Forum: Moteur de jeu GameBlender et alternatives
Topic: DarkRise
Subject: Re: DarkRise
PostÃ© par: Bobibou
Contribution le : 4/11/2010 21:21:57

Tu veux un script Python pour quoi faire ?
Si c'est pour les touches, Edddy t'a donné la base en briques logiques dans la deuxième image de son dernier post.
[edit] Oui, ça doit être ça.
- Bonne lecture.

[/edit]
Pour ce qui s'agit du script après, c'est pas le plus évident pour commencer, mais tu devrais y arriver quand même.

En gros, il faut récupérer à partir du sensor quelles touches ont été appuyées et activer en conséquence les actuators.
L'avantage, c'est que les touches peuvent être très facilement changées et même pendant le jeu si tu veux, en enregistrant dans des variables les touches de déplacement (par exemple). Enfin pour commencer, je te donne quelques indices :
Pour gérer les touches, il faut le module GameKeys donc "import GameKeys (as gk)" (sans les parenthèses, c'est juste que tu le mets où tu le mets pas, le "as gk", mais perso, je trouve ça plus simple d'écrire ensuite gk que GameKeys à chaque fois).
Après, il faut commencer par récupérer le controller (GameLogic.getCurrentController()) afin de récupérer tout ce qui est attaché.
Appelons le "cont" (au hasard
).
A partir de là, on va chercher le sensor keyboard, comme ceci :
keyboard = cont.sensors["nom_du_sensor_keyboard"]
Ça va jusque là ? C'est pas trop dur. Le principe, c'est qu'il faut toujours passer par le controller qui a lancé le script. C'est un des deux seuls lien, avec getCurrentScene() avec les objets et les briques. De cette façon, cont.sensors contient tous les sensors rattachés au controller, cont.actuators les actuators, cont.owner l'objet possesseur du controller...

A partir de ce sensor keyboard, on va extraire la liste des touches qui ont agit :
keylist = keyboard.events

keylist est, comme son nom l'indique, une liste. Elle est donc susceptible de contenir plusieurs éléments (si plusieurs touches ont été enfoncées par exemple). Pour traiter chaque touche, on va devoir parcourir la liste élément par élément. Pour cela, il existe ce que l'on appelle des boucles. En python, la boucle qui permet de parcourir une liste est la boucle **for** ... **in** : (ne pas oublier les deux points qui annoncent un bloc de texte indenté.
Ainsi, "for element in liste :" va exécuter la portion de code plusieurs fois en attribuant à la variable "element" successivement les différentes valeurs contenues dans liste. Donc si liste = ['a', 2, 1.0025], le code sera exécuté 3 fois : une avec element = 'a', une

avec element = 2 et une dernière avec element = 1.0022
Revenons à notre cas : admettons que keylist = [touche1, touche2]. On veut donc qu'une certaine variable (appelons là "key") prenne la valeur de touche1 puis la valeur de touche2, exécutant deux fois le même code.
On obtient donc ceci :
for key in keylist :
Ensuite, pour indiquer quelle partie du code il faudra répéter, il faut l'indenter, c'est-à-dire l'espacer un peu de la marge. Pour cela, tu peux utiliser ce que tu veux : un espace, deux espaces, quatre espaces, une tabulation, vingt tabulations...
Donc à partir de là, tout le traitement de la touche, celui qui sera effectué plusieurs fois, devra être indenté de façon régulière (toujours deux espaces par exemple, il ne faut pas changer en cours de route !).

En fait, les éléments de keylist sont eux-même des listes. Mais là, pas besoin de boucle car on connait la taille de ces listes : 2. Cela signifie qu'il existe deux éléments dans cette liste : key[0] et key[1]. Oui, le dernier élément est bien le 1 lorsqu'il y en a 2 car le zéro compte (il ne faut surtout pas l'oublier !).
key[0] correspond à la touche et key[1] à son statut.
Les statut possibles sont stockés dans les variables suivantes :
GameLogic.KX_INPUT_ACTIVE -> La touche est enfoncée
GameLogic.KX_INPUT_JUST_ACTIVATED -> La touche commence tout juste à être enfoncée
GameLogic.KX_INPUT_JUST_RELEASED -> La touche vient d'être relachée
GameLogic.KX_INPUT_NONE -> La touche n'est pas enfoncée
Dans le cas d'un mouvement, on vérifie si le statut est bien GameLogic.KX_INPUT_ACTIVE grâce à une condition.
Une condition commence par **if** et termine par deux points. Ces deux points signifie toujours que le code qui suit va être indenté d'un rang de plus qu'avant. Il était déjà indenté d'un rang pour la boucle, il sera donc indenté de deux rangs. Les différents rangs ne sont pas obligatoirement de même taille, mais c'est plus lisible d'être régulier quand même.

Donc on n'en arrive à là :
for key in keylist : if key[1] == GameLogic.KX_INPUT_ACTIVE : # Le code à n'exécuter que si la condition est vérifiée (quatre espaces dans mon cas) # (après le symbole "#", le code n'est plus pris en compte, ce qui permet de faire des commentaires plus lisibles que le code)
Les deux signes '=' ne sont pas une erreur. En effet, en programmation on fait la distinction entre le = d'assignation et le == de test logique.
Maintenant, si c'est bien le cas, on regarde quelle touche est enfoncée dans key[0].
Les valeurs possibles sont listées sur cette page.
Devant chaque constante (à toujours mettre en majuscules), il faut mettre GameKeys. ou gk. (selon la première ligne). Pour ne pas avoir à l'écrire, il aurait fallu, en première ligne, écrire "from GameKeys import *", ce qui n'est d'ailleurs pas une mauvaise idée.

Donc là encore on va devoir utiliser une condition en comparant grâce au '==' key[0] aux différentes touches attendues. Mettons qu'on ne puisse aller, comme dans ton jeu, qu'à droite (D) et à gauche (Q). On aura donc deux tests à faire :
for key in keylist : if key[1] == GameLogic.KX_INPUT_ACTIVE : if key[0] == GameKeys.DKEY : # Aller à droite elif key[0] == GameKeys.QKEY : # Aller à gauche

Tu remarqueras peut-être le "elif". C'est une contraction de else (sinon) et if. Cela veut donc dire "sinon, si" En effet, on est sûr que si la touche est D, ça ne sert à rien de regarder si elle est Q ensuite puisqu'elle ne peut pas être les deux à la fois. Si les deux sont appuyées en même temps, key[0] n'aura pas les deux valeurs, mais je rappelle que ce code est exécuté une fois par touche.

Donc ça permet d'éviter de faire trop réfléchir l'ordinateur, ce qui n'est jamais bon.

Ne reste plus qu'à voir comment aller à gauche et à droite, c'est-à-dire comment activer un actuator.
Pour cela, ce n'est pas bien compliqué : on utilise le controller (comme toujours) et on indique le nom de l'actuator, comme ceci :
cont.activate("actuator_pour_aller_a_droite"]
et pour le desactiver (on sais jamais
)
cont.deactivate("actuator_pour_aller_a_droite"]

Bon, je te laisse recoller les morceaux et voir ce que ça donne. En cas de problème, j'expliquerais volontiers les points sombres de mon histoire.