IUT, Université Jean Monnet, Saint-Étienne

SAé 2 Info2 : Projet d'Algorithmique et Programmation informatique

Année 2021-2022

Fabien Momey, Bruno Bernard, Paul Grandamme, Franck Gérossier, Vincent Grosso, Cyril Mauclair, Thierry Lagniet

INTRODUCTION :

Cette **Situation d'Apprentissage Évaluée (SAé)** va vous permettre de mettre en œuvre vos **compétences en programmation informatique** au travers de l'élaboration d'un **projet complet de réalisation d'une application en C++**, en **autonomie** et par **groupes de 4 à 5 étudiants**.
Cette SAé sera **évaluée**. Un **suivi régulier** sera mis en place par l'équipe d'enseignants évaluateurs, avec des **jalons à respecter** pour le rendu de certains **livrables** (rapport, codes, etc.), puis un **rendu final**.

PRÉREQUIS & RESSOURCES À DISPOSITION :

Les connaissances nécessaires pour ce projet ont été acquises tout au long du semestre 1 et en début de semestre 2 notamment lors de vos apprentissages dans les ressources **INFO1/INFO2** mais aussi dans d'autres ressources telles que **AUTO1/AUTO2**, ainsi qu'en **SAé ER**.
Ce **PROJET** étant essentiellement basé sur les acquis du **module INFO1** du premier semestre, vous aurez à disposition toutes les **ressources** associées sur **IUTSEGEII_S1_M1103_INFO1**. Un onglet **SAé2 Info2 - Ressources pour le projet d'informatique** a également été créé sur **IUTSEGEII_S1_M1103_INFO1** dans lequel vous retrouverez le **présent document** ainsi que des **ressources dédiées** *(à consulter régulièrement)*.

Résumé des notions à maîtriser :

- **Connaissances techniques :**
  - **Bases d'algorithmique et programmation informatique.**
  - Variables, types, structures de contrôle (conditions, boucles).
  - Tableaux, **structures**, **pointeurs**.
  - **Fonctions** : appel de fonctions, définition de fonctions, notion de **passage par valeur (entrées en « lecture seule » E)** et **passage par adresse (entrées/sorties en « lecture/écriture » E/S)**.
  - ***Allocation dynamique de mémoire.***
- **Mise en œuvre d'une solution algorithmique complète exploitant ces connaissances.**
- **Implémentation d'une solution algorithmique complète dans un langage de programmation (C++).**
- Structure d'un programme informatique :
  - fichier source principal "`main.cpp`".
  - fichiers d'en-tête "`.h`" : prototypes des fonctions et déclaration des structures.
  - fichiers sources "`.cpp`" : définition des fonctions.
- Écriture du code et compilation dans un environnement de développement intégré (IDE) :
  - Code::Blocks.
- **Vérification du code** à l'aide de **procédures de test** et **débogage**.

COMPÉTENCES & APPRENTISSAGES CRITIQUES MIS EN JEU :

C1-N1-CONCEVOIR

C1-N1-AC1 - Produire une analyse fonctionnelle d'un système simple

➟ Pour toutes les étapes de ce projet, vous devrez proposer une solution algorithmique complète et détaillée (découpage fonctionnel, algorithme principal, procédures de test), indépendamment du langage de programmation choisi, qui respecte les contraintes imposées dans le cahier des charges.

C1-N1-AC2 - Réaliser un prototype pour des solutions techniques matériel ou logiciel

➟ Vous devrez ensuite implémenter dans une application console C++ chaque étape du projet, en traduisant fidèlement la solution algorithmique élaborée.

C1-N1-AC3 - Rédiger un dossier de fabrication à partir d'un dossier de conception

➟ Vous devrez rédiger une **documentation complète** de votre programme, sous forme algorithmique et illustrée par des exemples issues de votre prototypage en C++. **Ce rapport doit pouvoir servir de référence à un programmeur tierce qui souhaiterait implémenter votre solution dans un autre langage de programmation**.

➟ Concernant votre programme C++, ce dernier devra aussi être **rigoureusement commenté**, et notamment toutes les fonctions avec leurs **{ Rôle, Entrées, Entrées/Sorties, Sortie }** clairement définis.

C2-N1(N3)-VERIFIER

C2-N1-AC1 - Appliquer une procédure d'essai

➟ Pour chaque étape du projet, lors de la phase de conception algorithmique et d'implémentation, vous devez réfléchir à une batterie de tests unitaires à mettre en place - **sous forme de procédures à implémenter ou à réaliser avec l'outil débogueur par exemple** - pour valider chaque élément de votre solution (programme principal, fonctions, structures, etc.). **Toutes les procédures de test devront être rigoureusement décrites et documentées dans votre rapport. Si ce sont des procédures de test implémentées, elles devront être intégrées au code (dans une librairie dédiée par exemple) et rigoureusement commentées**.

C2-N1-AC2 - Identifier un dysfonctionnement

➟ L'application des procédures d'essai que vous aurez définies doivent vous permettre de vérifier le bon fonctionnement de votre programme et d'identifier des dysfonctionnements éventuels.

C2-N1-AC3 - Décrire les effets d'un dysfonctionnement

➟ En cas d'identification d'un dysfonctionnement, vous devrez garder trace de vos tests et analyses en rédigeant une petite description dans votre journal de bord (cf. partie PORTFOLIO ci-dessous).

C2-N3-AC2 - Proposer une solution corrective à un dysfonctionnement

➟ Bien entendu, tout dysfonctionnement de votre programme devra être corrigé et l'application de votre procédure d'essai devra permettre de valider la modification. Cette dernière sera également notifiée dans votre journal de bord.

SUIVI & GESTION DE VOTRE PROJET :

PORTFOLIO D'APPRENTISSAGE ET D'ÉVALUATION

- Parallèlement au **dossier de documentation**, vous devrez, en fin de projet, également rédiger un compte-rendu de projet personnel relatant votre gestion de ce dernier en équipe et analysant, avec un regard critique, votre expérience afin d'auto-évaluer vos acquis dans chaque compétence/apprentissage critique qui auront été travaillés. Ce compte-rendu permettra d'alimenter votre Portfolio d'apprentissage et d'évaluation qui rentrera en ligne de compte dans l'évaluation de cette **SAé**.
- Pour faciliter la rédaction de ce compte-rendu, nous vous demandons de tenir, tout au long de votre travail sur le projet, un journal de bord personnel dans lequel vous pourrez consigner toutes vos actions, résultats et remarques « au jour le jour ». **Ce journal de bord servira de document de suivi pour les enseignants évaluateurs et pourra vous être demandé aux différents jalons.**

JALONS & LIVRABLES :

Tout au long de l'élaboration des phases du projet, un certain nombre de jalons seront à respecter, avec des dates fixées et des livrables à fournir (codes, journaux de bord, état de la documentation). **Ces livrables seront à déposer sur IUTSEGEII_S1_M1103_INFO1 dans l'onglet dédié SAé2 Info2 - Livrables pour le projet d'informatique**.

Les jalons sont indiqués par le cartouche suivant à la fin des phases concernées :

Jalon & Livrable n°*i*

Guettez ces indicateurs dans le sujet du projet, tous les détails y sont donnés.

Chacun de ces jalons sera évalué par les enseignants-évaluateurs, et des entrevues pourront être prévues. **Voici les dates butoires à respecter (les espace de dépôt sur Claroline Connect seront ouverts ces jours) :**

Jalon & Livrable n°1 ⇒ 01/04/2022

Jalon & Livrable n°2 ⇒ 15/04/2022

Jalon & Livrable n°3 ⇒ 06/05/2022

Jalon & Livrable n°4 ⇒ 24/05/2022

Jalon & Livrable n°5 ⇒ 10/06/2022

Sujet du projet - le cahier des charges

Dans ce projet, vous allez créer un **application informatique en C++** qui va vous permettre de réaliser un "jeu" simulant une bataille de vaisseaux spatiaux. À terme, l'objectif est de créer un système de combat au tour par tour entre 2 flottes (2 joueurs) composées de plusieurs vaisseaux, pouvant être de types différents. Le combat prendra fin lorsque l'une des 2 flottes est totalement détruite.

Pour mener à bien ce projet, nous avons décomposé sa réalisation **en différentes phases**, à valider au fur et à mesure par des procédures d'essai, pour élaborer petit à petit les éléments du jeu et la mécanique des combats de vaisseaux.

Le moteur de l'application à réaliser sera essentiellement basé sur l'utilisation de structures pour « modéliser » le type `vaisseau` comme un conteneur englobant un certain nombre de caractéristiques, ainsi que la définition de librairies de fonctions dédiées à manipuler des instances de ces vaisseaux.

*Nota bene : le "jeu" que vous allez coder est rudimentaire puisqu'il s'agira d'une simple **application console** (comme tous nos programmes habituels). L'objectif n'est pas de coder **STARWARS™ : Squadrons (Electronic Arts Inc.)**. Néanmoins, ce contexte vous offre la possibilité de mettre en application vos apprentissages en programmation informatique au travers d'un thème que l'on espère assez ludique. Profitez-en pour laisser parler votre imagination sans négliger la rigueur imposée par la finalité de ce travail.*

Avant-propos : organisation du travail de groupe

Au début de chaque nouvelle phase, nous vous suggérons de réfléchir en commun à la solution que vous élaborerez, puis de vous répartir les différentes tâches à réaliser :

- C1-N1-AC1 Élaboration des algorithmes.
- C1-N1-AC2 Implémentation de la solution.
- C1-N1-AC3 Rédaction de la documentation.
- C1-N1-AC1 Élaboration et réalisation des procédures d'essai.
- C1-N1-AC2&AC3 — C1-N3-AC2 Identification et description des dysfonctionnements, puis et débogage et correction.

Bien entendu, certaines tâches se recouvrent et peuvent être traitées par les mêmes membres du groupe. Attention également, d'autres tâches sont « asynchrones » et vont requérir que d'autres aient d'abord achevé leur propre tâche. **Pensez à organiser un système efficace de partage de codes et de documents afin que chaque membre du groupe puisse avoir accès aux développements du projet « à jour ».**
Vous êtes libres de vous organiser comme bon vous semble, mais nous vous suggérons d'alterner, d'une phase à l'autre ou même dans une même phase, les types de tâches afin que chacun puisse tester ses multiples compétences. Votre organisation (la gestion de votre projet) devra être clairement définie dans vos rapports respectifs et vos journaux de bord. **Vous devez également tenir un journal de bord de groupe.**

CONVENTIONS DE CODAGE À RESPECTER :

- Uniquement des caractères alphanumériques pour le nommage des éléments de code (variables, fonctions, etc.). Pas de caractère accentué, pas d'espace, pas de point, seul le caractère « underscore (tiret du 8) » est toléré.
- les **variables** doivent être nommées en minuscule.
  Exemple :
  ```
  int mavariable ;
  ```
- les **constantes** doivent être nommées en majuscule.
  Exemple :
  ```
  const int MACONSTANTE ;
  ```
- les **structures** doivent être nommées par un mot commençant par une majuscule.
  Exemple :
  ```
  struct Mastruct { } ;
  ```
- les **champs** de structures doivent être nommées en minuscules et commencer par le préfixe `m_`.
  Exemple :
  ```
  int m_champ1 ;
  ```
- les **fonctions** doivent nommées par un ou plusieurs mots attachés chacun commençant par une majuscule.
  Exemple :
  ```
  void MaFonctionSuperGeniale() ;
  ```
- les **entrées** et **entrées/sorties** des fonctions doivent être nommées en minuscule et commencer par le préfix `input_` pour les entrées et `output_` pour les entrées/sorties.
  Exemple :
  ```
  void MaFonctionSuperGeniale(const int* input_var1, float* output_var2) ;
  ```
  Note : les entrées (lecture seule) doivent être précédées du mot-clé `const` *pour bien les identifier comme étant en lecture seule ...*

Phase 1 : création du conteneur `Vaisseau` générique

- Dans cette première phase, il faut implémenter le « conteneur » `Vaisseau` - *autrement dit un* ***type*** `Vaisseau` - qui permettra d'encapsuler un certain nombre de caractéristiques :
  - le type du `Vaisseau` - une information textuelle qui doit servir d'identifiant sur la classe du vaisseau ;
  - le nom du `Vaisseau` - une information textuelle pour nommer le vaisseau (`"Faucon Millenium"` par exemple) ;
  - la résistance de coque - les « points (entiers) de vie » - d'un vaisseau ;
  - la puissance de feu - les « points (entiers) d'attaque » - d'un vaisseau.

  Il faudra alors déterminer un **type approprié** par chacune de ces caractéristiques. *Concernant les chaînes de caractères, ne pas oublier qu'il s'agit d'un tableau de caractères dont la taille doit être suffisante pour permettre de « stocker » des mots de longueur adéquate.*

↬ Associé à la déclaration du type `Vaisseau`, on demande l'implémentation des premières fonctionnalités :

- ➟ 1 fonctionnalité qui permet d'**afficher** les informations d'un vaisseau ;
- ➟ 1 fonctionnalité permettant d'**affecter** globalement des valeurs aux caractéristiques du vaisseau ;
- ➟ des fonctionnalités permettant d'**affecter** indépendamment une valeur à chaque caractéristique du vaisseau ;
- ➟ des fonctionnalités permettant de **renvoyer** respectivement les valeurs de chaque caractéristique.

