Dans */src/pystk2_gymnasium/*
Utilisation de la librairie gymanisum, dont la class “spaces”
C’est grâce à cette librairie qu’on peut recevoir les observations de notre environnement
Par ailleurs, on peut retrouver les observations dans :
/src/pystk2_gymnasium/envs.py
Il y a la fonction kart_observation_space (ligne 74)
Également la class BaseSTKRaceEnv

La fonction kart_observation_space permet d’ “initialiser” nos “spaces”, sachant que toutes les observations sont stockés dans un “dictionnaire” (la méthode spaces.Dict)
Dans la class BaseSTKRaceEnv, il y a la méthode get_observation(self, kart_ix, use_ai) qui nous permet donc d’obtenir nos observations

On a pu également afficher “obs” qui est une variable présente dans notre agent
En faisant par exemple :

```
for i,j in obs.items() :
            print(f"{i} : {j}")
```

Liste des observations :

Il est important de noter que les variables qui sont des vecteurs (x, y, z) que :

- **x** représente l’axe latéral, et donc **la gauche** (et donc la droite en négatif)
- **y** représente l’axe vertical, et donc ce qu’il y a **au dessus** (ou en dessous en négatif)
- **z** représente la direction vers laquelle le kart pointe, autrement dit **l’avant** (ou l’arrière en négatif)

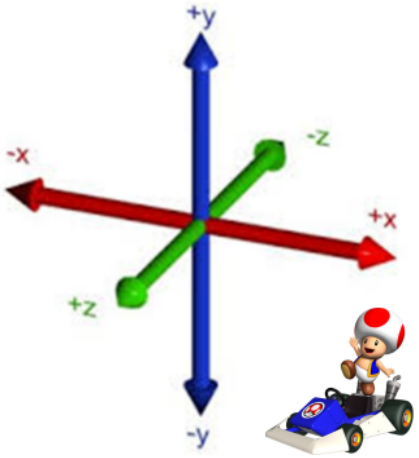
`distance_down_track`
Cette variable nous donne la distance du kart par rapport à la ligne de départ de la course. On peut mesurer notre progression depuis le début du circuit.

`energy`
C’est la quantité en stock de boost/nitro de notre kart.

`front`
C’est un vecteur en 3 dimensions qui nous indique l’orientation absolue du kart dans le jeu : [x, y, z] avec z : avant, x : gauche, y : haut.

Exemples :

- [0, 0, 1] : le kart regarde devant lui.
- [0.5, 0, 0.5] : le kart regarde à gauche (environ à 45°).
- [-0.5, 0, 0.5] : le kart regarde à droite, x est négatif car x représente, de base, le sens vers la gauche. Donc s'il est négatif alors on pointe vers la droite.



---

attachment

Cette variable décrit quelle "propriété physique" possède/subit notre kart. Il est en lien avec les items du jeu, et nous informe que l'item est actuellement actif.

En effet, cette variable a un lien avec les items suivants : bonus box, banana, nitro (small ou big), bubble gum, ou encore easter egg. Il permet d'identifier quel est l'item en question.

On peut supposer avec le README que :

- ➔ attachment = 0 → Aucun item
- ➔ attachment = 1 → Bonus box
- ➔ attachment = 2 → Banana
- ➔ attachment = 3 → Big Nitro
- ➔ attachment = 5 → Small Nitro
- ➔ attachment = 6 → Bubble Gum
- ➔ attachment = 7 → Easter Egg

Il est nécessaire de vérifier cette supposition et de corriger la documentation en conséquence.

Supposons également que attachment = 6.

Alors nous pouvons affirmer que notre kart possède un bubble gum (nous pouvons supposer que c'est un bouclier) et qu'il est activé.

C’est pourquoi, la variable *attachment_time_left* pourra nous indiquer pendant combien de temps.
Au contraire, nous pouvons supposer que attachment = 2.
Alors nous pouvons affirmer que notre kart subit l’effet banane et qu’il est donc étourdi pendant un laps de temps qui nous est donné grâce à la variable *attachment_time_left*.

---

attachment_time_left
Cette variable est liée à *attachment*.
Si la valeur est positive, cela veut dire que l’effet de l’objet attaché à notre kart est actif et cela nous indique également la durée pour laquelle il est encore actif (c’est donc une variable qui décroît avec le temps).
Si la valeur est nulle, alors nous n’avons aucun objet et donc *attachment* vaut 0.
Si la valeur devient nulle, alors l’objet passe d’actif à inactif et donc nous pouvons supposer que la valeur d’*attachment* est également modifiée.

