{EPITECH}
LEARN DIFFERENT*

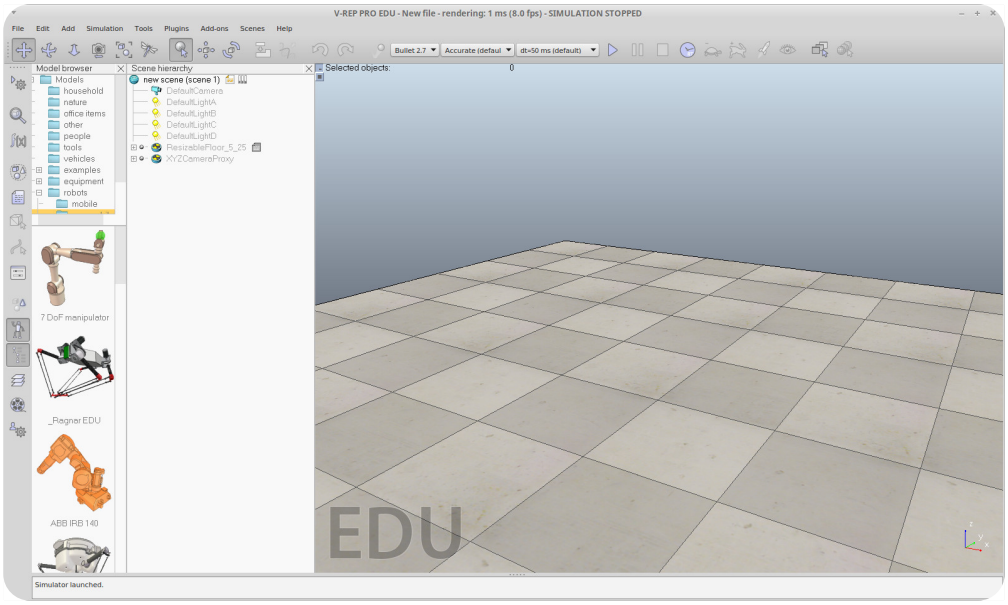
NEED4STEK

BOOTSTRAP

NEED4STEK

Throughout this Bootstrap, you're going to need to get familiar with the environment that the Need4stek project suggests. In order to continue, it's necessary to have carried out the three following tasks:

- ✓ read carefully the project description (several times, if needed, for certain sections),
- ✓ download CoppeliaSim Edu V4.1.0 and extract the archive.
- ✓ start up the simulation software by executing the `coppeliaSim.sh` script.



CoppeliaSim will then display an empty default scene you can tackle.

As stated in the project description: don't forget to put the physics engine on ODE!

Step 1 - CoppeliaSim scene

A scene is a file that contains the car that you are going to control and the track on which you will drive your racecar.
Two scenes are currently at yout disposal:

- ✓ `track_1.ttt`: this is the one you'll use to test your AI (project).
- ✓ `bootstrap_track.ttt`: this is a simplified scene (a straight, simple road) that you will use to complete this Bootstrap.

Once you're familiar with these new tools, you'll be able to create your own scenes. They could potentially even be added to the package.

Start by opening the `bootstrap_track.ttt` scene in CoppeliaSim, using one or another of these methods:

- ✓ Drag'n drop the file in CoppeliaSim from the file navigator
- ✓ File -> Open scene…

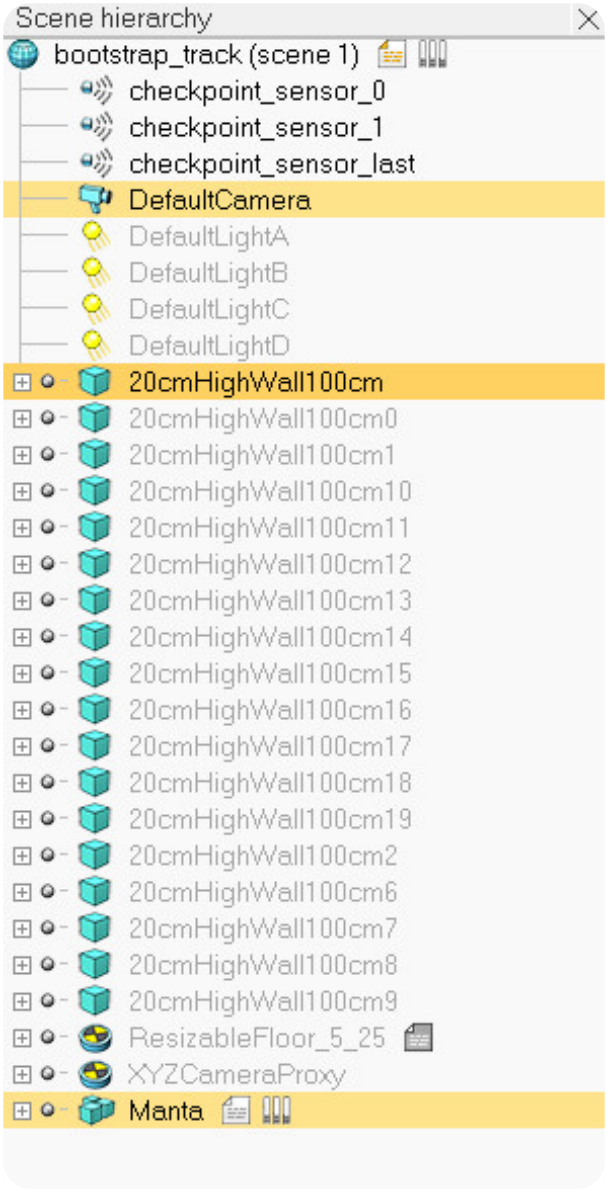
Once the scene is loaded, you'll notice the presence of different elements, the most important of which are described below:

Manta
A racecar model. This will be the racecar controlled by your actions. It contains sub-elements such as: the motor, onboard camera, …#br

20cmHighWall100cm
The walls bordering the track.#br

DefaultCamera
The camera that will directly observe the scene in CoppeliaSim on the righthand side of the screen (not to be confused with remoteApiCamera contained in the Manta.).#br

In this configuration, you could launch a simulation from CoppeliaSim, but nothing would happen gven that the only way to transmit commands to the car is to pass by the API used in the n4s binary that is provided.



If, during the simulation, you want to see the car's point of view in real time (1st person view), you need to right click in the `scene -> Add -> Floating View`.
A new window will appear and you will then need to tell it what to display. Roll out the Manta (from Scene hierarchy) and select the "remoteApicamera" camera. Then right click again on `Floating View -> View -> Associate view with vision sensor`.
Your car will then have first person view during the simulations.

Step 2 - n4s

Let's now focus on `n4s`, the binary that is going to enable you to communicate with CoppeliaSim so that you can control your car.
This binary is also located in the tarball that you have previously downloaded.

When started up, a prompt will be displayed to accept commands. It is strongly recommended to read the protocol section of the project description again (and again) in order to understand the commands to be entered so that you can make the simulation work and/or drive your race-car.

As soon as you manage to start your car, then manually stop the simulation, go on to the next step.
You can also take advantage of this step to test the commands and observe their effects, as well as on the `n4s` answers for these commands.

Step 3 - AI

Manually entering the commands to start the simulation is not really satisfactory. Making a program that decides by itself and automatically enter the command is way better!
The final goal of this Bootstrap is to create a binary, named `ai` that starts the simulation, moves the car and stops it 3 feet before hitting the rear wall, and then stop the simulation.

We're going to work toward this goal via several smaller objectives.
As a reminder, you'll need to execute the `pipes.sh` script in order to test your program. It will take care of "connecting" your `ai` program and the `n4s` binary.

Your program's standard output will automatically redirect toward the `n4s` binary (via a pipe). If you need to display debugging information, for example, you can write it on the error output.

Objective 1

Make a program that sends a command (whatever one you want) and waits for a response. Next, display this response on the error output.
Do the same thing with several different commands in order to discern, within your program, the different error/success messages that your `n4s` will respond to.

Objective 2

Make an `ai` program that starts up a simulation, then stops it after a certain number of cycles (you may choose the number that you want as long as it's within reason.)
A cycle is simply the equivalent of a well-timed loop. Basically, the `n4s` binary and the simulator will sychronize with each cycle.

The `CYCLE_WAIT` command enables you to receive this command's response a few cycles later. It proves to be useful if, during your design, you decide that certain actions completed by your `ai` won't be recalculated for a little while. For example, in right line, you're probably not going to need to recalculate the action you wish to complete for each cycle. You should be driving straight, so no need to ask yourself the same question with each cycle.

Objective 3

Enhance your `ai` program so that it is able to start up the simulation and then send a single command, just once, in order to move the car forward. Watch what happens.
Once you have succeeded in making the car move forward, you will certainly have noticed that, by only sending the command once, the car continues to move forward. Indeed, by sending the command, you have attributed a new desired speed value. This will not change unless you send the command again with a new value.

Objective 4

Enhance your `ai` so that it is able to start up the simulation and then drive the car to make it move to the end of the track, without touching the rear wall.

The `GET_INFO_LIDAR` command will be useful. Don't hesitate to launch (again) `n4s` alone in order to understand what this command returns, in addition to the indications presented in the project description.

Try to master the car's speed so that you don't violently come to a stop, instead, progressively stop just 3 feet before the wall.

Each square on the groud has a dimension of 20in x 20in.

v 2.1