Run Vignette step by step

You will find here a short -but detailed- tutorial on how to operate the *Vignette* tool.

We will train a SAC model on the *Pendulum* environment, and compute and save its *Vignette* for visualization purposes.

The following instructions can all be executed from the *example.sh* file.

Let's train *Pendulum* for 10000 steps and compute its *Vignette* to visualize a glimpse of the learning space around the 8000th step. We also wish to observe the relative location of the model at other learning steps.

If you have any question or remark about the tool, please email me at y.elrharbifleury@gmail.com

Training the model

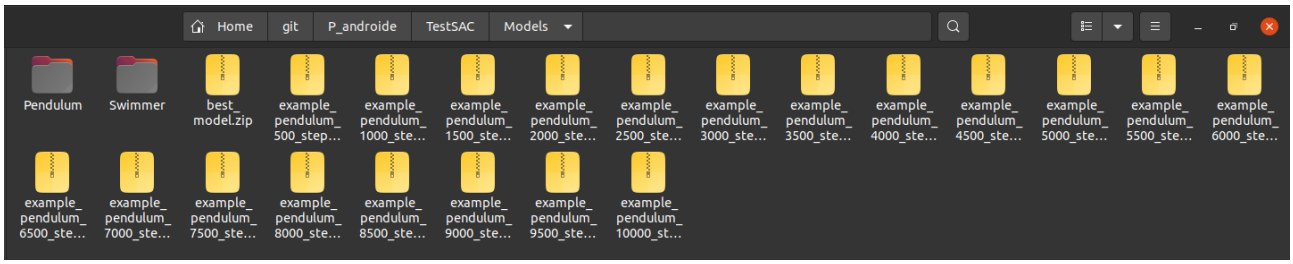
Firstly, one needs to train the model in the desired environment.

The *trainModel.py* file saves the learning process of SAC in the desired environment (*--env*) and with the desired name prefix (*--save_path, --name_prefix*).

It is also possible to change the algorithm's hyperparameters *tau, gamma, learning_rate*, and the type of *policy*.

For a training process of 10000 steps on *Pendulum* saved in the *Models* folder with a checkpoint every 500 steps :

```
python3 trainModel.py --env Pendulum-v0 --max_learn 10000 --save_freq 500 --save_path Models --name_prefix example_pendulum
```



You should end up with the training process saved as *.zip* files in the *Models* folder.

Using external politics

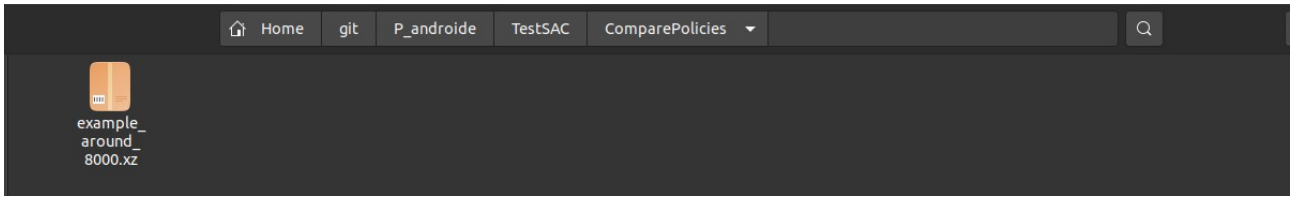
Before starting the *Vignette* algorithm, one should prepare if need be, the policies one would like to visualize as comparison on the final output.

The *preparePolicies.py* takes as input a list of policies and saves them in the right format. Those need only be saved inside of a *.zip* file, in *SB3*'s neural network format.

It takes as argument the input folder (*--input_folder*), the names of the policies separated by "; " (*--inputNames "policy1; policy2; policy3…"*) and the name of the output (*--outputFolder –outputName*).

To observe the relative location of the model at steps around the 8000th (whose *Vignette* we wish to compute) :

```
python3 preparePolicies.py --inputFolder Models –inputNames "example_pendulum_7000_steps; example_pendulum_7500_steps; example_pendulum_8500_steps; example_pendulum_9000_steps" --outputName "example_around_8000"
```



You should end up with the policies saved as *.xz* file in the desired folder.

Computing the Vignette

All the requirement for computing the *Vignette* are now met.

Vignette.py is the first step to compute a visualization of the learning space around an input policy.

It samples the policy's surroundings in its multi-dimensional space by computing the performance and the entropy along 2D lines cast from its position. Everything is then compiled into a *SavedVignette* object that will be used later.

