EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communication
Cours d'Informatique à la section SV
Sam J.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

PROGRAMMATION II

Examen Semestre II

Instructions :

— Vous disposez de une heure quarante cinq minutes pour faire cet examen (10h15 - 12h).

— Nombre maximum de points: 110 .

— Indiquez votre **NUMÉRO SCIPER** sur *chacune* des feuilles. **Une feuille sans identification ne sera pas corrigée.**

— Toute documentation est autorisée, hormis les corrigés des anciens tests ;

— Répondez sur les feuilles qui vous sont distribuées à cet effet (utilisez aussi le verso des feuilles si nécessaire). **Ne répondez pas sur l'énoncé**.

— Vous pouvez répondre aux questions en français ou en anglais.

— Veillez à **ne traiter *qu'un* exercice par feuille**.

— L'examen compte 3 exercices indépendants.

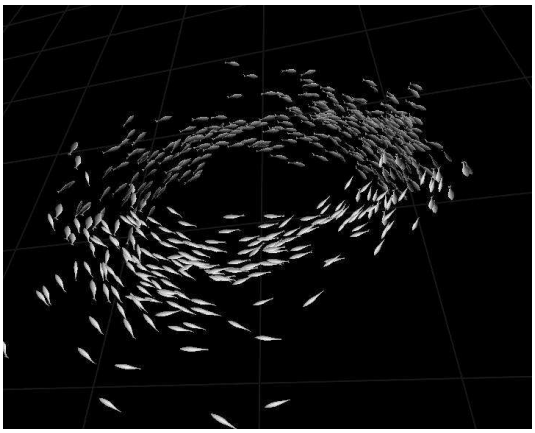
Vous pouvez commencer par celui que vous souhaitez

— Exercice 1: **56** points.
— Exercice 2: **39** points.
— Exercice 3: **15** points (+ 5 en bonus).

questions au verso ☞

Exercice 1 : Conception de programme et programmation [56 points]

On s'intéresse ici à simuler le déplacement coordonné de poissons dans un banc :



Un banc de poisson est un ensemble de poissons dont les déplacements sont régis par une force. Le mode de calcul de cette force sera explicité un peu plus loin.

Les poissons évoluent dans une monde *circulaire*, doté d'une position (un vecteur en deux dimensions) et d'un rayon. Les poissons **ne peuvent sortir** de ce monde.

Un poisson est caractérisé par :
- son poids
- son niveau d'énergie

La construction d'un poisson prendra en paramètre sa position et son poids. Il naîtra avec un niveau d'énergie donné (un nombre identique pour tous les poissons et donné dans un fichier de configuration de type `.json`: voir les contraintes de conception plus loin).

Un poisson meurt une fois son niveau d'énergie à zéro.

Un poisson sera assimilé à un corps circulaire pour réaliser les tests de collisions. Le rayon sera la moitié de la taille du poisson.

Vous anticiperez la présence de sous-classes de poissons.

Les fonctionnalités du programme de simulation devront être les suivantes:

- créer un banc poissons dont le nombre, les positions, niveaux d'énergie, et poids sont créées au hasard dans le monde (à partir de valeurs données dans le fichier `.json`).
- simuler le déplacement du banc.

Déplacement des poissons Le déplacement d'un poisson est régi par une équation de type :

$$\frac{d\overrightarrow{\mathbf{v}}(t)}{dt} = \overrightarrow{\mathbf{f}}(t) \tag{1}$$

où f est une force s'exerçant sur le poisson et conditionnant son déplacement. Cette force est la somme de trois composantes:

$$\overrightarrow{\mathbf{f}}(t) = \overrightarrow{\mathbf{f_c}}(t) + \overrightarrow{\mathbf{f_e}}(t) + \overrightarrow{\mathbf{f_a}}(t)$$

(pour simplifier, le poids sera négligé pour le calcul de la force. On considère qu'il n'est utile que pour les fonctionnalités d'affichage). Dans ce qui suit, $\overrightarrow{\mathbf{x}}(t)$ est la position du poisson et $\overrightarrow{\mathbf{v}}(t)$ sa vitesse.

1. la composante de cohésion $\overrightarrow{\mathbf{f_c}}$, permet aux poissons de rester groupés:

$$\overrightarrow{\mathbf{f_c}}(t) = c_{cohesion} * (\overrightarrow{\mathbf{c}}_{banc} - \overrightarrow{\mathbf{x}}(t));$$

   Où $\overrightarrow{\mathbf{c}}_{banc}$ est le centre du banc (position moyenne des poissons du banc), $c_{cohesion}$ est le coefficient de cohésion, une constante.

