Appendix A

Instructions Manual

Requirement	
Operating system	Ubuntu (tested on: v18.04 and v20.04)
Graphics Card	At least a 6 GB (recommend 8 GB) dedicated GPU.
Disk free space	About 21.5 GB.

A.1 Installation

Recommended:

Create a working environment as follows:

```
mkdir work
cd work
```

A.1.1 Install ML-MAS Framework with CARLA and LAV model

1. Extract the provided project folder into your work directory.

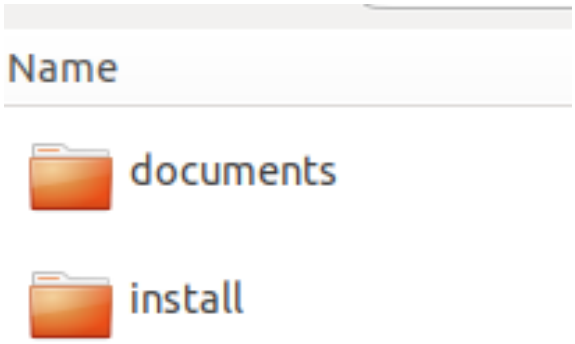

Figure A.1: The Project Extracted Folder.

2. Open the terminal and change the directory to the install folder.

   While you are in the work folder in the terminal, run the following:

```
cd install
```

3. Run the setup.sh

   This script will do the following:

1. Download the LAV model weights (200MB), and extract them to the right location in the project.

2. Download CARLA 0.9.10.1 (3.7GB) with the required additional maps (1.7GB), and extract them to the right location in the project.

```
./setup.sh
```

4. The project folder is ready.

   You can move the “MLMAS_Project“ folder to your favourite work location.

A.1.2 Prepare Jason to work locally

1. Install JDK 17.

```
sudo apt-get install openjdk-17-jdk
```

2. Navigate to the JasonLib folder.

   (MLMAS_Project > MLMAS_Framework > JasonAgent > JasonLib)

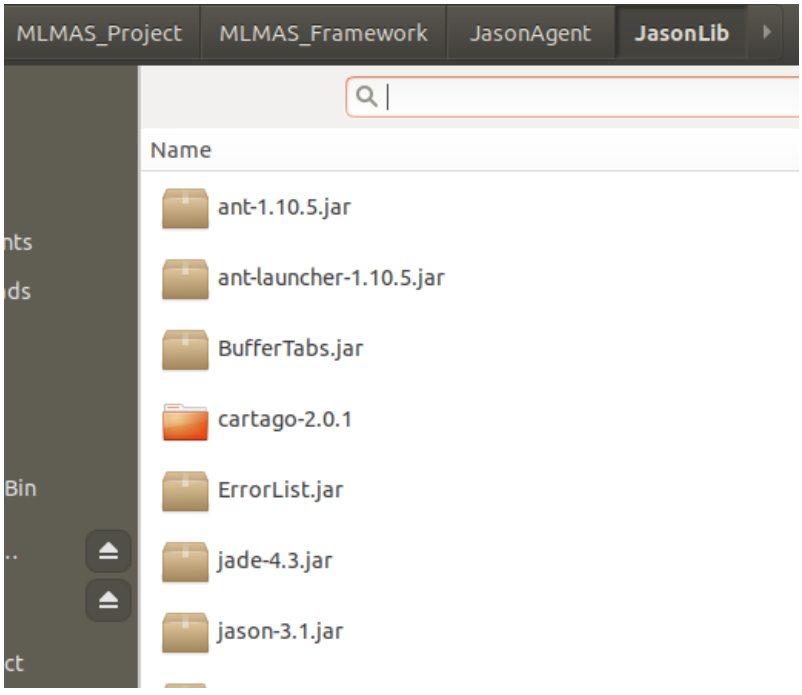

Figure A.2: Navigation to JasonLib Folder.

3. Double click in “Jason-3.1.jar“. (if not run, you need to add execution permission)

   - Confirm the “Jason-3.1.jar“ location.
   - Confirm the “ANT Libs“ folder location that points to this folder (JasonLib).
   - Click “Save Configuration and Exit“.

