Projet MINI_POO – 2023_S2

QUALITES ATTENDUES DU PROJET MINI_POO

Fiabilité : Il doit donner les résultats corrects attendus : déclarer les variables même si Python le fait dans certains cas, ce qui évite certaines surprises et des débogages laborieux…

Robustesse : Il doit gérer les erreurs évidentes de manipulation des utilisateurs et tester les types des arguments.

Convivialité : Il doit être agréable à utiliser (souris, icônes, menus…), et facile à prendre en main (le scénario d'utilisation semble « logique » à l'utilisateur).

Efficacité : Il doit donner des réponses claires et dans un temps raisonnable.

Compacité : Il doit occuper une place modérée en mémoire et sur le disque !

Lisibilité : Il doit être structuré en classes, fonctions et procédures toutes commentées et présentées clairement. Les types doivent être précisés dans les déclarations. Vos variables doivent être explicites. N'hésitez pas à développer vos propres classes pour plus de lisibilité !

Portabilité : Il doit être aisément transférable sur une machine d'un autre type (attention aux widgets ou aux librairies spécifiques, aux chemins d'accès codés spécifiquement pour votre machine…).

Flexibilité : Une partie de votre travail doit pouvoir être aisément utilisable par d'autres applications. Pour cela nous vous préconisons de dissocier fortement vos interfaces de vos traitements et de généraliser vos fonctions et procédures et bien entendu de travailler en utilisant le paradigme de la programmation orientée objet !

Et enfin : **Privilégiez le bon fonctionnement du programme à son aspect !** Le résultat n'est pas le seul point évalué : votre autonomie, votre travail personnel, l'originalité et la bonne exécution de vos solutions, le respect des consignes… sont d'autant d'éléments qui participent à votre évaluation !

Attention ! Le présent document peut être amené à subir des modifications, vérifiez fréquemment sur l'espace SAVOIR que vous travaillez bien sûr la dernière version de ce document !

Edition	Nature de l'évolution	Évolution	Date
V1.0	Création	Première rédaction complète	25/02/2024
V2.0	Ajout d'une classe	Ajout de la classe tronçon + équations	17/03/2024
V2.1	Clarifications	Ajout de détails sur les dépassements	19/04/2024
V2.2	Clarifications	Ajout de détails sur les puits	24/04/2024
V2.3	Corrections	Corrections sur le $\Delta_{overtake}$	29/04/2024
V2.4	**Clarifications**	**Ajout de détails sur les dépassements**	**22/05/2024**

SIMULATION D'UN WEEK-END DE POORMULA 1

★★☆ Algorithmes de traitement
★☆☆ Interface
★☆☆ Outils non vus en cours

On souhaite modéliser le déroulement d'un Grand Prix (GP) de POOrmula One (P1). Dans ce projet, les acteurs d'un WE de GP se côtoient (pilotes, commissaires …), les voitures de P1 évoluent sur un circuit et différents événements surviendront au cours de la course et doivent être intégrés au bon déroulement de l'exécution du programme.

Modélisation du circuit et de ses composants

Dans ce projet, on désire modéliser le **circuit** de manière simplifiée. Sur le circuit, où la compétition se déroule, des **voiture**s circulent pour un nombre limité et fixé de tours et des **commissaire**s (qui sont des **acteur**s **du WE**) y officient. Le circuit est découpé en **tronçon**s, chaque tronçon possède un point de départ et un point d'arrivée (dont les coordonnées cartésiennes sont données en kilomètre). Le point de départ du premier tronçon constitue la ligne de départ/arrivée ; il en va de même pour le point d'arrivée du dernier tronçon. Ces points seront reliés entre eux par de simples droites, voir figure 1. Chaque tronçon est caractérisé par sa cambrure : lent ; moyen ; rapide. Cette caractérisation est importante, car plusieurs propriétés en découlent.

