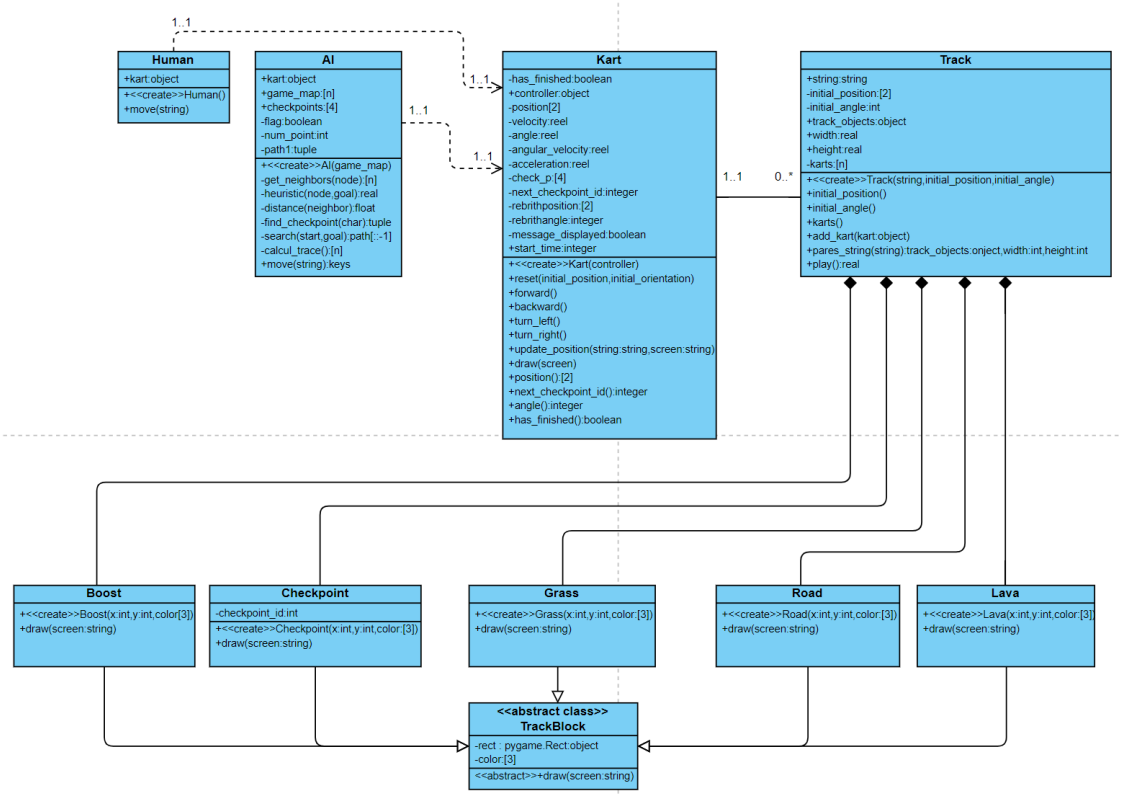

Justifications：

les relations:

Human / AI avec Kart : dépendance

la relation entre les classes Human/AI et Kart est représentée par une dépendance. Cela signifie que les classes Human/AI utilisent des objets de la classe Kart dans leurs méthodes, par exemple pour déplacer le kart via la méthode move.

1.Cependant, les classes Human/AI ne possèdent pas d'instance de la classe Kart comme attribut permanent.

2.La dépendance indique également que tout changement dans la classe Kart pourrait nécessiter des ajustements dans les classes Human/AI, puisque leur comportement dépend des interfaces de la classe Kart.

3.Cette relation est unidirectionnelle ; les classes Human/AI doivent connaître la classe Kart pour fonctionner, mais la classe Kart n'a pas besoin de connaître qui la contrôle, que ce soit un Human ou un AI.

Kart avec Track : Association

la relation entre les classes Kart et Track est représentée par une association.

1.Cela signifie que la classe Track contient des références à un ou plusieurs objets de la classe Kart, comme indiqué par les multiplicités '1..1' et '0..*'.

