
Sorbonne Université

Projet ANDROIDE : Visualisation du Paysage de valeur

# Rapport de projet

*Encadrant :*
Olivier SIGAUD

*Étudiants :*
Yannis ELRHARBI-FLEURY
Sarah KERRICHE
Lydia AGUINI

# Introduction

L'apprentissage par renforcement consiste en une recherche heuristique, dans un environnement donné, d'une stratégie permettant de maximiser une récompense.

Dans certains environnement, cette approche est moins performante que des algorithmes évolutionnaires. De plus, il est démontré que certaines méthodes d'entraînement tirent leur efficacité de transformations de l'espace d'apprentissage.

Cette recherche étant propre à l'environnement, et par construction s'effectuant dans un espace en grande dimension, il est nécessaire de s'intéresser au développement d'outils permettant de mieux comprendre ce processus et d'expliquer ces phénomènes.

Lors de notre projet, nous avons travaillé à l'amélioration de deux outils de visualisation mis au point les années précédentes. Ils permettent d'obtenir des aperçus du paysage de valeur autour d'un agent.

Lors de nos travaux, nous avons complètement réécrit le code de ces outils pour le rendre plus modulable et plus clair pour l'utilisateur. De plus, nous avons ajouté de nouvelles fonctionnalités de visualisation.

Dans ce rapport nous rappelons le fonctionnement des outils développés, puis détaillons notre implémentation de ceux-ci en présentant les nouvelles fonctionnalités. Nous démontrons ensuite l'intérêt de ces derniers grâce à des exemples d'utilisation, pour enfin évoquer des idées d'améliorations futures à apporter aux outils.

# Table des Matières

# 1 Principe de fonctionnement des outils de visualisation

Le principe de fonctionnement des outils développés les années précédentes repose sur une méthode d'échantillonnage de l'espace d'apprentissage selon des droites.

Les outils développés permettent d'obtenir un aperçu en deux dimensions de l'espace d'apprentissage d'un modèle, alors en grande dimension (de la taille du réseau de neurones).

Le premier outil, *étude de gradient*, permet d'observer la trajectoire du modèle lors de son apprentissage. Le second, *Vignette*, permet d'observer le paysage de valeur autour du modèle.

La sortie de ces outils est constituée d'un ensemble de lignes de pixels. Chaque ligne est une direction tirée dans l'espace d'apprentissage, le pixel en son centre correspond au modèle à partir duquel elle est tirée.

La direction est ensuite échantillonnée à une fréquence entrée par l'utilisateur. La valeur des échantillons, quantifiée par une carte de couleur, correspond à la récompense obtenue en cette position.



Figure 1: Un exemple de ligne composant la sortie des outils, une droite discrétisée puis coloriée en fonction de la récompense obtenue

Nous présentons maintenant leur principe de fonctionnement, et le format de leur sortie.

## 1.1 Etude de gradient

Le premier outil, *étude de gradient*, permet de suivre la descente de gradient d'un modèle.

Il consiste à prendre comme lignes les directions prises par la descente de gradient à chaque pas. Pour avoir une idée du déplacement effectué entre deux pas, on indique la position relative des modèles sur les droites : une pastille rouge pour la position au pas précédent, une verte pour la position au pas actuel.

De plus, le produit scalaire entre deux directions est représenté sur le côté droit de la sortie. L'utilisateur y lit une image du changement d'angle du modèle lors de la descente de gradient.

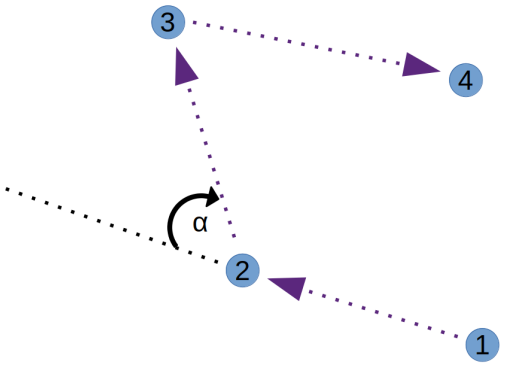


Figure 2: Descente de gradient en 2D, le modèle se déplace dans l'espace d'apprentissage en marquant un angle $\alpha$ entre deux pas.

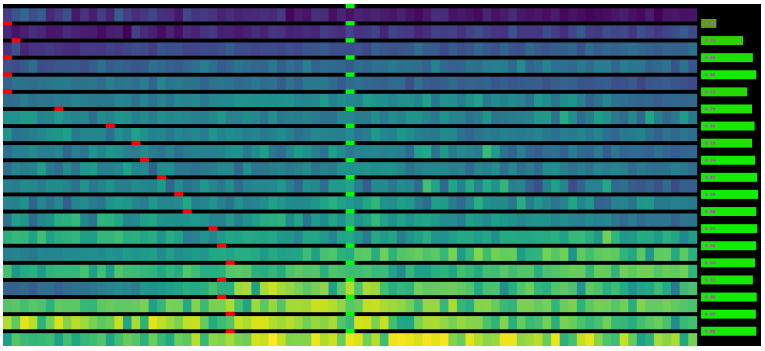

Figure 3: Un exemple de sortie de l'étude de gradient, algorithme SAC, environnement *Pendulum* entraînement enregistré tous les 500 pas de 500 à 10.000. La sortie se lit de haut en bas, plus la récompense est grande plus la couleur est claire. De haut en bas, on observe que le modèle se déplace de moins en moins rapidement vers une zone à forte récompense. De plus, l'angle entre chaque direction est faible car le produit scalaire, indiqué sur la droite, est proche de 1. Il pourrait être intéressant d'effectuer une étude de gradient autour des premiers pas, car on observe que le changement de direction est plus le important entre le pas 500 et le pas 1.000.

Ainsi, cet outil donne un aperçu en deux dimensions de la trajectoire prise par un modèle lors de son apprentissage (en n dimensions, n étant la taille du réseau de neurones).

## 1.2 Vignette

L'outil Vignette permet d'obtenir un aperçu des alentours d'un modèle.

Il consiste à échantillonner l'espace d'apprentissage grâce aux droites introduites précédemment. On obtient un aperçu de la boule centrée en un modèle en tirant aléatoirement à partir de celui-ci des droites partant dans des directions aléatoires.

Les directions tirées aléatoirement sont alors triées par ordre de proximité, dans le but de donner un meilleur aperçu des structures. Elles sont ensuite discrétisée à une certaine fréquence, ce qui permet de connaître la valeur de l'espace d'apprentissage le long de celles-ci.

