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 : 26/3/2018 15:03:37

Citation :
En fait, ce que je déteste pas dessus tout, c'est que je soit obligé de rédiger d'une façon arbitraire sans pouvoir comprendre.

C'est la raison pour laquelle je dis "pourquoi ?", "pourquoi faire ça ?" et "quel intérêt ?".

Je ne peux pas me contenter de dire "ok, tu as raison, je vais faire comme cela". Il faut que je puisse comprendre le programmeur pour bien programmer à mon tour.
Dans ce cas, je peux détailler sur ce morceau en particulier. Note préliminaire : j'ai tâché de respecter la logique de ce que tu avais fait, en partant du principe qu'il y avait des raisons. Ce n'est probablement pas ce que j'aurais fait en partant de zéro.
1/ Tu as décidé de factoriser l'accès aux "variables d'environnement" du script. Note : ce n'est pas une bonne idée, du moins pas de cette façon, mais je détaillerai pourquoi ailleurs. J'ai donc écrit une classe SceneElement qui fait ça.
2/ Tu as décidé de factoriser la lecture de sons. J'ai donc écrit une classe AudioPlayer qui fait ça.
3/ Tu as décidé que ces deux responsabilités, son et variables d'environnement, étaient liés. Personnellement je ne suis pas d'accord (ça n'a juste rien à voir), mais comme le but était d'apporter une petite correction et pas de tout refaire, j'ai gardé ce lien.
4/ Pour faire le lien entre les deux classes A et B, il y a en gros deux façons : A "possède une instance de" B, ou A "est un type particulier de" B. On a donc quatre choix :
- AudioPlayer possède une instance de SceneElement. C'est le plus propre, mais ça nuit un peu au but recherché qui était de simplifier l'écriture.
- SceneElement possède une instance d'AudioPlayer. Non. La plupart des SceneElement n'ont pas besoin de son.
- SceneElement est un type particulier d'AudioPlayer. Encore moins, pour le coup, ça n'a aucun sens.
- AudioPlayer est un type particulier de SceneElement. Admettons. Moins bon que le premier choix, mais plus proche de ce que tu avais écrit.

Donc AudioPlayer hérite de SceneElement.

Citation :
C'est à dire ? Pas assez de headers de couleurs ?
Les couleurs on s'en fout. Le minimum syndical pour coder confortablement, c'est 1/ un explorateur de fichiers, 2/ pouvoir ouvrir plusieurs fichiers dans des onglets, 3/ pouvoir ouvrir une définition à partir d'une référence (si un code mentionne une classe / fonction / constante, je peux ouvrir le fichier où elle est déclarée). Ensuite, idéalement, on veut bien de l'autocomplétion, de la correction syntaxique, des outils de refactoring...
Et Blender n'a rien de tout ça.

Citation :
Ça dépends: il m'arrive de faire des erreurs de frappe et je perds du temps des fois à chercher après l'erreur. Donc j'ai besoin de tester progressivement ce que je fais pour m'assurer du fonctionnement recherché.
Ma question était en fait rhétorique. Tu travailles sur une ligne de code à la fois, pas dans tout le code. Tu travailles sur une fonctionnalité à la fois, pas sur tout le système.
Et afficher tout le code à la fois est un handicap quand tu cherches une faute de frappe récente. Quand tu as fragmenté ton script en 40 fichiers, tu sais que ta faute de frappe n'est pas dans un des 39 qui sont actuellement fermés. Ce qui réduit le volume de fichier à relire de 97,5%...

Citation :
Fragmenter le code "à l'excès", soit avoir 40 scripts, par exemple, me semble contre-productifs: Si je dois commencer à voyager d'un script à l'autre, pour vérifier que l'erreur n'est pas là et bien ailleurs, en sachant qu'après 5 essais je commence à faire tourner mon moulin à jurons... Ça deviendrai invivable.
Euh, rien de tout ça n'a de sens.
Quand tout est bien divisé, tu sais où est l'erreur. C'est une des raisons pour lesquelles on divise...

Note bien un truc : en Java, on considère qu'il y a un problème de design quand une classe dépasse 100 lignes. C'est une limite un peu arbitraire et ça dépend des équipes et des entreprises, certaines vont te dire 50, d'autres tolèreront jusqu'à 200, mais dans l'ensemble, tout le monde considère que 100 lignes est une bonne taille. Et en Java, on ne peut déclarer qu'une classe par fichier. Donc, plus ou moins tout le monde s'entend pour dire que 100 lignes par fichier est la taille à ne pas dépasser. Et en 100 lignes de Java, on fait moins qu'en 100 lignes de Python.

La division en plein de fichiers permet de simplifier la navigation en limitant la quantité de code à fouiller et en exploitant l'arborescence de fichiers pour trier les informations. Elle permet de limiter les bugs, puisque ce qui est dans un fichier ne peut pas affecter ce qui est dans un autre sauf si c'est explicitement mentionné ("import"). Elle permet de les identifier facilement : si l'affichage des textes se passe mal, tu vas dans le(s) fichier(s) qui gèrent l'affichage des textes, il y a peu de chances que le bug viennent de la gestion des sons. Sauf si la gestion des sons contient la ligne "...post_draw = [play]"

Citation :
Pour BGL, je devais poser une question sur la résolution et le positionnement du texte mais je suppose que ça attendra ?
Ben en fait, c'est surtout que je n'aurai pas de réponse. Autant la programmation en général c'est mon métier, autant le BGE en particulier je n'y connais rien. Note : ne prends pas ça comme une excuse pour m'ignorer, les principes dont je parle sont à peu près universels

Citation :
Dois-je en déduire que je dois te donner un compte-rendu sous forme d'une partie de mon code reconditionné ?

Je ne vais pas te donner des devoirs, hein. Si je disais ça, c'est parce que je m'étais laissé emporter et que j'avais encore une page d'explications qui n'étaient pas en rapport avec le problème immédiat.

Citation :
Après mes essais dans mon coins, je peux commencer éventuellement par le script des IA des mes animaux, il est assez petit. Et te partager mes modifications.
Oh, partage tes essais dans ton coin aussi

Et on trouve sur Google des exercices de POO, ça peut être un meilleur début que de modifier tes scripts (un exercice conçu comme tel est généralement plus clair qu'un cas concret).