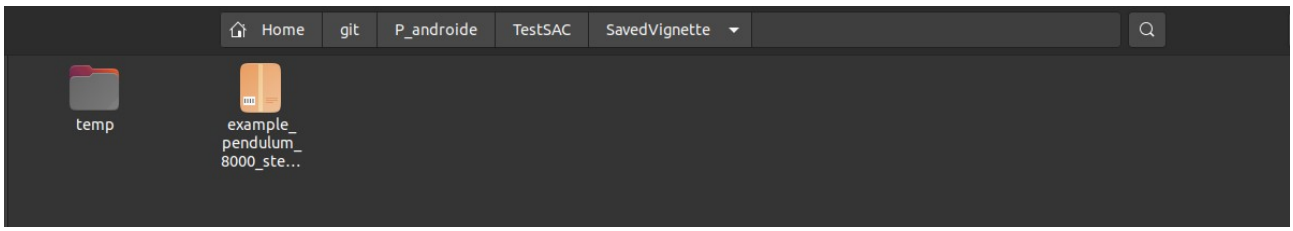
It takes as input the policy from where the sampling is done (*--inputFolder –inputName*), and several parameters regarding the *Vignette* : the number of directions sampled (*--nb_lines*), the number of evaluation per pixel (*--eval_maxiter)*, its range (*--minalpha –maxalpha*), its resolution (*--stepalpha*), and the path of the comparison policies computed previously (*--policiesPath*).

Please note that if the parameters provided (notably the range and the resolution) don't allow for the input policies to be included inside of the *Vignette*, they will be automatically changed.

One should also enter the desired output information (*--directoryFile –outputName*).

To compute the *Vignette* at the 8000th step, along with the desired input policies :

```
python3 Vignette.py --env Pendulum-v0 --inputFolder Models --inputName example_pendulum_8000_steps --eval_maxiter 1 --nb_lines 10 --policiesPath ComparePolicies/example_around_8000.xz
```



You should now have a *.xz* containing a *SavedVignette* object.

If anything fails during the saving process (not the computing), the output will be saved in the *SavedVignette/temp* folder.

Transform functions (work in progress)

When reading the results, one can wish to apply transformations to the output in order to emphasize certain things or improve readability.

We provide a tool to do just that.

The *transformFunction* contains a function and its parameters. It allows the user, once the Vignette has been computed, to transform the visualization with a function and to modify its parameters in real time.

An example of how to instantiate such an object can be found in the *transformFunction.py* file.

Please note that due to this feature being a work in progress, the latter only correctly works with one-arguments functions for now.

Visualize the results

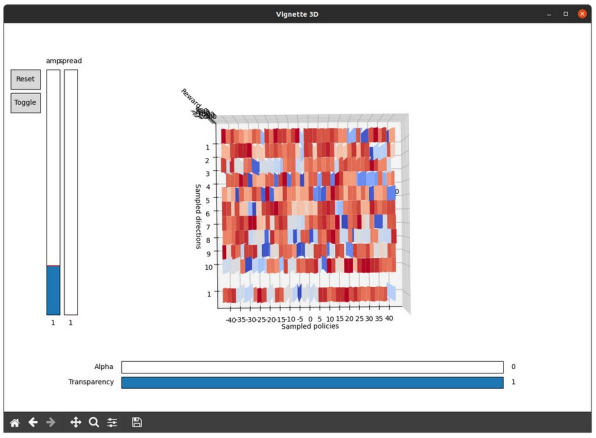
To visualize the computed *Vignette* one needs to run *SavedVignette*'s plotting functions.

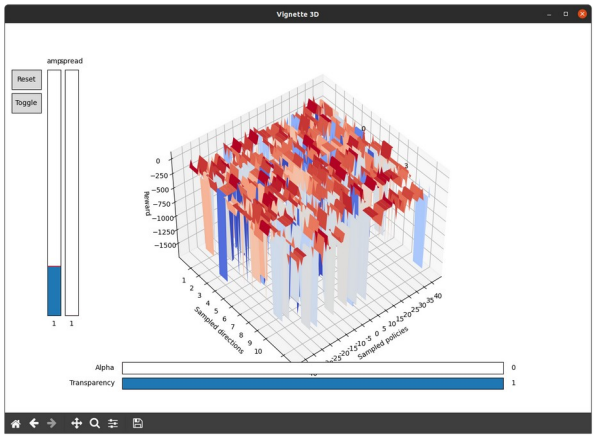
SavedVignette.show2D method can be called to compute and show the 2D *Vignette*. *SavedVignette.plot3D* and *SavedVignette.show3D* both need to be called in order to show the 3D *Vignette*.