Dans le cas du circuit, pour chaque type de tronçon (lent, moyen, rapide), un delta de temps, $\Delta_{overtake}$, est défini en seconde. Ce delta de temps permet de savoir si un dépassement est possible entre deux pilotes de POOrmula One. Si le temps entre les deux concurrents est supérieur à ce delta alors le dépassement n'est pas possible, sinon il faudra connaitre la possibilité qu'a un pilote de dépasser son adversaire (expliqué dans la description de la classe **pilote**). Cependant, deux pilotes ne peuvent pas se dépasser si l'un d'eux change de tronçon (*c.-à-d.* pour se dépasser, les pilotes doivent être sur le même tronçon), *sauf* s'il passe la ligne de départ arrivée. Finalement, chaque circuit à un temps de passage par les puits ; ce temps, en seconde, est indépendant des voitures, des équipes ou des pilotes. Le passage par les puits (ou stand) est nécessaire pour changer les pneus de la voiture. Nous considérons aussi que le temps d'immobilisation de la voiture est identique entre toutes les équipes. De plus, le début et la fin des puits seront indiqués dans le fichier *.xml* par « sortie_puits » et « entree_puits ». Elles représentent l'entrée et la sortie des puits via un pourcentage de la longueur du tronçon par rapport à son commencement. Lorsqu'un pilote est dans les puits, alors il se fait dépasser par les autres concurrents en piste.

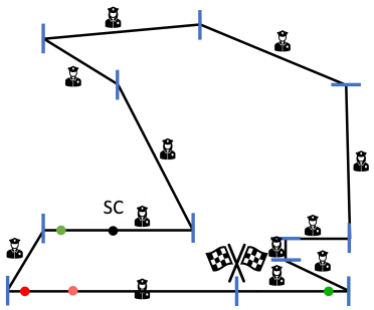

Figure 1 : Exemple de modélisation d'un circuit et des acteurs qui y sont présents (commissaires, pilotes de P1 et de SC).

Les sections suivantes détaillent chacun des éléments ; de leurs caractéristiques ; ainsi que leurs interactions.

Modélisation des voitures

Deux types de **voiture**s sont modélisés : des voitures de course, les POOrmula One (P1) ; et la voiture de sécurité (Safety Car [SC]). On ne modélise ni la voiture médicale ni les différents engins pouvant se trouver en piste ; on considère qu'en cas d'accident, les pilotes et voitures accidentées sont pris en charge et retirés automatiquement de la course. La mécanique derrière la gestion de la SC est présentée plus tard dans le document. Les deux types de voitures partagent cette caractéristique : la vitesse moyenne par type de tronçon. On ne considère pas de profil de vitesse (en km/h) dans chaque tronçon. Il est imaginable que ces données proviennent de simulations numériques (*p. ex.* 1D).

Cette caractéristique est complétée par les propriétés suivantes, pour la SC elles ne sont pas clairement définies, ou n'ont pas d'intérêts dans les règles qui sont imposées à cette modélisation. Ainsi, les P1 ont ces propriétés supplémentaires :

- L'équipe de P1, puisqu'il y a deux voitures par équipe.
- Le nom ou le modèle de la voiture de P1,
- La fréquence en nombre de tours d'arrêt aux puits.

Noter que la fréquence d'arrêt est décrite par un intervalle de valeur entière $[f_1, f_2]$ qui est donnée dans le fichier xml. Après chaque arrêt et au début de la course, un nombre est tiré aléatoirement dans cet intervalle, pour chaque voiture. Ce nombre est indépendant du style de conduite du pilote, et il dépend uniquement de la voiture. De plus, il n'y a qu'une seule qualité de gomme pour tous les concurrents, normalement interchangeable lors de l'arrêt, ainsi aucune modélisation de stratégie de course n'est proposée. À titre d'information, les gommes dites tendres sont, en général, plus rapides pour un nombre faible de tours, les gommes dures sont plus lentes, mais durent plus longtemps.

Modélisation des acteurs du WE

Les acteurs du WE sont au nombre de trois : les commissaires, le directeur de course et les pilotes. On souhaite que chaque acteur puisse se présenter et que cette présentation apparaisse dans un fichier de log ou à défaut dans la console. Par ailleurs, l'ensemble des acteurs partage un nom et un prénom.

