PREPA SIMULATOR

Cheat Sheet : Écriture de scripts

Motivation :

Jusqu'à la v0.2.0, le jeu était relativement linéaire : le joueur pouvait certes se déplacer sur une carte et parler aux PNJ, mais ces PNJ avaient des dialogues uniques et disaient donc uniquement ce qu'on leur indique de parler (ie. le texte inclus dans la BDD). L'implémentation du Sac a certes permis un nouveau degré de liberté pour le joueur, mais un nouveau moyen plus poussé d'interagir avec les PNJ était nécessaire.
L'implémentation du Scripting ~~rend à présent le jeu logarithmique~~ autorise une plus grande liberté dans les interactions possibles.

Structure du ScriptManager :
- un accumulateur booléen
- un accumulateur numérique
- un accumulateur liste, utilisé pour les infobox

Comment écrire son propre script ?
Il faut d'abord créer une nouvelle entrée dans la BDD, section « scripts ».
Ensuite, définir une nouvelle fonction du même nom dans le fichier `scripts.py`.
La fonction doit retourner une liste composée des instructions sous forme de chaîne de caractères. Préférer les doubles guillemets triples. Et surtout : NE JAMAIS UTILISER LES SIMPLES GUILLEMETS TRIPLES. ~~C'est moche.~~

Liste des fonctions et exemples :
※ *Fonctions générales*
- `runscript(script)`
Exécute le script dont le nom est donné en argument sous forme de chaîne de caractères.

- `interrupt()`
Met fin à l'exécution du script courant.

- `label(label_tag)`
Indique la présence d'un label, dont le nom est `label_tag`, à partir duquel reprendre l'exécution. Ne fait absolument rien sauf si la commande `goto` est présente, accompagnée du nom du label.
Préférer les noms de labels utilisant des caractères alphanumériques uniquement.

- `goto(label_tag)`
Reprend l'exécution à partir du label dont le nom est spécifié en argument (bien sûr en tant que chaîne de caractères).

※ *Fonctions booléennes*

- `compare_obj_qty(obj_id, operator, qty)`

Compare la quantité actuelle de l'objet ayant pour ID `obj_id` à la quantité `qty`.
Le résultat de la comparaison, stocké dans l'accumulateur booléen, dépend de l'argument `operator` :
`sup` >=
`inf` <=
`eq` =

- `iftrue(command)`, `iffalse(command)`

Exécute la commande `command` si l'accumulateur est `True` (resp. `False`).
`command` doit être une chaîne de caractères. Préférer les doubles guillemets simples.

Ex :

```
compare_obj_qty(0, sup, 50)
      iftrue(…)
```

fait quelque chose si le joueur dispose de plus de 50 exemplaires de l'objet d'ID 0.

※ *Fonctions numériques*

- `ran(inf, sup)`

Génère un nombre entier aléatoire dans l'intervalle [`inf`, `sup`]. Le résultat est stocké dans l'accumulateur numérique.

- `compare(operator, qty)`

Comme `compare_obj_qty`, mais avec des nombres.

Ex :

```
ran(1, 6)
compare('eq', 1)
      iftrue(...)
```

simule le lancer d'un dé et fait quelque chose si le résultat est égal à 1.

- `put(value)`

Place une valeur dans l'accumulateur, indépendante de sa valeur précédente.

- `math(operator, value)`

Effectue l'opération souhaitée avec l'accumulateur, et y place le résultat. `operator` est un caractère, les opérateurs supportés pour le moment sont `+ - * /`
Par exemple, `math('*', 10)` avec une valeur stockée de 15 place dans l'accumulateur la valeur 150.
En cas de division par 0, une valeur arbitrairement grande (sans souci de signe) est mise par défaut.

※ *Fonctions sonores*

- `chg_music(track)`

Change la musique courante en celle spécifiée en argument.

- `sfx(fx)`

Joue l'effet sonore mis en argument.

※ *Fonctions graphiques*

- `changelayer(layer)`

Change le calque d'affichage du joueur. Les arguments possibles sont `'bg'` pour l'arrière-plan et `'fg'` pour le premier plan.

※ *Fonctions des flags des PNJ*

- `checknpcflag(npc, flag_id)`

Place la valeur du flag `flag_id` du PNJ spécifié en argument, dans l'accumulateur booléen.

- `setnpcflag(npc, flag_id)`

Modifie la valeur du flag `flag_id` du PNJ spécifié en argument.

