

Conduite autonome 2D

2023/2024

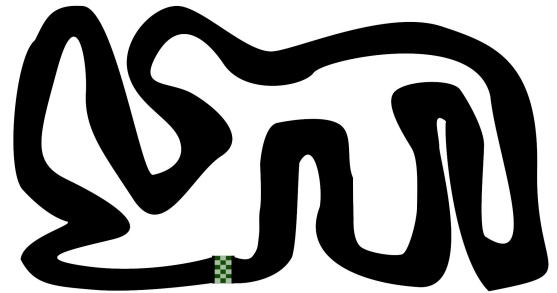
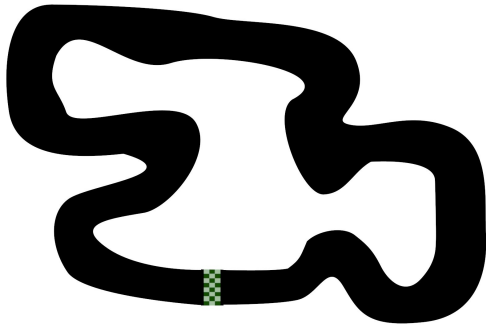
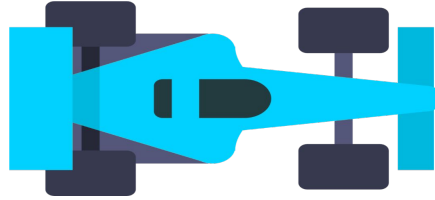
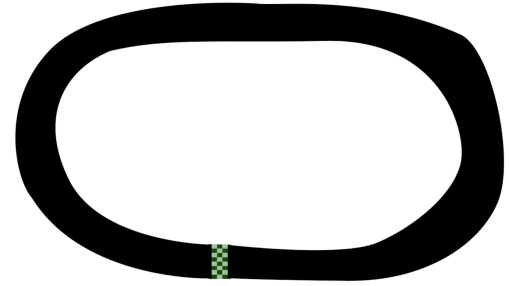
Cardot Clément
Guilbaud Maxime
Guais Clément

Sommaire

- Explication de la problématique
- Données entrées
- Algorithme NEAT
- Migration Stable-baselines
- Algorithmes A2C & PPO
- Multi-processing
- Mise en place des secteurs
- Optimisation récompenses
- Résultats
- Conclusion / ouverture

