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 : 25/3/2018 13:26:56

Citation :
Donc, tu veux dire que "init" n'est pas obligatoire quand on créer un classe ? Car moi, c'est ce que j'avais compris.
Ce n'est pas vraiment ce que je voulais dire dans ce cas précis, mais non, en effet, __init__ n'est pas obligatoire.
Comme son nom l'indique, c'est un initialiseur. C'est une méthode qui s'assure que l'état de l'objet est correctement initialisé, et a priori elle n'est rien censée faire de plus.
Par exemple, si j'écris :
class Foo: def add(self, element): data.append(element) foo = Foo() foo.add("foo")
Je vais me retrouver avec une erreur de type variable non déclarée (ou pointeur nul, dans les langages plus stricts).
Pour que mon objet Foo puisse ajouter des éléments à sa collection, il faut que la collection existe ; et dans le cas du Python, par défaut le champ n'existe même pas. Le rôle d'__init__, c'est de s'assurer que tout ce dont l'objet a besoin pour fonctionner est bien en place.

Citation :
Si je comprends bien, ça veux dire que je pourrai très bien appeler une fonction de cette classe, tout en étant a l'extérieur (de la classe) ? (ex: ma_classe.ajouter_un_texte(arg1, arg2, arg3))
Je serais tenté de répondre oui, mais avec le flou qui semble régner entre "classe" et "objet"...
Une classe déclare des méthodes, qui décrivent le comportement de ses instances. Quand tu as un objet d'une classe donnée, tu peux lui envoyer un message pour que la méthode correspondante s'exécute. Un exemple :
class Bar(): def __init__(self, data): self.data = data def output(): print(data) # Ailleurs dans le code, on n'est à présent plus dans la déclaration de la classe. bar1 = Bar("Hello,") bar2 = Bar("World !") bar1.output() bar2.output()
La console affichera "Hello, World !". Avec peut-être un retour à la ligne au milieu, je ne sais pas comment print() se comporte de ce côté-là.

Citation :
Tu as dis que le "post_draw" est un aimant à bug (ou est-ce l'ensemble ... = [write] ?), c'est bien juste ?
Non !
J'ai dit que le fait de *remplacer* la collection de méthodes à appeler après le rendu était un aimant à bugs. Cette variable est accessible partout, donc potentiellement utilisée partout ; tu ne sais pas qui s'est inscrit précédemment, tu ne sais pas qui a besoin qu'un truc en particulier se fasse après le rendu, et tu viens tout désinscrire.

Comme dans mon code d'exemple, remplace ça par ....post_draw.append(write) et tout ira bien.

Citation :
Dans la doc python de blender ? C'est impossible. Ou as-tu trouvé cette phrase qui parle d'une telle hérésie ? Comment peux-tu, dans un environnement interactif, exécuter le script une seule fois ?
À ce stade, la seule chose que je trouve à répondre, c'est "réfléchis". Tu noteras que dans l'Histoire, le mot "hérésie" a rarement été employé par ceux qui avaient raison...
Explique-moi le rapport entre l'interactivité et le fait d'exécuter un script en boucle.
Genre l'initialisation du monde. Tu recrées tout ton monde à chaque frame ?
Le spawn des ennemis. Tu respawn tes ennemis à chaque frame ?
L'interactivité t'oblige à avoir des boucles qui vérifient les entrées utilisateur. Le temps réel t'oblige à avoir des boucles pour le comportement des divers personnages.

Mais mettre en place le système d'affichage... Une fois qu'il y est, il ne bouge plus, non ? Pourquoi le refaire en boucle ?

Maintenant, si la question était réellement "comment", un sensor Always en *décochant* les deux boutons avec trois pointillés en bas à gauche du cadre. Premier résultat Google, soit dit en passant.

Citation :
Comment peux-tu en une seule fois, virer un texte en vert si l'on place le pointeur dessus et le virer en rouge quand le pointeur est ailleurs (donc on suppose qu'il est rouge de base) ?
Quand le pointeur arrive sur l'élément, on appelle un script qui modifie le texte à afficher. Quand il repart, on en appelle un autre. Le détail des senseurs et tout ça m'échappe, mais en fait, ça n'a rien à voir avec la question.

Si tu reprends le code que je t'ai proposé (ce serait sympa de l'étudier avant de poser des questions auxquelles j'ai déjà répondu), tu verras que le TextStorage est présent dans le dictionnaire. Donc, la liste des textes à afficher est bien accessible là où tu en as besoin, genre dans ton MouseOver. Par contre, TextStorage et TextWriter n'ont besoin d'être créés qu'une seule fois.

Citation :
Je pense que l'on à pas bien interpréter ce que j'ai dit, je laisse passer.
J'espère bien.

Citation :
Pourquoi suis-je obligé de séparer ça ? Point de vue éthique, on est d'accord, mais pourquoi créer juste une classe pour ça ? Pour le lecteur, peut-être ?