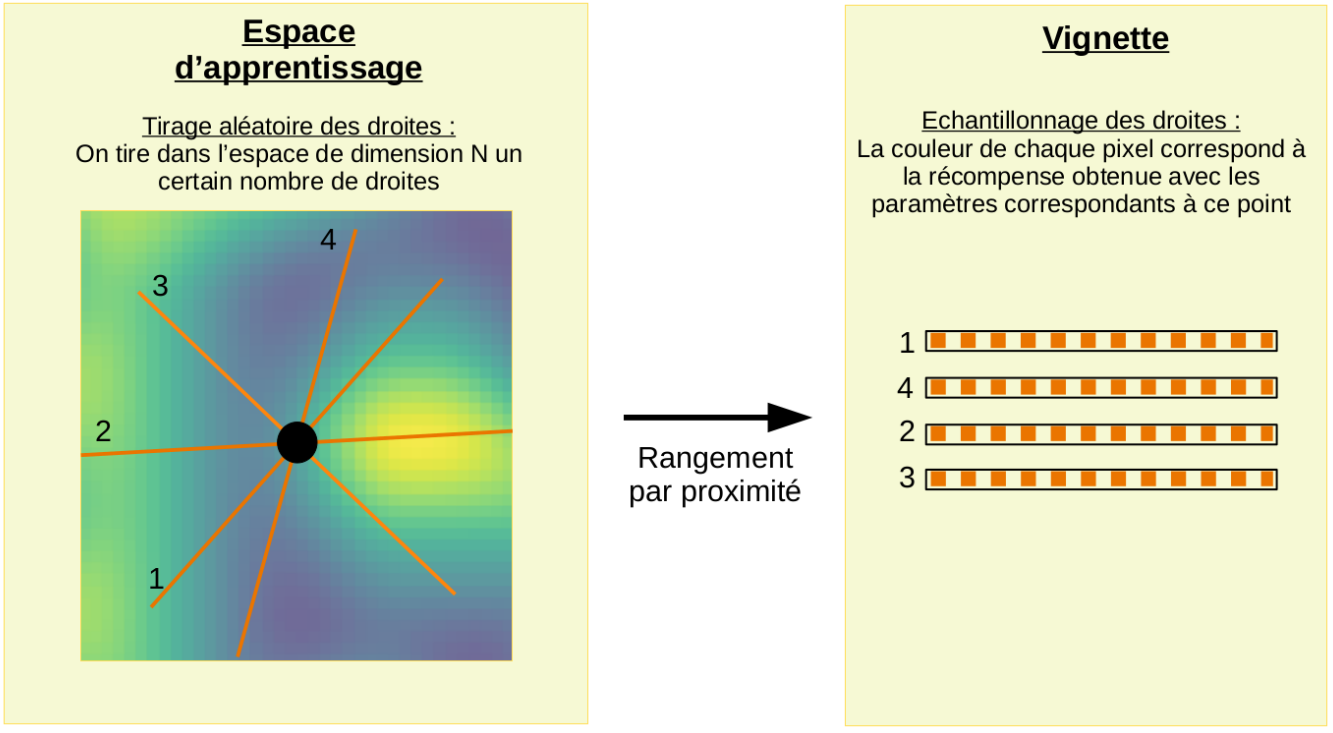


Figure 4: Tirage des lignes de Vignette dans des directions aléatoires, centrées en un modèle initial. Elles sont ensuite groupées par proximité et sont échantillonnées pour donner un aperçu de l'espace d'apprentissage.

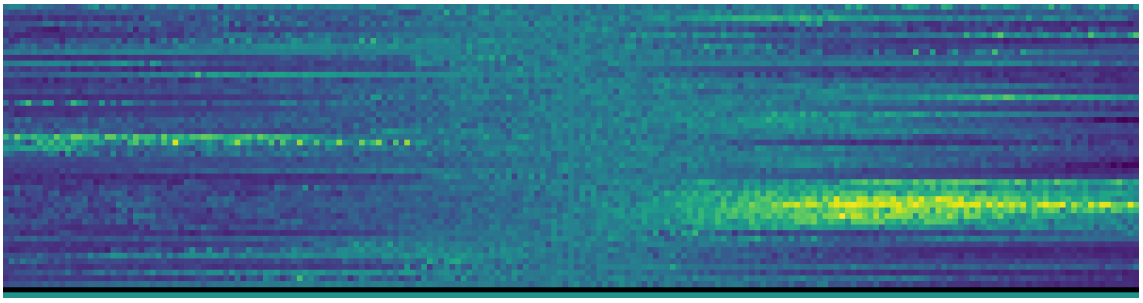

Figure 5: Un exemple de sortie de *Vignette*, algorithme SAC, environnement *Pendulum* entraîné pendant 5.000 pas, 50 directions tirées aléatoirement. La politique entrée est située au centre de la *Vignette*. Autour du modèle, on observe un environnement bruité, de moyenne récompense. De plus, certaines zones en bordure de la boule observée sont clairement à faible récompense, tandis que deux zones à proximité offrent une meilleur récompense. On en déduit que ce sont deux zones dans lesquelles la descente de gradient est susceptible de converger.

On obtient alors une représentation en 2D de l'espace d'apprentissage, qui était alors en grande dimension (de la taille du réseau de neurones). De plus, on observe bien la conservation des structures de l'espace d'apprentissage dans la *Vignette*, avec l'apparition d'ensembles de même récompense.

Notons que du fait du faible nombre de directions tirées par rapport à la dimension de l'espace et de la discretisation des droites, cette représentation n'est que partielle. Il est possible que la *Vignette* passe à côté de structures.

# 2 Nouvelles fonctionnalités

Lors de notre projet, nous avons fait le choix d'offrir à l'utilisateur toutes les fonctionnalités nécessaires pour effectuer une analyse complète du paysage de valeurs d'un environnement et d'un algorithme.

Ainsi, nous proposons un ensemble de fonctionnalités permettant d'aller de l'entraînement d'un modèle jusqu'à l'affichage des résultats.

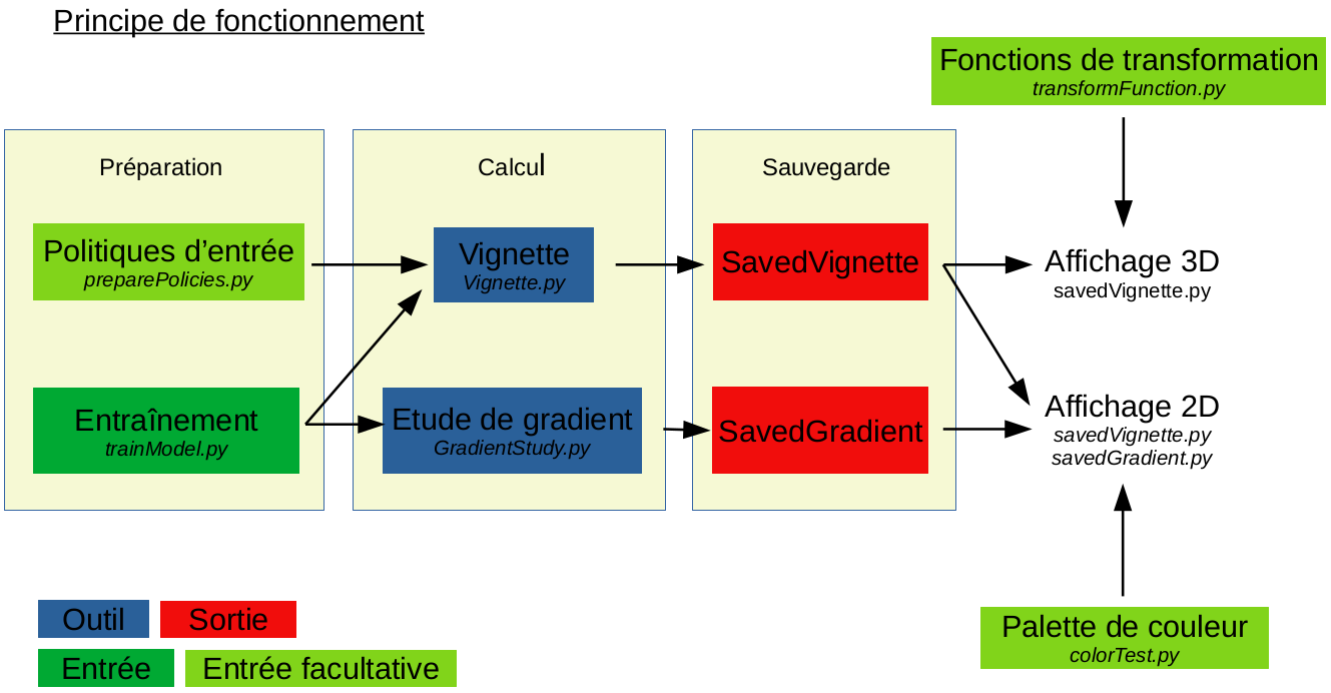


Figure 6: Processus d'utilisation des outils

L'utilisation des outils s'effectue en trois phases :

- préparation des entrées
- calcul de la *Vignette* ou de l'étude de gradient
- sauvegarde des paysages calculés
- manipulation des sauvegardes pour l'affichage

Dans cette partie, nous détaillons chacune de ces phases.