2. la composante d'évitement $\overrightarrow{\mathbf{f_e}}$ permet aux poissons de ne pas se heurter les uns aux autres

$$\overrightarrow{\mathbf{f_e}}(t) = c_{evitement} * \frac{\overrightarrow{\mathbf{d}}}{\| \overrightarrow{\mathbf{d}} \|^2}$$

   $\overrightarrow{\mathbf{d}}$ est la moyenne des distances entre $\overrightarrow{\mathbf{x}}$ et la position de chaque autre poisson.

3. la composante d'alignement $\overrightarrow{\mathbf{f_a}}$ permet aux poissons de s'aligner à la vitesse du groupe

$$\overrightarrow{\mathbf{f_a}}(t) = c_{alignement} * (\overrightarrow{\mathbf{v}}_{banc} - \overrightarrow{\mathbf{v}}(t));$$

   où $\overrightarrow{\mathbf{v}}_{banc}$ désigne la vitesse moyenne du banc (moyenne des vitesses de tous les poissons) et $c_{alignement}$ une constante

Le calcul des nouvelles positions et vitesse d'un poisson soumis à la force $\overrightarrow{\mathbf{f}}$ et après écoulement d'un pas de temps dt se fera au moyen d'un intégrateur numérique de type RG4 ou EC, **selon les mêmes modalités que celles utilisées lors de votre projet**.

Les constantes impliquées doivent pouvoir être modifiées via le fichier de configuration.

Le poisson perd de l'énergie à chaque pas de déplacement et disparaît du banc s'il meurt.

Question 1 : Conception [31 points]

1. on vous demande d'écrire le prototype de la (ou des) classe(s) permettant de représenter le problème ci-dessus en un programme C++ orienté objets ;
2. il s'agit de donner les prototypes en C++ de ces classes (attributs et **prototypes des méthodes**). Ces prototypes devront contenir les membres nécessaires pour mettre en oeuvre toutes les fonctionnalités souhaitées, **qu'elles soient explicitement demandées ou suggérées par les spécifications de l'énoncé** ;
3. dans les cas où cela n'est pas clair, vous noterez en commentaire la signification du type de retour des méthodes et ce que font les méthodes dans les grandes lignes.
4. ne négligez ni les constructeurs/destructeurs ni les droits d'accès.

Voir le verso avant de répondre ☞

Les contraintes à respecter sont :

1. Vous n'utiliserez pas de fonctions globales (mais uniquement des méthodes de classes).
2. Votre conception ne doit pas nécessiter de dupliquer du code.
3. Vous supposerez que les classes fournies dans le projet sont à votre disposition: tous les utilitaires(`Vec2d` etc.), les classes `Updatable` et `Drawable`, les générateurs aléatoires, les intégrateurs (`DiffEqSolver`, rappelé en annexe), le noyau de simulation avec la classe `Application`. Vous supposerez aussi l'existence de la classe `CircularBody`, telle que codée dans le projet.
4. Vous supposerez que lien de votre conception avec la classe `Application` se fait par un objet modélisant le monde et accessible via la fonction `getAppEnv()`. Vous supposerez aussi l'existence de la fonction `getAppConfig()` permettant l'accès à un fichier de configuration de type `.json`
5. Votre conception doit comporter tous les attributs et méthodes nécessaires à la mise en oeuvre de la simulation souhaitée.
6. Vous indiquerez quelles fonctionnalités parmi celles que vous proposez auront recours à `getAppConfig` et/ou `getAppEnv` et pourquoi.
7. Vous inclurez aussi des méthodes de dessin là où cela vous semble pertinent.
8. il n'est pas nécessaire de préciser les inclusions C++.
9. vous porterez une attention particulière à la nature des méthodes (normales, const, virtuelles ou virtuelles pures), au liens d'héritage ainsi qu'aux droits d'accès aux attributs et aux méthodes.

On ne vous demande pas d'écrire de programme complet, ni le corps des méthodes, mais uniquement leurs prototypes. Il est inutile se spécifier des directives d'inclusion.

Question 2 : Programmation [10 points]

Au vu de votre conception, donnez le corps de la méthode permettant le calcul de la force régissant le déplacement d'un poisson.

Question 3 : Extension de la conception [15 points]

On souhaite ajouter des sources de nourritures immobiles qui seraient générées aléatoirement dans le monde (assimilables à des corps circulaires). Par exemple, à chaque pas de simulation on décide de générer `m` sources de nourritures à des positions aléatoires. `m` serait aussi tiré au hasard.

Les poissons vont alors cibler ces sources de nourriture lors de leurs déplacements.

