Traduction des jeux de Ren'Py : Un guide, par les moskys

moskys#0576

¡Hola !

Appelez-moi moskys. Comme vous le savez sûrement, si vous êtes venu à ce guide, Ren'Py est un moteur de jeu libre et gratuit (utilisé pour la création de romans visuels, en général) qui est devenu l'un des logiciels les plus populaires utilisés par les créateurs amateurs pour développer leurs projets de jeux. Basé sur le langage de programmation Python, sa simplicité et sa flexibilité sont un grand avantage, car il permet d'obtenir des résultats plus qu'acceptables sans connaissances préalables en programmation. Et l'une des nombreuses fonctions que Ren'Py inclut est de faciliter la traduction des jeux créés avec ce programme. Ainsi, après avoir joué pendant près de trois ans et traduit manuellement en espagnol (et pour PC) plusieurs jeux de complexité diverse, je me suis encouragé à écrire ce guide pour expliquer le processus à tous ceux qui veulent commencer à traduire ou qui sont assez curieux pour savoir à quoi je consacre mon tempslibre.

Traduit avec www.DeepL.com/Translator (version gratuite)

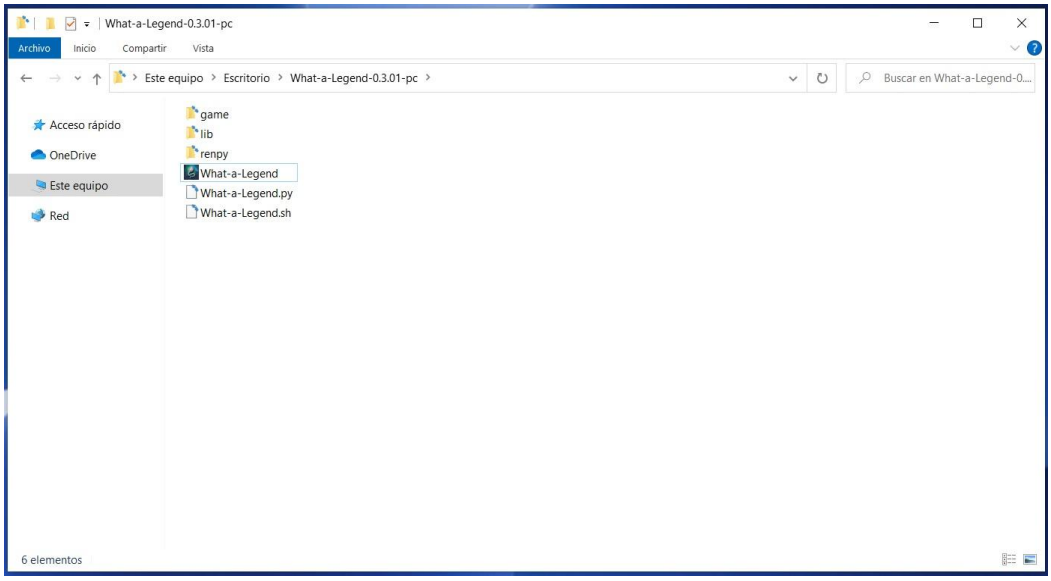
OUTILS ESSENTIELS (ET GRATUITS) - versions au 31-déc-2020

- **Ren'Py SDK 7.3.5. ->** https://www.renpy.org/latest.html
- **UnRen v.09.dev ->** https://f95zone.to/threads/unren-bat-v0-8-rpa-extractor-rpyc-decompiler-console-developer-menu-enabler.3083/ *(des liens vers un forum à contenu pour adultes)*
- **Editeur de texte:**
  - **Notepad++ 7.9.1 ->** https://notepad-plus-plus.org/downloads/
  - **Atom ->** https://atom.io/

1.- INTRODUCTION

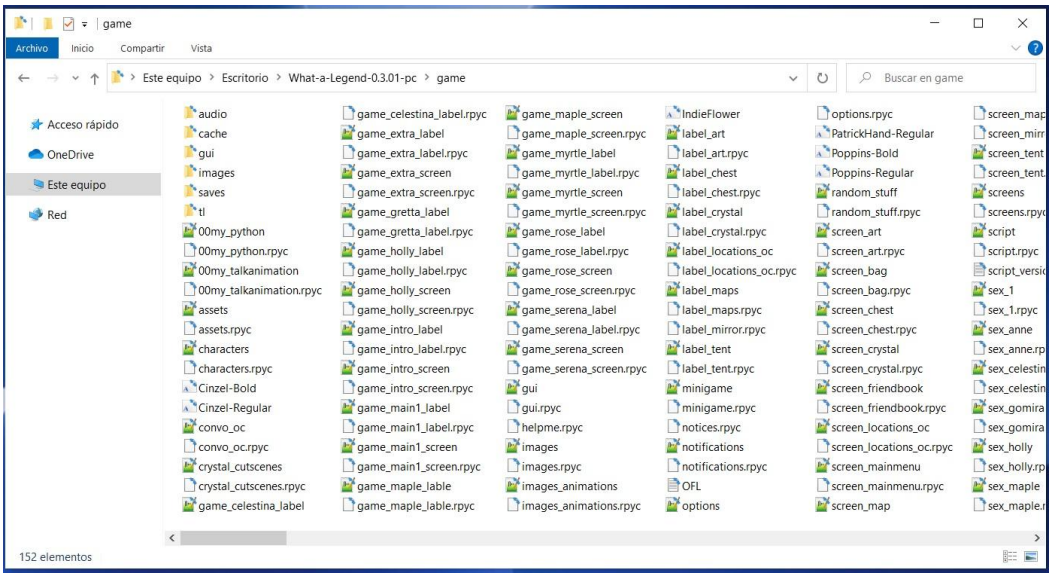
1.1.- Under the hood of a Ren'Py game

Il est évident, mais pour traduire un jeu, il faut d'abord avoir un jeu. Cherchez donc tout jeu Ren'Py que vous avez téléchargé et jetez un coup d'œil à son dossier racine. Vous verrez quelque chose comme ceci :



Habituellement, nous nous contentons de lancer le fichier exécutable et de jouer. Mais regardons maintenant les sous-dossiers de l'arbre que nous y voyons, et c'est là que la magie opère. En gros, dans le dossier "renpy", nous pouvons trouver les fichiers contenant les fonctions préprogrammées, et dans le dossier "lib", nous avons tous les fichiers techniques qui font fonctionner les jeux sur les différents systèmes d'exploitation. Ces deux dossiers nous permettent de jouer au jeu sans télécharger aucun autre programme spécifique pour celui-ci ; on pourrait dire que ces dossiers transforment les jeux Ren'Py en programmes auto-exécutants.

Le jeu lui-même se trouve dans le dossier "game". Pas de panique, celui-ci est particulièrement complexe :



Il y a plusieurs sous-dossiers que nous pouvons négliger pour l'instant pour nous concentrer sur les dossiers individuels. Ren'Py recommande aux créateurs de laisser tous les fichiers à la vue de tous, comme le font les développeurs de "What a Legend". Cela nous permet de voir toute une série de fichiers apparemment clonés, certains avec une extension .rpy et d'autres avec une extension .rpyc.

|Top|

1.2 - Trois concepts de base et deux commandements

Les fichiers dans le dossier "jeu" sont les scripts. C'est dans ces scripts que les développeurs incluent tout le code de programmation et les textes du jeu. Pour ce faire, il leur suffit d'écrire ce qu'ils ont besoin d'écrire dans un éditeur de texte et de sauvegarder le fichier avec une extension spécifique à Ren'Py appelée .rpy.

Ainsi, en utilisant un programme de base comme le Bloc-notes de Windows (ou, de préférence, un meilleur programme, comme Atom ou Notepad++, pour n'en nommer que deux gratuits et simples utilisés en programmation), nous pouvons ouvrir ces fichiers .rpy et voir ce que signifie la création d'un jeu Ren'Py. En prenant l'exemple.de."Quellelégende!",si.nous.ouvrons.le.fichier.nommé."convo_oc.rpy",.nous.pouvons.le.voir.dans.ses.premières.lignes:

```
1  # ==================== OLD Capital Base Conversations ==========================================
2  # Being stopped at the gate of the old capital ===============================================
3  label convo_gate:
4      call hide_ui from _call_hide_ui_104
5      call silent from _call_silent_42
6
7      scene scene_oc_bridge_gate_talk
8      if current_hour == "Night" or current_hour == "Evening":
9          show kevin at g_cright:
10             xalign 0.6
11         show pov at m_left
12         with quickfade
13         show pov ewide bup mdislike hfshock hbshock
14         show kevin hbstop eangry
15         kevin "STOP!" with vpunch
16         show kevin hbpoint edoubt
17         show pov -mdislike hfneu1 hbneu1
18         kevin "Did you manage to get a passage permit?"
19         show pov mno bdoubt eneub hfhead
20         show kevin eneu hbneu
21         pov "Umm..."
22         show pov eneu bsad msad
23         show kevin esad hfspearmove
24         kevin "I'm sorry, buddy..."
25         show pov mcry eflat bneu hfneu1
26         kevin "...but no permit, no passage. That is the law."
```

Tout ce qui se trouve à droite d'un symbole # est un commentaire qui n'apparaîtra pas dans le jeu, mais que les créateurs utilisent pour se guider dans leur code. Nous verrons plus tard à quel point ce symbole peut être utile aux traducteurs.

À la ligne 3, nous voyons un mot, l'étiquette, qui va être important. Les étiquettes sont des parties du scénario. La taille de ces portions dépend du développeur du jeu, mais elles correspondent généralement à une scène spécifique. Dans ce cas, ce label correspond à une conversation qui apparaîtra à l'écran lorsque les joueurs feront une action qui l'activera. En raison du langage Python, tout ce qui se passe dans cette scène est codé avec une indentation.

> Premier commandement de Ren'Py : vous respecterez toujours les indentations et ne les marquerez JAMAIS avec Tab, mais avec la barre d'espacement (4 espaces, en général). Si Ren'Py détecte une tabulation ou une indentation incorrecte quelque part dans les scripts (y compris les fichiers de traduction), le jeu ne démarrera pas.

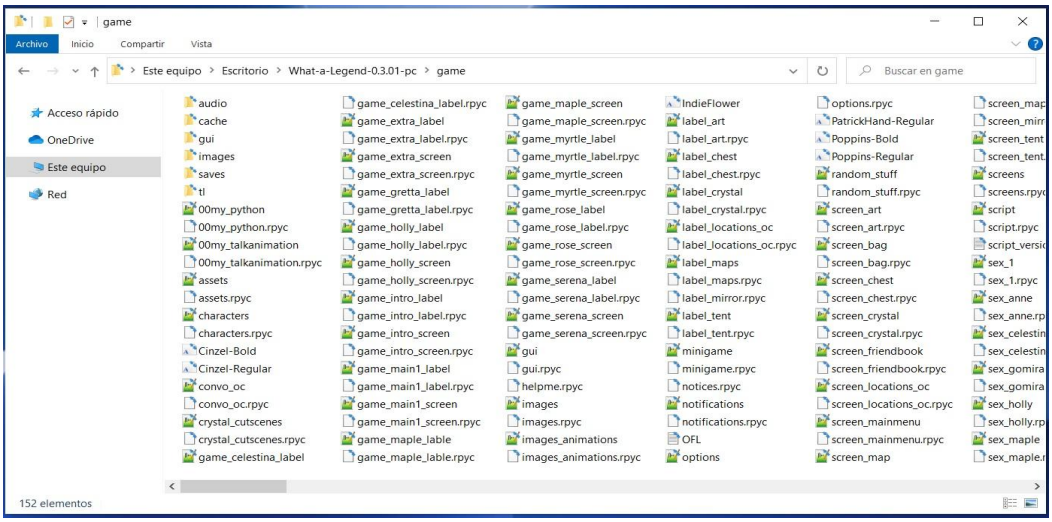
Si nous faisons défiler le script, nous verrons dans ce même label "convo_gate" plusieurs fonctions pour afficher et cacher des images et pour déterminer quelle partie de la conversation sera affichée en fonction des variables du jeu déclenchées par les joueurs pendant leur partie. Nous pouvons également voir quelques phrases citées : ce sont les lignes de dialogue qui apparaîtront à l'écran. Ces phrases citées sont celles que nous devrons traduire, et elles sont appelées des chaînes de caractères, ou des chaînes de texte.

> Le deuxième commandement de Ren'Py : vous devez toujours fermer les citations. Les guillemets non fermés (ou non ouverts, pour ce qui vaut) empêcheront le jeu de commencer.

|Top|

1.3 - Comment (je pense) fonctionne Ren'Py ? Le troisième commandement

En supposant que je ne sois pas un expert en informatique et qu'il est très probable que je dise quelque chose de stupide, je vais essayer d'expliquer très rapidement ce qui se passe lorsque nous lançons un jeu Ren'Py.



Vous souvenez-vous de la liste des fichiers clonés dans le dossier "jeu" ? Eh bien, ce ne sont pas exactement des clones. Ren'Py gère les fichiers .rpyc, qui sont une compilation des fichiers modifiables enregistrés avec une extension .rpy. Cette compilation se produit pour la première fois lorsque les développeurs ouvrent le jeu dans leur environnement de développement. Un AST (arbre syntaxique abstrait) est généré sur la base du contenu disponible à ce moment et cet AST sera la base de tout ce qui sera ajouté ultérieurement lors des compilations successives. En suivant ses algorithmes, Ren'Py attribuera une identification à chaque élément de programmation des fichiers .rpy en fonction de sa position dans le script et de sa propre nature (texte, variable, fonction, etc.), et avec ces identifications, il créera des fichiers .rpyc avec le même nom qui seront lus pendant le jeu.

Chaque fois que le jeu est lancé, Ren'Py recherche les fichiers .rpy et, le cas échéant, les compare avec leurs .rpyc respectifs, en recherchant les identifications créées lors de la compilation initiale. Si des modifications ont été apportées au fichier .rpy depuis la dernière compilation du fichier .rpyc, Ren'Py détectera à la fois les éléments inchangés et ceux qui ont été déplacés, il respectera leurs identifications précédentes et mettra à jour le fichier .rpyc avec les modifications. En simplifiant beaucoup, imaginons que la ligne numéro 3 du fichier .rpy soit initialement compilée de manière à ce que cette ligne soit identifiée par un "3" dans le fichier .rpyc. Si, dans les versions ultérieures du fichier .rpy, cette ligne n'est pas modifiée mais devient la ligne numéro 7, elle restera identifiée par un "3" dans le fichier .rpyc. Ainsi, même si le fichier .rpy final ne peut guère ressembler au premier créé par le développeur du jeu, son fichier .rpyc correspondant sera toujours compilé en suivant les relations créées dans le premier AST d'origine, car certains de ces éléments initiaux sont toujours là.

Si nous les supprimons à un moment donné, Ren'Py générera de nouveaux fichiers .rpyc mais en utilisant la version actuelle des scripts .rpy. Cela signifie que les relations et les identifications qui seront désormais créées dans l'AST ne seront pas les mêmes que dans les fichiers originaux, car elles sont créées à partir d'un script .rpy qui ressemble au premier. Si l'on reprend l'exemple absurde d'avant, la ligne 7 du fichier .rpy qui était appelée "3" dans le .rpyc supprimé parce qu'il était repris de versions précédentes, serait désormais identifiée par un "7" car l'AST est généré à partir de zéro. Le jeu démarrera et les nouveaux jeux ne devraient pas être affectés, mais les fonctions qui utilisent les références AST, comme le chargement de jeux sauvegardés dans des versions précédentes, pourraient cesser de fonctionner correctement. Et, dans certains cas que nous verrons plus tard, les traductions pourraient également souffrir de certains problèmes.

