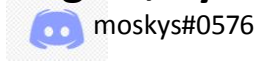




Traducción de juegos Ren'Py: Una guía, by moskys



¡Hola!

Llamadme moskys. Como seguramente sepáis, si habéis llegado hasta esta guía, Ren'Py es un motor gratuito y de código abierto para crear juegos (tipo novela visual, generalmente) que se ha convertido en uno de los más utilizados por desarrolladores amateurs para realizar sus proyectos. Basado en el lenguaje de programación Python, su sencillez y flexibilidad es una gran ventaja, ya que permite obtener resultados más que aceptables sin tener conocimientos previos de programación. Y dentro de las muchas funcionalidades que incluye Ren'Py está la de facilitar la traducción de los juegos creados con este programa.

Así que, después de unos tres años jugando y traduciendo manualmente al castellano (y para PC) varios juegos de diversa complejidad, me he animado a escribir esta guía para explicar el proceso a todo aquel que quiera empezar a traducir o tenga curiosidad por saber a qué dedico mis ratos libres.

1.- INTRODUCCIÓN:

- [1.1.- Destripando un juego Ren'Py](#)
- [1.2.- Tres conceptos básicos y dos mandamientos](#)
- [1.3.- ¿Cómo \(creo que\) funciona Ren'Py? El tercer mandamiento](#)
- [1.4.- UnRen y archivos .rpa](#)
- [1.4.- Ren'Py SDK](#)

2. A TRADUCIR:

- [2.1.- Generando los archivos de traducción \(y aprendiendo el cuarto mandamiento\)](#)
- [2.2.- Las dos funciones de traducción \(y otro mandamiento más\)](#)
- [2.3.- Ayudando y/o sustituyendo al extractor](#)
- [2.4.- La opción de cambio de idioma](#)
- [2.5.- Traduciendo actualizaciones](#)

3.- Y ESTO POR QUÉ NO ME SALE

- [3.1.- Detección de errores y bugs](#)
- [3.2.- Líneas con exceso de texto](#)
- [3.3.- Fuentes que no admiten caracteres especiales](#)
- [3.4.- Traducción de imágenes](#)
- [3.5.- Traducción de variables de texto](#)
- [3.6.- Polisemia: Traducciones distintas para palabras iguales](#)

4.- EL PARCHE

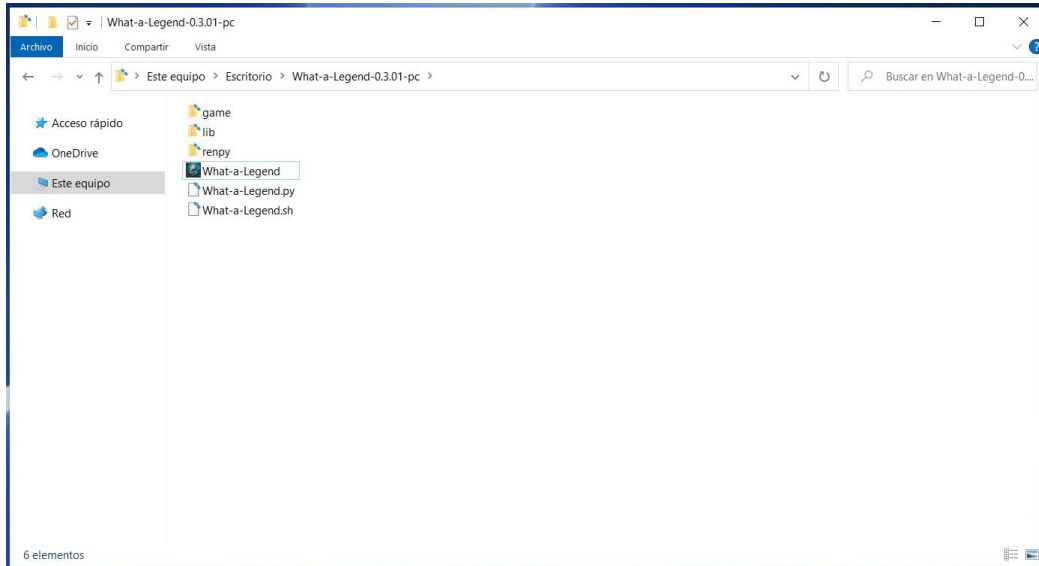
HERRAMIENTAS IMPRESCINDIBLES (Y GRATUITAS) – versiones a 31/12/2020

- **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/> (*link a un foro con contenido para adultos*)
- **Editor de textos:**
 - **Notepad++ 7.9.1** -> <https://notepad-plus-plus.org/downloads/>
 - **Atom** -> <https://atom.io/>

