Compte Rendu semaine 16/02/2026

Réfléchir sur l'implémentation d'un code qui permet de drift en fonction des différents virages

L'anticipation est la clé du drift : on ne commence pas à drifter pendant le virage, on prépare le drift avant. Le drift s'active avec l'étiquette drift. Pour cela il faut représenter 2 conditions pour être efficace: la vitesse suffisante: on ne peut pas drifter si on roule trop lentement + angle d'attaque: il faut braquer le volant en même temps que l'on appuie sur le bouton. On peut modifier adapteAcceleration pour gérer le drift. Nous allons lui faire renvoyer deux valeurs: l'accélération (déjà renvoyé par la fonction) **ET** un booléen pour le drift. Il faut mettre à jour la fonction choose_action pour récupérer les valeurs.

A réfléchir: augmenter la vitesse en fonction de l'accélération (n'est pas une très bonne idée)→ il serait donc beaucoup plus juste de définir une vitesse pour déterminer l'accélération lors d'un drift. La force centrifuge sera beaucoup plus importante à cause de l'augmentation de l'accélération. Donc le kart sera hors de la piste.

Il faudra donc prendre le vecteur «velocity», en prenant les coordonnées et calculer la norme de ce vecteur. Et lors d'un drift nous allons augmenter cette vitesse pour que le drift soit le plus efficace possible (usage du nitro maybe), et diminuer l'accélération peut empêcher le kart de sortir de la piste → faire calcul de vitesse en fonction des virages. Pour les items: prendre le nitro (bouteille bleue) permet d'accélérer fortement.

ALGORITHME drift

#Mesurer l'état actuel : Calculer la vitesse réelle et la courbure du virage à venir. $(v = \sqrt{x^2 + y^2 + z^2})$

#Vérifier l'éligibilité => Est-ce que la courbure est assez forte ? (Seuil de virage)

    #Est-ce que j'ai assez de vitesse ? (On ne drift pas à 2 km/h)

#Phase d'entrée (Anticipation) :

    #Si éligible : Activer drift = True et maintenir un steer (volant) prononcé.

    #Gestion de l'accélération : Réduire légèrement l'accélération au début pour laisser le kart "pivoter" sans glisser tout de suite vers l'extérieur.

#Phase de sortie (Nitro) :

    #Si le drift est stabilisé et que la vitesse chute trop à cause du dérapage : Activer le nitro pour compenser la perte de vitesse latérale et "s'arracher" du virage.

$\vec{x} = \vec{OM} = x\vec{u_x} + y\vec{u_y} + z\vec{u_z}$ $\quad\quad$ $\|x\| = \|\vec{OM}\| = \sqrt{x^2+y^2+z^2}$

$\vec{u_x} = \cos(\varphi)\times\vec{u_x} + \sin\varphi\vec{u_y}$

$\vec{u_\varphi} = -\sin(\varphi)\vec{u_x} + \cos(\varphi)\vec{u_y}$

Vitesse instantanée à un temps t $\quad \vec{v}(t) = \dfrac{d\vec{OM}}{dt} = \dfrac{d\vec{R}}{dt}$

$$\frac{d\vec{R}(t)}{dt} = \frac{d(x(t)\vec{u_x})}{dt} + \frac{d[y(t)\vec{u_y}]}{dt} + \frac{d[z(t)\vec{u_z}]}{dt}$$

$$= \frac{dx}{dt}(t)\vec{u_x} + \frac{d\vec{u_x}}{dt} + \frac{dy(t)}{dt}\vec{u_y} + \frac{d\vec{u_y}}{dt} + \frac{dz(t)}{dt}\vec{u_z} + \frac{d\vec{u_z}}{dt} = \frac{dx}{dt}(t)\vec{u_x} + \frac{dy}{dt}(t)\vec{u_y} + \frac{dz}{dt}(t)\vec{u_z}^{*}$$

OR les vecteurs unitaires $\vec{u_x}, \vec{u_y}, \vec{u_z}$ sont nulles car dans notre référentiel les vecteurs sont fixes (il n'y a pas de variation de la position de ces vecteurs donc $\dfrac{d\vec{u_x}}{dt} = 0$)

$$\frac{d\vec{u_r}}{dt} = \frac{d\cos(\varphi)\vec{u_x}}{dt} + \frac{d\sin(\varphi)\vec{u_y}}{dt}$$

$$= -\dot{\varphi}\sin(\varphi)\vec{u_x} + \dot{\varphi}\cos(\varphi)\vec{u_y}$$

$$= \dot{\varphi}\left(-\sin(\varphi)\vec{u_x} + \cos(\varphi)\vec{u_y}\right)$$

$$= \dot{\varphi}\vec{u_\varphi}$$

* on peut noter $\dfrac{dx}{dt} = \dot{x}$

$$v(t) = \frac{d\vec{R}}{dt} = \dot{x}(t)\vec{u_x} + \dot{y}(t)\vec{u_y} + \dot{z}(t)\vec{u_z}$$

$$= \frac{d\vec{r}}{dt} = \frac{dr\vec{u_r}}{dt} = \dot{r}\vec{u_r} + r\left(\frac{d\vec{u_r}}{dt}\right) = \dot{r}\vec{u_r} + r\dot{\varphi}\vec{u_\varphi}$$

Pour avoir l'accélération, on dérive la vitesse par rapport au temps.

$$\vec{a}(t) = \frac{d\vec{v}(t)}{dt} = \frac{d\dot{r}\vec{u_r}}{dt} + \frac{dr\dot{\varphi}\vec{u_\varphi}}{dt} = \frac{d\dot{r}}{dt}\vec{u_r} + \dot{r}\frac{d\vec{u_r}}{dt} + \frac{dr}{dt}\cdot\dot{\varphi}\vec{u_\varphi} + r\frac{d\dot{\varphi}}{dt}\vec{u_\varphi} + r\dot{\varphi}\frac{d\vec{u_\varphi}}{dt}$$

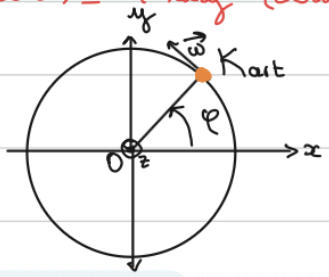
$$= \ddot{r}\vec{u_r} + \dot{r}\dot{\varphi}\vec{u_\varphi} + \dot{r}\dot{\varphi}\vec{u_\varphi} + r\ddot{\varphi}\vec{u_\varphi} + r\dot{\varphi}(-\dot{\varphi}\vec{u_r})$$

$$= \ddot{r}\vec{u_r} + \dot{r}\dot{\varphi}\vec{u_\varphi} + \dot{r}\dot{\varphi}\vec{u_\varphi} + r\ddot{\varphi}\vec{u_\varphi} - r\dot{\varphi}^2\vec{u_r}$$

$$\vec{a}(t) = \boxed{(\ddot{r} - r\dot{\varphi}^2)\vec{u_r} + (2\dot{r}\dot{\varphi} + r\ddot{\varphi})\vec{u_\varphi}}$$

On travaille sur la variation des angles lors de la course. On a le vecteur vitesse angulaire qui exprime la vitesse angulaire d'un système ainsi que l'axe autour duquel le système tourne.

$\vec{\omega}(t) = \dot{\varphi}(t)\vec{u_y}$ (Dans SuperTuxKart, la hauteur est en fonction de l'axe des y)



Si l'on suppose qu'un virage est circulaire, on a un rayon fixé à R avec un référentiel fixe avec l'origine au centre du cercle.

Comme $r = R$ et R ne varie pas alors $\dfrac{dr}{dt} = \dfrac{dR}{dt} = 0$

Ainsi $\vec{v}(t) = r\dot{\varphi}\vec{u_\varphi} + \dot{r}\vec{u_r} = R\dot{\varphi}\vec{u_\varphi} + \vec{0} = R\omega\vec{u_\varphi}$

$$\vec{a}(t) = \frac{d\vec{v}}{dt} = (\underbrace{\ddot{x}}_{=0} - x\dot{\varphi}^2)\vec{u_r} + (2\dot{x}\dot{\varphi} + x\ddot{\varphi})\vec{u_\varphi}$$

$$= (-R\dot{\varphi}^2)\vec{u_r} + R\ddot{\varphi}\vec{u_\varphi}$$

$$= -R\omega^2\vec{u_r} + R\dot{\omega}\vec{u_\varphi}$$

Précédemment, nous avons supposé que les virages suivant un mouvement circulaire OR cette supposition néglige de nombreux cas qui peuvent faire SORTIR le kart de la piste ou être moins rapide.

Pour cette raison, on va se concentrer sur un mouvement curviligne. On introduit ainsi la base de Frenet (origine qui dépend du temps, idem pour le rayon). On a ainsi $\vec{u_T}$ (tangent à la trajectoire dans le sens du mouvement et $\vec{u_N}$ est normale (perpendiculaire) à la trajectoire.

