QUALITY DIVERSITY

Quality diversity algorithms, rather than finding a single good solution to a problem, constructs an archive of different high performing solutions. "Different" can be defined in various ways, an obvious–and only recently possible–example being solutions that look different to a vision model [1]. A popular[1] quality diversity algorithm is the pedagogically named "Multi-dimensional Archive of Phenotypic Elites" (MAP-Elites) [2]. Think of it as having a fitness dimension, on which we maximize, and a set of behavioral dimensions we want to cover/explore.

1 | Evolutionary algorithms

Today (September 16th, 2025) we will be playing with evolutionary algorithms, as a way to get ready for quality diversity algorithms. Evolutionary algorithms are at their core, extremely intuitive: randomly mutate, see what works, and then further mutate on that. Inspired by the frequent bifurcation of species into two sexes*, we can further mix parts of one good solution with another, combining them to get a new (perhaps even better solution).

Evolutionary optimization pseudocode

```
for generation in range(N)
    fitness = eval_function(population)
    idxs = argsort(population)
    population = population[idxs]
    population = cross_over(population, n)
    population = mutate(population)
end
```

You will now:

1. Select a test function for optimization[2]
2. Implement it in python (and visualize it)
3. Find its optimal solution using an evolutionary algorithm from the lecture
4. Make some cool plots of the results (get creative)

[1] At least amongst the researches teaching you

[2] https://en.wikipedia.org/wiki/Test_functions_for_optimization

2 | PCGYM

Recall that Gym [3] is a framework for reinforcement learning environments, consisting of an `init` and a `step` method (the same as those in our `aigs/games.py` file). Note further that (as the term suggests) procedural content generation (PCG) focuses on the procedure that generates a given piece of content, rather than the content itself. To that effect we have made `pcgym` (itself derived from `pcgrl` [4]) that enables quick ideation of levels, supporting the kind of methods this lab is meant to have you play with. You will now:

1. Explore `pcgym`[3] and have a random agent play a level of any game.
2. Replace the random agent with a randomly initialized agent, that maps game states to action.
3. Improve the randomly initialized agent using the basic evolutionary algorithm used in the previous task.
4. Bonus: think about whether the cross-over operator makes sense for your agent, and why / why not this is the case.

3 | A*

A* (A-star) is a pathfinding algorithm combining cost-so-far and estimated cost-to-goal, balancing shortest-path (like Dijkstra) with goal-directed search (like greedy best-first). In games like Mario, A* is used to guide agents efficiently through levels by evaluating possible moves via a cost function (e.g., distance, obstacles) and a heuristic (e.g., estimated steps to the flag), generating optimal or near-optimal action sequences toward the goal. You will now:

1. Think a bit about A* (what it is and how you would implement it)
2. Implement A* or find an implementation online and have it play a level from `pcgym`

[3] My fork of `gym-pcgrl` modified to work with our course. It is located at https://github.com/syrkis/pcgym but is already included in our environment

4 | Content generation

We can optimize levels, just like we can optimize players. Thinking about what it means for a level to be "fit", you must now:

1. Define a fitness function for a level in `pcgym` (A^* can play a role in evaluating a level)
2. Define two behavioral dimensions, meaning ways in which levels can be different, that are not fitness related (e.g., number of jumps).
3. Generate an archive of good levels that are different from one another with (Timothée's beloved) MAP-Elite algorithm.

Index of Sources

[1] A. Kumar et al., "Automating the Search for Artificial Life with Foundation Models," no. arXiv:2412.17799. arXiv, Dec. 2024. doi: 10.48550/arXiv.2412.17799.

[2] J.-B. Mouret and J. Clune, "Illuminating Search Spaces by Mapping Elites," no. arXiv:1504.04909. arXiv, Apr. 2015.

[3] M. Towers et al., "Gymnasium: A Standard Interface for Reinforcement Learning Environments," no. arXiv:2407.17032. arXiv, Nov. 2024. doi: 10.48550/arXiv.2407.17032.

[4] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius, "PCGRL: Procedural Content Generation via Reinforcement Learning," no. arXiv:2001.09212. arXiv, Aug. 2020. doi: 10.48550/arXiv.2001.09212.