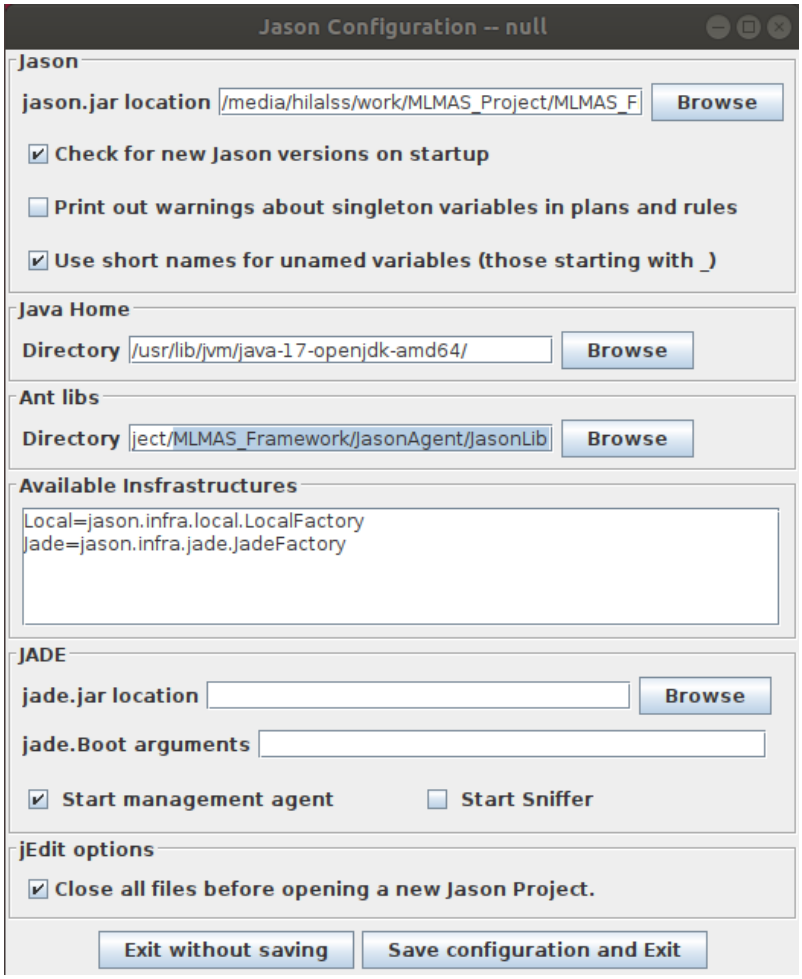

Figure A.3: Jason Configuration.

A.1.3 Final Project File Structure

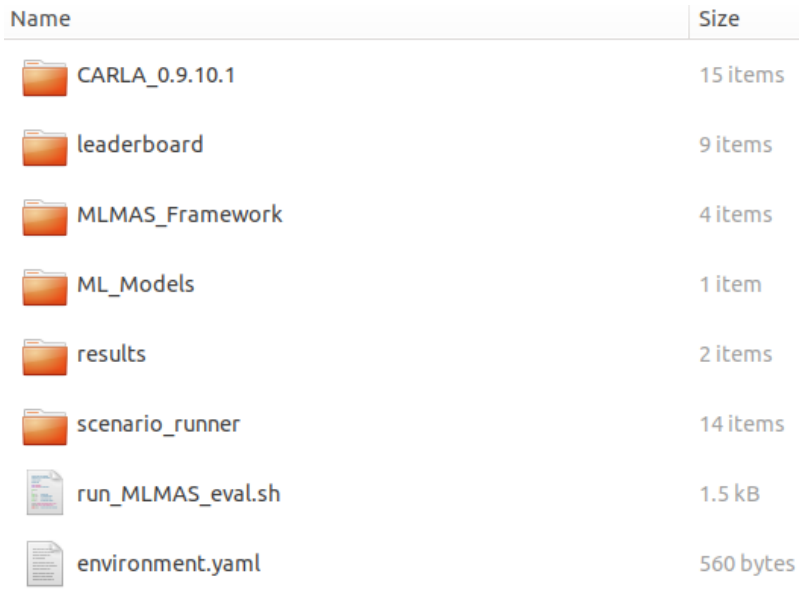

Figure A.4: The Project Root Folder.

