V2.2

PISCINE CODING FACTORY

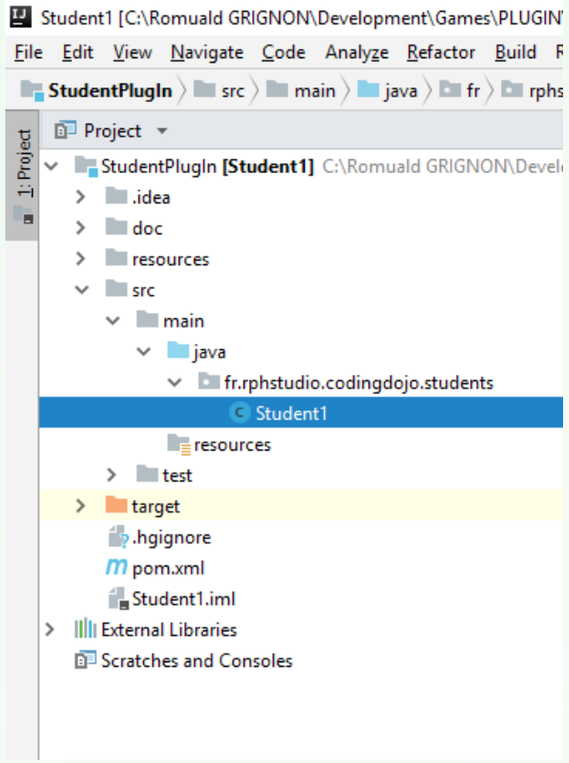
Pierre-Henri FRICOT
Romuald GRIGNON
Robin PENEA
Joachim THIBOUT

Coding Factory by ESIEE-IT
CCI Paris Ile-de-France
Cergy-Pontoise / Paris-Montparnasse

COURSE DE VAISSEAUX : préparation

Préparation de l'environnement

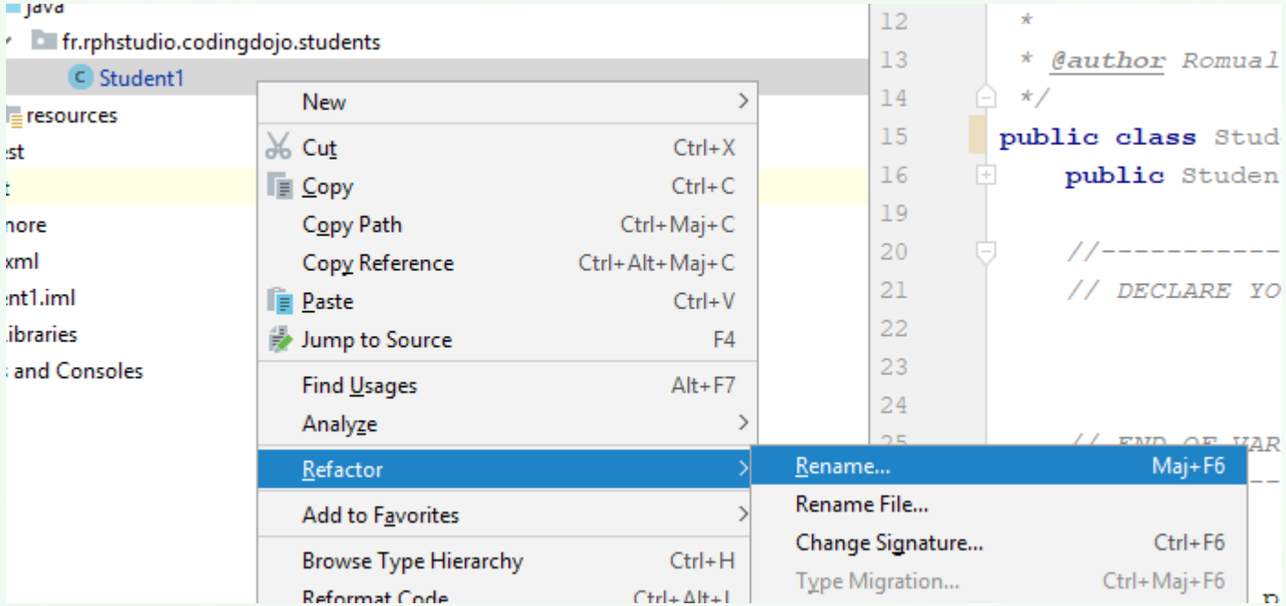
- Une fois le code du projet récupéré à :
  - https://github.com/Romuald78/codingdojo_studentplugin
- Ouvrez le projet sous IntelliJ. Pour cela, ouvrez le fichier pom.xml à la racine du dossier, et indiquez que vous voulez l'ouvrir en tant que « projet »
- Dépliez le répertoire des sources jusqu'à trouver le fichier '**Student1**'
- Double-cliquez dessus pour l'ouvrir
- Vous devriez voir apparaître le code d'origine qui est fourni

