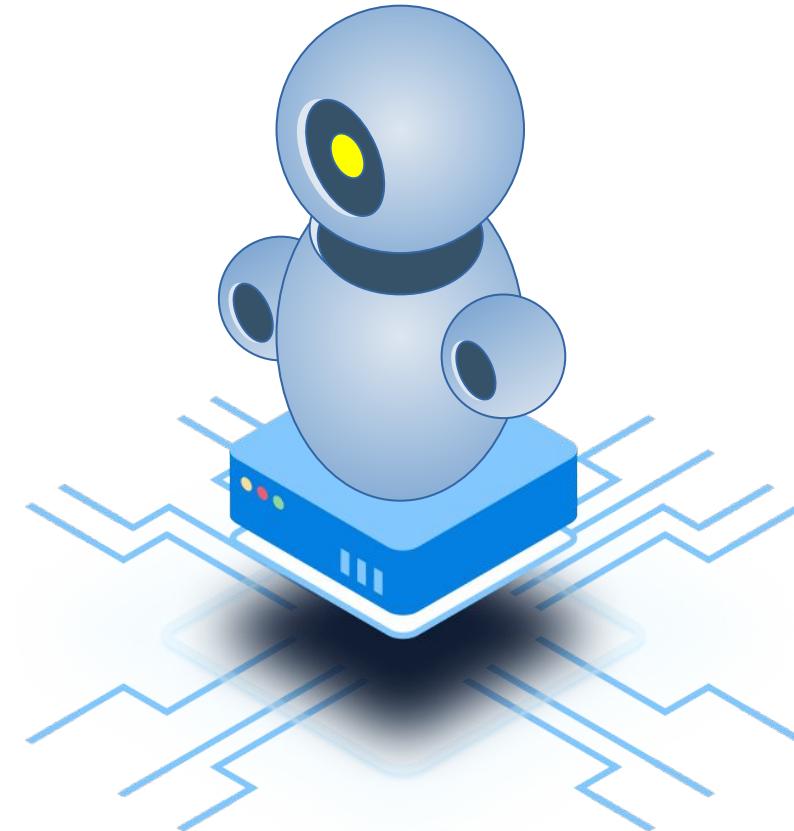
<https://github.com/chon-group/distributedAndEmbeddedAI/blob/main/workshops/sepex/2024/projects/giacomoAgent.chon>



JASON stdlib → <https://jason-lang.github.io/api/jason/stdlib/package-summary.html>

Introdução ao Jason → https://www.researchgate.net/publication/228828757_Introducao_ao_desenvolvimento_de_sistemas_multiagentes_com_Jason

EMBEDDED MAS



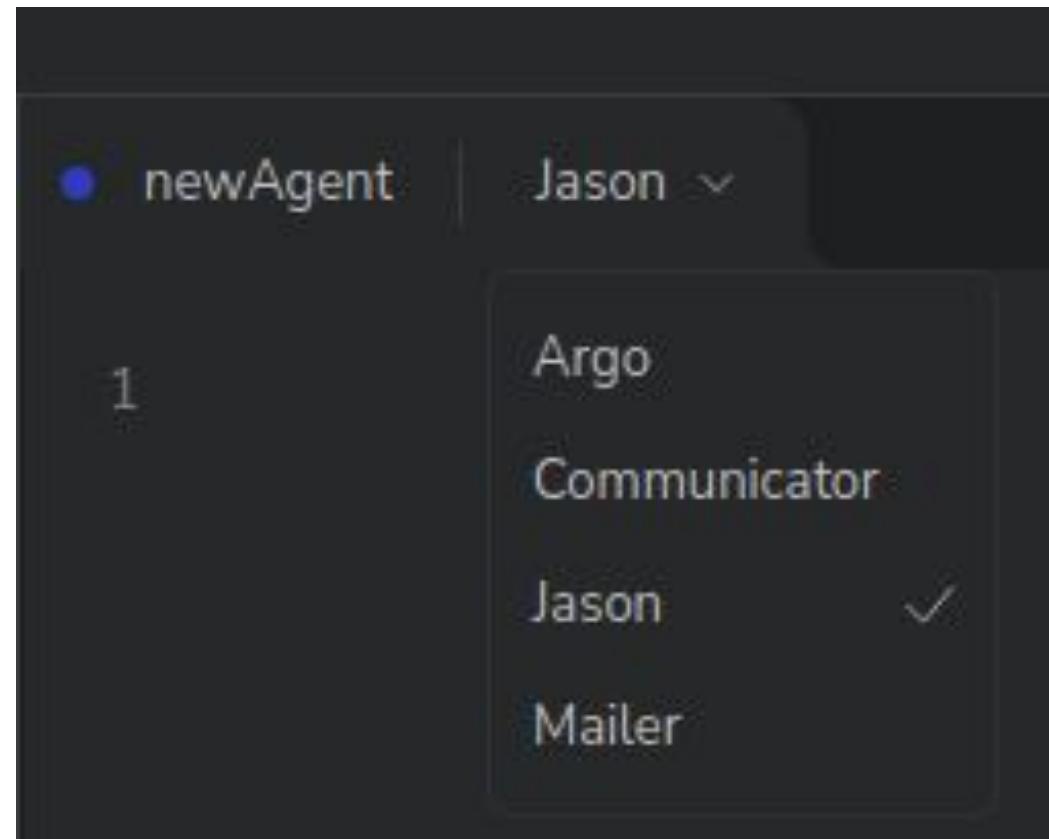
Jason Embedded

Jason Embedded is a spin-off Version of Jason for IoT and Embedded MAS:

- **Autonomy** as a characteristic an Embedded MAS provides to a device that does not need external architectures to control and manage it.
- **Communicability** is the agent's ability to communicate with other agents in its MAS or another MAS.
- **Mobility** as the agents' ability to move from one MAS to another.
- **Fault Tolerance and Adaptability** are defined as the ability of an agent to identify if a physical resource is absent or not answering commands and to overcome this situation by modifying its goals.
- **Distributed Ledger Technologies** technologies to facilitate the agreement between agents.

Jason Embedded

- **Jason Embedded** also provides **new types of agents** to control hardware devices and to communicate with agents hosted in other MAS.



Jason Embedded

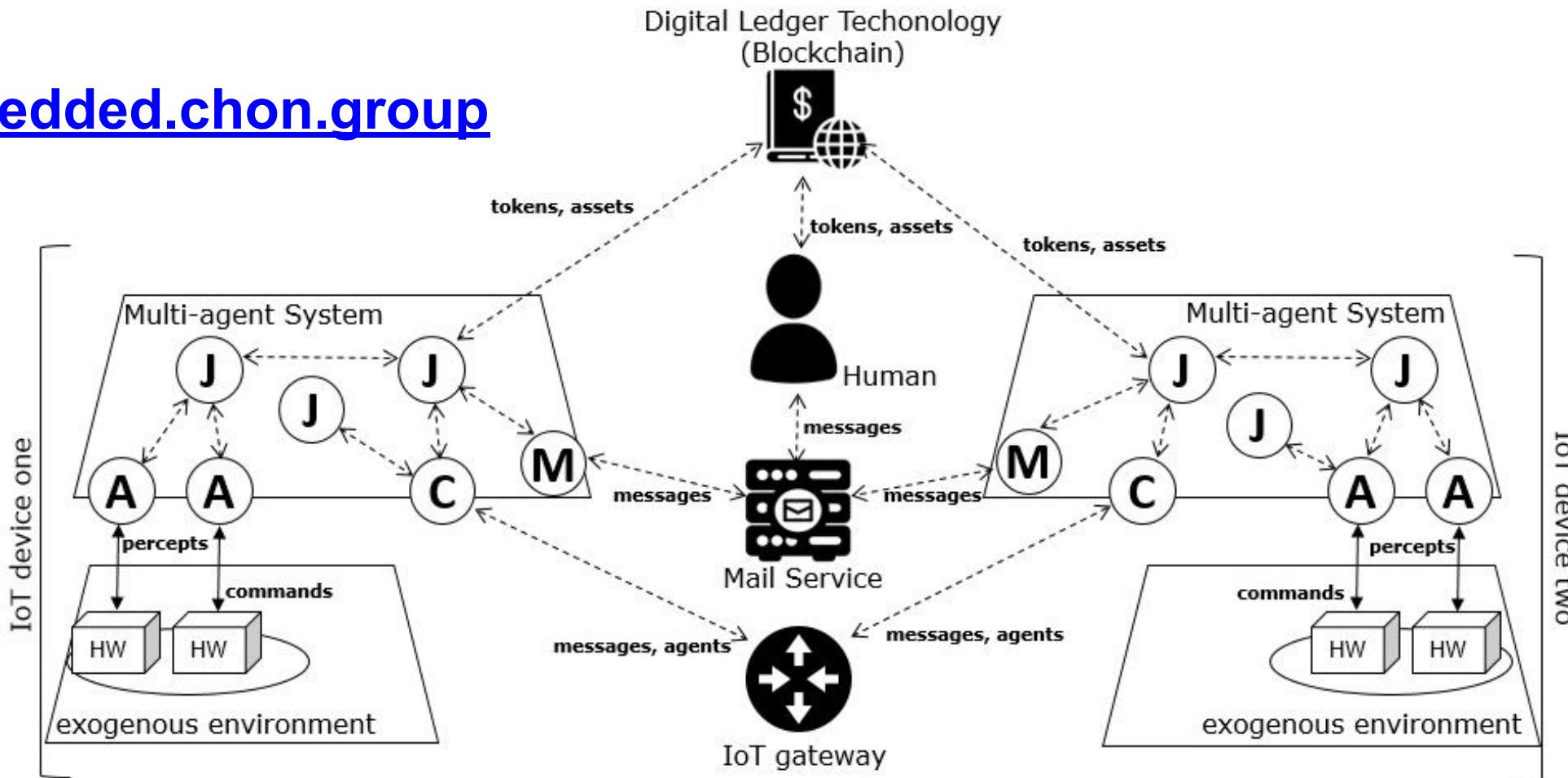
- **Jason Embedded** also provides **new types of agents** to control hardware devices and to communicate with agents hosted in other MAS.
 - **ARGO**: is a customized agent architecture built on the Jason framework that extends Jason's standard agents by adding the **ability to control microcontrollers** [Pantoja et al., 2016].
 - **Hermes (Communicator)** : is another customized agent architecture built on Jason's Standard agent by adding the **ability to communicate** with agents from **other MAS**, which also have Communicator agents [Pantoja et al., 2018].
 - **Mailer** : agent architecture based on the Jason framework to enable secure and simplified **human-agent interaction** [Lazarin et al., 2024].

Jason Embedded

- **Mobility** is based on **bio-inspired protocols** [Jesus et al., 2021].
 - The protocols simulate natural behaviors that could be explored in collaborative tasks using IoT devices.
 - One or more agents or even the whole MAS could move from one device to another to take control of the destination device (**Predation**) or to use it as a temporary non-hazardous relationship (**Mutualism** and **Inquilinism**) if both parties accept the protocol.
- **Middleware** that allows cognitive agents to manipulate Tokens and NFT in blockchain network [Lazarin et al., 2023].

Jason Embedded

<https://jasonembedded.chon.group>



Standard Agent



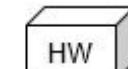
Argo Agent



Communicator Agent



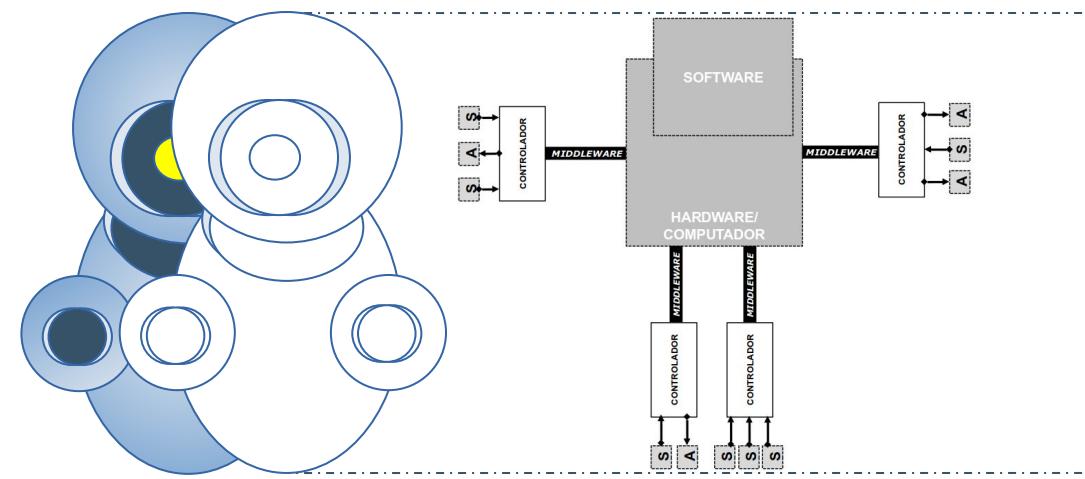
Mailer Agent



Hardware

ARGO AGENT

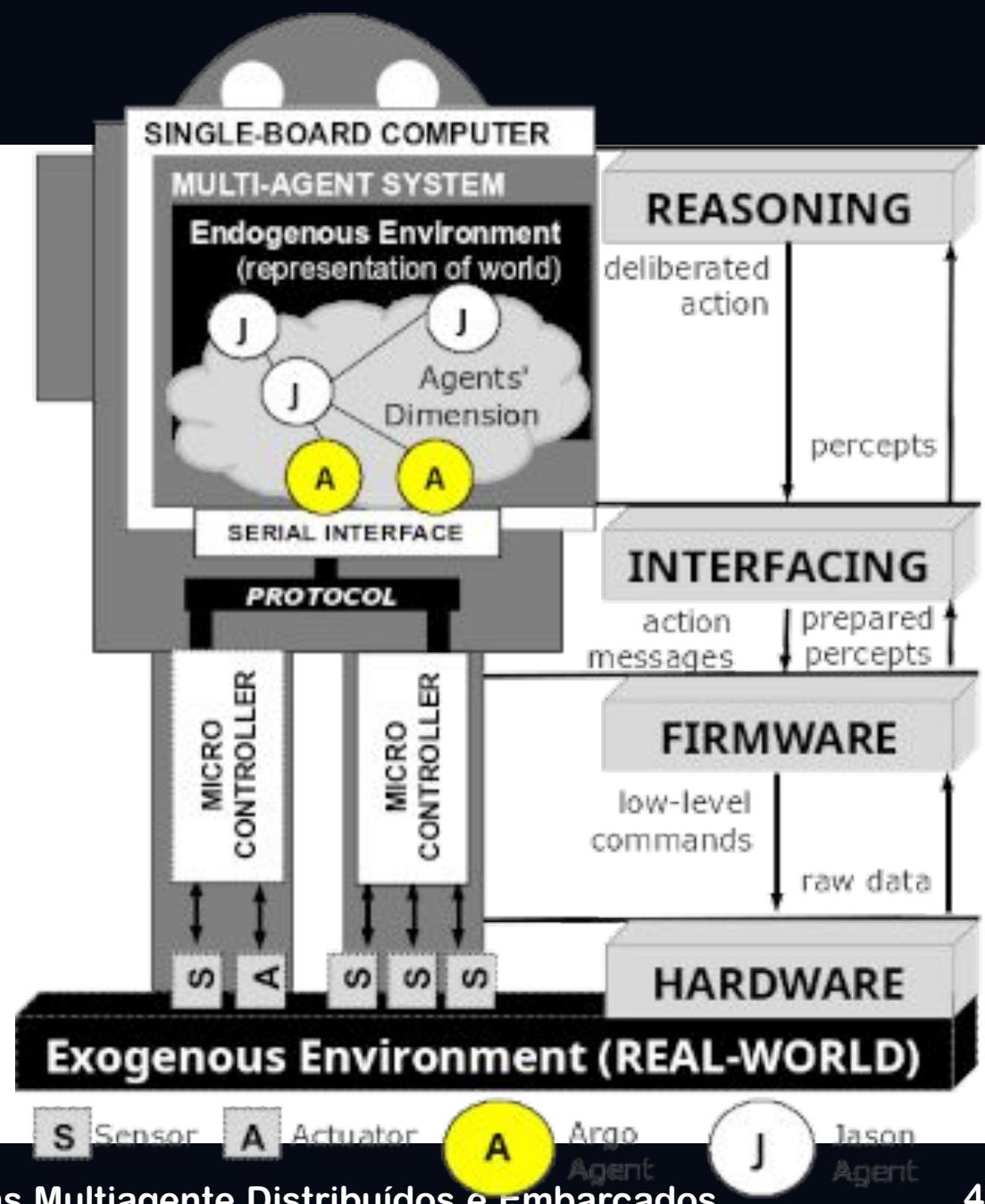
<https://argo.chon.group>



Argo for Jason

ARGO is a customized agent architecture that **interfaces** sensors and hardware actuators.

