# Résumé des fonctions du projet se trouvant dans envs.py, _init_.py, definition.py (env prof)

dans le répertoire src/agents/team2 , on peut trouver notre agent 2:
initialement, dans

## agent2.py:

Notre objectif est de passer d'un agent qui fait n'importe quoi (random) à un agent capable de suivre la piste et de gagner.
dans la fonction
pystk_process.py:
dans la fonction je vais expliquer les from et import:

**1- from functools import partial, partialmethod:**

**funtools** est un module standard de python qui sert à manipuler des fonctions.
**partial**: c'est une **fonction** définie dans functools qui sert à créer une nouvelle fonction à partir d'une autre, en **fixant certains arguments**.
import logging:
Un **module standard Python** pour écrire des messages de log( permet de decrire ce que fait le programme)

**2- from multiprocessing import Pipe, Process:**

**multiprocessing**:
Un module standard Python pour faire du **multi-process**.

**Process**
Une **classe** qui permet de créer un nouveau process Python. il sert à lancer une fonction dans un autre processus

**Pipe** : une fonction qui permet la communication du processus pere au processus fils

**connection**: c la classe des objets renvoyés par Pipe()

**sys**: donne acces au systeme python

**3- from typing import List, Optional**

**Optional**: le type est soit ce type soit None ex: Optional[int]
**pystk2**: pystk2 est une bibliothèque Python qui permet de contrôler le jeu SuperTuxKart depuis Python. SuperTuxKart est une machine de c++ donc on a besoin de traducteur.

on a la classe **class PySTKRemoteProcess**:
world; track; race sont les attributs.
pygtk2.Race = la course en cours (la simulation)

pygtk2.World State = l'état du monde (positions, phase, etc.)

pygtk2.Track = la piste (géométrie, segments)

***__init__*** *: C'est le* ***constructeur*** *: appelé quand on fait PySTKRemoteProcess(with_graphics).*

*je ne vais pas detailler. c plus je crois pour l'affichage.*

la méthode **list_tracks**: renvoie la liste des pistes disponibles dans STK
def list_tracks( self) -> List[str]:
La méthode **close(self)** permet d'arrêter proprement la course . self doit être remplacé par stk quand on l'utilise
**warmup_race**(config) :démarre une nouvelle course et attends que le STK soit prêt
il stoppe l'ancienne course si elle existe et crée une nouvelle course
get_world() : récupérer l'état actuel renvoie un message si pas de course sinon il met à jour les positions, vitesses, .. et renvoie World
race_step(*args) : avancer la simulation avec actions
get_kart_action(kart_ix) : lire l'action décidée par l'IA STK à utiliser que quand use_ai=True

II- envs.py:
1- les importations;
de typing, on a importé les types, **Any**, **Classvar**, **Dict**, **List**, **Optional**, **Tuple**, **TypedDict**
2- les fonctions:
**def kart_action_space():** ne prend aucun paramètre. elle renvoie un dictionnaire de commandes possibles( les descriptions des commandes)

- ce que contient ce dictionnaire:

```
    # Acceleration
    "acceleration": spaces.Box(0, 1, shape=(1,)),
    # Steering angle
    "steer": spaces.Box(-1, 1, shape=(1,)),
    # Brake
    "brake": spaces.Discrete(2),
    # Drift
    "drift": spaces.Discrete(2),
    # Fire
    "fire": spaces.Discrete(2),
    # Nitro
    "nitro": spaces.Discrete(2),
    # Call the rescue bird
    "rescue": spaces.Discrete(2),
```

on peut accélérer entre 0 et 1:  0 ne pas accélerer 1 accélérer à fond.
steer: on peut tourner fort à gauche -1; tout droit 0; +1 tourner fort à droite
**def kart_observation_space(use_ai):** elle définit ce que l'agent reçoit dans obs( obs a été defini dans getstate; elle recoit space return par kart_obser…
**obs["center_path_distance"]**: C'est un **nombre** qui indique **à quelle distance le kart est du centre de la piste**.