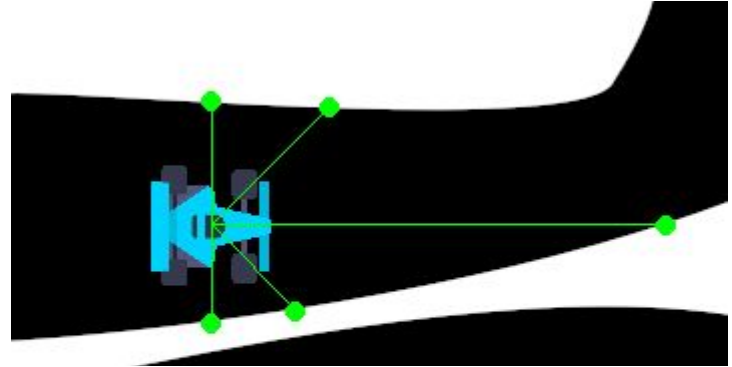
Explication de la problématique

- Des circuits
- Une voiture



Données entrées

- 5 capteurs :
 - distance par rapport au centre de la voiture



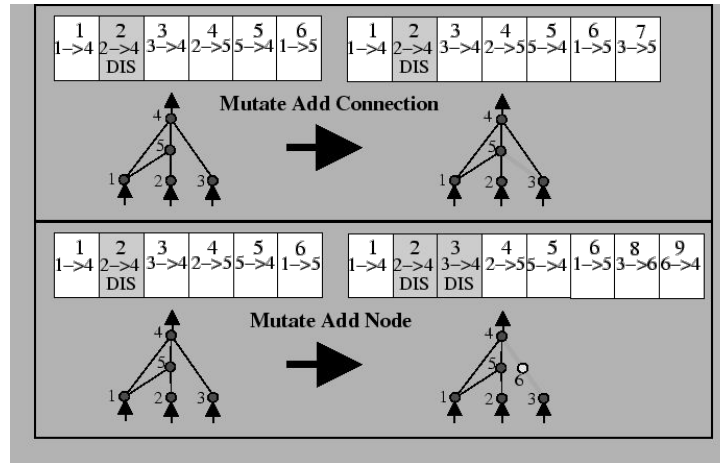
- 1er constat :
 - La complexité de la carte d'entraînement va influencer la qualité des données d'entrées

Algorithme NEAT (NeuroEvolution of Augmenting Topologies)

Principe : Algorithme génétique qui fait évoluer la structure même du réseau de neurones avec une complexité graduelle

Paramétrage :

- population
- nombre de générations



Test:

- Programme paramétré
- test des modèle entraînés

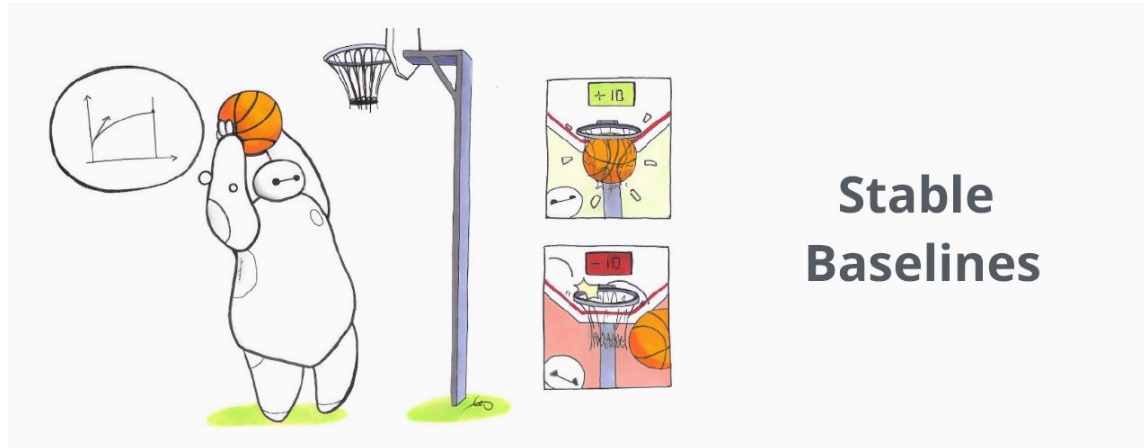
Migration Stable-baselines

Objectif :

- Challenger l'algo NEAT
- Comparer différend algo pour utiliser le meilleur

Pourquoi Stable-baselines :

- Documentation complète
- facile à prendre en main
- Plusieurs algorithmes interchangeables



Algorithmes A2C & PPO

A2C (Advantage Actor-Critic):

- Actor : prend des décisions
- Critic : évalue la valeurs des actions et calcule l'avantage
- Advantage : change la politique de l'acteur

