Travail #1 : X-BOT Live!

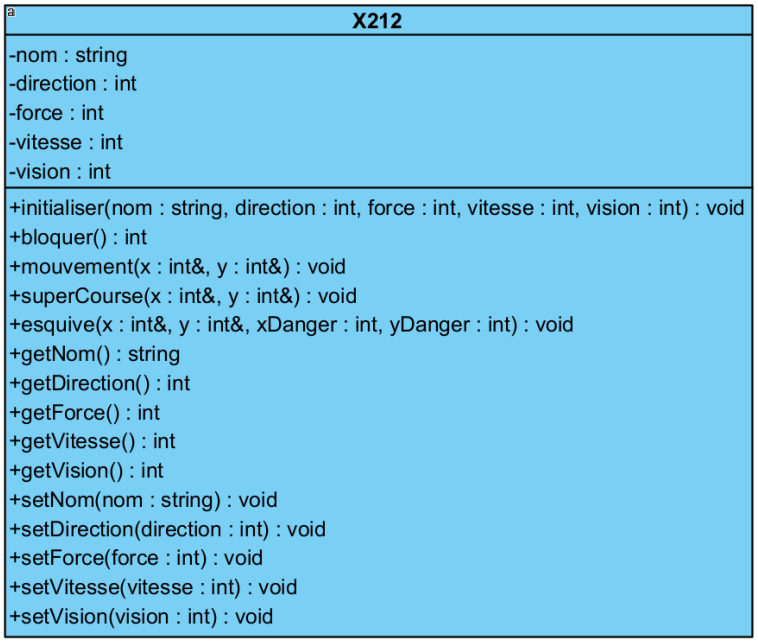
Consignes spécifiques

- Création d'un projet C++ en Console
- Ce travail se réalise individuellement ou en équipe de 2. Une seule remise par équipe
  - Votre **projet** dans Visual Studio, **compressé**
  - Supprimer les dossiers DEBUG, RELEASE et .vs de vos projets (diminuer la taille)
  - Fournir des **captures d'écran** de chaque test, elle confirme le fonctionnement de votre projet (Ajuster le main en conséquence afin que vos noms apparaissent)
  - Les captures d'écrans sont dans un dossier « captures » contenu dans votre projet.

Étape #1 : Création des X-BOTS (semaine #2)

- Il existe 6 X-BOTS différents et chacun doit être programmé
- Chaque X-BOT est défini dans ses propres fichiers de classe.
  - Un fichier .h pour la définition de classe (ex : X212.h)
  - Un fichier cpp contenant l'implémentation (ex : 212.cpp)
- Utiliser le modèle UML fourni pour créer votre structure, elle sera réutilisée par la suite
  - Utiliser des méthodes « inline » et des « const » aux endroits appropriés.
  - Vous pouvez ajouter des membres privés au besoin
- Des consignes spécifiques sont ajoutées à chaque diagramme
  - Certaines parties de code sont fournies, vous pouvez les utiliser au besoin
- Utiliser le main fourni sur Moodle pour tester vos XBOT

Le X-BOT « X212 »