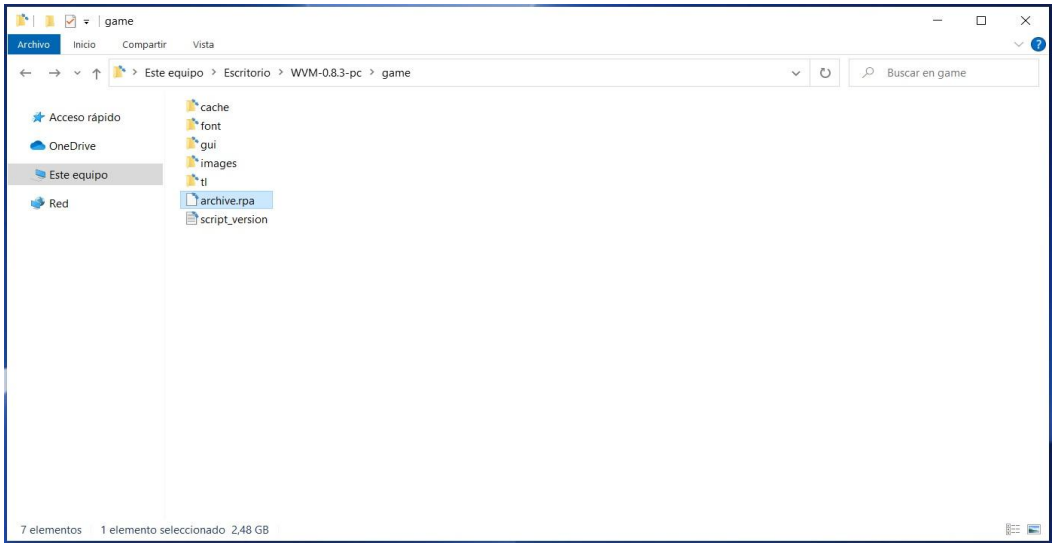
Le troisième commandement de Ren'Py : Ne supprimez pas les fichiers .rpyc du jeu original, car cela peut causer des problèmes involontaires aux joueurs.

|Top|

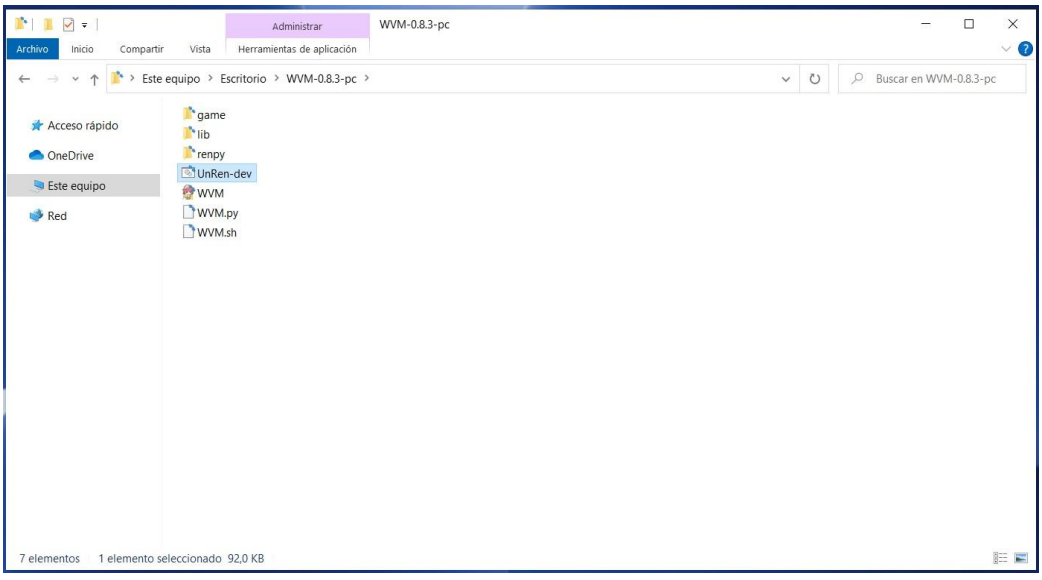
1.4 - Archives UnRen et .rpa

Il est possible que, lorsque vous avez ouvert le dossier "jeu", vous n'ayez rien vu de tout ce que je viens de mentionner. C'est parce que, lorsqu'il s'agit de construire le jeu pour la sortie, Ren'Py offre aux développeurs plusieurs options. L'option recommandée est d'inclure tous les scripts individuels au format .rpy et .rpyc, comme nous l'avons vu dans l'exemple "Quelle légende", mais ce n'est qu'une recommandation. Comme un jeu Ren'Py n'a besoin que de ses scripts .rpyc pour fonctionner, certains développeurs n'incluent que les scripts .rpyc. De cette façon, le jeu a une taille légèrement plus petite et les joueurs n'ont pas d'accès direct au code du jeu (ce qui pourrait être une raison plus importante).

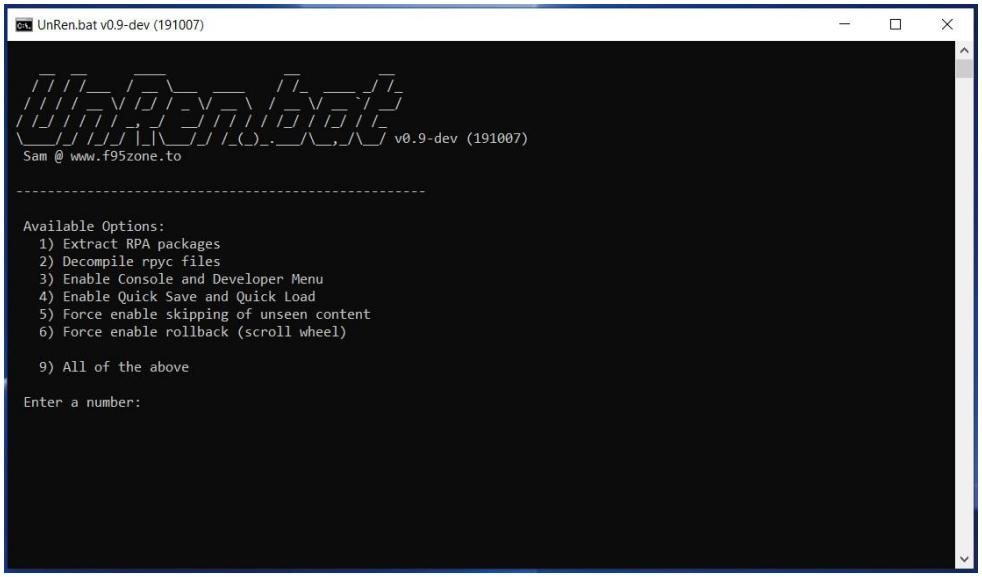
Et il y a une autre possibilité (assez courante, je dirais), car nous pourrions découvrir que dans le dossier "game" il n'y a pas de scripts avec les extensions .rpy ou .rpyc, car le développeur a choisi une autre option donnée par Ren'Py pour construire le jeu : compresser les scripts et autres fichiers (comme les images) dans un fichier .rpa. De cette façon, au lieu d'une liste de fichiers .rpy et .rpyc, nous pourrions avoir quelque chose comme cela dans le dossier "game" :



Tout d'abord, nous pouvons confirmer que, comme vous le savez probablement déjà, ce n'est pas un problème. Le jeu fonctionnera parfaitement tel quel, mais pour le traduire, nous devrons faire un détour supplémentaire, car nous avons besoin des scripts au format .rpy. Heureusement, il existe plusieurs outils qui permettent d'effectuer les tâches nécessaires sans que nous ayons à apprendre le langage Python. Pour sa simplicité, je pense que le plus facile à gérer est UnRen. LINK (attention : lien vers un forum avec beaucoup de contenu pour adultes)
Une fois UnRen téléchargé, nous le dézippons et le collons dans le dossier racine du jeu qui nous intéresse, au même niveau que l'exécutable du jeu. C'est plus facile à comprendre si vous le voyez :



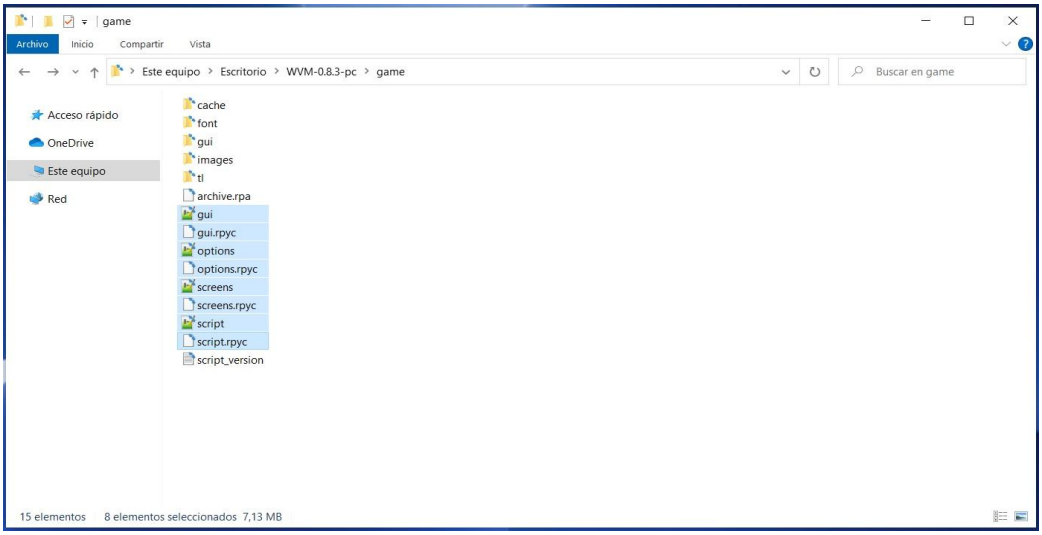
Il ne s'agit alors que de le faire fonctionner et de sélectionner ce que nous voulons faire. C'est l'écran de démarrage de UnRen :



Comme vous pouvez le voir, ici nous allons uniquement utiliser le clavier. Il nous suffit d'appuyer sur la touche correspondant à ce que nous voulons faire. L'option 1) va "dézipper" les fichiers .rpa, en extrayant tout son contenu dans le dossier "game". Il se peut que dans ce fichier .rpa il n'y ait que des scripts avec une extension .rpyc, donc, après avoir terminé cette option 1), nous devrions demander à UnRen d'exécuter l'option 2), qui consiste à décompiler les fichiers .rpyc pour créer des scripts au format .rpy que nous pourrions déjà utiliser pour notre traduction.

Nous pouvons simplement sélectionner l'option 9) qui, en une seule étape, fait la même chose que 1) et 2) et permet également d'ouvrir la console du jeu et le menu du développeur (ce qui est utile pour examiner et modifier les variables, pour accéder à des parties spécifiques du script, et aussi pour recharger le jeu automatiquement lorsque nous introduisons un changement dans la traduction), ainsi que d'autres options que le développeur du jeu aurait pu désactiver.

Eh bien, nous avons laissé UnRen faire son travail et à la fin, si nous retournons dans le dossier "jeu", nous verrons que, là où auparavant il n'y avait qu'un fichier avec une extension .rpa, apparaissent maintenant les scripts qui ont été compressés à l'intérieur.



Nous pourrions maintenant commencer à traduire. Mais, tout d'abord, laissez-moi utiliser un paragraphe pour expliquer ce qui va se passer avec le jeu. Dans le dossier "game", il y a des scripts avec l'extension .rpy, des scripts avec l'extension .rpyc et, en outre, dans le fichier "archive.rpa", il y aura les mêmes scripts .rpyc (et peut-être même les .rpy). Nous avons donc maintenant des fichiers clonés.

Heureusement, Ren'Py est conçu pour résoudre ces problèmes de clonage de manière simple : s'il trouve deux scripts portant le même nom, il exécute le plus récent. Et, dans ce cas, les plus récents sont ceux que nous venons d'extraire avec UnRen. Par conséquent, nous pourrions supprimer le fichier "archive.rpa" (et ce serait même recommandé, à condition que nous ayons un moyen de le récupérer plus tard, juste au cas où).

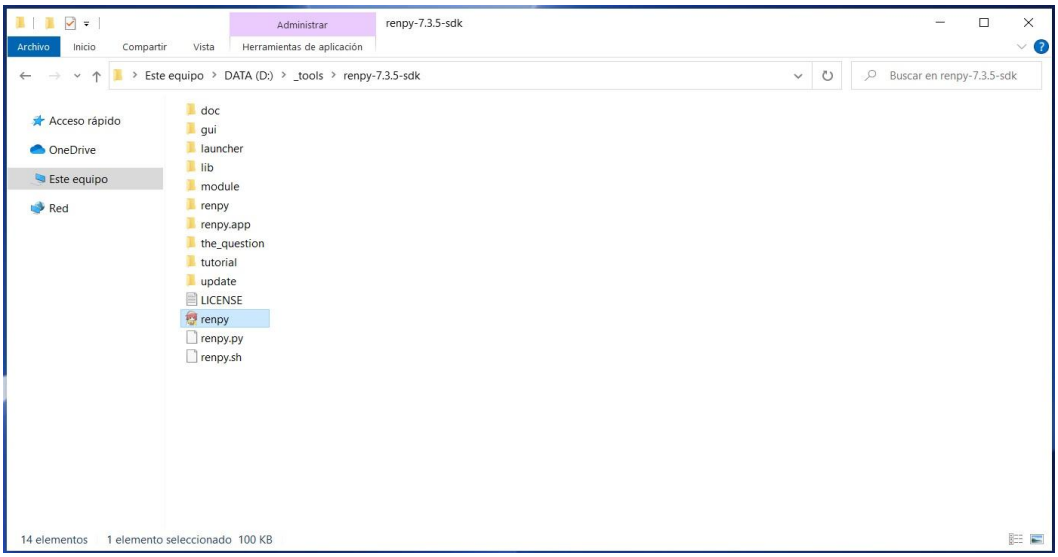
Et encore une autre précision avant de poursuivre. Si le fichier .rpa contenait à la fois les scripts .rpy et .rpyc, ceux que nous voyons maintenant dans le dossier "jeu" sont exactement les mêmes que ceux qui provenaient de l'ordinateur du développeur du jeu, car ils viennent d'être extraits par UnRen. Mais si dans le fichier .rpa il n'y avait que des scripts avec l'extension .rpyc, les scripts .rpy que nous avons maintenant dans le dossier "game" sont juste une approximation créée par UnRen sur la base de l'algorithme identifié utilisé pour compiler les scripts .rpyc. Ils peuvent donc ne pas être exactement comme les scripts .rpy originaux : ils conviennent simplement pour créer les scripts .rpyc actuels. Par exemple, tous les commentaires que le développeur aurait notés après un symbole # sont perdus, car ils ne sont jamais compilés dans les scripts .rpyc et, logiquement, UnRen, en ne lisant que les .rpyc, ne peut pas savoir s'il y avait des commentaires dans les fichiers .rpy originaux.

|Top|

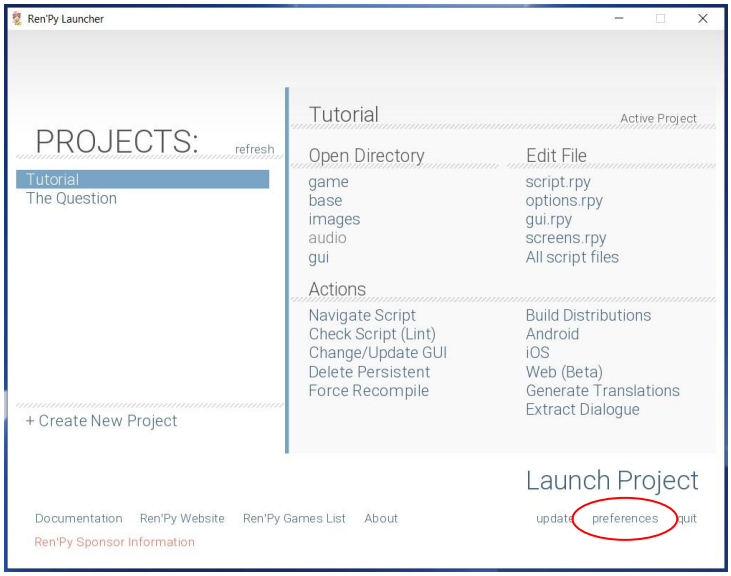
1.4.- Ren'Py SDK

Comme nous l'avons vu, les jeux Ren'Py fonctionnent sans installer de logiciel supplémentaire pour les lire. Ils sont auto-exécutables et nous permettent également de modifier leurs scripts à l'aide d'un simple éditeur de texte. Logiquement, cependant, pour les créer, il faut un logiciel très spécifique.