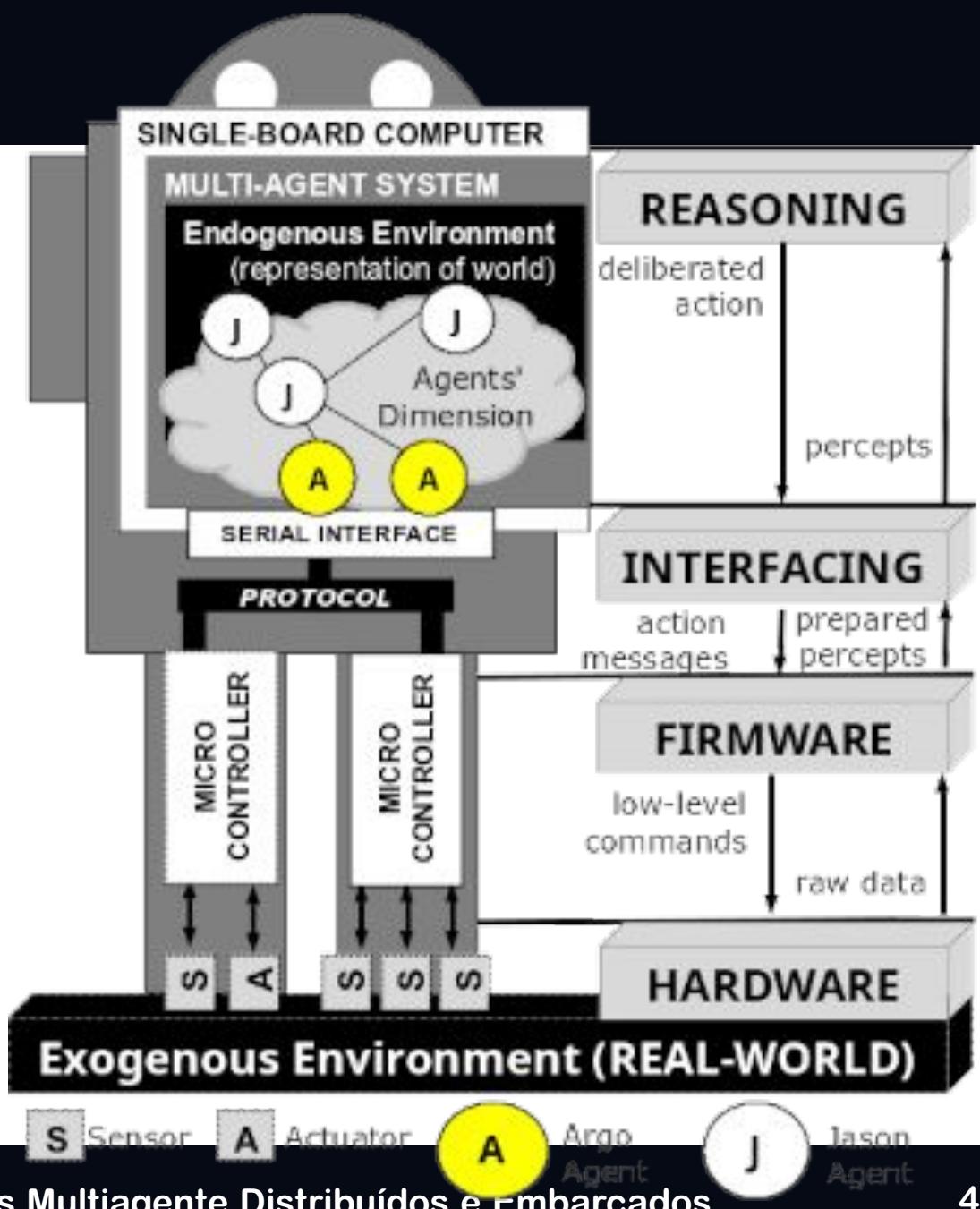
ARGO aims to be a practical architecture for **programming embedded BDI agents** using **Jason** and microcontroller boards.



Argo for Jason

The **ARGO** allows to:

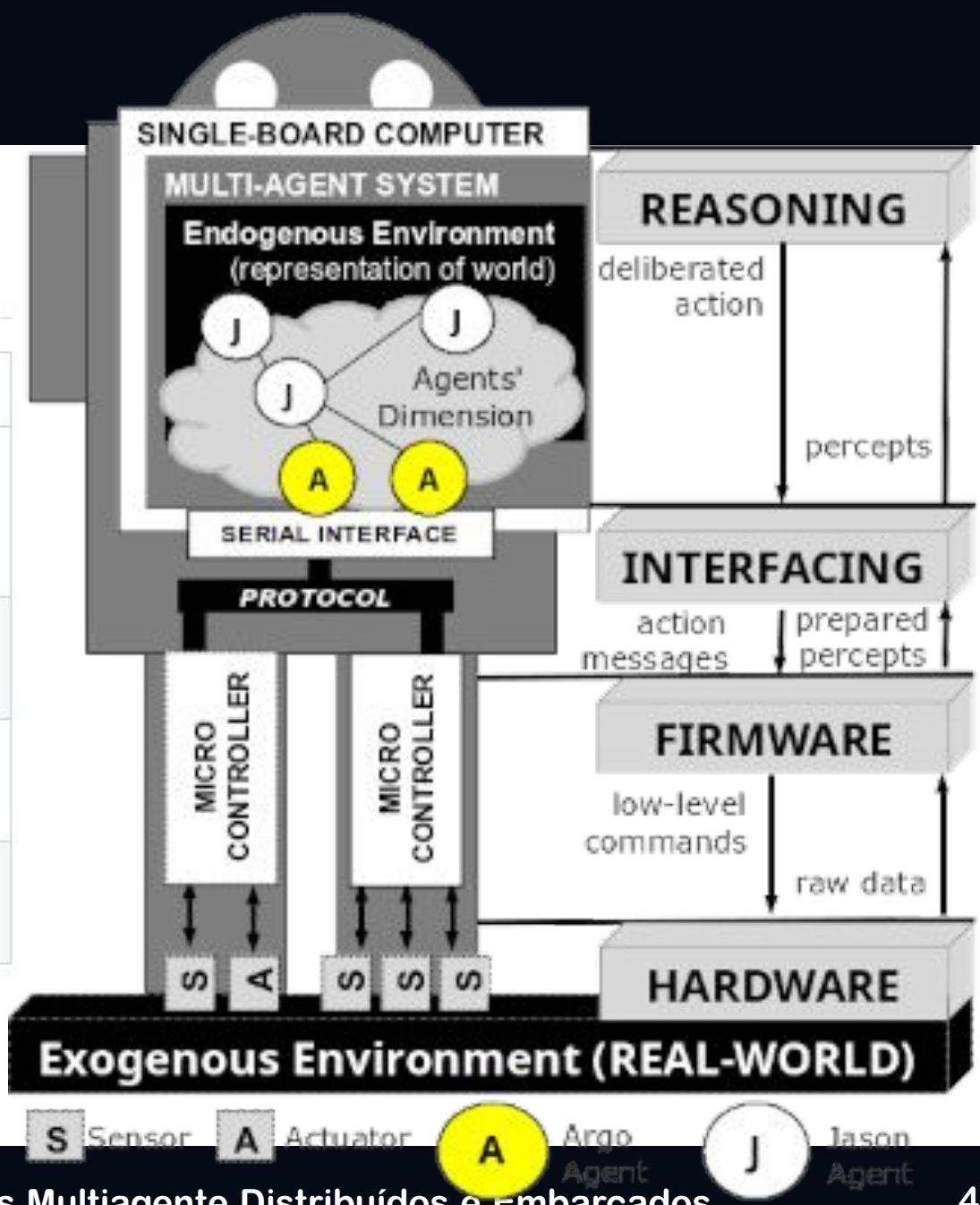
1. to direct control actuators at runtime;
2. to receive sensor perceptions within a predefined period automatically;
3. to change which devices are being accessed at runtime;
4. to communicate with other agents in Jason;
5. whether or not to perceive the real world at runtime;
6. to change perception filters at runtime.



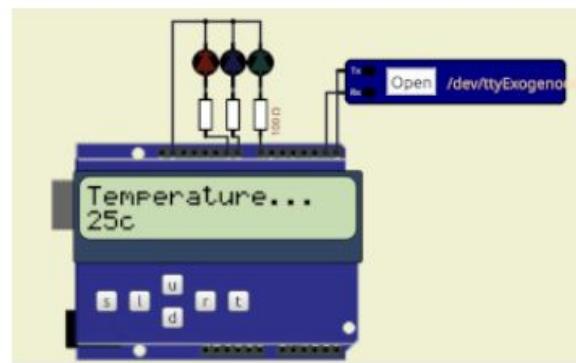
Argo for Jason

ARGO Internal Actions

Action	Description
.argo.port(P)	defines which serial port should be used by the agent, where P is a literal representing the port identification(e.g., COM8, ttyUSB0).
.argo.limit(M)	defines the sensing interval, where M is a value in milliseconds.
.argo.percepts(open close)	decides whether or not to perceive the exogenous environment (physical world).
.argo.act(A)	sends to the hardware an action, represented by literal A , to be executed by a microcontroller;



Exercise: LCD KeyPad



Adjust the ARGO agent to:

- turn on the **green LED** if the rain intensity less than 50mm;
- turn on the **yellow LED** if the rain intensity exceeds 50mm;
- turn on the **red LED** if the the rain intensity exceeds 80mm;

Percepts:

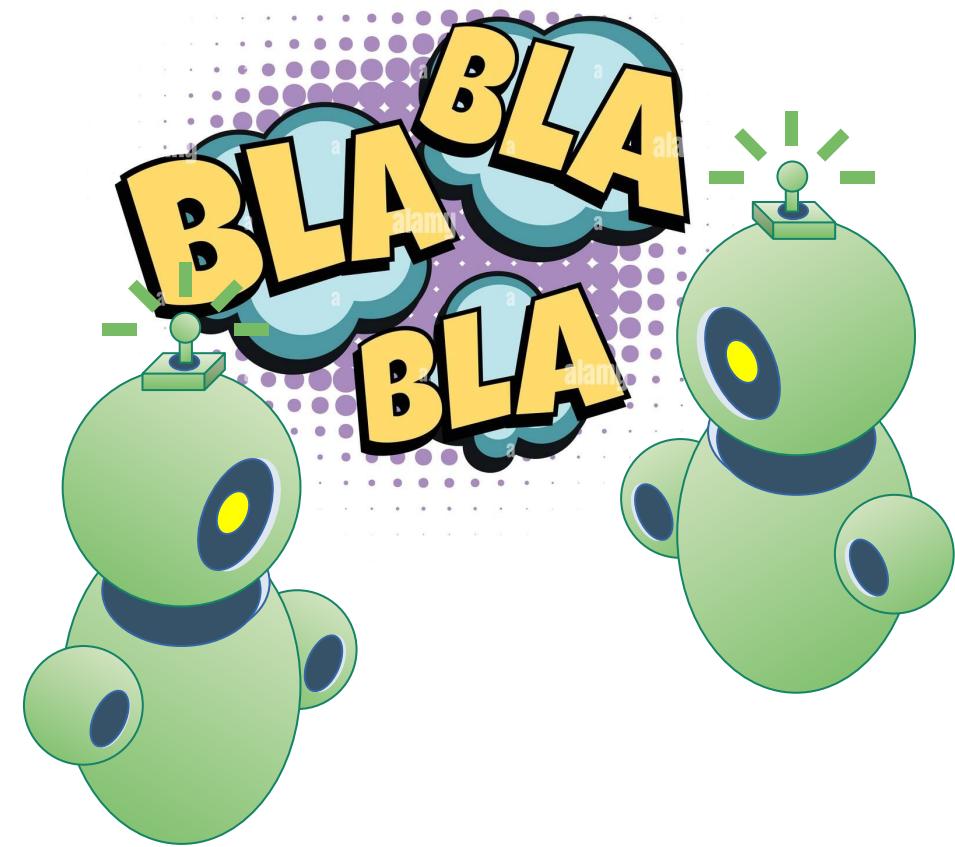
```
device(arduinoWithLCDKeypadShield)[source(percept)].           // the device name
humidity(H)[source(percept)].                                // H is a Integer
rainLast24hrs(R)[source(percept)].                           // R is a Integer
temperature(T)[source(percept)].                            // T is a Integer
```

Actions:

```
.argo.act(redAlert)                                         // turn on the RED LED
.argo.act(yellowAlert)                                       // turn on the YELLOW LED
.argo.act(greenAlert)                                       // turn on the GREEN LED
.argo.act(alertOff)                                         // turn off all LED
.argo.act(M)                                                 // print the String M in the LCD Display
```

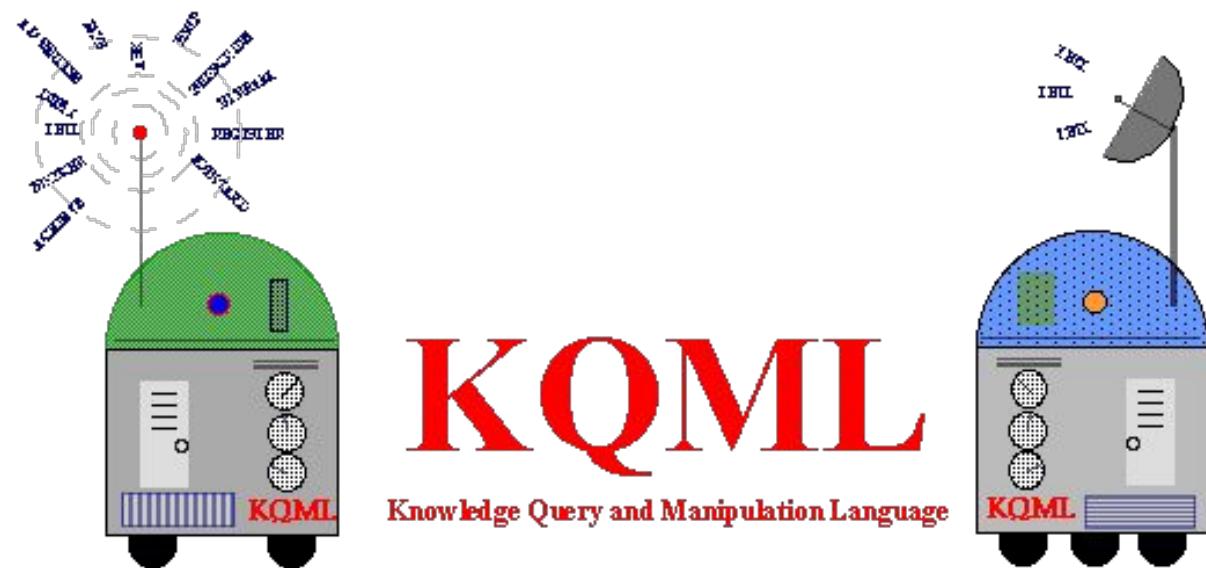
<https://github.com/chon-group/distributedAndEmbeddedAI/blob/main/workshops/sepex/2024/projects/lcdKeyPad.cho>

COMMUNICABILITY



Jason Framework: Communication

Based on the theory of Speech Acts [1] and KQML [2] (Knowledge Query and Manipulation Language), a communication protocol for knowledge-based systems.



[1] M. Bierwisch, "Semantic Structure and Illocutionary Force", em Speech Act Theory and Pragmatics, J. R. Searle, F. Kiefer, e M. Bierwisch, Orgs., Dordrecht: Springer Netherlands, 1980, p. 1–35. doi: [10.1007/978-94-009-8964-1_1](https://doi.org/10.1007/978-94-009-8964-1_1).

[2] T. Finin, R. Fritzson, D. McKay, e R. McEntire, "KQML as an agent communication language", em Proceedings of the third international conference on Information and knowledge management - CIKM '94, Gaithersburg, Maryland, United States: ACM Press, 1994, p. 456–463. doi: [10.1145/191246.191322](https://doi.org/10.1145/191246.191322).

Jason Framework: Performatives

The performatives that are currently available for agent communication in Jason are largely inspired by KQML. We also include some new performatives, related to plan exchange rather than communication about propositions. The available performatives are briefly described below, where s denotes the agent that sends the message, and r denotes the agent that receives the message [1].