"Créer une classe juste pour ça"... Cette formulation laisse entendre que créer une classe est un effort surhumain, un coût à payer, qu'on cherche à économiser un maximum. C'est entièrement faux. En fait, cette phrase remporte le prix de l'hérésie de toute cette conversation

Une classe, c'est la brique de base de ton code, c'est ton principal outil de simplification du problème. La créer ne demande aucun effort, juste le fait d'écrire "class:" et éventuellement d'ouvrir un nouveau fichier.
Tu gagnes en lisibilité, en modularité, en tout.

Citation :
"super" se réfère à "SceneElement" ? Mais enfin, comment est-ce possible ? Si je place "class audioplayer" en premier, c'est lui "super" alors ?
Ça c'est les bases de la POO.
Super renvoie à l'objet courant (self donc) considéré comme une instance de sa superclasse. La superclasse étant ici SceneElement...

Citation :
Seul game doit faire apparaitre des créatures.
Euh ben c'est le cas ici, puisque les méthodes concernées sont déclarées par la classe Game...

Citation :
Donc, je pompe l'objet current_game (ou game, plutôt ?) du dico, pour pouvoir faire ça ?
L'objet current_game est une instance de la classe Game.
Quant à la gestion de l'inventaire... Le simple fait d'avoir une classe "game" qui fait quelque chose est un gros signal d'alarme. Un nom pareil, ça laisse entendre que ça va faire plein de trucs. Or on n'a pas le droit de faire plein de trucs. En fait, j'aurais dû la renommer Spawner plutôt que Game.
Donc non, la gestion d'inventaire, tu ne la mets pas dans Spawner.

Citation :
Il me semblait que c'était dangereux d'avoir beaucoup de scripts... Outre la facilité de lire ceux-ci, pourquoi faire ça ?
Pourquoi dangereux ?
Et la facilité de lecture ne te paraît pas un argument suffisant ? Te balader dans un fichier de 4 000 lignes, tu trouves ça fun ?

Citation :
- Pas de init sauf si variable il y a.
Ou si l'objet a besoin pour fonctionner de faire "quelque chose". L'idée d'__init__, c'est qu'une fois qu'elle est exécutée, tu puisses dire "ça y est, j'ai un [truc]". Il ne fait encore rien puisqu'on ne lui a rien demandé, mais il est là, il est complet. Tout ce qui n'est pas nécessaire à l'objet pour être complet, pour (i]exister[/i], n'a pas sa place dans __init__.

Citation :
- classe est un objet, variable = objet, donc variable est un "label", un surnom donné à la classe.
Quoi ? **Non !**
Je n'aurais pas dû mentionner qu'une classe est un objet, ça t'a emmêlé les pinceaux. Une classe est avant tout un plan, un modèle, qui décrit des objets. Elle décrit les données qu'ils contiennent, et le travail qu'il sont capables de faire.

Prenons la classe Personne. Cette classe déclare un champ nom, un champ âge, et un champ organes qui est une collection. Ça revient à dire "une Personne a un nom, un âge et des organes" ; c'est l'état. La classe déclare aussi les méthodes parler(), manger() et marcher(). Ce qui revient à dire "une Personne peut parler, manger et marcher" ; c'est le comportement.
Je suis une Personne. Je suis un objet, une instance de la classe Personne. Comme toute Personne, j'ai un nom, "DaWaaaaghBabal", un âge, 28 ans, et tout un tas d'organes en

parfait état. Je sais parler, manger et marcher, comme toute Personne.
La dernière caractéristique d'un objet, c'est son identité : si tu croisais un autre DaWaaaaghBabal de 28 ans me ressemblant comme deux gouttes d'eau, ce ne serait quand même pas moi, seulement mon clone.
Tu es une Personne. Tu es un objet, une instance de la classe Personne. Étant une Personne, tu as un nom, "Redstar", un âge inconnu, et tout un tas d'organes que j'espère en bon état. Tu sais aussi parler, manger et marcher.
Étant deux Personnes, on nous décrit de la même façon. On sait qu'on peut nous demander la même chose : parler, manger, marcher, et qu'on le fera de la même façon.
Nous sommes deux objets. "La dernière Personne à avoir posté un message" est une variable. Ça va être tantôt toi, tantôt moi, mais dans un cas comme dans l'autre ça ne change rien à notre identité ou à notre état. Cette variable *référence* un de nous deux.

Et j'oubliais : à la naissance, on appelle __init__(nom). __init__ fait trois choses : elle met l'âge à 0, elle assigne au champ nom la valeur choisie par les parents, et elle crée tout un tas d'organes qu'elle stocke dans la collection organes. Ouais, j'avais de mauvaises notes en biologie.

Honnêtement, si tu as déjà écrit plusieurs milliers de lignes de code en comprenant aussi mal les concepts les plus fondamentaux, tu peux te préparer psychologiquement à réécrire ton programme de zéro. Ce sera plus rapide qu'essayer de corriger ce que tu as déjà.
En tout cas, la prochaine étape pour avancer dans ton projet, ça va être de le mettre en pause (du moins l'aspect prog) pour écrire des programmes beaucoup plus simples mais en te concentrant sur les concepts fondamentaux.