Autonomous and Adaptive Systems
Project Guidelines

Giorgio Franceschelli
PostDoc @UNIBO
giorgio.franceschelli@unibo.it
giorgiofranceschelli.github.io

Mini-Project for the Exam

You need to develop a project and:

- Write an up to 6-page short report (paper-style) to be submitted in advance (to both the professor and me: giorgio.franceschelli@unibo.it) **and** a repo with the code (either a notebook or .py files).
- You should not send the code (especially a zip file), but please send a link to a repository (e.g., GitHub).
- The project **must be submitted before the closing date for signing-up for the exam**.
- Prepare max 3 slides + slide for the title (and ideally a working demo) to be discussed during the oral exam.

The project will contribute to **1/3 of the final mark**. So, in terms of complexity, the project is equivalent to **2-3 credits**.

Mini-Project for the Exam

The code and the report must be original! Reusing/repackaging code is not considered a valid project.

If you use lines of code taken from somewhere, you will need to clearly state it – and the core part (we will see soon what I mean) of the project in any case must be original.

Please note that the report/code will be checked for plagiarism.

The report must be submitted using the NeurIPS LaTex style that can be found at this address:

https://media.neurips.cc/Conferences/NeurIPS2025/Styles.zip

Important: the title and type of mini-project does not need to be approved in advance.

Mini-Project for the Exam

The project must be of one of the following two types. Either:

- A project based on the Stag-Hunt environments; or
- A project based on a custom environment designed by you.

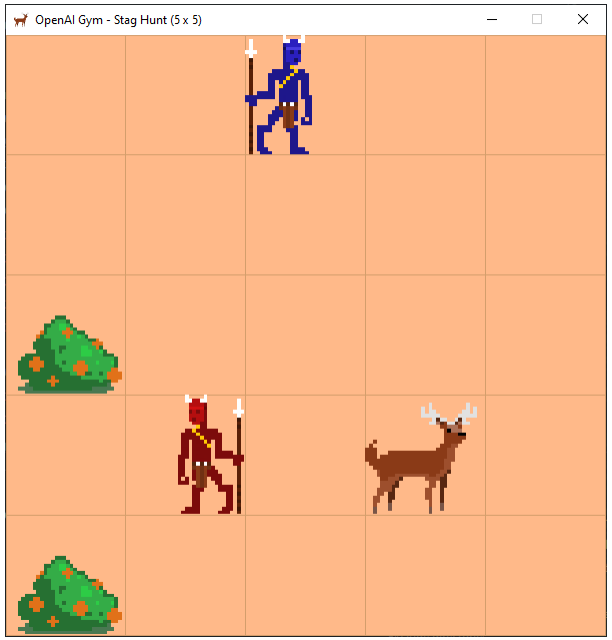
Mini-Project for the Exam

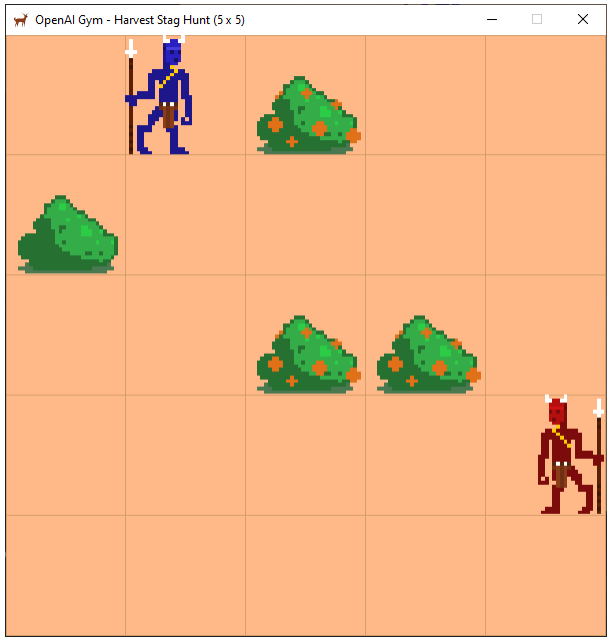
The project must be of one of the following two types. Either:

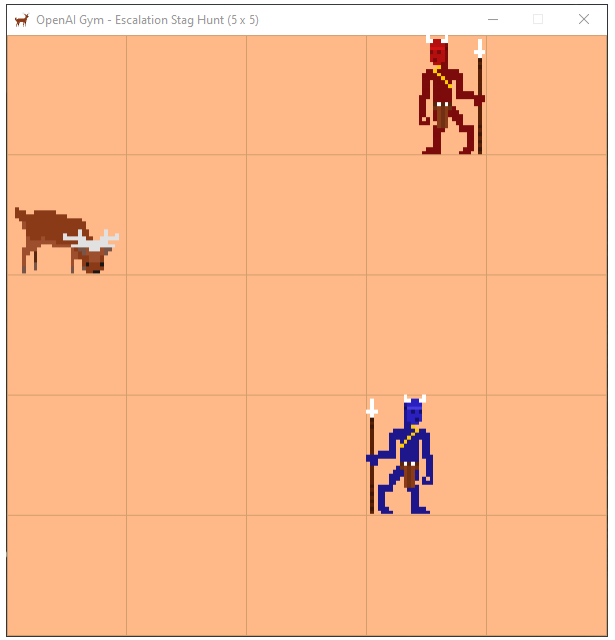
- **A project based on the Stag-Hunt environments; or**
- A project based on a custom environment designed by you.