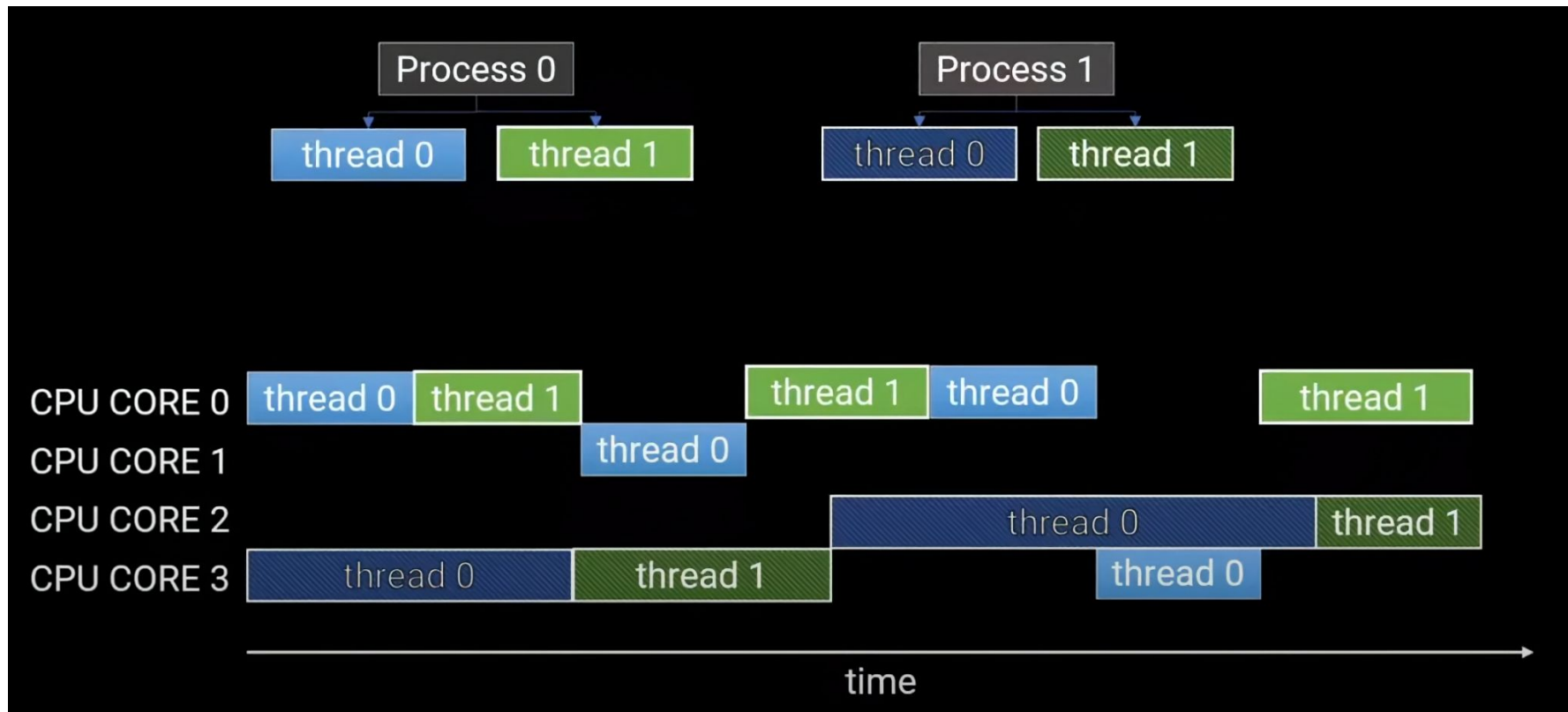
PPO (Proximal Politic Optimization):

- Ratio de probabilité : changement d'une politique à l'autre
- Fonction objectif clippée : fonction utilisé pour changer la politique des acteurs
- mise à jour en lot : évite des changement trop drastique

Différence de stabilité:

- PPO met en place une contrainte sur la modification de la politique afin de garantir la stabilité de l'apprentissage.