Ren'Py SDK est l'application qui nous permet de créer des jeux Ren'Py - et leurs traductions aussi. La première étape pour traduire un jeu Ren'Py est donc de le télécharger. C'est gratuit, propre et totalement fiable. LINK
Une fois que vous l'avez téléchargé et extrait sur votre ordinateur, ouvrez son dossier et vous verrez quelque chose comme ceci :

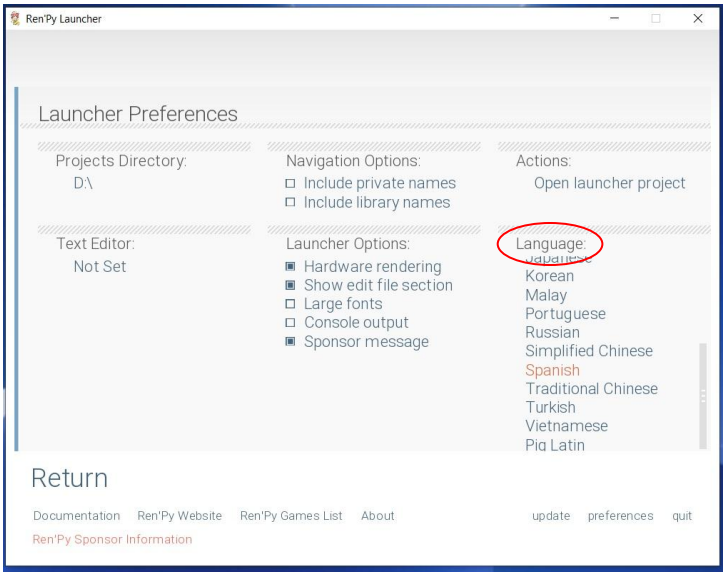


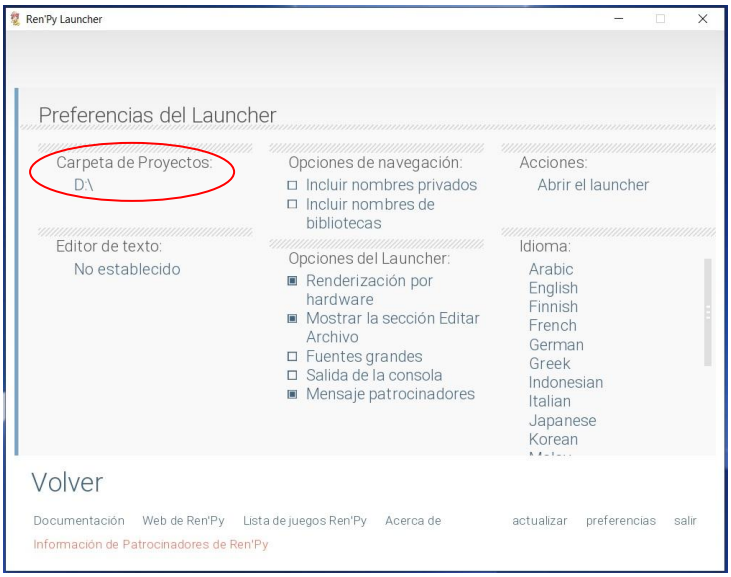
C'est-à-dire une structure très similaire à celle des jeux, avec plusieurs sous-dossiers et un exécutable. Parmi les sous-dossiers, nous avons "the_question" et "tutorial", qui sont deux tutoriels pour apprendre à utiliser Ren'Py. Cela ne fait pas de mal de les consulter, mais ils n'ajoutent pas grand-chose aux traductions. Nous devons juste lancer l'exécutable nommé "renpy" et commencer à travailler. Après avoir défini quelques paramètres, l'écran principal apparaîtra, généralement en anglais. Donc, tout d'abord, changeons la langue de l'application pour notre propre langue.



Tout d'abord, nous devons cliquer dans les "préférences" en bas à droite de la fenêtre.

Sous l'étiquette "Langue", à droite de cet écran de "préférences", nous verrons une liste des langues disponibles. Le SDK Ren'Py peut être exécuté dans toutes ces langues, il suffit donc de rechercher celle que nous souhaitons. Dès que nous cliquons dessus, la langue de l'écran change.





Maintenant, le logiciel fonctionne dans notre langue (l'espagnol, dans mon cas) et nous n'aurons plus jamais besoin de le modifier. L'option encerclée, appelée "Projects Folder", est le répertoire dans lequel Ren'Py SDK stockera et recherchera les jeux. Ici, nous devons sélectionner le répertoire où se trouve le dossier racine du jeu que nous voulons traduire. C'est-à-dire que si vous avez "What a Legend !" dans C/Documents, alors sélectionnez C/Documents. Cliquez sur "Back" et tous les jeux de ce dossier apparaîtront dans la liste "Projects", en plus des deux tutoriels qui sont inclus dans le SDK de Ren'Py.

[Top]

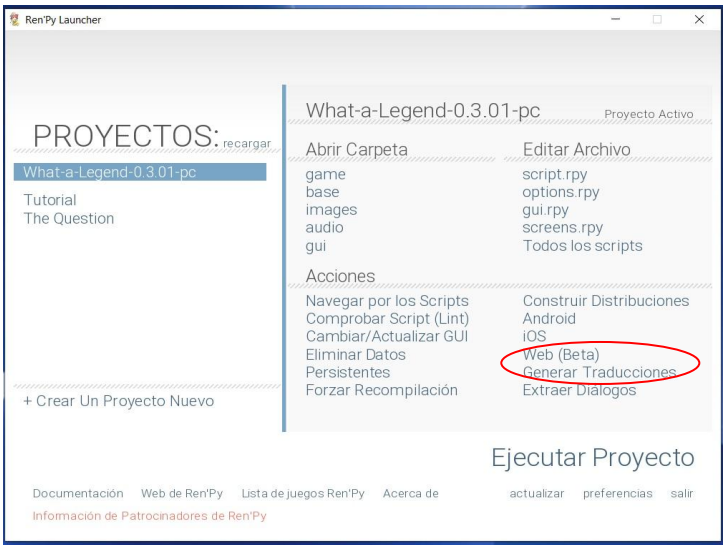
2.- TRADUISONS !

Soit parce que le développeur nous a facilité la tâche, soit parce que nous nous sommes battus pour y parvenir, l'important est qu'à ce stade, nous avons notre jeu avec ses scripts .rpy totalement exposés et prêts à être traduits. Mais, malheureusement ou non, Ren'Py ne traduit rien, il aide juste à extraire les textes traduisibles et fait apparaître les traductions là où elles doivent être. Le SDK de Ren'Py génère les scripts que nous utiliserons pour créer la traduction, avec toutes les commandes nécessaires ; cependant, il n'est pas toujours aussi facile d'extraire complètement tous les textes traduisibles que de cliquer sur un bouton, et parfois nous devrons faire un travail d'édition approfondi des scripts originaux. Mais, puisque nous sommes arrivés jusqu'ici, nous allons cliquer sur ce bouton que vous regardez sûrement déjà avec des yeux avides : Générer des traductions.

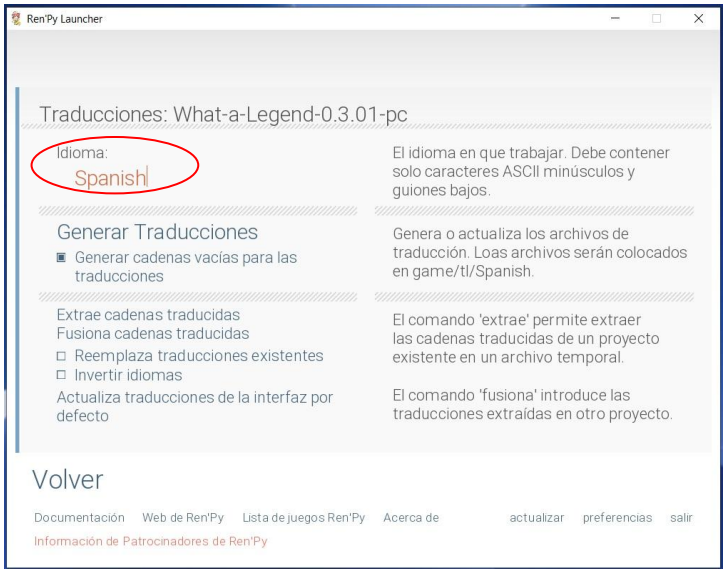
|Top|

2.1.- Génération de fichiers de traduction (et apprentissage du quatrième commandement)

Après avoir sélectionné le jeu que nous voulons traduire (il apparaîtra en surbrillance dans la section "Projets"), cliquez sur le bouton "Générer des traductions" sur le côté droit de l'écran.



Dans la case "Langue", nous devons écrire la langue dans laquelle nous voulons traduire le jeu. C'est juste une identification interne, donc nous pouvons écrire ce que nous voulons (notez qu'aucun caractère spécial n'est autorisé, tel que ñ ou voyelles accentuées). Le plus important est de se souvenir de ce que nous avons écrit ici et de toujours garder à l'esprit le quatrième commandement de Ren'Py :



> Le quatrième commandement de Ren'Py : Pour Ren'Py, une lettre majuscule n'est pas égale à une lettre minuscule, jamais, en aucune circonstance.

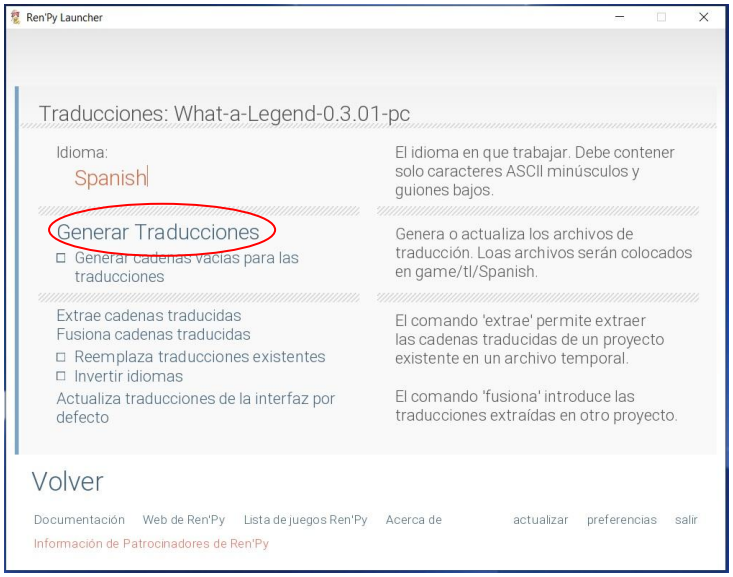
Vous pouvez donc écrire espagnol, espagnol, espanol, français, francais, ship, yellow ou Monkey. Tout ce que vous voulez. Mais, évidemment, il est conseillé d'utiliser un mot compréhensible. Et, comme je l'ai dit, le plus important est de se rappeler ce que vous avez écrit et comment vous l'avez écrit. Par exemple, j'utilise toujours le mot "espagnol", avec un "S" majuscule, et c'est ce que vous verrez dorénavant dans les exemples fournis.

Ensuite, il y a plusieurs options, mais la seule qui nous importe est celle qui dit : "Générer des chaînes vides pour les traductions" (ou tout ce qui est dit dans votre langue). Ici, comme presque toujours, tout est une question de goût. Lorsque nous traduisons, nous aurons toujours une ligne visible avec le texte original pour nous guider, mais l'idée est que, si nous sélectionnons cette option, les scripts de traduction seront générés avec quelques lignes vides pour que nous puissions y écrire directement notre traduction. Si nous ne sélectionnons pas cette option, les scripts seront générés avec ces lignes écrites dans la langue originale du jeu et nous devrons supprimer ces mots et les remplacer par leur traduction. Ici, nous pouvons voir une comparaison côte à côte de l'aspect des fichiers de traduction si nous cochons cette case (à gauche) et si nous ne la cochons pas (à droite).

```
# TODO: Translation updated at 2020-12-11 13:16

# game/convo_oc.rpy:15
translate Spanish convo_gate_fdd309da:

    # kevin "STOP!" with vpunch
    kevin "" with vpunch

# game/convo_oc.rpy:18
translate Spanish convo_gate_e5ccb99b:

    # kevin "Did you manage to get a passage permit?"
    kevin ""

# game/convo_oc.rpy:21
translate Spanish convo_gate_bc693870:

    # pov "Umm..."
    pov ""

# game/convo_oc.rpy:24
translate Spanish convo_gate_6a0bcf57:

    # kevin "I'm sorry, buddy..."
    kevin ""

# game/convo_oc.rpy:26
translate Spanish convo_gate_26193626:

    # kevin "...but no permit, no passage. That is the law."
    kevin ""
```

```
# TODO: Translation updated at 2020-12-11 13:41

# game/convo_oc.rpy:15
translate Spanish convo_gate_fdd309da:

    # kevin "STOP!" with vpunch
    kevin "STOP!" with vpunch

# game/convo_oc.rpy:18
translate Spanish convo_gate_e5ccb99b:

    # kevin "Did you manage to get a passage permit?"
    kevin "Did you manage to get a passage permit?"

# game/convo_oc.rpy:21
translate Spanish convo_gate_bc693870:

    # pov "Umm..."
    pov "Umm..."

# game/convo_oc.rpy:24
translate Spanish convo_gate_6a0bcf57:

    # kevin "I'm sorry, buddy..."
    kevin "I'm sorry, buddy..."

# game/convo_oc.rpy:26
translate Spanish convo_gate_26193626:

    # kevin "...but no permit, no passage. That is the law."
    kevin "...but no permit, no passage. That is the law."
```

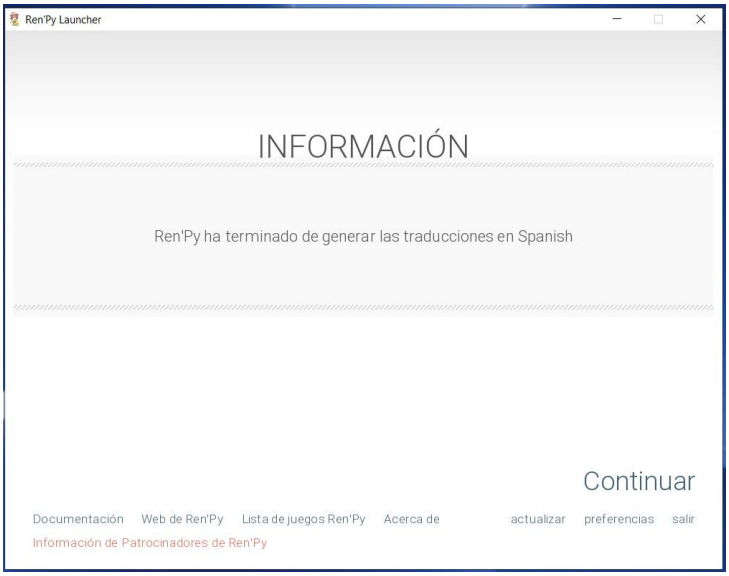
Il peut sembler plus pratique (c'est en fait le cas) de générer des chaînes vides, mais, si d'une manière ou d'une autre nous oublions de traduire une ligne, lorsque les joueurs atteignent cette ligne dans le jeu, absolument aucun texte ne sera affiché à l'écran. Dans l'autre cas, le texte serait affiché dans la langue d'origine, ce qui peut nous être utile lorsque nous testons une traduction incomplète. C'est particulièrement important pour les menus, car cela nous permettra d'avoir toutes les options actives même si nous ne les avons pas encore traduites.



Personnellement, je préfère NE PAS générer de chaînes vides, donc je laisse cette case non cochée, mais si vous allez utiliser des traducteurs automatiques, vous pourriez être intéressé par la génération de chaînes vides.

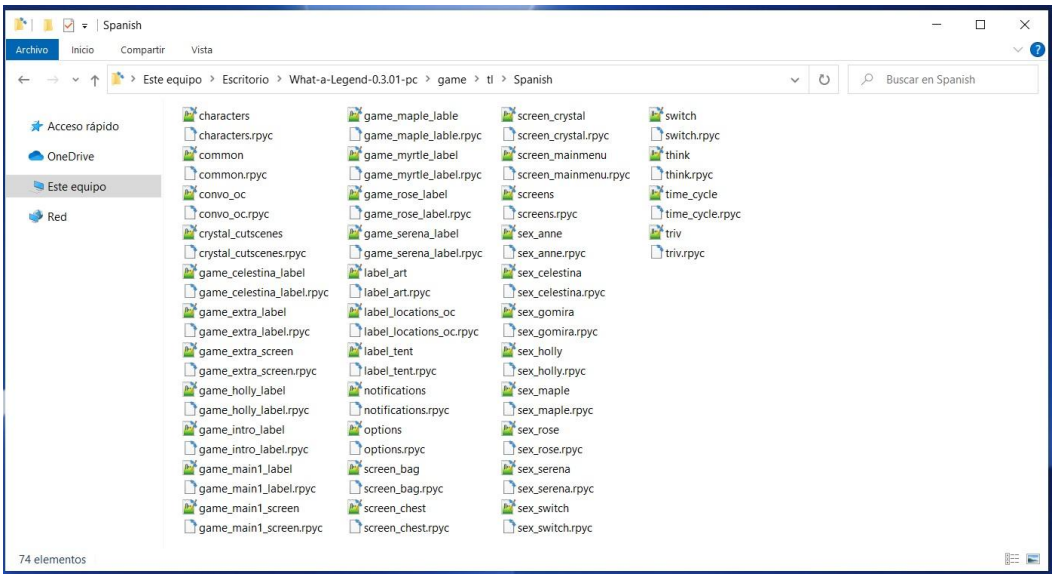
Il ne nous reste plus qu'à cliquer sur le bouton "Générer des traductions" (celui qui est encerclé). Comme l'indique le texte informatif dans la colonne de droite, en faisant cela, Ren'Py SDK créera les fichiers de traduction à l'intérieur du dossier "game/tl/Spanish" (le nom de ce dossier sera celui que nous avons écrit dans la case "Language").

