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 file).
- 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 project must be of one of the following two types. Either:

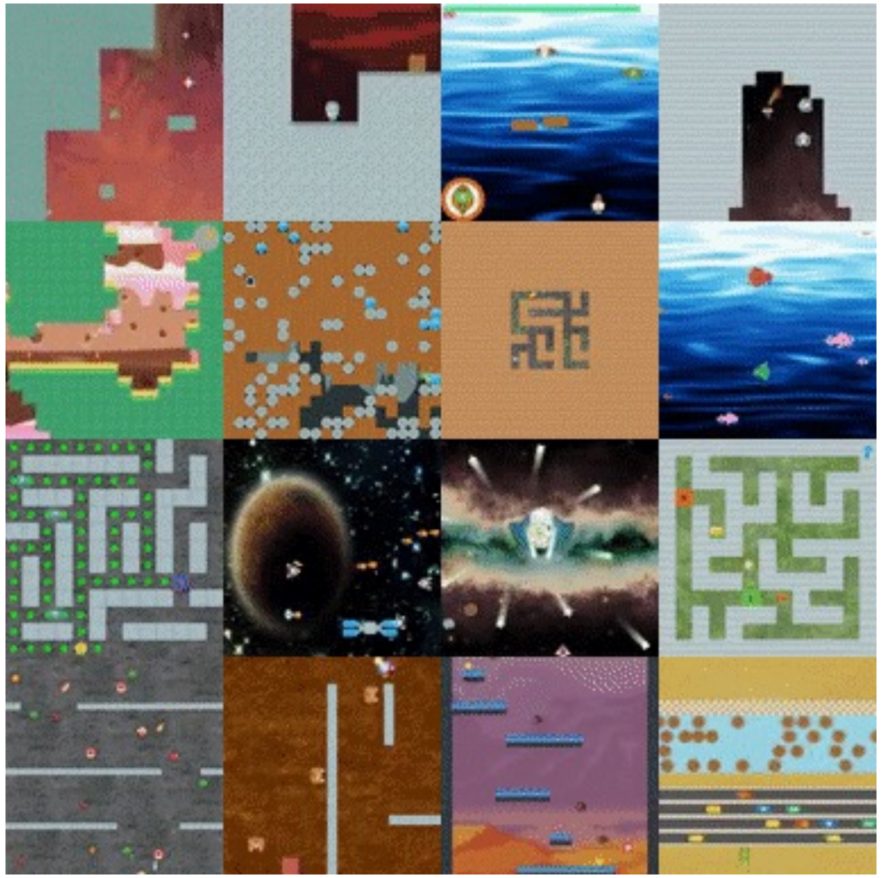
- A project based on the procgen environment; or
- A project based on a custom environment designed by you.

Mini-Project for the Exam

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

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

ProcGen



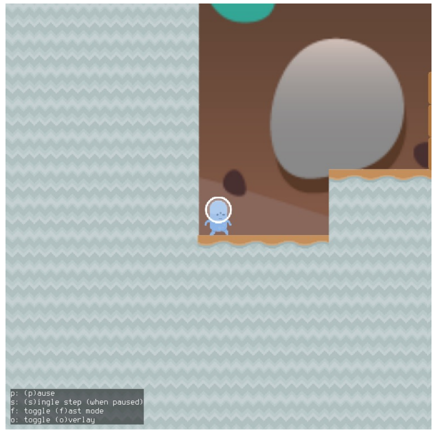
ProcGen is a suite of 16 procedurally-generated environments that represent the most famous benchmark for generalization.

Its goal is to provide a direct measure of how an agent learns generalizable skills, i.e., skills useful across different versions of the same game.