*Note importante : chacune de ces fonctionnalités est donc dédiée à un vaisseau « appelant » qui doit donc pouvoir être modifié par la fonctionnalité. Pensez « **passage par adresse** » ...*

Phase 2 : « duel de vaisseaux »

↬ On demande à présent de définir une fonctionnalité permettant d'**ôter un nombre entier de points de vie** à un vaisseau cible. En d'autres termes, un vaisseau doit pouvoir voir sa résistance de coque diminuée d'un nombre entier quelconque qui serait donc un paramètre de la fonctionnalité. *Il faut prévoir que le nombre de points de vie ne puisse aller en-dessous de zéro.*
Cette fonctionnalité met en lumière le fait qu'un vaisseau peut être détruit si sa résistance de coque tombe à zéro. Il faut alors mettre à jour le code existant pour permettre d'afficher cette information.

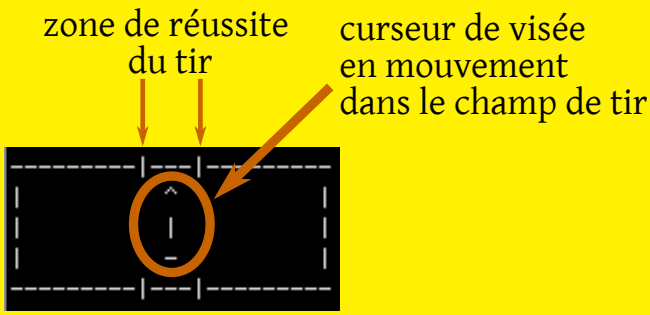
↬ **On propose d'implémenter une première application de test permettant de simuler un « duel de vaisseaux ».** Ce dernier fonctionnera selon un système de jeu « tour par tour ». On itérera ce processus jusqu'à ce qu'un des deux vaisseaux soit détruit (résistance de coque tombée à 0).
Voici un pseudo-algorithme de ce à quoi peut ressembler le duel à implémenter :

Répéter

> Décision du tour de vaisseau assaillant{« *Quel vaisseau doit lancer l'assaut ?* »}
> Appui sur la touche [Echap] pour démarrer un assaut.
> **Lancement du « moteur de jeu » simulant la visée du vaisseau et le tir**
> **⇒ renvoie VRAI en cas de réussite et FAUX sinon.**
> *{ Le tir sera déclenché par un appui sur la touche [ESPACE].*
> *Le moment où l'appui a été déclenché déterminera si le tir a réussi ou non. }*
> Si (resultat_tir = VRAI)
> le vaisseau assaillant inflige des dégâts au vaisseau cible.
> Sinon
> le vaisseau assaillant manque sa cible (aucun dégât).
> FinSi
> **Jusqu'à (*destruction d'un des 2 vaisseaux*)**

FinRépéter
Afficher le vainqueur.

Le « moteur de jeu » : on propose un système de visée basé sur un affichage dynamique d'un curseur de visée se déplaçant dans un champ de tir (cf. figure ci-dessous).



- L'affichage de ce système de visée se fera sur le terminal de l'exécutable. Le curseur de visée se déplacera rapidement de gauche à droite dans le champ de tir (de manière cyclique). L'utilisation de l'instruction système `system("cls")` permet de « »raffraîchir le terminal à chaque nouvel affichage du déplacement du curseur.
- Le déclenchement du tir se fera via un appui sur la touche [ESPACE]. La détection de l'appui sera géré grâce à la fonction `GetAsyncKeyState` de la librairie `windows.h` qui permet de « scruter » de manière **asynchrone (gestion d'évènements)** l'appui sur une touche spécifique du clavier ([ESPACE], [Echap], etc.).
- La réussite du tir sera conditionnée par cet appui au moment opportun où le curseur de visée entrera dans la **zone de réussite du tir**.
- Tout ce système sera confié à une fonction `Assaut` qui renverra un booléen `true` ou `false` suivant la réussite où l'échec du tir.
- Les itérations des assauts successifs de chaque vaisseau sera gérée dans le `main` avec appel de cette fonction `Assaut` dans la boucle de duel.

Ce moteur de jeu a été implémenté et vous est fourni. Il est disponible dans l'archive `moteur_de_jeu.zip` récupérable dans l'onglet **SAé2 Info2 - Ressources pour le projet d'informatique** sur IUTSE-GEII_S1_M1103_INFO1. Il contient :

- La librairie des fonctions (nommée `moteur_de_jeu` ⇒ fichier d'en-tête `.h` + fichier source `.cpp`) implémentant le moteur et notamment la fonction `Assaut`.
- Un petit programme `main` d'exemple manipulant le moteur. *Il faudra bien sûr tester ce code puis adapter le programme au cahier des charges du projet.*

Jalon & Livrable n°1 ⇒ 01/04/2022

Pour ce jalon, vous devrez fournir dans une archive `ZIP` intitulée `Jalon_1_n°binome.zip` :

- Votre code fonctionnel et correctement documenté.
- Vos journaux de bord, personnel et de groupe, au format numérique PDF.