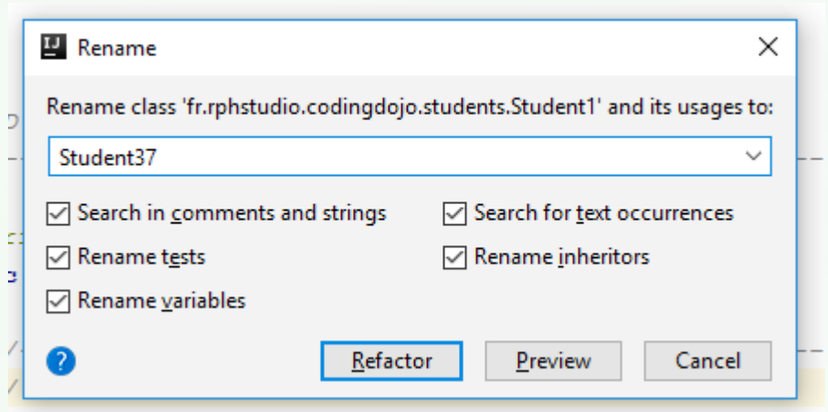



```
15  public class Student1 extends PodPlugIn {
16      public Student1(Pod p) { super(p); }
19
20      //-------------------------------------------------------
21      // DECLARE YOUR OWN VARIABLES AND FUNCTIONS HERE
22
23
24
25      // END OF VARIABLES/FUNCTIONS AREA
26      //-------------------------------------------------------
27
28      @Override
29      public void process(int delta)
30      {
31          //-------------------------------------------------------
32          // WRITE YOUR OWN CODE HERE
33
34          setPlayerName("Student 1");
35          selectShip( shapeNumber: 1);
36          setPlayerColor( r: 255, g: 255, b: 255, a: 255);
37
38          moveToNextCheckPoint( speed: 0.5f);
39
40          // END OF CODE AREA
41          //-------------------------------------------------------
42      }
```

Préparation de l'environnement

- Ici le projet est configuré pour le groupe '**1**'. Vous allez devoir modifier ce fichier afin de le renommer avec votre numéro de groupe. Par exemple, si vous portez le numéro '**37**', vous allez devoir renommer le fichier en '**Student37**'
- Pour effectuer cette opération, faîtes un clic-droit sur le fichier '**Student1**' et choisissez '**refactor**' → '**rename**'. Entrez le nom voulu et validez

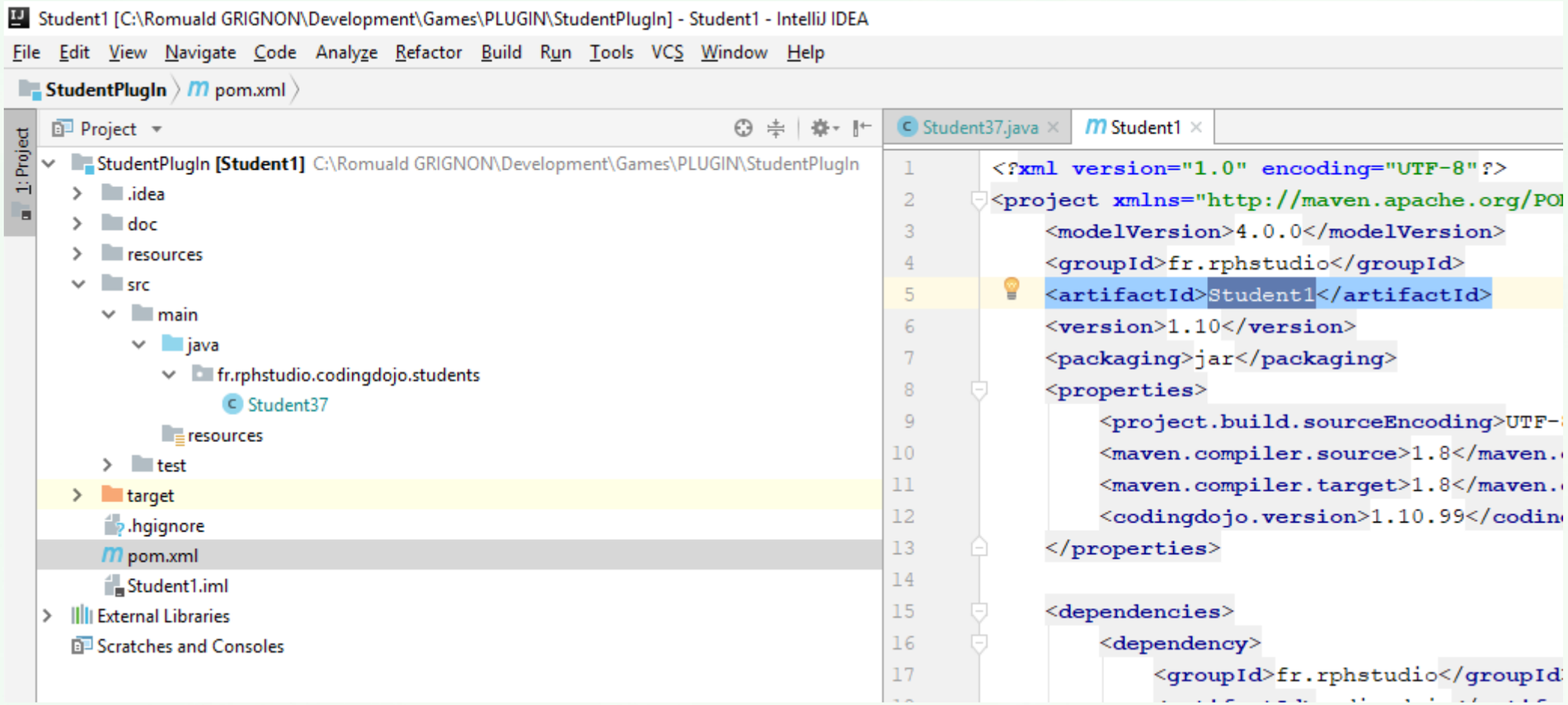




- Les numéros 78, 87 et 89 sont des numéros réservés. Vous pouvez choisir un numéro de groupe entre 1 et 100