S'il n'y a plus de problème, ce message apparaîtra lorsque le SDK de Ren'Py aura fini de fonctionner. Nous pouvons cliquer sur "Continuer", pour revenir à l'écran principal, ou fermer directement l'application Ren'Py SDK.



Si nous allons dans le dossier "game" de notre jeu, dans le sous-dossier "tl", nous verrons deux autres dossiers : un nommé "None" et un autre avec le mot que nous avons écrit dans la boîte de langue du SDK de Ren'Py. Dans ce cas, "Espagnol" :

Et dans le dossier "Espagnol", nous trouverons les scripts de traduction. Qu'a fait le SDK de Ren'Py pour les créer ? Eh bien, il a parcouru tous les scripts .rpy originaux, un par un dans l'ordre alphabétique, et, chaque fois qu'il a trouvé un texte traduisible, il a généré un fichier .rpy (et son .rpyc correspondant) avec le même nom que le script original, en y inscrivant les chaînes qui doivent être traduites.



De plus, un script supplémentaire aura été généré : nommé "common.rpy", il contiendra les commandes traduisibles des menus de Ren'Py qui ne sont pas spécifiques au jeu. Par exemple, le message d'alerte qui apparaît lorsque nous fermons le jeu, nous demandant de confirmer notre choix, ou les mois et jours de la semaine qui seront utilisés pour nommer nos jeux sauvegardés. C'est un fichier volumineux et assez fastidieux à traduire, mais il y a un point positif : en général, il est commun à la plupart des jeux créés avec la même version du SDK de Ren'Py, de sorte qu'il peut être interchangeable d'un jeu à l'autre. Nous pouvons donc réutiliser un script "common.rpy" traduit que nous avons déjà dans un jeu précédent. Bien sûr, avant de le remplacer, nous devons toujours vérifier que les chaînes de caractères correspondent, car parfois il y a de petites modifications et nous pourrions casser quelque chose de manière involontaire.

Pour l'instant, la traduction ne consiste qu'à ouvrir les scripts .rpy du dossier "tl/Spanish" avec un éditeur de texte et à commencer à travailler dessus. Nous pouvons changer le nom de ces scripts (et même les fusionner dans un seul fichier), puisque Ren'Py cherchera les traductions chaîne par chaîne et ne se soucie pas de savoir dans quel fichier elles sont stockées, mais la pratique standard est "ne touchez à rien" : de cette façon, il est plus facile d'identifier chaque script traduit avec son script original.

|Top|

2.2.- Les deux fonctions de traduction (et encore un autre commandement)

Il n'est pas rare que, la première fois que nous commençons à traduire, nous écrivions quelques lignes dans notre langue et que nous nous disions soudain : "Attendez, où sont les options pour le joueur ? Le jeu ne m'a-t-il pas demandé ici si je voulais faire une chose ou une autre ?" Pas de panique : dans les fichiers de traduction, les textes traduisibles n'apparaissent pas exactement dans le même ordre que dans le script original. Lorsqu'il s'agit d'extraire les chaînes traduisibles de chaque script, Ren'Py les divise en deux catégories : celles qui forment le bloc de dialogue et celles qui se réfèrent aux options de menu, aux variables, aux noms de caractères, etc. Pourquoi ? Eh bien, parce que chacun de ces groupes va utiliser une fonction d'extraction et de traduction différente.

Les chaînes qui forment le bloc de dialogue ne généreront aucun problème d'extraction car le SDK de Ren'Py les identifiera sans problème. Par contre, ce qu'il en fera causera davantage de nuisances et même quelques maux de tête lors de la traduction et de la création de notre patch de traduction. Parce que Ren'Py les crypte pour économiser la mémoire : en utilisant la méthode de cryptage hexadécimal MD5 sur 8 octets, chaque chaîne est identifiée par un code alphanumérique qui dépend de l'étiquette où se trouve ce texte dans le script original et du contenu de la chaîne elle-même.

On peut mieux le voir avec un exemple. Tout d'abord, rappelons les premières lignes du script "convo_oc.rpy" du jeu "Quelle légende !

```
1  # ==================== OLD Capital Base Conversations ==========================================
2  # Being stopped at the gate of the old capital ================================================
3  label convo_gate:
4      call hide_ui from _call_hide_ui_104
5      call silent from _call_silent_42
6
7      scene scene_oc_bridge_gate_talk
8      if current_hour == "Night" or current_hour == "Evening":
9          show kevin at g_cright:
10             xalign 0.6
11         show pov at m_left
12         with quickfade
13         show pov ewide bup mdislike hfshock hbshock
14         show kevin hbstop eangry
15         kevin "STOP!" with vpunch
16         show kevin hbpoint edoubt
17         show pov -mdislike hfneu1 hbneu1
18         kevin "Did you manage to get a passage permit?"
19         show pov mno bdoubt eneub hfhead
20         show kevin eneu hbneu
21         pov "Umm..."
22         show pov eneu bsad msad
23         show kevin esad hfspearmove
24         kevin "I'm sorry, buddy..."
25         show pov mcry eflat bneu hfneu1
26         kevin "...but no permit, no passage. That is the law."
```

La première ligne avec du texte traduisible (la première "chaîne") est la ligne 15 du script original : sous l'étiquette "convo_gate", un personnage identifié comme Kevin dit "STOP !" (avec vpunch, un effet qui va " secouer" l'écran. Le reste du code jusqu'à ce point ne contient aucun texte à afficher à l'écran

En générant le script de traduction avec des chaînes vides, Ren'Py a transformé toutes ces informations :

```
# TODO: Translation updated at 2020-12-11 13:16

# game/convo_oc.rpy:15
translate Spanish convo_gate_fdd309da:

    # kevin "STOP!" with vpunch
    kevin "" with vpunch

# game/convo_oc.rpy:18
translate Spanish convo_gate_e5ccb99b:

    # kevin "Did you manage to get a passage permit?"
    kevin ""

# game/convo_oc.rpy:21
translate Spanish convo_gate_bc693870:

    # pov "Umm..."
    pov ""

# game/convo_oc.rpy:24
translate Spanish convo_gate_6a0bcf57:

    # kevin "I'm sorry, buddy..."
    kevin ""

# game/convo_oc.rpy:26
translate Spanish convo_gate_26193626:

    # kevin "...but no permit, no passage. That is the law."
    kevin ""
```

On peut voir que, pour chaque chaîne, Ren'Py a généré un bloc de quatre lignes. Analysons ces blocs :

```
# game/convo_oc.rpy:15
translate Spanish convo_gate_fdd309da:

    # kevin "STOP!" with vpunch
    kevin "" with vpunch
```

Les lignes qui commencent par le symbole # sont simplement informatives et nous pourrions les supprimer sans conséquences. La première nous indique l'emplacement de cette ligne dans le script original : il s'agit de la ligne 15 du fichier "convo_oc.rpy", à l'intérieur du dossier "game". L'autre ligne commencée par # est le texte original à traduire, qui nous est aimablement offert par Ren'Py pour que nous sachions quoi écrire dans notre langue. Et la dernière ligne est celle où nous allons écrire notre traduction (nous ne traduirons que ce qui est entre guillemets, nous devrions laisser le reste tel quel). Si nous avions généré la traduction sans chaînes vides, le texte de cette ligne apparaîtrait à nouveau en anglais.

La clé se trouve à la deuxième ligne. C'est ce qui active la fonction de traduction pour cette ligne spécifique du code original. Lorsque nous jouons à une version traduite, Ren'Py exécute le jeu selon les fichiers .rpyc originaux du dossier "jeu", mais il a été ordonné d'afficher les textes traduits au lieu des chaînes originales. Cette commande indique à Ren'Py que, dans chaque ligne du bloc de dialogue, il doit rechercher dans les scripts une ligne avec la commande "translate" plus la langue active ("espagnol", dans ce cas, ce qui est ce que nous avons écrit lors de la génération des fichiers de traduction) plus le code de cryptage qui correspond à la chaîne originale. Ce code de cryptage commence par le nom de l'étiquette où se trouve cette ligne spécifique ("convo_gate") et se poursuit avec le résultat de l'application du système MD5 au contenu de la ligne (apparemment, dans ce système de cryptage, "STOP !" équivaut à fdd309da).

S'il y avait une autre chaîne identique (un autre "STOP !") dans cette même étiquette "convo_gate", Ren'Py devrait lui attribuer le même code de cryptage. Mais cela pourrait entraîner des conflits, c'est pourquoi il ajoutera un _1 à la fin du code de la deuxième ligne, s'il y avait encore une autre chaîne identique, il ajouterait un _2 et ainsi de suite. Ainsi, les codes de cryptage sont toujours uniques pour chaque chaîne, même pour les chaînes identiques dans le bloc de dialogue.

Mais que se passe-t-il si on se réfère au script original et qu'on écrit "Stop !" au lieu de "STOP ! Dans ce cas,la traduction de "STOP !" ne fonctionnera plus, car le code MD5 de "Stop !" sera différent et Ren'Py ne le trouvera pas dans les scripts de traduction, donc le texte original sera affiché à la place. Cela cause quelques désagréments lors de la traduction de nouvelles versions de jeux encore en développement, car il est courant que les créateurs corrigent des fautes de frappe mineures et réécrivent quelques phrases ici et là. Nous devrons retraduire certaines chaînes de caractères que nous avions déjà traduites auparavant, car même la modification la plus minime changera entièrement leur code de cryptage et elles seront complètement différentes pour le moteur de Ren'Py. Cela se produit également lorsque son label est renommé, même si le contenu de la chaîne n'est pas modifié

> Le cinquième commandement de Ren'Py : Toute modification minime apportée à la chaîne de caractères originale invalidera la traduction préexistante de cette ligne.

Heureusement, s'il s'agit d'un simple déplacement de ligne (la chaîne passe de la ligne 7 à la ligne 8, mais son contenu ne change pas et son libellé ne change pas non plus), la traduction précédente continuera à fonctionner, puisque ce déplacement n'implique pas de changement du code de cryptage. La ligne #commentée qui nous indique où se trouve cette chaîne dans le script original sera dépassée, mais cela n'a pas d'impact majeur sur notre traduction.

Mais, comme je l'ai dit, Ren'Py utilise deux systèmes de traduction. En plus du cryptage basé sur le code, pour le reste des chaînes qui n'appartiennent pas au bloc de dialogue, un système plus simple de traduction directe est utilisé. De quelles chaînes s'agit-il ? Eh bien, comme je l'ai dit au début de cette section, je parle des options qui sont présentées au joueur pendant le jeu, mais aussi des noms des personnages, des variables potentielles utilisées dans le jeu qui ont une valeur de texte, des menus du système (préférences, sauvegarder et charger le jeu...), etc.

Lorsque Ren'Py lance un jeu traduit, toutes ces chaînes seront remplacées à l'écran par une substitution directe, sans aucun détour ni cryptage. Pour ce faire, lors de la génération des scripts de traduction, le SDK de Ren'Py les regroupe tous à la fin du document, sous la commande "translate Spanish strings". C'est le début de la deuxième section du script de traduction "convo_oc" de "Quelle légende !

```
8001 translate Spanish strings:
8002
8003     # game/convo_oc.rpy:107
8004     old "Permit"
8005     new ""
8006
8007     # game/convo_oc.rpy:107
8008     old "Old Capital"
8009     new ""
8010
8011     # game/convo_oc.rpy:107
8012     old "Helping pixies"
8013     new ""
8014
8015     # game/convo_oc.rpy:107
8016     old "Dungeon"
8017     new ""
8018
8019     # game/convo_oc.rpy:107
8020     old "Go back"
8021     new ""
8022
8023     # game/convo_oc.rpy:320
8024     old "Passage permit"
8025     new ""
8026
8027     # game/convo_oc.rpy:320
8028     old "Challenge"
8029     new ""
```

Comme vous pouvez le voir, il n'y a pas de codes étranges ici. Et si la commande "Traduire" était appliquée auparavant à chaque ligne, maintenant ils sont tous traduits en utilisant une seule commande. Nous avons toujours une première ligne avec le symbole # pour nous informer de l'emplacement de la chaîne traduisible dans le script original, mais ensuite il oublie les codes de cryptage et va directement afficher le texte original avec l'"ancienne" commande ; et ensuite nous avons une "nouvelle" commande avant le texte qui devrait apparaître lorsque le jeu lance une traduction en langue "espagnole" (celle que nous avons écrite dans le SDK de Ren'Py lorsque nous générions les scripts de traduction).

Contrairement à ce qui se passait auparavant, aucune chaîne répétée n'apparaîtra désormais non seulement dans ce script, mais aussi dans l'ensemble des scripts de traduction. Lorsque le SDK de Ren'Py passe en revue les scripts originaux pour générer les fichiers de traduction, il n'extrait chaque chaîne que la première fois qu'elle est trouvée et ignore les doublons qui pourraient exister. Ainsi, lors de la lecture, chaque fois que cette chaîne apparaîtra, elle sera remplacée par la traduction que nous avons introduite cette fois-là. Cela nous évite de répéter le travail, mais devient un inconvénient lorsque nous rencontrons des chaînes identiques qui peuvent avoir des significations différentes selon le contexte. Par exemple, pensez à une chaîne qui est répétée plusieurs fois pendant le jeu mais qui dit simplement "Bon" : parfois elle peut signifier "Correct" et parfois elle peut se référer au côté droit de quelque chose, ou indiquer une direction, et peut-être que dans notre langue nous utilisons un mot différent pour chaque concept. Mais le SDK de Ren'Py n'extrait que la première chaîne "Right", donc nous ne pouvons la traduire qu'une seule fois et cette traduction apparaît toujours, donc parfois le texte traduit n'a pas de sens. Nous verrons comment résoudre ce problème plus tard.

En utilisant l'exemple ci-dessus, une fois que nous aurons écrit dans ce script la traduction de la chaîne "Permis", cette traduction apparaîtra tout au long du jeu chaque fois qu'il y aura une chaîne "Permis" en dehors du bloc de dialogue. Mais attention : selon le quatrième commandement de Ren'Py, s'il y avait une autre chaîne avec le texte "Permis", nous devrions la traduire, car Ren'Py est parfaitement sensible à la casse.

Pour en revenir au sujet, pourquoi Ren'Py utilise-t-il deux fonctions de traduction différentes ? Eh bien, pour des questions d'agilité et d'utilisation efficace de la mémoire. Techniquement, ce système de traduction de remplacement direct pourrait être appliqué à absolument toutes les chaînes de caractères du jeu, mais généralement le bloc de dialogue est assez large et se compose de phrases beaucoup plus longues qui ne sont presque jamais répétées. Ainsi, lorsque le programme doit afficher une traduction, il faut moins de temps pour trouver et remplacer des milliers de codes cryptés que des milliers de lignes entières qui sont généralement plus longues que ces codes. Cependant, ces autres chaînes de menu sont généralement plus courtes et beaucoup moins nombreuses, et il est possible qu'elles soient plus fréquemment répétées dans les scripts, de sorte que Ren'Py peut se permettre de faire une recherche spécifique pour le contenu exact de ces chaînes sans décalage notable pendant le jeu.

Quoi qu'il en soit, ce ne sont là que quelques notions parfaitement oubliables. Le plus important est de savoir que, pour le SDK de Ren'Py, il existe deux types de "chaînes" et que chacun de ces types utilise un système de traduction différent, et c'est pourquoi ils apparaîtront séparément dans les scripts de traduction : nous trouverons d'abord toutes les chaînes appartenant au bloc de dialogue, puis toutes celles correspondant aux menus, aux options et aux variables.

|Top|

2.3.- Aider et/ou remplacer l'extracteur

La raison pour laquelle nous avons expliqué dans le point précédent les deux types de fonctions de traduction de Ren'Py est que, malheureusement, le SDK de Ren'Py n'est pas toujours capable de détecter absolument toutes les chaînes que nous devrions traduire : oui, il extraira toutes celles qui intègrent le bloc de dialogue (celles qui seront traduites grâce au code de cryptage), mais il ne pourra peut-être pas identifier toutes les chaînes qui sont traduites par la méthode de remplacement direct (bien qu'il extraie les options qui permettent aux joueurs de prendre des décisions pendant le jeu).

Par exemple, pour que le SDK de Ren'Py puisse extraire automatiquement les noms des personnages, le développeur du jeu aurait dû définir ces personnages d'une manière très spécifique que, par ignorance, de nombreux développeurs n'utilisent pas. Mais s'ils ne le font pas, nous pouvons le faire.

