COMPTE RENDU SEMAINE DU 9/02/2026:

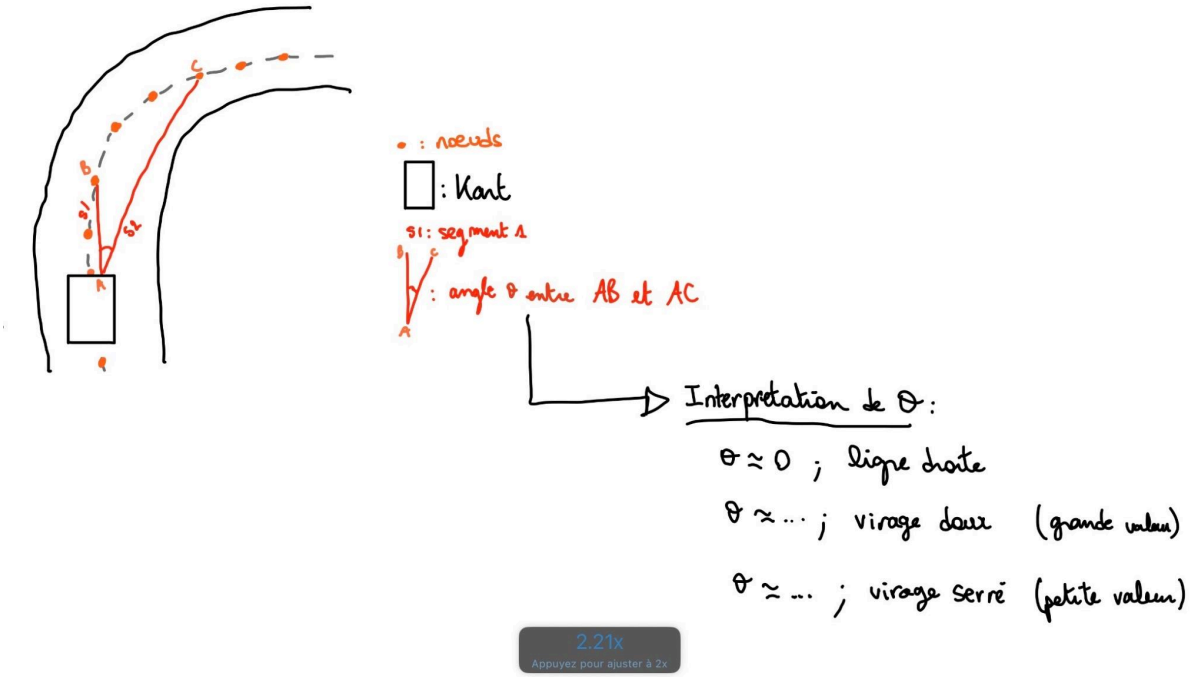
Fonction permettant de détecter les virages et de stocker la valeur de la courbe du virage dans une liste et indique à quel indice il a été trouvé:

```
   def anticipeVirage(self):

       nodes_path = obs["paths_start"] #liste des neoud de la piste
       nb_nodes = len(nodes_path)
       path_lookahead = 5

       virages = [] #liste resultat pour stocker les virages détectés

       for i in range (nb_nodes - path_lookahead): #boucle pour le second (noeud
loin=anticipation)

           curr_node = nodes_path[i] #le premier noeud qu'on rgd (noeud proche)
           lookahead_node = nodes_path[i+path_lookahead] #noeud loin

           x1, z1 = current_node[0], current_node[2] #coordonnées pour angle
           x2, z2 = lookahead_node[0], lookahead_node[2]

           angle1 = np.arctan2(x1, z1)
           angle2 = np.arctan2(x2, z2)

           curvature = abs(angle2 - angle1)

           if curvature > 0.1:  # seuil à ajuster
               virages.append({ "index": i, "curvature": curvature })


       print("Virages détectés :", virages)

       return virage
```

Réflexion:

Idée globale pour détecter un virage:
Prendre les coordonnées de deux points, un point proche du kart et un point plus loin. Puis pour savoir si ya virage ou pas dans les environs, on calcule l'angle entre le segment formé par le nœud de la position actuelle du kart avec le nœud proche et le segment formé par le nœud actuel et le nœud loin.