Pilotes

Le pilote est un acteur du WE. Il conduit une P1 ou une SC. Dans le cas de la P1, deux pilotes partagent une même équipe. Le pilote, en plus de son nom et de son prénom, à une nationalité et un surnom. Afin de pouvoir suivre un pilote, sa position en piste doit être connue ; elle peut être donnée soit avec les coordonnées cartésiennes ou sa position dans le tronçon. La position du pilote (d'une SC ou d'une P1) est importante, car elle permet de savoir si :

- Le pilote se trouve dans le delta de temps pour pouvoir dépasser son adversaire direct,
- Si le pilote peut dépasser un concurrent en cas d'accident en piste.

Noter que le pilote de la SC n'est jamais dépassable ! De plus, si un accident survient, elle commence son tour à la sortie des puits. Les pilotes de P1 doivent en tout temps rester derrière la SC. De plus, lorsque la SC est déployée, les pilotes ne peuvent pas se dépasser entre eux, même pour un drapeau

bleu (l'utilisation de ces drapeaux est expliquée dans le texte). De même, lorsqu'un drapeau jaune est déployé, ce dernier est actif dans la zone sous contrôle d'un commissaire de piste, *c.-à-d.* dans le tronçon où il est affecté. Si le pilote se trouve dans ce tronçon et si le commissaire agite le drapeau, alors il ne peut pas dépasser ni être dépassé.

Le pilote de P1 a plusieurs propriétés supplémentaires par rapport au pilote de la SC. Ainsi, en plus de sa position en piste, il doit avoir son rang parmi les autres compétiteurs, ce qui n'est pas nécessaire pour le pilote de la SC, car il ne participe pas à la course ! De même, le pilote de P1 possède un numéro. Noter que la position initiale des pilotes est déterminée comme suit : le meneur est placé sur la ligne de départ/arrivée, les suivants sont séparés par 6 mètres.

Ce paragraphe traite des drapeaux bleus. Les pilotes les plus lents peuvent être dépassés par les meneurs de la course. Cela arrive lorsque le delta de temps entre le meneur et le retardataire est inférieur à 1 seconde. Dans ce cas, le **Directeur de Course** (DC) ordonne aux commissaires de déployer un drapeau bleu. Une fois le drapeau bleu agité, le retardataire laisse passer l'autre voiture. Sans commissaires, pas de drapeau bleu, et donc pas de dépassement possible via cette règle !

En plus, chaque pilote de P1 dispose d'un certain nombre de paramètres caractérisant ses performances. La plupart sont tirées aléatoirement et uniformément dans un intervalle à partir de paramètres définis dans le fichier XML fourni.

Le premier est la variation de performance $\delta_{perf} \in [d_1, d_2]$ propre à chaque pilote. L'intervalle de performance d'un pilote est unique et une nouvelle valeur est tirée d'une loi uniforme ($\mathcal{U}[d_1, d_2]$) à l'entrée dans chaque nouveau tronçon. Si la valeur tirée est inférieure à 0, alors le pilote sera plus lent que ce que « peut » la voiture avec un pilote « moyen » à son bord, inversement, si cette valeur est supérieure à 0, alors le pilote sera plus rapide.

Il est possible de connaitre la position, avec les coordonnées cartésiennes, du pilote P_k à l'itération $i+1$ pour un tronçon j :

$$\blacksquare_{P_k} pos^{i+1}_{tronçon_{j|j+1}} = \blacksquare_{P_k} pos^{i}_{tronçon_j} + \left(1 + \delta^{P_k}_{perf}\right) v_{tronçon_j} \delta t \times \begin{bmatrix} \cos \alpha_j \\ \sin \alpha_j \end{bmatrix} \quad 1.1$$

On notera par la suite $v^{P_k}_{tronçon_j} = \left(1 + \delta^{P_k}_{perf}\right) v_{tronçon_j}$ la vitesse d'un pilote P_k sur le tronçon j ; δt est donné à 0.01 sec ; enfin, α_j est l'angle du tronçon j. Noter qu'il est possible que la mise à jour de la position soit reportée au tronçon $j+1$; dans ce cas, il faudra reporter la distance restante.