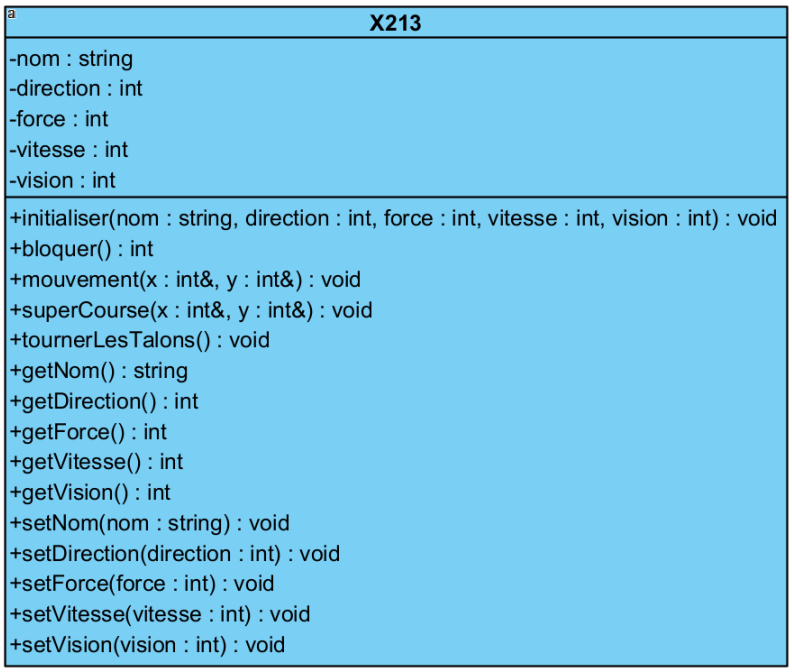


Méthode/attribut	Description/note
Initialiser	Valeur possible de direction : 0 à 3 Aucune validation sur les autres attributs
bloquer	Retour : (vitesse/3 + force)

Mouvement	On reçoit la position en référence et X212 avance dans la direction définie (voir l'attribut direction)
	Le déplacement s'effectue de la façon suivante : (vitesse + 1)
	
	Par exemple
	
	`void X212::mouvement(int& x, int& y)`
	`{`
	`  int deplacement = vitesse + 1;`
	`  switch (direction)`
	`  {`
	`  case 0: x -= deplacement; break;`
	`  case 1: y += deplacement; break;`
	`  case 2: x += deplacement; break;`
	`  case 3: y -= deplacement; break;`
	`  }`
	`  x = max(min(x, 9), 0);`
	`  y = max(min(y, 9), 0);`
	`}`

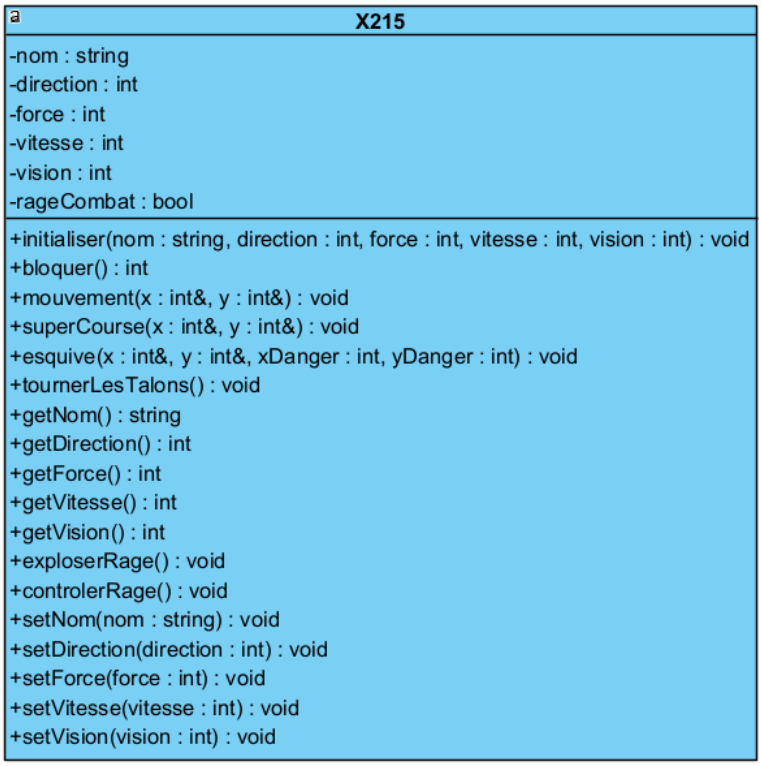
superCourse	Identique à mouvement sauf que le déplacement s'effectue de la façon suivante : (vitesse + (vitesse * force/10))
Esquive	X212 tente d'éviter un obstacle. Il peut réaliser 2 actions: changer de direction et bouger à vitesse/2. Inventer une règle simple qui lui permettra d'être à au moins 3 points du danger. S'il en est incapable, il ne bouge pas! *Cette méthode est un défi, elle ne sera pas notée. (elle peut être vide)*