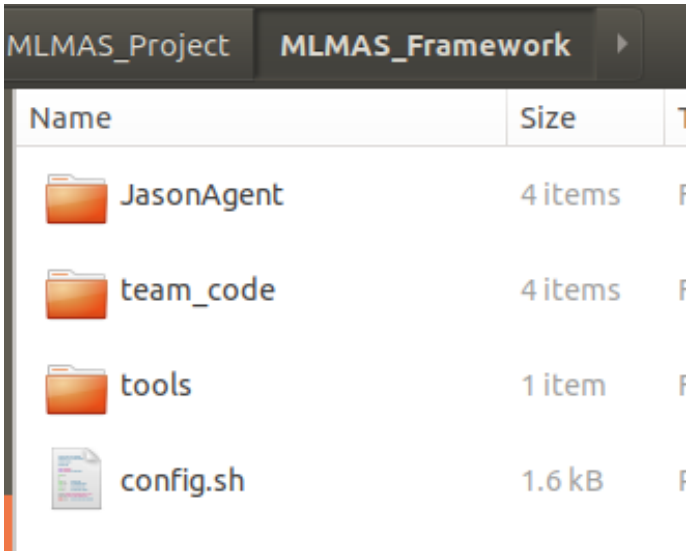

Figure A.5: The MLMAS_Framework Folder.

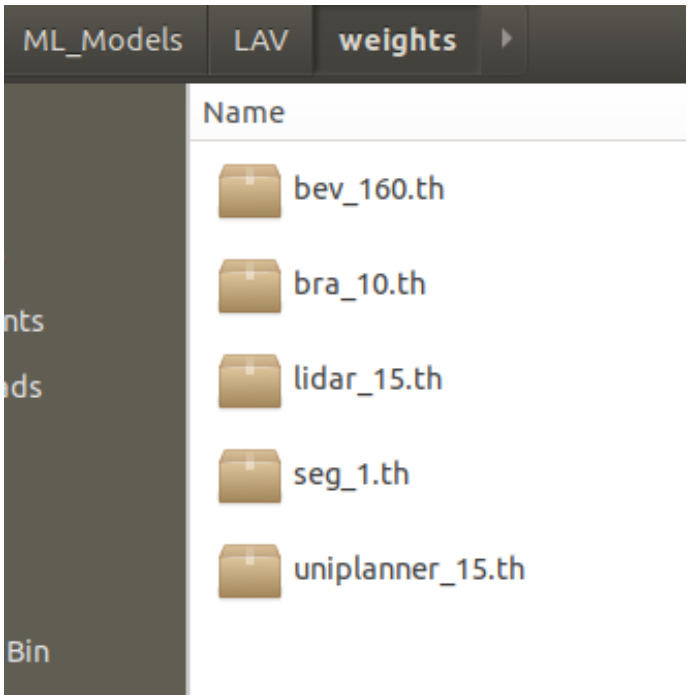

Figure A.6: The LAV Model Weights Folder.

A.1.4 Prepare Python Environment and Install Anaconda

1. Install Anaconda .
   - Download the Anaconda shell-ready file form here.
   - Run the script.
2. Create “mlmas” python environment.
   - Open the terminal.
   - Navigate to the “MLMAS_Project“ folder.

```
cd MLMAS_Project
```

- Creat the “mlmas” anaconda environment.

```
conda env create -f environment.yaml
```

Note: the environment will automatically install the required python version with the required libraries.

3. Activate the new environment “mlmas”.

```
conda activate mlmas
```

4. Install the required PyTorch library.

Note: The following commands will include installing “PyTorch“, a graphics card driver and Cuda specific. For your pc, you may need to access the following supported link.	
Get the right Graphics card, Cuda. (note: currently, up to Cuda 11.6 version is supported by PyTorch)	CUDA Toolkit 11.3 Installation link
Get the right PyTorch based on the Cuda version you have.	PyTorch Installation Link

Hint: The instruction in this manual is based on the following pc specifications: **Operating system**: Ubuntu 18.04 desktop. **Graphics Card**: Nvidia GP104M [GeForce GTX 1070 Mobile] **GC Driver**: Nvidia-driver-465 **Cuda**: Cuda 11.3

1. Install PyTorch with Cudatoolkit.

```
conda install pytorch==1.11.0 torchvision==0.12.0 torchaudio==0.11.0 cudatoolkit=11.3 -c pytorch
```

2. Install “torch-scatter“ and other required libraries.

```
pip install torch-scatter torch-sparse torch-cluster torch-spline-conv torch-geometric -f https://data.pyg.org/whl/torch-1.11.0+cu113.html
```