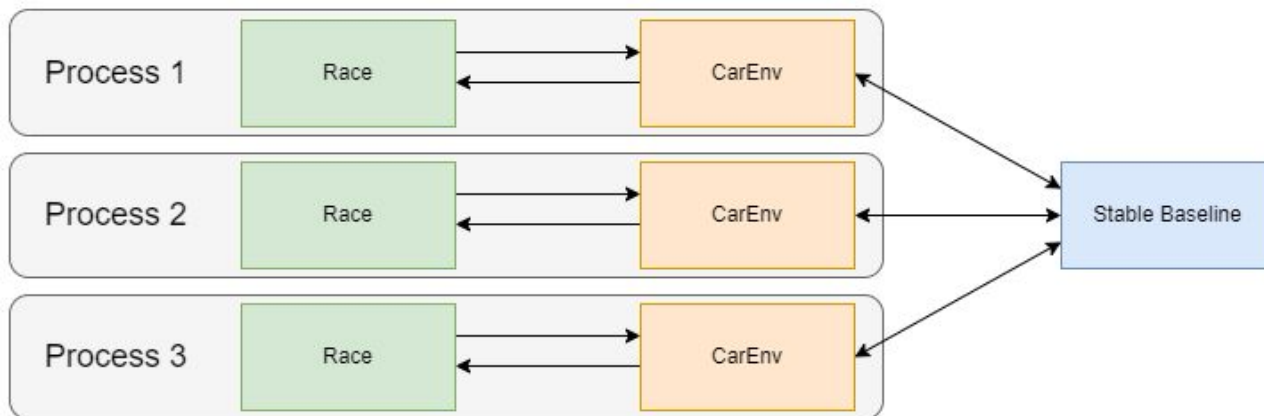
Multi-Processing - Courte introduction



Multi-Processing - Implémentation

Stable Baselines permet le multiprocessing

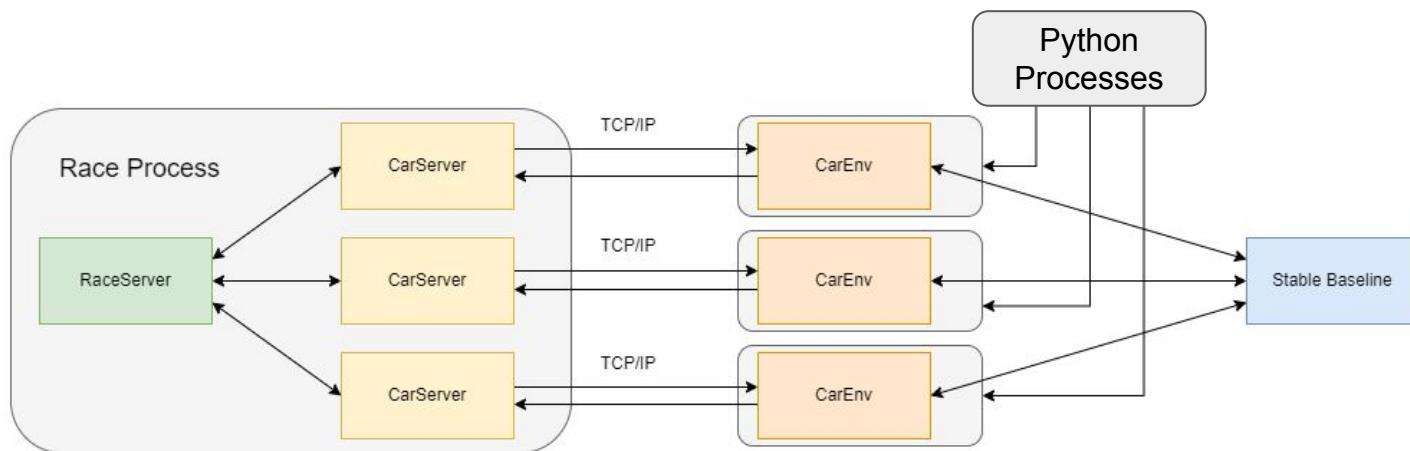
Mais chaque process obtient alors un environnement dédié (un circuit par voiture)



Multi-Processing - Implémentation

Afin de simuler toutes les voitures sur le même circuit, nous allons devoir changer l'architecture de notre projet !

Voyons notre circuit comme un serveur (RaceServeur)
qui possède un certain nombre de slots (CarServer)
pouvant chacun être piloté par un client (CarEnv)




Multi-Processing - Conclusion

Résultats :

- Vitesse d'apprentissage démultiplié (n Core)
- Utilisation optimale de la puissance de calcul