Comme les créateurs de "Quelle légende !" ont fait leurs devoirs, nous facilitant la vie à nous, traducteurs, pour ces exemples nous allons utiliser l'autre jeu que nous avions déjà utilisé pour parler des fichiers .rpa et de l'extracteur UnRen. Voyons donc comment le développeur de "WVM" définit un des personnages impliqués dans l'histoire. Dans ce cas, dans le fichier "script.rpy", il a créé un "MC intérieur" pour indiquer aux joueurs que ce que nous lisons dans la boîte de dialogue est une pensée de notre propre personnage :

```
224 define mcm = Character ("Your thoughts",color="#FFFFFF", who_outlines=[ (2, "#000000") ], what_outlines=[ (2, "#000000") ])
```

C'est une ligne typique du code de Ren'Py. Elle commence par une commande (define) qui indique ce que fait cette ligne spécifique, en l'occurrence la définition d'une variable. Cette variable se voit attribuer un nom (mcm) qui sera utilisé dans le reste du script pour l'identifier, et elle est ordonnée pour se comporter comme une variable de type caractère (Character). Maintenant, Ren'Py sait que, chaque fois qu'il trouve les lettres mcm avant une chaîne de caractères, il doit afficher un nom au-dessus de la zone de texte pour que le joueur sache quel personnage parle. Et qu'est-ce qui y sera affiché ? Ce qui apparaît ensuite entre parenthèses : le nom du personnage ("Vos pensées"), dans une couleur spécifique, avec une ligne entourant les lettres pour les mettre en évidence.

Évidemment, "Vos pensées" est une chaîne que nous devrions traduire, mais, si nous générons les scripts de traduction avec le SDK de Ren'Py, elle n'apparaîtrait nulle part. Les mystères de Ren'Py. Que pouvons-nous faire ? Eh bien, il y a trois options. La première option, que nous rejetons purement et simplement, est de l'accepter et de laisser la chaîne s'afficher toujours en anglais. La deuxième option est de faire comprendre à Ren'Py SDK que ce texte cité est une chaîne de caractères que nous voulons traduire et pas seulement un code interne comme celui qui indique la couleur du texte. Regardez : Nous avons modifié le script original pour mettre la chaîne "Your thoughts" entre parenthèses et nous avons tapé un soulignement _ avant les parenthèses. Cette combinaison de symboles _( ) permettra au SDK de Ren'Py d'identifier le texte cité entre parenthèses comme une chaîne traduisible.

```
224 define mcm = Character (_("Your thoughts"),color="#FFFFFF", who_outlines=[ (2, "#000000") ], what_outlines=[ (2, "#000000") ])
```

Et si nous générions maintenant les scripts de traduction, nous trouverions cela dans la section de traduction directe :

```
translate Spanish strings:

    # script.rpy:224
    old "Your thoughts"
    new "Tus pensamientos"
```

C'est le résultat après l'avoir traduit en espagnol, bien sûr. Mais l'important est que, si nous n'avions pas modifié le script original pour remplacer "Your thoughts" par _("Your thoughts"), cette chaîne n'apparaîtrait pas dans les scripts de traduction.

Cela ne signifie pas que nous n'aurions jamais pu le traduire : comme il s'agit d'une traduction directe, nous pourrions toujours l'écrire dans notre script de traduction sans l'aide du SDK de Ren'Py. C'est la dernière des trois options que j'ai mentionnées précédemment : au lieu de modifier les scripts originaux, nous écrivons dans les scripts de traduction les chaînes traduisibles que le SDK de Ren'Py n'a pas détectées. Pour ce faire, nous irions dans la section du script de traduction où apparaissent les traductions directes (celle qui commence par la fonction translate Spanish strings :) et nous écririons une fonction de traduction avec le même format que celui que nous avons vu ci-dessus : en respectant toujours l'indentation des 4 espaces, ainsi que les citations comme le disent les deux premiers commandements de Ren'Py, dans une ligne nous écririons l'ancienne commande suivie de la chaîne à traduire (en respectant scrupuleusement son écriture, comme le disent les quatrième et cinquième commandements de Ren'Py), et ensuite nous écririons en dessous la nouvelle commande suivie de la traduction. La ligne qui commence par un symbole # ne serait pas nécessaire car elle est simplement informative.

Personnellement, j'ai l'habitude de réviser et de modifier les fichiers originaux d'abord pour que le SDK de Ren'Py puisse ensuite faire un travail d'extraction complet. Bien sûr, il m'arrive de manquer une chaîne de caractères et de devoir régénérer les scripts de traduction plusieurs fois, ce qui n'est pas plus compliqué que d'ouvrir à nouveau l'application Ren'Py SDK et de cliquer sur "generate translations". D'autres traducteurs préfèrent modifier les scripts originaux le moins possible, ils génèrent donc ces scripts de traduction au début puis les modifient manuellement pour inclure tout ce que le SDK de Ren'Py n'a pas extrait. C'est juste une question de goûts personnels et d'habitudes de travail.

Que vous vouliez aider le SDK de Ren'Py au préalable ou que vous préfériez écrire dans les scripts de traduction les chaînes de caractères manquantes, vous devrez parcourir les scripts originaux et rechercher ce type de commandes :

- `Character("...")`, used to define characters, as we have seen in the example
- `text "..."`, used to display text on a specific screen, outside of the textbox
- `textbutton "..."`, used for texts that perform an action when clicking on them
- `tooltip '...'`, used to display a pop-up message when hovering over a point
- `renpy.input("...")`, used to allow in-game typing (to let players name characters, usually)
- `$ renpy.notify('...', 'unlock')`, used to display messages after completing an action

En outre, nous avons des variables de texte. Nous pouvons trouver ici plusieurs possibilités, mais aucune d'entre elles ne sera automatiquement extraite par le SDK de Ren'Py :

- `default` *variable_name* = "..."
- `define` *variable_name* = "..."
- `$` *variable_name* = "..."

Il s'agit donc d'envelopper tous ces guillemets du symbole _() et de générer les scripts de traduction, ou de copier son contenu (citations comprises) directement dans un script de traduction (peu importe lequel) avec l'ancienne commande et la nouvelle commande ci-dessous avec sa traduction. Notez que vous devez copier tout ce qui apparaît entre guillemets, et pas seulement le texte à traduire, car il y a parfois des balises à l'intérieur des symboles {} qui sont utilisées pour afficher le texte en gras, en italique, avec une autre police ou une taille différente, etc. Ces balises font également partie de la chaîne que Ren'Py recherchera et remplacera, elles doivent donc apparaître derrière l'ancienne commande afin d'aider Ren'Py à identifier correctement cette chaîne exacte.

Enfin, une remarque supplémentaire : le symbole _( ) n'est utilisé que pour extraire les chaînes de caractères entre parenthèses. Une fois extraites dans un script de traduction, nous pouvons supprimer ce symbole et la traduction fonctionnera correctement, tout comme si nous écrivions ces chaînes manuellement dans le script de traduction. Par conséquent, lors de la traduction de nouvelles versions du même jeu, il ne serait pas nécessaire de modifier à nouveau les scripts originaux, car nous pouvons réutiliser les anciens scripts de traduction dans lesquels ces chaînes apparaissent déjà traduites. En outre, il n'est pas nécessaire d'introduire le script modifié dans le patch de traduction, puisque les joueurs n'ont pas besoin du symbole _( )* : une fois que la chaîne traduisible apparaît dans le script de traduction, le texte traduit s'affiche à l'écran. J'admets que j'ai appris cela relativement récemment et que, si j'avais su plus tôt, je me serais épargné quelques heures de travail inutile.

|Top|

2.4.- L'option de changement de langue

La dernière chose que nous devrions faire avant de commencer à traduire, ou peut-être la première, est de penser à la manière dont nous allons activer la traduction dans le jeu, une fois qu'elle aura été créée. Voulons-nous que le jeu commence toujours dans notre langue ? Allons-nous ajouter l'option de changement de langue dans le menu principal ou dans le menu des options ? Préférons-nous laisser les joueurs décider de la langue dans laquelle ils veulent jouer chaque fois qu'ils commencent le jeu ? Et comment faisons-nous tout cela, avec une question en texte clair ou en créant un écran avec des drapeaux et d'autres éléments visuels ? Comme toujours, cela dépend des goûts personnels du traducteur, du temps disponible et des compétences de codage, et bien sûr aussi de la configuration du jeu original.

La façon la plus simple est de faire en sorte que le jeu commence toujours dans la langue souhaitée. Pour ce faire, il suffit d'ouvrir le script de n'importe quel jeu (de préférence le script "gui.rpy", car une grande partie de la configuration du jeu y est définie, mais en fait cela n'a pas d'importance et nous pouvons même en créer un nouveau) et d'écrire cette ligne de code, sans indentation :

```
define config.language = "Spanish"
```

Où "espagnol" est, rappelons-le, le mot que j'ai entré dans le SDK de Ren'Py lors de la génération des scripts de traduction ; et donc la référence utilisée par les commandes "translate", donc vous utilisez le terme que vous avez choisi. Si nous ne faisons rien d'autre, le jeu démarrera toujours dans notre langue et les joueurs n'auront aucune option dans le jeu pour revenir à la langue d'origine : même après avoir désactivé cette ligne de code (en la supprimant ou en écrivant un symbole # au début de la ligne) le jeu ne démarrera plus dans la langue d'origine, car par défaut les jeux Ren'Py démarrent toujours dans la dernière langue dans laquelle ils ont été joués. Un nouveau config.language doit être établi avec la langue d'origine, qui pour Ren'Py est toujours nommée None.

La meilleure pratique, dans tous les cas, consiste à toujours ajouter une option de changement de langue dans le menu Préférences. En supposant que le jeu utilise l'écran de préférences que Ren'Py incorpore par défaut, il faudrait aller dans le script original "screens.rpy", chercher la section du menu "Préférences" et la faire ressembler à ceci :

```
718 init -501 screen preferences():
719     tag menu
720
721
722     use game_menu(_("Preferences"), scroll="viewport"):
723
724         vbox:
725
726             hbox:
727                 box_wrap True
728
729                 if renpy.variant("pc"):
730
731                     vbox:
732                         style_prefix "radio"
733                         label _("Display")
734                         textbutton _("Window") action Preference("display", "window")
735                         textbutton _("Fullscreen") action Preference("display", "fullscreen")
736
737                 vbox:
738                     style_prefix "radio"
739                     label _("Rollback Side")
740                     textbutton _("Disable") action Preference("rollback side", "disable")
741                     textbutton _("Left") action Preference("rollback side", "left")
742                     textbutton _("Right") action Preference("rollback side", "right")
743
744                 vbox:
745                     style_prefix "check"
746                     label _("Skip")
747                     textbutton _("Unseen Text") action Preference("skip", "toggle")
748                     textbutton _("After Choices") action Preference("after choices", "toggle")
749                     textbutton _("Transitions") action InvertSelected(Preference("transitions", "toggle"))
750
751                 vbox:
752                     style_prefix "pref"
753                     label _("Language")
754                     textbutton _("English") action Language(None)
755                     textbutton _("Español") action Language("Spanish")
```

Le code exact à écrire serait le suivant (n'oubliez pas de marquer les indentations avec la barre d'espacement):

```
vbox:
    style prefix "pref")
    label _("Language")
    textbutton _("English") action Language(None)
    textbutton _("Español") action Language("Spanish")
```

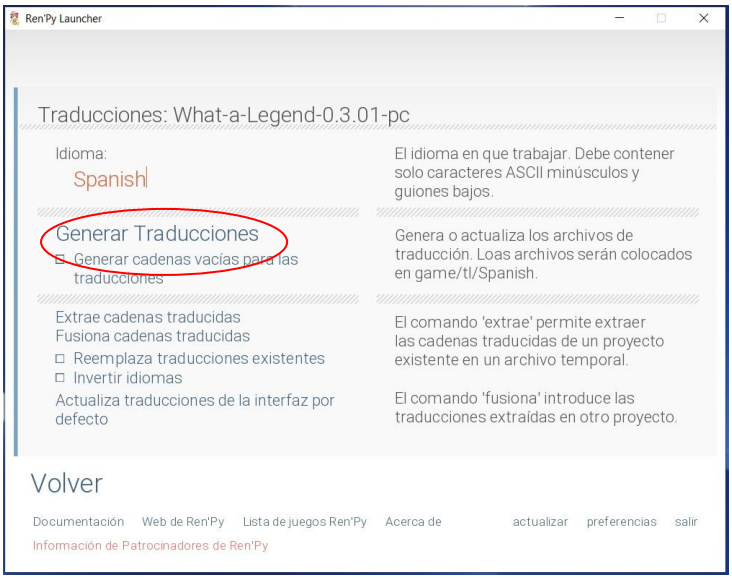
La chaîne _("English") suppose que la langue originale du jeu est l'anglais ; nous devrions écrire la bonne mais en laissant le reste de la ligne tel quel, car Ren'Py considérera toujours cette langue originale comme la langue "None". Et, évidemment, _("Español")devrait être remplacé par le nom de votre langue dans votre propre langue, tandis que la commande d'action Language("") devrait contenir le mot que vous avez utilisé pour générer les scripts de traduction.

Vous avez peut-être remarqué qu'il y a plusieurs chaînes à l'intérieur du symbole _( ), donc Ren'Py SDK va les extraire. Personnellement, je ne traduis que le titre, "Langue", et je laisse les autres non traduites. Pourquoi ? Eh bien, parce que de cette façon, les noms des langues apparaîtront toujours écrits dans leur propre langue, quelle que soit la langue dans laquelle nous jouons. Par exemple, si par hasard un joueur qui ne parle pas notre langue trouve notre traduction et exécute le jeu dans notre langue, en cliquant sur le menu des préférences, il verra rapidement le mot "English" écrit en anglais et comprendra facilement qu'il peut y changer de langue.

Quant aux autres options, la vérité est qu'il y en a autant que de jeux et de traducteurs, je ne vais donc pas les détailler toutes. Il suffit de regarder comment le menu des préférences du jeu est codé (s'il est personnalisé), de faire un peu de rétro-ingénierie, de chercher dans les forums en ligne et d'utiliser les tutoriels de Ren'Py pour créer des écrans, des écrans de veille, des cartes d'images, des boutons... Il y a tout un monde de possibilités, aussi complexes et visuellement attrayantes que vous le souhaitez. |Top|

2.6.- Traduction des mises à jour

En ces temps de Patreon, il est très courant que les jeux ne soient pas publiés en tant que projets achevés, mais en format épisodique, et nous sommes obligés de mettre à jour nos traductions au fur et à mesure que de nouvelles versions sont publiées. La procédure n'a pas de complications majeures... à moins que nous ne voulions la faire comme le suggère le SDK de Ren'Py. L'application officielle de Ren'Py nous offre la possibilité d'extraire les traductions existantes de l'ancienne version d'un jeu et de les insérer dans la nouvelle, en mettant à jour les scripts de traduction avec le nouveau contenu. Sympa, hein ? Eh bien, pas tant que ça. Voyons voir.



Pour ce faire, de manière "officielle", nous lancerions le SDK Ren'Py, nous sélectionnerions dans le dossier Projets la version précédente du jeu que nous voulons traduire (donc, l'ancienne version que nous avons déjà traduite) et, dans l'écran Générer les traductions, nous cliquerions sur "Extraire les traductions des chaînes de caractères".
Une fois ce processus terminé, nous retournons à l'écran de démarrage du SDK de Ren'Py, nous sélectionnons la nouvelle version du jeu (celle que nous voulons traduire maintenant) et, de retour dans cet écran Générer des traductions, nous cliquons sur "Fusionner les traductions de chaînes de caractères".

Théoriquement, à la fin du processus, nous aurions, dans le dossier "game/tl/Spanish" de la nouvelle version du jeu, notre traduction précédente mélangée avec les nouvelles chaînes traduisibles de cette nouvelle version. Le problème est que cette procédure officielle ne fonctionne pas comme elle le devrait, et les traductions ne se confondent tout simplement pas : si un original

n'a pas du tout changé d'une version à l'autre, son script de traduction sera reporté de l'ancienne version à la nouvelle, mais si le script original de la nouvelle version ne correspond pas à l'ancienne, le SDK de Ren'Py générera un script de traduction entièrement nouveau sans aucune trace de la traduction préexistante (même pour les chaînes qui restent inchangées), nous obligeant à répéter le travail de traduction.