Voici ci dessous un schéma explicatif qui a permis d'arriver à cette stratégie:



Nb noeuds : len(nodes_path)
-> Pour adapter la valeur de look_ahead

```
nodes_path = obs["paths_start"]
nodes_path = [
    [x0, y0, z0],  # nœud 0
    [x1, y1, z1],  # nœud 1
    [x2, y2, z2],  # nœud 2
]
```

self.path_lookahead = 5
→ on regarde le 5e point après la position actuelle du kart.

Pour exploiter la fonction anticipeViarge() il faudra utiliser la liste virage renvoyé pour regarder les curvatures et adapter le comportement de l'agent en fonction de ca.
Pour la suite (fonction qui adapte la vitesse) il faudra faire différent niveau de virages avec différentes courbures car on adaptera la vitesse du kart différemment.

self.curvatures = [0.01, 0.02, 0.15, 0.30, 0.28, 0.05, ...]

Interprétation de self.curvature :
valeur proche de 0 → ligne droite
valeur moyenne → virage doux (< 0.2)
valeur grande → virage serré (0.25 - 0.5)

Chaque valeur correspond à une zone de la piste.

Fonction permettant d'adapter l'accélération dans diverses situations dont notamment les virages serrés , les lignes droites , les virages moyens.

```
71      def adapteAcceleration(self,obs):
72          #le but va etre d'adpater l'acclération dans diverses situations dont notamment
73          #les virages serrés, les lignes droites ou une legere curvature
74          liste_virage=self.anticipeVirage(obs)
75          acceleration = 0.90
76          if len(liste_virage) < 1 :  # s'il n'y a pas de virage
77              acceleration = 1.0  # conduite normale on pourrait augmenter légèrement l'accélération -> à décider
78          else :
79              proche_virage = liste_virage[0]
80              curvature = proche_virage["curvature"]
81              #print (curvature) # permet d afficher la variation des angles pour determiner les courbures
82              if curvature > 2.80 :
83                  #drift = True
84                  acceleration = acceleration - 0.30
85              elif curvature > 1.80 and curvature <=2.80: # virage serré
86                  acceleration= acceleration - 0.25
87                  #drift = False
88              elif curvature > 0.85 and curvature <= 1.80:  #virage moyen
89                  acceleration = acceleration - 0.20
90                  #drift = False
91              else :      You, 7 hours ago • ajout du code adapteAcceleration + modification…
92                  acceleration = acceleration - 0.10
93                  #drift = False
94          return acceleration #drift
95      #on travaillera sur les drifts apres depuis cette fonction
96
97
```

Pour aboutir à l'adaptation de l'accélération durant la course, le but a été d'implémenter un code qui détecte les types de virages en fonction de leur angle (voir le code anticipeVirage) à partir d'une liste qu'on a nommée liste_virage.

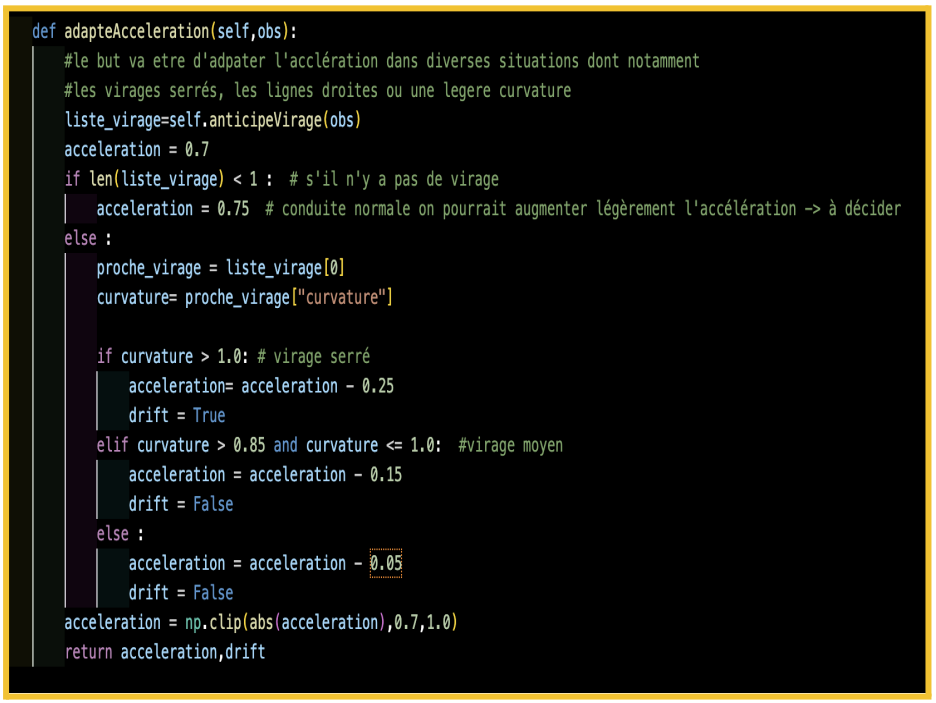
Dans un premier temps nous appliquons la condition suivante : **if len(liste_virage) < 1** , en effet cette condition permet de vérifier si une liste est vide. Dans le cas où la liste est vide, alors cela signifie qu'il n'y a aucun virage sur la piste à un instant t donné. Ainsi si cette condition est fausse, nous analyserons tous les types de virages possibles : que ce soient des virages serrés , des lignes peu courbés ou des virages moyens.