3. Test the installation.

- Ensure that PyTorch is installed

```
python -c "import torch; print(torch.__version__)"
```

- Verify the CUDA version PyTorch was installed

```
python -c "import torch; print(torch.version.cuda)"
```

A.1.5 Install Vulkan that required to run CARLA Simulator

1. Install Vulkan utility.

```
sudo apt-get update
sudo apt install libvulkan1 vulkan-utils
```

2. Install Vulkan SDK.

```
wget -qO - http://packages.lunarg.com/lunarg-signing-key-pub.asc | sudo apt-key
add -
sudo wget -qO /etc/apt/sources.list.d/lunarg-vulkan-bionic.list
http://packages.lunarg.com/vulkan/lunarg-vulkan-bionic.list
sudo apt update
sudo apt install vulkan-sdk
```

A.2 Configuration

Note: the project is ready and configured to run locally with the LAV model.

A.2.1 The ML-MAS framework main configuration file (config.sh)

Navigation: (MLMAS_Project > MLMAS_Framework > config.sh).

The configuration file allows the following:

1. Configure the Jason-CARLA bridge IP and port.

2. Specify the ML model that is to be integrated.

3. Specify the routes and scenarios files to be used for evaluation.

4. Specify the Leaderboard metrics results file location (the same location and file naming will be used for the additional Jason agent metrics CSV file).

5. Option to run ready “Jason agent” locally while evaluating.

config.sh

```
# ============================
# Jason - Carla Bridge configuration
# ============================
export BRIDGE_SERVER_IP=127.0.0.1
export BRIDGE_SERVER_PORT=60111

# ============================
# The ML Model configuration
# ============================
# The ML model (AutonomousAgent) agent location.
export ML_MODEL=ML_Models/LAV/team_code/lav_agent.py
# The ML model configuration file location.
export TEAM_CONFIG=ML_Models/LAV/team_code/config.yaml
# add any required python path required by the model
export PYTHONPATH=$PYTHONPATH:ML_Models/LAV/team_code


# ============================
# The Scenario Routes configuration
# ============================
export ROUTES=leaderboard/data/validation_routes/lav_collisions_tiny.xml
export SCENARIOS=leaderboard/data/longest6/eval_scenarios.json

# ============================
# The storing location of the results and the records
# ============================
# [The results json] work as checkpoint that you can resume to the route when stoped.
# and also the final leader board results will be stored there.
export CHECKPOINT_ENDPOINT=results/MLMAS_LAV_results.json # results file

# The [record path] will store the record log of the running scenarios
# which can be used to rerun the scenario and check the collisions and blocks
# using the provided utility code in this project.
# **Note: required direct path
export RECORD_PATH=$(pwd)/results/records/MLMAS_LAV/

# ============================
# Jason Agent
# ============================
# # set it to true, if you want to run the jason agent localy.
# # and automatically, but if it is in remote pc, set it to false.
run_jason_agent=true
```

Figure A.7: The ML-MAS Framework Configuration File.

A.2.2 Jason Agent configuration (config.properties)

Navigation: (MLMAS_Project > MLMAS_Framework > JasonAgent > carla_agents > config > config.properties).

The configuration file allows changing the Bridge server IP and Port. (default is localhost).

Note: the JasonAgent folder can be copied to another PC or server. While you only need to change this configuration to reach the server IP and port.

config.properties

```
# Jason-CARLA Bridge configuration

SERVER_IP = 127.0.0.1
SERVER_PORT = 60111
```

Figure A.8: The Jason Agent Configuration File.

A.3 Running

The following instruction describes how to run the CARLA simulator, followed by running the ML-MAS evaluation and finally describing how to utilise the utility code provided.

A.3.1 Running CARLA Simulator

1. Open the terminal and navigate to the CARLA while you are in the project folder (MLMAS_Project) as follows:

```
cd CARLA_0.9.10.1/
```

2. (option) Set the stack memory to unlimited (to overcome full stack memory (rarely happens)):

```
ulimit -s unlimited
```

3. Run the Simulator:

```
./CarlaUE4.sh -vulkan
```

Figure A.9: Running CARLA Simulator Screenshot.