Le second est lié à la possibilité pour un pilote de doubler son adversaire. Avant de savoir si un dépassement est réussi, trois conditions s'appliquent : le pilote dépassant doit être plus rapide sur le tronçon que le pilote dépassé ; la différence de temps entre les deux voitures doit être inférieur au $\Delta_{overtake}$; les deux pilotes doivent être dans le même tronçon. Afin de vous aider à gérer les cas où plusieurs pilotes se dépassent en même temps, la Figure 2 vous est proposée. On considéra que le pas de simulation est suffisamment fin pour éviter les chevauchements, c.-à-d. si un pilote de rang supérieur passe devant un pilote de rang inférieur sans être passé par les différentes conditions, juste avec la mise à jour des positions. Si vous constatez des chevauchements, réduisez le pas de simulation par 10 et relancez la simulation.

La différence de temps entre deux pilotes (avec le pilote P_k devant le pilote P_{k+1}), à condition que les deux voitures se trouvent sur le même tronçon, se calcul de cette manière :

$$\Delta t_{P_{k+1}-P_k} = \frac{\left\| {}^{P_{k+1}}pos^{i}_{tronçon_j} - {}^{P_k}pos^{i}_{tronçon_j} \right\|_2^2}{v^{P_{k+1}}_{tronçon_j}} \qquad 1.2$$

Pour modéliser le dépassement, des nombres seront tirés aléatoirement en obéissant à une loi uniforme comprise entre 0 et 1 ($\mathcal{U}[0,1]$). Si le nombre tiré dépasse le seuil de performance en dépassement du pilote, $h_{overtake}$, alors on considère que le pilote peut dépasser son adversaire. Plus le seuil est bas, plus le pilote a des facilités à dépasser son adversaire. Cette règle ne s'applique pas si un drapeau bleu est agité, car dans ce cas $h_{overtake} < 0$ (obligation de laisser passer) ; de même $h_{overtake} > 1$, s'il y a un drapeau jaune (interdiction de doubler).

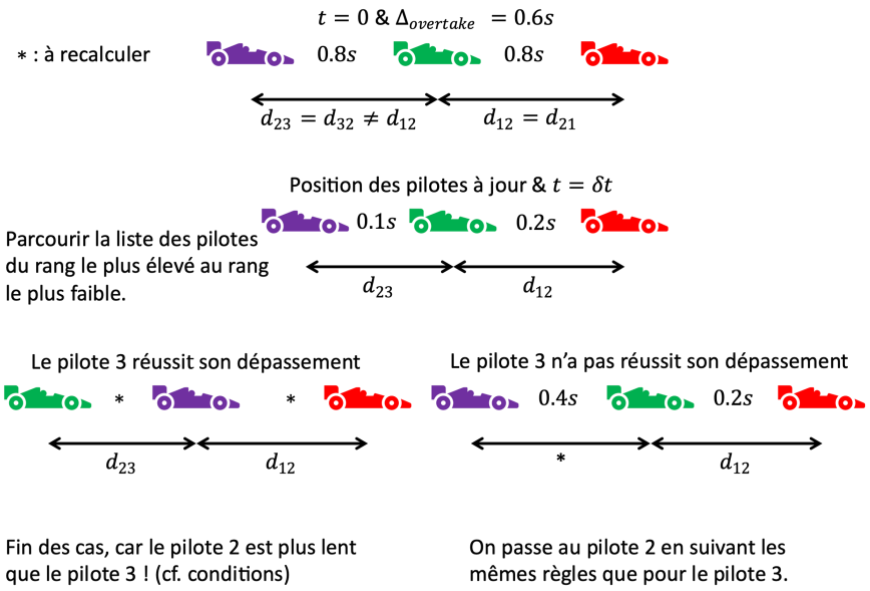

Figure 2 : Schéma illustrant la mise à jour des pilotes dans le cas des dépassements.