1.- INTRODUCCIÓN

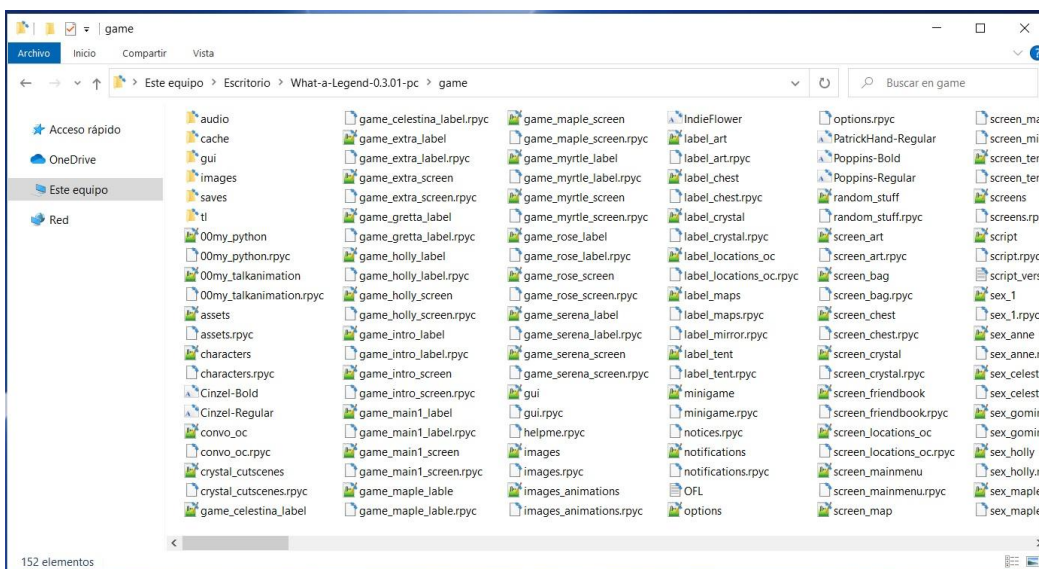
1.1.- Destripando un juego Ren'Py

Es obvio, pero, para poder traducir un juego, primero hay que tener un juego. Así que buscad cualquier juego Ren'Py que tengáis descargado y mirad lo que hay en su carpeta raíz. Algo así:



Normalmente, lo que hacemos es abrir el archivo ejecutable y jugar. Pero ahora vamos a fijarnos en las tres subcarpetas que vemos ahí, que es donde se produce la magia. A grandes rasgos, en la carpeta “renpy” se encuentran los archivos que contienen las funciones preprogramadas, y en la carpeta “lib” todos los archivos técnicos que hacen que los juegos se ejecuten en los diversos sistemas operativos. Estas dos carpetas son las que nos permiten jugar sin habernos descargado un programa específico para ello; es decir, las que convierten a los juegos en programas autoejecutables.

En la carpeta “game” es donde se encuentra el juego en sí. No os asustéis, este es especialmente complejo.



Hay varias subcarpetas que de momento podemos pasar por alto para fijarnos en los archivos sueltos. Ren'Py recomienda a los creadores que, a la hora de empaquetar sus juegos, dejen los archivos a la vista, como hacen los desarrolladores de “What a Legend!” Esto nos permite ver toda una serie de archivos aparentemente duplicados, unos con formato .rpy y otros con formato .rpyc.

[\[Arriba\]](#)

1.2.- Tres conceptos básicos y dos mandamientos

Estos archivos de la carpeta “game” son los **scripts**. Un script es, en traducción literal, un guion. Aquí es donde el desarrollador incluye la programación y los textos del juego. Para ello escribe lo que tenga que escribir en un procesador de texto y guarda el archivo con formato .rpy, específico de Ren'Py.

Así que, usando un programa básico como el Bloc de Notas de Windows (o uno mejor, como [Atom](#) o [Notepad++](#), por citar dos gratuitos y sencillos usados en programación), podemos abrir esos archivos .rpy y ver en qué consiste esto de crear un juego Ren'Py. En el ejemplo de “What a Legend!”, si abrimos el archivo “convo_oc.rpy” vemos algo así en sus primeras líneas.

```

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 hfneul hbneul
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 hfneul
26            kevin "...but no permit, no passage. That is the law."

```

Todo lo que quede a la derecha de un símbolo # son comentarios que no aparecerán en el juego pero que sirven al creador para orientarse en su código. [Más adelante](#) veremos la utilidad que tiene ese símbolo para los traductores.

En la línea 3 vemos una palabra, **label**, que va a tener su importancia. Las labels, o etiquetas, son porciones del guion. El tamaño de estas porciones depende del desarrollador del juego, pero generalmente corresponden a una escena concreta. En este caso, a una conversación que aparecerá en pantalla cuando hagamos alguna acción que la active. Por el lenguaje Python, todo lo que ocurre en esta escena se codifica con una sangría.

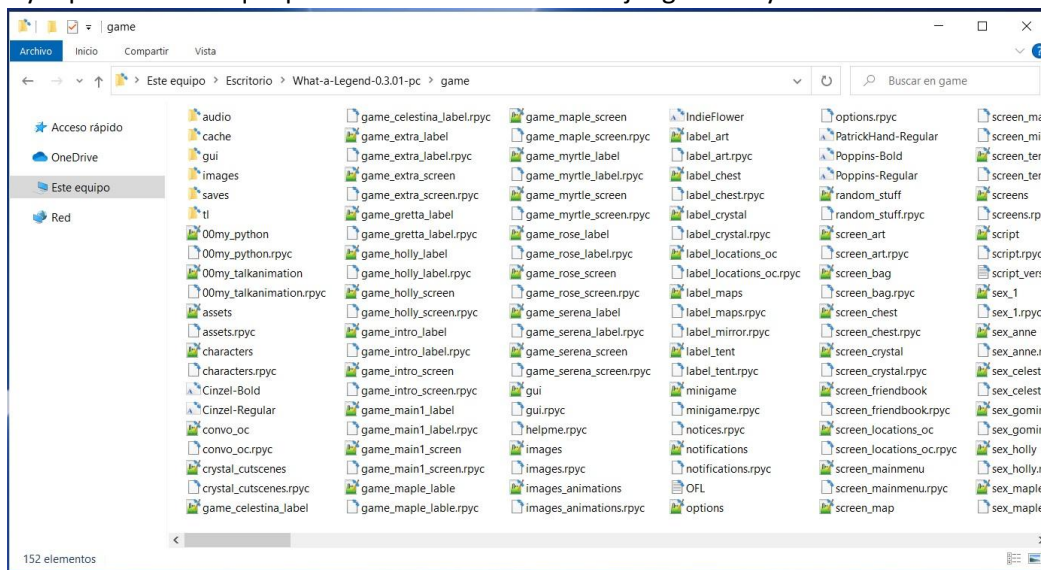
Primer mandamiento de Ren'Py: las sangrías se respetan y NUNCA se marcan con el tabulador, sino con la barra espaciadora (4 espacios, generalmente). Si Ren'Py detecta una tabulación o un fallo de sangría en alguna parte de sus scripts (incluyendo los de traducción) el juego no arrancará.