Le X-BOT « X213 »



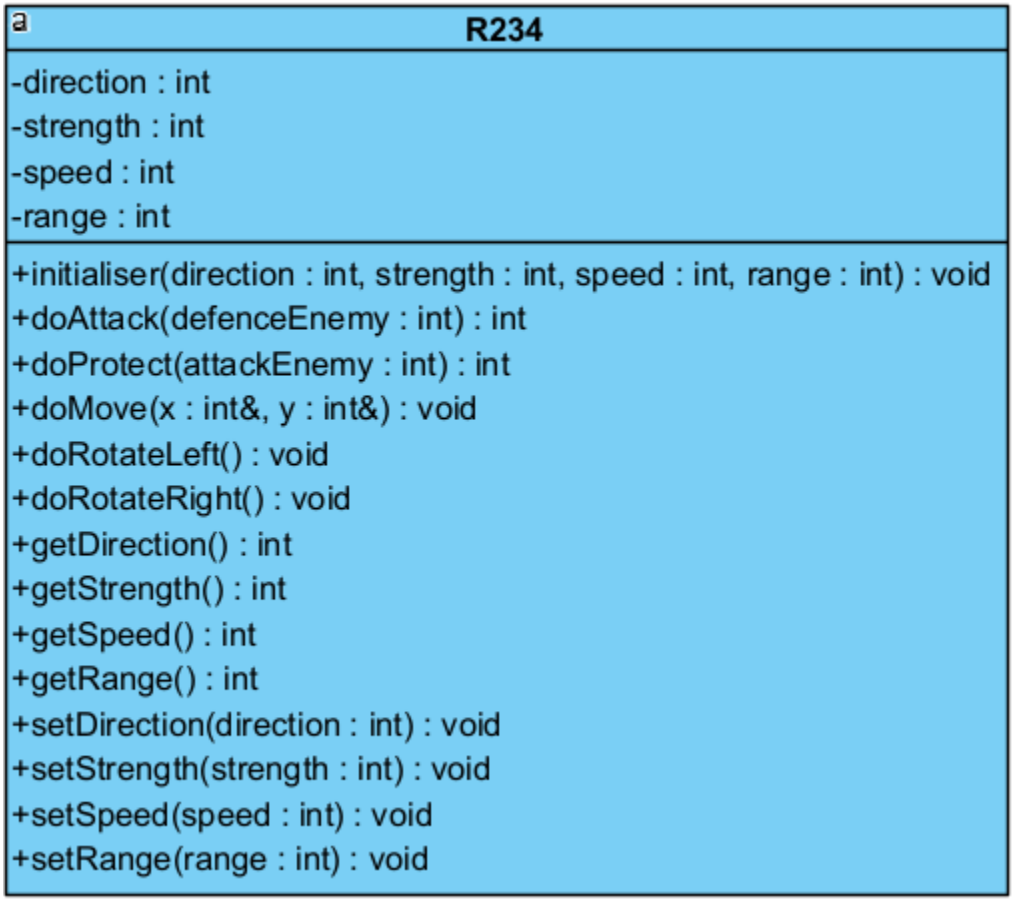
Méthode/attribut	Description/note
initialiser	Valeur possible de direction : 0 à 3 Aucune validation sur les autres attributs
bloquer	Retour : (vitesse/3 + force)
mouvement	Identique à X212
superCourse	Identique à X212
tournerLesTalons	Le X-BOT tourne de 180 degrés! (voir attribut direction)

Le X-BOT « X215 »