An example of their use is provided in *savedVignette.py*'s main.

Therefore, the results can be visualized by running *savedVignette.py :*

```
python3 savedVignette.py --directory SavedVignette --filename example_pendulum_8000_steps
```

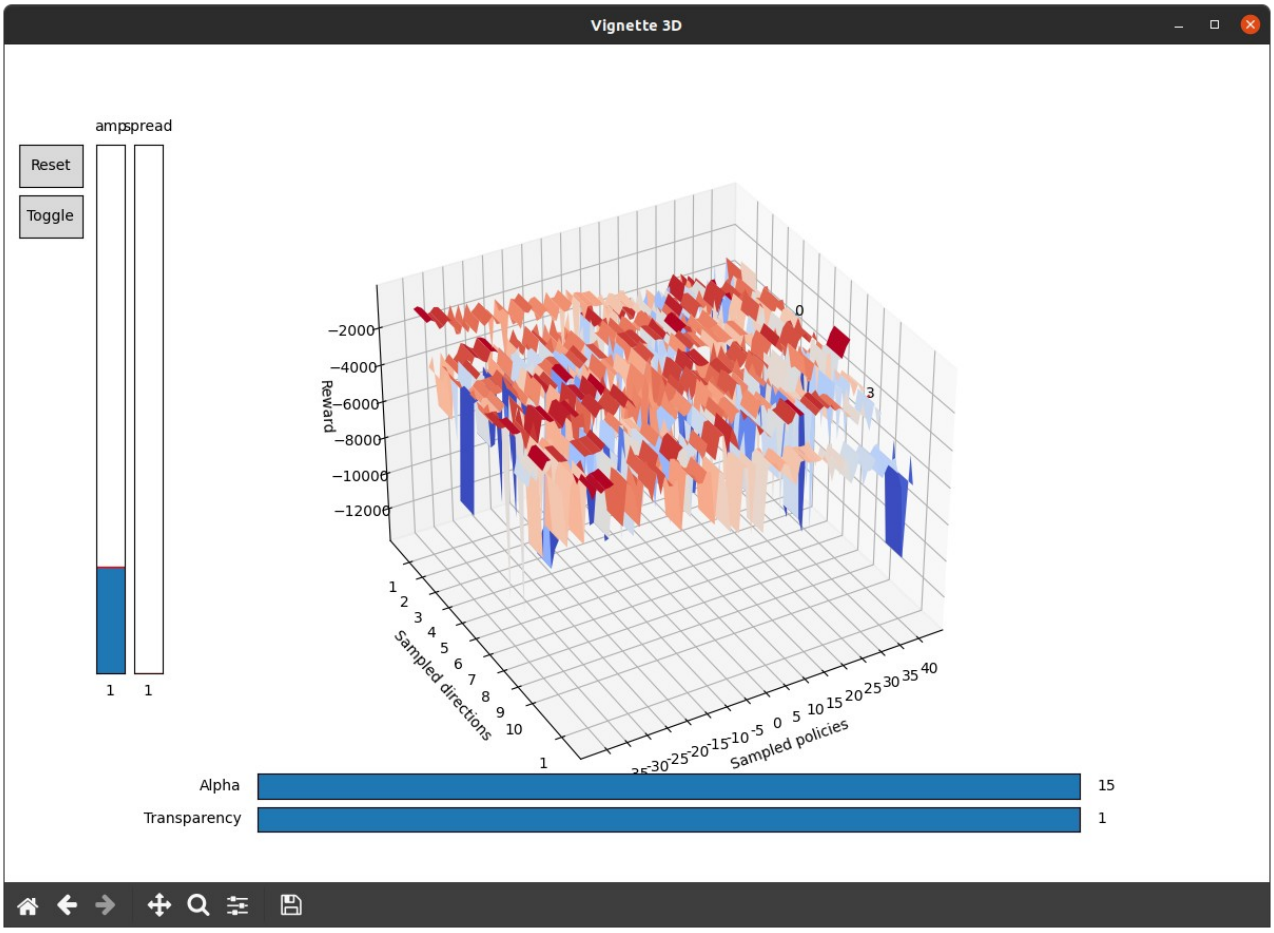




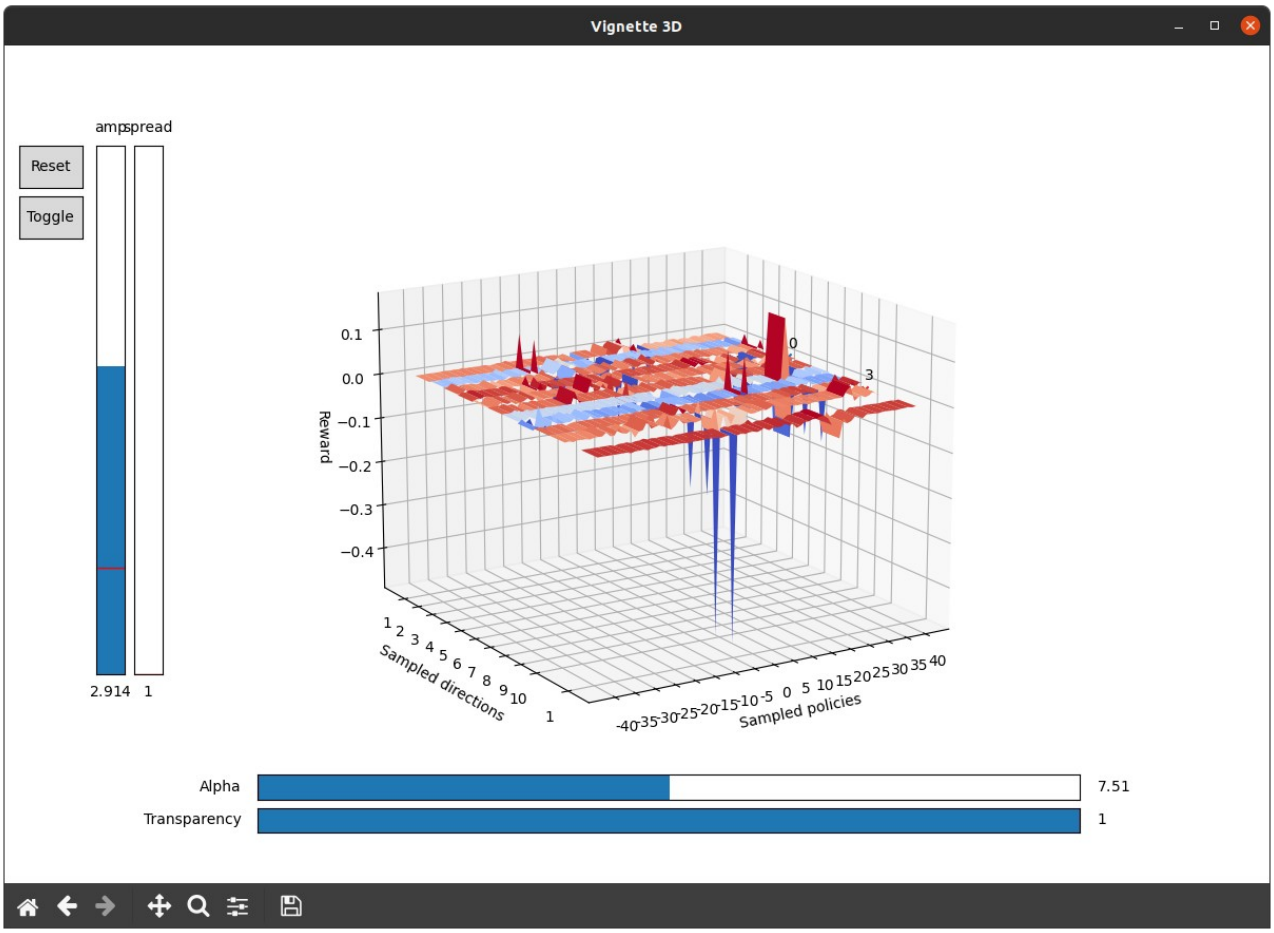
Once it has been run, apart from the *Vignette* itself, you should see several sliders and buttons :

- On the left hand side:
  - A *Reset* button
  - A *Toggle* button toggling the *transformFunction* on and off
  - Sliders allowing the user to change the *transformFunction*'s parameters, as many as there are arguments
- At the bottom:
  - *Alpha* slider, changes the amount of entropy
  - *Transparency* slider, changes the transparency of the surfaces for better readability

An interesting result easily observable from our tool is the smoothing of the learning space thanks to entropy regularization:



Pendulum being a rough learning space, it emphasizes the need for transform functions. *TransformFunction.isolate* is a function that allows to isolate the surroundings' extrema:



Note that the comparison policies are still there, you just need to decrease the opacity of the graph to be able to see them: