SimTaxi : Standards de qualité

28 décembre 2002

Table des matières

1 Remarque préliminaire

Ce document a été écrit dans le but d'uniformiser l'écriture du code source de SimTaxi. Ceci est nécessaire pour les raisons suivantes :

- Chaque développeur a ses propres règles pour écrire du code. Pour cette raison il est parfois difficile de relire le code de quelqu'un d'autre. Utiliser les mêmes conventions assurera une compréhension plus facile du code.
- Les standards de qualité permettent d'obtenir du code plus robuste qui facilite la détection des bugs.
- Il est plus facile de reprendre un projet où des standards de qualité qui ont été définis.

2 Règles de qualités

Cette section décrit les différents standards de qualité à appliquer au projet SimTaxi.

2.1 Noms des fichiers

Les noms de fichiers Python (module) devront respecter le nom de la classe principale qu'ils contiennent afin d'éviter les problèmes de portabilité entre les différents systèmes d'exploitation. Le nom des autres fichiers sont en minuscule ainsi que les extensions.

Rappelons qu'en Python un fichier est un module (contenant des classes, des fonctions, ...) et qu'un dossier est un paquetage.

2.2 Tabulations

Les tabulations sont de longueur 4 (utiliser des espaces). La longueur d'une ligne ne doit pas excéder 78 caractères.

2.3 Expressions

Chaque opérateur mathématique et l'opérateur d'affectation doivent être précédés et suivis d'un espace. Une virgule (',') est toujours suivie d'un espace.

2.4 Noms

Les noms ne doivent pas comporter de souligné ou de caractères spéciaux. La séparation des mots dans un identificateur est indiqué grâce à une majuscule.

Un nom de classe commence toujours par une majuscule. Tous les autres identificateurs commencent par une minuscule.

Exemples :

```
# Définition d'une classe
class SimTaxiApp :

 # Déclarations de fonctions et de variables
 def maFonction(unEntier, ...):
        maVariableEntiere = unEntier
```

2.5 En-têtes

Chaque fichier doit comporter l'en-tête suivante :

```
#!/usr/bin/env python
"""
Description succincte du module (1 ligne maximum) terminée par un point.

Description plus complète de ce module :
-A quoi sert-il?
-Les classes qu'il contient
-Principe de fonctionnement
-Comment l'utiliser

$Id$
"""

__version__ = "$Revision$"
__author__ = "EI5a, eivd, SimTaxi (Groupe Burdy)"
__date__ = "2002-10-30"
```

Les identifiants mis entre $ seront automatiquement mis à jour par le CVS lors des commit. Il ne faut plus les toucher après la création du fichier.

Les classes devront comporter l'en-tête suivante :

```
"""
Description succincte de la classe (1 ligne maximum) terminée par un point.

Description plus complète de la classe :
-A quoi sert-elle?
-Principe de fonctionnement
-Comment l'utiliser
"""
```

Cette en-tête se place sous la ligne de déclaration de la classe.
Exemple :

```
class MaClasse
    """
    Une classe d'exemple.

    Cette classe ne sert que pour exemple. Elle n'apporte aucune fonctionnalité
    et n'est pas utilisable.
    """
```

Enfin les en-têtes de méthodes se présentent sous cette forme :

```
def methode(self, param):
    """
    Courte description sur une ligne finie par un point.

    Description plus complète si nécessaire.

    nom (type) -- Description du paramètre

    retourne (type) -- Description de l'objet retourné par la fonction
                       (si elle en retourne un)

    - depuis - n°version : Depuis quelle version cette méthode existe

    - autheur - Nom
    """
```

Exemple :

```
def methodeExemple(self, param1, param2):
    """
    Cette méthode n'a pas d'action.

    param1 (tuple(int, list)) -- Tuple (nombre de bagages, clients)

    param2 (Taxi) -- le taxi pour ...

    retourne (bool) -- si le taxi peut le faire...

    - depuis - 1.0

    - depuis - Vincent Decorges
    """
```

2.6 Description des types

Dans le cas où un type n'est pas connu, il n'est pas utile de le préciser.

Les types les plus courants en Python sont : `tuple, list, int, float, dict, str, ...` pour connaître le type d'un objet il existe toujours une fonction `type`.
Exemple :

```
>>> type("blabla")
<type 'str'>
```

Le type booléen n'existe pas. Il est représenté par des entiers. Mais nous le considèreront quand même comme un type. Rappelons que 0 = faux, tout le reste = vrai.

Tout comme l'exemple de la rubrique En-Têtes il est commode de préciser les types contenus dans un tuple.

2.7 Commentaires

Les commentaires peuvent contenir des caractères accentués.

Les commentaires entre `"""` décrivant nos En-Têtes, sont utilisés pour générer la documentation technique. La phase de génération comporte un formatage de texte. Ce qui explique pourquoi il faut mettre une ligne vide entre deux phrases pour passer à la ligne (ce qui peut paraître lourd dans le code). Les détails de ce formatage ne sont pas décrit dans ce document, mais vous pouvez déjà visionner (dans la doc technique) le résultat d'un `--` en milieu de ligne ou d'un `-` en début de ligne.