Le moyen le plus efficace est donc de télécharger la nouvelle version du jeu, de copier-coller dans son dossier "game" le dossier "game/tl/Spanish" que nous avons dans la version précédente (c'est-à-dire nos anciens scripts de traduction), et de générer les fichiers de traduction de la nouvelle version comme s'il s'agissait d'un jeu entièrement nouveau. Si nous choisissons la MÊME langue que celle de la version précédente (c'est-à-dire si nous écrivons dans la case "langue" du SDK Ren'Py le même mot que nous utilisions auparavant, donc "espagnol", dans mon cas), et que nous ne cochons pas l'option "Remplacer les traductions existantes", le SDK Ren'Py mettra simplement à jour les scripts de traduction existants avec les chaînes pour lesquelles il ne peut pas trouver de traduction valable. À la fin de chaque script de traduction, les nouvelles chaînes traduisibles du jeu (le nouveau contenu) et les chaînes qui ont été modifiées depuis la dernière version traduite seront ajoutées, à nouveau divisées en deux blocs (un premier pour les chaînes du bloc de dialogue et un second pour les options et les menus). Le tri des scripts de traduction en fonction de leur date de dernière modification nous permettra de voir rapidement ceux qui ont un nouveau contenu à traduire.

Cette même procédure est utilisée pour mettre à jour nos scripts de traduction après avoir modifié les scripts originaux pour inclure le symbole _( ) si nécessaire, au cas où Ren'Py SDK n'aurait pas extrait au départ toutes les chaînes traduisibles. Autrement dit, nous modifierions les scripts originaux et nous générerions une nouvelle traduction sans remplacer les scripts existants, puis Ren'Py SDK inclura, à la fin du script de traduction, une nouvelle liste avec toutes les chaînes qui n'avaient pas été détectées auparavant.

|Top|

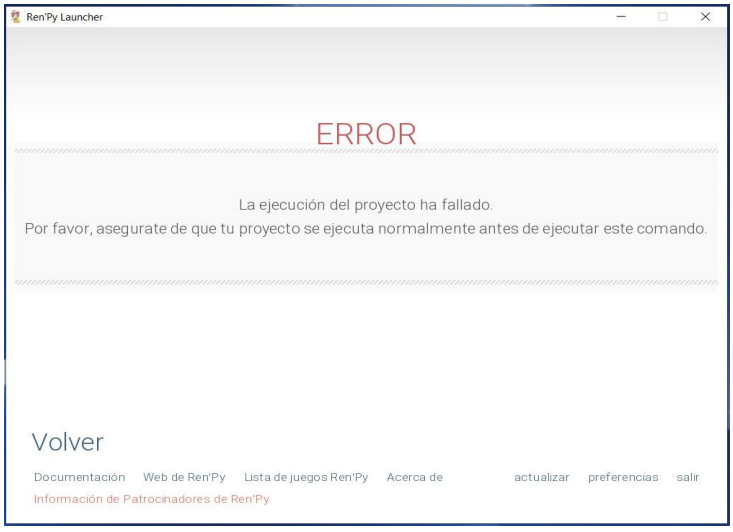
3.- POURQUOI JE NE PEUX PAS LE FAIRE FONCTIONNER

Après vous avoir dit tout cela, si vous avez pu créer votre propre traduction et que tout fonctionne correctement, alors félicitations pour avoir été un élève si attentif, et aussi pour avoir choisi pour vos pratiques un jeu assez simple. Car le plus normal est que, lorsque vous jouez à votre version traduite, de temps en temps, certains textes apparaissent encore non traduits, même si vous les avez traduits dans les scripts de traduction. Ensuite, je vais essayer d'expliquer certains des incidents les plus courants.

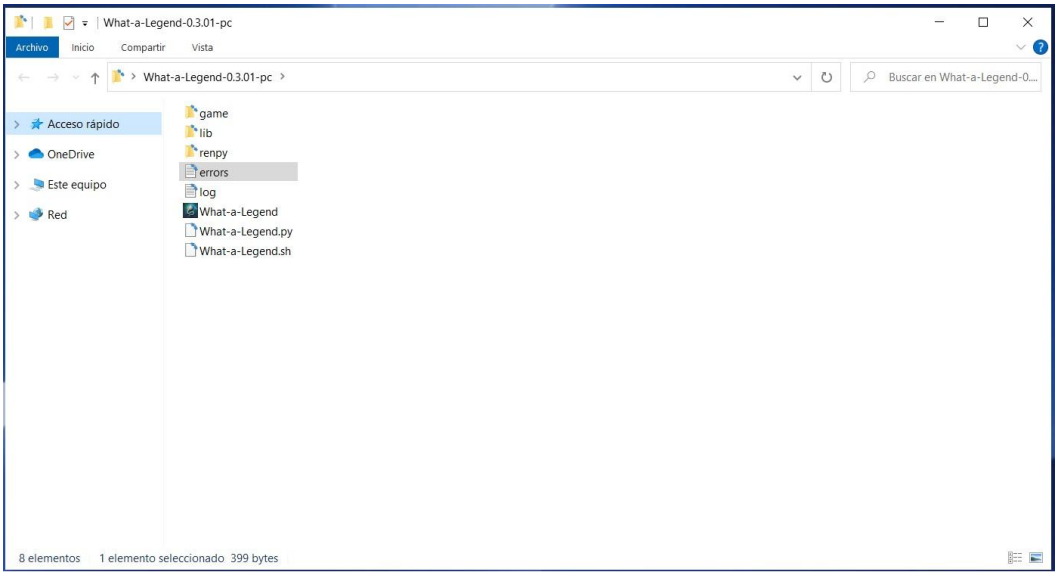
|Top|

3.1.- Détection des bugs et des erreurs

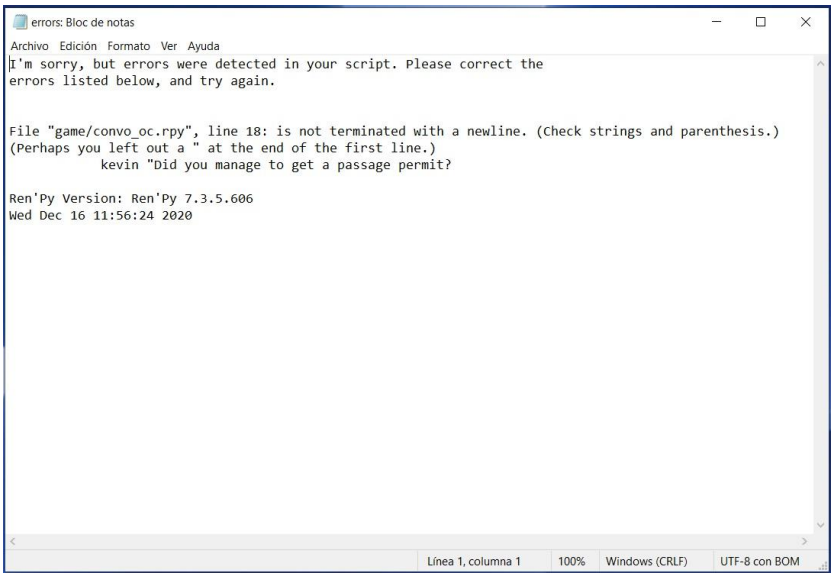
Tout d'abord, nous devrions nous familiariser avec l'écran que Ren'Py utilise pour nous avertir quand quelque chose ne va pas. Dans un monde idéal, les jeux seraient publiés sans bogues et sans erreurs critiques. La première capture d'écran fatale à laquelle un traducteur pourrait être confronté devrait donc être celle qui nous avertit que le SDK de Ren'Py ne pourrait pas générer les scripts de traduction. Cela ne peut que signifier que nous avons commis une ou plusieurs erreurs lors de l'édition des scripts .rpy originaux. Nous avons ici la version espagnole de cet écran :



Afin de voir ce qui s'est exactement passé, nous devrions vérifier le dossier racine du jeu. Là, à côté du fichier .exe du jeu, plusieurs fichiers .txt apparaîtront.

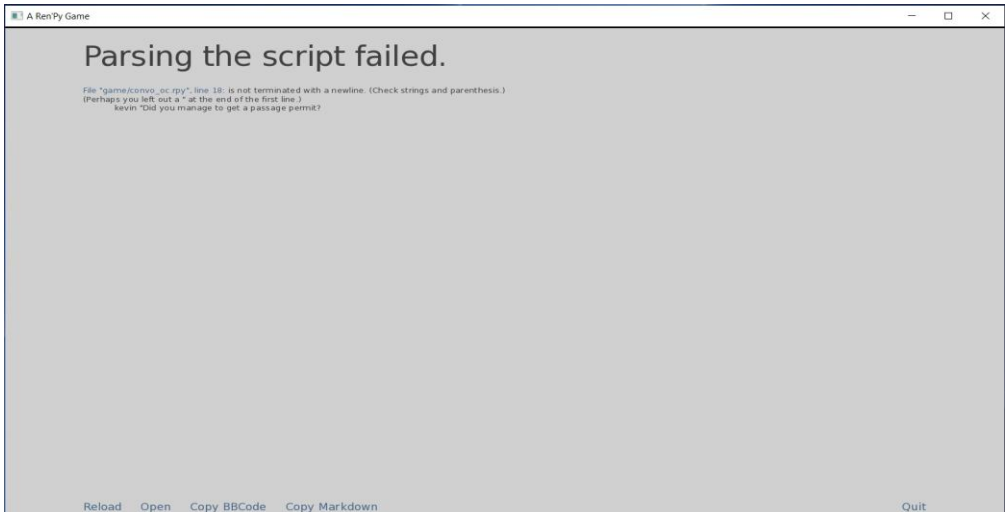


Nous devrions nous concentrer sur celui qui s'appelle "errors.txt", qui peut être ouvert avec n'importe quel éditeur de texte. Là, nous verrons le message d'erreur indiquant la ou les lignes de code qui ont causé le problème.

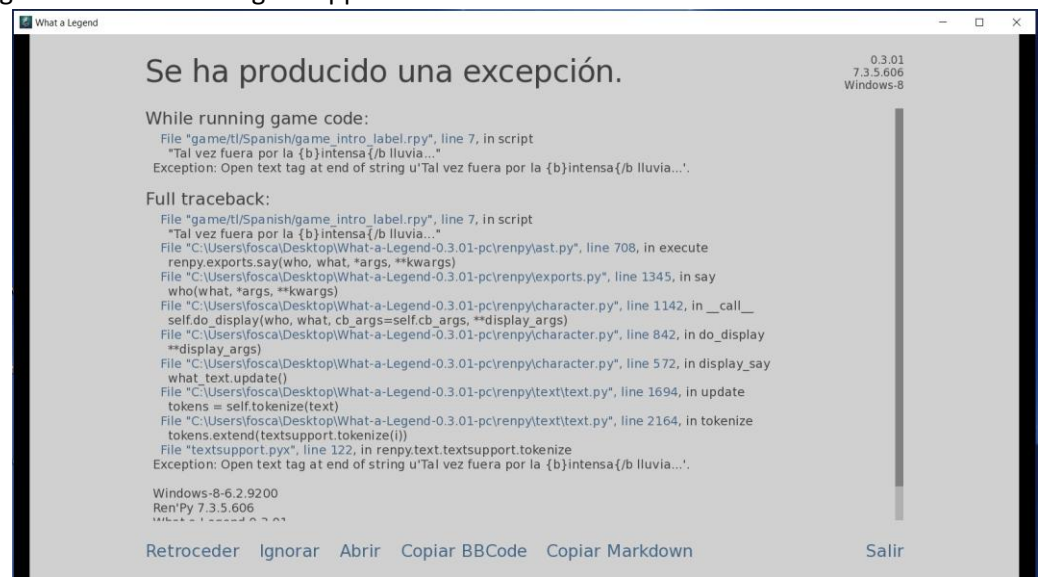


Dans ce cas, c'est quelque chose d'aussi habituel que d'oublier de fermer les guillemets (sur la ligne 18 du script "convo_oc"), mais cela aurait pu être une indentation marquée avec Tab, ou une balise non fermée, ou autre. Il nous suffit d'ouvrir ce fichier, de le réparer, d'enregistrer les modifications et de retourner au SDK de Ren'Py pour réessayer ce que nous faisions. Si vous voyez un message plus complexe, ou s'il apparaît avant même d'avoir modifié quoi que ce soit, le problème vient généralement du fait que vous avez utilisé une version obsolète de UnRen ou de Ren'Py SDK.

Ce message aurait également été affiché si nous avions essayé de lancer le jeu en cliquant sur son exécutable. La capture d'écran serait alors quelque chose comme ça, avec les mêmes informations que le fichier "errors.txt" :



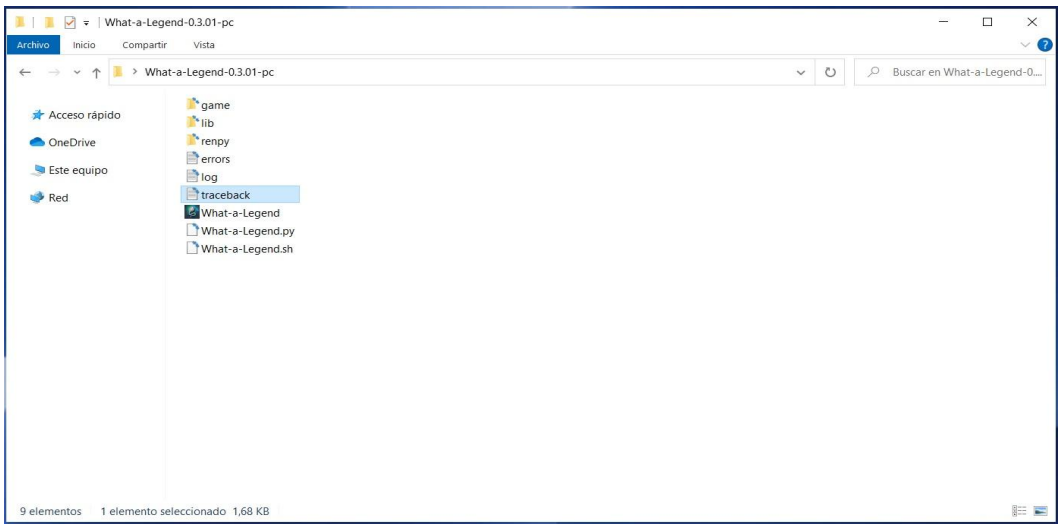
Parfois, l'erreur n'est pas critique dans le sens où elle nous permet de lancer le jeu et de générer les fichiers de traduction, mais elle génère une exception pendant le jeu. Généralement, ces échecs sont dus à une étiquette mal orthographiée ou à un problème avec les variables du jeu. C'est alors que, pendant notre jeu, cet écran gris avec toutes ces lignes apparaît soudainement :



Sur cet écran, nous avons la possibilité d'essayer d'ignorer l'exception et de reprendre le jeu (avec une forte probabilité d'obtenir d'autres écrans d'avertissement et de plantages du jeu). Nous pouvons également revenir à un point antérieur pour sauver notre jeu avant d'essayer de réparer ce qui ne va pas. En outre, nous pouvons copier ce message au format BBCode (ce qui nous permettra de l'insérer dans les forums en ligne) ou le copier en tant que Markdown (que nous pourrions alors joindre à un message de discorde) juste au cas où nous voudrions demander de l'aide.

Cependant, il s'agit presque toujours d'erreurs mineures qui, même si elles ne proviennent pas de notre travail, sont relativement faciles à corriger. Pour les corriger, nous devons toujours prêter attention à la première ligne du message, puisqu'elle indique l'emplacement de la ligne de code qui a causé le problème. Dans ce cas, dans le script de traduction "game_intro_label.rpy" (si vous regardez de plus près, il est dit qu'il se trouve dans le dossier "game/tl/Spanish", et c'est pourquoi je sais que c'est le script de traduction et non pas l'original), ligne 7, il y a une balise non fermée (une balise a été ouverte avec la commande {b} pour mettre un mot en gras, mais elle n'a pas été correctement fermée). Et à la dernière ligne juste avant la section info (où l'on peut trouver le système d'exploitation du lecteur, la version du logiciel et la date et l'heure de l'exception), la cause de l'exception apparaît à nouveau, mais sans aucune référence au script où elle s'est produite. Il suffit de regarder ces deux lignes pour localiser et corriger l'erreur.

Maintenant, en regardant à nouveau le dossier racine du jeu, nous verrions qu'un fichier "traceback.txt" a été généré. Ce fichier contient les mêmes informations que celles que nous avons vues dans l'écran d'exception du jeu. Ce fichier "traceback.txt" et le fichier "errors.txt" que nous avons vu auparavant sont régénérés à chaque fois qu'une exception se produit, en supprimant son contenu préexistant pour n'afficher que le numéro le plus récent.