A.3.2 Running the ML-MAS Framework Evaluation

1. Make sure that the CARLA simulator runs, as in the previous step.
2. Open a new terminal and Activate the “ML-MAS” environment.

```
conda activate mlmas
```

3. run the following from the project folder (MLMAS_Project):

```
./run_MLMAS_eval.sh
```

When you run the evaluation, it will evaluate the framework through the provided routes and scenarios specified in the configuration file and report the Leaderboard results and the additional Jason agent metrics in the results folder.

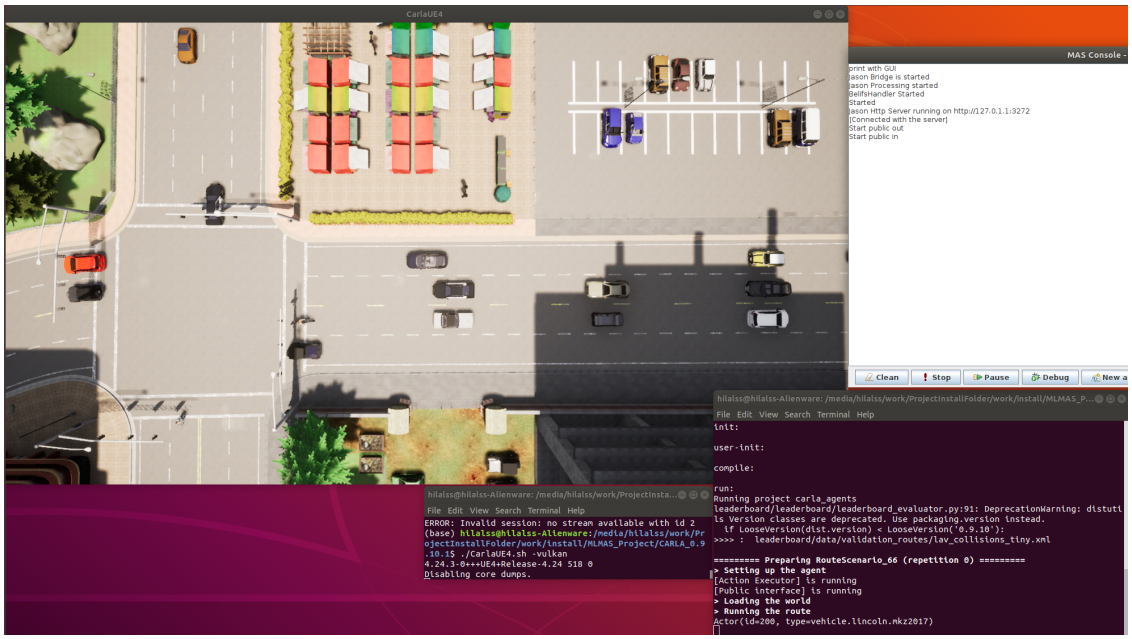

Figure A.10: Running the Evaluation.

Note: CARLA Leaderboard scenario runner will use the result JSON file as a checkpoint for the following purpose:

1. If the scenario stops for any reason, it will resume to the last scenario, when rerun.

2. If the evaluation is completed, It will not allow another run until you either change the results file name or backup that result in another location and re-run the evaluation. This is to make sure that you will not accidentally lose the results.

A.3.3 Utility Code

Navigation: (MLMAS_Project > MLMAS_Framework > tools > utility.ipynb)
This is a jupyter notebook code that helps in utilising the generated scenarios records and results, which provide the following:
1. Utilising the scenarios records logs:

- List the required information where there is a car block in the scenario.

- List the required information for a collision in the scenario.

- Rerun the scenario at a specific time (e.g., when the collision happened).

The code will connect with the CARLA Simulator and run the specific scenario from the recorded log.

Hint: The recorded logs will be available in the configured location in (config.sh) after the evaluation.

```
# The [record path] will store the record log of the running scenarios
# which can be used to rerun the scenario and check the collisions and blocks
# using the provided utility code in this project.
# **Note: required direct path
export RECORD_PATH=$(pwd)/results/records/MLMAS_LAV/
```

Figure A.11: The Location of the Stored Records Logs After Evaluation.

The output record logs for this project can be downloaded from here:

- LAV Longest6 Evaluation Records (1.1GB).

- ML-MAS Longest6 Evaluation Records (1.2GB).