Comment a-t-on choisi ces valeurs précisément ?
Ce code a fait l'objet de nombreux tests sur plusieurs courses. En effet , les valeurs qui ont été choisies pour la variable curvature ont été déduites grâce aux valeurs d'affichage de la fonction ancitipeVirage. Cependant, nous avons rencontré plusieurs problèmes lors des premières conditions. La valeur des curvatures n'étaient pas bien adaptées car un virage qui a un angle supérieur à 0,1 ne peut être considéré comme un virage , cela correspond à ~ 5 ° , une valeur proche de 0°. Le kart n'avançait donc pas en avant mais bien vers l'arrière ou parfois il était comme bloqué sur la piste.

Pour cette raison, nous avons décidé d'augmenter la valeur permettant de valider la condition de la fonction précédente (anticipeVirage) à 0.35 ce qui équivaut à 20,05 ° environ. Ce qui est donc plus cohérent dans notre cas. Après ces ajustements, les conditions de la fonction adapteAcceleration(self,obs) ont été modifiées à nouveau.

C’est ainsi que nous avons déterminé des seuils d’identification de divers virages pour adapter l’accélération.

Code d’essai avec problème d intervalles + condition fausse



```python
def adapteAcceleration(self,obs):
    #le but va etre d'adpater l'acclération dans diverses situations dont notamment
    #les virages serrés, les lignes droites ou une legere curvature
    liste_virage=self.anticipeVirage(obs)
    acceleration = 0.7
    if len(liste_virage) < 1 :  # s'il n'y a pas de virage
        acceleration = 0.75  # conduite normale on pourrait augmenter légèrement l'accélération -> à décider
    else :
        proche_virage = liste_virage[0]
        curvature= proche_virage["curvature"]

        if curvature > 1.0: # virage serré
            acceleration= acceleration - 0.25
            drift = True
        elif curvature > 0.85 and curvature <= 1.0:  #virage moyen
            acceleration = acceleration - 0.15
            drift = False
        else :
            acceleration = acceleration - 0.05
            drift = False
    acceleration = np.clip(abs(acceleration),0.7,1.0)
    return acceleration,drift
```

De plus, les conditions imposées étaient incohérentes, il était donc préférable de donner un intervalle pour que la condition soit vraie. Par le code ci-dessus, le kart sortait instantanément de la piste.
Nous commençons donc la course avec une accélération égale à 0.90. En fonction des virages présents sur la piste , cette accélération diminue. Lors de la première course lancée, à un virage très serré, le kart sort de la piste. Or à partir de la deuxième course, notre kart est toujours premier (déduction faite après 9 courses).
À noter
Dans ce code, nous avons voulu rajouter une variable locale drift qui , en fonction des virages , va donner là possibilité au kart de prendre plus stratégiquement les virages avec un boost d’accélération à la fin du drift. Cependant les drifts doivent être travaillés en profondeur et en cohérence avec les valeurs de retour de la fonction **anticipeVirage**.

Fonction pour corriger la trajectoire du kart par rapport au centre:

```python
def coorection_centrePiste(self, obs):
    """
    Calcule la correction nécessaire pour rester au centre de la piste.
    """
    #center_path contient le vecteur vers le centre de la route
    center_path = obs.get('path_start', np.array([0, 0, 0]))

    dist_depuis_center = center_path[0]

    correction = dist_depuis_center * 0.5

    return np.clip(correction, -1.0, 1.0)
```

Objectif de cette fonction:

Cette fonction a pour but de corriger la trajectoire du kart en le redirigeant au centre de la piste lorsque le kart dévie de la piste en roulant dangereusement les limites (bordures) de la route. Il aide au maintien du kart sur la piste.