Si vamos bajando por el script, dentro de esta label vemos varias funciones para mostrar y ocultar imágenes y determinar qué parte de la conversación se va a mostrar según las variables acumuladas. Además, entre medias aparecen salpicadas algunas frases, que son los diálogos que aparecerán en pantalla. Esas frases entrecorridas son las que tendremos que traducir, y se denominan **strings**, o cadenas de texto.

Segundo mandamiento de Ren'Py: las comillas se cierran siempre. Unas comillas sin cerrar (o sin abrir, lo mismo da) equivalen a un juego que no arranca.

1.3.- ¿Cómo (creo que) funciona Ren'Py? El tercer mandamiento

Partiendo de la base de que no soy informático y es muy probable que diga alguna burrada, voy a intentar explicar muy rápidamente lo que pasa cuando arrancamos un juego Ren'Py.



¿Recordáis la lista de archivos duplicados dentro de la carpeta “game”? Pues no están exactamente duplicados. Ren'Py ejecuta los archivos .rpyc, que son una compilación de los archivos editables en formato .rpy. La compilación se realiza la primera vez que el desarrollador abre el juego en su entorno de creación, generándose un AST (árbol de sintaxis abstracta) en función del contenido disponible en ese momento y que será la base de todo lo que se vaya añadiendo después a las compilaciones. Siguiendo sus algoritmos, Ren'Py asignará unas identificaciones a cada elemento de programación de los archivos .rpy en función de su posición en el script y su naturaleza (texto, variable, función, etc.) y con esas identificaciones creará unos archivos .rpyc del mismo nombre que serán los que irá leyendo durante el juego.

Cada vez que se arranca el juego, Ren'Py busca archivos .rpy y, si los hay, los compara con sus respectivos .rpyc, buscando las identificaciones creadas en la compilación inicial. Si se han introducido cambios en el archivo .rpy, Ren'Py detectará las cosas que no han cambiado o simplemente se han movido de sitio, respetará sus identificadores anteriores y actualizará el .rpyc con el resto de novedades. Simplificando mucho, imaginemos que la línea número 3 del archivo .rpy se compila inicialmente de forma que en el archivo .rpyc se identifica con un 3. Si en futuras versiones del .rpy esa línea no se modifica pero pasa a ser la número 7, en el archivo .rpyc se mantendrá siempre con el 3. Así, aunque el archivo .rpy definitivo se parezca muy poquito al primero que creó el desarrollador del juego, su .rpyc correspondiente estará compilado siguiendo las relaciones que se crearon en el primer AST que se originó.

¿Y todo esto por qué lo cuento? Pues porque, si los borramos en algún momento, Ren'Py generará unos .rpyc nuevos pero usando como base la versión actual de los .rpy. Y esto significa que las relaciones e identificaciones que se crearán ahora en el AST no serán iguales que las iniciales, porque se están creando a partir de un .rpy que se parece poco al primero. En el ejemplo absurdo de antes, la línea 7 del .rpy que en realidad en el .rpyc borrado era la 3 porque venía arrastrada de versiones anteriores, al crearse un nuevo .rpyc desde la nada será identificada con el número 7. El juego arrancará y las partidas nuevas no tendrían por qué verse afectadas, pero funciones que usan las referencias del AST, como la carga de partidas guardadas en versiones anteriores del juego, pueden dejar de funcionar correctamente. Y, en determinados casos [que luego veremos](#), las traducciones también podrían dar problemas.

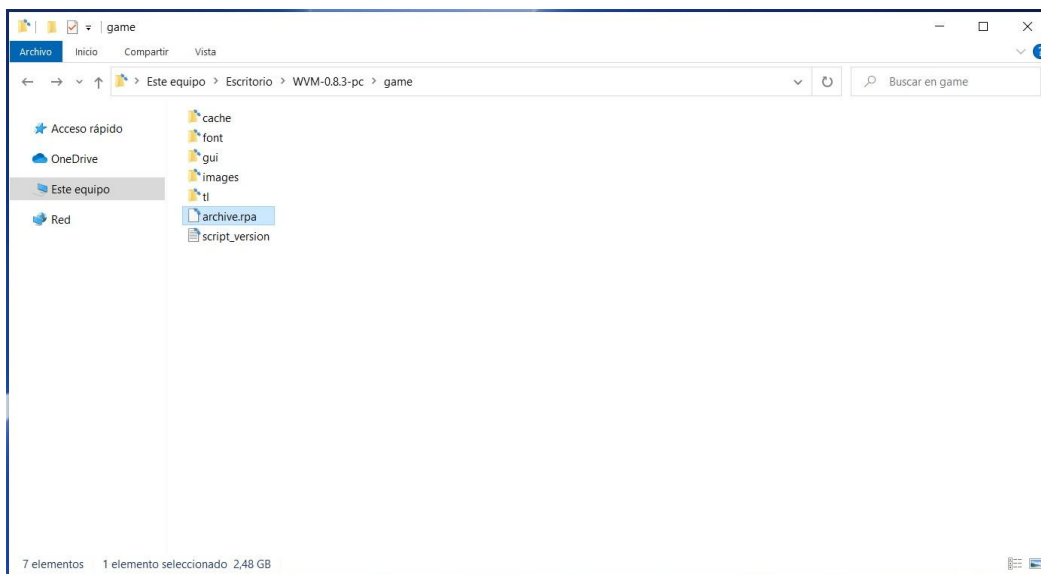
Tercer mandamiento de Ren'Py: borrar archivos .rpyc del juego original puede generar problemas. No lo hagas si no tienes una buena excusa (y yo no sé ninguna).