PERFORMATIVE	DESCRIPTION
tell	s intends r to believe (that s believes) the sentence in the message's content to be true;
untell	s intends r not to believe (that s believes) the sentence in the message's content to be true;
achieve	s requests that r try to achieve a state of the world where the message content is true;
unachieve	s requests that r try to drop the intention of achieving a state of the world where the message content is true;
tellHow	s informs r of a plan;
untellHow	s requests that r disregard a certain plan (i.e., delete that plan from its plan library);
askIf	s wants to know if the content of the message is true for r;
askAll	s wants all of r's answers to a question;
askHow	s wants all of r's plans for a triggering event;

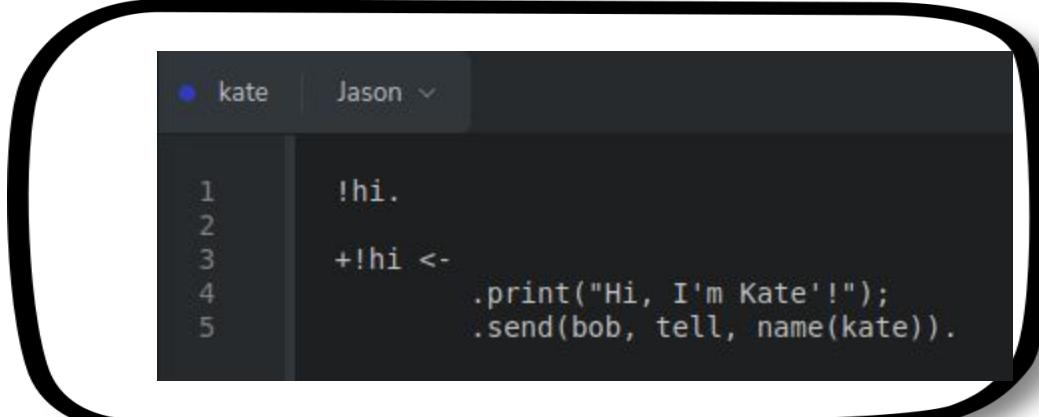
[1] R. H. Bordini e J. F. Hübler, "BDI Agent Programming in AgentSpeak Using Jason", em Computational Logic in Multi-Agent Systems, F. Toni e P. Torroni, Orgs., em Lecture Notes in Computer Science, vol. 3900. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, p. 143–164. doi: [10.1007/11750734_9](https://doi.org/10.1007/11750734_9).

Jason Framework: Message Structure

.send(receiver, ilf, message, answer, timeout)
.broadcast(ilf, message)

- **receiver**
 - Name of the agent receiving the message (or list of recipients)
- **ilf**
 - Illocutionary force of the speech act (KQML)
- **message**
 - Message content
- **answer**
 - A term that will store the response (optional field)
- **timeout**
 - Timeout in milliseconds to receive a response (optional field)

Illocutionary Forces: tell



```
!hi.  
+!hi :- .print("Hi, I'm Kate'!"); send(bob, tell, name(kate)).
```

Inspection of agent **bob**

- Beliefs

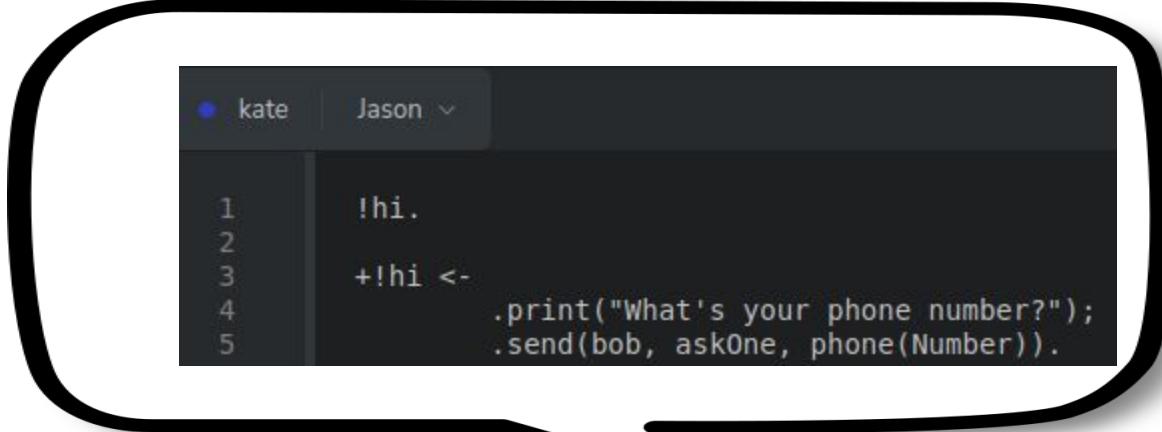
name(kate)[source(kate)].



Illocutionary Forces: achieve

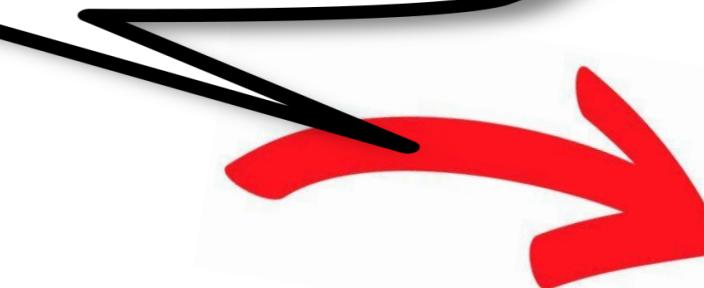


Illocutionary Forces: askOne



```
kate Jason >

1 !hi.
2
3 +!hi <-
4     .print("What's your phone number?");
5     .send(bob, askOne, phone(Number)).
```



Illocutionary Forces: askOne



Illocutionary Forces: askOne



Example: Chief and Glutton

<https://github.com/chon-group/distributedAndEmbeddedAI/blob/main/workshops/sepx/2024/projects/chiefGlutton.chon>



Image credits:

https://smurfs.fandom.com/wiki/Chef_Smurf

<https://smurfs.fandom.com/pt/wiki/Fominha>

TRANSACTIONS

<https://github.com/chon-group/Velluscinum/wiki>



Using digital assets in the relationships between cognitive agents:

the transfer of funds;

registration of ownership of artifacts;

declaration of promises or agreements.

Allowing the agents to manipulate assets and wallets **directly** in the DLT.

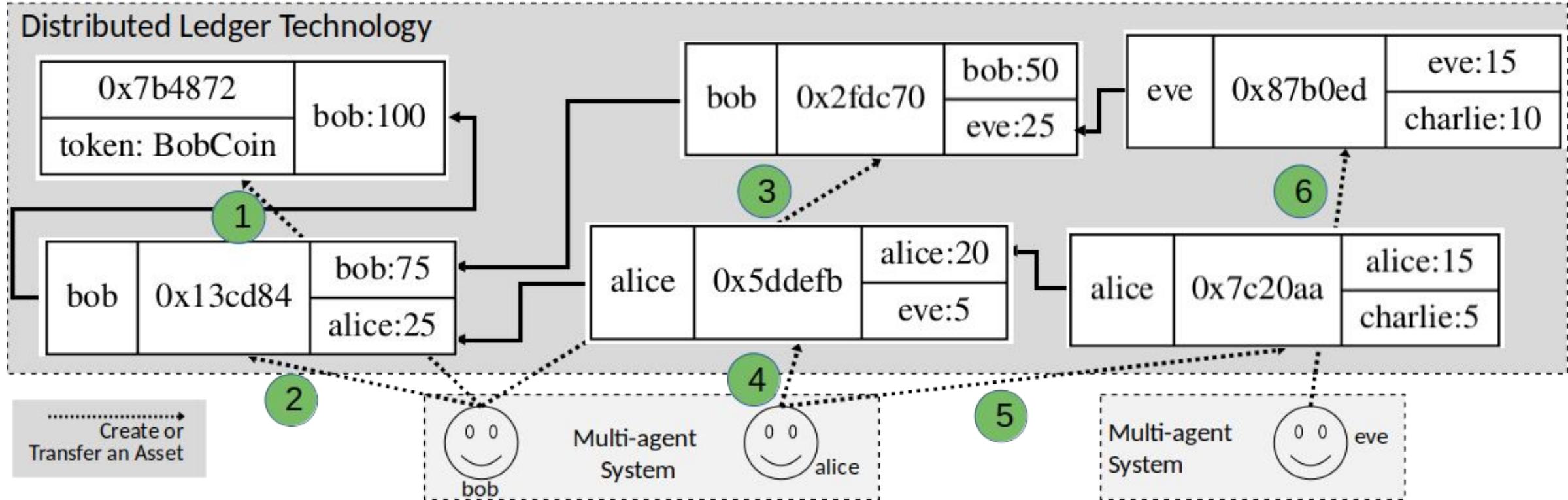
Distributed ledger technology



Velluscinum



Divisible asset such as cryptocurrency



buildWallet-CLI

Creates two files with an ECDSA keyset in Base58.

DESCRIPTION

```
velluscinum buildWallet <filename>
```



Parameters	Description
<filename>	The name of the files that will storage the public and private ECDSA key.

EXAMPLE

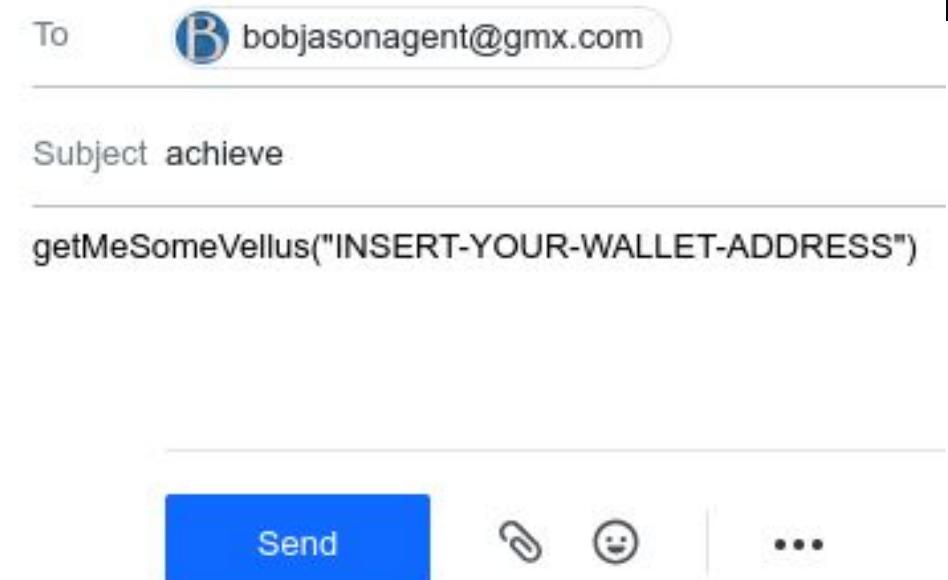
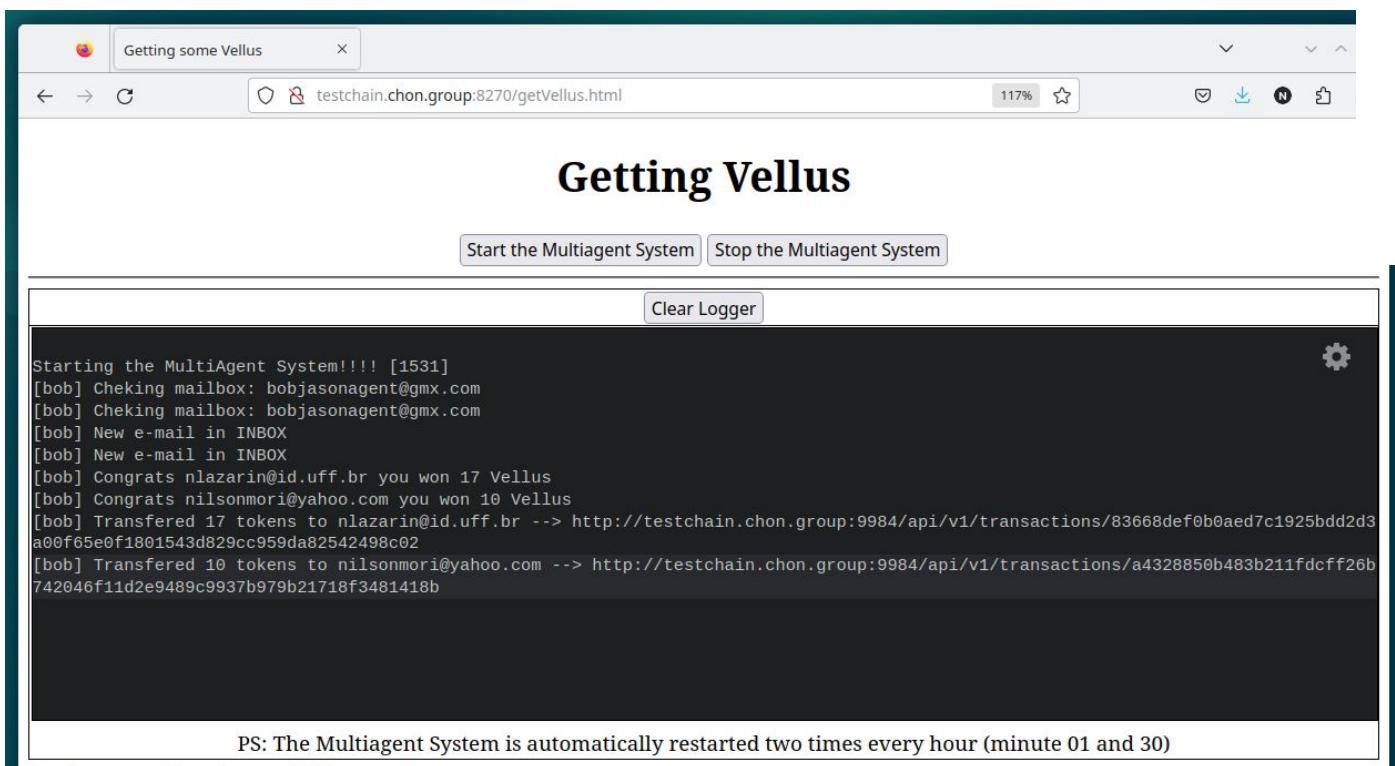
```
velluscinum buildWallet bob
```



```
nilson@dell:~$ velluscinum buildWallet bob
[Velluscinum] Build Wallet... 5nbmmHUSjqsAL6D6jbpKpXQW9UQTHg4YpD84hweNEmLH
nilson@dell:~$ ls
bob.privateKey  bob.publicKey
nilson@dell:~$
```

Getting some Vellus

<http://testchain.chon.group:8270/getVellus.html>



<https://github.com/chon-group/distributedAndEmbeddedAI/blob/main/workshops/sepx/2024/projects/chiefGlutonVellus.chon>



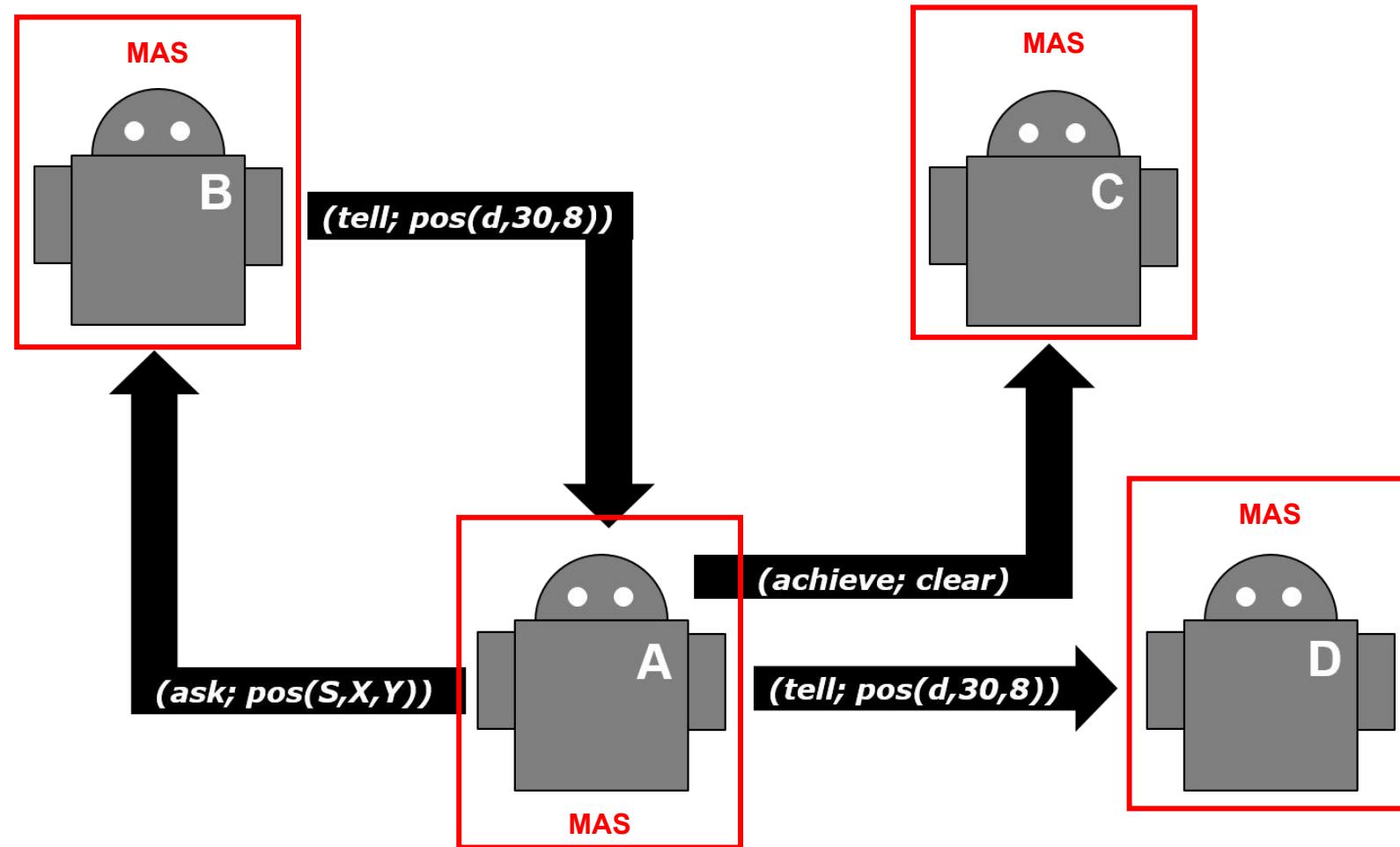
Image credits:

https://smurfs.fandom.com/wiki/Chef_Smurf

<https://smurfs.fandom.com/pt/wiki/Fominha>

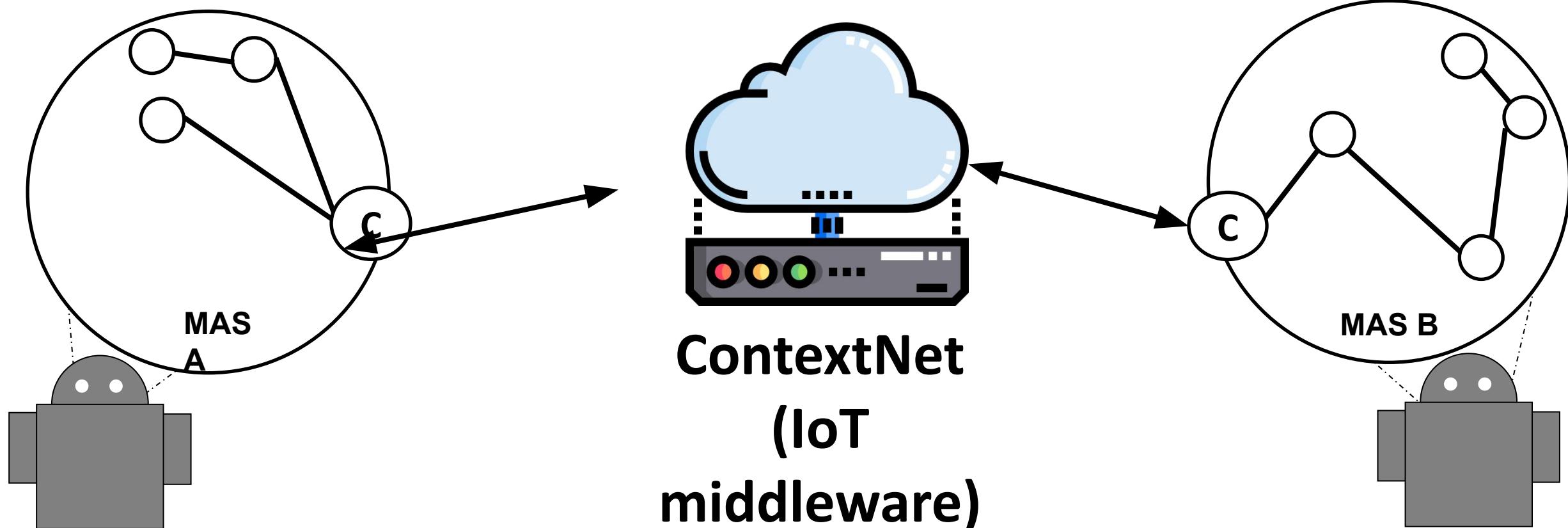
https://smurfs.fandom.com/pt-br/wiki/%C3%81vore_de_Dinheiro

Collaborative MultiAgent System



Hermes Agents: Server Communication

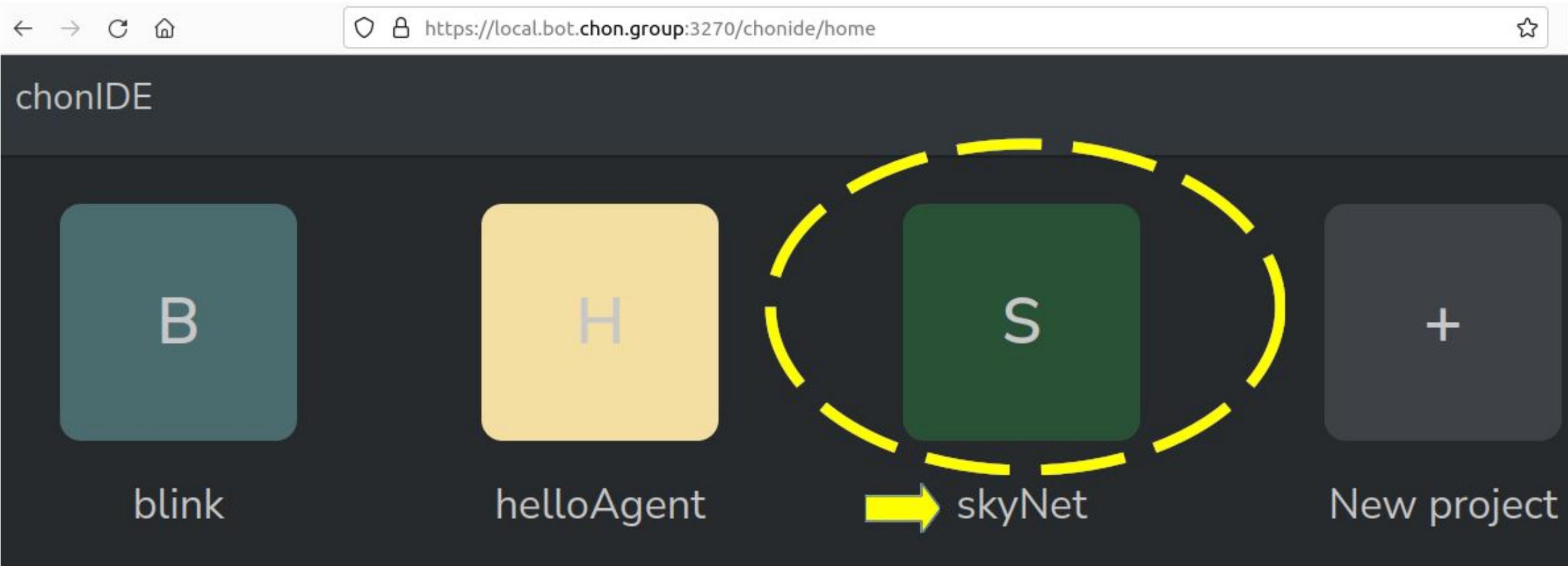
Public Server: skynet.chon.group - UDP Port: 5500



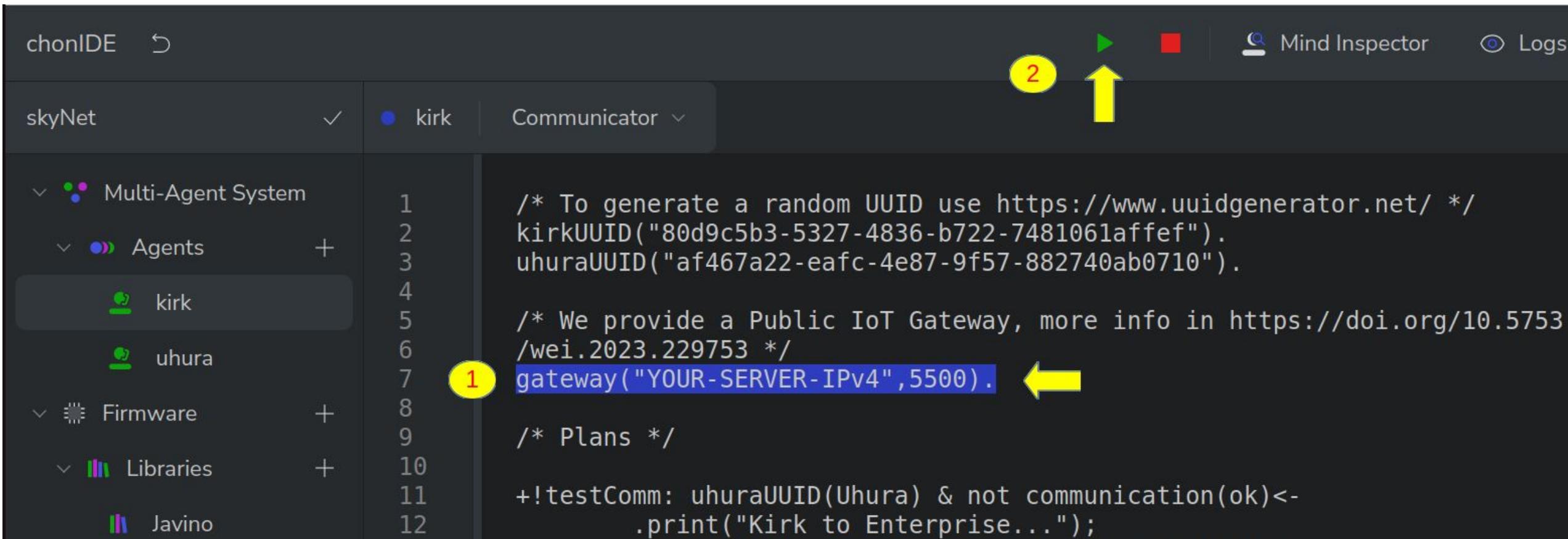
Hermes Agents: Internal Actions

Hermes	<code>.hermes.configureContextNetConnection("X", G, E, U)</code>	Configures an ContextNet network. Where <i>X</i> is a string that represents an network name; <i>G</i> is a literal that represents the FQDN or the network address of an gateway; <i>E</i> is a number that represents the network port of an gateway; <i>U</i> is a literal that represents the identification of the device in the network.
	<code>.hermes.connect("X")</code>	Joins at "X" ContextNet network.
	<code>.hermes.sendOut(D, f, M)</code>	Dispatches a message to another MAS. Where <i>D</i> is a literal that represents the identification of the recipient MAS; <i>f</i> is a illocutionary force (<i>tell</i> <i>untell</i> <i>achieve</i> <i>unachieve</i>); <i>M</i> is a literal that represents the message.
	<code>.hermes.moveOut(D, b, A)</code>	Carries over the agents to other MAS. Where <i>D</i> is a literal that represents the identification of the recipient MAS; <i>b</i> is a bio-inspired protocol (<i>inquilinism</i> <i>mutualism</i> <i>predation</i>); <i>A</i> is one, all, or a set of agents (i.e., all, agent or [agent ₁ , agent ₂ , agent _n])
	<code>.hermes.disconnect("X");</code>	Disconnects at "X" ContextNet network.

Hello SkyNet



Hello SkyNet



```
/* To generate a random UUID use https://www.uuidgenerator.net/ */
kirkUUID("80d9c5b3-5327-4836-b722-7481061affef").
uhuraUUID("af467a22-eafc-4e87-9f57-882740ab0710").

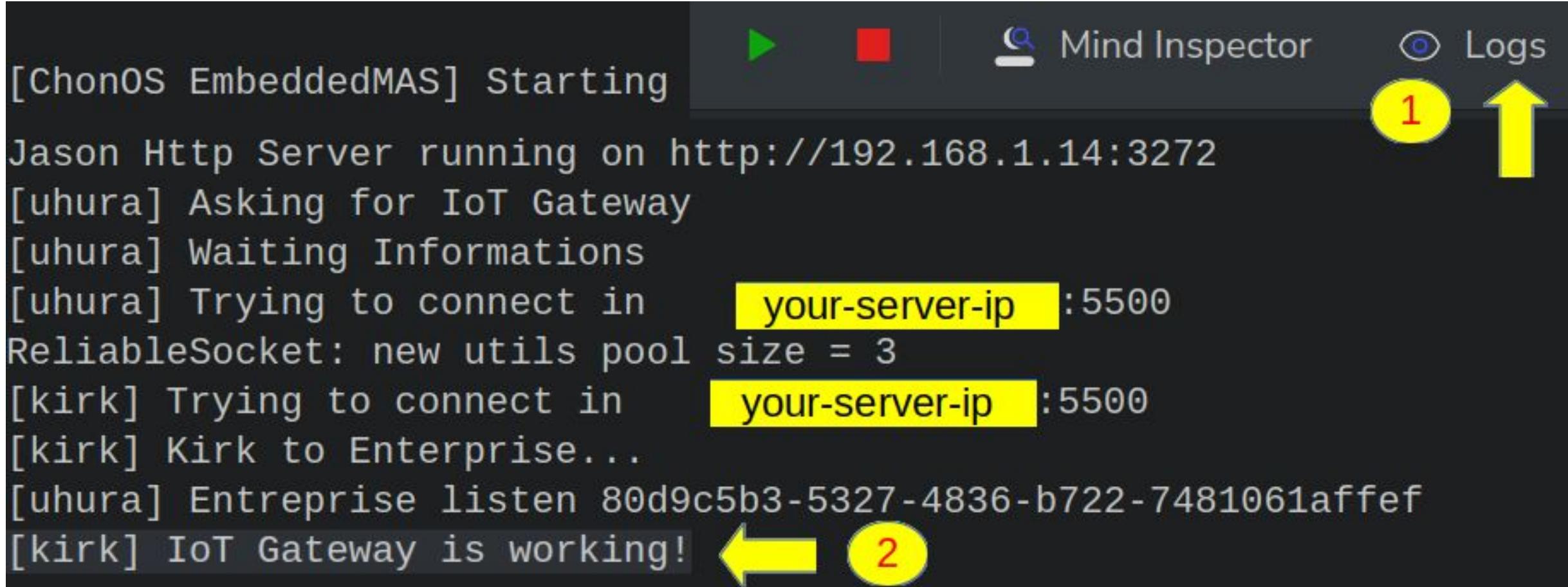
/* We provide a Public IoT Gateway, more info in https://doi.org/10.5753
/wei.2023.229753 */
gateway("YOUR-SERVER-IPv4", 5500).

/* Plans */

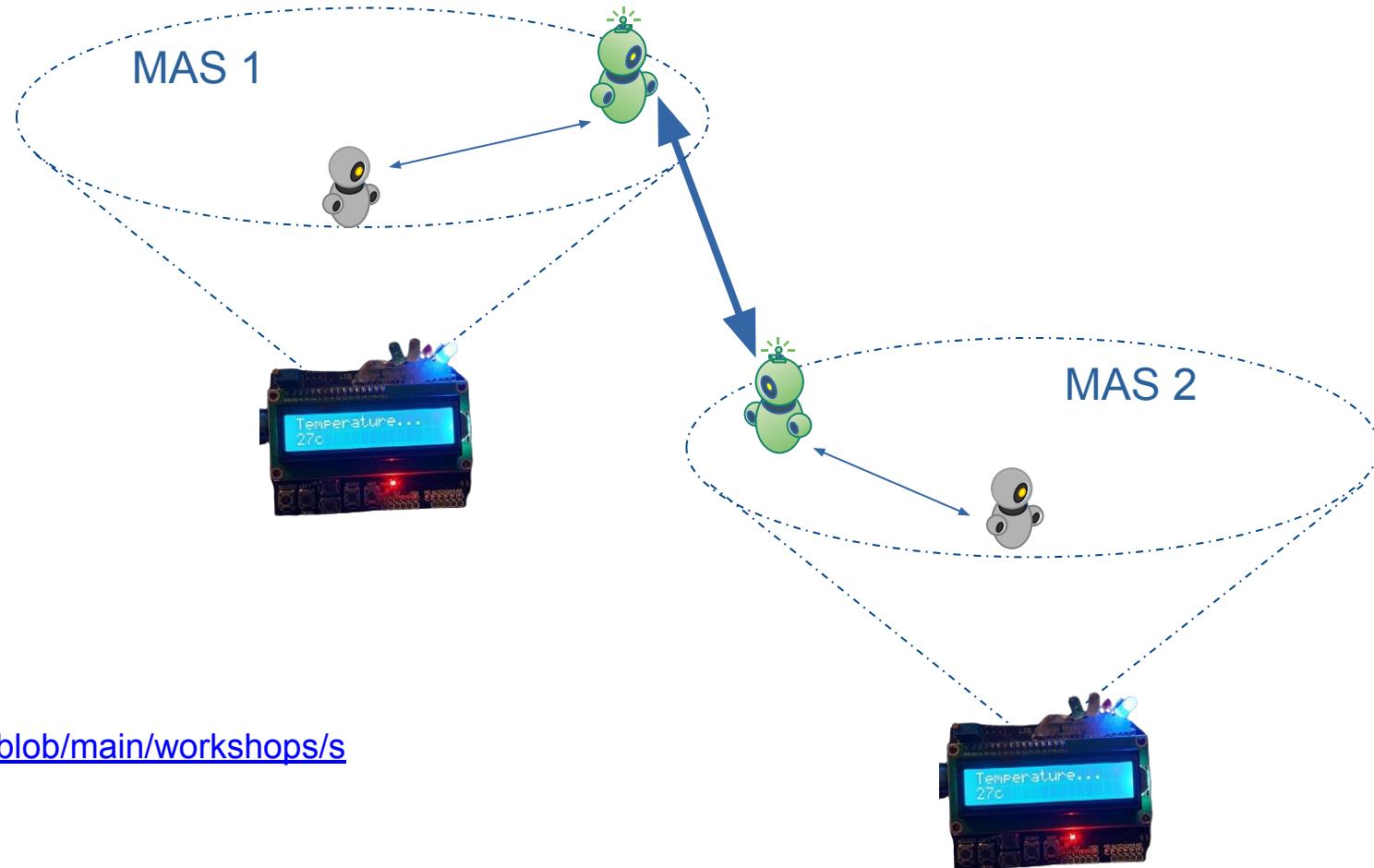
+!testComm: uhuraUUID(Uhura) & not communication(ok)<-
.print("Kirk to Enterprise...");
```