Pour savoir si le dépassement est réussi, un autre seuil est défini : la probabilité d'avoir un accident si le pilote est dépassé ou s'il dépasse. Elle se conçoit de la même manière que le cas précédent, chaque pilote possède un seuil de provoquer un accident : h_{crash}. Si ce seuil est dépassé lors de la génération d'un nombre aléatoire, pour l'un des pilotes, alors il y a un accident. Cette probabilité ne s'applique pas en cas de drapeau bleu ou dans le cas de tronçons sous drapeau jaune, ici $h_{crash} > 1$ pour les deux pilotes. S'il un accident survient, un drapeau jaune est déployé automatiquement dans le secteur sous contrôle du commissaire de piste. C'est au directeur de course que revient la décision de déployer une SC. De même, c'est lui qui décide lorsqu'un drapeau jaune ou la SC s'arrête.

Noter que s'il y a un accident, c'est le pilote qui provoque l'accident qui est retiré du jeu, cela peut être vrai pour les deux pilotes lors de la manœuvre. De plus, si le pilote n'est pas hors course, alors il perd 15 secondes dans la manœuvre. En cas de dépassement réussi, la position des deux pilotes est interchangée. Si le dépassement n'aboutit pas et ne provoque pas d'accident, alors le pilote qui effectue la manœuvre de dépassement perd 0.3 seconde (voir Équation 1.2.).

Noter qu'un nouveau nombre aléatoire est tiré avant chaque tentative de dépassement. Pour TOUS les seuils présentés dans cette section.

Directeur de course

Comme décrit plus haut, c'est le directeur de course qui décide de déployer la SC. Ce choix, est modélisé de la même manière à l'aide du seuil h_{SC}. À chaque accident, un nombre est tiré aléatoirement et la SC est déployé ou non suivant sa valeur et h_{SC}. Ce nombre est divisé par deux si les deux pilotes sont accidentés et sortent de la course.

Finalement, pour chaque accident, le directeur de course choisit la durée sous laquelle la SC ou le drapeau jaune sont actifs. Cette durée est tirée aléatoirement entre t_1 et t_2 via une loi uniforme ($\mathcal{U}[t_1, t_2]$).

Commissaire

Les commissaires de piste sont en contact direct avec le directeur de course. C'est eux qui agitent les drapeaux (bleu ou jaune). Ils possèdent une affectation par tronçon, il est possible que des tronçons soient laissés sans commissaires ; empêchant ainsi les pilotes de connaître l'état des drapeaux en piste.

Boucle de simulation – structure

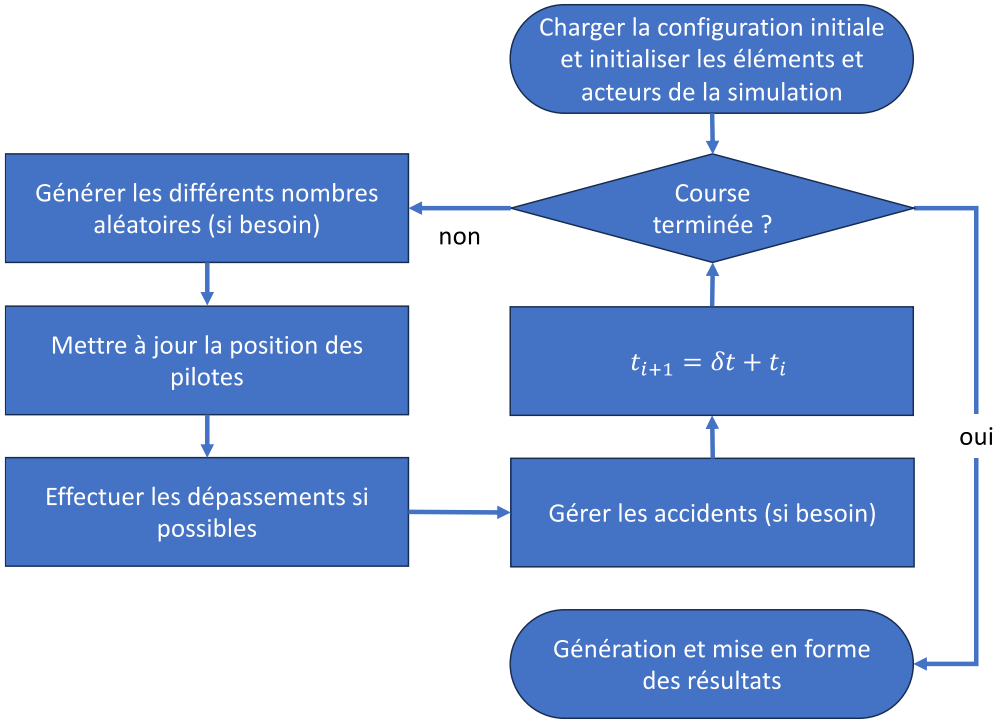

Figure 3 : Boucle de simulation définie d'un point de vue macroscopique.