### Portage à *stable-baselines-3*

Lors de nos travaux, nous avons été contraints de réécrire le code des outils. En effet, celui-ci était écrit pour fonctionner sur un environnement particulier (*Mujoco Swimmer*) sous l'algorithme TD3.

Nous avons procédé à un portage vers la librairie *Stable-baselines-3*, comportant un ensemble fiable d'implémentations d'algorithmes d'apprentissage par renforcement en PyTorch. Le code de cette librairie est accessible sur github et celle-ci propose une documentation détaillée de ses implémentations.

Ainsi, pour chacun des outils, il est possible pour l'utilisateur de changer facilement l'algorithme d'apprentissage, ses hyper-paramètres et l'environnement utilisé.

## 2.1 Phase de préparation

Avant toutes choses, pour fonctionner, les outils ont besoin de recevoir en entrée un modèle entraîné sous forme de réseau de neurones.

Grâce à leur implémentation sous *stable-baselines-3 (SB3)*, les outils peuvent recevoir n'importe quel réseau de neurones au format PyTorch.

Nous proposons un exemple d'application de *SB3* dans le fichier *trainModel.py*. L'utilisateur peut alors entraîner un réseau de neurones sous l'environnement souhaité, en sauvegardant les étapes de l'apprentissage à un rythme choisi.

De plus, l'étude portant sur une descente de gradient, l'utilisateur peut fournir en entrée de *Vignette* une liste de politiques. Il peut alors observer la position relative de chacune des

politiques de la liste avec la politique centrale de la *Vignette*.

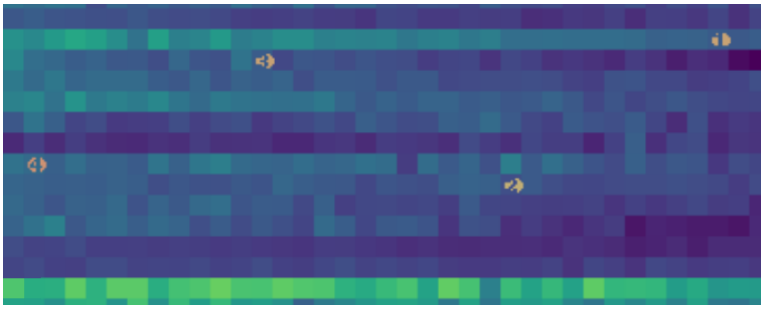

Figure 7: Extrait d'une *Vignette* affichant des politiques d'entrée

Nous détaillons comment ces politiques d'entrée sont prises en compte lors du calcul de *Vignette* dans la partie suivante.

## 2.2 Phase de calcul

*L'étude de gradient* prend en entrée un ensemble de politiques, correspondant à la progression de la descente de gradient pour le modèle entraîné. Comme décrit précédemment, il calcule un suivi de la descente de gradient effectuée par le modèle.

Pour *Vignette* la possibilité d'entrer une liste de politiques à situer dans la sortie rajoute des étapes de calculs. Leur prise en compte s'effectue en deux étapes.

La première étape consiste à ajuster la fréquence d'echantillonnage des droites.

En effet, on rappelle que l'utilisateur donne en argument de *Vignette* une fréquence d'échantillonnage. Cette fréquence correspond à la résolution de chaque ligne. Par conséquent, *Vignette* dispose d'une portée limitée. On ne peut observer qu'un aperçu de la boule ayant pour centre le modèle central, et un rayon résultant de la résolution choisie.

Il est possible que des politiques d'entrée soient en dehors de cette boule.