Similaire à la base polaire, $\vec{u_T}$ correspond à $\vec{u_\varphi}$ et $\vec{u_N}$ à $-\vec{u_R}$.

$\vec{v} = v(t)\vec{u_T}$

$$\vec{a} = \frac{d\vec{v}}{dt} = \frac{dv(t)}{dt}\vec{u_T} = \frac{dv(t)}{dt}\vec{u_T} + \frac{d\vec{u_T}}{dt} = \dot{v}\vec{u_T} + v\dot{\varphi}\vec{u_N}$$

$$= \dot{v}\vec{u_T} + \frac{v^2}{R}\vec{u_N} \longrightarrow \text{on sait que } \omega = \dot{\varphi}\vec{u_z} \text{ et } v = R\omega \Leftrightarrow \omega = \frac{v}{R}$$

$$= a_T\vec{u_T} + \frac{v^2}{R}\vec{u_N}$$

Plus la vitesse augmente + le rayon diminue

Plus le rayon est petit, alors il y a une forte variation de la direction du vecteur vitesse

$a_T\vec{u_T}$: lié à la variation de la vitesse

$\frac{v^2}{R}\vec{u_N}$: lié à la courbure de la trajectoire

Par la deuxième loi de Newton : $\Sigma F = ma = m\frac{v^2}{R} \Leftrightarrow a_{max} = \frac{v^2}{R} \Leftrightarrow v = \sqrt{a_{max} \times R}$??

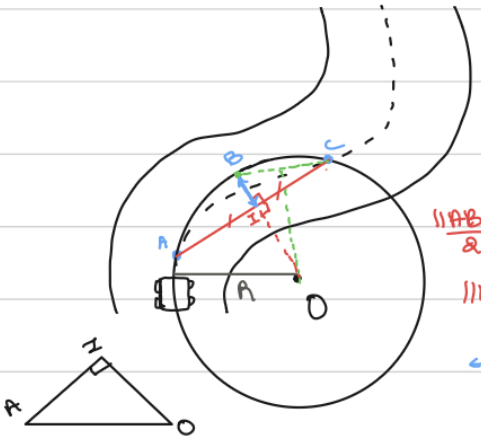
si $m\frac{v^2}{R} > F$ alors sortie de virage

v : vitesse limite avant de virager

Comment déterminer R ?

Si $v_{actuelle} > v_{lim}$ => il faut freiner et atteindre cette vitesse limite pour rester sur la piste

Comment déterminer le rayon R ?



$\frac{\|AB\|}{2} = \|AI\| = \|IB\|$

$\|AB\|$ → corde

↔ flèche $\|IS\|$

$$r = \frac{\|AB\|}{8 \times \|IS\|} + \frac{\|IS\|}{2}$$

par le théorème de Pythagore : $OA^2 = OI^2 + IA^2$

$$R^2 = (R-h)^2 + \left(\frac{d}{2}\right)^2$$

On note $\frac{AB}{2} = AM = \frac{d}{2}$

$$R^2 = R^2 - 2Rh + h^2 + \frac{d^2}{4} \quad \leftarrow \text{On substitue}$$

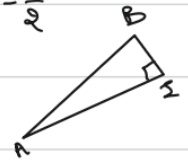
$$\Rightarrow 0 = -2Rh + h^2 + \frac{d^2}{4}$$

$$\Leftrightarrow -2Rh = -h^2 - \frac{d^2}{4}$$

$$\Leftrightarrow 2Rh = h^2 + \frac{d^2}{4}$$

$$\Leftrightarrow R = \frac{h^2}{2h} + \frac{d^2}{4 \times 2h}$$

$$\boxed{R = \frac{h^2}{4} + \frac{d^2}{8h}}$$



$$\left(\frac{AC}{2}\right)^2 + BI^2 = AB^2$$

on cherche BI^2 :

$$BI^2 = AB^2 - \left(\frac{AC}{2}\right)^2$$

$$BI = \sqrt{AB^2 - \left(\frac{AC}{2}\right)^2}$$

$BI = h$

Se renseigner sur comment faire des test correctement en python et aide a faire cette tâche

Il y a 3 niveaux :

1. Tests Unitaires (Unit Tests) → but : Tester une seule fonction isolée. (très rapide + facile)
2. Tests d'Intégration → but : Tester si deux fonctions marchent ensemble. (rapide + moyen)
3. Tests de Simulation (End-to-End) → but : Lancer le jeu et tester la course entière. (lent + difficile)

1. La bibliothèque standard : unittest

Python possède un module intégré très puissant appelé unittest. Il permet de définir des cas de test ("test cases") et des assertions ("assert") pour vérifier les résultats.
/!\ Il faut toujours commencer par des tests unitaires sur les fonctions de calcul (exemples : anticipeVirage, adapteAcceleration) avant de lancer le jeu complet.