Hello SkyNet

```
[ChonOS EmbeddedMAS] Starting
[uhura] Asking for IoT Gateway
[uhura] Waiting Informations
[uhura] Trying to connect in      your-server-ip :5500
ReliableSocket: new utils pool size = 3
[kirk] Trying to connect in      your-server-ip :5500
[kirk] Kirk to Enterprise...
[uhura] Enterprise listen 80d9c5b3-5327-4836-b722-7481061affef
[kirk] IoT Gateway is working!
```



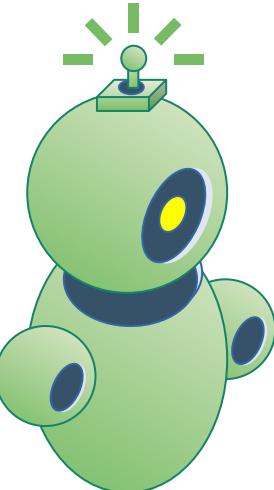
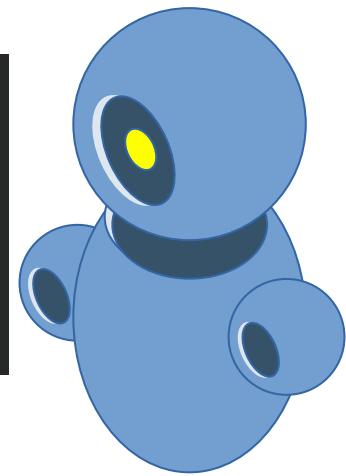
Example: MAS Communication



<https://github.com/chon-group/distributedAndEmbeddedAI/blob/main/workshops/sepex/2024/projects/communicatorAGENT.chon>

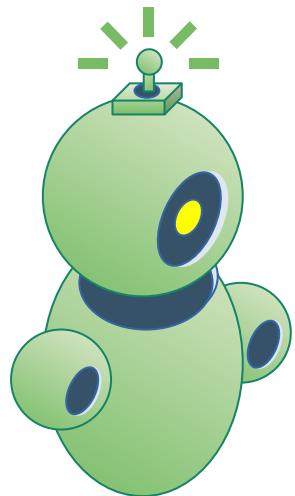
Example: MAS 1

```
+rainLast24hrs(R): R > 50 <-
    .print("New perception-> rainLevel: ",R," mm");
    .concat("Attention rain level is ",R," mm", RainLevelMessage);
    .send(agentHERMES,achieve,forward(RainLevelMessage)).
```



```
+ ! forward(Message) <-
    ?neighborUUID(N);
    .print("Forwarding a message to ", N);
    .sendOut(N, achieve, showMessage(Message)).
```

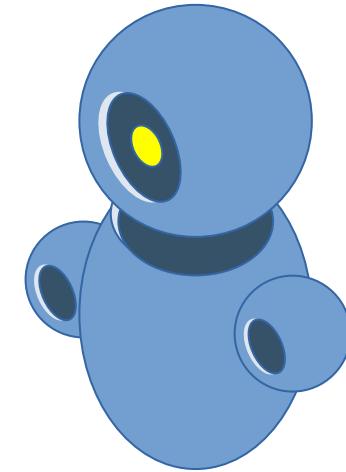
Example: MAS 2



```
+!showMessage(Message)[source(Device)] <-
  .print("I received the following message: ", Message);
  .send(agentARGO, achieve, infoLCD(Message)).
```



```
+!infoLCD(Message) <-
  .random(R); .wait(2000*R);
  .argo.act(Message).
```



Simple Exercise

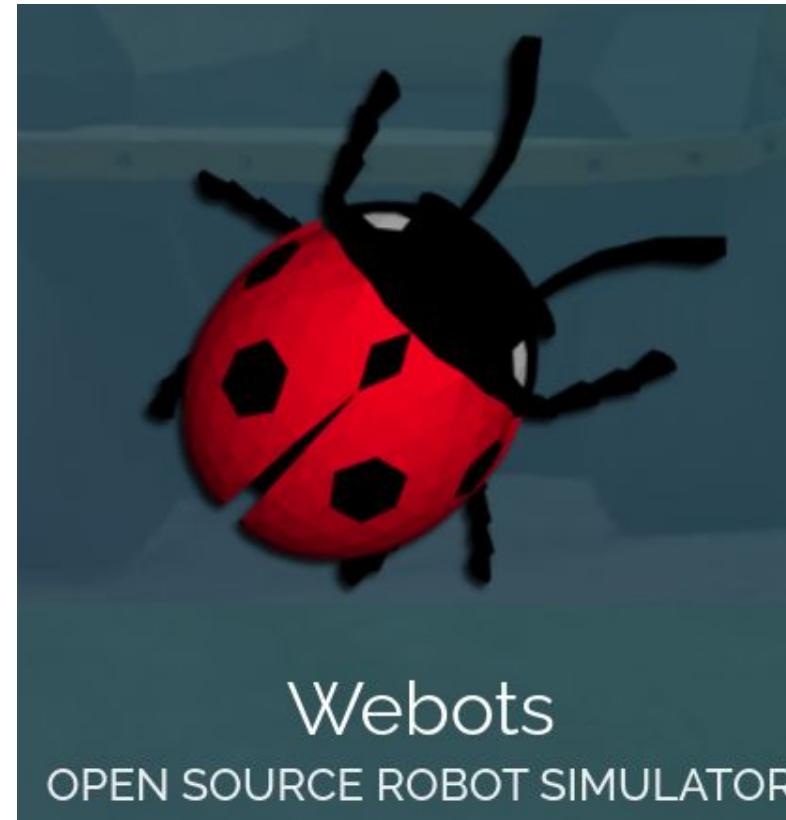
Adjust to:

- turn on the **green LED** in **MAS 2** if the rain intensity less than 50mm in **MAS 1**;
- turn on the **yellow LED** in **MAS 2** if the rain intensity exceeds 50mm in **MAS 1**;
- turn on the **red LED** in **MAS 2** if the the rain intensity exceeds 80mm in **MAS 1**;

OTHER TOOLS

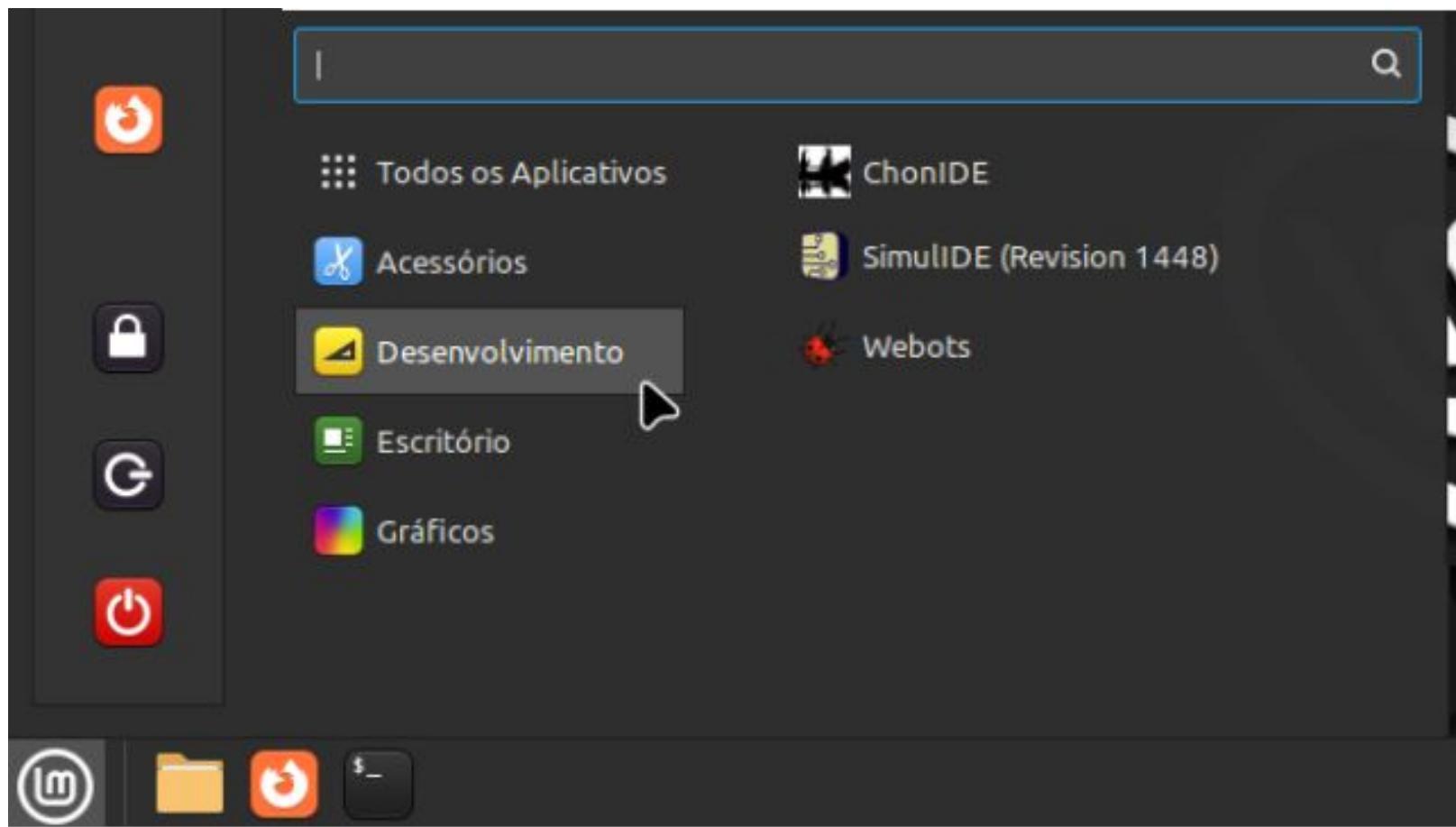


Webots



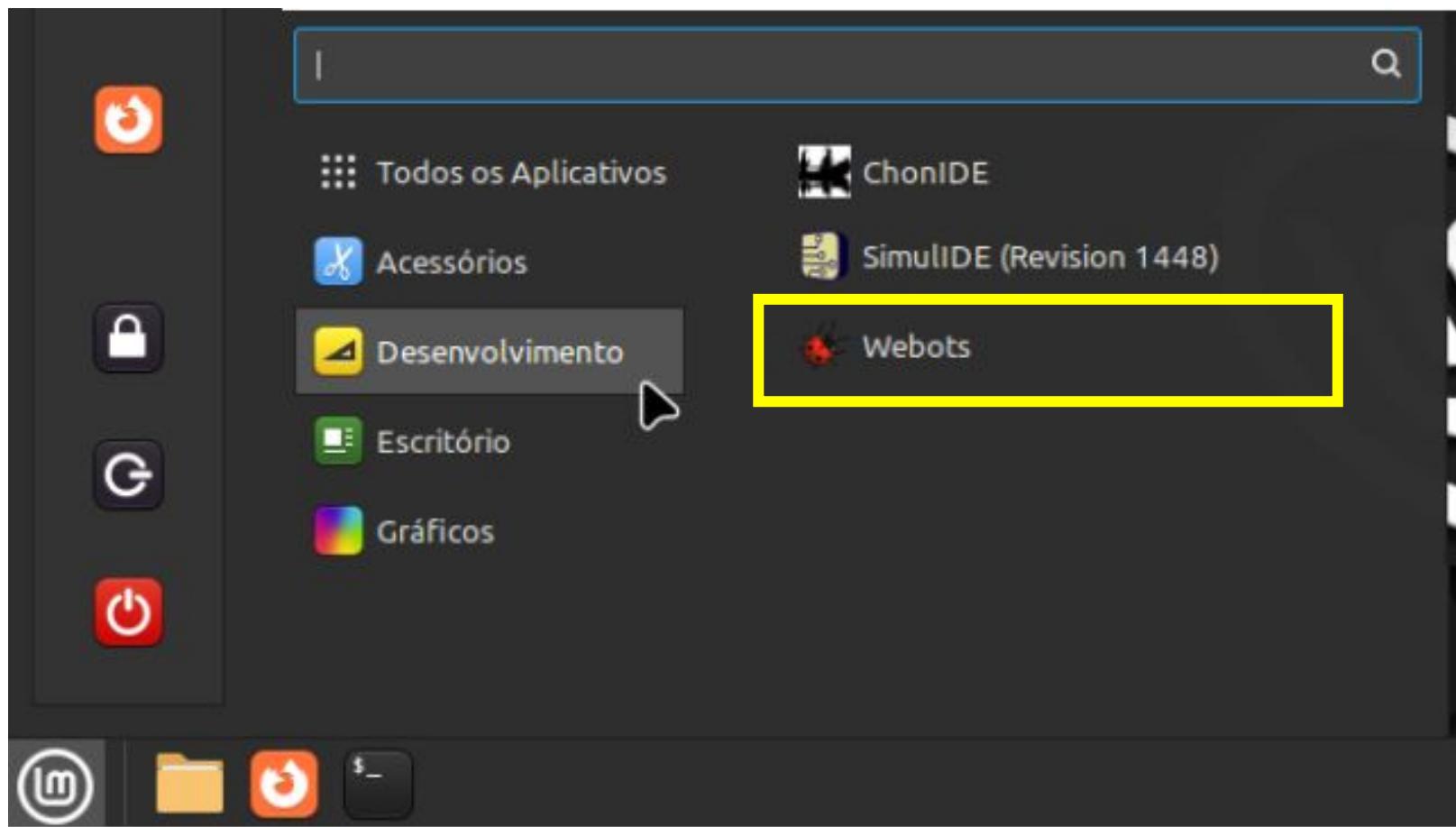
Michel, O. (1998). Webots: Symbiosis Between Virtual and Real Mobile Robots. In: Heudin, JC. (eds) Virtual Worlds. VW 1998. Lecture Notes in Computer Science(), vol 1434. Springer, Berlin, Heidelberg.
https://doi.org/10.1007/3-540-68686-X_24

Webots



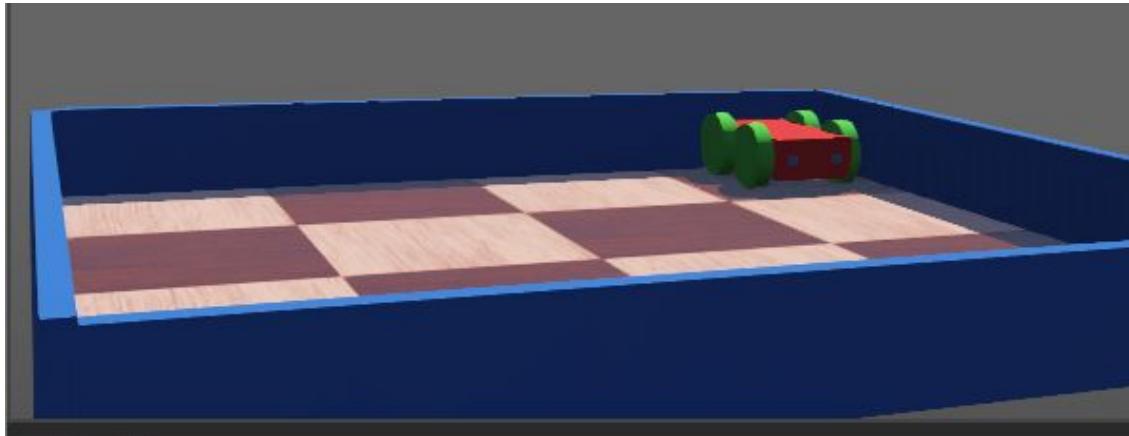
Manual de instalação
<https://cyberbotics.com/doc/guide/installation-procedure#installing-the-debian-package-with-the-advanced-packaging-tool-apt>

Webots



Manual de instalação
<https://cyberbotics.com/doc/guide/installation-procedure#installing-the-debian-package-with-the-advanced-packaging-tool-apt>