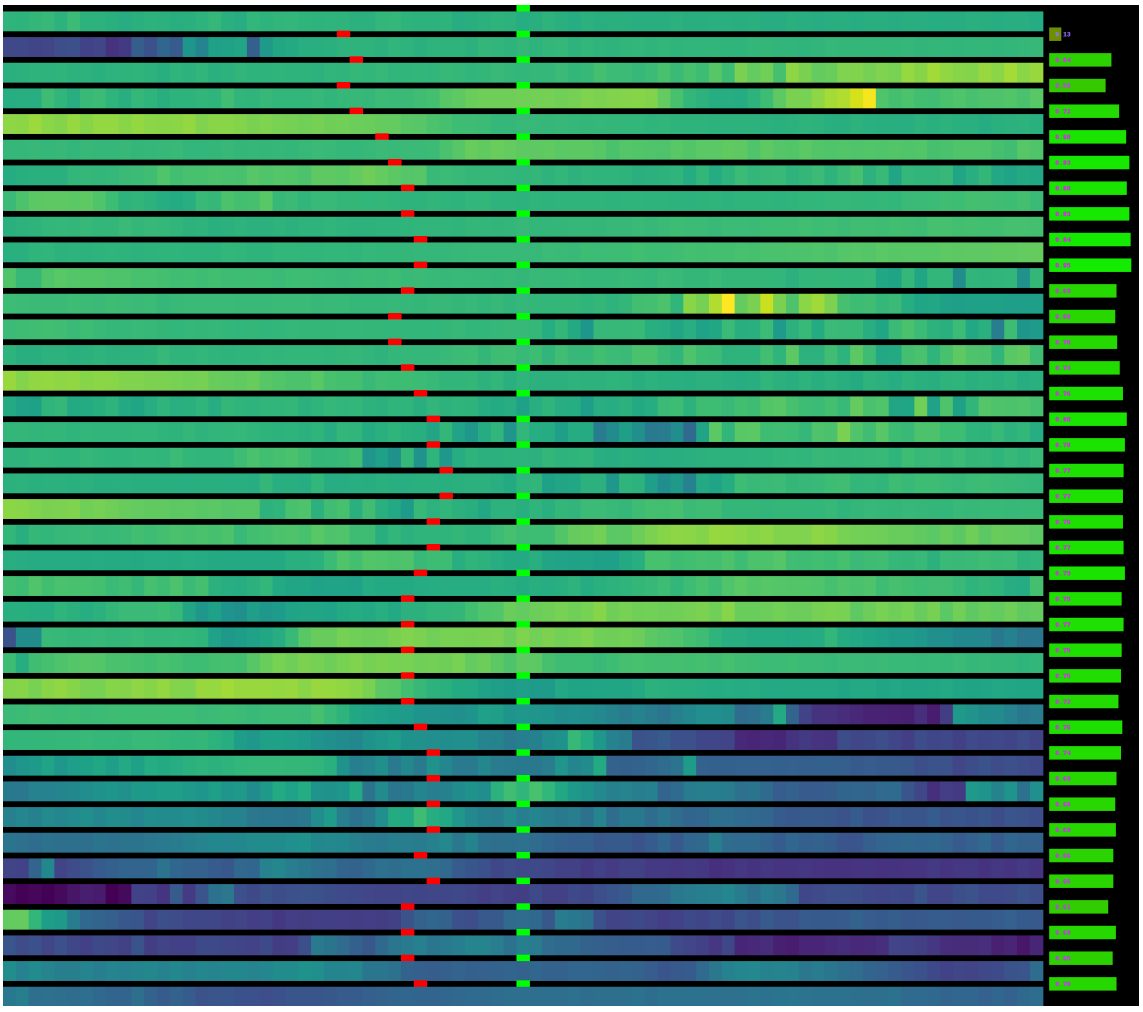

Figure 8: Etude de gradient sur Swimmer 250 pas à 10.000 pas sauvegardé tous les 250 pas, algorithme SAC. On note tout d'abord que la normalisation des couleurs s'effectue sur les valeurs extrêmes observées, ce point sera abordé dans la section Phase d'affichage. On remarque que le modèle se déplace en tatonnant dans une zone uniforme en suivant globalement la même direction (faible angle entre chaque ligne), il finit même par réduire sa récompense. On en déduit que l'initialisation de Swimmer s'effectue dans une zone à faible récompense, plutôt uniforme, ce qui empêche la descente de gradient de converger.

Nous avons donc fait le choix d'imposer une baisse automatique de la fréquence d'échantillonnage de façon à atteindre toutes les politiques.

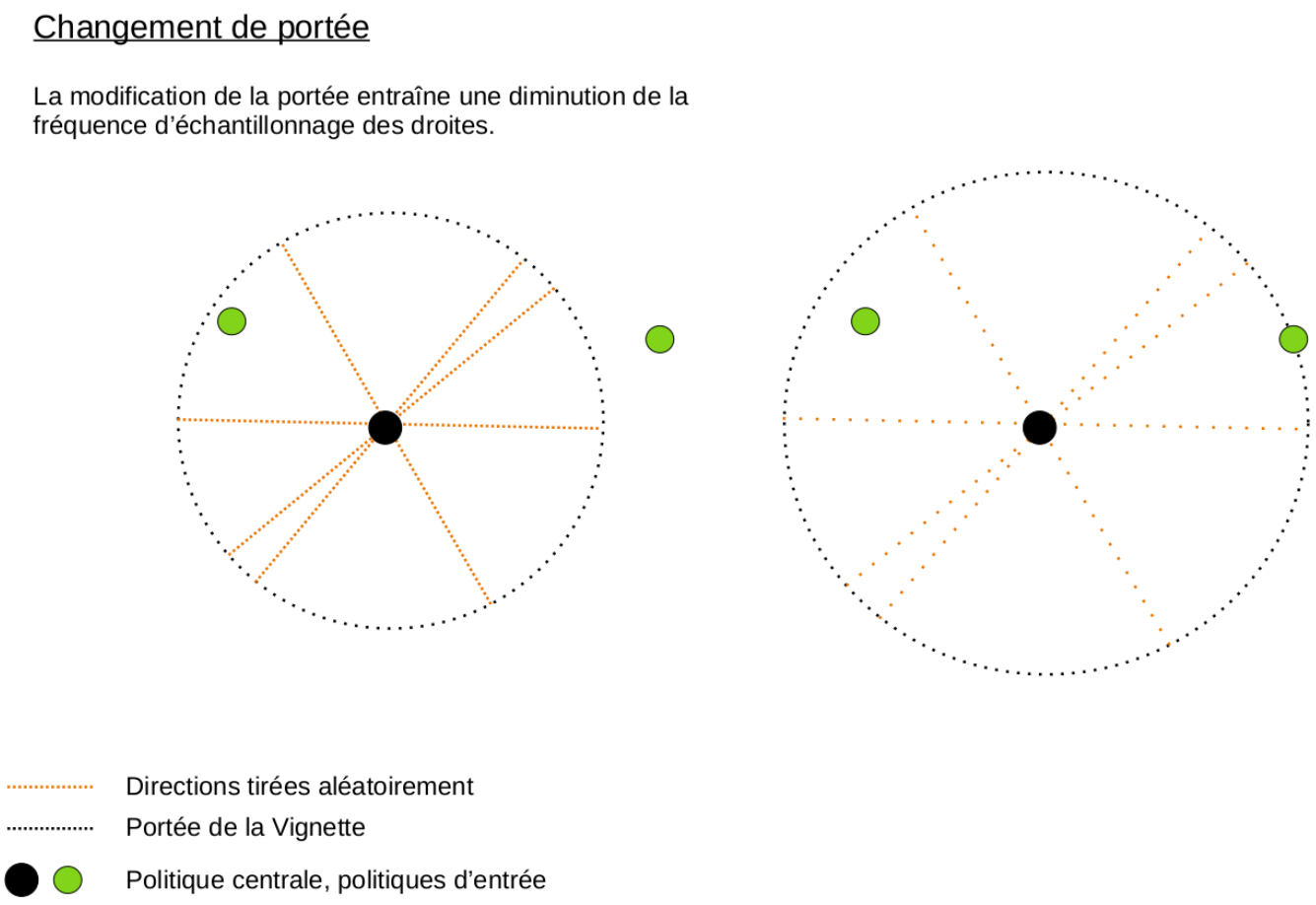


Figure 9: Première étape : ajustement de la portée

La seconde étape consiste à insérer les directions passant par les politiques d'entrée dans le résultat final.

Le nombre de lignes tirées (la hauteur de la Vignette) étant un paramètre de l'utilisateur, il convient de remplacer certaines de ces lignes (donc directions) par celles correspondant aux politiques d'entrée.

Pour que l'introduction de ces politiques bouleverse le moins possible la Vignette d'origine, on insère les directions correspondantes à la place des directions qui en étaient les plus proches.