Visualisation de la donnée

Imagine une barre de progression qui se vide au-dessus de ton kart :

- ➔ **Début :** attachment: Nitro & time_left: 3.0 (Pleine puissance)
- ➔ **Milieu :** attachment: Nitro & time_left: 1.5 (Moitié de l'effet consommé)
- ➔ **Fin :** attachment: None & time_left: 0.0 (Retour à la normale)

---

items_position
La variable items_position est une liste de vecteurs 3D où chaque vecteur représente la distance de chaque item par rapport à notre kart / détecté autour du kart.
Rappel : Les axes sont :

- z → devant le kart
- x → à gauche du kart
- y → au-dessus du kart

Exemple : Si items_position = [1.5, 0.0, 10.0], alors l’item est à 1.5 mètres à gauche et 10 mètres devant.

---

items_type

La variable items_type est complémentaire à items_position. Elle nous indique le type de chaque item présent dans items_position.

Exemple :

- items_position[3] = [1.5, 0.0, 10.0]
- items_type[3] = Banane

---

jumping

Variable booléenne indiquant si le kart est en train de sauter ou pas.

---

karts_position

Cette variable est une “liste” comprenant des vecteurs 3D
Il y a autant de vecteurs qu’il y a d’autres kart (s’il n’y a que toi, alors la liste est vide)

Exemple :

karts_position = [[10, 0, 0], [20, 5, -30], [0, 0, 20]]
Le deuxième vecteur nous indique qu’il y a un kart se situant à :

➔ 20 unités à gauche de notre kart
➔ 5 unités au dessus de notre kart
➔ -30 unités, autrement dit 30 unités derrière notre kart

---

max_steer_angle

Valeur maximale possible de l’angle de manœuvre du volant sachant votre vitesse actuelle. Plus on va vite, moins on peut tourner fort et plus la valeur est faible.

---

center_path_distance

La variable représente la distance entre le kart et le centre de la piste.
center_path_distance = **0** : le kart est parfaitement au centre
Valeur positive : le centre de la piste est à gauche du kart
Valeur négative : le centre de la piste est **à** droite du kart

---

center_path

Vecteur qui représente la direction vers laquelle le kart doit pointer vers le centre de la piste.

---

paths_start

Se trouve dans src/pystk2_gymnasium/envs.py au ligne 116-118:

```
116             "paths_start": spaces.Sequence(
117                 spaces.Box(float("-inf"), float("inf"), dtype=np.float32, shape=(3,))
118             ),
```

Les valeurs de “paths_start” peuvent être n’importe quel nombre(négatif, positif ou zéro) et il est un groupe de 3 nombres.
“paths_start” représente les points de départ des segments du circuit, il indique où commence chaque partie du parcours.
pour trouver “paths_start” de la première segment:

```
paths_start = obs["paths_start"] # liste des points de debut des segments du circuit,
                                 # les segments sont stockés sous forme de listes de vecteurs 3D (x, y, z)
start = paths_start[0] # point de debut du premier segment
print("Le point de debut du premier segment est:", start) # [ 2.3094554   0.12107077 -4.79539   ]
```

---

paths_end

Se trouve dans src/pystk2_gymnasium/envs.py au ligne 119-121:

```
119             "paths_end": spaces.Sequence(
120                 spaces.Box(float("-inf"), float("inf"), dtype=np.float32, shape=(3,))
121             ),
```

Les valeurs de “paths_end” peuvent être n’importe quel nombre(négatif, positif ou zéro) et il est un groupe de 3 nombres.
“paths_end” représente les points d’arrivée des segments du circuit, il indique où finit chaque partie du parcours.
pour trouver “paths_end” de la première segment:

```
paths_end = obs["paths_end"] # liste des points de fin des segments du circuit
end = paths_end[0] # point de fin du premier segment = la debut du dexieme segment
print("Le point de fin du premier segment est:", end) # [2.3262134  0.17199045 5.2716656 ]
```

---

paths_width

Se trouve dans src/pystk2_gymnasium/envs.py au ligne 113-115:

```
113             "paths_width": spaces.Sequence(
114                 spaces.Box(0, float("inf"), dtype=np.float32, shape=(1,))
115             ),
```

Les valeurs de “paths_width” sont des nombres supérieurs ou égal à 0(largeur ne peut pas être négative), chaque élément est un seul nombre(qui n’est pas de coordonnées 3D).
“paths_width” représente la largeur de chaque segment du circuit.
pour trouver “paths_width” de la première segment:

```
paths_width = obs["paths_width"] # liste des largeurs des segments du circuit
width_premiere_segment = paths_width[0] # largeur du premier segment
print("La largeur du premier segment est:", width_premiere_segment) # [12.549374]
```

---

paths_distance

Le circuit dans SuperTuxKart n'est pas géré comme une simple image, mais comme une succession de points invisibles appelés **Nœuds** (Nodes) qui forment la ligne médiane de la route.

- Un **segment** est l'espace compris entre deux nœuds consécutifs (Nœud A -> Nœud B).
- L'IA ne "voit" pas le virage complet, elle voit qu'elle est actuellement sur le segment qui relie le point A au point B.