2. The utility code can be used to display the results tables and graphs.

Running the code:

1. Activate the new environment “mlmas”.

```
conda activate mlmas
```

2. *Install Jupyter notebook (if not installed yet in “mlmas” environment).

```
conda install -c anaconda jupyter
```

3. Run Jupyter notebook: (should be run from “mlmas” environment).

```
jupyter notebook
```

4. Navigate to the utility.ipynb and run it from the browser.

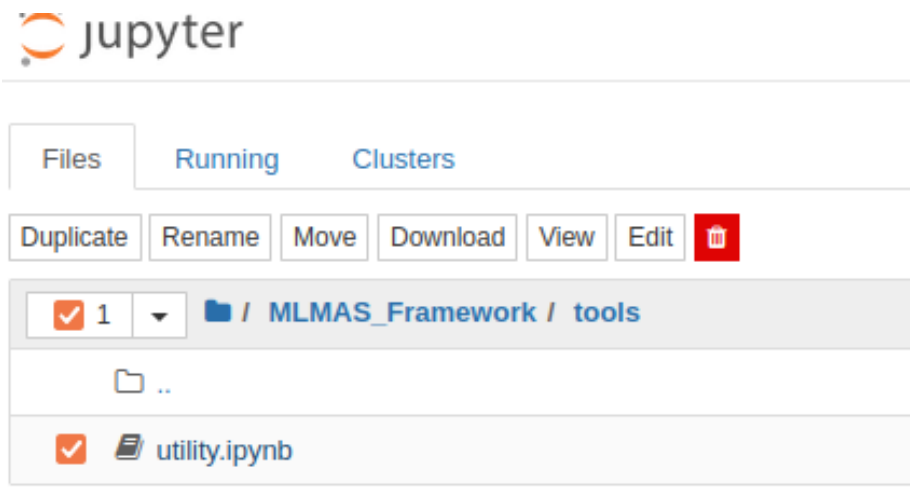

Figure A.12: Navigating in Jupyter Notebook to the Utility code file.