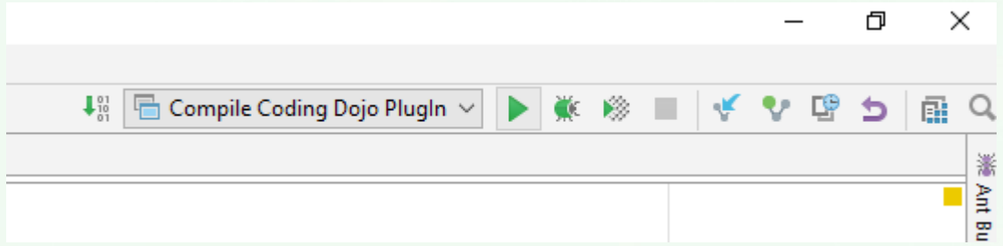
Préparation de l'environnement

- Visualisez le fichier '**pom.xml**', et modifiez la ligne qui contient '**Student1**' par le numéro de votre groupe (ici dans l'exemple remplacez par '**Student37**')



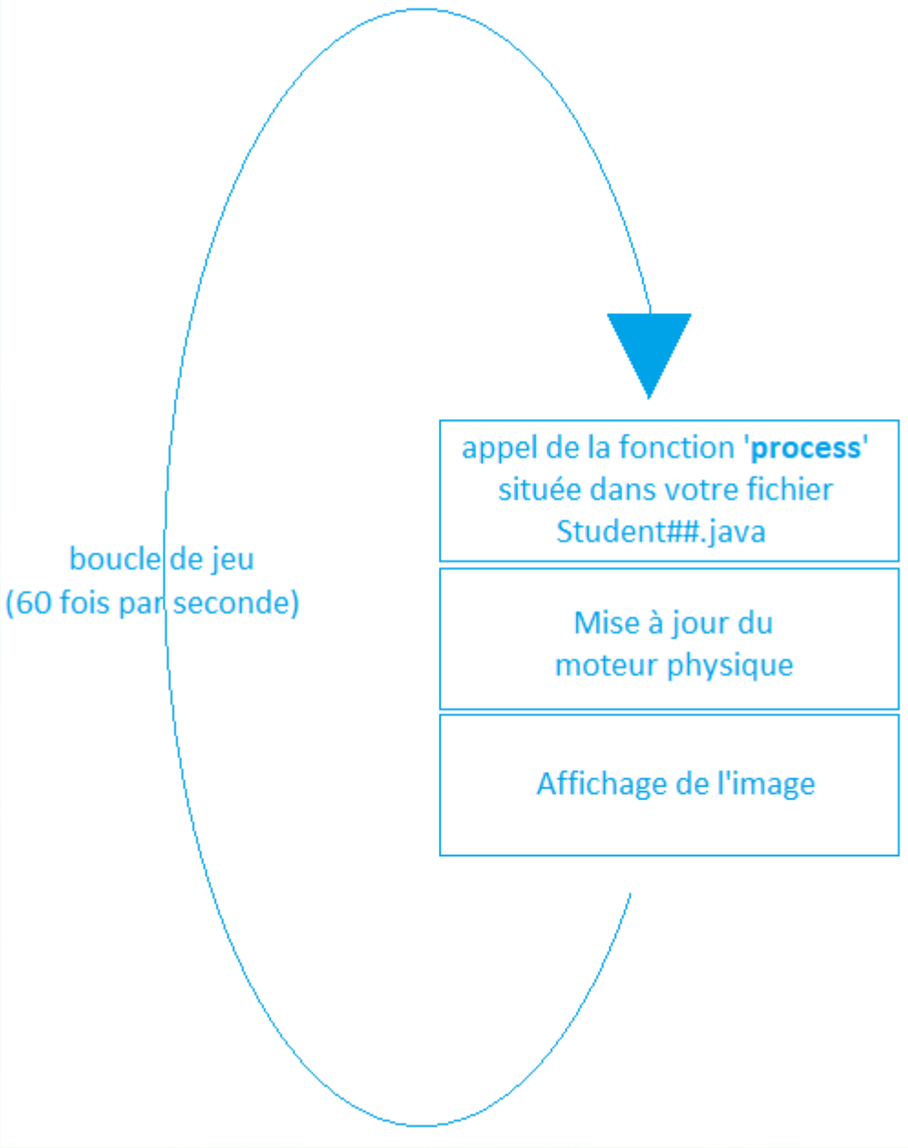
Préparation de l'environnement

- Maintenant nous allons tester l'exécution du projet en cliquant sur le bouton '**play**'



- Cette opération permet de compiler votre code et de créer un fichier nommé '**Student##.jar**' dans le sous-répertoire '**/resources/students**' de votre dossier de projet (avec ## le numéro de votre groupe tel que configuré précédemment)
- Normalement l'application devrait se lancer automatiquement après la compilation : si vous voyez les vaisseaux se déplacer c'est que vôtre environnement est prêt : il ne reste plus qu'à coder
- Ici, le mot-clé static qui était demandé durant les exercices n'est plus nécessaire. Pour vos fonctions ou variables globales, utilisez simplement la syntaxe du cours
- Vous verrez qu'il existe un autre fichier déjà généré, **Teachers-2.1.jar**, qui contient le code des vaisseaux des formateurs : un challenge de plus à relever ^^

Fonctionnement du code



- Votre code est appelé 1 fois par image (environ 60 fois par seconde)
- Votre code pourra récupérer l'état de votre vaisseau (position, vitesse, …) et décider d'effectuer des actions en retour (accélérerou freiner, tourner, …)
- Ces actions auront un impact sur la physique de votre vaisseau
- Une fois votre fonction terminée de s'exécuter, le moteur physique sera mis à jour, et l'image à l'écran sera renouvelée