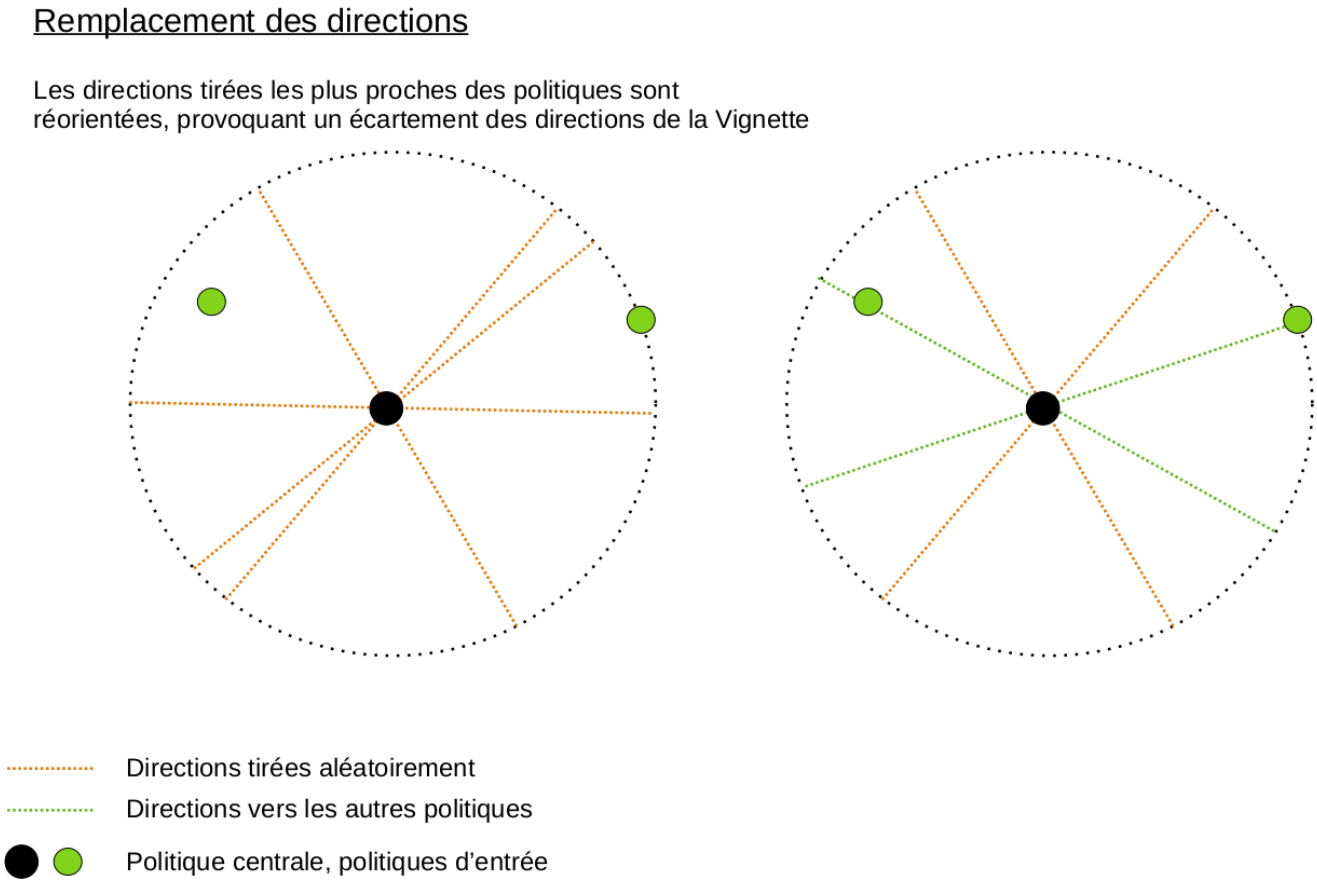


Figure 10: Seconde étape : insertion des directions

On constate dans l'exemple de la figure 10 que cette étape a tendance à écarter les directions les unes des autres.

Cela a pour inconvénient de provoquer des discontinuités dans la détection de structures. Au contraire, l'utilisateur pourrait préférer concentrer les directions dans des faisceaux, permettant un meilleur niveau de détail autour de directions particulières. C'est un point abordé dans la partie Future works du rapport.

Une solution pourrait être de, au lieu de choisir la direction la plus proche, prendre la direction la plus isolée et la remplacer par la direction vers la politique.

C'est un problème flagrant dans notre représentation 2D simplifiée de l'espace d'apprentissage. Mais en réalité, le nombre de directions tirées est bien inférieur à la dimension de l'espace, ce qui réduit l'importance du problème.

## 2.3 Phase de sauvegarde

Une fois les calculs effectués, les données sont sauvegardées dans un objet de type *SavedVignette* ou *SavedGradient*. Chacun de ces objets garde en mémoire les directions ayant servis aux calculs, la valeur des récompenses pour chaque pixel.

Cette approche permet un traitement ultérieur des données si l'utilisateur le souhaite, cependant celles-ci prennent beaucoup de place en mémoire.

Nous utilisons une compression au format *.xz* (algorithmes *LZMA/LZMA2*). Bien que ce format soit relativement lent pour la compression, il est très rapide en décompression.

Cela le rend idéal pour notre application : la lenteur de compression est négligeable par rapport au temps de calcul des outils (quelques secondes contre quelques heures), mais la rapidité de décompression est parfaite car l'utilisateur sera amené à fréquemment charger les résultats (pour faire des essais de modification des attributs par exemple).

La taille d'un fichier sauvegardé est de l'ordre de 100Mb.

## 2.4 Phase d'affichage

Tout comme dans la version des années précédentes, nous offrons la possibilité d'afficher les résultats en tant qu'image 2D. Grâce à la fonctionnalité de sauvegarde nous avons pu implémenter une gestion des palettes de couleur.

Nous avons constaté que cette gestion des couleurs était utile. En effet elle permet aux personnes malvoyantes de régler le contraste des couleurs. De plus, certains écrans ont du mal à distinguer les faibles variations d'intensité de couleur entre les pixels.

L'utilisateur peut créer ses propres palettes (grâce au fichier *colorTest.py*), ou utiliser des palettes de couleurs préfaites dans *matplotlib*.

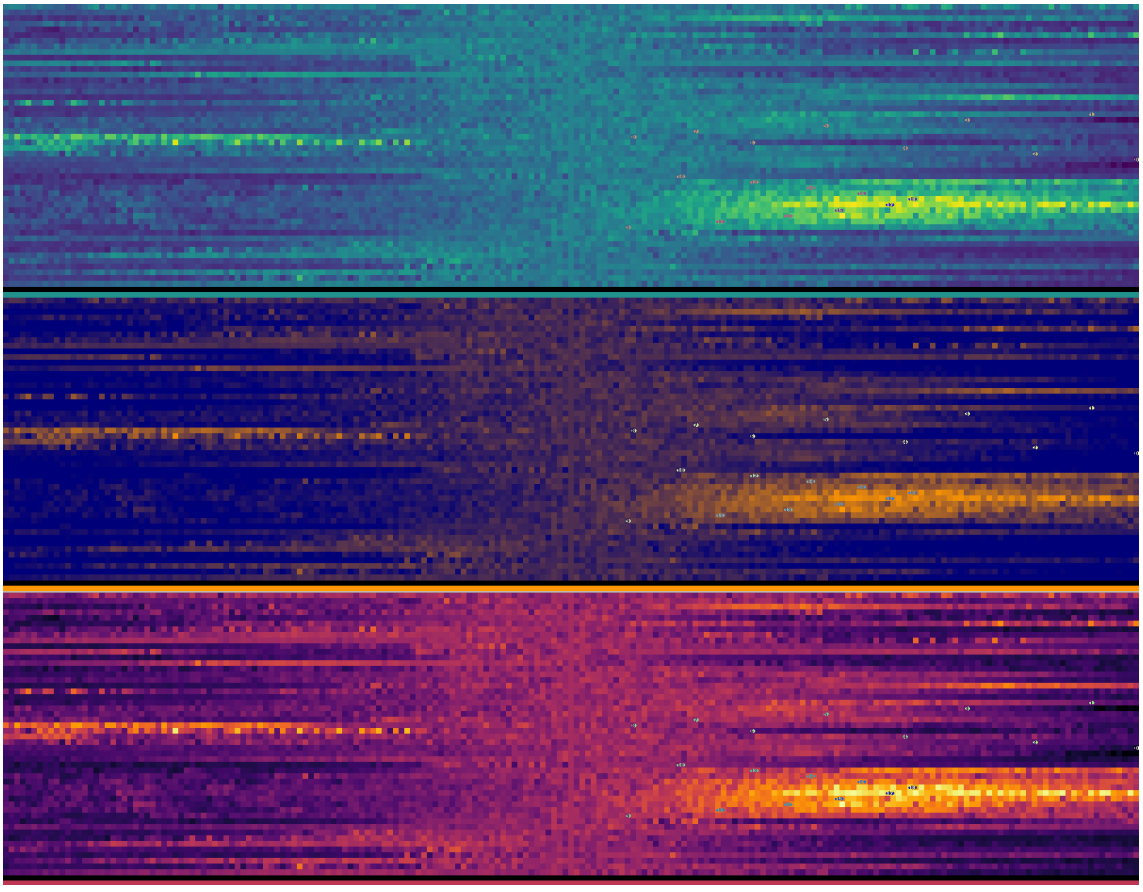

Figure 11: Différentes palettes de couleur sur la même *Vignette*.