Un composante supplémentaire devra alors être ajoutée à la force :

$$\overrightarrow{\mathbf{f_{cible}}}(t) = c_{cible} * (\vec{\mathbf{x}}_{nourriture} - \vec{\mathbf{x}}(t));$$

où c_{cible} est une constante et $\vec{\mathbf{x}}_{nourriture}$ est la position de la source de nourriture la plus proche de $\vec{\mathbf{x}}(t)$.

Une source de nourriture est caractérisée par une quantité (qui conditionnera la taille du corps circulaire à dessiner) et une position.

1. Comment proposeriez-vous d'étendre votre conception pour intégrer les sources de nourriture à votre simulation ? Indiquez ce que vous ajoutez et à quel endroit. Si une nouvelle classe est ajoutée donnez sont contenu (attributs et prototypes de méthodes).

2. Comment feriez-vous, en ajoutant une sous-classe aux poissons, pour intégrer la nouvelle composante au calcul de la force sans toucher au code existant (pour le calcul de la force) et sans dupliquer de code?

3. Comment l'attraction exercée par la source de nourriture la plus proche peut-être connue de cette sous-classe ?

4. Donnez le code du calcul de la force dans cette sous-classe.

suite ☞

Exercice 2 : Questions de cours [39 points]

Répondez clairement et succinctement aux questions suivantes :

① **[10 points]** Soit le programme suivant :

```
0.  class A {
1.  public:
2.      A() {
3.          cout << "A()" << endl;
4.      }
5.      ~A() {
6.          cout << "~A()" << endl;
7.      }
8.  };

9.  class B : public A {
10. public:
11.     B() {
12.         cout << "B()" << endl;
13.     }
14.     ~B() {
15.         cout << "~B()" << endl;
16.     }
17. };

18. int main() {
19.     B* ptr_b(new B());
20.     delete ptr_b;
21.     {
22.         B b;
23.     }
24.     cout << "fin" << endl;
25.     return 0;
26. }
```

(a) Qu'affiche t-il ? expliquez brièvement ce qui se passe lors des affichages.

(b) Que devient l'affichage si l'instruction `delete ptr_b` (ligne 20) est déplacée juste avant le `cout << "fin" << endl;` ?

(c) Que se passe t-il si l'on ne met pas l'instruction `delete ptr_b` ?

(d) Que se passe t-il si l'on ajoute l'instruction `delete b;` juste après la ligne 22 ? juste après la ligne 23 ?

② **[14 points]** Soit le programme suivant (on suppose que toutes les inclusions nécessaires sont faites) :

```
1.  class A {
2.  public :
3.      void m1() {
4.          cout << a_ << endl;
5.      }
6.      virtual void m2() {
7.          ++a_;
8.      }
9.  private:
10.     int a_ = 1;
11. };

12. class B : public A {
13. public :
14.     void m1() {
15.         m2();
16.         cout << b_ << endl;
17.     }
18.     void m2() {
19.         ++b_;
20.         A::m1();
21.         m2(2);
22.         cout << b_ << endl;
23.     }

24.     void m2(int i) {
25.         b_+= i;
26.     }
```

```
27. //Suite de la classe B
28. private:
29.     int b_ = 2;
30. };

31. int main() {
32.     A* b(new B());
33.     b->m1();
34.     b->m2();
35.     return 0;
36. }
```

(a) Qu'affiche t-il ? expliquez brièvement ce qui se passe lors de ces affichages.

(b) Qu'affiche t-il si l'on supprime le mot clé `virtual` dans la déclaration de `A:.m2()` ? Justifiez.

(c) Pour chacune des méthodes suivantes, indiquez si elle **ne peut pas** être marquée `const` : `A::m1()`, `A::m2()`, `B::m1()`, `B::m2()`, `B::m2(int)` ? Justifiez brièvement.

(d) Pour chacune des méthodes suivantes, indiquez si elle **ne peut pas** être marquée `override` : `A::m1()`, `A::(m2()`, `B::m1()`, `B:m2()`, `B::m2(int)` ? Justifiez brièvement.

(e) Quelle instruction choisiriez vous de mettre dans `B::m1()` pour que cette méthode affiche aussi la valeur de `a_`;

i. `cout << a_ << endl;;`

ii. `A::m1();`

Justifiez votre choix.