- valeur proche de **0** → le kart est bien centré
- valeur **positive** → le kart est décalé d'un côté
- valeur **négative** → le kart est décalé de l'autre côté

obs["paths_start"] et obs["paths_end"] sont des **segments de piste devant le kart**.

Chaque segment est défini par :

- un point de départ (paths_start)
- un point d'arrivée (paths_end)

Les coordonnées sont données **dans le repère du kart** :

- **z** = devant
- **x** = gauche/droite2.

**obs["velocity"]:** C'est un vecteur de vitesse du kart.

- sa norme = vitesse
- sa direction = direction du mouvement

**class BaseSTKRaceEnv(gym.Env[Any, STKAction]):**

BaseSTKRaceEnv est la **classe de base** des environnements de course SuperTuxKart. Elle implémente toute la logique commune pour :

1. **initialiser** SuperTuxKart via un processus séparé (PySTKProcess),
2. **configurer** une course (reset_race),
3. **mettre à jour** l'état du monde (world_update),
4. **construire** l'observation (get_observation),
5. **calculer** reward/terminaison/infos (get_state).

**LES MÉTHODES**

1. **def initialize(self, with_graphics: bool):** **avoir STK prêt** et connaître les pistes.

   Crée _process si ce n'est pas déjà fait.
   Charge TRACKS en appelant list_tracks() dans le processus STK.

2. **def __init__( self, *, render_mode=None, track=None, num_kart=3, max_paths=None, laps: int = 1,difficulty: int = 2,):** **objectif :** préparer l'environnement, sans encore démarrer une course.
3. **def reset_race(random, options=None):** créer la configuration STK de la course à lancer

nb: il y a d'autres  methodes dans la classe mais elles semblent se concentre beaucoup plus sur la course que sur notre agent. Donc, on ne va pas parler d'elles

4. **get_observation(kart_ix, use_ai):** Cette méthode transforme les données STK brutes en une observation exploitable :

   Cette méthode transforme les données STK brutes en une observation exploitable :

   **a) Passage au repère du kart : kartview(x)**

- transforme les coordonnées du monde en coordonnées relatives au kart.
- dans ce repère :
  - z = devant
  - x = gauche/droite

objectif : simplifier le pilotage (tout est vu "depuis le kart").

**b) Trier items et karts "par pertinence"**

- calcule positions relatives des autres karts et des items
- les trie en mettant d'abord ce qui est **devant** le kart

objectif: observation plus utile (évite un ordre aléatoire).

**c) Calculer la distance au centre de la piste**

- récupère le segment de piste courant (node kart.node)
- calcule un vecteur orthogonal vers la ligne centrale (x_orth)
- en déduit :
  - center_path_distance (distance signée)

- center_path (vecteur vers le centre)

**d) Retour final**

Renvoie un dictionnaire contenant notamment :

- velocity, max_steer_angle, distance_down_track
- center_path_distance, center_path
- items_position/items_type, karts_position
- paths_start/paths_end/paths_width/paths_distance

**class STKRaceEnv(BaseSTKRaceEnv):**

Dans STKRaceMultiEnv, l'environnement gère une course **multi-agent** : au lieu d'un seul obs, il renvoie **un dictionnaire d'observations** { "0": obs0, "1": obs1, ... }, et **chaque agent reçoit son propre obs** construit via get_observation. Les espaces Gym sont aussi des dictionnaires : action_space["0"] correspond à l'action de l'agent 0 et observation_space["0"] à son observation, ce qui signifie que chaque agent agit indépendamment tout en partageant la même course. Lors du reset(), chaque agent est associé à un kart précis, soit via rank_start (position imposée), soit via une position libre choisie automatiquement. L'environnement mémorise ensuite cette correspondance dans self.kart_indices, ce qui permet d'appliquer les actions au bon kart et de renvoyer l'observation correcte pour chaque agent.