Afin de gérer la partie simulation, permettant de gérer la dynamique de la course, nous vous proposons la Figure 3. La boucle de simulation, définie de façon macroscopique, à préciser, détailler puis implémenter. Afin de rendre la simulation plus réaliste, un pas de simulation est défini par la valeur $\delta t = 0.01$ sec, voir l'équation 1.1. Le temps est mis à jour avant chaque itération. La course s'arrête lorsque tous les pilotes ont franchi la ligne départ/arrivée et quand le drapeau à damier est agité. Le drapeau à damier est agité lorsque le meneur de la course est dans le dernier tour et approche de la

ligne de départ/arrivée pour la dernière fois. Comme tous les drapeaux, c'est au commissaire d'exécuter cette tâche. Ainsi, les retardataires, ceux qui se sont fait dépasser par le meneur une ou plusieurs fois, peuvent faire moins de tours.

C'est donc dans la phase d'initialisation que la position initiale des pilotes est définie avec tous leurs attributs et propriétés, que les commissaires sont positionnés et que les différents objets sont instanciés. Le but de la simulation est donc de voir puis d'analyser comment évolue ce système pendant un nombre choisi de tours. Un compte rendu de la course est attendu, avec l'intégralité des événements (drapeaux, SC, …), sous format d'un fichier log construit comme vous le souhaitez.

Travail à réaliser

Cette section propose une décomposition du travail à réaliser. Certaines étapes sont indépendantes des autres. Les mots écrits en gras dans cet énoncé doivent apparaitre dans votre programme sous la forme d'une classe (en cas de doute, demandez à l'enseignant). Vous pouvez en ajouter d'autres si vous en avez besoin ou si vous les jugez nécessaires ou utiles. Il est attendu un code respectant les bonnes pratiques (nommage et portées des variables, typages des méthodes et fonctions, usage des accesseurs/mutateurs, exploitation de l'héritage…). Bien entendu votre programme doit utiliser au plus et au mieux l'approche orientée objet (maximiser le nombre de fonctions dans les classes plutôt que dans le __main__).

Les principales fonctionnalités attendues du programme sont :

- Concevoir l'ensemble des classes, leurs attributs et méthodes permettant de décrire une configuration de GP et ses constituants tout d'abord, d'en charger une depuis l'un des fichiers XML fournis.
- Réaliser la gestion et la simulation de la course, en considérant dans un premier temps que les dépassements et accidents ne sont pas possibles, donc ni drapeaux jaunes ni SC. Pour cela le fichier XML gp_1.xml est disponible, il vous propose un cas d'étude à tester pour vérifier le bon fonctionnement de votre programme.
- Dans le second cas, on se propose de gérer les dépassements. Notamment dans le premier et dernier tronçon. À cette fin, le fichier gp_2.xml vous propose un exemple de simulation à réaliser mettant l'accent sur la dynamique des dépassements entre les pilotes, sans les accidents.
- Terminer par la gestion des accidents. C'est le fichier gp_3.xml et les suivants qui supportent la description de ces cas plus complets.
- Réaliser une interface utilisateur permettant de réaliser les fonctionnalités précisées dans la section « interface utilisateur ».
- Afin de pouvoir suivre le bon comportement de votre programme, vous pouvez mettre en place un log (dans la console, voire dans un fichier) qui décrit à chaque pas de simulation de qui a été fait par le simulateur.

Toutes les initiatives peuvent être intéressantes. Si vous souhaitez en prendre, n'hésitez pas à en parler à l'enseignant pour avis et validation avant de débuter de longs développements inutiles !