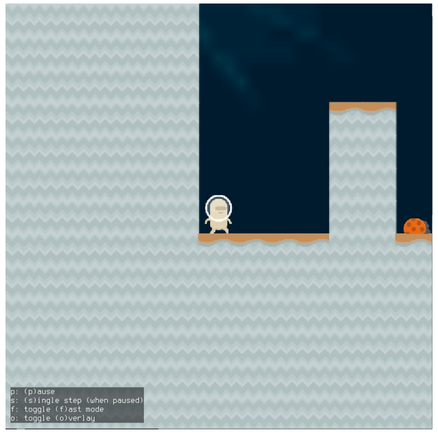
ProcGen

Procedurally generated = generated algorithmically through a combination of human-generated content (the building blocks of the games) and computer-generated **randomness**.

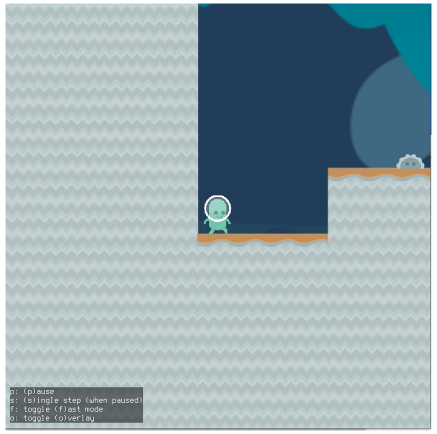
In this way, each episode is different! All possible episodes share the same characteristics (e.g. same reward functions, same type of obstacles, ecc.) but the sequences of actions required to solve them are different.



```
p: (p)ause
s: (s)ingle step (when paused)
f: toggle (f)ast mode
o: toggle (o)verlay
```



```
p: (p)ause
s: (s)ingle step (when paused)
f: toggle (f)ast mode
o: toggle (o)verlay
```



```
p: (p)ause
s: (s)ingle step (when paused)
f: toggle (f)ast mode
o: toggle (o)verlay
```

ProcGen

If all episodes are different but require the same skills, they are suitable for evaluating generalization capabilities: the agent cannot just learn a sequence of actions, it must learn what it means to solve that family of episodes.

More practically, the skills learned to solve a sub-set of all possible episodes should allow the agent to solve the entire set of possible episodes!

Therefore, you should train your agent constraining the environment to a fixed sub-set of episodes (also called levels), and then you should evaluate it on a version of the same environment without that constraint.

ProcGen with gym

```
>>> pip install gym  # please note: gym, not gymnasium
>>> pip install procgen

>>> import gym
>>> game = ‘coinrun’  # or any other procgen game
>>> env = gym.make(‘procgen:procgen-’+game+‘-v0’)
```

And then you can use it as a classic gym environment!

In any case, check the documentation:

https://github.com/openai/procgen

ProcGen with gym

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

- `num_levels=0   # the number of unique levels that can be generated (0 for unlimited levels).`
- `start_level=0   # the first level to be played (to be set with rand_seed for reproducibility).`
- `distribution_mode=‘hard’   # the complexity (i.e. number of timesteps required) of the game – I suggest you ‘easy’ mode.`
- `use_backgrounds=True    # whether to use human-designed backgrounds (with colors and variations across levels) or pure black backgrounds (useful for debug or with limited compute resources).`

And many others (see https://github.com/openai/procgen).

ProcGen with gym

Usually, training considers 200 levels in easy mode and 500 in hard mode.
For example:

```
>>> import gym
>>> game = ‘coinrun’
>>> seed = 1
>>> train_env = gym.make(‘procgen:procgen-’+game+‘-v0’,
num_levels=200, start_level=seed, rand_seed=seed,
distribution_mode=‘easy’)
>>> test_env = gym.make(‘procgen:procgen-’+game+‘-v0’,
start_level=seed, rand_seed=seed, distribution_mode=‘easy’)
```

ProcGen – Final Notes

ProcGen is a very hard benchmark – we do not expect you to solve all games or reach state-of-the-art results.

You will have to implement a (fairly) advanced RL algorithm, which at least is able to outperform a random agent on a sub-set of the 16 environments.

Total reward can be used a performance indicator, but others are possible (see papers discussed during the lectures).

Hint: start with a very simple configuration (e.g. no background, 1 level) and once you are confident your agent is learning properly, increase difficulty!

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:

- A project based on the procgen environment;
- **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 I 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.