Si vous n'êtes pas parvenus jusqu'à ce point, vous devez rendre **votre projet en l'état** à la date prévue.

Phase 3 : fonctionnalité `Attaquer`

- On souhaite à présent définir une fonctionnalité `Attaquer` « appelée par un vaisseau attaquant (l'appelant) » qui va infliger des dégâts (la cible va donc **subir des dégâts**) du nombre de sa **puissance de feu**.
  Note : cette fonctionnalité doit pouvoir utiliser des fonctionnalités déjà implémentées afin de favoriser l'intégrité de votre code.

Phase 4 : modification du conteneur `Vaisseau` et adaptation du duel

- On doit modifier le nom de la fonctionnalité `Attaquer` en `AttaquerBasic` qui constituera la **fonctionnalité d'attaque standard** du `Vaisseau`.
- En prévision des phases suivantes, on crée à présent une nouvelle fonctionnalité `AttaquerSpecial` qui constituera une **méthode d'attaque spécialisée** du `Vaisseau`. Pour le moment, **conteneur** `Vaisseau`

que vous avez implémenté constitue une classe générique de vaisseau, qui ne possède en soi aucune spécialisation. La fonctionnalité `AttaquerSpecial` **ne doit donc rien faire hormis afficher un message spécifique**.
C'est lors des phases suivantes du projet que seront définies des ***classes spécialisées de vaisseaux****, pour lesquelles la fonctionnalité* `AttaquerSpecial` *devra être redéfinie pour permettre de réaliser des actions d'attaque spécifiques.*

↬ On doit à présent **adapter le programme de duel de vaisseaux** précédemment programmé dans la **phase 2** afin qu'il utilise les nouvelles fonctionnalités `AttaquerBasic` et `AttaquerSpecial`.
Il faut alors intégrer un système (de votre choix, aléatoire par exemple) de sélection de l'attaque basique ou de l'attaque spéciale lors d'une itération du duel.

Note : cette adaptation doit être implémentée dans un nouveau programme principal, afin de garder trace de l'ancien programme. Vous pouvez créer un nouveau projet Code::Blocks et y importer vos librairies de fonctions pour y implémenter un nouveau `main`.

Jalon & Livrable n°2 ⇒ 15/04/2022

Pour ce jalon, vous devrez fournir dans une archive `ZIP` intitulée `Jalon_2_n°binome.zip` :

↬ Votre code fonctionnel et correctement documenté.

↬ Une première version de la documentation technique du code, au format numérique PDF, contenant a minima les algorithmes des fonctions et programmes développées jusqu'à maintenant.

↬ Vos journaux de bord, personnel et de groupe, au format numérique PDF.

Si vous n'êtes pas parvenus jusqu'à ce point, vous devez rendre **votre projet en l'état** à la date prévue.

Phase 5 : définition de 2 conteneurs « spécialisés » du `Vaisseau` : `Croiseur` et `Chasseur`

On propose de créer des **vaisseaux de classes spécialisées**, c'est-à-dire des vaisseaux qui vont intégrer, en plus des caractéristiques génériques à tout vaisseau, des caractéristiques et fonctionnalités particulières afin d'introduire un peu de variabilité et de personnalisation dans la future constitution de **flottes de vaisseaux**.
Comment faire ? Il s'agit de créer des nouveaux conteneurs « spécialisés » qui doivent « hériter » des caractéristiques d'un conteneur `Vaisseau`. Pour éviter de « recopier » les caractéristiques génériques dans ces nouveaux conteneurs, il faut globalement inclure un conteneur de type `Vaisseau` comme une caractéristique des conteneurs spécialisés ⇒ on aura ainsi « encapsuler » toutes les caractéristiques génériques et on pourra avoir accès aux fonctionnalités implémentées pour gérer cette caractéristique `Vaisseau` intégrée au conteneur spécialisé.
L'objectif de cette phase est donc de :

↬ créer 2 conteneurs - *2 nouveaux* ***types*** *de vaisseaux* - `Croiseur` et `Chasseur` qui doivent « spécialiser » le conteneur `Vaisseau`. Pour le moment ces conteneurs doivent être **vides** hormis la caractéristique `m_vaisseau` définissant ce lien « d'héritage » d'un type `Vaisseau` dans ces conteneurs.
D'un point de vue logique, un `Chasseur` *ou un* `Croiseau` ***est*** *un* `Vaisseau`*. D'un point de vue implémentationnel, on le modélise par le fait qu'un* `Chasseur` *ou un* `Croiseau` ***contient*** *un* `Vaisseau`*.*