Pour implémenter cette fonction nous avons d'abord établi un algorithme. Il est composé de 5 étapes.

ALGORITHME

#RÉCUPÉRER LES DONNÉES DES CAPTEURS

#On regarde l'étiquette 'path_start' dans les observations.
#Si elle est vide, on utilise une valeur par défaut [0, 0, 0] pour ne pas planter.

#IDENTIFIER LA POSITION LATÉRALE

#On extrait la première valeur du vecteur = X (cette valeur nous dit à quelle distance nous sommes du centre de la route)
- Si X est positif : le centre est à DROITE.
- Si X est négatif : le centre est à GAUCHE.

#CALCULER LA FORCE DE CORRECTION

#On multiplie cette distance par un coefficient
#Cela permet d'avoir un volant progressif :
- Petite déviation = petit coup de volant.
- Grosse déviation = gros coup de volant.

#APPLIQUER LES LIMITES PHYSIQUES

#On force le résultat à rester entre -1.0 et 1.0 (=> le volant du kart ne peut pas tourner à l'infini)

#ENVOYER L'ORDRE AU VOLANT

#On retourne la valeur finale pour qu'elle soit ajoutée à la direction.

Pour la ligne `center_path = obs.get('path_start', np.array([0, 0, 0]))`

- **si** il y a 'path_start' (= contient les coordonnées du début de la route juste devant le kart) alors center_path='path_start'
- **sinon** center_path est un vecteur par défaut mit a [0,0,0] jusqu'à qu'il retrouve le centre. ex: si le kart vient de tomber dans le vide et ne voit plus la route

Ensuite, la ligne `dist_depuis_center = center_path[0]`

- dist_depuis_center contient le point X qui représente le décalage (gauche/droite) par rapport au centre, prenons quelques exemples:
  - si le chiffre est **0**: le kart est au milieu.
  - si le chiffre est **positif** prenons 2.0: Le centre est à 2 mètres sur la droite de l'agent
  - si le chiffre est **négatif** prenons -2.0: Le centre est à 2 mètres sur la gauche de l'agent

Pour continuer, c'est cette ligne correction = distance_from_center * 0.5 qui va imposer la force avec laquelle on va tourner le volant, exemples:

- si le kart est à 2 mètres du centre: `2.0*0.5=1.0` => l'agent va tourner le volant au maximum de 1.0 pour revenir vite
- si le kart est seulement à 0.2 mètre, qui est un petit écart: `0.2*0.5=0.1` => l'agent va tourner le volant très doucement pour se replacer

=> Cette ligne permet à la conduite du kart d'être bien droit, moins de "zigzag" et ne pas avoir de conduite brusque.
La valeur multiplicateur 0.5 est un compromis qui semble correct, 1.0 n'aura aucun impact sur la correction et 0.1 serait trop faible pour une bonne correction.

Pour finir, la ligne return np.clip(correction, -1.0, 1.0)

- np.clip est une barrière de sécurité c'est à dire qu'elle permet de vérifier que la correction ne dépasse pas l'intervalle (-1.0,1.0) => c'est les limites physiques du volant, car un volant ne tourne pas infiniment
- On peut représenter cela de la façon suivante:

$$f(x) = \begin{cases} -1.0 \text{ si } x < -1.0 \\ 1.0 \text{ si } x > 1.0 \\ x \text{ sinon} \end{cases}$$

Nous utilisons cette fonction dans la fonction choose_action:

```python
# CALCUL DE LA CORRECTION POUR RESTER AU CENTRE DE LA PISTE
correction_piste = self.coorection_centrePiste(obs)

# COMBINAISON DE LA DIRECTION DU CHEMIN ET DE LA CORRECTION DE PISTE
final_steering = np.clip(steering + correction_piste, -1, 1)

# ADAPTATION DE L'ACCELERATION SELON LE VIRAGE POUR NE PAS SORTIR DE LA PISTE
acceleration = 0.8 if abs(final_steering) < 0.3 else 0.4

action = {
    "acceleration": acceleration,
    "steer": final_steering,
    "brake": False,
    "drift": False,
    "nitro": False,
    "rescue": False,
    "fire": False
}
```

On combine la direction du chemin (qui à déjà été implémentée précédemment) ainsi que la correction de piste.