- Entre 2 appels de votre fonction process, les variables disparaissent car elles sont locales à la fonction, comme vu précédemment
- Pour mémoriser des informations d'un appel à l'autre, il vous faudra utiliser des variables globale

- Vu que votre code est appelé séquentiellement dans la boucle de jeu, il est impératif d'éviter des boucles infinies qui auraient comme conséquence de geler l'application
- Rassurez vous nous avons mis en place un mécanisme permettant de détecter les traitements trop longs et de ne pas geler l'application en « tuant » le traitement incriminé

COURSE DE VAISSEAUX : description

Visuel de l'application

- L'application est découpée en 3 grandes parties :
  - Les commandes possibles en haut à gauche
  - Les statuts de chaque vaisseau et le classement sur la droite
  - La zone de course au centre

Commandes de l'application

- Changer de piste avec les flèches GAUCHE et DROITE
- Changer de niveau de code avec les flèches HAUT et BAS (voir règles spéciales)
- Changer le déroulement du temps avec les touches + et – (ou P et M)
  - Il est possible de modifier le temps avec les rapports de [1:8] à [x256] par puissance de 2
  - Si vous réduisez le temps en dessous de [1:8] le jeu passe en pause
- Activer les collisions entre vaisseaux avec la touche C (par défaut les collisions ne sont pas activées et les vaisseaux se chevauchent en mode 'ghost race')
- Activer le mode 'batterie' avec la touche B (voir règles spéciales)
- Activer le mode 'nombre de tours infini' avec la touche I
- Faire défiler le classement avec les touches PAGE UP et PAGE DOWN
- Basculer en mode fenêtré avec la touche F11
- Afficher le nombre d'images par seconde avec la touche F
- Quitter l'application avec la touche ESCAPE

Statut des vaisseaux

- Des indicateurs visuels existent pour préciser l'état de chaque vaisseau
- Les informations suivantes sont disponibles à tout instant :
  - Nom du vaisseau fourni par l'algorithme (également affiché au dessus du vaisseau lui-même)
  - Nombre de tours terminés
  - Barre de progression du tour en cours
  - Barre de niveau de batterie (reste à 100% quand le mode batterie est désactivé)
  - Barre de chargement du boost (voir règles spéciales)
  - Meilleur temps de tour
  - Temps total de course
  - Classement du vaisseau ou état spécifique (ex : cause de disqualification)

Statut des vaisseaux

- Un symbole décrit l'état du vaisseau pendant la course :

  [-] le vaisseau est en pleine course : état par défaut

  [#] un nombre indiquant le classement final du vaisseau (dans le cas où le vaisseau vient à manquer d'énergie, son classement peut déjà être fourni avant la fin de la course)

- Le symbole peut indiquer que le vaisseau a été disqualifié :

  [?] le vaisseau est sorti trop loin de l'écran et s'est perdu dans l'espace

  [!] le vaisseau a utilisé un appel de fonction non autorisé par le niveau de code actuel, ou bien le vaisseau a appelé plusieurs fois des fonctions pour accélérer ou plusieurs fois des fonctions pour tourner (dans la même frame) entraînant un problème au niveau système : le vaisseau ne fonctionne alors plus jusqu'à la fin de la course

  [T] le vaisseau a mis trop de temps pour rejoindre le prochain checkpoint

  [~] l'algorithme du vaisseau a généré une boucle infinie : ce vaisseau sera disqualifié pour cette course ainsi que toutes les suivantes !

COURSE DE VAISSEAUX : règles spéciales

Niveau de code

- Pour coder l'algorithme de déplacement de votre vaisseau, vous avez à votre disposition plusieurs fonctions qui vous permettront de réaliser diverses opérations
- Ces fonctions sont fournies pour que vous puissiez avoir un code fonctionnel dès le départ, et il vous faudra recoder ces fonctions vous même étape par étape
- Dans la documentation fournie, ces fonctions ont un niveau de code maximal
- Si le niveau de code de la course est supérieur strictement à celui de la fonction que vous utilisez, alors cette fonction ne s'exécutera pas et votre vaisseau sera disqualifié
- Pour pouvoir faire fonctionner votre vaisseau avec un niveau de code plus élevé, vous devrez coder vous-même la fonctionnalité fournie par la fonction que vous appelez
- Une description plus détaillée des étapes de niveaux de code est disponible dans les diapositives qui suivent

Mode Batterie

- Lorsque l'on active le mode 'batterie' à l'aide de la touche B, les vaisseaux démarrent une course pendant laquelle les accélérations leur font consommer de l'énergie.
- La barre d'énergie est visible sur l'interface graphique, et la valeur restante de batterie est accessible en appelant la fonction **getShipBatteryLevel** qui retourne un pourcentage d'énergie restante (valeur décimale entre 0.0 et 100.0)
- Lorsque l'énergie arrive à 0, le vaisseau ne peut plus bouger et la course s'arrête pour lui. Il conservera son classement si les autres vaisseaux ne le dépassent pas entre temps, sinon il perdra les places en conséquence
- Il est possible de récupérer de l'énergie de 2 manières :
  - En freinant (appel à **accelerateOrBrake** avec paramètre négatif) on récupère un peu d'énergie proportionnellement à la vitesse actuelle du vaisseau et la puissance de freinage. Dans tous les cas, cela ne compensera pas la perte d'énergie due aux accélérations
  - En passant au-dessus d'un checkpoint qui porte le symbole électrique, on recharge sa batterie : il convient donc de faire stationner ou ralentir suffisamment son vaisseau au dessus d'un checkpoint de recharge le temps que la batterie soit remplie, avant de reprendre le cours de la course
- Le nombre de tours par course est de 10 en mode normal, et de 50 en mode batterie