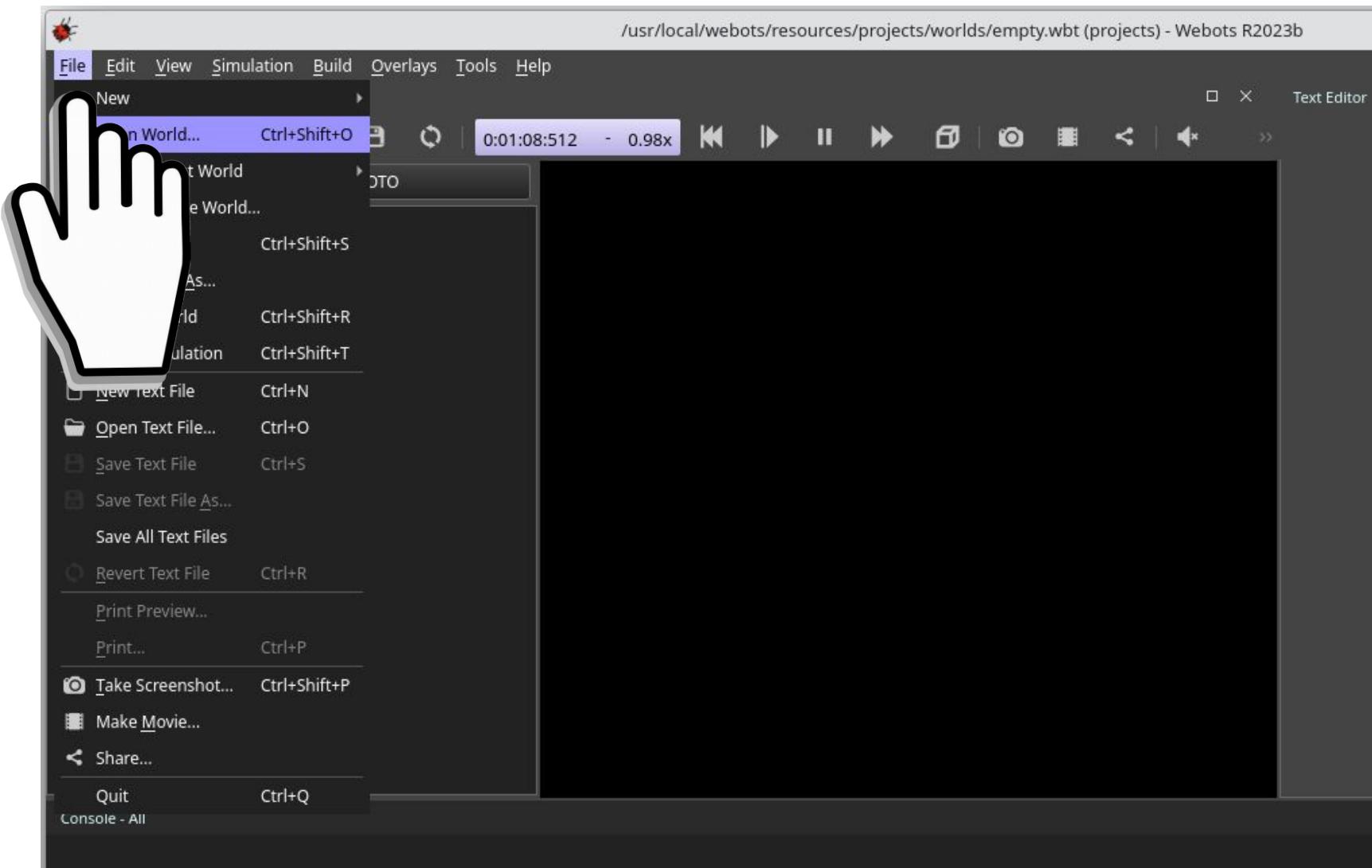
Webots



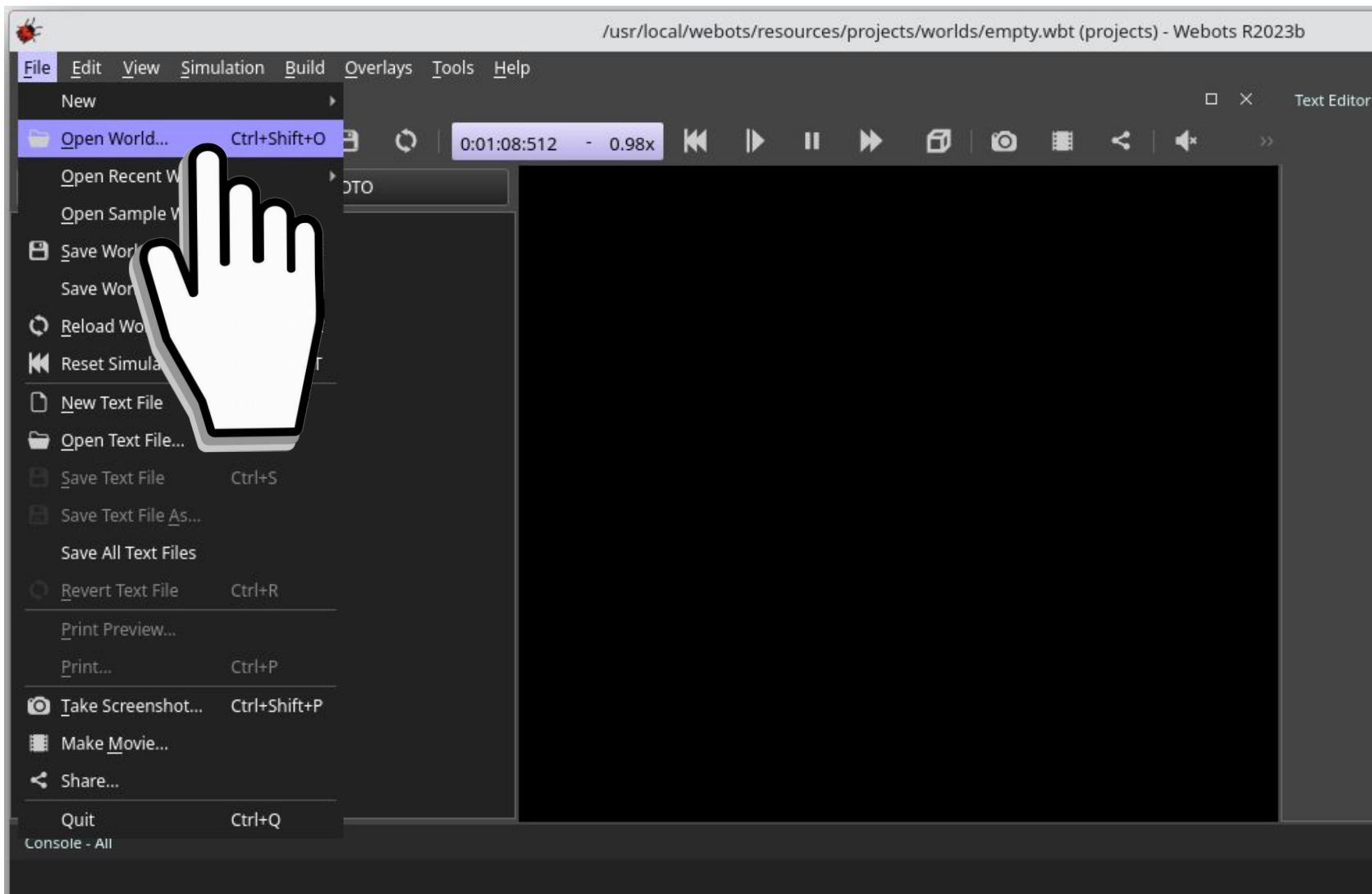
```
SEPEX2024_SMAsOnWebots/
├── controllers
│   └── four_wheels_collision_avoidance
│       ├── four_wheels_collision_avoidance.c
│       └── JavinoCLibrary
│           ├── include
│           │   └── javino.h
│           └── src
│               └── javino.c
└── Makefile
├── LICENSE
├── README.md
└── WebotsExample.chon
└── worlds
    ├── world01.wbt
    ├── world02.wbt
    ├── world03.wbt
    ├── world04.wbt
    ├── world05.wbt
    └── world06.wbt
```

```
git clone https://github.com/bptfreitas/SEPEX2024_SMAsOnWebots
```

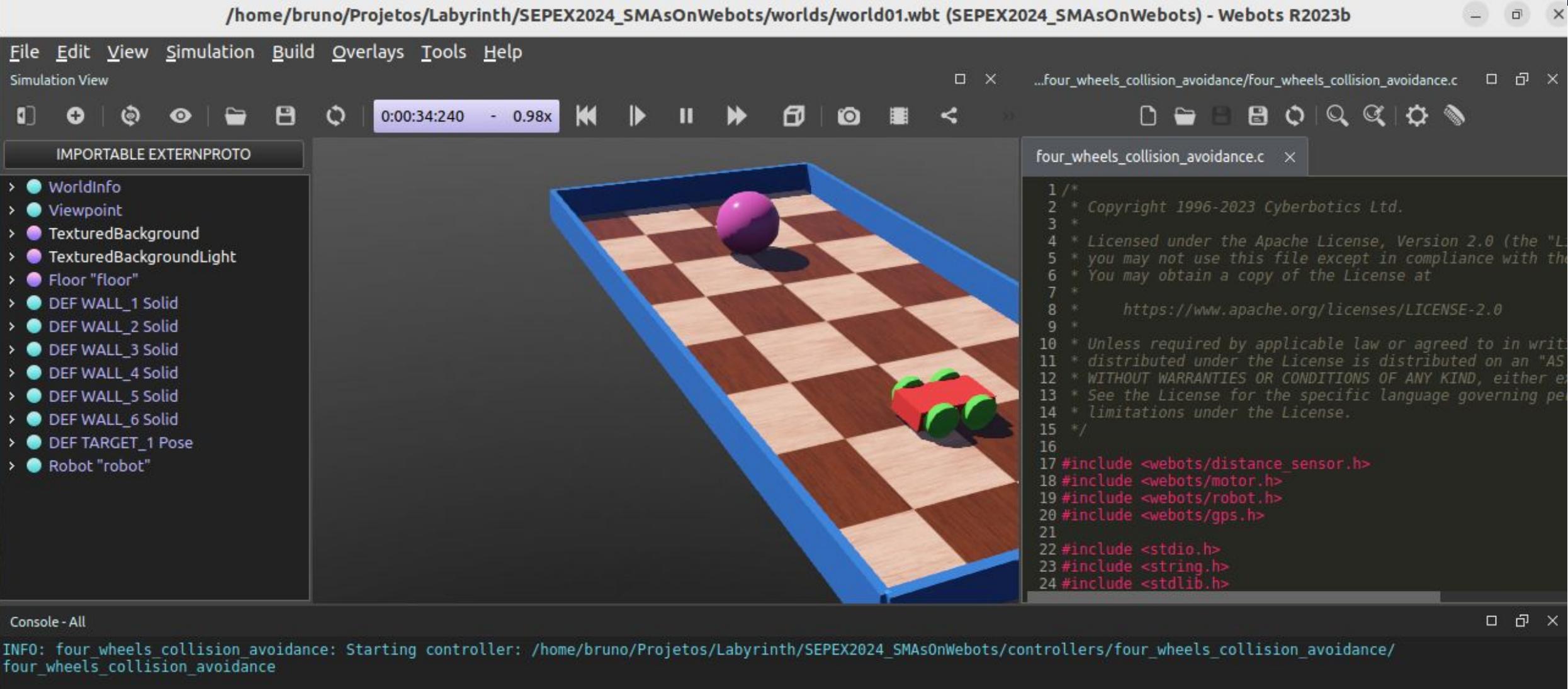
Webots



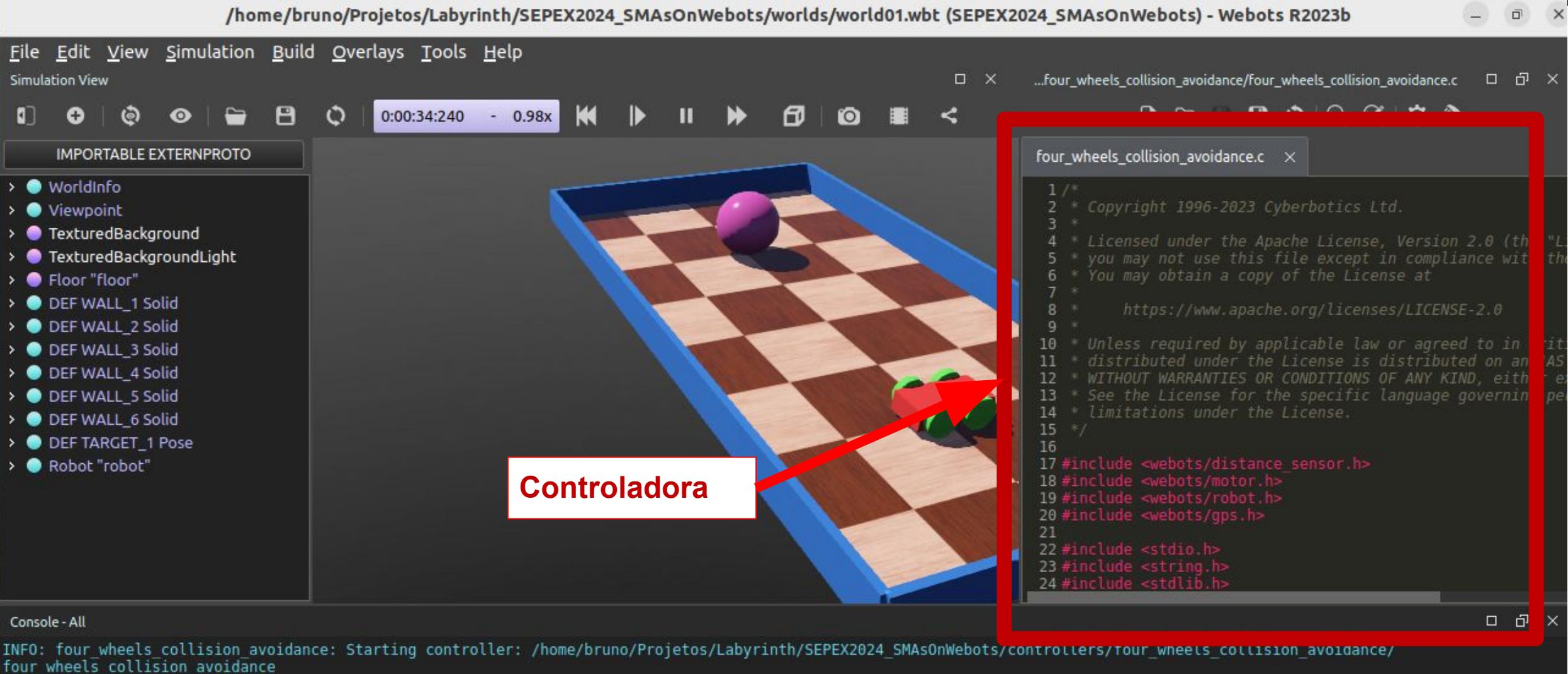
Webots



Webots



Webots



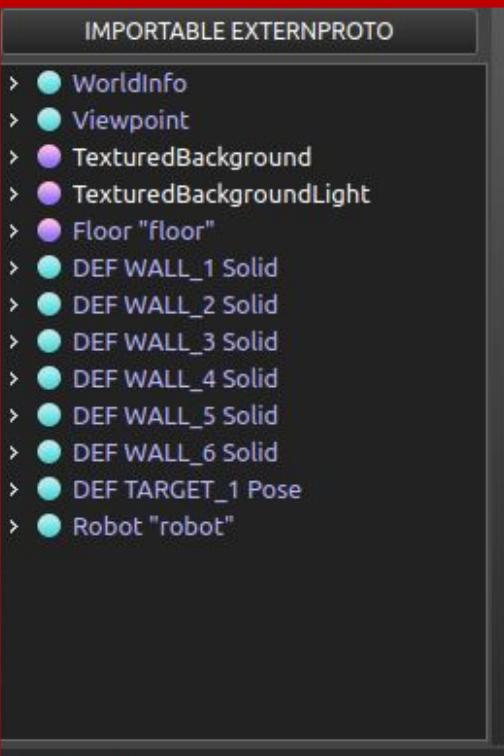
Webots

/home/bruno/Projetos/Labyrinth/SEPEX2024_SMAsOnWebots/worlds/world01.wbt (SEPEX2024_SMAsOnWebots) - Webots R2023b