2. Contraire du Tests Unitaires, Tests d'Intégration

Les Tests d'Intégration vérifient que plusieurs fonctions travaillent bien ensemble.
Dans notre cas le test d'intégration idéal consiste à vérifier la chaîne :
Données (obs) → Détection de virage → Calcul de vitesse → Action finale.
Pour faire un Test d'Intégration, on utilise la même structure que pour les tests unitaires, mais on fait appel à plusieurs méthodes de la classe dans le même test.

3. Le Test de Simulation (End-to-End)

Le Test de Simulation (End-to-End) est le test "vérité".
C'est le moment où l'on vérifie si l'agent est réellement capable de remplir sa mission du début à la fin dans son environnement réel.
On teste toute la chaîne sans exception c'est à dire :

- L'entrée (images et données brutes)
- Le cerveau (fonctions implémentées)
- La sortie (actions exécutées)
- Le résultat (ligne d'arrivée franchie)

C'est un test qui prend en compte les imprévus physiques.

Comprendre l'héritage en POO

L'héritage est un concept fondamental de la Programmation Orientée Objet (POO) qui permet à une classe (l'enfant) de récupérer les attributs et les méthodes d'une autre classe (le parent).
C'est comme dans la vraie vie : un enfant hérite des caractéristiques de ses parents, mais peut aussi avoir ses propres particularités.

1. Pourquoi utiliser l'héritage ?

• Réutilisation du code : on n'a pas besoin de réécrire les mêmes fonctions pour des objets similaires.
• Organisation : Cela permet de structurer le code de manière logique (du plus général au plus spécifique).

2. **Syntaxe et Construction**

On utilise la classe mère entre parenthèses lors de la définition de la classe fille.

```
class Mere:
def init(self, nom):
self.nom = nom
def saluer(self):
    print(f"Bonjour, je suis {self.nom}")
class Fille(Mere): # Fille hérite de Mere
def init(self, nom, age):
# super() appelle le constructeur du parent pour initialiser 'nom'
super().init(nom)
self.age = age
```

3. **Surcharge de méthode (Overriding)**

La classe fille peut redéfinir une méthode de la classe mère pour modifier son comportement.
On peut utiliser super() pour appeler la méthode originale.

```
class Fille(Mere):
# ... init ...def saluer(self):
    # Modification du comportement de la méthode mère
    print("--- Début de la salutation ---")
    super().saluer()  # Appelle la méthode originale
    print(f"J'ai {self.age} ans.")
    print("--- Fin de la salutation ---")
```

4. **Vérification de l'héritage**

Python propose des fonctions intégrées pour vérifier les relations entre classes et objets.

```
#Vérifier si une classe hérite d'une autre
if issubclass(Fille, Mere):
print("Fille hérite bien de Mere.")
#Vérifier si un objet est une instance d'une classe (ou de ses enfants)
mon_objet = Fille("Alice", 25)
if isinstance(mon_objet, Mere):
print("mon_objet est bien un type de Mere.")
```

3. **Les points clés:**

• class Enfant(Parent) : La syntaxe pour hériter.
• super() : Une fonction magique qui permet à l'enfant d'appeler les méthodes du parent (très utilisé dans le **init** pour initialiser les attributs du parent).
• Surcharge (Overriding) : L'enfant peut redéfinir une méthode du parent pour qu'elle agisse différemment.

Se documenter sur les fichiers de configuration YAML

YAML (= Ain't Markup Language) est le format de texte standard utilisé dans presque tous les projets d'IA et de robotique pour séparer les paramètres (vitesse, seuils de drift) du code (la logique).
→ Ne contient pas de calculs, seulement des données.
Comme en Python, l'indentation est importante pour montrer quel élément appartient à quel groupe.
-Structure Clé (= format d'écriture) → Valeur: on écrit le nom du paramètre, deux points, et la valeur. Elle utilise des paires Clé: Valeur (nom: Kart).

Exemples :
vitesse_max: 1.0
nom_du_pilote: "DemoPilote"
-Pas de Tabulations
-Utiliser toujours des espaces (2 espaces par niveau).
-Types de données, YAML supporte :
• les chaînes de caractères
• les nombres
• les booléens
• les listes
• les dictionnaires imbriqués
Les listes: on utilise un tiret (-) pour créer une liste d'éléments.
Exemple :
circuits_preferes:

- Cocoa Temple
- Cornice

But du fichier config YAML:
Au lieu de changer les valeurs directement dans le fichier agent2.py et de devoir relancer tout le code, on met ce chiffre dans un fichier config.yaml.
Le code Python va lire ce fichier.
On utilise la bibliothèque PyYAML.