※ *Fonctions des objets*

- `get_object(obj_id, qty)`

Le joueur obtient l'objet d'ID `obj_id` en quantité souhaitée.

- `toss_object(obj_id, qty)`

Le joueur perd l'objet d'ID `obj_id` en quantité souhaitée. Attention, la commande n'a pas d'effet si le joueur dispose de moins d'objets. Pour retirer tous les objets d'un type indifféremment de leur quantité, utiliser l'argument `'all'` (pratique pour les objets rares).
Utiliser ce script librement lors de dialogues avec Lentsch.

※ *Fonctions des boîtes de dialogue*

- `dialogue(talking, dialogue_id)`

Engage avec le PNJ `talking` son dialogue associé d'ID `dialogue_id`.
ATTENTION, `talking` est un objet de type `Npc` ! Si le dialogue est engagé avec le PNJ auquel le joueur parle, utiliser `self.current_npc` à la place.

- `loadtext(text_id)`

Charge dans l'accumulateur associé :

- une ligne du texte de l'infobox désignée par son identifiant dans le dictionnaire des textes d'infobox si l'argument est un entier
- la ligne de texte passée en argument si c'est une chaîne de caractères

- `infobox()`

Génère une infobox en utilisant tout le texte présent dans son accumulateur, puis détruit le contenu de ce dernier.

※ *Fonctions des drapeaux*

- `testflag(map_id, flag_id)`

Obtient la valeur du drapeau `flag_id` de la map donnée en argument. La valeur d'un drapeau est un entier (0 ou 1), mais le résultat de cette fonction est un booléen placé dans l'accumulateur booléen.

- `raiseflag(map_id, flag_id)`

Lève le drapeau `flag_id` de la map donnée en argument. Cela ne provoque pas d'erreur si le drapeau correspondant est déjà levé.

- `lowerflag(map_id, flag_id)`

Idem, mais le drapeau correspondant est baissé.

Ex :

```
testflag(12101,0)
      iftrue('lowerflag(12101,0)')
      iftrue(''')loadtext('Le flag 0 de la L101 va être baissé')''')
      iffalse('raiseflag(12101,0)')
      iffalse(''')loadtext('Le flag 0 de la L101 va être levé')''')
infobox()
```

alterne le flag 0 de la L101 et affiche un message en adéquation.

※ *Fonctions de mouvement*

- `move(direction, pix, sprint = False)`

Engage un mouvement de direction `direction` et de longueur `pix`, exprimée en pixels. L'argument `sprint` détermine si le joueur est en train de sprinter.
Les directions supportées sont :

```
up
right
down
left
```

Le mouvement est interrompu lorsque le joueur heurte un mur ou utilise un warp. ~~Après si vous voulez qu'il continue indéfiniment à marcher, il suffit de demander ;)~~

À titre d'exemple final, voici un script entièrement commenté utilisé à l'origine sur le PC de la H009 (les numéros de ligne servent juste à la lisibilité du script ici, ils ne sont pas nécessaires #cobol) :

```
1     testflag(-1, 0)
            iftrue('loadtext(’’’Le câble HDMI est branché à l’ordinateur.’’’)')
            iftrue('infobox()')
            iftrue('interrupt()')
5     loadtext(’’’Il y a de la place pour un câble HDMI derrière la tour.’’’)
      infobox()
      compare_obj_qty(10, 'sup', 1)
            iffalse('interrupt()')
      loadtext(’’’Vous branchez le câble HDMI à l’ordinateur.’’’)
10    loadtext(’’’La souris marche maintenant depuis l’autre côté de la salle !’’’)
      infobox()
      toss_object(10, 'all')
      raiseflag(-1, 0)
```

Ligne 1 : on regarde l'état du flag 0 de la salle actuelle : la H009
Lignes 2 à 4 : si le flag est levé alors le câble a déjà été branché, il n'y a rien à faire, on quitte le script.
Sinon, l'exécution continue à partir de la ligne 5.
Lignes 5 à 6 : infobox de notification
Lignes 7 à 8 : on regarde si le joueur possède le câble HDMI dans son Sac. Si non, l'événement n'a pas lieu d'être, le script s'arrête.
Si le joueur est en possession d'un câble HDMI, le script continue à partir de la ligne 9.
Lignes 9 à 11 : infobox de notification
Ligne 12 : le câble HDMI disparaît de l'inventaire du joueur.
Ligne 13 : le flag 0 de la salle actuelle (H009) est levé pour préparer la prochaine exécution de ce script.