File Edit View Simulation Build Overlays Tools Help

Simulation View

...four_wheels_collision_avoidance/four_wheels_collision_avoidance.c



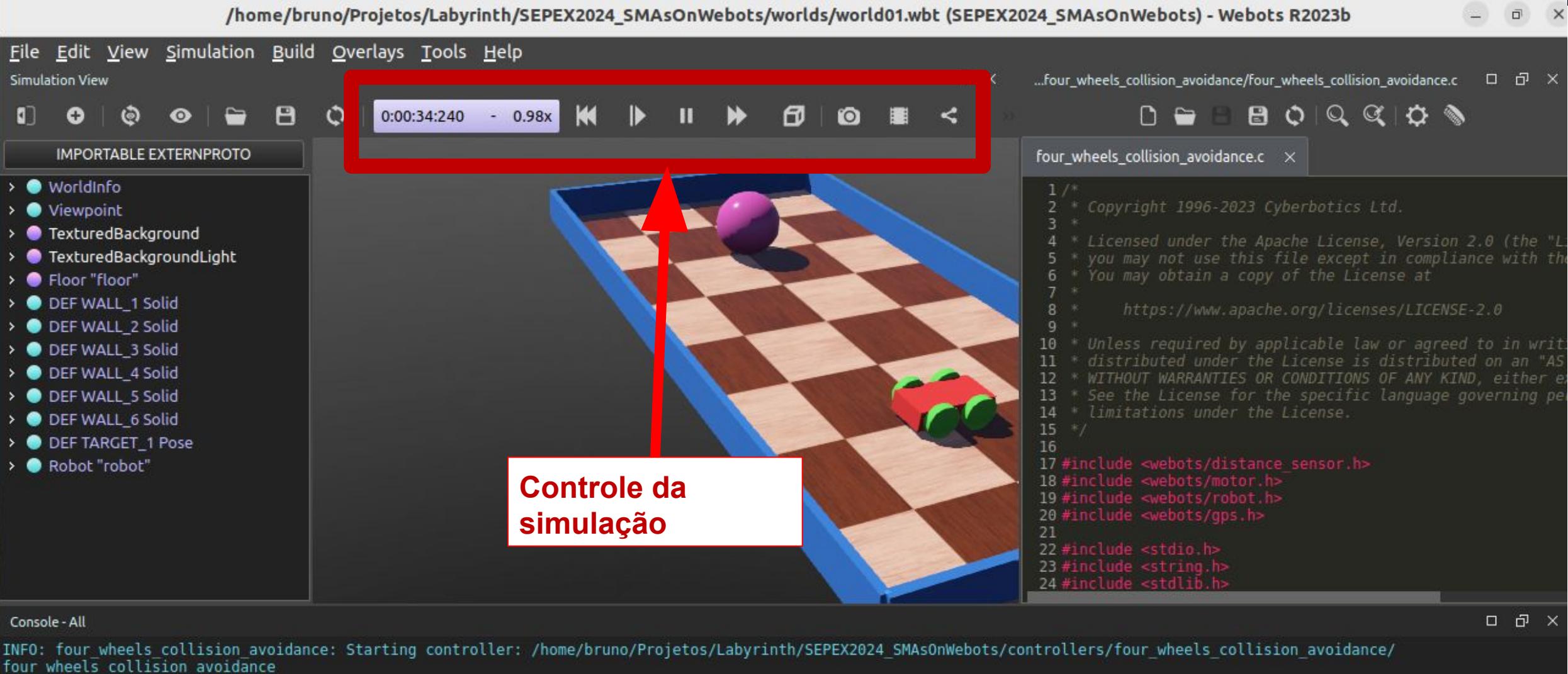
Objetos da cena

four_wheels_collision_avoidance.c

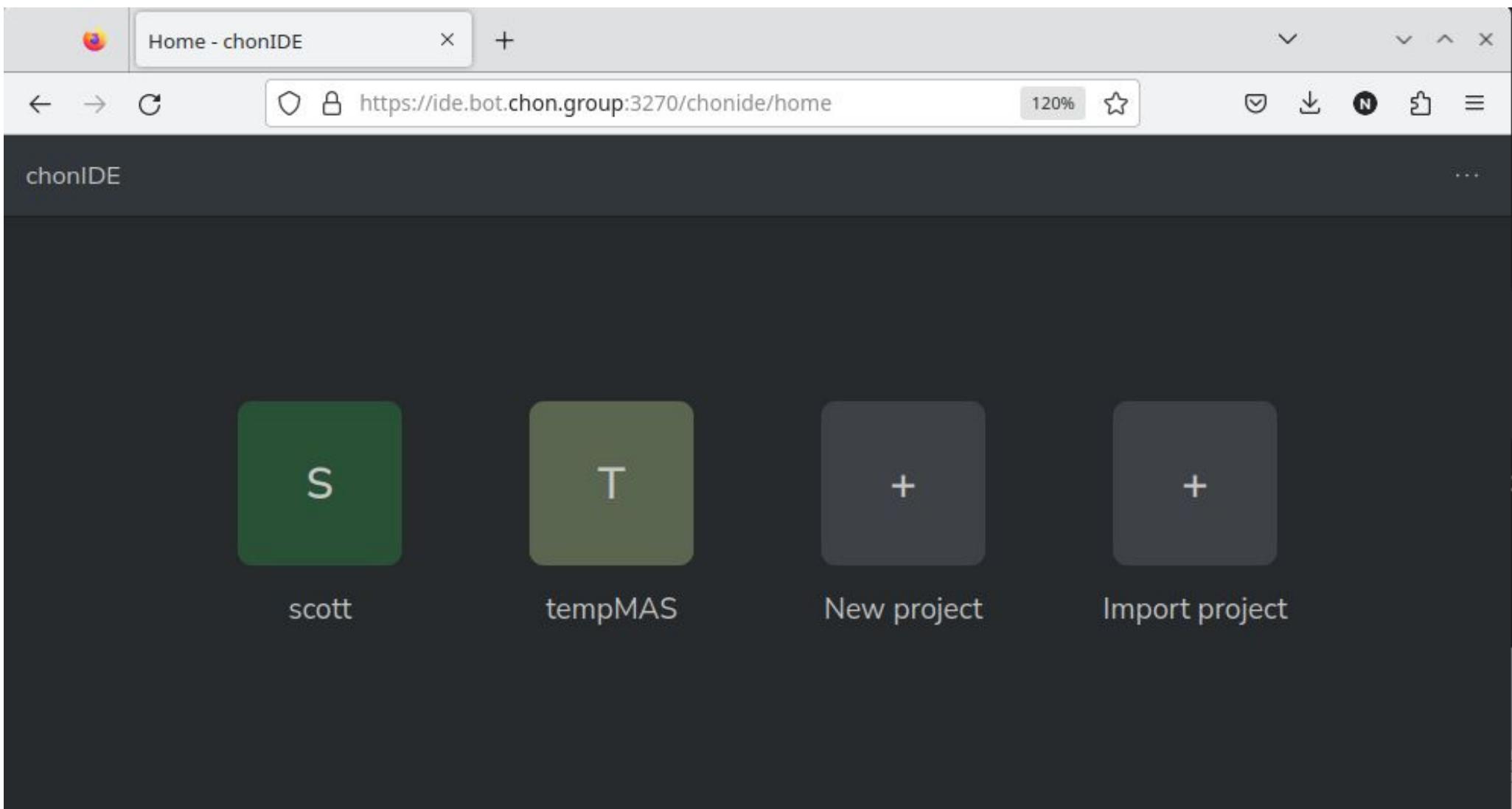
```
1 /*
2 * Copyright 1996-2023 Cyberbotics Ltd.
3 *
4 * Licensed under the Apache License, Version 2.0 (the "License");
5 * you may not use this file except in compliance with the License.
6 * You may obtain a copy of the License at
7 *
8 *      https://www.apache.org/licenses/LICENSE-2.0
9 *
10 * Unless required by applicable law or agreed to in writing,
11 * distributed under the License is distributed on an "AS
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
13 * or implied. See the License for the specific language governing
14 * permissions and limitations under the License.
15 */
16
17 #include <webots/distance_sensor.h>
18 #include <webots/motor.h>
19 #include <webots/robot.h>
20 #include <webots/gps.h>
21
22 #include <stdio.h>
23 #include <string.h>
24 #include <stdlib.h>
```

INFO: four_wheels_collision_avoidance: Starting controller: /home/bruno/Projetos/Labyrinth/SEPEX2024_SMAsOnWebots/controllers/four_wheels_collision_avoidance/four_wheels_collision_avoidance