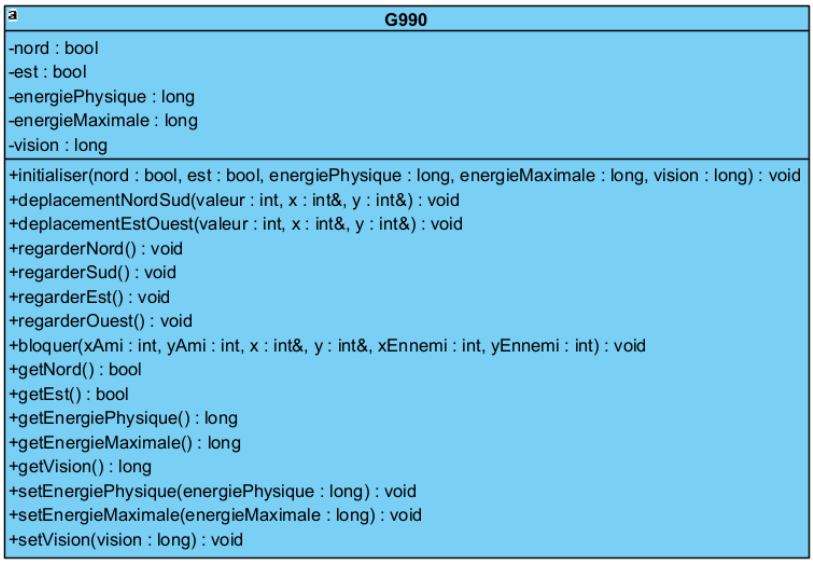
Méthode/attribut	Description/note
initialiser	Valeur possible de direction : 0 à 3 Par défaut, rageCombat est à false
bloquer	Si rageCombat = true Retour : force * 3 Sinon Retour : (vitesse/3 + force)
mouvement	Identique à X212
superCourse	Identique à mouvement sauf que le déplacement s'effectue de la façon suivante : (vitesse + (vitesse * force/10)) Si rageCombat = true Impossible de faire la super course, X215 effectue plutôt un mouvement normal
esquive	X213 tente d'éviter un obstacle. Il peut réaliser 2 actions: changer de direction et bouger vitesse/2. Inventer une règle simple qui lui permettra d'être à au moins 3 points du danger (il tourne les talons et s'enfui dans la direction opposée). S'il en est incapable, il ne bouge pas| *Cette méthode est un défi, elle ne sera pas notée. (elle peut être vide)*
tournerLesTalons	Le X-BOT tourne de 180 degrés!!!
exploseRage	rageCombat = true
controlerRage	rageCombat = false

Le X-BOT « R234 »



Méthode/attribut	Description/note
initialiser	Valeur possible de direction : 0 à 3 ~~Par défaut, rageCombat est à false~~
doAttack	Retour : (strength-defenseEnemy) Si inférieur à 0, retourner 0
doProtect	Retour : (attackEnemy-(speed+strength)/2) Si inférieur à 0, retourner 0
doMove	Avance de « speed » dans la direction courante
doRotateLeft	Changer de direction vers la gauche
doRotateRight	Changer de direction vers la droite

Le X-BOT « G990 »