Supplémentaire:
La ligne, acceleration = 0.8 if abs(final_steering) < 0.3 else 0.4

- Ici on utilise abs() (= valeur absolue)=> transforme les nombres négatifs en positifs.
- si on tourne à droite à 0.5 ou à gauche à -0.5, abs donne 0.5 => car peu importe si on tourne à gauche ou à droite, ici ce qui nous intéresse c'est de savoir à quel point on tournes fort.
- Si le volant est à moins de 30%, on considère que le kart est en ligne droite ou dans un virage très léger
  - => si la condition est vraie on met l'accélération 80% de puissance.
  - => sinon (volant est braqué à +30%) on met l'accélération 40% de puissance.

Ce sont des valeurs qui peuvent être modifiées, en fonction des observations de l’équipe. Pour l’instant, ces valeurs semblent être plutôt correctes.

Enfin, on remplace la valeur des clés “acceleration” et “steer” dans le dictionnaire action par les variables acceleration et final_steering.

Caractéristiques du drift:

- Le drift permet de prendre des virages plus serrés tout en conservant un maximum de vitesse.
- La vitesse diminue pendant le drift à cause de l'augmentation de la friction.
- Si l'inertie est trop forte, le véhicule sort de la piste.
- Maintenir un drift permet d'obtenir un bonus d'accélération.

Problématique (Le dilemme):

- Un drift à haute vitesse risque de causer une sortie de piste, tandis qu'un drift à basse vitesse entraîne une perte de temps importante.

Solutions proposées:

- Comme le kart se tourne vers l'intérieur du virage pendant un drift, il est préférable de commencer le drift plus à l'extérieur de la courbe.
- Juste avant de sortir de la piste, il faut relâcher brièvement la direction ou effectuer un contre-braquage rapide.

Philosophie du code ci-dessous:

```
def choose_action(self, obs):
  accel = 0.7
  drift = False
  if self.recovery_steps > 0:
        self.recovery_steps -= 1
        return {
            "acceleration": 0.0,
            "steer": 0.0,
            "brake": True,
            "drift": False,
            "nitro": False,
            "rescue": False,
            "fire": False,
        }
  velocity = np.array(obs["velocity"])
  speed = np.linalg.norm(velocity)
  phase = obs["phase"]
  nodes_path = obs["paths_start"]
  if phase > 2:
        if speed < 0.2:
            self.stuck_steps += 1
        else:
            self.stuck_steps = 0
  if self.stuck_steps > 7:
        self.recovery_steps = 15
        self.stuck_steps = 0

  if len(nodes_path) > self.path_lookahead:
        target_node = nodes_path[self.path_lookahead]
        angle = np.arctan2(target_node[0], target_node[2])
        steering = np.clip(angle * 2, -1, 1)
  else:
        steering = 0
  if abs(steering) > 0.8:
        drift = True
        accel=0
  action = {
        "acceleration": accel,
        "steer": steering,
        "brake": False,
        "drift": drift,
        "nitro":False,
        "rescue":False,
        "fire": False
        }
  return action
```

- La priorité absolue est de terminer la course. La conception se limite aux fonctions de base : avancer, tourner et éviter les blocages.

- Accélération : Accélérer en continu, sauf en cas de problème.

- Gestion du blocage (Stack) : Si le compteur de blocage dépasse un seuil, arrêter d'accélérer, freiner et reculer.
    - Si la vitesse est trop faible, augmenter le compteur de blocage.

- Virage : Si l'angle entre le kart et le prochain nœud est trop grand, déclencher un drift.

Explication du code:

- velocity (Vecteur vitesse) : Récupère la direction et la vitesse actuelle du kart sous forme de vecteur (X, Y, Z).

- speed (Vitesse scalaire) : Calcule la « norme » du vecteur pour obtenir une valeur scalaire.

- nodes_path (Chemin de nœuds) : Récupère la liste des coordonnées définissant la trajectoire à suivre sur le circuit.

- np.arctan2 : Calcule l'écart angulaire (en radians) entre le nœud cible et la position actuelle du kart.
    - target_node[0] : Décalage gauche/droite.
    - target_node[2] : Distance avant/arrière.

Points à améliorer:

- Le programme d'évitement des blocages est trop primitif et cause des pertes de temps. Une optimisation est indispensable.

- Le système de drift est primitif aussi: il ne permet pas de maintenir une vitesse élevée sans que le kart ne soit aspiré vers l'intérieur du virage.