Dans tous les cas, il s'agit simplement de localiser et de corriger l'erreur, de sauvegarder les scripts et de réessayer ce que nous faisions lorsque l'exception est apparue, en espérant que c'était le seul bogue. Bonne chance.

|Top|

3.2.- Lignes avec trop de texte

En général, les mots et les phrases écrits en anglais sont plus courts que dans la plupart des langues. Cela peut amener certaines traductions à dépasser les limites de la zone de texte que nous voyons à l'écran, ou à obstruer visuellement d'autres éléments de l'interface utilisateur. Le jeu devient difficile ou désagréable à lire et parfois même la fonctionnalité de certains boutons est altérée. Il y a une triple solution à ce problème, qui est assez simple.

Pour commencer, nous pouvons toujours essayer de réécrire notre ligne pour dire la même chose avec moins de caractères. Si le résultat ne nous convainc pas et que nous traduisons la chaîne d'un bloc de dialogue, nous pouvons profiter de l'énorme liberté que Ren'Py nous donne pour écrire le texte traduit. Car, bien que le SDK de Ren'Py ne nous offre qu'une ligne pour traduire la chaîne originale, nous pouvons la diviser en deux lignes ou plus, à condition de respecter les commandements de Ren'Py concernant les citations et les tabulations. Regardez l'exemple :

```
# game/script.rpy:10
translate Spanish label_start_XXXXXXX

    # mc "Hello. My name is John."
    mc "Hola."
    mc "Me llamo John."
```

Ainsi, alors que dans le script original le texte est affiché dans un seul message, dans la traduction espagnole il sera divisé en deux messages, sans autre conséquence. En fait, nous pouvons introduire toutes les variations imaginables, comme l'ajout de fonctions, la modification de variables, etc. En effet, bien que la fonction de traduction soit destinée à remplacer une chaîne de texte par une autre chaîne de texte dans une langue différente, elle remplace en fait une ligne de code entière par ce que nous écrivons dans le script de traduction, de sorte que rien ne peut vraiment nous empêcher de remplacer cette ligne de code (qui se trouve être juste un texte dit par un caractère) par un bloc de code tout à fait différent.

La troisième option, un peu plus complexe, consiste à modifier le script original "gui.rpy", où, entre autres choses, les développeurs du jeu peuvent définir la taille de la police et la largeur de la zone de texte. Pour la taille de la police, nous devons rechercher la commande "define gui.text_size" et réduire le nombre qui apparaît à côté. Et pour augmenter la largeur de la zone de texte, nous augmenterons le nombre qui apparaît à côté de la commande "define gui.dialogue_width". Ces solutions de contournement nous permettront d'écrire plus de caractères dans chaque chaîne traduite et elles fonctionnent généralement bien dans tous les jeux, bien que cela puisse briser l'esthétique du jeu et que vous puissiez trouver quelques difficultés techniques selon le niveau de personnalisation de l'interface du jeu. Mais pour obtenir un résultat satisfaisant, il suffit de faire quelques recherches et de tester vos modifications.

|Top|

3.3.- Les polices qui n'autorisent pas les caractères spéciaux

Légèrement lié au précédent, dans le sens où une des solutions est d'éditer le fichier "gui.rpy", nous avons le problème de ces jeux qui utilisent une police de texte qui ne contient pas certains caractères spéciaux tels que les voyelles accentuées ou tout autre caractère assez spécifique à certaines langues. Par conséquent, lors de l'exécution de la version traduite, les mots apparaîtront à l'écran sans ces lettres ou avec un symbole impair.à.la.place.
La solution la plus rapide consiste à ouvrir le fichier original "gui.rpy" et à rechercher la commande "define gui.text_font" pour remplacer la police utilisée par DejaVuSans, qui ne nécessite aucune action supplémentaire puisqu'elle est intégrée à Ren'Py. Voici à quoi devrait ressembler votre code :


```
define gui.text_font = "DejaVuSans.ttf"
```

Si la police qui ne supporte pas les caractères spéciaux de notre langue est celle utilisée pour les noms des personnages du jeu, ou une autre police spécifique utilisée dans les menus, vous pouvez la retrouver dans le même script et la modifier de la même manière.
Une autre option consiste à définir une police plus adaptée que nous aimons plus et qui possède ces caractères spéciaux. Dans ce cas, en plus d'éditer le fichier "gui.rpy" comme nous venons de le voir (mais évidemment avec le nom de la police choisie au lieu de DejaVuSans), nous devrons inclure dans notre patch le fichier .ttf de ladite police, afin qu'elle reste stockée dans le dossier "jeu".

Malheureusement, en choisissant l'une de ces deux options, le jeu ne peut pas être rejoué avec la police originale, quelle que soit la langue. Nous pouvons donc choisir une solution intermédiaire, un peu plus complexe, qui consiste à définir une police spécifique pour notre langue, de sorte que la version originale du jeu s'affiche exactement comme son développeur l'avait prévu, même après avoir appliqué notre patch de traduction. Dans le fichier original "gui.rpy", nous devrions écrire ceci :

```
init python:
    translate_font("Spanish", "myfont.ttf")
```

Où "myfont.ttf" est la police que nous avons choisie pour notre langue (et cette langue étant "l'espagnol" dans mon cas). S'il ne s'agit pas d'une des polices standard supportées par Ren'Py, nous devrons inclure le fichier .ttf dans notre patch de sorte que cette fois-ci, il soit stocké dans le dossier "game/tl/Spanish" et non dans le dossier "game".

Dans tous les cas mentionnés ci-dessus, nous devons nous souvenir d'inclure le fichier "gui.rpy" édité dans notre patch, afin qu'il remplace celui que les joueurs qui ont téléchargé le jeu original ont dans leur dossier "game".

Et enfin, nous pourrions aussi faire un all-in et modifier la police originale avec un logiciel d'édition de police, pour concevoir les caractères manquants. Évidemment, dans ce cas, nous devrions également inclure le fichier .ttf résultant dans notre patch pour remplacer le fichier original dans le dossier "game", mais nous n'aurions pas besoin de modifier le fichier "gui.rpy" du tout.

|Top|

3.4.- Traduire les images

L'une des tâches les plus longues à réaliser est la traduction des images qui contiennent du texte. Parfois, les développeurs du jeu décident qu'une partie importante de l'action doit être basée sur un SMS, par exemple, ou tout autre élément visuel (une capture d'écran d'ordinateur, un titre de journal, une affiche sur le mur ou tout ce que vous pouvez imaginer). Bien que Ren'Py propose plusieurs solutions de codage pour écrire ces textes dans les scripts et, par conséquent, en faciliter la traduction, les développeurs trouvent généralement plus pratique de créer simplement une image avec le texte qui y est inclus.

Si nous avons la chance que, pendant que l'image est affichée, il y ait aussi un certain dialogue, dans notre traduction, nous pouvons simplement ajouter le texte aux mots du personnage, comme s'il le lisait à voix haute.

Nous pouvons aussi "sous-titrer" l'image grâce à une solution que nous avons utilisée pour faire tenir les chaînes traduites dans la zone de texte : nous pouvons diviser la traduction de cette chaîne en plusieurs lignes et y écrire la traduction du message de l'image, peut-être en italique ou d'une autre manière qui aide les joueurs à savoir ce qui s'y passe.

```
# game/script.rpy:12
translate Spanish label_start_XXXXXXX

# mc "Look. He sent me a message."
mc "Mira. Me envió un mensaje."
"{i}(And here we translate the message that is displayed on the image.){/i}"
```

Si, malheureusement, nous n'avons pas de lignes de dialogue associées à l'image, nous pouvons en créer une vierge dans le script original. Pour ce faire, la première étape consiste à ouvrir le script original, à trouver la ligne de code dans laquelle nous pensons que Ren'Py est ordonné pour montrer l'image (nous pouvons utiliser des chaînes de texte proches pour la trouver) et ensuite, en respectant l'indentation, nous insérerions une nouvelle ligne en dessous, avec une chaîne vide (en n'écrivant que des guillemets), comme ceci :

```
scene black
show sms00 #This is the command used to show the image that contains text
"" # This is the new blank string we insert in the script, to translate the pic
pause
hide sms00.jpeg
mc "Wow."
```

Ensuite, nous (re)générons les fichiers de traduction et nous pouvons maintenant inclure la traduction du texte de l'image comme traduction de cette nouvelle chaîne vide. Bien entendu, pour que la traduction s'affiche correctement, nous devons inclure dans notre patch le script original que nous venons de modifier.

Cependant, il arrive que cette option ne soit tout simplement pas viable ou n'aboutisse pas à un bon résultat esthétique. Dans ce cas, la seule solution consiste à modifier l'image originale (ou à en créer une nouvelle) avec un logiciel de retouche d'images comme Photoshop ou GIMP (ou même MS Paint, s'il s'agit d'une retouche très simple). Tout d'abord, on se reporterait au script original pour trouver le nom de l'image (généralement celui qui apparaît après les commandes de la série ou de la scène). Ensuite, nous cherchons dans le dossier "game/images" et nous l'éditons pour écrire son texte dans notre langue. Dans un monde idéal, nous pourrions demander au développeur du jeu de nous fournir l'image de base, sans texte, pour faciliter ce travail d'édition. Bonne chance.

Une fois l'édition terminée, nous ne devons pas remplacer l'image originale dans ce dossier "*jeu/images*" ; nous devons plutôt enregistrer une copie avec nos modifications, avec le même nom et le même format d'image, dans un chemin identique mais à l'intérieur de notre dossier de traduction (dans mon cas, à l'intérieur de "*tl/Espagnol*"). Ainsi, si l'image originale est enregistrée sous "*game/images/ch1/sms00.jpeg*", nous DEVONS enregistrer l'image traduite sous "*game/tl/Spanish/images/ch1/sms00.jpeg*". De cette façon, lorsque nous jouons à la version traduite du jeu et que l'image nommée "*sms00.jpeg*" doit être affichée, Ren'Py la cherchera d'abord dans le sous-dossier "images" du dossier de traduction, et seulement si l'image n'y est pas trouvée, elle affichera celle du dossier "*game/images*" d'origine. Et, logiquement, si nous jouons dans la langue originale du jeu, l'image originale sera affichée.

|Top|

3.5.- Traduction des variables de texte

Lorsque nous avons parlé des limites du SDK de Ren'Py et de ses difficultés à extraire toutes les chaînes traduisibles, nous avons dit que les variables qui contiennent du texte comme valeur ne sont pas automatiquement détectées par la fonction d'extraction. Quelle que soit la façon dont nous parvenons finalement à les inclure dans les scripts de traduction, le problème est que, généralement, cette traduction ne sera pas affichée plus tard à l'écran, alors qu'elle devrait l'être.

Lorsqu'une variable de texte apparaît à l'écran, elle a été codée comme un élément intégré dans une chaîne de caractères. Nous le remarquerons dans les scripts car, dans la chaîne, un objet portant le nom de la variable apparaîtra entre [crochets]. Parfois, la chaîne est constituée exclusivement de cet objet, et parfois, elle ne constitue qu'une partie d'une phrase. Par exemple, dans l'un des scripts originaux du jeu "City of Broken Dreamers", on trouve ceci :
Dans ce cas, le développeur du jeu a décidé qu'un certain personnage utilisera une insulte qui variera de manière aléatoire dans chaque jeu. Pour ce faire, dans un autre script (parfois nous devons faire beaucoup de recherches), le développeur a défini une variable qui est en fait une liste de différentes insultes.

```
56      show bg ch1sonja3 with dissolve
57      $ insult = renpy.random.choice(insults)
58      "Unknown Woman" "Come on now, [insult]."
```

```
139 default insult = ""
140 default insults = ["jackoff","fuck-face","cock muncher","retard","cumb dunt","dick mitten","fuck-knuckle"]
```

La fonction de la ligne 57 sélectionnera aléatoirement une insulte dans la liste que nous voyons à la ligne 140, et l'insulte choisie sera la valeur de la variable nommée "insulte" (qui a été définie comme vide à la ligne 139). Ensuite, à la ligne 58, le personnage "Femme inconnue" utilisera cette insulte en parlant à notre MC.

Par conséquent, notre première tâche serait de traduire la liste des insultes de la ligne 140, qui contient en fait plusieurs chaînes de caractères, une pour chaque insulte citée. En faisant appel à notre imagination, nous traduisons chacune d'elles avec les anciennes et les nouvelles commandes et, dans la chaîne de traduction du bloc de dialogue, nous ajoutons l'appendice !t à l'objet portant le nom de la variable :

```
571  # game/chapter1/ch1sonja.rpy:58
572  translate Spanish ch1sonja_cfc9ab60:
573
574      # "Unknown Woman" "Come on now, [insult]."
575      "Desconocida" "Venga ya, [insult!t]."
```

Ainsi, lorsque l'on joue dans la langue originale du jeu, la variable sera affichée dans cette langue, puisque le script original n'a pas été modifié, tandis que la version traduite affichera le contenu traduit. Si nous ne traduisons que les insultes sans inclure l'appendice !t dans cette autre chaîne, l'insulte apparaîtra toujours en anglais lorsque nous jouerons au jeu traduit, car notre chaîne traduite continuera à dire à Ren'Py d'afficher la variable originale [insult] au lieu de sa valeur traduite [insult!t].

Notez que cela ne concerne que les variables textuelles réelles, et non celles qui sont utilisées pour nommer les personnages du jeu. Parfois, pour faciliter l'écriture des chaînes de caractères, les développeurs de jeux remplacent les noms complets de certains personnages par la définition de leur variable "caractère" respective. Nous le remarquerons car la variable que nous voyons intégrée dans la chaîne est la même que celle que nous voyons au début des lignes de dialogue desdits caractères, avant les guillemets. Dans ce cas, la traduction de cette variable (si nécessaire) est faite automatiquement par Ren'Py et nous n'avons pas besoin de faire autre chose. Par exemple, il y a plusieurs personnages dans le jeu "Deliverance" qui sont connus par un nom commun, traduisible, et non par leur vrai prénom.

```
8  define li = Character("Lieutenant", ctc="ctc_default",ctc_pause="ctc_default") #fowler
```

Dans ce cas, la variable nommée [li] fait référence à l'un d'entre eux (Lieutenant, car il est chef de police), mais comme il s'agit d'une variable "caractère", Ren'Py affichera automatiquement sa valeur originale ou sa traduction disponible, selon la version du jeu à laquelle nous jouons. Il nous suffit donc de traduire "Lieutenant" en utilisant les anciennes et les nouvelles commandes et, lorsque cette variable apparaît dans une chaîne de traduction, il n'est pas nécessaire d'ajouter l'appendice !t.

```
177  # game/script.rpy:350
178  translate Spanish interrogation_3864c01a:
179
180      # co "[li], you have a visitor. Mayor wants to see you."
181      co "[li], tiene una visita. El alcalde quiere verle."
```

Avec ces notions, nous ne devrions avoir aucun problème à faire en sorte que tout le texte traduisible du jeu soit correctement affiché dans notre langue.

Ce qui suit est une solution d'urgence qui ne doit être utilisée qu'en dernier recours.

Il existe des jeux assez complexes, et parfois nous ne pourrons pas voir le texte correctement traduit parce que nous ne savons pas comment localiser la chaîne qui contient l'objet avec la variable (cela peut arriver surtout dans les écrans de menu personnalisés, les inventaires, etc.) La solution d'urgence consiste à modifier les scripts originaux et à entourer la définition de cette variable de parenthèses, en ajoutant un double trait de soulignement avant le signe d'ouverture :

`define` *variable_name* `= __("...").`

Cela permettra, d'une part, au SDK de Ren'Py d'extraire la chaîne traduisible et, d'autre part, en utilisant le double soulignement, notre traduction sera toujours affichée à l'écran... même lorsque l'on joue au jeu dans sa langue d'origine. C'est dommage, mais parfois nous ne voyons tout simplement pas d'autre option. Bien sûr, il est possible qu'en faisant cela, nous générions certains problèmes de logique interne au jeu, car la valeur de cette variable peut être utilisée dans certaines expressions logiques qui déterminent les chemins du jeu ou le contenu à bloquer ou à montrer au joueur. Dans ces cas, pour éviter les bugs et autres incohérences, nous devons inclure la variable entre les symboles __( ) également dans ces expressions.