(f) Peut-on ajouter l'instruction `b->m2(2);` au programme principal ? Justifiez votre réponse.

suite ☞

③ **[7 points]** Soit le programme suivant (on suppose que toutes les inclusions nécessaires sont faites) :

```
1.  class A {
2.
3.  };

4.  class B : public A {
5.  public:
6.     void affiche(ostream& out) const {
7.         out << "Je suis un B" << endl;
8.     }
9.  };

10. class C : public A {
11. public:
12.   void affiche(ostream& out) const {
13.       out << "Je suis un C" << endl;
14.   }
15. };
```

```
16. int main()
17. {
18.     A* b(new B());
19.     cout << b << flush;
20.     A* c(new C());
21.     cout << c << flush;
22.     return 0;
23. }
```

Indiquez quelles lignes de code ajouter et où pour que le `main` tel que fourni, affiche :

```
Je suis un B
Je suis un C
```

en réutilisant les méthodes `affiche` existantes.

④ **[6 points]** Vous concevez un jeu peuplé de personnages divers qui peuvent être des guerriers. Chaque guerrier possède sa propre arme. Un guerrier de ce jeu peut être cloné. Le guerrier issu du clonage doit appartenir au même bataillon.

```
class Guerrier {
private:
    Arme* arme;
    Bataillon* bataillon;
};
```

Dans ce contexte, comment proposez-vous d'écrire le constructeur de copie de la classe `Guerrier` ci-dessus:

(a) s'il n'y a pas de sous-classes de `Arme` ?

(b) s'il y a des sous-classes de `Arme` ?

Donnez le code en C++. Si le code contient des appels à des méthodes annexes, expliquez simplement le rôle de ces méthodes en français et pourquoi vous les avez introduites.

⑤ **[2 points]** Dans le contexte d'un jeu, un programmeur a écrit l'ébauche de code suivante :

```
class Personnage {
public:
   virtual void fight(Personnage&) = 0;
};

class Guerrier : public Personnage {

public:
    void fight(Personnage&) {
     // code ici
    }
    ~Guerrier() {
        delete arme;
        delete bouclier;
    }

private:
    Arme* arme;
    Bouclier* bouclier_;
    //...
};
```

Le jeu est conçu de sorte à ce que lorsqu'un guerrier meurt on enterre avec lui arme et bouclier et le destructeur est programmé en conséquence.
Si le jeu doit travailler avec des collections de `Personnage*` que manque t-il à ce code pour bien gérer la mort des guerriers ? Justifiez votre réponse.

suite ☞

Exercice 3 : Déroulement de programme [15 points] (+ 5 en bonus)

Le programme suivant compile et s'exécute sans erreurs. Qu'affiche-t-il ? Justifiez votre réponse en expliquant les points *importants* (ne paraphrasez pas le code, mais montrez que vous avez compris !).

Que devient l'affichage si l'on ajoute `private: int a = 1;` à la classe `B` ? (bonus)

```
1.  #include <iostream>
2.  using namespace std;

3.  class A {
4.  public:
5.      A(int val)
6.          :a(val) {
7.          cout << "A() : " << a << endl;
8.      }

9.      void m1(int i) {
10.         a += i;
11.         cout << "A::m1 : " << a << endl;
12.     }

13.     virtual void m2() {
14.         m1(2);
15.         m3();
16.         m4();
17.         cout << "A::m2 : " << a << endl;
18.     }

19.     virtual void m3() {
20.         ++ a;
21.         cout << "A::m3 : " << a << endl;
22.     }

23.     virtual void m4() = 0;

24.     virtual ~A() {
25.         cout <<  "~A" << endl;
26.     }

27. protected:
28.     int a;
29. };

30. class B : public A {
31. public:
32.     B(int val)
33.         :A(val) {
34.         cout << "B() : " << a <<  endl;
35.     }
36.     B()
37.         :B(2) {
38.     }
```

```
39.     //Suite de la classe B
40.     void m1(int i, int j = 2) {
41.         a = i + j;
42.         cout << "B::m1 : " << a << endl;
43.     }

44.     virtual void m2() {
45.         m1(1);
46.         A::m2();
47.         cout << "B::m2 : " << a << endl;
48.     }

49.     void m4() {
50.         a += 2;
51.         cout << "B::m4 : " << a << endl;
52.     }
53.     virtual ~B() {
54.         cout <<  "~B"  << endl;
55.     }
56. };

57. void test(A& a, int val) {
58.     cout << "1-> " << flush;
59.     a.m1(val);
60.     cout << "2-> " << flush;
61.     a.m2();
62.     cout << "3-> " << flush;
63.     a.m3();
64.     cout << "4-> " << flush;
65.     a.m4();
66. }

67. int main() {
68.     B b;
69.     cout << "test:" << endl;
70.     test(b, 2);

71.     return 0;
72. }
```