Chaque instance de Kart est associée à une instance de Track, et une instance de Track peut être associée à zéro ou plusieurs instances de Kart.

2.Cette association est bidirectionnelle, car chaque Kart doit se situer sur un Track, et chaque Track peut contenir plusieurs Karts en compétition.

3.L'association indique également que la classe Track peut être responsable de la gestion des objets Kart, par exemple en les ajoutant au circuit via la méthode add_kart, tandis que les objets Kart peuvent nécessiter l'accès à des informations sur le Track pour leur fonctionnement.

Track avec les classes Road, Grass, Lava, Checkpoint et Boost : composition

la relation entre la classe Track et les classes Road, Grass, Lava, Checkpoint et Boost est représentée par une composition.

1.Cela signifie que la classe Track est responsable de la création et de la gestion des instances de ces classes, qui constituent les différentes parties du circuit de course.

2.Les instances de ces classes ont une dépendance de cycle de vie avec la classe Track : si l'objet Track est détruit, les objets Road, Grass, Lava, Checkpoint et Boost qu'il contient sont également détruits car ils n'ont pas de raison d'être en dehors du circuit.

3.La composition indique également que l'objet Track est le propriétaire exclusif de ces parties composantes, qui ne sont pas partagées avec d'autres objets et sont inhérentes à la structure du circuit.

Track avec les classes Road, Grass, Lava, Checkpoint et Boost : class abstraite/ inheritance/ polymorphisme

'TrackBlock' représente une **classe abstraite** qui définit une **interface** commune pour plusieurs sous-classes. Elle établit des **méthodes abstraites** comme **draw** et des attributs de base tels que **rect** et **color**. Cette **classe abstraite** sert de contrat stipulant que toutes les sous-classes qui en **héritent** doivent implémenter les méthodes définies.

Les classes **'Road', 'Grass', 'Lava', 'Checkpoint'** et **'Boost'** sont des sous-classes de **'TrackBlock'** et incarnent la notion de **polymorphisme**. Chacune de ces sous-classes implémente la méthode **draw** de **manière spécifique**, permettant ainsi des **comportements différents** lors de leur invocation, bien qu'elles partagent la même signature de méthode. Ce concept de polymorphisme permet aux objets de se comporter différemment selon leur type réel, bien qu'ils soient traités comme des instances de '**TrackBlock**'. Cela facilite l'extension et la modification des comportements des objets sans affecter les autres parties du code qui utilisent ces objets comme des instances de la classe abstraite '**TrackBlock**'.

encapsulation

Pour la classe **TrackBlock**, étant donné que **__rect** est une instance de **pygame.Rect**, elle

est utilisée directement dans la classe et n'est pas accessible de l'extérieur, il en va de même pour **__color**. Concernant les sous-classes, tous les attributs sont privés, mais la méthode **draw** doit être publique.

Dans la classe **Kart**, l'instruction **self.controller.kart = self** permet à l'objet contrôleur d'accéder directement et de contrôler l'instance de **Kart** associée, lorsque cela est nécessaire. Tous les attributs peuvent être privatisés, à l'exception de controller qui est passé de l'extérieur. Les attributs **__position, __next_checkpoint_id et __angle** doivent être utilisés dans la classe AI, et **__has_finished** doit être utilisé dans la classe **Track**, donc des fonctions d'accès **getter** sont nécessaires pour les retourner. Il n'est pas nécessaire d'utiliser des fonctions de modification **setter** pour définir les valeurs des variables dans la classe **Kart**, car toutes les méthodes de **Kart** doivent être contrôlées par d'autres classes.

Dans la classe **AI**, tous les attributs peuvent être privatisés et n'ont pas besoin de getters pour transmettre des informations à l'extérieur, car ils sont utilisés pour le calcul de l'IA. En ce qui concerne les méthodes de la classe **AI**, seule la méthode **move** doit être définie comme publique, les autres méthodes peuvent être définies comme privées et n'ont pas besoin de transmettre des paramètres à l'extérieur.