Forum: Moteur de jeu GameBlender et alternatives
Topic: Bonne conception des classes (vertes, pour les enfants :-D)
Subject: Re: Python - l'instruction return

PostÃ© par: DaWaaaaghBabal
Contribution le : 23/3/2018 14:51:19

Alors là je vais te gronder : **tous tes noms doivent être explicites**. Toto, ça ne veut rien dire. On ne peut pas lire "toto" et savoir ce que c'est censé faire ou à quoi ça sert.

Il faut savoir qu'en réalité, quand on programme, on ne passe que 10% de son temps à écrire du code. Les 90% qui restent, on les passe à en (i]lire[/i]. Pour comprendre comment il marche, comment on l'utilise, ou pourquoi il ne marche pas. Donc, tous les raccourcis qui rendent l'écriture plus facile, plus rapide, au détriment de la clarté, de la lisibilité et de la compréhensibilité du code, sont à proscrire.
Et en plus, c'est gratuit, la longueur des noms n'a aucun impact sur les performances.

Pour la même raison, ta classe Text est mal nommée. Un objet de la classe Text a pour charge de stocker des morceaux de texte et de les afficher. Ce n'est donc pas un texte.
Tu devrais renommer toto en Text, comme c'était avant, et renommer cette classe principale en, je sais pas, TextWriter par exemple.

Ah, et un nom de classe commence, par convention, par une majuscule. Ce n'est pas "toto" mais "Toto"

Passées les questions de forme, attaquons-nous au fond. Un principe important en programmation en général, et tout particulièrement en POO, c'est la séparation des responsabilités. Globalement, chaque élément du code, que ce soit une classe, une méthode ou carrément une bibliothèque, doit s'occuper d'une seule tâche. Plus le code est découpé, plus il est modulaire, et plus il est facile à étendre, modifier, corriger et débugger.

En gardant cette idée en tête : dans son constructeur, un objet Text modifie une propriété de l'objet auquel le script est attaché. Non mais de quoi je me mêle ? C'est un double problème : 1/ ça déborde du rôle d'un constructeur, qui est là pour construire l'objet, pas plus, et 2/ ça outrepasse le rôle d'un Text, qui est là pour stocker des textes et les afficher, pas pour modifier son voisin. Cette modification d'un champ de l'objet parent doit dégager.

D'ailleurs, ça pose un autre gros problème : ton Text est une classe indépendante. Donc, on peut l'instancier un peu comme on veut, en principe. Sauf que nom, puisqu'il faut qu'on soit dans un contexte où getCurrentController() va renvoyer quelque chose de cohérent. Cette contrainte 1/ ne résulte pas de l'intention du code, pour afficher du texte normalement on se moque du contexte, et 2/ n'est pas explicite. On ne le sait pas à la lecture du code, aucune règle de syntaxe ne l'indique, il faut le deviner.

Ensuite, autre responsabilité que le Text ne devrait pas avoir, et son constructeur encore moins, c'est qu'il assigne une variable globale (bge.logic.getCurrentScene().post_draw),

potentiellement utilisée par beaucoup de monde, et *remplace* sa valeur. Post_draw est censé être une liste de méthodes : on inscrit des méthodes pour qu'elles soient appelées au bon moment. Tel que tu le fais ici, tu écrases la liste, autrement dit tu désinscris toutes les méthodes inscrites précédemment. C'est un coup à tout casser. Et là encore, s'inscrire auprès de la scène, ce n'est pas son travail.

On pourrait aussi dire que stocker des tronçons de texte *et* les afficher, c'est trop. La responsabilité du stockage étant très simple, on peut la laisser où elle est, mais dans l'idéal il faudrait la virer.

Parlons maintenant du stockage. Pourquoi dans le dictionnaire ? Pourquoi est-ce que ce Text devrait stocker ses données dans une variable globale ? La liste des textes à afficher n'intéresse à priori que lui. Autant qu'il la garde pour lui plutôt que d'étaler ses affaires partout. C'est à ça que sert l'état d'un objet, donc ses champs. D'ailleurs, en l'état, ton code ne devrait pas marcher. Tu références dico['list_text'] pour y ajouter des éléments, mais à aucun moment cette variable n'est initialisée. Tu devrais avoir une erreur de type pointeur null. Est-ce que dico['list_text'] est initialisé ailleurs ?

Pour régler tout ça, il faut changer un peu l'architecture. On va découper Text en TextList et TextWriter ; Toto devient TextItem qui est un peu plus clair. Ces trois classes finissent dans un module, qu'on appellera textwriter.py. Toute mention du dictionnaire disparaît, et on n'importe même plus le module bge. Ce qu'on a à présent, c'est un ensemble autonome : un Writer stocke une List et en extrait des Items qu'il affiche. Le reste du monde, ces classes n'en ont rien à faire. Le code modifié est ici

Ensuite, tu vas avoir un scripts qui va initialiser les choses : import bge from dictionnary import dico # dictionary s'écrit avec un seul N ;-) import textwriter list = TextStorage() writer = TextWriter() dico['list_text'] = list writer.list = list bge.logic.getCurrentScene().post_draw.append(self.write)
À ce stade, la méthode post_draw() va bien, *entre autres* (donc sans écraser tout ce qu'elle aurait pu faire d'autre), appeler un TextWriter et lui demander d'afficher ses textes, qu'il ira chercher dans une TextList, qui est stockée dans le dico pour être accessible par le reste du code. Le TextWriter, lui, est caché : personne n'a besoin de savoir qu'il est là, personne ne peut interagir avec lui, il n'existe que parce que post_draw appelle une de ses méthodes.

Ce script est à placer à un endroit où il s'exécutera une seule fois.

Et pour finir, à chaque fois que tu veux ajouter quelque chose à ton affichage, ça se fait en une ligne : dico['list_text'].add("Hello, World !", 0.7, 0.42, [0.0, 1.0, 0.0, 1.0], "Arial")

La possibilité d'effacer un texte est laissée comme exercice pour le lecteur.

Et pour répondre à ta dernière question : on pourrait intégrer la classe TextItem à la classe TextStorage. Le résultat serait une classe à part entière, mais qui ne serait accessible et utilisable que depuis l'intérieur de TextStorage. Le TextStorage serait le seul à savoir que les TextItems existent.
Ça peut avoir du sens, mais dans ce cas précis non : le Writer utilise implicitement des TextItem, puisqu'il les extrait de son TextStorage. Ça *fonctionnerait*, mais ça ne serait pas très cohérent.