[\[Arriba\]](#)

1.4.- UnRen y los archivos .rpa

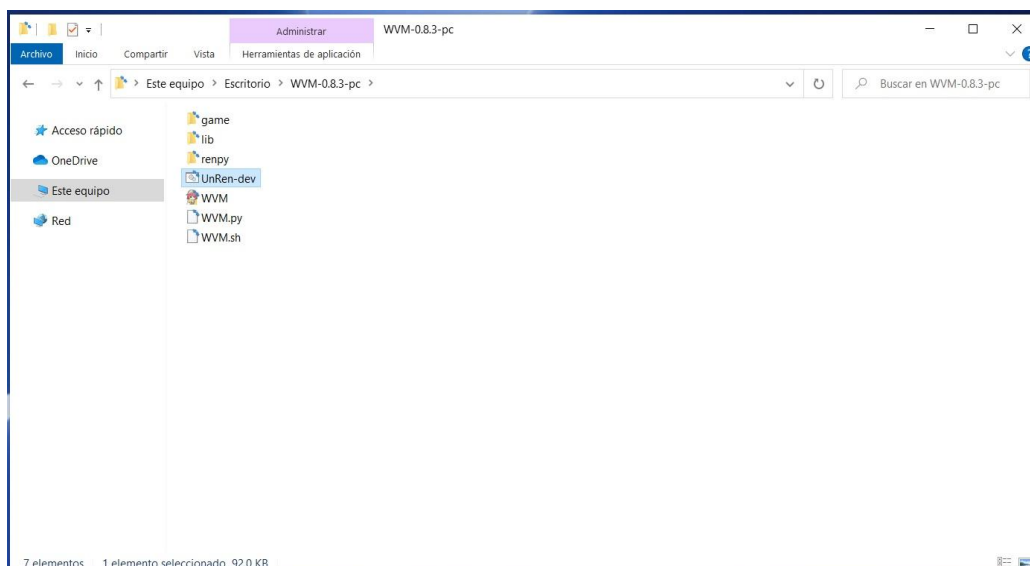
Puede pasar que, al abrir la carpeta `game`, no hayáis visto nada de todo lo que acabo de comentar. Y es que, a la hora de empaquetar el juego para publicarlo, Ren'Py les ofrece a los desarrolladores varias opciones. La recomendada es incluir los scripts sueltos en formato `.rpy` y `.rpyc`, como hemos visto en el ejemplo de "What a Legend!", pero es solo eso, una recomendación. Como el juego solo necesita scripts en formato `.rpyc` para funcionar, hay desarrolladores que solo incluyen los scripts en `.rpyc`. Así el juego ocupa algo menos de espacio en disco y (lo que a veces es más importante) los jugadores no tienen acceso directo a la programación del juego.

Y otra de las posibilidades con las que podemos encontrarnos, bastante común, es que en la carpeta "game" no haya scripts ni en formato `.rpy` ni en formato `.rpyc`, sino que el desarrollador haya aplicado otra de las opciones para lanzar el juego: comprimir los scripts y otros archivos (como las imágenes) en un fichero `.rpa`. Es decir, en vez de una lista de archivos `.rpy` y `.rpyc`, es posible que en la carpeta "game" haya algo así:

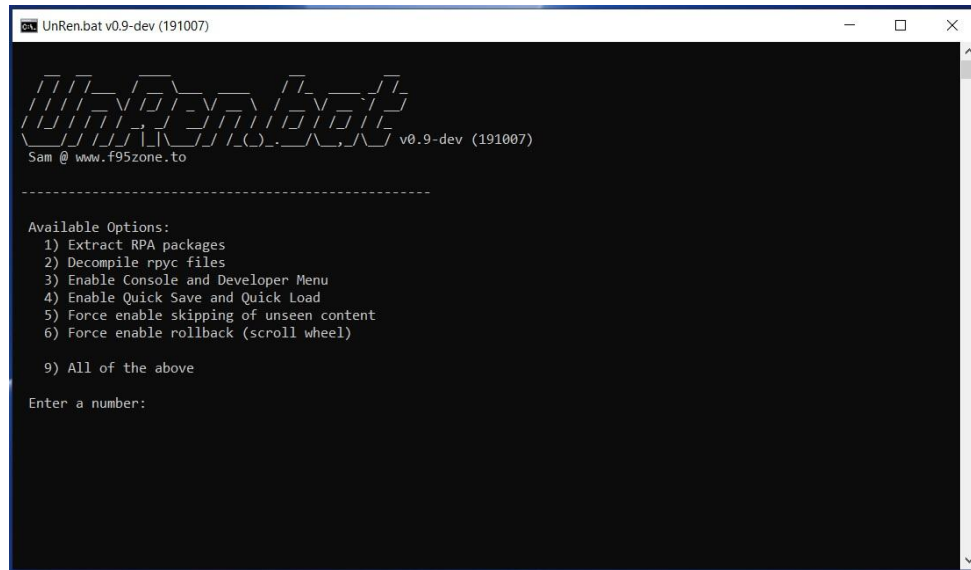


Lo primero de todo es confirmar que, como seguramente ya sepáis, no se trata de ningún problema. El juego va a funcionar perfectamente tal y como está, pero para poder traducirlo vamos a tener que dar algún rodeo extra, ya que necesitamos los scripts en formato `.rpy`. Afortunadamente, existen varias herramientas que nos permiten realizar las tareas necesarias sin tener que aprender el lenguaje Python. Por su simplicidad, creo que la más manejable es UnRen. [LINK](#) (ojo: enlace a un foro con contenido para adultos)

Una vez descargado UnRen, lo descomprimos y lo pegamos dentro de la carpeta del juego que nos interese, en el mismo nivel en el que esté el ejecutable del juego. Es más fácil verlo, supongo:



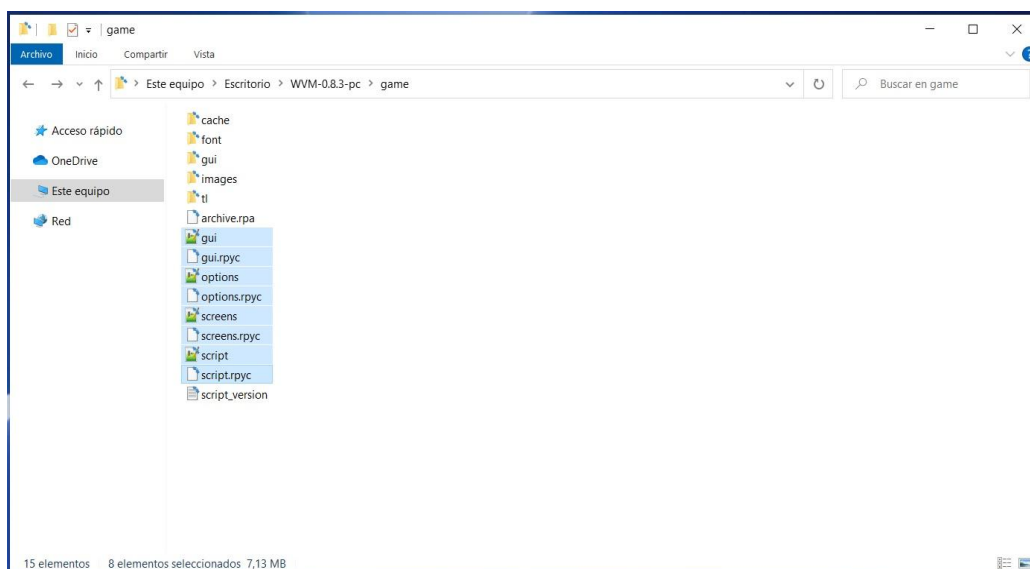
Ya solo es cuestión de abrirlo e indicarle lo que queremos hacer. Esta es la pantalla de inicio del programa:



Como se puede observar, aquí solo vamos a tener que manejar el teclado. No tiene más complicación que pulsar la tecla correspondiente a lo que queramos hacer. La opción 1) va a descomprimir los archivos .rpa, extrayendo todo su contenido en la carpeta “game”. Puede darse el caso de que dentro de ese archivo .rpa solo hubiera scripts en formato .rpyc, con lo cual, después de concluir esa opción 1), tendríamos que pedirle a UnRen que ejecutara la opción 2), que consiste en descompilar los archivos .rpyc para crear unos scripts en formato .rpy que ya podríamos usar para traducir.

Podemos acabar antes seleccionando la opción 9), que en un solo paso hace lo mismo que las dos anteriores y además nos habilita la opción de abrir la consola y el menú del desarrollador (útil para ver y modificar variables para acceder a partes concretas del script, y también para recargar el juego automáticamente cuando introduzcamos algún cambio en la traducción), así como otras opciones que el creador del juego podría haber deshabilitado.

Bien, pues dejamos que UnRen haga su trabajo y al final, cuando entremos de nuevo en la carpeta “game”, veremos que donde antes solo había un archivo en formato .rpa, ahora nos han aparecido los scripts que venían comprimidos dentro del mismo.



Ahora ya podríamos comenzar a traducir. Pero antes, un inciso para explicar lo que va a ocurrir con el juego. En este momento, dentro de la carpeta game hay scripts en formato .rpy, scripts en formato .rpyc y, además, dentro del fichero “archive.rpa” estarán esos mismos scripts .rpyc (y quizás hasta los .rpy). Por tanto, ahora sí tenemos archivos duplicados. Afortunadamente, Ren'Py está diseñado para solventar estas duplicidades de una forma fácil: **si encuentra dos scripts con el mismo nombre, ejecuta el más reciente**. Y, en este caso, el más reciente es el que acabamos de extraer con UnRen. Por lo tanto, podríamos borrar (y sería hasta recomendable, siempre que tengamos una forma de recuperarlo, por si acaso) el fichero “archive.rpa”.

Y otra aclaración más antes de seguir. Si en el archivo en formato .rpa venían comprimidos tanto los scripts .rpy como los .rpyc, los que ahora vemos sueltos en la carpeta “game” son exactamente los mismos que salieron del ordenador del desarrollador del juego. Pero si en el fichero .rpa solo había scripts en formato .rpyc, los scripts que ahora tengamos en formato .rpy son una simple aproximación creada por UnRen basándose en el AST que haya identificado dentro del .rpyc. Pero esto no significa que sea igual al .rpy original. Por ejemplo, desaparecen todos los comentarios que el desarrollador hubiera anotado tras un símbolo #, ya que no se compilan en el .rpyc y lógicamente UnRen, al leer solo el .rpyc, no puede saber si en el .rpy original había comentarios o no.

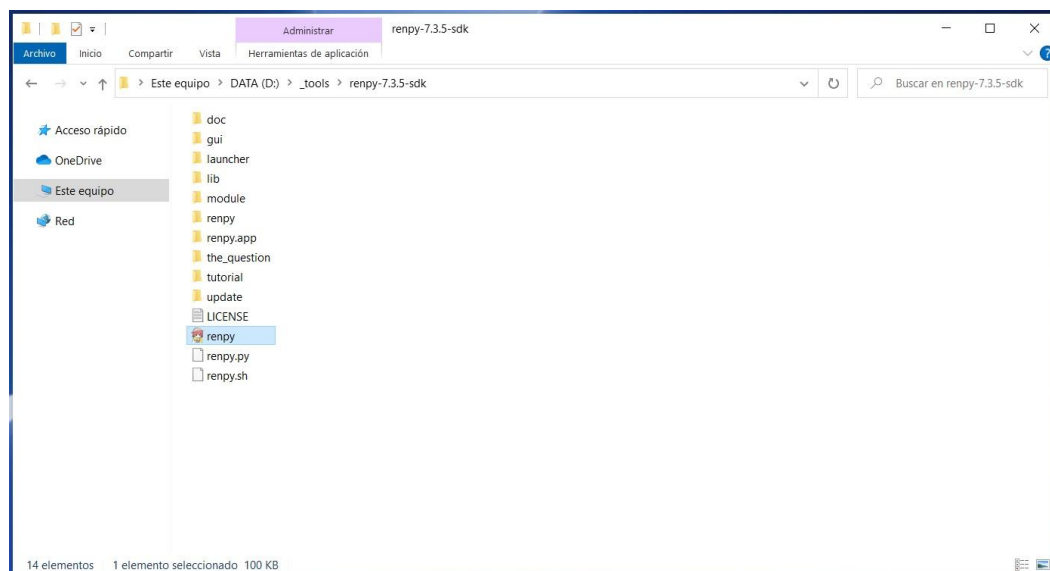
[|Arriba|](#)

1.4.- Ren'Py SDK

Como hemos visto, los juegos Ren'Py funcionan sin necesidad de instalar ningún programa extra que los lea. Son autoejecutables y además nos permiten modificar sus scripts con un simple editor de texto. Ahora bien, para crearlos sí hace falta un programa, lógicamente.

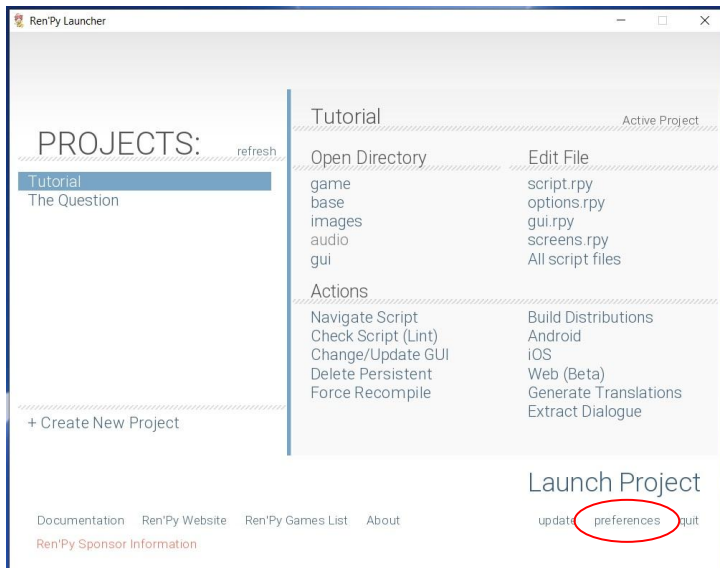
Ren'Py SDK es la aplicación que permite crear juegos Ren'Py, y también sus traducciones. Así que el primer paso para traducir es bajárselo. Es libre, limpio y gratuito. [LINK](#)

Una vez descargado y descomprimido, abris la carpeta y veréis algo así:



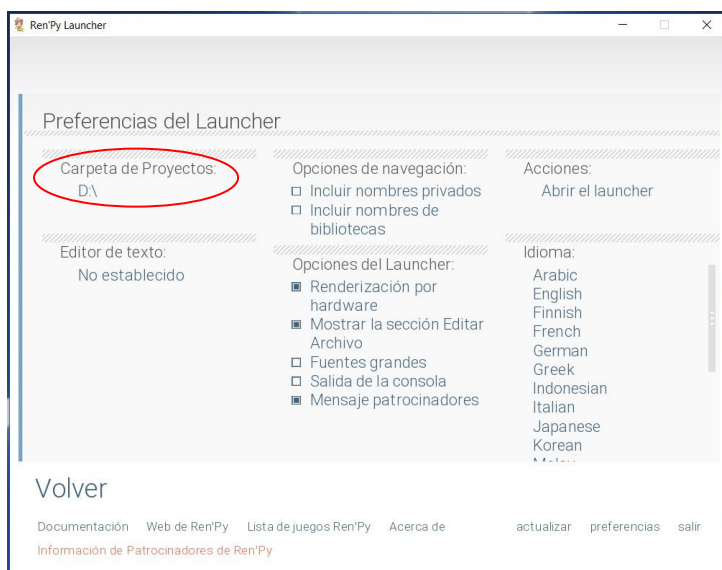
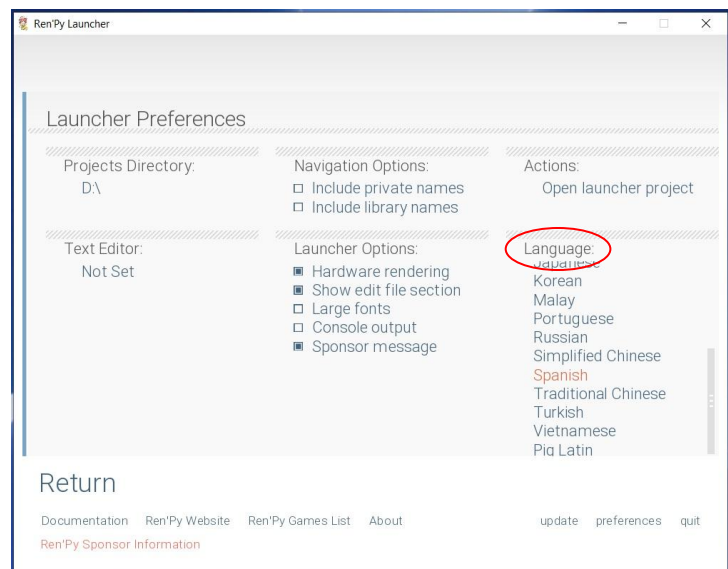
Es decir, una estructura muy similar a la de los juegos, con varias subcarpetas y un ejecutable. Entre las subcarpetas están “the_question” y “tutorial”, que son dos tutoriales para aprender a manejar Ren'Py. No viene mal echarles un vistazo, pero no aportan gran cosa a las traducciones.

Ya solo es cuestión de abrir el ejecutable “renpy” y empezar a trabajar. Tras configurar una serie de parámetros, aparecerá la pantalla principal, generalmente en inglés. Así que, antes de nada, vamos a cambiar el idioma para trabajar en español.



Para empezar, hay que pinchar en “preferencias” en la parte inferior derecha de la ventana.

En la parte derecha de esa pantalla de preferencias aparece una lista de idiomas disponibles (“Language”) para ejecutar Ren'Py. Buscamos nuestro Spanish y, en cuanto pinchemos, la pantalla cambiará de idioma.



Ahora ya tenemos el programa funcionando en español y no hará falta que volvamos a cambiarlo. Lo que ahí se llama “Carpeta de Proyectos” es el directorio donde el Ren'Py SDK almacenará y buscará los juegos. Así que solo se trata de seleccionar ahí el directorio donde tengáis la carpeta raíz del juego que queráis traducir. Es decir, si tenéis la carpeta de “What a Legend!” en Mis Documentos, pues seleccionad Mis Documentos. Le dais a “Volver” y en la lista de proyectos aparecerán todos los juegos que haya en esa carpeta, además de los dos tutoriales que se incluyen en el Ren'Py SDK.

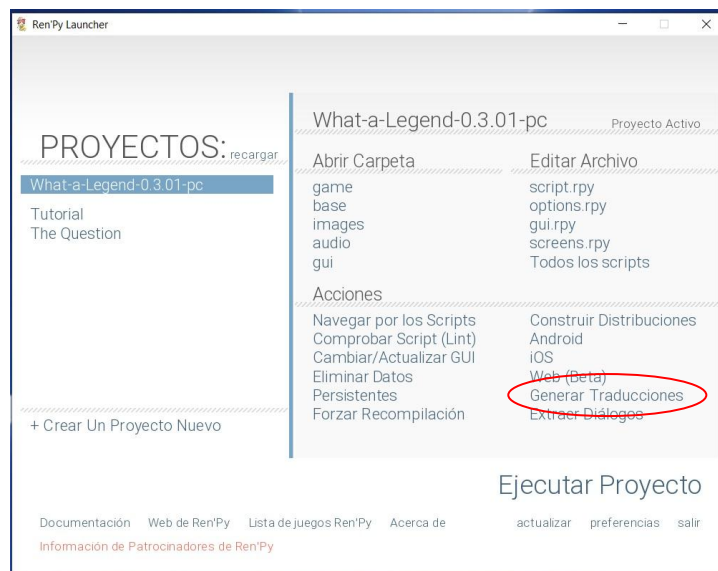
[\[Arriba\]](#)

2.- A TRADUCIR

Ya sea porque el desarrollador nos lo ha puesto fácil, o porque nos hemos buscado la vida para conseguirlo, lo importante es que a estas alturas tenemos nuestro juego con sus scripts en formato .rpy al descubierto y dispuestos para ser traducidos. Pero, por suerte o por desgracia, Ren'Py no traduce nada, simplemente ayuda a extraer los textos a traducir y hace que las traducciones se muestren luego donde corresponda. El Ren'Py SDK genera los scripts que sirven de base a la traducción, con los textos y los comandos necesarios; sin embargo, conseguir que lo haga bien y extraiga íntegramente los textos a traducir no siempre es tan fácil como darle a un botón, y en ocasiones tendremos que hacer un trabajo de edición de los scripts originales. Pero, ya que estamos aquí, vamos a darle a ese botón que seguro que ya estáis mirando con ojos golosos: Generar Traducciones. [|Arriba|](#)