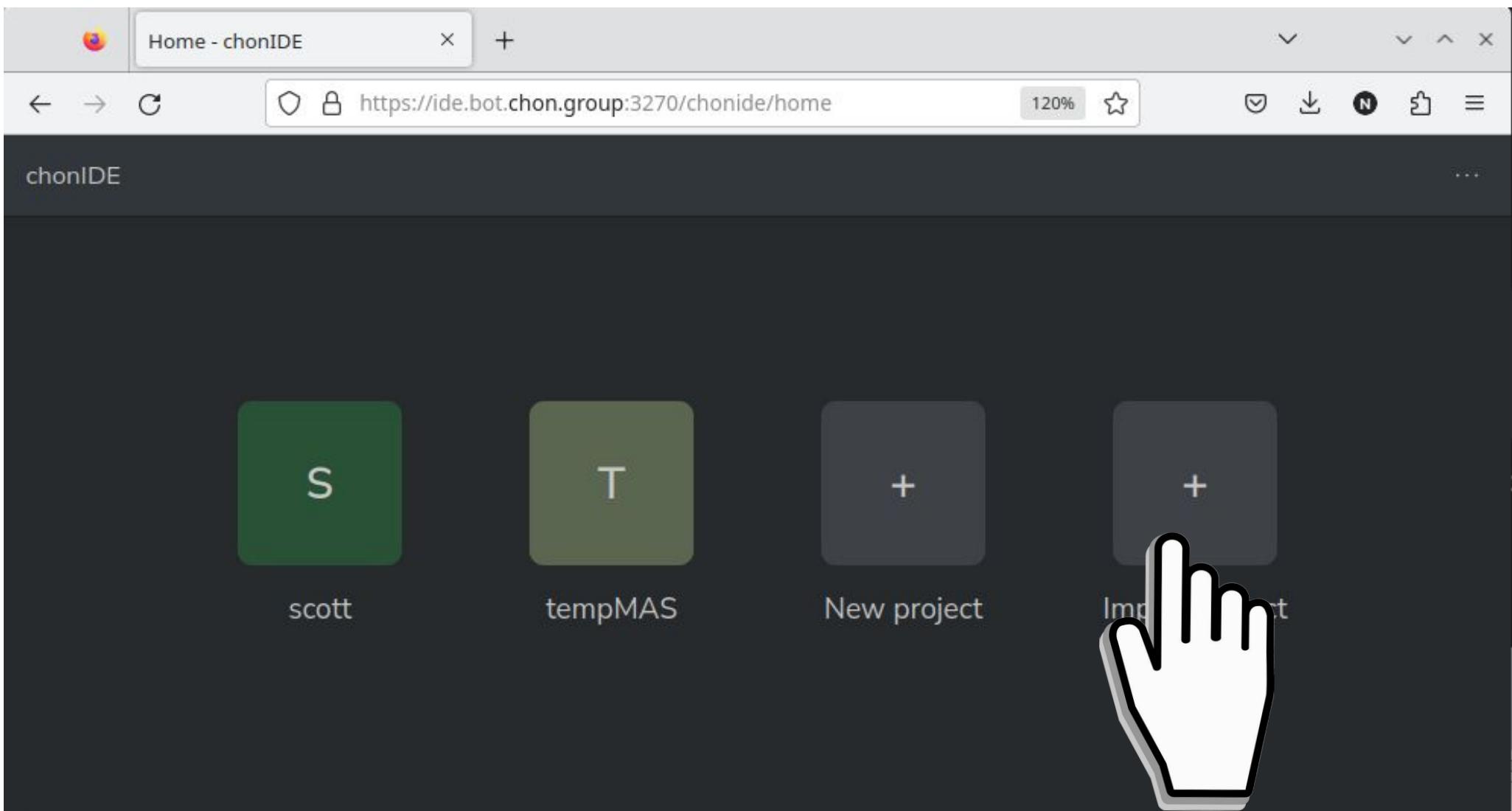
Webots



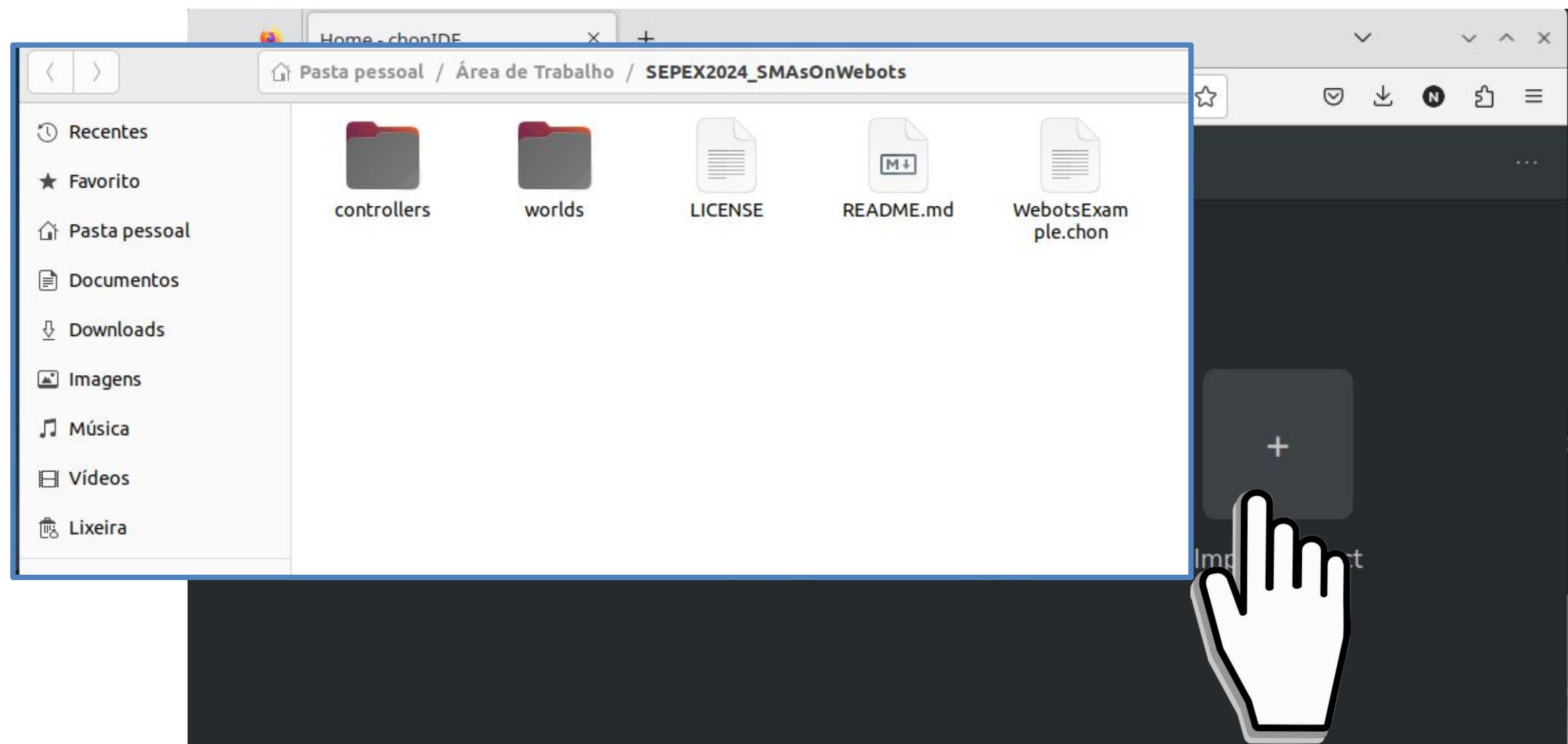
Webots



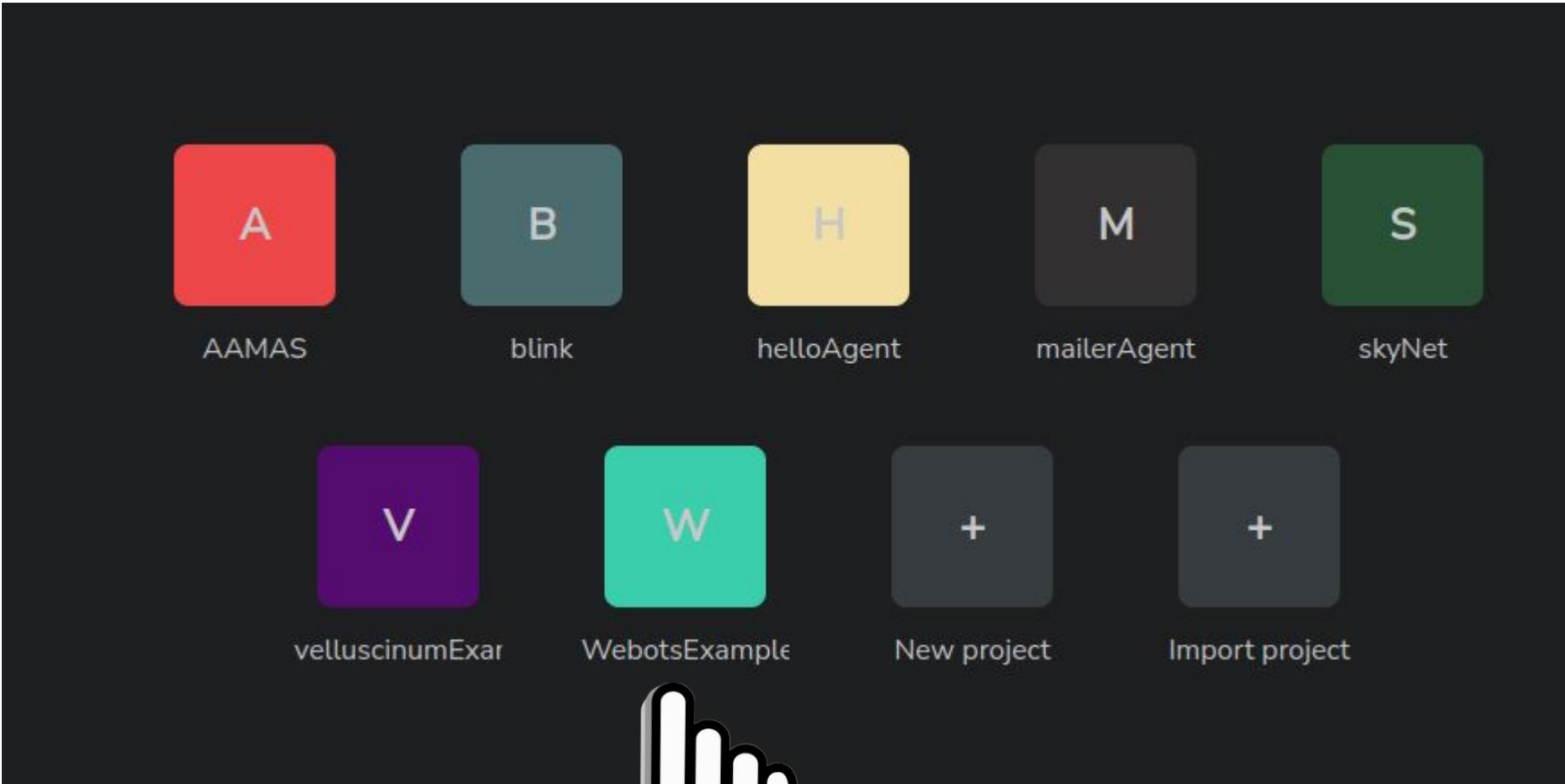
Webots



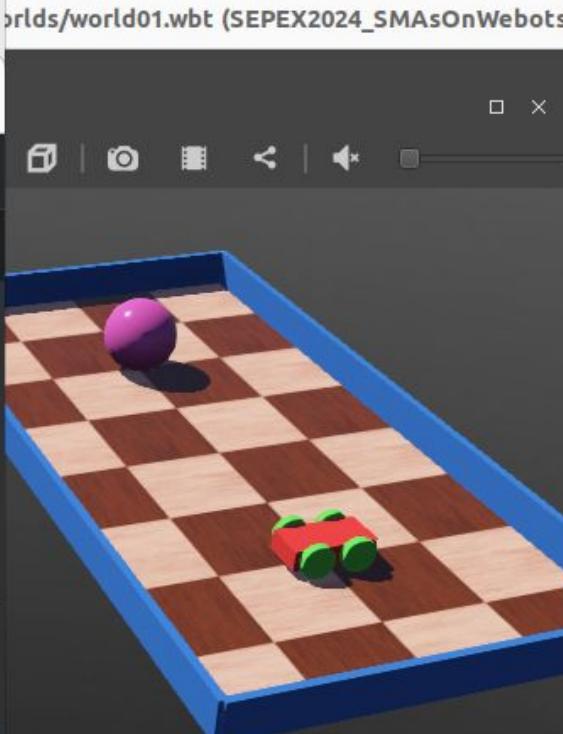
Webots



Webots



Webots

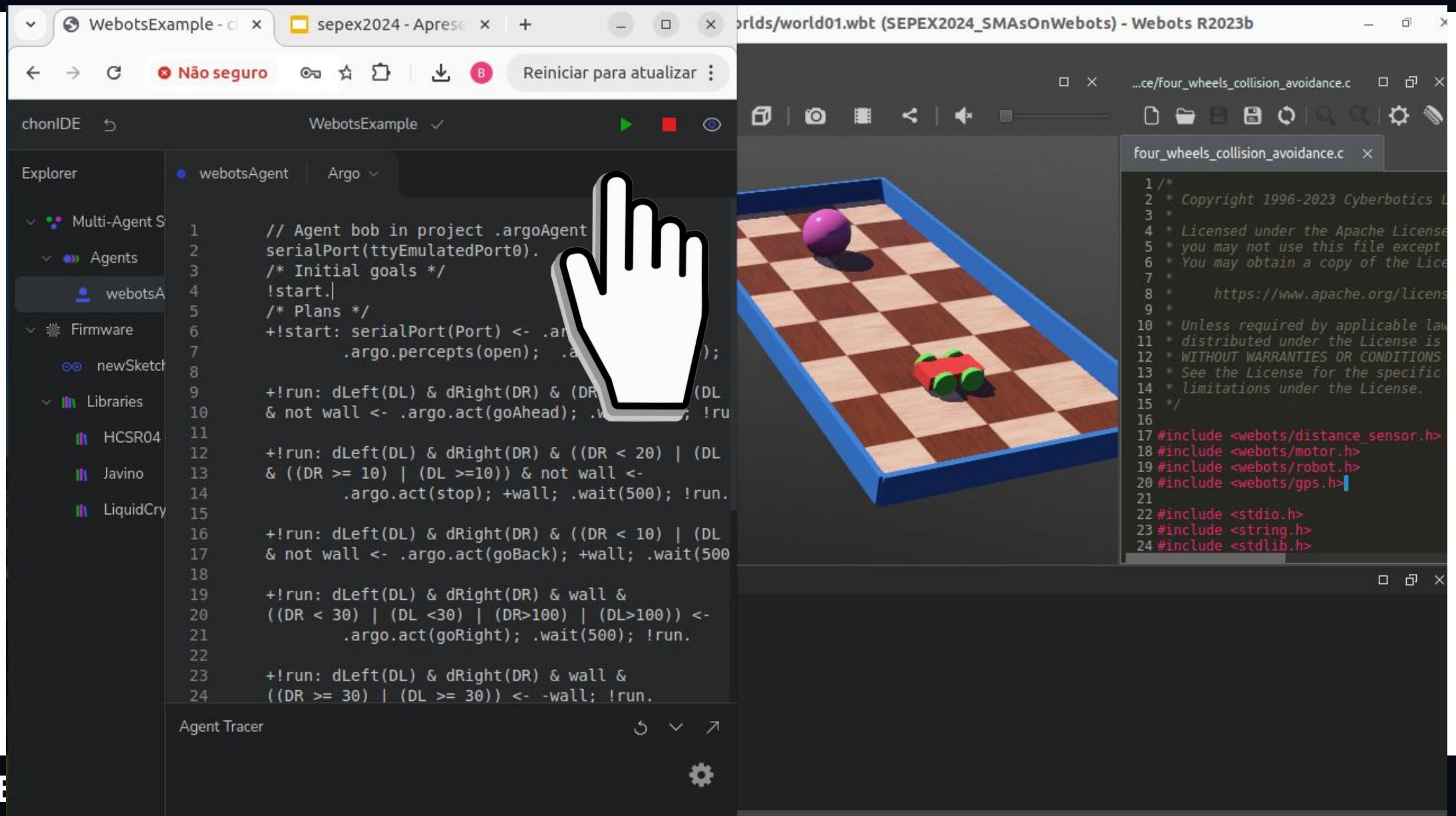


The screenshot shows the Webots IDE interface. The top bar displays tabs for "WebotsExample - c" (active), "sepex2024 - Apresentação", and "worlds/world01.wbt (SEPEX2024_SMAsonWebots) - Webots R2023b". Below the tabs, there are browser-like controls: back, forward, search, and refresh, along with a message "Não seguro" (Not secure) and a button "Reiniciar para atualizar" (Restart to update).

The main workspace is divided into several panes:

- Explorer:** Shows the project structure with "Multi-Agent System" selected, containing "Agents" (with "webotsAgent" highlighted) and "Firmware" (with "newSketch" selected). Other items include "Libraries" (HCSR04, Javino, LiquidCry).
- Code Editor:** Displays the C code for "four_wheels_collision_avoidance.c". The code implements a navigation plan for an agent named "bob" using the ArgoAgent API.
- 3D View:** A 3D rendering of a robot with green wheels and a purple body on a checkered board.
- Code Editor (right):** Displays the full content of the "four_wheels_collision_avoidance.c" file, including the Apache License header and the C code.
- Agent Tracer:** A small window at the bottom left showing a timeline with a play button.

Webots



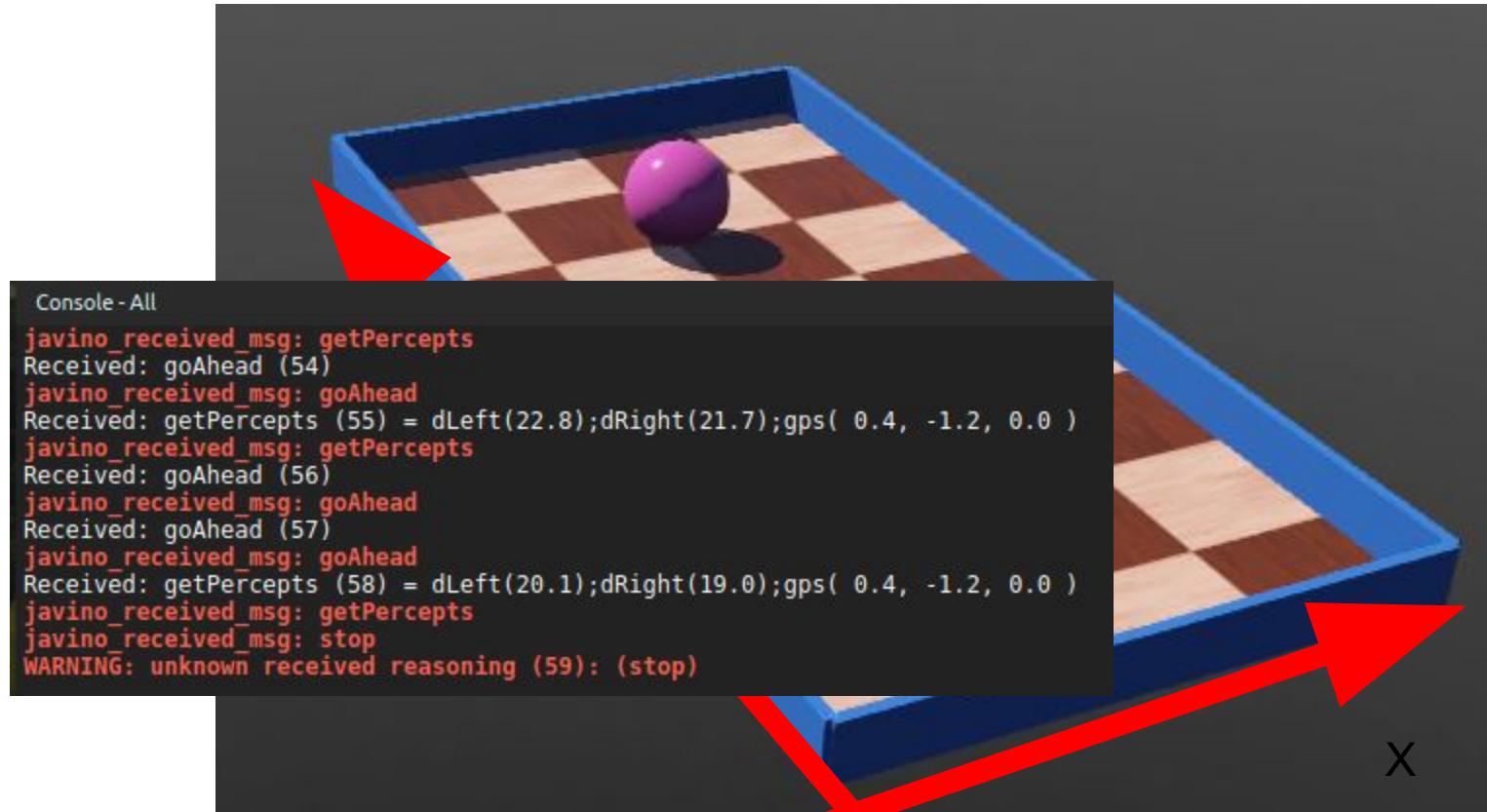
Webots

ACTIONS

- goBack
- goAhead
- goLeft
- goRight

PERCEPTIONS

- Left Distance sensor
 - dLeft (*distance*)
- Right Distance sensor
 - dRight (*distance*)
- GPS
 - gps(*x, y, z*)
 - *x, y* and *z* are distances from the Target



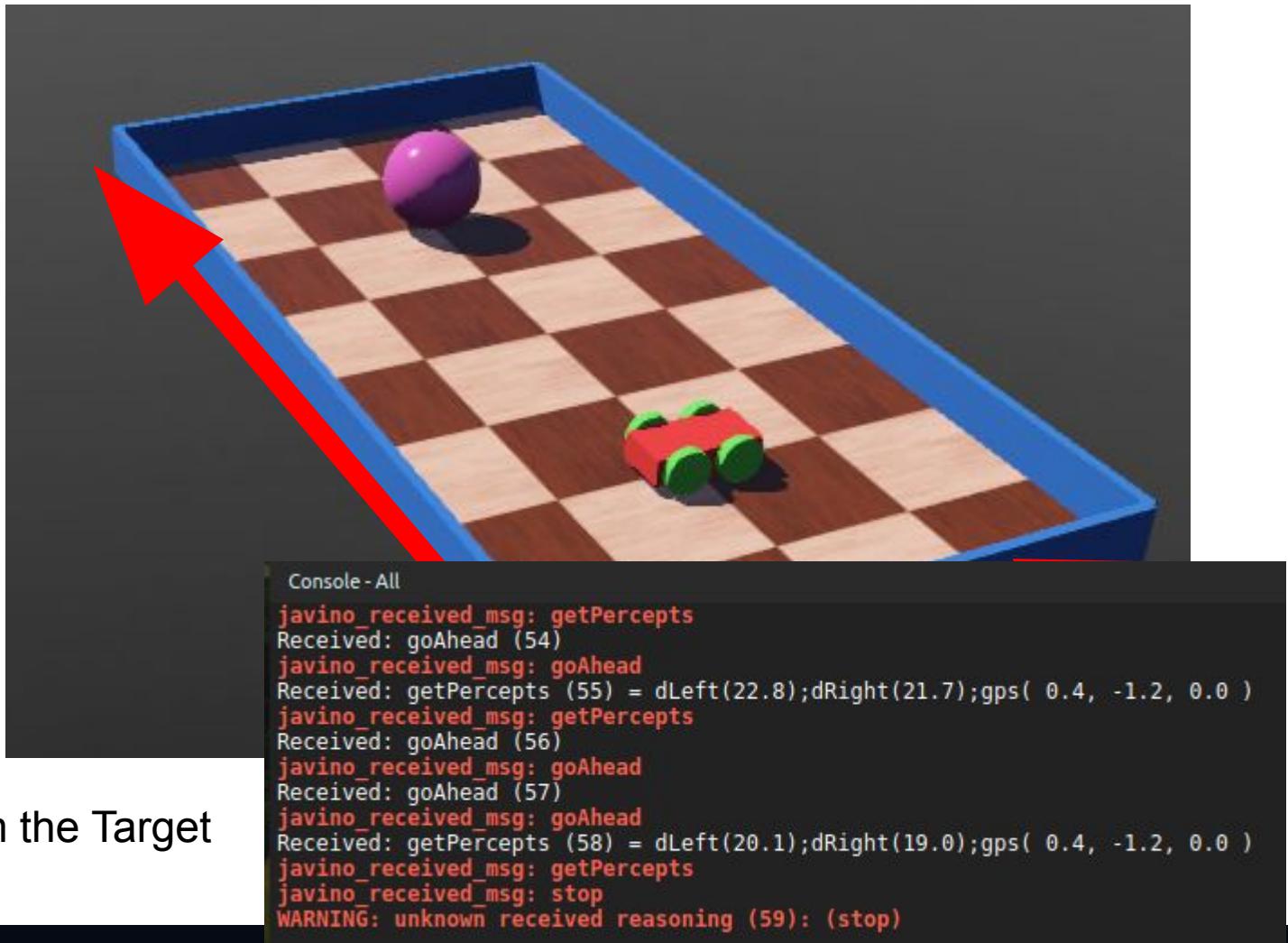
Webots

ACTIONS

- goBack
- goAhead
- goLeft
- goRight

PERCEPTIONS

- Left Distance sensor
 - dLeft (*distance*)
- Right Distance sensor
 - dRight (*distance*)
- GPS
 - gps(*x, y, z*)
 - *x, y* and *z* are distances from the Target



Acknowledgements

THANK YOU!