Par exemple, imaginons que nous ayons une variable appelée "fruit" dont la valeur originale est "Pomme". Quelque part dans le code, nous devrions trouver sa définition :

```
default fruit = "Apple"
```

Plus tard, pendant le jeu, le joueur peut changer ce fruit pour un autre, donc dans le scénario on pourrait trouver quelque chose comme ça :

```
$ fruit = "Orange"
```

Imaginons que nous ayons trouvé ces deux chaînes de caractères et que nous les ayons traduites avec les anciennes et les nouvelles commandes dans nos scripts de traduction :

```
translate Spanish strings:

    old "Apple"
    new "Manzana"

    old "Orange"
    new "Naranja"
```

Mais dans le jeu, il y a un écran d'inventaire qui montre une liste des différents objets en possession du joueur, l'un d'eux étant la variable [fruit], et pour une raison quelconque, nous ne pouvons pas trouver dans les scripts originaux les lignes de codage de cet écran, donc nous ne pouvons pas extraire cette chaîne spécifique [fruit] et la traduire par [fruit!t] dans le script de traduction. Cela signifie que, même en jouant à la version traduite du jeu, le texte affiché sur cet écran d'inventaire sera "Apple" ou "Orange", toujours en anglais. La solution de la force brute serait d'extraire avec le double soulignement les chaînes que nous avons localisées :

```
default fruit = __("Apple")
$ fruit = __("Orange")
```

Ainsi, chaque fois que la variable [fruit] est mentionnée dans les scripts originaux, la valeur qui sera affichée à l'écran sera notre traduction pour ces mots, même si elle n'est pas optimale lorsqu'elle est jouée dans la langue originale. Mais il peut arriver que cette variable "fruit" soit utilisée à un moment donné pour afficher une certaine ligne de dialogue ou une ligne différente, en fonction du fruit que le joueur possède à ce moment. Quelque chose comme ceci :

```
if fruit == "Apple"
    mc "I like red apples."
if fruit == "Orange"


    mc "I like orange juice."
```

Si nous laissons les scripts originaux comme cela, le jeu s'effondrerait en atteignant cette partie du script car la valeur de la variable "fruit" serait notre traduction pour les mots "Apple" ou "Orange", et non "Apple" ou "Orange" en anglais, qui est la façon dont l'expression est codée à l'origine. Il se peut que cela ne provoque pas d'exception critique, et selon le reste du code, le joueur peut même ne rien remarquer (ou peut-être juste un petit saut ou une incohérence dans les dialogues), mais ce n'est bien sûr pas ce que voulait le développeur du jeu. Pour résoudre ce problème, nous devrions également utiliser les symboles __( ) dans l'expression logique :

```
if fruit == __("Apple")
    mc "I like red apples."
if fruit == __("Orange")
    mc "I like orange juice."
```

Si nous avions réussi à traduire correctement la fameuse chaîne [de fruits], toutes ces modifications ne seraient pas nécessaires, puisque l'expression logique fonctionnerait en interne avec sa valeur originale (dans la langue d'origine du jeu) même si la traduction était affichée à l'écran.

3.5.- Polisemy : des traductions différentes pour des mots égaux

Cette question a été soulevée en parlant des deux fonctions de traduction utilisées par Ren'Py. Les chaînes ne supportent qu'une seule traduction : en fait, si nous essayons de les traduire deux fois (en écrivant deux anciennes commandes avec la même chaîne, par exemple), le jeu ne démarrera tout simplement pas et le message d'erreur indiquera qu'il y a une traduction en double. Pour pouvoir démarrer le jeu, nous devrons supprimer l'une d'entre elles.

Mais, comme je l'ai dit dans cet exemple, nous trouvons parfois des chaînes identiques qui nécessitent des traductions différentes, comme le mot anglais "Right" qui, entre autres choses, pourrait signifier "Correct" ou juste un côté ou une direction opposée à "left". La solution consiste à modifier le script original pour que ces chaînes identiques cessent d'être identiques, sans affecter le jeu dans la langue d'origine. Et pour cela, il suffit d'utiliser le symbole #.

Comme nous l'avons déjà dit, tout ce qui se trouve à droite d'un symbole # n'apparaîtra pas à l'écran et est écarté par Ren'Py dans tous ses processus... à moins que nous ne le placions dans une étiquette. Les balises sont des commandes intégrées dans les chaînes de caractères avec des symboles { }, et sont généralement utilisées pour changer la taille et la couleur de la police, pour mettre en évidence la phrase en gras ou en italique, etc. Ren'Py lit ces balises comme faisant partie de la chaîne qui les contient et exécute ce qu'elles commandent, mais leur contenu littéral n'est pas affiché à l'écran. Donc, si dans l'une de ces balises nous incluons un texte après le symbole #, la balise n'ordonnera pas à Ren'Py de faire quoi que ce soit et Ren'Py n'en affichera pas le contenu, mais nous aurons alors une chaîne différente de celle qui n'a pas de balise, et les deux auront l'air identique à l'écran.

Par conséquent, nous trouverons la chaîne que nous voulons traduire différemment et nous y incorporerons une balise dans laquelle nous écrirons quelque chose qui nous permettra de l'identifier plus tard, comme la traduction que nous voulons appliquer. Ainsi, lors de la lecture et de l'extraction des chaînes de caractères, le texte "Right" et le texte "Right{#Direction}" ne sont plus les mêmes pour Ren'Py, bien qu'ils soient tous deux affichés comme "Right" à l'écran. Il suffit ensuite de régénérer les scripts de traduction (ou d'y inclure manuellement cette nouvelle chaîne "balisée", avec l'ancienne et lui attribuer une traduction différente avec la nouvelle commande. A terme, nous devrions avoir

```
old "Right"
new "Correcto"
```

quelque chose comme ceci:

```
old "Right{#Direction}"
new "Derechatranslate Spanish strings:"
```

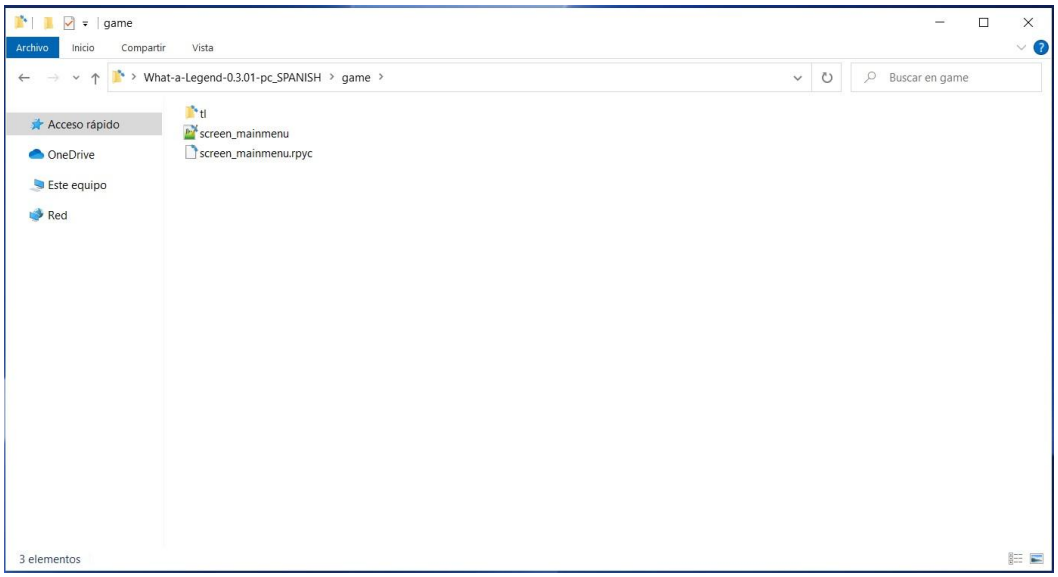
N'oubliez pas que, dans ce cas, il ne suffit pas d'écrire cela dans le script de traduction : nous devons également modifier le script .rpy original pour intégrer la balise dans la chaîne que nous voulons traduire différemment, de sorte que, lors de la lecture de la version traduite, Ren'Py recherchera la traduction de la chaîne marquée. Et, évidemment, nous devons nous souvenir d'inclure les scripts modifiés dans le patch.

4.- LE PATCH DE TRADUCTION

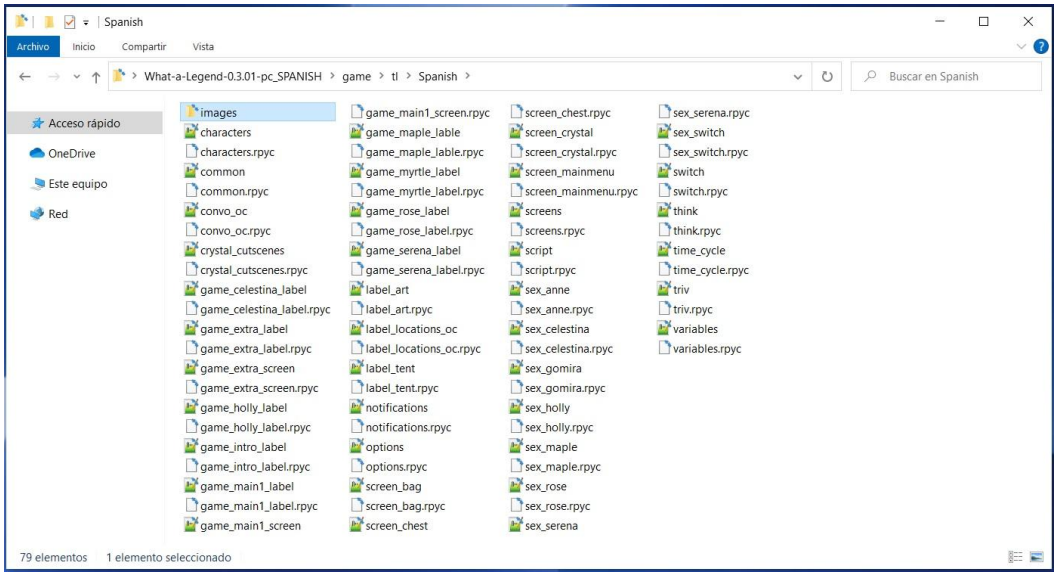
Une fois que notre traduction est faite (et testée), il est temps de la partager avec le reste du monde. Mais, pour fonctionner, notre patch doit respecter la structure des dossiers du jeu original. C'est-à-dire que les scripts .rpy originaux que nous avons édités (ainsi que leurs scripts .rpyc respectifs) et les fichiers .ttf des polices utilisées pour résoudre les problèmes mentionnés au point 3.3 doivent se retrouver dans le dossier "jeu" du joueur, qui doit remplacer les fichiers originaux par ceux de notre patch. Et, évidemment, nous devons également inclure un dossier "tl", et à l'intérieur de celui-ci un autre sous-dossier (appelé "espagnol" dans mon cas, car c'est la langue que j'ai écrite dans le SDK de Ren'Py) qui contiendra nos scripts de traduction et, le cas échéant, les images et les polices nécessaires, comme cela a été expliqué plus haut dans le guide.

Le moyen le plus simple (pour moi, du moins) est de créer un dossier "jeu" avec tous ces fichiers, de sorte que les joueurs n'auront qu'à le coller dans le répertoire racine de leur jeu original et accepter de remplacer et/ou d'écraser tous les fichiers correspondants.

Voici, par exemple, à quoi ressemble le dossier "jeu" de mon patch de traduction "What a Legend !



Dans ce patch, il m'a suffi d'inclure le script "*screen_mainmenu.rpy*", où j'ai ajouté l'option de changement de langue. Ce fichier remplacera le script original "*screen_mainmenu.rpy*" qui existe dans le dossier "*jeu*" du joueur. Logiquement, il y a aussi tout le dossier "*tl*" que j'avais dans mon ordinateur, avec le sous-dossier "*Spanish*" et ma traduction complète à l'intérieur, qui ressemble à ceci :



Vous pouvez y voir tous les scripts de traduction et aussi un sous-dossier nommé "*images*" avec les images du jeu que j'ai dû traduire, qui sont également stockées dans des dossiers du même nom que ceux contenant les images originales dans le dossier "*jeu/images*", comme expliqué au point 3.4.

Donc, pour que les joueurs puissent profiter de tout notre travail, nous devons simplement inclure dans notre patch de traduction les fichiers traduits ainsi que tous les scripts originaux qui ont été modifiés pour autre chose que l'extraction des chaînes avec les symboles _( ) (rappelez-vous que Ren'Py n'a pas besoin de ce symbole pour afficher la chaîne traduite, une fois que cette chaîne est présente dans les scripts de traduction avec les anciennes et les nouvelles commandes). C'est-à-dire : les scripts .rpy que nous avons édités pour permettre la traduction correcte des mots polysémiques, les fichiers "gui.rpy" et/ou "*screens.rpy*" édités pour changer la langue ou la police du jeu (ainsi que les fichiers .tff de ces polices), et leurs fichiers .rpyc respectifs, tout cela sera dans le patch de traduction.

Cependant, mes correctifs incluent généralement absolument tous les scripts originaux que j'ai édités, de sorte que d'autres traducteurs peuvent les utiliser pour obtenir toutes les chaînes traduisibles extraites par leur SDK Ren'Py. Mais "Quelle légende !"

Les développeurs écrivent déjà leur code avec ces symboles _( ) là où c'est nécessaire, donc dans ce cas il n'y avait rien d'autre à ajouter.

Enfin, un dernier commentaire. Je recommande toujours d'inclure les scripts .rpyc dans le patch (c'est-à-dire ceux qui correspondent aux scripts .rpy originaux que nous avons édités, et qui resteront dans le dossier "jeu" du joueur). Si nous ne les avons jamais supprimés depuis que nous avons téléchargé le jeu original (ou depuis que nous les avons extraits avec UnRen afin de pouvoir générer les scripts de traduction), ces scripts .rpyc que nous avons dans notre ordinateur respecteront l'AST créée lorsqu'ils ont été compilés à l'origine par le développeur du jeu, même s'ils ont été modifiés par nous lors de l'édition de leurs scripts .rpy respectifs. Ainsi, lorsque les joueurs téléchargent notre patch et remplacent les fichiers .rpyc qu'ils ont dans leur ordinateur (qui, s'ils ne les ont jamais supprimés, sont également les mêmes que ceux que nous avions au début de notre processus de traduction), la structure de base des nouveaux scripts .rpyc restera la même et ils ne devraient pas avoir de problèmes pour charger les jeux qu'ils avaient sauvegardés lorsqu'ils jouaient à la version originale du jeu avant d'appliquer notre patch (de toute façon, il est toujours conseillé de charger d'abord ces anciennes sauvegardes dans la langue originale du jeu, puis de changer de langue pour continuer à jouer à partir de ce point de sauvegarde).

Que se passerait-il si nous n'incluons pas les scripts .rpyc dans le patch de traduction ? Cela dépend. En général, si le jeu original est conditionné selon les recommandations de Ren'Py (avec les scripts .rpy et .rpyc exposés dans le dossier "jeu", comme c'était le cas pour "Quelle légende"), il ne devrait y avoir aucun problème : nos scripts .rpy modifiés remplaceront les scripts originaux que les joueurs ont sur leur ordinateur et, au lancement du jeu, leurs fichiers .rpyc seront mis à jour avec les changements inclus dans nos scripts .rpy. Ce processus ne devrait rien casser... en supposant que le joueur n'ait jamais supprimé les scripts .rpyc originaux du jeu (s'ils ont été supprimés par le joueur avant l'installation de notre patch, nous ne pouvons pas être responsables des erreurs qui pourraient survenir en raison d'un changement d'AST lors du chargement d'anciennes sauvegardes).

Le problème vraiment important peut se poser lorsque les scripts du jeu original sont compressés dans une archive .rpa : si nous n'incluons dans notre patch que les scripts .rpy modifiés, Ren'Py créera de nouveaux fichiers .rpyc sur l'ordinateur du joueur, et il y a un risque qu'ils ne soient pas générés avec la même structure AST que ceux que nous avons extraits du jeu original, qui sont ceux que nous avons utilisés pour créer notre traduction (et qui sont aussi ceux que les joueurs ont encore compressés dans leur fichier .rpa). Dans ce cas, il n'y a pas que les anciennes sauvegardes qui peuvent être compromises : il peut arriver que la fonction qui exécute la traduction du "code de cryptage" (chaînes du bloc de dialogue) ne détecte pas la traduction existante. Dans ce cas, les menus et options du jeu seraient affichés dans notre langue (car ces chaînes sont traduites par remplacement direct grâce aux anciennes et nouvelles commandes) mais les dialogues de ces scripts édités seraient affichés non traduits, dans la langue originale du jeu. C'est une erreur assez étrange mais qui peut se produire, surtout avec des jeux relativement anciens qui, peut-être, ont été compilés à l'origine avec une ancienne version du SDK de Ren'Py. Quoi qu'il en soit, nous éviterons cette situation en incluant dans notre patch les scripts .rpy et .rpyc : bien qu'édités et mis à jour, ils suivent toujours la structure AST des scripts originaux que nous avons extraits avec UnRen, et cette structure correspondra également à l'ancienne structure de sauvegarde du joueur. Mieux vaut prévenir que guérir.

|Top|