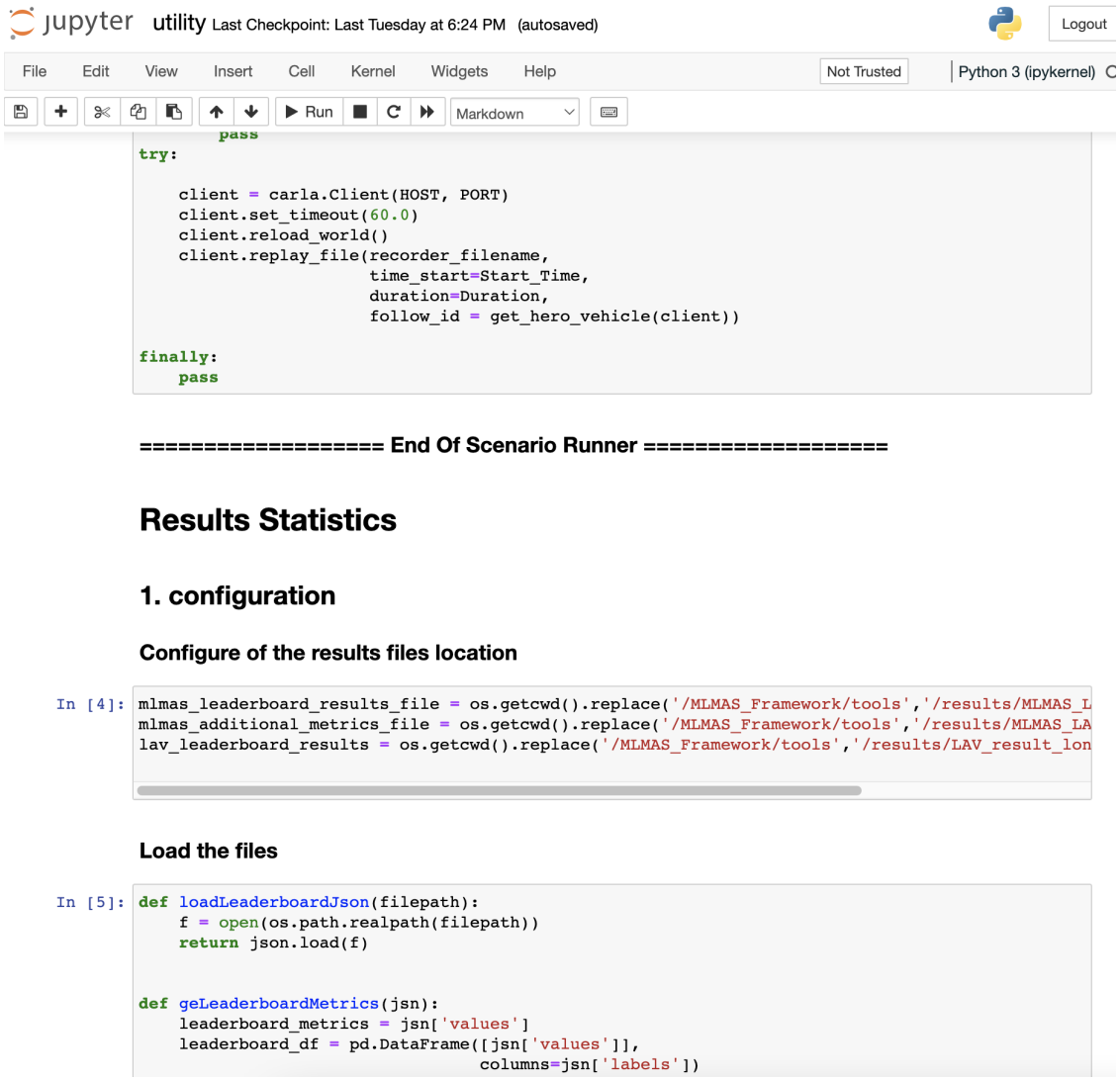

Figure A.13: The Utility Code Screenshot.

A.4 Modifying

This section describes how to modify the project source code and then deploy it.

A.4.1 Jason Agent Code

1. Install Eclipse IDE with Jason plugging.

   Follow the official instructions here.

2. Import the Jason project.

   - Navigate to the project file:

   (MLMAS_Project/MLMAS_Framework/JasonAgent/carla_agents/carla_agents.mas2j).

   - here: is a described how to import a Jason project.

3. Change the bridge server IP if required.

   If you run the project on a different pc than the server, then change the server IP and port configuration accordingly, as described in the Configuration section.

   The file structure of the Jason agent code and description:

Source Path: MLMAS_Project/MLMAS_Framework/JasonAgent/carla_agents/		
File Name	**Path from the source**	**Description**
carla_agents.mas2j	/	The Jason project file.
config.properties	/config/	The configuration file changes the bridge server IP and port.
carla_control.asl	/src/asl/	The Jason code for the carla_control agent is responsible for applying the rational intentions and plans based on the received beliefs.
BeliefsHandler.java	/src/java/carla_agents/	The responsible class manages the beliefs based on the received sensor data from CARLA.
CarlaEnv.java	/src/java/carla_agents/	The Jason environment class in java.
JsonProcessing.java	/src/java/carla_agents/	A class responsible for packing and unpacking the JSON format messages.
JasonCarlaBridge.java	/src/java/CarlaSocket/	The Jason-CARLA bridge class acts as an independent client to handle communication with the server bridge.
PublicIn.java	/src/java/CarlaSocket/	A class handles all received messages in separate threads as in the bridge design.
PublicOut.java	/src/java/CarlaSocket/	A class handles all sending messages in separate threads as in the bridge design.

Table A.1: The BDI Agent source code files.

Code deployment:

1. If the project is imported directly from:

   (MLMAS_Project/MLMAS_Framework/JasonAgent/carla_agents/)

   - The code will run as it is without doing any further steps.

2. If the project developed in different location:

   - Copy the (carla_agents) folder into:

   (MLMAS_Project > MLMAS_Framework > JasonAgent/).

   - Done, it will be ready to be utilised by the framework.