La majeure partie de nos travaux a consisté à développer une visualisation en 3D de l'outil Vignette. Dans cette visualisation, chaque pixel prend pour hauteur la récompense obtenue.

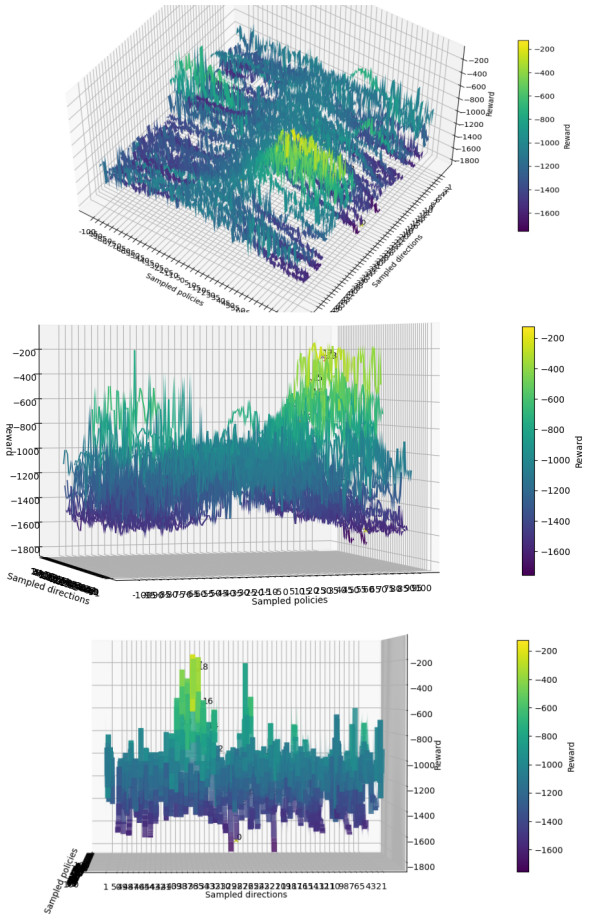


Figure 12: Vignette 3D, algorithme SAC, environnement *Pendulum* entraîné pendant 5.000 pas, 50 directions tirées aléatoirement. Il est plus facile d'apréhender les structures présentes dans le paysage de valeur. Le caractère bruité du paysage est flagrant.

Cette visualisation contient bien les mêmes informations que la version en 2D, cependant elle permet une approche plus intuitive de la structure de l'espace échantillonné.

On remarque qu'en vue de dessus on retrouve bien la Vignette en 2D.

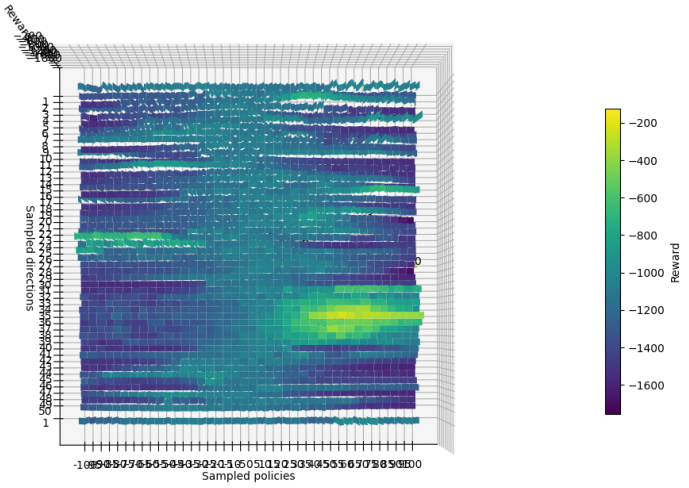


Figure 13: Vue de dessus de la Vignette 3D, on retrouve la Vignette 2D

Nous avons ajouté un curseur permettant de changer l'opacité des surfaces par soucis de visibilité. On peut alors plus facilement lire la position des politiques d'entrée de Vignette.

La Vignette de la figure 14 correspond à un entraînement de Pendulum sur 10.000 pas de temps enregistré tous les 500 pas. La politique centrale est le pas de temps n°5000, est les politiques d'entrée correspondent aux 19 autres enregistrements du n°500 au n°10000.

Dans l'image vue de côté, on observe clairement le processus de montée de gradient : *Pendulum* se déplace vers des zones de plus en plus hautes (amélioration de la récompense). Cela nous donne un argument pour confirmer le bon fonctionnement de notre réécriture de *Vignette* et du portage en 3D.