Boost

- Il est possible d'effectuer une action de boost qui va temporairement augmenter la vitesse de votre vaisseau
- Une jauge de boost est disponible sur l'interface graphique et son niveau est accessible via l'appel à la fonction **getShipBoostLevel** qui retourne un pourcentage de chargement du boost (valeur décimale entre 0.0 et 100.0)
- Pour activer ce boost il faut appeler la fonction **useBoost**. Cet appel va vider la jauge à 0%. Il faudra donc attendre la fin de son rechargement pour pouvoir le réutiliser
- Attention, si la jauge de chargement de boost n'est pas remplie totalement à 100%, l'utilisation du boost réduira la vitesse du vaisseau temporairement au lieu de l'augmenter : pensez-donc bien à effectuer un test de la jauge avant de l'utiliser
- L'utilisation du boost ne consomme pas d'énergie quand le mode batterie est activé
- L'utilisation du boost doit être synchronisée avec le placement dans le circuit : une activation juste avant un virage ne ferait qu'envoyer votre vaisseau loin du chemin optimal et vous ferait perdre un temps précieux

COURSE DE VAISSEAUX
Documentation du code

Documentation du code

- Lorsque vous avez récupéré le projet, un sous-répertoire appelé « doc » contient un fichier HTML (index.html) qui réunit l'ensemble des fonctions que vous pouvez utiliser pour votre algorithme de pilotage de vaisseau
- Cette documentation est divisée en plusieurs catégories, notamment :
  - [Display] : esthétique du vaisseau (couleur, forme, nom)
  - [Actions] : commandes principales du vaisseau (accélérer/ralentir, tourner, activer le boost)
  - [Status] : état physique de votre vaisseau (position, angle, vitesse, niveaux de batterie et boost)
  - [Global] : progression dans la course (nombre de tours, progression dans le tour courant, numéro du prochain checkpoint à valider)
  - [CheckPoints] : informations sur les checkpoints (positions, recharge électrique)
  - [Trigonometry] : toutes les fonctions nécessaires aux calculs d'angles et de distances
- N'hésitez pas à faire appel aux formateurs quant à l'utilisation de cette documentation

COURSE DE VAISSEAUX
Evaluation

Evaluation des étudiants

- Malgré l'aspect ludique de l'application, le travail effectué sur la programmation de l'algorithme de votre vaisseau sera évalué (mais non noté)
- Cette (auto-)évaluation se fait à la fois sur l'avancement en terme de niveau de code, mais aussi sur le résultat fonctionnel
- Au moins 1 fois par demi-journée, plusieurs courses avec des mesures de performances seront lancée par les formateurs. A l'issue de ces courses, seront récupérés les classements et niveaux de code de chaque vaisseau
- Le classement des résultats se fera d'abord par niveau de code décroissant, puis, pour les ex-æquo, par classement de vos vaisseaux dans la course
- L'évaluation se fait avec le mode batterie activé

COURSE DE VAISSEAUX
Niveaux de code

Niveau de code #01

- C'est le niveau de départ. Il ne contient qu'une seule fonction, qui est celle fournie dans votre code : **moveAndRecharge**
- Cette fonction va faire en sorte de tourner votre vaisseau vers le prochain checkpoint et le faire accélérer dans cette direction.
- Si le niveau de batterie est trop faible, il se dirigera vers le premier checkpoint avec la fonction de recharge, et ne reprendra le cours de la course qu'une fois sa batterie revenue à un niveau acceptable

Niveau de code #02

- Au niveau 2, la fonction **moveAndRecharge** ne fonctionne plus, il faut donc la remplacer
- Le but ici est d'utiliser la fonction **updateChargingMode** qui fournit en retour une valeur booléenne sur le besoin ou non de charger son vaisseau en énergie
- En fonction du besoin de recharge, il faudra effectuer un branchement conditionnel
  - Soit aller vers le prochain checkpoint de la course (**moveToNextCheckPoint**)
  - Soit aller vers le premier checkpoint de recharge (**moveToFirstChargingCheckPoint**)

Niveau de code #03

- Au niveau 3, les deux fonctions de déplacements précédentes ne sont plus disponibles
- Il faut donc maintenant séparer en 2 la tâche qu'elles remplissaient : c'est à dire se tourner vers la cible, et accélérer
- Pour cela vous pourrez utiliser les fonction **turnTowardNextCheckPoint** et **turnTowardFirstChargingCheckPoint** pour vous tourner vers la cible que vous désirez (respectivement le prochain checkpoint ou un checkpoint de recharge)
- Il vous restera à utiliser la fonction **accelerateOrBrake** pour accélérer en direction de votre cible
- Remarquez que maintenant vous pouvez accélerer à des niveaux différents en fonction de votre cible

Niveau de code #04

- Au niveau 4, les 2 fonctions de rotation précédentes ne sont plus disponibles
- En remplacement nous allons utiliser la fonction **turnTowardPosition** qui nécessite de connaître les coordonnées X et Y de votre cible
- Vous utiliserez donc les fonctions **getNextCheckPointX**, **getNextCheckPointY pour** récupérer les coordonnées du prochain checkpoint
- Vous utiliserez **getFirstChargingCheckPointX**, **getFirstChargingCheckPointY** pour récupérer les coordonnées du premier checkpoint de recharge
- Ces coordonnées seront passées en paramètres à la première fonction présentée ici

