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 : 24/3/2018 0:12:42

Citation :
C'est c'est pas toujours évident de pouvoir séparer certaines choses parce que tu as des dépendances par-ci et par-là.
Des excuses, toujours des excuses

. Si des dépendances nuisent à la séparation des responsabilités, en général, la solution est d'éliminer la dépendance. Note que dans le code que j'ai modifié, TextWriter ne dépend plus de rien, le module est autonome. Les dépendances, c'est le mal, de toute façon.

Citation :
Je t'invite à écumer en diagonale la version 0.3.2 de mon projet, comme ça, tu pourra me dire avec un cas concret sous la main, là où j'ai mal agencés mes affaires.
L'échelle du projet s'y prête mal. Sur un cas particulier je veux bien, sur l'ensemble ça fait plus de travail que je ne suis prêt à en fournir.

Citation :
'Pas d'accord: cette "chose" permet l'affichage du texte. D'où vois-tu que c'est une variable ?
Je me suis mal exprimé, mais j'ai raison sur le fond. D'où je le vois ? De la syntaxe du code. Et si ça ne te suffit pas, de la documentation officielle de Python.
BGE est un module. Logic, ça peut être une classe, un objet, on s'en moque en vrai. getCurrentScene() est une méthode, qui renvoie une KX_Scene si mes souvenirs sont bons. "Instruction" est juste, mais c'est un terme trop vaste pour être utile ici (ça couvre aussi bien un appel de méthode qu'une assignation ou une comparaison). Mais jusque-là, tu as bien compris.

C'est l'étape suivante qui pose problème. Post_draw est une variable. Pour paraphraser la documentation, c'est une collection de callables, donc d'objets qu'on peut appeler, par exemple des méthodes. L'idée est qu'après avoir dessiné la scène, on va appeler tous les éléments de la collection. Post_draw est en gros "la liste des trucs qu'on fera après le rendu". Ce n'est pas une instruction, c'est une collection d'instructions. Et ton code remplace cette collection par une nouvelle (via "... = [write]"), ce qui revient à désinscrire tout ce qu'il pouvait y avoir avant. C'est un aimant à bug.

Ce n'est donc pas une action. C'est une assignation. Comme l'indique le "="...

Citation :
Parce que, dans mon projet, mon dictionnaire communique avec toutes les scènes
Ce n'est pas ce que j'ai demandé. Le dictionnaire, OK, il a sa place dans ton système dans l'ensemble, pas de problème de ce côté-là, ce n'est pas la seule façon de faire mais ça marche.

Ce que je conteste, c'est ce que cette information en particulier fait dans le dictionnaire. La liste des textes à afficher n'a rien à faire là, c'est une information qui ne concerne que le TextWriter. Cf mon exemple modifié : le TextStorage s'y trouve, mais il *encapsule* ces données (en gros, il les cache aux yeux du reste du code, toute interaction avec la liste de textes passe par lui).

L'encapsulation est un autre principe important : ne doit avoir accès à une information que celui qui en a vraiment besoin.

@ Hook
Citation :
C'est pas la peine, déjà de base cherche pas à comprendre ne met pas de condition dans la __init__ vu qu'elle ne sera exécuté QU'UNE SEULE FOIS (lors de la création d'un objet).
Là pour le coup tu rates un détail. On vérifie si l'objet auquel le script est rattaché a été initialisé, ça ne concerne pas l'objet qu'on est en train de construire. Cette méthode est exécutée à chaque fois qu'on crée un objet de la classe Text. Et on ne sait pas à quelle fréquence ça se passe. Si ça se trouve, il instancie tout le temps des Texts pour rien, la vérification a du sens.
En gros : l'architecture est bancale, donc le scotch un peu partout se justifie