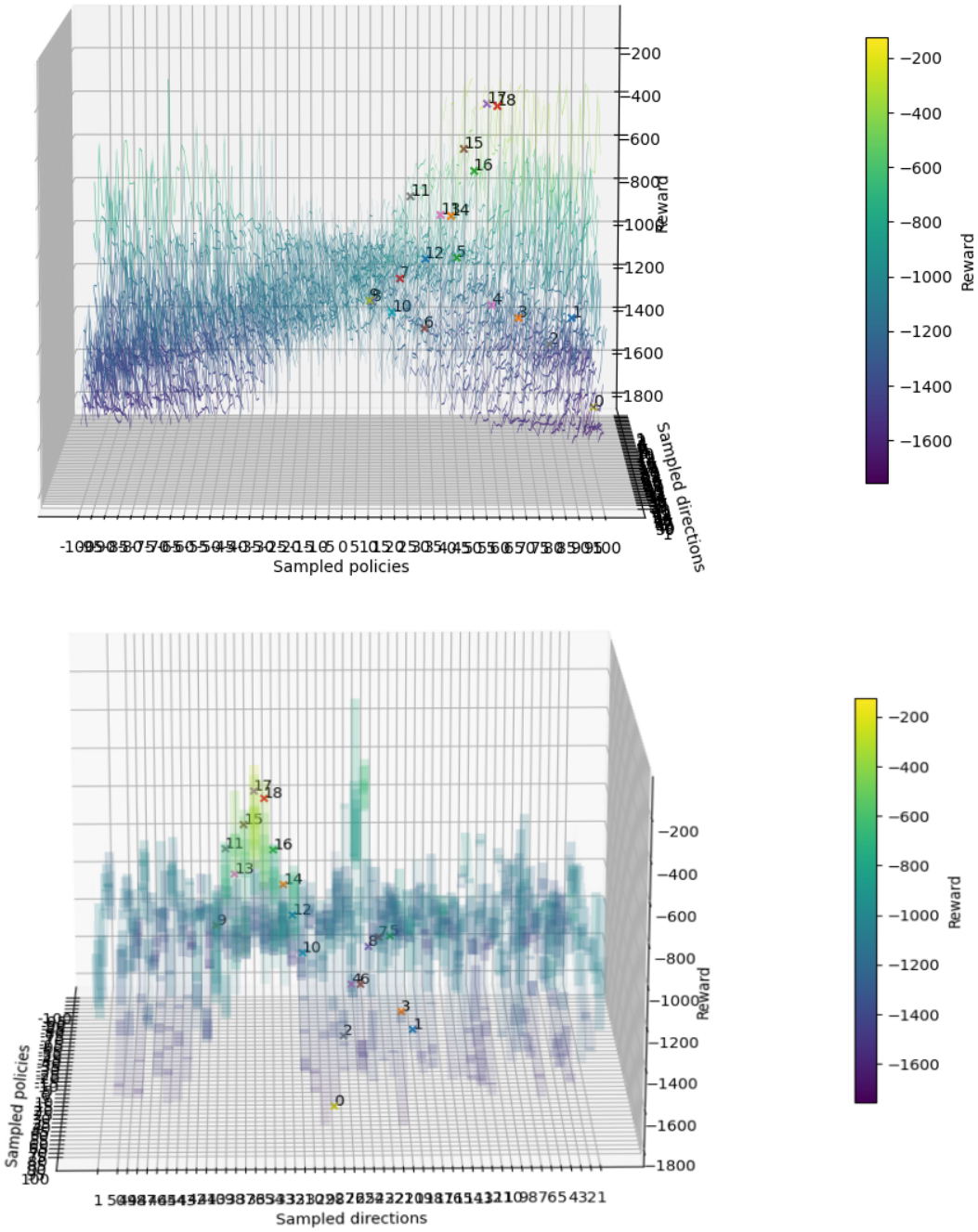


Figure 14: Vue de côté de la Vignette3D. Les politiques en entrée de la *correspondent* aux sauvegardes effectuées tous les 500 pas. On observe le processus de descente de gradient : le déplacement progressif du modèle vers une zone à forte récompense.

### 2.5 Accessibilité

D'autres améliorations ont été apportées, notamment l'ajout de barres de progression pour l'utilisateur. Il a désormais une idée plus précise de la quantité de temps restant pour les calculs.

De plus, le code a été clarifié et largement commenté. En utilisant le mode d'emploi, et en s'inspirant de fichiers *.sh* contenant des instructions de base, l'utilisateur peut désormais rapidement prendre en main les outils.

Dans chaque fichier gérant les entrées et les sauvegardes des outils, nous avons laissé à la disposition de l'utilisateur des instructions types.

## 3 Exemples d'utilisation des outils

Les outils sont à un stade de développement assez avancé pour être utilisés dans d'autres projets, on rappelle que nous avons mis à disposition un mode d'emploi détaillant leur utilisation pas à pas.

### 3.1 Projets utilisant Vignette

Lors de nos travaux, nous avons bénéficié des retours d'un autre groupe P-ANDROIDE travaillant avec nos outils.

Le groupe d'Hector Kohler et Damien Legros cherche à comprendre les différences de résultats donnés par deux méthodes de recherche de politique sur l'environnement *Pendulum*.

La première méthode étudiée est la *Cross entropy method*, la seconde est une méthode de descente de gradient *Policy gradient*.

Après avoir remarqué que la distance entre politique successives est grande dans le cas de *CEM* et petite dans l'autre cas, ils ont decidé d'utiliser *Vignette* pour directement observer l'espace des politiques. Ils l'ont intégré à leur code, et cela leur a permis de constater que l'initialisation des réseaux de neurones était située dans une zone plate en terme de récompense.

Ainsi, ils expliquent la différence de performance entre les deux méthodes par la capacité de *CEM* à rapidement sortir de cette zone pour découvrir de bonnes politiques.

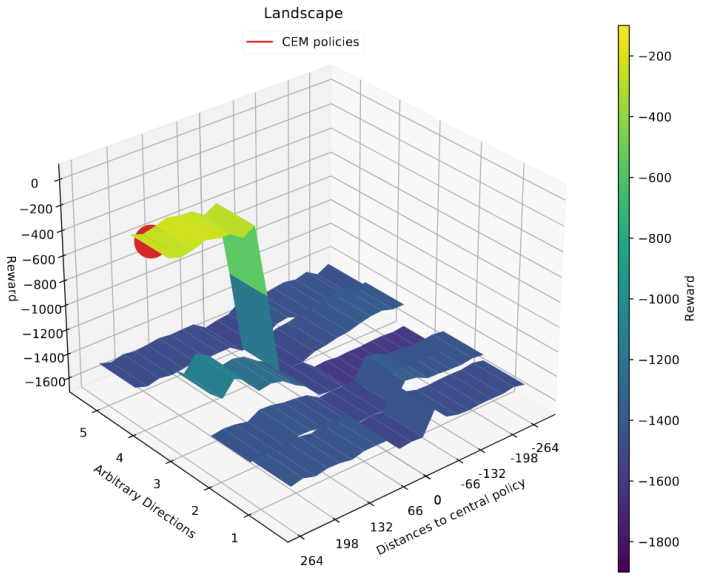


Figure 15: Exemple de *Vignette* utilisée par le groupe de Hector Kohler et Damien Legros. *Vignette* autour du point d'initialisation du réseau de neurones. On observe que la politique initiale est située dans une zone uniformément faible en terme de récompense, alors que *CEM* en est éloignée dans une zone de haute récompense.

### 3.2 Régularisation de l'entropie

## 4 Future works

Dans cette partie, nous détaillons des idées pour aller plus loin dans le développement des outils de visualisation du paysage de valeurs.

### 4.1 Méthode des faisceaux

L'outil *Vignette* permet à l'utilisateur d'obtenir un aperçu global de l'espace d'apprentissage autour d'une politique. *Etude de gradient* lui permet d'observer un aperçu du chemin prit par celle-ci lors de la descente de gradient.

Cependant, plus les sauvegardes du processus d'apprentissage sont espacée dans le temps, moins les visualisations proposées sont pertinentes. En effet, les outils ne donnent qu'un aperçu partiel de l'environnement : plus le nombre de pas entre deux politiques est grand, moins ceux-ci nous informent sur l'environnement dans lequel la descente de gradient a été effectuée.

Dès lors, il apparaît nécessaire de développer un nouvel outil permettant de comparer la direction globale prise par le modèle entre deux sauvegardes avec le paysage de valeur autour de la direction.

L'idée est de reprendre le principe de *Vignette* en focalisant les directions tirées aléatoirement dans le sens de déplacement de la descente de gradient.

Cela revient à échantillonner un faisceau reliant un agent à un autre en trois étapes :