Niveau de code #05

- Les fonctions de récupération de coordonnées précédentes ne sont plus disponibles au niveau 5
- Nous allons utiliser pour cela la fonction générique **getNextCheckPointIndex** pur récupérer le numéro du prochain checkpoint à rallier
- De la même manière en utilisant **getFirstChargingCheckPointIndex** nous pouvons récupérer le numéro du premier checkpoint de recharge
- Ce numéro peut être passé en paramètre des fonctions **getCheckPointX** et **getCheckPointY** pour récupérer les coordonnées de n'importe quel checkpoint à paritr de son numéro
- Comme pour le niveau précédent, à partir des coordonnées, vous pouvez vous tourner vers votre cible

Niveau de code #06

- Ici c'est la fonction **updateChargingMode** qui n'est plus disponible au niveau 6
- Nous n'avons donc plus moyen automatique de savoir si nous devons recharger le vaisseau ou non : c'est donc cette tâche qu'il s'agit de résoudre ici
- Vous utiliserez le concept de variable globale pour mémoriser le fait que vous êtes en train de recharger ou non
- Vous mettrez à jour cette variable en comparant le niveau de batterie restante grâce à la fonction **getShipBatteryLevel**

Niveau de code #07

- Ici c'est la fonction qui recherche l'index du premier checkpoint de recharge qui n'est plus disponible
- Il va falloir trouver ce checkpoint par vos propres moyens en utilisant le concept de boucle
- En parcourant tous les numéros de checkpoint possibles (le nombre maximum de checkpoints est fourni par la fonction **getNbRaceCheckPoints**), il faudra tester si chaque checkpoint possède la possibilité de recharger grâce à la fonction **isCheckPointCharging**
- Si vous trouvez un checkpoint qui possède cette capacité, il faudra mémoriser son numéro
- Le reste est commun au code du niveau précédent, à partir du numéro de checkpoint, vous pouvez retrouver sa position, et orienter votre vaisseau dans cette direction
- A ce stade votre vaisseau recherche tout seul son point de recharge quand il en a besoin

Niveau de code #08

- Ici c'est la fonction **turnTowardPosition** qui n'est plus disponible
- Il va falloir utiliser la fonction **turnToAngle** qui permet de s'orienter vers un angle absolu donné
- Il existe une fonction **getAbsoluteAngleFromPositions** qui permet de calculer l'angle nécessaire pour orienter son vaisseau en fonction d'une position de destination
- Il faudra passer la position du vaisseau lui meme, récupérable grâce aux fonctions **getShipPositionX** et **getShipPositionY**
- Il faudra faire ce travail à la fois pour le prochain checkpoint quand le vaisseau a suffisamment d'énergie, et pour le premier checkpoint de recharge quand il a besoin de refaire le plein
- A ce stade, vous orientez votre vaisseau en indiquant une direction absolue (un angle)

Niveau de code #09

- A ce stade, c'est la fonction de rotation **turnToAngle** qui n'est plus disponible
- Il va falloir indiquer simplement à votre vaisseau de tourner de manière relative (vers la gauche ou vers la droite) et de combien de degrés
- Pour vous aider, il existe une fonction **getRelativeAngleDifference** qui calcule la différence entre 2 angles
- Grâce à l'orientation de votre vaisseau (angle de départ) et la direction vers votre cible (calculée dans votre code du précédent niveau) vous allez pouvoir calculer l'angle relatif à appliquer à votre vaisseau
- Il ne vous reste plus qu'à utiliser la fonction **turn** pour orienter votre vaisseau
- A ce stade, vous pilotez votre vaisseau en utilisant seulement 2 fonctions d'actions : **accelerateOrBrake** et **turn**.
- En d'autres termes, vous avez maintenant 2 commandes, l'une permet d’accroître ou de réduire la vitesse (accelerateOrBrake), et l'autre permet de diriger vers la gauche ou la droite votre vaisseau (turn)

Niveau de code #10

- A partir du niveau 10 et au delà, seuls des aspects trigonométriques sont impliqués dans la progression du niveau de code
- Si l'aspect mathématiques vous rebute, vous pouvez rester au niveau 9 et vous concentrer sur les « fonctionnalités » permettant à votre vaisseau d'être plus efficace dans ses courses
- A ce stade, c'est le calcul des distances entre 2 points qui n'est plus disponible
- La fonction **getDistanceBetweenPositions** doit être recodée
- C'est un simple calcul du théorème de Pythagore qui doit être fait ici en utilisant la fonction racine carrée **sqrt**
- A ce stade, toutes les distances de votre algorithme sont calculées à l'aide de votre propre code

Niveau de code #11

- A ce niveau, la fonction **getAbsoluteAngleFromPositions** n'est plus disponible
- Vous allez donc devoir calculer vous même les angles absolus nécessaires à votre algorithme
- Pour cela vous utiliserez la fonction tangente inverse (**atan2**) pour réaliser cette fonctionnalité
- A ce stade, tous les angles absolus de votre algorithme sont calculés par votre propre code

Niveau de code #12

- A ce niveau, la fonction **getRelativeAngleDifference** n'est plus disponible
- Vous allez donc devoir calculer vous même les différences d'angles présentes dans votre algorithme
- A ce stade, tous les angles relatifs de votre algorithme sont calculés par votre propre code
- Votre code n'utilise plus aucune fonction spécifiquement développée pour vous aider. Vous êtes autonome : il ne vous reste qu'à vous focaliser sur les « fonctionnalités » qui permettront de rendre votre algorithme plus performant et donc à votre vaisseau de gravir les places du classement