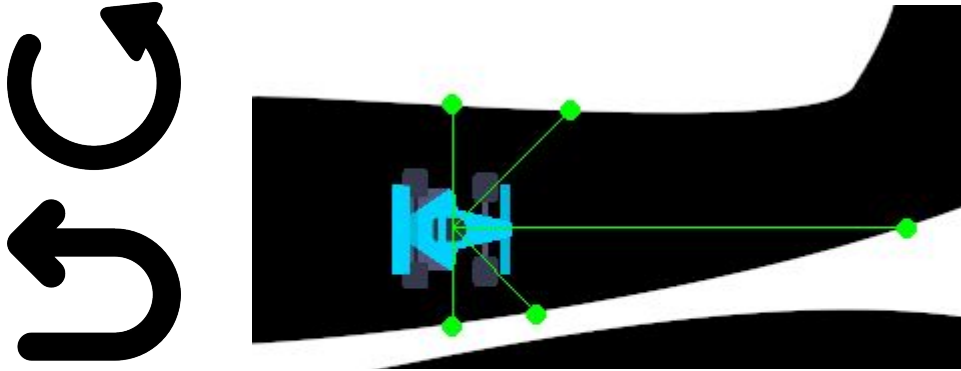
1 Core = 5 mins
2 Core = 2 mins 30 secs
4 Core = 1 mins 15 secs
8 Core = 37.5 secs



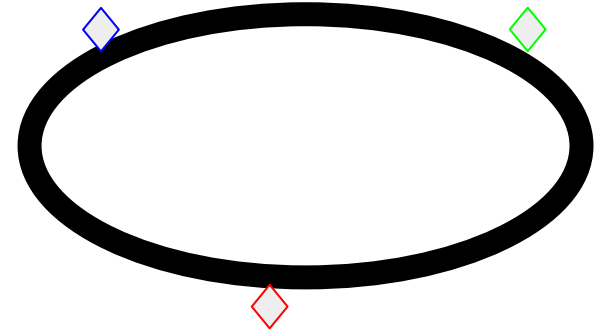
Amélioration possible :

- Utilisation de la puissance GPU

Mise en place des secteurs



- Demi-tours
- boucles



- pénalités
- récompenses

Optimisation des récompenses

- Distance
- Temps
- Vitesse

A2C	Seulement Distance	Distance + Vitesse
Carte 1	1min 2sec	26sec
Carte 2	2min 38sec	1min 3sec
Carte 3	7min 12sec	2min 35sec
Carte 4	3min 6sec	1min 31sec

Résultats

NEAT

	Carte 1	Carte 2	Carte 3	Carte 4
Nombre de Génération	2	5	40	9
Temps d'apprentissage	~5sec	~10sec	4min 50sec	~44sec

A2C

	Carte 1	Carte 2	Carte 3	Carte4
Temps d'apprentissage	~25sec	1min25	5min25	2min04

Résultats

NEAT

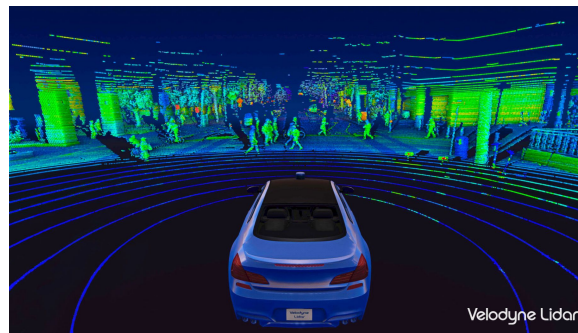
	Carte 1	Carte 2	Carte 3	Carte 4
Modèle carte 1				
Modèle carte 2				
Modèle carte 3				
Modèle carte 4				

A2C

	Carte 1	Carte 2	Carte 3	Carte 4
Modèle carte 1				
Modèle carte 2				
Modèle carte 3				
Modèle carte 4				

Ouverture

- Entraînement multi-circuit
- Complexité des radars
 - LIDAR
 - Caméra
- Environnement 3D
 - CARLA
 - DonkeyCar
- Voiture miniature



Références

- Vidéo Youtube de la chaîne “NeuralNine” [Self-Driving AI Car Simulation in Python](#)
- Code source de NeuralNine <https://github.com/NeuralNine/ai-car-simulation>
- Vidéo Youtube de la chaîne “Dave’s Space” [threading vs multiprocessing in python](#)
- Stable-Baselines Documentation : [lien](#)
- DonkeyCar <https://www.donkeycar.com>
- CARLA Sim <http://carla.org/>