Documentation sur les fichiers de configuration .yaml

Rôle :
Le fichier yaml sert de couche de configuration . Il permet de modifier les parametres de la simulation sans avoir à toucher au code source .

Règles de syntaxe :
Il est structuré par des indentations .
Interdiction formelle des tabulations : Les caractères de tabulation provoquent des erreurs de lecture.
Utilisation des espaces : L'indentation doit se faire uniquement avec des espaces .
Chaque niveau décalé vers la droite est un "enfant" de la ligne supérieure.
Clé-valeur : les données sont stockées sous la forme Clé: Valeur . Il faut toujours laisser un espace après les deux points .
Liste : pour définir une liste d'éléments (par exemple la liste des adversaires) il nous faut des tirets à chaque ligne .

Faire un fichier de configuration:

Objectif: Le but du fichier de configuration est de pouvoir modifier directement à partir du fichier yaml les variables qu'on souhaite afin de ne pas avoir à faire constamment des modifications sur le fichier agent2.

OmegaConf c'est un moyen de lire des fichiers de configuration (comme notre configDemoPilote.yaml) dans Python.

Au lieu de faire ca: (avec yaml)
config = yaml.safe_load(open("config.yaml"))
print(config["model"]["path"])
-> trop compliqué, dictionnaire

On fait ca :(avec omegaconf)
from omegaconf import OmegaConf
config = OmegaConf.load("configDemoPilote.yaml")
print(config.model.path)
->notation pointé des variables

->on met notre fichier de config dans la variable config pour ne pas avoir a écrit configDemoPilote a chaque fois dans agent2.py (trop long).

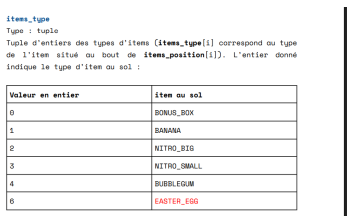
Extrait de notre fichier de configuration:

```yaml
virages:
 drift: 2.80   #seuil nécessitant un drift (très serré)
 serrer:       #intervalle virages serrés
   i1: 1.80
   i2: 2.80
 moyen:        #intervalle virages moyen
   i1: 0.85
   i2: 1.80
```

pour accéder à la valeur 0.85:
-> config.virages.moyen.i1

Fonction qui réagit aux obstacles:

Tableau des items trouve dans le doc du team 1:



items_type
Type : tuple
Tuple d'entiers des types d'items (**items_type**[i] correspond au type de l'item situé au bout de **items_position**[i]). L'entier donné indique le type d'item au sol :

Valeur en entier	item au sol
0	BONUS_BOX
1	BANANA
2	NITRO_BIG
3	NITRO_SMALL
4	BUBBLEGUM
6	EASTER_EGG

Comportement du kart face aux objets (items) sur la piste :

```python
def reactions_items(self, obs):
```

1. Récupérer les obstacles sur la piste à partir de obs['items'].

2. Utiliser une boucle for pour parcourir chaque item.

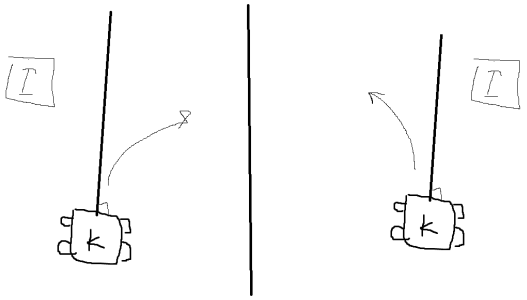
3. Vérifier la distance entre chaque item et le kart.

4. Ignorer l'item s'il se trouve derrière le kart ou s'il est trop éloigné.

5. Vérifier le type de l'item.

    a. S'il s'agit d'une boîte ou d'un nitro, aller le récupérer.
        i. Ajuster l'angle pour se diriger vers l'item le plus proche du kart.

    b. Sinon, dans le cas d'obstacles à éviter, entreprendre une manœuvre d'évitement.
        i. Si pos[0] > 0 (l'obstacle est à droite du kart), braquer le volant vers la gauche.
        ii. Sinon, braquer le volant vers la droite.



    c. L'évitement des mauvais objets est prioritaire sur la récupération des bons objets.