Interface utilisateur

L'interface est à réaliser lorsque la structure objet est fonctionnelle (vous avez ainsi testé toutes les classes et les relations possiblement admissibles à l'aide de tests unitaires que vous avez définis et de fonctions générant des cas tests). Cette interface doit permettre de :

- Charger une configuration de GP (configuration spatiale des pilotes, ainsi que de leurs voitures) préalablement enregistrée au format XML et fournie sur l'espace SAVOIR du module MINI_POO.
- Permettre le lancement de la simulation avec ou sans l'affichage des différents pas de simulation. Pour la partie affichage, rester sur des solutions simples, comme montré à la Figure 1.
- Afficher (puis enregistrer dans un fichier XML de résultat dont vous définirez le format) l'évolution des pilotes au cours de la course, les accidents, les sorties de SC, etc.

N'hésitez pas à exploiter l'ensemble des widgets disponibles dans la librairie `tkinter` et ses sous-librairies. Il n'est pas demandé d'avoir recours à des solutions plus complexes de représentation 3D (`pygame` ou autres moteurs 3D). N'oubliez pas que la fonctionnalité est plus valorisée que l'esthétique dans l'évaluation finale du projet.

N'oubliez pas non plus qu'il est possible de définir des classes pour effectuer la gestion des interfaces comme vues en ED.

Conseils et méthodologie de travail

Dans le contexte de conception et d'implémentation en groupe, le travail en binôme peut-être complexe à mettre en place : il est donc nécessaire d'adopter une méthode de travail efficiente et rigoureuse. Pour cela :

- **Travaillez de façon continue**, évitez le stress d'une réalisation en dernière minute (oubliez la nuit de la POO) ! Vous avez plusieurs **séances encadrées**, **mettez-les à profit** en arrivant avec des questions et/ou des propositions ! Venez à la première séance en ayant au moins lu cet énoncé !
- **Définissez ensemble** le **diagramme de classes** que vous souhaitez implémenter à partir de la lecture attentive de l'énoncé. Cherchez à ce que ce diagramme soit le plus détaillé possible (les attributs doivent être typés, les méthodes également (arguments et retours)). Pour travailler sur le même modèle, vous pouvez utiliser le site https://app.diagrams.net/ (choisir « UML class diagram ») pour concevoir de façon collaborative ce modèle structurant commun. N'y passez pas plus d'une séance !
- **Répartissez-vous l'implémentation de la structure des classes**. Par structure, on entend : l'ajout de la classe et d'éventuelles relations d'héritage, le constructeur, la définition et le typage des attributs, la gestion des pointeurs, les propriétés (et donc par extension : les accesseurs et mutateurs) et l'ajout des méthodes définies dans le diagramme de classe. Pour ces méthodes, il n'est pas encore question de rédiger tout le code, mais uniquement la déclaration, la documentation (avec les symboles ''' en début de fonction) et les vérifications des types des arguments passés. N'oubliez pas que vous pouvez définir une fonction ou une classe sans pour autant détailler le code ou les instructions qui la constituent grâce à l'instruction `pass` de python. De même, préférez coder dans les classes vos fonctions plutôt

que dans le __main__ : le but de cette UEF est d'apprendre la POO non pas de continuer à coder en procédural !

- Maintenant que le squelette est ainsi réalisé, **répartissez-vous** ensuite la réalisation de **chaque méthode**. Définissez ensemble comment faire pour les tester de façon indépendante. N'hésitez pas à documenter votre code (non pas toutes les lignes, mais plutôt le principe employé, les contraintes et/ou spécificités). Pour faciliter le travail d'équipe, vous pouvez également utiliser des services de code collaboratif en ligne tels que : Visual Studio Live Share, Google collab, CodeSandbox, CodePen, Codeanywhere, Codeshare…
- Cherchez à faire de la **vérification croisée** : cela vous permettra de comprendre le travail de votre binôme, mais également de permettre de lever des erreurs qu'il ne voit plus ou qu'il n'aurait pas vues.
- Ne cherchez pas à réaliser toutes les fonctionnalités demandées d'un coup, travaillez par étapes (l'ordre proposé dans la section « Travail à réaliser » peut vous aider en cela), que vous 0validez au fur et à mesure avant de passer à la fonctionnalité suivante.