ScaRLib: A Framework for Cooperative Many Agent Deep Reinforcement learning in Scala – Artefact Guidelines

Davide Domini[1][0009−0006−8337−8990], Filippo Cavallari[1][0009−0006−4050−9989], Gianluca Aguzzi[1][0000−0002−1553−4561], and Mirko Viroli[1][0000−0003−2702−5702]

Alma Mater Studiorum – Università di Bologna
{filippo.cavallari2}@studio.unibo.it
{davide.domini2,gianluca.aguzzi,mirko.viroli}@unibo.it

1 Introduction

This document describes the artefact associated with the tool paper entitled ScaRLib: A Framework for Cooperative Many Agent Deep Reinforcement learning in Scala" and accepted at COORDINATION'23. More specifically, we demonstrate the validity and usefulness of our tool by claming the Available and Functional (And Reusable) badges, elucidating how these claims were substantiated, and offering instructions on utilizing the artefact to duplicate the simulations and outcomes featured in the principal paper.

2 Claims

We claim the following badges:

C.1 **Available** (2) since the library is published on Maven[1] (also in Zenodo[2]) and the simulation is available through Zenodo[3].

C.2 **Functional (And Reusable)** (1.2) since the artefact is:
- **documented**, since the library has a README explaining the main abstractions and how to use them, it has a template [4] that can be used to create new ScaRLib projects and a simulation where it actually shows how to use it;
- **consistent**, since it is the implementation of what is described in the paper, both at library and simulation level;
- **complete**, since the information provided is comprehensive and contains all the necessary details to replicate the results presented in the paper;
- **exercisable**, since it is possible to run the simulations and to observe the results leveraging the scripting included

[1] https://central.sonatype.com/artifact/io.github.davidedomini/ScaRLib/1.6.4
[2] https://doi.org/10.5281/zenodo.7830970
[3] https://doi.org/10.5281/zenodo.7831045
[4] https://github.com/ScaRLib-group/ScaRLib-experiments-startup

- **reusable**, since the library has been published on manve central and can be integrated into any modern java project by following the instructions described in the project template.

In the following section, we will describe how to use the artefact to replicate the simulations and outcomes featured in the paper.

3 Installation and Smoke test.

We have created a VirtualBox virtual machine with the minimum installation required to perform the experiments. To import the OVA file associated with the virtual machine, you will need to open VirtualBox and select *File* and then *Import Appliance*. After that, you will need to choose the OVA file and click on *Import*. For more information, please visit `https://www.virtualbox.org/manual/ch01.html#ovf`.

If you are using your own computer, please ensure that the following installations are present:

- JDK (version 17 or above)
- Python (version 3.9 or above)

To run a smoke test, follow the below steps:

- Clone the repository at the designated location using the command:
  `git clone <repo url>`
  in the VM, the repository is already cloned in the /Desktop folder called ScaRLib-flock-demo
- Open the terminal inside the folder
- Install the dependencies using the command:
  `pip install -r requirements.txt`
- Execute the command: `./gradlew runSmokeTest`
- If everything is working properly, you should be able to see a swarm of drones similar to the one shown in Figure 9 (a) of the paper.

Now you are ready to reproduce the experiments described in the paper.

4 Project structure

The project repository is structured as follows:

- `./src`: this folder contains everything necessary to simulate the flock of drones. Here you will find both the main code for the training phase and the code for the evaluation phase. You can also find the YAML description for each simulation.
- `/runs`: contains the logs of the simulations related to training phases.
- `./network`: this folder contains the neural network used in the evaluation phase. Here will be also stored the weights of the trained neural network foreach each simulation.
- `process.py`: the python script used to generate the charts shown in Figure 10
- `./data`: contains the data used to generate the charts shown in Figure 10

5 Reproducing the simulations and charts

To reproduce the simulations and the charts discussed in the Section 4, follow the below steps:

- By executing the following command it is possible to start the learning process for the flocking case study

```
./gradlew runCohesionAndCollisionTraining
```

  If you want to visualize the progress of the simulation you can use

```
/gradlew runCohesionAndCollisionTrainingGui
```

  If you want to see the evolution of the reward function (Figure 8), you can leverage tensorboard by executing the following command:

```
tensorboard --logdir runs
```

  This could be particularly slow on a local machine (∼2 hours). For this reason, we have already uploaded the logs of the training phase to the `runs/` directory and the weights of the neural network to the `networks/network` file.
- To replicate the evaluation phase of several deployments shown in Figure 9, run the following command:

```
./gradlew runCohesionAndCollisionEval
// with the gui:
./gradlew runCohesionAndCollisionEvalGui
```

  This will launch 3 batch of simulation. Each batch will contain 16 simulations. The first batch consists of 50 nodes, the second one of 100 nodes and the third one of 200 nodes. This will produce the data used to generate the charts shown in Figure 10 in the folder `./data`. Also in this case, we have already uploaded the data to the `data/` directory since the evaluation phase is very time consuming (∼1 hour).
- To generate the figures, execute `python plotter.py`. Make sure to install the required Python dependencies on your local machine with `pip install -r requirements.txt`. The plots will be saved in the folder `./charts` after the script runs successfully..