Annexe : Règles de développement

Langue

On écrit notre code en anglais, pour que le projet puisse continuer à être exploité par la suite à un niveau international.

Indentation

- **Tabulation à 4 espaces**, et non le caractère '\0x9' (TAB).

Espaces de noms

- On n'utilise jamais les mots-clé 'using' ni 'using namespace'.>

Fichers headers

- On les nomme avec l'extension **.hpp**. Un Header typique (**ici header.hpp**) sera de la forme :

```
#ifndef HEADER_HPP
#define HEADER_HPP

/* Insérer des déclarations de classe
   et de prototypes de fonction ici. */

#endif /* HEADER_HPP */
```

Variables

- On nomme nos variables avec des minuscules séparées par des underscores.

```
int my_var;
```

- Les constantes '#define' ou 'const' écrites en majucules.

```
const int MY_CONST = 42;
#define YOUR_CONST 13
```

- On nomme un tableau par le pluriel des éléments qu'il représente :

```
unsigned phone_numbers[8];
std::string thread_arguments[16];
```

- Lorsqu'on veut un tableau extensible (dont on veut facilement augmenter/diminuer la taille), on utilise un std::vector.

```
std::vector<int> my_telephones;
```

Fonctions/Méthodes

- On les nomme en camelCase.

```
void myFunction();
int anotherCoolFunction();
```

Classes

- On les nomme en CamelCase en commençant par une capitale.

```
class HautParleur;
```

Nouveautés du C++

- **Références.** En C++, on dispose de variables de pointeur, de valeur, mais aussi de **références**.
  Les accesseurs renverront une référence sur un objet seulement si un retour de valeur n'est pas pratique ou coûteux.

```
/* Déclaration d'une variable de référence. */
int& ma_variable;
/* Ce getter renvoie une référence vers un attribut
   (et non une copie de sa valeur). */
int& getVar();
/* Lorsqu'on la modifiera, l'attribut sera modifié,
   comme avec un pointeur. */
```

- **On n'utilise pas malloc() et free().** En C++, nous avons les opérateurs **new** et **delete**.

- On peut instancier un objet sans qu'il soit dynamique. (i.e sans **new**)
  Ainsi, tant que possible, **nous ne ferons pas d'allocation dynamique.**

```
#include <cstdlib>
#include <SFML/System.hpp>

void threadFunc(void *arg) {}

int main(int argc, char *argv[]) {
    char c;
    /* Comme une variable normale,
           elle sera détruite à la fin de la fonction. */
    sf::Thread my_thread(&threadFunc, &c);
    /* Ca marche aussi. Pas besoin de new. */
    sf::Thread my_thread2 = sf::Thread(&threadFunc, &c);
    exit(EXIT_SUCCESS);
}
```