↬ adapter le programme de duel de vaisseaux précédemment programmé dans la **phase 4** afin de tester un duel entre 1 `Chasseur` et 1 `Croiseur`.
L'interaction entre 2 vaisseaux spécialisés doit utiliser les fonctionnalités du type générique `Vaisseau` *-* `AttaquerBasic` *et* `AttaquerSpecial`*. Il faut donc trouver un moyen pour que des vaisseaux spécialisés puissent « appeler » ces fonctionnalités.*

Note : une fois de plus, cette adaptation doit être implémentée dans un nouveau programme principal, afin de garder trace des anciens programmes.

Phase 6 : spécialisation du `Croiseur`

La spécificité du `Croiseur` est de disposer d'une **arme spécifique**, le **canon**, matérialisée par 1 nouvelle caractéristique :

➟ la puissance de feu du canon.

La spécialisation du `Croiseur` consiste en :

↬ l'ajout des nouvelles caractéristiques au conteneur `Croiseur`.

↬ la définition de nouvelles fonctionnalités permettant d'**affecter** (globalement et individuellement) des valeurs à aux caractéristiques spécifiques d'un croiseur.

↬ la re-définition de la fonctionnalité permettant d'**afficher** les caractéristiques d'un croiseur.
On devra donc afficher aussi bien les nouvelles caractéristiques que les caractéristiques génériques (pour ces derniers on devra ré-utiliser la fonctionnalité `Afficher` *du conteneur* `Vaisseau`*).*
Note : d'un point de vie implémentationnel, la re-définition de fonctionnalité va faire appel à la technique de ***surcharge de fonction****. Cette dernière consiste à redéfinir une fonction avec le même nom qu'une autre mais des paramètres d'entrée différents. Ainsi dans notre cas, il s'agit d'adapter la fonctionnalité* `Afficher` *du conteneur* `Vaisseau` *pour que l'≪ appelant ≫ soit un* `Croiseur`*.*

Il ne faut pas oublier de réaliser des procédures de tests unitaires pour valider chaque nouvel élément ajouté au projet.

↬ la re-définition (sur le même principe que la fonctionnalité `Afficher`) de la fonctionnalité `AttaquerSpecial` qui va infliger les dégâts d'un **tir de canon**.
Cette attaque spéciale ne doit avoir qu'1 chance sur 2 de réussir.

Note : Si tout est implémenté correctement, le programme de duel entre un croiseur et un chasseur de la phase 5 doit fonctionner correctement sans apporter de modification.

Phase 7 : spécialisation du `Chasseur`

La spécificité du `Chasseur` est de disposer d'une **arme spécifique**, les **torpilles**, matérialisée par 2 nouvelles caractéristiques :

➟ la puissance de feu des torpilles ;

➟ le ≪ stock ≫ disponible de torpilles (qui sera donc décrémenté à chaque utilisation et finira par tomber à zéro).

La spécialisation du `Chasseur` suit les mêmes étapes que la spécialisation du `Croiseur`. La seule différence sera dans la re-définition de la fonctionnalité `AttaquerSpecial` **qui aura 100% de chances de réussite mais sera limitée en nombre d'utilisation, géré par l'une des caractéristiques spéciales du** `Chasseur`**.**

Jalon & Livrable n°3 ⇒ 06/05/2022

Pour ce jalon, vous devrez fournir dans une archive `ZIP` intitulée `Jalon_3_`*`n°binome`*`.zip` :

↬ Votre code fonctionnel et correctement documenté.

↬ Une nouvelle version de la documentation technique du code, au format numérique PDF.

↬ Vos journaux de bord, personnel et de groupe, au format numérique PDF.

Si vous n'êtes pas parvenus jusqu'à ce point, vous devez rendre **votre projet en l'état** à la date prévue.

Phase 8 : consolidation du moteur du programme

Pour le moment, les programmes de duel entre 2 vaisseaux sont conçus spécifiquement pour gérer un combat entre 2 vaisseaux dont on connait la classe :

➟ Un `Vaisseau` contre un `Vaisseau` ;

➟ Un `Croiseur` contre un `Chasseur` ;

On souhaite pouvoir généraliser le programme de duel pour que son **algorithme principal** soit le même quels que soient les 2 vaisseaux ≪ duellistes ≫, autrement dit quels que soient les 2 vaisseaux déclarés dans le **lexique principal**.

