Autonomous and Adaptive Systems
Project Guidelines

Giorgio Franceschelli
PhD Student @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 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 (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/NeurIPS2024/Styles.zip

Mini-Project for the Exam

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

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

Overcooked

Overcooked is a benchmark for **fully cooperative multi-agent systems**.

It is commonly used to test out the robustness of collaborative agents, but can also serves to evaluate the generalization abilities of cooperative agents.

The goal of the game is to deliver soups as fast as possible. Each soup requires placing up to 3 ingredients in a pot, waiting for the soup to cook, and then having an agent pick up the soup and delivering it.

Overcooked

In order to optimize the reward received (i.e., total number of soups delivered), the two agents must develop cooperative strategies and coordinate to solve the task together.

In addition, it is possible to change the kitchen layout to train and/or test the agents' generalization capabilities across different levels.

[Note that another degree of generalization can also be reached by changing the partner of the agent!]

If all layouts are different but require the same skills, they are suitable for evaluating generalization capabilities; the agents cannot just learn a sequence of combined actions.

More practically, the skills learned to solve a sub-set of all possible layouts should allow the agents to solve different, unseen layouts!

Overcooked in practice

```
>>> git clone https://github.com/HumanCompatibleAI/overcooked_ai
>>> cd overcooked_ai
>>> pip install -e .
```

```
>>> from overcooked_ai_py.mdp.overcooked_env import OvercookedEnv, Overcooked
>>> from overcooked_ai_py.mdp.overcooked_mdp import OvercookedGridworld
>>> base_mdp = OvercookedGridworld.from_layout_name("cramped_room") # or other layout
>>> base_env = OvercookedEnv.from_mdp(base_mdp, info_level=0, horizon=400)
>>> env = Overcooked(base_env=base_env, featurize_fn=base_env.featurize_state_mdp)
```

In any case, check the documentation:
https://github.com/HumanCompatibleAI/overcooked_ai

Overcooked in practice

Once you have defined your env as seen, you can use it as a Gym environment (with reset() and step() methods and observation_shape and action_shape fields).

Notes:

1) env.observation_space returns a wrong interval of numbers (but a correct len of each state)
2) env.step() does not call .reset() when the game is finished, so you need to manually call env.reset() if done
3) The state returned by step() and reset() does not only contain the array representing the current state, but also other information
4) env.render() simply returns the rgb pixels of the state - to directly visualize the state use StateVisualizer().display_rendered_state() (see documentation).

Overcooked in practice

As we saw, the environment is defined on a single layout.

If you want to train or test on generalization capabilities, you need some workaround, e.g.:

```
class GeneralizedOvercooked:
    def __init__(self, layouts, info_level=0, horizon=400):
        self.envs = []
        for layout in layouts:
            base_mdp = OvercookedGridworld.from_layout_name(layout)
            base_env = OvercookedEnv.from_mdp(base_mdp, info_level=info_level, horizon=horizon)
            env = Overcooked(base_env=base_env, featurize_fn=base_env.featurize_state_mdp)
            self.envs.append(env)
        self.cur_env = self.envs[0]
        self.observation_space, self.action_space = self.cur_env.observation_space, self.cur_env.action_space
    def reset(self):
        idx = random.randint(0, len(self.envs)-1)
        self.cur_env = self.envs[idx]
        return self.cur_env.reset()
    def step(self, *args):
        return self.cur_env.step(*args)
    def render(self, *args):
        return self.cur_env.render(*args)
```

Overcooked – Final Notes

Overcooked is a hard benchmark, especially when using it for generalization – we do not expect you to reach state-of-the-art results.

The minimum goal is to implement a (fairly) advanced algorithm and solve a single layout (e.g., achieving > 50 mean reward in cramped_room). However, if you want to be able to get the maximum score, you will need to extend it to the generalization problem, i.e., by considering multiple layouts and/or multiple pairs of agents.

Our evaluation will not be based (only) on achieved performance, but (also) on your reasoning behind your implementation choices!

From your code+report we expect to see a description of what you have done, your design choices, a presentation and discussion of the key results.

Mini-Project for the Exam

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

- A project based on the procgen environment; 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-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, but 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

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 signatures plus `action_space` and `observation_space` attributes.

While I suggest you to use the original gym, you can also use gymnasium by sticking with the new signature for `step()`.

Here to check for its implementation and documentation:

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

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, ecc.

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.