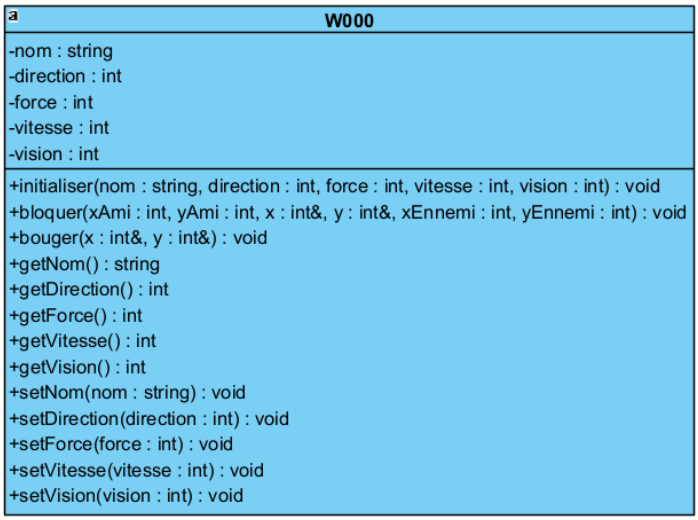


Méthode/attribut	Description/note

initialiser	Si nord = true, le X-BOT se dirige vers le nord. Sinon, vers le sud
	Si est = true, le X-BOT se dirige vers l'est. Sinon, vers l'ouest
	
	Voici un exemple de code
	`void G990::initialiser(bool nord, bool est, long energiePhysique, long energieMaximale, long vision)`
	`{`
	`  this->nord = nord;`
	`  this->est = est;`
	`  setEnergiePhysique(energiePhysique);`
	`  setEnergieMaximale(energieMaximale);`
	`  setVision(vision);`
	`}`

deplacementNordSud	La valeur reçue correspond au déplacement demandé sur l'axe nord-sud, la direction dépend de l'attribut nord Le déplacement maximum est défini par **« energieMaximale »**
deplacementEstOuest	La valeur reçue correspond au déplacement demandé sur l'axe est-ouest, la direction dépend de l'attribut est Le déplacement maximum est défini par **« energieMaximale »**
regarderNord	G990 regarder maintenant vers le nord nord = true
regarderSud	G990 regarder maintenant vers le sud nord = false
regarderEst	G990 regarder maintenant vers l'est est = true
regarderOuest	G990 regarder maintenant vers l'ouest est = false
bloquer	G990 tente de se placer entre l'ami et l'ennemi. Attention à la direction ou il regarde, il devra peut-être se tourner pour bouger. Il ne peut bouger de plus de « vitesseMaximale » Expert seulement : *Cette méthode est un défi, elle ne sera pas notée. (elle peut être vide)*

Le X-BOT « W000 »



Méthode/attribut	Description/note
Initialiser	Valeur direction possible : 0 à 3
Bloquer	W000 tente de se placer entre l'ami et l'ennemi. Attention à la direction ou il regarde, il devra peut-être se tourner pour bouger. Il ne peut bouger de plus de « vitesse» Expert++ seulement : *Cette méthode est un défi, elle ne sera pas notée. (elle peut être vide)*
Bouger	On reçoit la position en référence et W000 avance dans la direction définie. Le déplacement s'effectue de la façon suivante : (vitesse+2)

Étape #2 : Les constructeurs et destructeurs (semaine #3)

Avant de débuter cette section

- Assurez-vous d'avoir compléter l'étape #1
- Pour chaque test, assurez-vous d'avoir réalisé vos captures d'écran
- Insérer la classe « Compteur » de l'exercice formatif de la semaine 3.
    - Ajouter un #include sur cette librairie dans votre fichier main.cpp
- Mettre en commentaire toutes les fonctions de tests de l'étape #1. Elles ne compileront plus.
- Recopier le code suivant dans votre projet et adapter votre fonction « main ».

```
void testChargementXBots()
{
	X212 x212("x212", 0, 4, 3, 5);
	X213 x213("x213", 0, 4, 3, 5);
```

```
        X215 x215("x215", 0, 4, 3, 5);
        R234 r234(0, 4, 3, 5);
        G990 g990(true, false, 7, 7, 5);
        W000 w000("W000", 0, 4, 3, 5);

        X212 x212Copie(x212);
        X213 x213Copie(x213);
        X215 x215Copie(x215);
        R234 r234Copie(r234);
        G990 g990Copie(g990);
        W000 w000Copie(w000);

        cout << "X212 : " << x212.getNom() << "/" << x212Copie.getNom() << endl;
        cout << "X213 : " << x213.getNom() << "/" << x213Copie.getNom() << endl;
        cout << "X215 : " << x215.getNom() << "/" << x215Copie.getNom() << endl;
        cout << "R234 : " << r234.getStrength() << "/" << r234Copie.getStrength()
<< endl;
        cout << "G990: " << g990.getEnergiePhysique() << "/" <<
g990Copie.getEnergiePhysique() << endl;
        cout << "W000 : " << w000.getNom() << "/" << w000Copie.getNom() << endl;
}

int main()
{
        //testsXBots();                     // Semaine #2
        testChargementXBots();              // Semaine #3
        cout << Compteur::getInformation() << endl;
        cin.get();
}
```

