Modifications exécutées dans le cadre du test de C++

Noms : Mevolhon Claire

Rapport

Question 1 : Nested Headers

Énoncé

Currently for each class the column header is only "Name.field" Where name is the name of the object and "field" is the name of the data that is logged.

We would like to have a more complete information of the hierarchy of data in the header of the columns.

For example instead of "Foot.Position" we would like to have "JohnnyFive.LeftLeg.Foot.Position".

Ce que j'ai modifié

J'ai ajouté un paramètre string nommé prefix à chaque méthode formatHeader. Ce prefix doit être soit une chaîne vide, soit contenir le chemin précédent suivi d'un point '.' . Il sera affiché au début de la chaîne de sortie.

J'ai donc modifié les méthodes en cascade qui appellent cette méthode formatHeader. En effet elles ajoutent désormais l'attribut nom_ de leur classe à l'attribut prefix.

Pour ce faire j'ai également changé la visibilité de l'attribut name_ de la classe AbstractLogObject (private →protected) pour que celui-ci soit visible depuis ses classes filles.

Question 2 : Generic API

Énoncé

For now if we add a new private member to some class, for example if we added a third leg to `Robot`, we would need to change the `formatHeader` and `getCurrentValue` of the Robot class.

This is not very generic and error prone (what if we mix the order in wich we call formatHeader and getCurrentValue?).

Change the code so that it is simpler to add a new attribute which is already derived from `AbstractLogObject`.

Ideally we would like to handle most of the task in the Abstract class and avoid having to redefine the virtual methods.

You can apply the same pattern for formatHeader, getCurrentValue and compute.

hint: Think how to handle the parenting relationship between objects. You could add a method "addChild()" somewhere.

Think polymorphism.

CE QUE J'AI MODIFIÉ

J'ai ajouté un attribut `std::set<AbstractLogObject*> complexObects_` dans la classe abstraite AbstractLogObject. Il est complété dans les classe filles pour contenir leur différents attributs dits complexes.

Ainsi, dans la méthode `formatHeader` j'itère dans cette liste plutôt qu'écrire à la main chaque attribut.

Le code étant ainsi identique pour chaque classe fille, j'ai donc modifié seulement la classe parente et supprimé la méthode dans les classes filles.

```
void const AbstractLogObject::formatHeader(std::string &header, std::string const& prefix)
{
    // log our private int values of the AbstractLogObject's attributes
    std::string result("");

    for (std::set<std::string>::const_iterator it = headers_.begin(); it != headers_.end(); it++)
    {
        std::string tmp = prefix + name_ + "." + *it + "\t";
        header += tmp;
    }

    // Then call the method for complex objects of the descendant class.
    std::string const prefix2 = prefix + name_+ '.';
    for (std::set<AbstractLogObject*>::const_iterator it = complexObects_.begin(); it != complexObects_.end(); it++)
        (*it)->formatHeader(header, prefix2);
}
```

Attention : Désormais, lors de l'ajout d'un attribut dans une classe, il ne faut pas oublier de l'ajouter dans cette liste `complexObject_` lors du constructeur.

Par la suite, pour les méthodes `compute` et `getCurrentValues`, j'ai créé les méthodes associées `computeAttributes` et `getCurrentAttributesValues` , pour lesquels je rappelle les méthodes `compute` ou `getCurrentValues` pour chacun des attributs.

La méthode compute n'est plus abstraite : par défaut elle appelle computeAttributes.

J'ai donc ainsi utilisé ces nouvelles méthodes dans les classes filles.

QUESTION 3 :

ÉNONCÉ

For now the program is only able to log int variables.

But in a real situation we would like to log variables with other types.

How would you do it for standard types : float, bool, unsigned int etc ?

Update the code with your new system.

Add some fields with theses types in some classes to show it works.

For example you can change "Position" from int to Float, add a boolean flag "IsEnabled" in the motors etc.

CE QUE J'AI MODIFIÉ

Le plus intuitif serait de créer un template pour ces méthodes. Malheureusement elles sont virtuelles et C++ n'accepte pas de méthode générique virtuelles.

Après quelques essais, je ne suis pas parvenu à quelque chose de bien…

Je suis donc passé à la question suivante.

QUESTION 4 : OPTIONAL VALUES

ÉNONCÉ

Ok now that we have a good generic system, we want to add a feature to make some values optional.
In a real scenario, we can have a lot of values to log, and we don't always need to log everything.
Basically the sensor values are important to keep, but the result of algorithms that are based on
these sensor values can be reconstructed afterwards.
Change the AbstractClass to add a flag for each value saying if it's optional or not.
Add a way to tell the program to enable or disable the optional values.