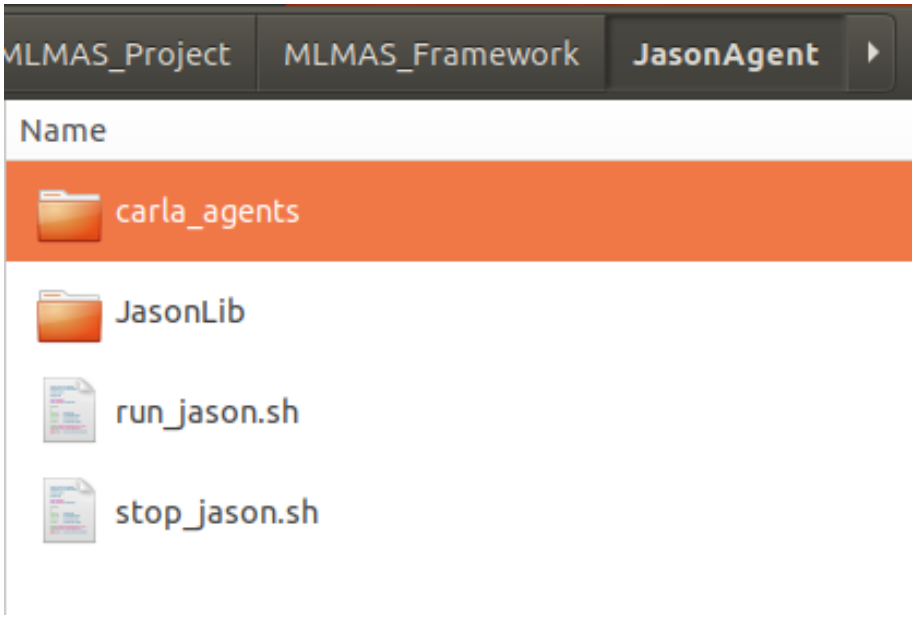

Figure A.14: The Location of the Deployed Jason Agent.

A.4.2 The Project Python Code

Use your favourite python editor to modify the code:

Source Path: MLMAS_Project/MLMAS_Framework/team_code/		
File Name	**Path from the source**	**Description**
jason_carla_bridge.py	/	The server part of the CARLA-Jason Bridge.
ml_mas_agent.py **(The Integration Class)**	/	The integration class inherits the CARLA leaderboard (AutonomousAgent class) to be ready to be evaluated by it. This class automatically loads the other ML model based on the configuration, receives and processes the sensor's data, and orchestrates the controls between the ML model and the Jason agent.

Table A.2: The project Python source code files.

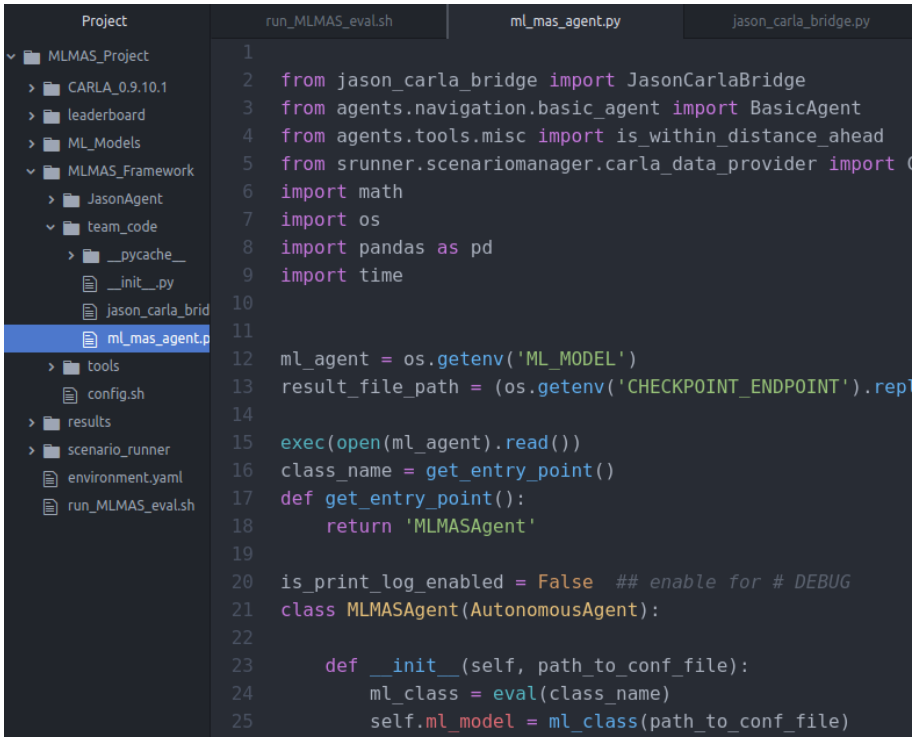

Figure A.15: The Integration Class Pythone code in Editor.