Stag-Hunt

Stag-Hunt is a set of gym-like environments for multi-agent cooperation. In particular, it provides 3 grid-based stochastic games (Hunt, Harvest, and Escalation) that aim to explore prosocial behaviors in multi-agent RL.







Stag-Hunt

Multi-agent: There are two agents involved, so at each time step you will have to provide two actions to the environment (at least one agent must be trained via RL, but more about this later).

Prosocial: As in the more famous Prisoner's Dilemma, maximizing the cumulative reward requires both agents to cooperate; but cooperating when the other agent is not leads to a punishment (so not cooperating leads to a positive, but non-optimal, return).

Stochastic: Each episode starts with the agents in the same positions, but the goals' and obstacles' positions are randomized.

Customizable: The size of the state (as an NxN grid), the length of each episode, and all involved rewards and punishments are not fixed and can be modified (even though I suggest starting from default values).

Stag-Hunt with Gym(nasium)

```
>>> git clone https://github.com/giorgiofranceschelli/Gymnasium-Stag-Hunt.git
>>> cd Gymnasium-Stag-Hunt
>>> pip install .

>>> import gymnasium as gym
>>> import gymnasium_stag_hunt
>>> game = ‘Hunt’ # or ‘Harvest’ or ‘Escalation’
>>> configs = {...} # all env arguments you want to customize
>>> env = gym.make(‘StagHunt-’+game+‘-v0’, **configs)
```

And then you can use it as a classic gymnasium environment! But remember: you have two agents, so two actions to pass as a list of size 2, two states and two rewards to get, etc.

In any case, check the repo:

https://github.com/giorgiofranceschelli/Gymnasium-Stag-Hunt

Stag-Hunt with Gym(nasium)

Apart from the `env_name`, `make()` accepts several interesting arguments:

- `grid_size=(5,5) # dimension as (M,N) of the simulation grid (at least (M,N) = (3,3))`
- `obs_type='image'  # the type of observation to use: 'image' for RGB tuples (pixel colors), 'coords' for a coordinate array with the x,y coordinates of each agent and goals/obstacles`
- `max_timesteps=1000 # the number of timesteps for each episode before it ends`
- `enable_multiagent=True # whether to control both actors or to have one of them acting randomly or naively (to be set via opponent_policy)`
- `flip_obs=True # whether to invert agents positions when returning the second obs (only for obs_type=='coords')`

And many others to control each single reward and punishment involved in the specific environment.

Stag-Hunt – Final Notes

The minimum necessary is to train two agents that learn to correctly cooperate on one Stag-Hunt environment among Hunt, Harvest, and Escalation.

To get a higher mark, you can go beyond this threshold in several ways:

- Learning agents that can cooperate with multiple partners
- Learning agents that can cooperate on more than one environment
- Studying under which rewarding/punishing conditions cooperation is still possible
- Studying whether your approach scales up with state size
- Using LLMs as RL agents
- … Something I could not come up with in the first minute of reasoning!

Mini-Project for the Exam

The project must be of one of the following two types. Either:

- A project based on the Stag-Hunt environments; or
- **A project based on a custom environment designed by you.**

Custom Environments

As an alternative, you can create your own environment in a gym(nasium)-like style to be solved by any RL algorithm.

The custom environment can implement a board game, a card game, a maze-like or any other sort of video game, can be single-agent or multi-agent, and it can also encapsulate other tasks to be solved with RL, e.g. generative modeling.

The only requirement for the environment is to be **new** (not already available online) and **not too simple** (at least more complex than let's say tic-tac-toe).

The degree of complexity of the environment will be taken into consideration for the assessment.

Custom Environments with Gym(nasium)

A custom environment should work exactly as a classic environment. The best way to define a new one is by sub-classing `gym.Env`.

You will then need to define at least `reset()` and `step()` methods complying with the standard gym(nasium) signatures plus `action_space` and `observation_space` attributes.

Here to check for its implementation and documentation:

https://github.com/openai/gym/blob/master/gym/core.py # old gym library

https://gymnasium.farama.org/ # new gymnasium library

Custom Environments – Rule of Thumb

It is important that we can easily run some straightforward experiments on your environment, just to make sure it works as expected.

In general, make sure it can work with a simple code like this:

```
>>> env = CreateEnv(...)
>>> _, _ = env.reset()
>>> _, reward, _, _, _ = env.step(env.action_space.sample())
>>> print(reward)
```

Custom Environments – Final Notes

Defining a working custom environment is not as simple as it seems – the same state can be represented in different ways, there may be invalid actions to deal with, several reward functions can be used for the same problem, etc.

In addition, you also need to implement an RL algorithm, even a simple one, to learn how to solve your environment (as a demonstration of the correctness of your custom environment).

From your code+report we expect to see what you have done for both environment and agent, with a focus on the problem you have modeled and the motivations for your implementation choices.

Final Remarks - IMPORTANT

The final mark for the project will acknowledge three things:

- The implementation
- The report
- The oral discussion

In particular, the oral discussion is crucial – you cannot achieve full marks if you are not able to explain and reason about what you have done, even if your code, results, and report is excellent!