COURSE DE VAISSEAUX
Fonctionnalités avancées

Fonctionnalités

- Maintenant que vous avez atteint un niveau de code suffisant (niveau 9+), votre algorithme ne contient que des fonctions que vous avez codées. Il vous reste à améliorer cet algorithme afin que votre vaisseau puisse être plus performant
- Il y a plusieurs axes de travail qui ont chacun leur impact, et vous pouvez travailler sur ceux que vous voulez, dans l'ordre que vous voulez :

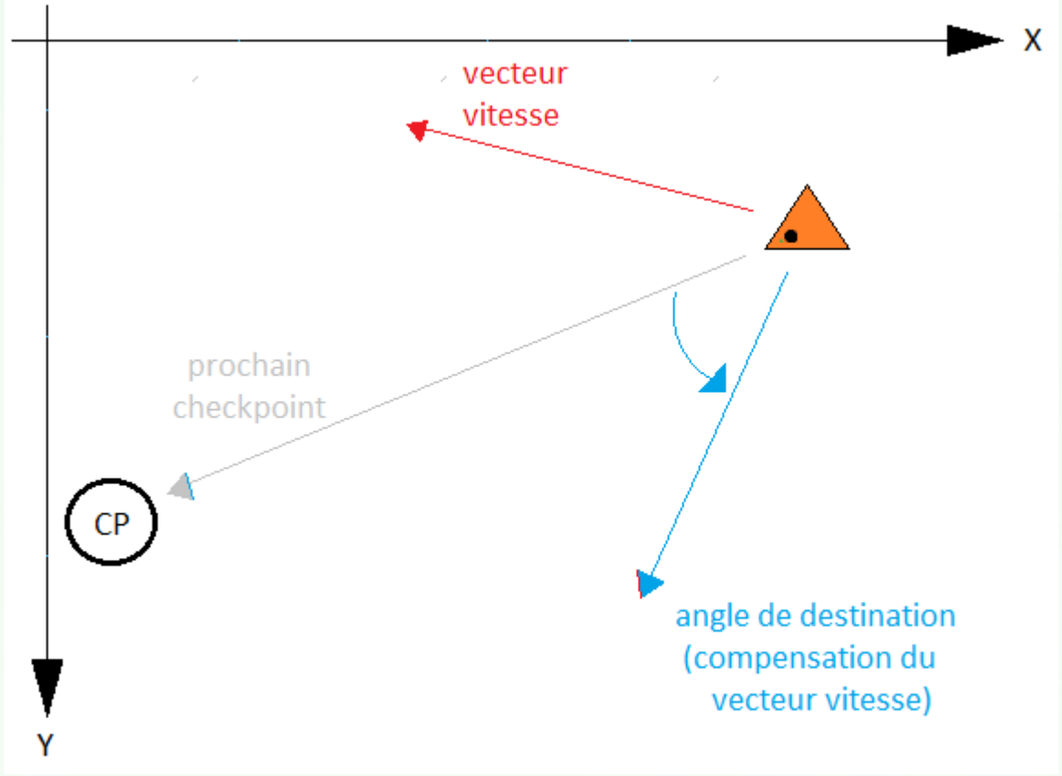
- #A – Optimisation de trajectoire
- #B – Economie d'énergie
- #C – Utilisation du 'Boost'

Optimisation de trajectoire

- Freinage en arrivant sur le checkpoint :
  - Il peut être intéressant de freiner en arrivant sur le prochain checkpoint afin de ne pas trop 'glisser' et ainsi être dans une position plus intéressante d'un point de vue de la trajectoire optimale
  - Pour cela vous pouvez vérifier la distance qui vous sépare du prochain checkpoint et freiner quand vous passez en dessous d'un certain seuil
  - Il sera peut être nécessaire de vérifier l'angle d'arrivée sur le checkpoint, la distance n'étant pas forcément le seul élément utile à gérer
  - Avec 3 checkpoints alignés, freiner en arrivant sur celui du milieu sera contre productif
- Drift :
  - En arrivant à proximité du prochain checkpoint, ET en étant bien aligné dans sa direction, vous pouvez commencer à orienter votre vaisseau vers le second checkpoint, a minima en arrêtant l'accélération, voire en freinant, et ainsi voir votre vaisseau partir en glissade sur le côté afin d'être bien aligné pour la suite de la course
  - Le fait d'être bien aligné vers le prochain checkpoint concerne votre vecteur vitesse et non pas l'angle vers lequel s'oriente votre vaisseau. Utilisez bien les composantes du vecteur vitesse **getShipSpeedX** et **getShipSpeedY** pour vous aider dans cette tâche
  - Si votre vitesse est trop réduite, partir en drift sera contre-productif

Optimisation de trajectoire

- Compensation de vecteur vitesse :
  - Avec l'inertie, votre vaisseau va très probablement dépasser le prochain checkpoint et va être poussé vers l'extérieur du virage que vous essayez d'effectuer
  - Il va donc falloir compenser ce vecteur vitesse en vous tournant un peu plus vers l'intérieur du virage
  - Attention à ne pas trop compenser sinon votre vaisseau va perdre la vitesse qu'il avait au début du virage et cela deviendra contre-productif
  - il s'agira donc de 'limiter' d'un angle maximum la compensation
  - Combiner cette fonctionnalité avec un freinage adéquat avant d'arriver sur le checkpoint pourrait s'avérer très performant



Economie d'énergie

- Ralentir suffisamment en arrivant sur des checkpoints de recharge vous permettra de faire remonter votre niveau de batterie et peut vous permettre de recharger moins de fois au cours de la course mais vous perdez du temps à chaque tour c'est donc un compromis à trouver
- Globalement, accélérer vous fait perdre de l'énergie mais vous permet de gagner du temps sur chaque tour donc le conseil (totalement subjectif) des formateurs c'est :

« Full Throttle » !!!