↬ Pour ce faire, il faut rendre **génériques** les appels aux fonctionnalités `AttaquerBasic` et `AttaquerSpecial` pour que la compilation du programme ≪ détecte automatiquement ≫ les classes de vaisseaux antagonistes :

➟ Un `Vaisseau` attaque un `Croiseur` ;

➟ Un `Croiseur` attaque un `Vaisseau` ;

➟ Un `Chasseur` attaque un `Chasseur` ;

➟ Un `Croiseur` attaque un `Croiseur` ;

➟ etc.

On va procéder en **surchargeant** ces fonctionnalités en autant de versions que de cas possibles.

↬ On peut alors générer autant d'exécutables du programme de duel en ne changeant avant compilation que la déclaration des 2 vaisseaux antagonistes.

Jalon & Livrable n°4 ⇒ 24/05/2022

OBJECTIF MINIMAL DU PROJET

Il faut arriver à rendre ce code fonctionnel, et tout doit être finalisé pour le rendu final du projet.
Pour ce jalon, vous devrez fournir dans une archive `ZIP` intitulée
`Jalon_4_n°binome.zip` :

↬ Votre code fonctionnel et correctement documenté.

↬ La documentation technique du code, au format numérique PDF.

↬ Vos journaux de bord, personnel et de groupe, au format numérique PDF.

Si vous n'êtes pas parvenus jusqu'à ce point, vous devez rendre **votre projet en l'état** à la date prévue.

Phase 9 : création d'un conteneur `Flotte`

↬ L'objectif final est de pouvoir gérer une **bataille globale entre 2 flottes de vaisseaux**. Pour ce faire, on propose de créer un conteneur `Flotte` qui contiendra plusieurs vaisseaux :

➟ 1 `Croiseur` ;

➟ 2 `Chasseur` ;

➟ 3 `Vaisseau`. *Note : ces vaisseaux doivent être regroupés par type, autrement dit une caractéristique du conteneur `Flotte` correspond à un groupement de vaisseaux d'un certain type ⇒* ***le type approprié est donc à déterminer.***

↬ Une instance de `Flotte` doit disposer des fonctionnalités suivantes :

➟ 1 fonctionnalité permettant d'**afficher** l'état de la `Flotte` : informations sur chaque `Vaisseau`. Cette fonctionnalité **doit entre autres appeler les fonctionnalités `Afficher` de chaque `Vaisseau`, `Croiseur` et `Chasseur` de la flotte**.

➟ plusieurs fonctionnalités (1 par type de vaisseau) permettant de **modifier les caractéristiques d'un `Vaisseau`, `Croiseur` ou `Chasseur` de la flotte**.
Note : la désignation d'un `Vaisseau`, `Croiseur` ou `Chasseur` au sein de la `Flotte` doit être géré par un ***numéro d'ordre*** *:*

- $\{1, 2, 3\}$ *pour l'un des 3* `Vaisseau`.
- $\{4, 5\}$ *pour l'un des 2* `Chasseur`.
- $\{6\}$ *pour le* `Croiseur`.

Ce ***numéro d'ordre*** *(type `int`) sera un des paramètres d'entrée des fonctionnalités* ***d'affectation****, afin de désigner le bon vaisseau.*

➟ 1 fonctionnalité qui renvoie `VRAI` si l'ensemble des vaisseaux de la flotte sont détruits, et `FAUX` sinon.

↬ Ce nouveau conteneur étant assez conséquent, un soin particulier doit être porté aux **procédures de tests unitaires** afin de valider rigoureusement le bon fonctionnement de ce conteneur.

Phase 10 : l'ultime bataille

↬ Nous voici enfin à l'objectif final de ce projet : **la mise en place d'un « jeu de combat » entre 2 `Flotte` au tour par tour**. Voici l'algorithme simplifié à coder dans votre programme principal :

Création et initialisation des flottes.
Répéter

> Définir le tour du joueur courant.
> Afficher l'état des flottes.
> Choisir le vaisseau attaquant puis le vaisseau ciblé.
> Choisir le type d'attaque.
> Résoudre l'attaque.
> **Jusqu'à (*destruction d'une des 2 flottes*)**

FinRépéter
Afficher le vainqueur.

- ↬ La mise en œuvre de requiert d'ajouter au conteneur `Flotte` une fonctionnalité `ChoixVaisseau` qui doit récupérer (en entrée/sortie) un pointeur vers le `Vaisseau`, `Croiseur` ou `Chasseur` en fonction du **numéro d'ordre** du vaisseau dans la flotte donné en entrée.
  Il faudra créer 3 surcharges de cette fonctionnalité pour qu'elle puisse récupérer, suivant le cas, un pointeur vers l'un des 3 types de `Vaisseau`.
- ↬ Dans un premier temps, on devra **tester le système de choix de vaisseau** en proposant une version du programme de duel permettant définir 2 flottes et d'organiser un duel entre 2 vaisseaux de ces flottes.
- ↬ Enfin, on mettra en place tout **le programme du jeu**.