Consignes spécifiques

- Pour chacun des 6 X-BOTS, ajouter un constructeur, un constructeur par copie et le destructeur
    - La méthode « initialiser » ne sera plus utilisée. Elle doit demeurée dans la classe pour la correction.
- Le constructeur a exactement les mêmes paramètres que la méthode « initialiser » (vous pouvez les copier). Les instructions sont également les mêmes.
- En suivant l'exemple de la classe « Lait » de l'exercice formatif, ajouter les appels de fonction du Compteur (3 par classe)
- *Astuce Visual Studio : On peut compiler une classe à la fois. Ouvrir le cpp puis F7*

```
----- ERIC ------

X212 : x212/x212
X213 : x213/x213
X215 : x215/x215
R234 : 4/4
G990: 7/7
W000 : W000/W000


------------------------------
 NB constructeurs       : 6
 NB constructeurs copie : 6
 NB destructeurs        : 12
```

Une fois vos classes complétées et compilées, exécuter le nouveau test (N'oubliez pas de prendre une capture d'écran)

Étape #3 : Les pointeurs de robots (semaine #4)

Avant de débuter cette section

- Assurez-vous d'avoir compléter l'étape précédente
- Recopier la fonction « testChargementXBots » de l'étape précédente et renommer la nouvelle fonction « testPointeursXBots »
- Recopier le code suivant dans votre projet et adapter votre fonction « main ».

```
void testeur()
{
   TesteurXBots testeur1(new X212("X212", 1, 1, 1, 1),new G990(true, true, 1, 1, 1));
   TesteurXBots testeur2(testeur1);
}

int main()
{
   //testsXBots();            // Semaine #2
   //testChargementXBots();   // Semaine #3
   testPointeursXBots();      // Semaine #4
   testeur();                 // Semaine #4

   cout << Compteur::getInformation() << endl;
   cin.get();
}
```

Fonction « testPointeursXBots »

- La fonction conserve le même rôle (création, utilisation et destruction de XBOTs)
- Transformer la fonction en respectant les consignes suivantes
  - Tous les objets de la fonction sont gérés par des pointeurs
  - Les constructeurs, constructeurs par copie et destructeurs doivent balancer
- Prendre une capture d'écran de votre test

```
----- ERIC ------

X212 : x212/x212
X213 : x213/x213
X215 : x215/x215
R234 : 4/4
G990: 7/7
W000 : W000/W000



------------------------------
 NB constructeurs       : 6
 NB constructeurs copie : 6
 NB destructeurs        : 12
```

Création de la classe « TesteurXBots»

La classe est simple (elle sera utilisée par la fonction « testeur »). Respecter les consignes suivantes (il n'y a pas de get, set ou autre fonction). Compiler et ajouter le #include au main

- Elle possède 2 attributs : un pointeur de X212 et un pointeur de G990
- Le constructeur reçoit les deux XBOTS en paramètres
- Programmer le constructeur par copie et le destructeur
- Utiliser le Compteur

Fonction « testeur »

- La fonction est appelée à partir du « main » et elle ne doit pas être modifiée (elle a seulement 2 instructions)
- Les constructeurs, constructeurs par copie et destructeurs doivent balancer
- Prendre une capture d'écran de votre test

La section #1 du projet est maintenant terminé. Vous pouvez maintenant déposer votre projet dans Moodle en respectant les consignes définies en début de document