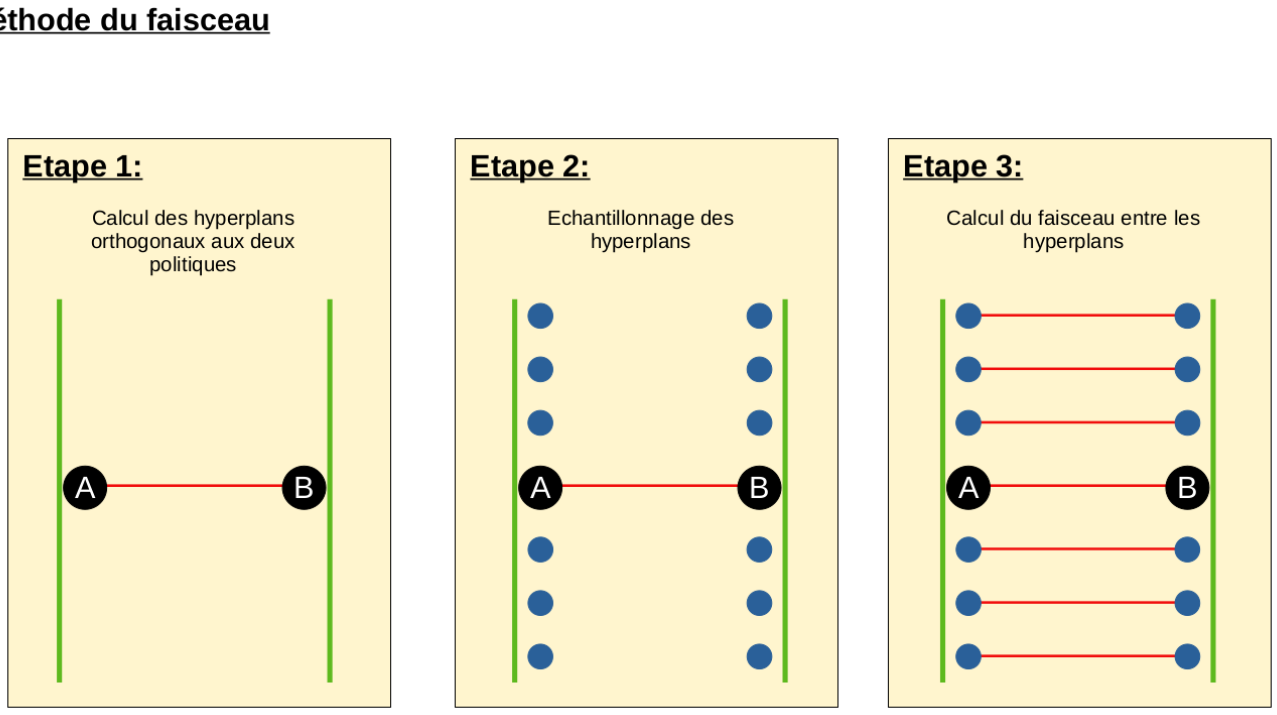


Figure 16: Etapes de la méthode du faisceau

Dans un premier temps, on calcule la doite (direction globale) prise entre deux politiques A et B. On cherche ensuite les deux hyperplans orthogonaux à celle-ci, centrés en ces points.

Ensuite, on échantillonne ces hyperplans de manière uniforme pour obtenir un ensemble de politiques semblables à A et semblables à B.

Enfin, à chacune de ces politiques on en fait correspondre une autre sur l'autre hyperplan. On calcule les droites les reliant pour les échantillonner sur le modèle de *Vignette* et *étude de gradient* (voir explications).

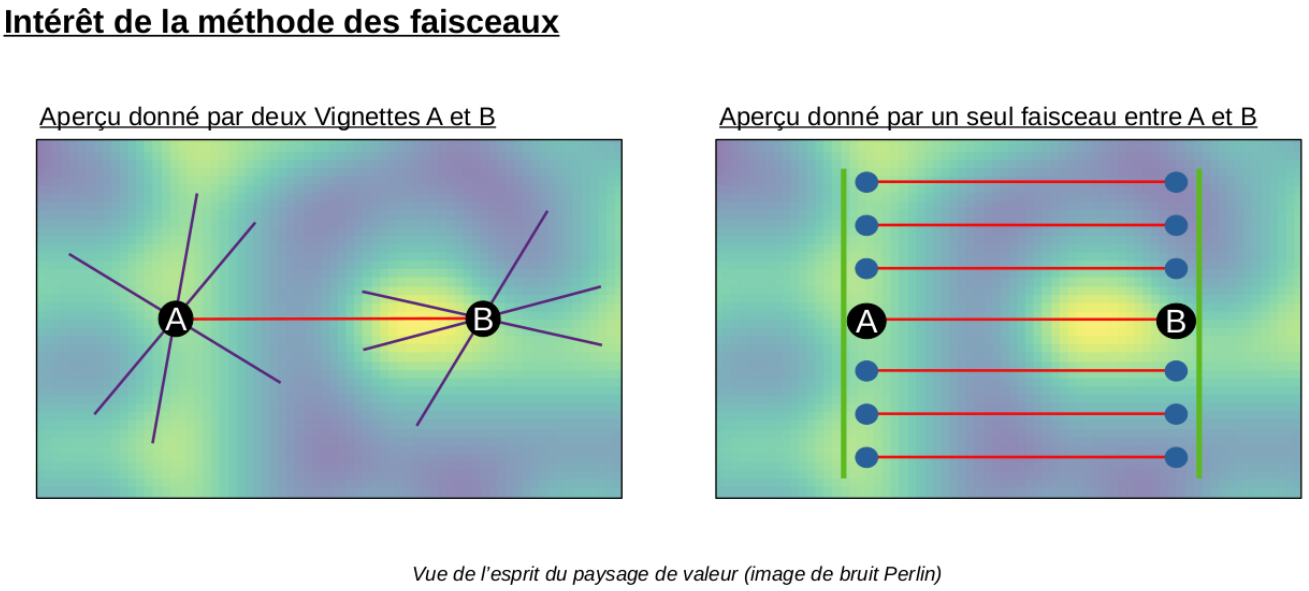


Figure 17: Intérêt de la méthode du faisceau : focaliser l'étude du paysage de valeur dans le sens de la descente de gradient

On constate l'intérêt que pourrait avoir un tel outil : combiner le principe de *Vignette* et de l'*étude de gradient* pour mieux détecter les structures rencontrées lors de la descente de gradient.

## 4.2 Fonctionnalités de "qualité de vie"

A cause du principe de généralisation, les performances des politiques ne sont pas constantes. Il est donc nécessaire de les calculer plusieurs fois pour obtenir la performance moyenne. Cela cause un problème de complexité pour le calcul de nos outils.

En effet, chaque pixel correspond à une politique différente. L'utilisateur est donc amené à faire un compromis entre la précision souhaitée et le temps de calcul disponible.

Nous proposons comme développement futur, la possibilité de calculer les outils en plusieurs passes, à différents niveaux de précision. Les résultats seraient calculés pour un faible nombre d'évaluations (donc une faible précision), permettant à l'utilisateur d'observer un aperçu des sorties pendant leur calcul.

De plus, nous proposons d'implémenter dans les objets *SavedVignette* et *SavedGradient* des méthodes permettant de segmenter l'exécution. L'utilisateur pourrait alors interrompre les calculs pour les reprendre plus tard.

Enfin, les échantillonnages des directions tirées dans *Vignette* ou les échantillonnages des directions entre chaque pas de *étude de gradient* étant indépendants entre eux, il est possible de les calculer sur plusieurs coeurs. Nous pensons que cela pourrait améliorer la vitesse d'exécution des outils.

# Conclusion

Lors de ce projet, nous avons enrichi les outils développés les autres années. Après les avoir porté à la librairie *stable-baselines-3*, nous les avons rendu plus faciles d'utilisation en reprenant le code de zéro.

Il est désormais plus facile pour un utilisateur de comprendre leur principe de fonctionnement et leur architecture, notamment à l'aide du cahier des charges (*user-manual*), de ce rapport et des nombreux commentaires détaillant l'exécution du code.

Grâce à leur implémentation sous *SB3* et à l'approche objet lors de leur développement, ceux-ci sont aisément modulables et utilisables dans le cadre d'autres projets.

Ainsi, de futurs utilisateurs peuvent ajouter de nouvelles fonctionnalités sans avoir à modifier la structure des données.

Enfin, ils pourront s'inspirer de notre code pour développer de nouveaux outils de visualisation 2D ou 3D d'un espace de grande dimension, tels que la méthode des faisceaux introduite précédemment.