Jalon & Livrable n°5 ⇒ 10/06/2022

OBJECTIF FINAL DU PROJET

Si vous arrivez jusqu'à ce jalon et que votre programme est totalement opérationnel (donc « jouable »), **vous êtes les rois du monde** !
Pour ce jalon, vous devrez fournir dans une archive `ZIP` intitulée `Jalon_5_n°binome.zip` :

- ↬ Votre code fonctionnel et correctement documenté.
- ↬ La documentation technique du code, au format numérique PDF.
- ↬ Vos journaux de bord, personnel et de groupe, au format numérique PDF.
- ↬ **Une soutenance de votre projet sous forme d'une vidéo de 15 minutes.** *Les modalités seront explicitées plus tard.*

Si vous n'êtes pas parvenus jusqu'à ce point, vous devez rendre **votre projet en l'état** à la date prévue.

Phase 11 : quelques objectifs bonus (facultatifs)

- ↬ On propose dans cette phase quelques pistes d'exploration permettant de profiter de ce projet pour **faire monter vos compétences de programmeurs**.
- ↬ Ces **objectifs** sont **facultatifs** mais feront l'objet d'une **bonification** dans l'évaluation de votre projet : ils sont à explorer en totale liberté et autonomie.
  Vous êtes également libres d'explorer d'autres pistes suivant où vous porte votre curiosité « informatique ».
  Vous devrez vous-mêmes rechercher les ressources (tutoriels, documentations) pour vous aider à comprendre l'objectif et le mettre en œuvre dans votre programme.
- ↬ Voici la liste (non exhaustive) des objectifs proposés :

- ➟ **Étudier la possibilité de créer des flottes personnalisées (nombre de vaisseaux, choix des types de vaisseaux) en utilisant notamment le principe d'allocation dynamique de mémoire.**
- ➟ **Explorer la notion de « pointeur générique » (*void pointer* en anglais) `void*` pour faciliter la gestion de choix de vaisseaux dans un conteneur `Flotte` (en évitant trop de surcharges).**
- ➟ **Explorer la gestion de fichiers (création/ouverture/lecture/écriture).** Par exemple :
  - écrire dans un fichier une partie de la sortie d'exécution (par exemple l'état de la partie à chaque tour)
  - inventorier dans un ou plusieurs fichiers toutes les « parties jouées » (à chaque exécution du programme) en précisant (suggestion) : le nom des joueurs, la date de la partie, sa durée, le nombre de tours de jeu, le vainqueur, etc.
  - écrire dans des fichiers au préalable les caractéristiques de chaque vaisseau d'une flotte, puis initialiser les flottes en allant lire ces caractéristiques dans les fichiers.
- ➟ **Améliorer le moteur de jeu gérant la visée et le tir.** Par exemple :
  - proposer un affichage graphique plus élégant du système de visée mais toujours en mode « terminal » ;
  - **aller plus loin en utilisant d'autres librairies/API (en langage C) plus graphiques pour gérer ce moteur.** Par exemple :
    - l'API **ncurses** : [`https://fr.wikipedia.org/wiki/Ncurses`] ; [`https://invisible-island.net/ncurses/`].
    - la **Simple DirectMedia Layer (SDL)** : [`https://fr.wikipedia.org/wiki/Simple_DirectMedia_Layer`] ; [`https://zestedesavoir.com/tutoriels/1014/utiliser-la-sdl-en-langage-c/la-sdl/`] ; [`https://www.libsdl.org/`].
- ➟ Explorer l'utilisation d'un outil de versionnage de code tel que **Git**, notamment via la plateforme (type « cloud ») dédiée **GitHub** : [`https://git-scm.com/`] ;[`https://github.com/`].
- ➟ Explorer l'utilisation d'un outil de génération de documentation à partir des commentaires d'un code tel que **Doxygen** : [`https://www.doxygen.nl/index.html`].
- ➟ **N'hésitez pas à explorer d'autres pistes qui vous intéressent ou à proposer d'autres améliorations du programme (autre système de jeu, enrichissement des types de vaisseaux, etc.).**

Certains de ces objectifs annexes peuvent être traités en parallèle de l'élaboration du projet. Vous pouvez organiser votre travail d'équipe pour que certains d'entre vous se dédient à ces tâches. Quoi qu'il en soit, les phases 1 à 10 restant les objectifs prioritaires et obligatoires.