It should be set before the simulation starts, because once the headers are generated we won't be able to remove them or the data won't make sense. So it is set once and never changed after.

ANNALYSE

J'aurais aimé je pense un peu plus de détails pour ce problème afin de mieux comprendre la manière de foncionner et peut-être discuter des dépendances entre les variables.

De même j'aurais aimé discuter sur les pointeurs. Les attributs sont créés dans les classes filles et la classe mère contient à la fois une liste de string comportant les noms de variables et une hashMap avec le nom de ces attributs et la valeur correspondante. J'aurais aimé savoir l'avantage de cette méthode comparé à mettre directement des pointeurs vers les variables de la classe fille ? (un peu comme j'ai fait pour les attributs « complexes ») . J'ai hésiter à changer et mettre des pointeurs mais j'ai peur qu'il y ait une raison derrière, par exemple à cause d'accès concurrents si on parallélise ?

CE QUE J'AI MODIFIÉ

Désormais l'utilisateur décide dès le début à l'appel du programme quelles données il veut afficher (toutes, seulement les non optionnelles, ou choisir plus précisément).

```
C:\Users\canne\Documents\test_cpp\git\testCpp\Default>simulator -help
Usage : simulator [-all | -some | -nothing | -help]
no tag : it will do the same than with the tag -nothing
 -all : all the variables will be computed and printed
 -some : only some variables will be printed. The program will ask you which ones.
 -none : only the non optional variables will be computed and printed.
C:\Users\canne\Documents\test_cpp\git\testCpp\Default>
```

J'ai également différencié le fait que la donnée soit optionnelle et le fait qu'elle soit affichée, car selon moi il ne s'agissait pas de la même chose.

J'ai supposé que certaines variables devaient obligatoirement toujours être calculées : c'est une vérité qui concerne directement l'attribut et ceci est donc défini directement par la classe comportant cet attribut. Pour cela j'ai modifié la méthode **`registerValue`**`(std::string const& name, bool isOptional)` qui prend désormais en paramètre un booleen à true si la variable est optionnelle. J'ai stocké ces informations dans la map `optional_`. Toutes les valeurs non optionnelles seront automatiquement calculées. (modification effectuée dans la méthode compute.)

Le fait qu'une variable doive être affichée est stocké dans une autre map : `isToLog_` .
Le main appelle la fonction **`initialiseIsToLog`** du robot en fonction de l'option choisie pour initialiser cette map. Par la suite, les méthode formatHeader et getCurrentValues ont été modifiées pour ne traiter que les valeurs à afficher.

Ceci permet de créer plus tard au besoin des dépendances entre variables. Par exemple si on veut afficher une variable B qui dépend de la variable A, nous pouvons modifier optional_[A] = false sans que la variable A ne soit affichée.
Pour cela, le plus simple serait de redéfinir la classe **`initialiseIsToLog`** dans la classe fille en appelant dedans la méthode de la classe parente puis en modifiant la valeur optional_[A].

QUESTION 4 BIS :

ÉNONCÉ

What is interesting with a simulation is that we can replay several times the scenario.

Change the program so that we can store its output to a file.

Then add a way to load the values from an existing file and replay the program,

it should output a new log file.

The idea behind this is that we can load values from sensors and then check that internal algorithms behave as expected given these sensor values.

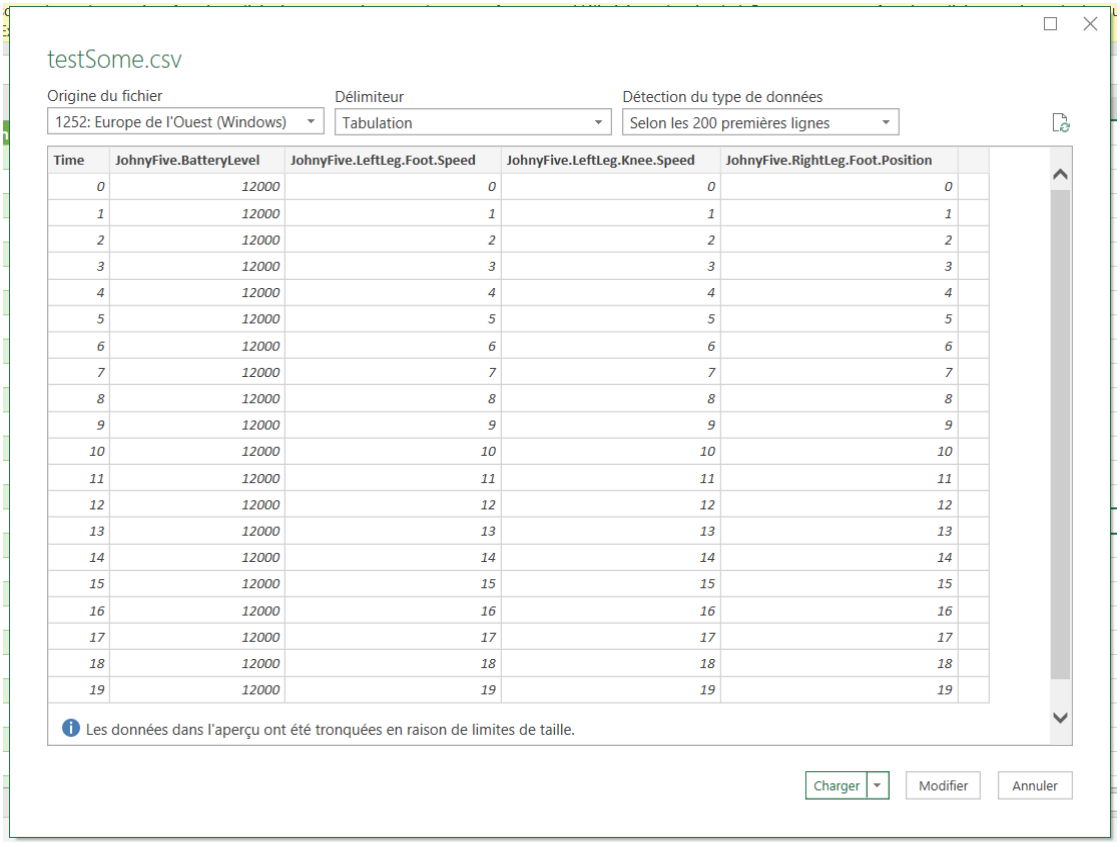
CE QUE J'AI MODIFIÉ

Ajout d'un nouveau tag !

```
simulator -save .\testSome.csv -some
```

On peut désormais ajouter le tag -save suivi d'un nom de fichier. Attention si ce fichier existe déjà, celui-ci sera écrasé.

Je conseille de sauvegarder dans un fichier CSV :) car par la suite il est possible de l'importer depuis par exemple excel en précisant le délimiteur 'tabulation', et on obtient ainsi chaque données dans une vraie colonne excel :D



Time	JohnyFive.BatteryLevel	JohnyFive.LeftLeg.Foot.Speed	JohnyFive.LeftLeg.Knee.Speed	JohnyFive.RightLeg.Foot.Position
0	12000	0	0	0
1	12000	1	1	1
2	12000	2	2	2
3	12000	3	3	3
4	12000	4	4	4
5	12000	5	5	5
6	12000	6	6	6
7	12000	7	7	7
8	12000	8	8	8
9	12000	9	9	9
10	12000	10	10	10
11	12000	11	11	11
12	12000	12	12	12
13	12000	13	13	13
14	12000	14	14	14
15	12000	15	15	15
16	12000	16	16	16
17	12000	17	17	17
18	12000	18	18	18
19	12000	19	19	19

RECHARGEMENT D'UN FICHIER :

Je ne suis pas sûre de bien comprendre ce qui est demandé ici.
Nous avons désormais dans un fichier les différentes valeurs en fonction du temps des attributs que nous voulions sauvegarder.

Doit-on dans ce cas pouvoir décider de sauvegarder/afficher d'autres attributs au besoin mais de ne « compute » que les attributs qui ne viennent pas du fichier donné en argument ?

Dans ce cas, pour chaque temps, je compute seulement les attributs demandés par l'utilisateur et qui ne sont pas présents dans le fichier, et je récupère à chaque temps les valeurs du fichier.
Ceci permet donc en cas de dépendance entre attributs, de potentiellement compute un attribut en fonction d'un autre dont la valeur est fixée par le fichier.

J'ai donc, dans le cas de la lecture d'un fichier, charger les headers dans un vecteur, puis j'ai traité un à un chacun des temps du fichier.

J'ai considérer que les temps devaient être les mêmes que ceux du fichier.
À chaque temps (ie ligne du fichier), je parcours l'arborescence du robot pour mettre à jour la map values_ et je mets en même temps la map optional_ à true pour chacune de ces valeurs pour ne pas les « compute ». Puis j'appelle la méthode r.compute normalement.