- Arrêt complet à la recharge :
  - Il faut tenter de se stabiliser le plus rapidement possible au dessus du point de recharge afin de diminuer au maximum le temps d'attente
  - Pour cela il faut garder le « pied sur le frein » quand on est au dessus
  - Si le vaisseau, avec l'inertie, s'éloigne du checkpoint il faut réaccélerer dans sa direction et forcer le freinage complet à nouveau une fois au dessus
  - Se stabiliser au dessus du point de recharge aura un effet très bénéfique sur le temps global

Economie d'énergie

- Point de recharge le plus proche
  - Votre boucle qui recherche le premier checkpoint de recharge peut être modifiée en vérifiant la distance entre chaque checkpoint et votre vaisseau, et ne retenir QUE le checkpoint le plus proche
  - Ainsi, dans les courses où il y a plusieurs points de recharge, vous vous assurerez de vous diriger vers le plus proche à tout moment de la course (vous ne savez pas quand votre vaisseau aura besoin de se recharger)
  - Vous atteindrez le point de recharge plus tôt → Vous consommerez moins d'énergie pour atteindre le point de recharge → Vous réduirez le temps de recharge (il y a moins à recharger)
- Recharge Now !
  - Globalement le principe que nous avons adopté c'est de passer en mode recharge quand la batterie passe sous un seuil que nous considérons comme critique
  - Mais il peut s'avérer intéressant de recharger son vaisseau avant d'atteindre ce seuil critique (un seuil légèrement plus haut) car le prochain checkpoint devant nous possède la capacité de recharge
  - Cette technique permet d'éviter de passer le checkpoint à fond et, une fois lancé dans la suite de la course, le niveau de batterie devient faible et le vaisseau stoppe ce qu'il fait pour faire demi tour vers le point de recharge : ici on anticipe de manière simple la perte d'énergie et on s'arrête au premier point de recharge rencontré

Economie d'énergie

- Recharge partielle
  - Vers la fin de la course (dans les derniers tours) il peut être intéressant de ne recharger la batterie que partiellement car il ne reste presque plus de distance à parcourir. Ceci afin de ne pas perdre trop de temps à attendre de l'énergie que l'on n'utilisera pas.
  - Cette fonctionnalité est en lien avec la suivante : car il faudra peut-être une mesure de la consommation moyenne par tour ;-)
- Historique de consommation
  - On peut améliorer le principe global de recharge en tenant à jour un historique de la consommation « entre checkpoints »
  - On mesure le niveau de batterie à chaque changement de checkpoint, et on stocke les différences afin d'avoir un histogramme des consommations par portions de course
  - Grâce à ces données, lorsque nous nous dirigerons vers un checkpoint avec recharge, il sera possible d'estimer la consommation restante jusqu'au prochain checkpoint, et donc savoir si il faut s'arrêter maintenant
  - Cette fonctionnalité permet d'éviter à votre vaisseau d'interrompre sa route lorsqu'il doit se recharger : il anticipe les rechargements et ne dévie jamais de la trajectoire normale de la course

Utilisation du Boost

- Le boost permet de voir sa vitesse augmenter très vite pendant un court instant
- Il ne consomme pas d'énérgie ; il est donc très utile pour repartir d'un point de recharge par exemple, où la vitesse est nulle

- Boost simple :
  - Il faut vérifier que le niveau de boost est de 100% avant de l'utiliser sinon le vaisseau se verra ralentir
  - Il faut vérifier que le vaisseau est bien orienté dans la direction que l'on souhaite avant de l'activer
  - Il faut vérifier que le prochain checkpoint n'est pas trop proche, sinon le boost vous fera partir trop loin derrière le checkpoint et cela sera contre-productif
  - Cette première étape d'utilisation du boost sera simplement un test de distance et d'alignement vers votre prochaine cible
  - Attention tout de même à l'inertie : il est sage de vérifier que le vaisseau est bien orienté mais aussi que le vecteur vitesse est bien orienté pour garantir une utilisation optimale du boost

Utilisation du Boost

- Boost avec 2 checkpoints alignés :
  - Dans le paragraphe précédent nous avons indiqué qu'il fallait que la distance du prochain checkpoint soit grande pour utiliser sans risque le boost
  - Il est possible de modifier légèrement cette condition si les 2 prochains checkpoints sont « alignés », alors cela veut dire que nous avons devant notre vaisseau une très grande ligne droite et c'est avec la distance du second checkpoint que nous devons faire la comparaison
  - D'un point de vue mathématiques, il sera peu probable que le vaisseau et les 2 checkpoints soient correctement alignés mais ici il faut tester que la variation de l'angle n'est pas trop importante afin de pouvoir activer le boost malgré tout
- Boost et économie d'énergie :
  - Il est possible de réduire sa consommation en arrêtant (ou diminuant) l'accélération, et ce, tant que la vitesse du vaisseau est sous l'influence du boost
  - Attention tout de même, le fait d'arrêter d'accélérer ne permettra plus de compenser la trajectoire il convient d'être prudent quant à la manière de le faire