La variable `paths_distance` renvoie un vecteur `[Début, Fin]` indiquant la position curviligne de ces nœuds par rapport à la ligne de départ.

- **Début (Index 0)** : Distance en mètres du début du segment actuel depuis le départ.
- **Fin (Index 1)** : Distance en mètres de la fin du segment actuel depuis le départ.
- **Longueur du segment** : `Fin - Début`.

Donc au final on nous renvoie la distance du noeud derrière nous par rapport à l'origine et la distance du noeud suivant par rapport aussis à l'origine

Les segments n'ont **pas** une longueur fixe :

- **Segments longs (>10m)** : Utilisés dans les lignes droites, car il faut peu de points pour définir une droite.
- **Segments courts (<8m)** : Utilisés dans les virages. Plus le virage est serré, plus les segments sont courts et nombreux pour créer une courbe fluide.

Interprétation pour l'agent :

- Si `Fin - Début` est petit, l'agent est probablement dans un virage complexe.
- Si `Fin - Début` est grand, l'agent est probablement sur une ligne droite où il peut accélérer.

```
p_dist_current = obs['paths_distance'][0] # paths_distance est une liste de vecteurs. On regarde juste le premier (le segment actuel)
p_dist_current_debut = p_dist_current[0] # La distance du noeud qui commence le segment par rapport à l'origine
p_dist_current_fin = p_dist_current[1] # La distance du noeud qui fini le segment par rapport à l'origine
```

---

powerup

Contrairement à attachment, cette variable nous indique quel item nous avons en main (et non si celui-ci est actuellement actif). De la même manière, un nombre est associé à un item, et nous pouvons supposer que 0 signifie que nous possédons aucun item.

Pour mieux différencier les deux :

Caractériques	powerup (Inventaire)	attachment (État actif)
Moment	“J’ai l’objet en main.”	“L’objet est en train d’agir.”
Action	L’Agent doit décider d’appuyer sur “Feu”.	L’Agent subit ou profite de l’effet. Dans le cas d’un Nitro, il en profite.
Exemple (Nitro)	Tu as une icône de Nitro en stock.	Le feu sort de ton pot d’échappement.

---

shield_time

Cette variable représente la **durée restante d'invulnérabilité** du kart. Elle indique pendant combien de temps le kart est protégé contre les dégâts, les ralentissements et les effets de statut négatifs.

- **shield_time == 0.0** : Le kart est **vulnérable**. Tout impact (tir, banane, mur violent) aura un effet immédiat.
- **shield_time > 0.0** : Le kart est **invincible**. Les projectiles traversent le kart sans le toucher et les collisions sont sans conséquences graves.

---

skeed_factor

Une valeur numérique (float) mesurant l'intensité du dérapage actuel du kart.

Observation :

- **1.0 ou moins** : Le kart a une adhérence normale ou commence à peine à tourner.

- **Supérieur à 2.0** : Le kart est en dérapage important (drift).
- **Plafond observé** : Lors de mon test en dérapage forcé (`drift=True`), la valeur a atteint un maximum stable de **2.5**.

Utilité pour l'IA : L'agent peut surveiller cette valeur pour optimiser ses turbos. Maintenir un `skeed_factor` élevé pendant une certaine durée permet de charger le "Mini-Turbo" (étincelles bleues, puis rouges). L'IA peut décider de relâcher le bouton de saut uniquement quand ce facteur a été élevé assez longtemps.

`velocity`

Se trouve dans src/pystk2_gymnasium/envs.py au ligne 88-90:

```
88              "velocity": spaces.Box(
89                  float("-inf"), float("inf"), dtype=np.float32, shape=(3,)
90              ),
```

Les valeurs de “velocity” peuvent être n’importe quel nombre(négatif, positif ou zéro) et il est un groupe de 3 nombres. “Velocity” représente l’état de mouvement actuel du kart, il indique dans quelle direction le kart se déplace et à quelle vitesse.
pour vérifier la vitesse du kart:

```
velocity = (0, 0, 1) # les trois valeurs se represente (gauche/droite, haut/bas, avant/arriere) comme (x,y,z)
                     #la valeur z peut etre negatif
# pour la vitesse on ne regarde que la composante z (avant/arriere): plus il est grand, plus le kart est rapide
vitesse = velocity[2] # on recupere la valeur z
if vitesse > 1:
    print("Le kart est rapide (peut-etre en train d'utiliser du nitro)")
elif vitesse < 0.5:
    print("Le kart est lent(peut-etre arrete ou en train de affecté par une banane ou un chwing-gum)")
```

pour vérifier si le kart saute:

```
velocity = (0, 2, 1) #la deuxieme valeur est y correspondant haut/bas
if velocity[1] > 0:
    print("Le kart saute") # si la valeur y est positive, le kart est en train de sauter
else:
    print("Le kart est sur le sol") # sinon, il est sur le sol
```