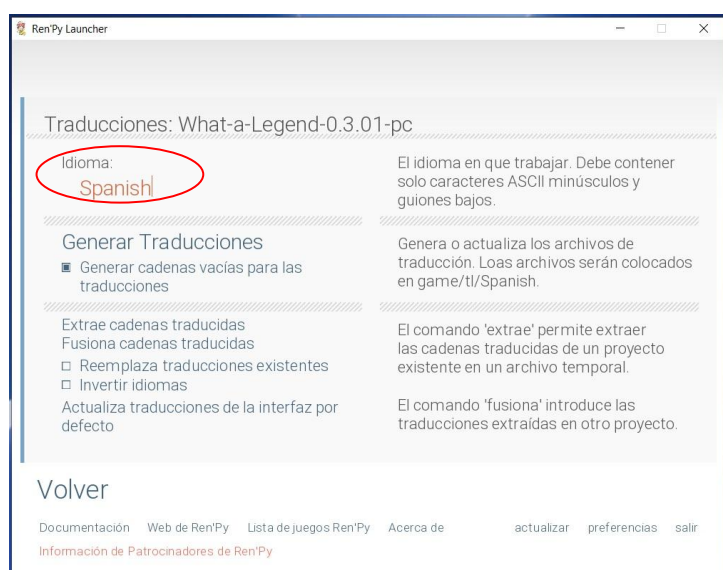
2.1.- Generando los archivos de traducción (y aprendiendo el cuarto mandamiento)

Tras seleccionar el juego que queremos traducir (aparecerá resaltado en la sección “Proyectos”, pulsamos el botón “Generar Traducciones” que hay en la parte derecha de la pantalla.



En la casilla “Idioma” debemos escribir el idioma al que queremos traducir el juego. No es más que una identificación interna, así que podemos poner lo que queramos (salvo caracteres especiales, como la ñ o vocales acentuadas). Lo importante es acordarnos de lo que hemos puesto aquí y tener muy en cuenta el cuarto mandamiento de Ren'Py:

Cuarto mandamiento de Ren'Py: Una letra mayúscula no es igual que una letra minúscula, nunca, bajo ningún concepto.

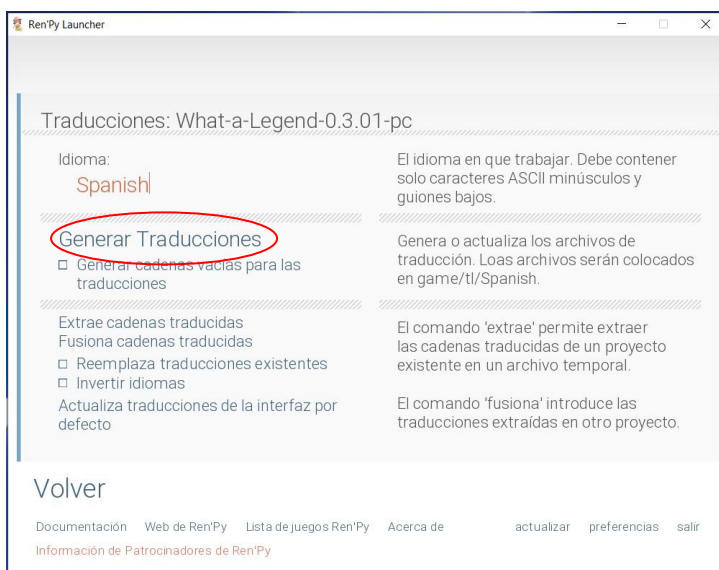


Así pues, podéis poner Spanish, spanish, espanol, Espanol, barco, Burro o amarillo. Lo que queráis. Pero, obviamente, es recomendable usar algo comprensible. Y, como digo, lo importante es recordar lo que ponéis y cómo lo ponéis. Yo, por deformación profesional, uso la palabra “Spanish” y esa será la que veréis de aquí en adelante en los ejemplos.

Después hay varias opciones, pero de entrada la única que nos interesa es la que indica “Generar cadenas vacías para las traducciones”. Aquí, como casi siempre, todo va en función de gustos. Aunque al traducir siempre tendremos visible una línea con el texto original para guiarnos, la idea es que, si seleccionamos esa opción, los scripts de traducción se generarán con unas líneas en blanco para que escribamos directamente nuestra traducción. Si no la seleccionamos, los scripts se generarán con las líneas en el idioma original del juego y tendremos que borrar esas palabras y sustituirlas por la traducción. Aquí podemos ver una comparativa de cómo quedarían los archivos de traducción si marcamos esa casilla (izquierda) y si no la marcamos (derecha).

<pre> 1 # TODO: Translation updated at 2020-12-11 13:16 2 3 # game/convo_oc.rpy:15 4 translate Spanish convo_gate_fdd309da: 5 6 # kevin "STOP!" with vpunch 7 kevin "" with vpunch 8 9 # game/convo_oc.rpy:18 10 translate Spanish convo_gate_e5ccb99b: 11 12 # kevin "Did you manage to get a passage permit?" 13 kevin "" 14 15 # game/convo_oc.rpy:21 16 translate Spanish convo_gate_bc693870: 17 18 # pov "Umm..." 19 pov "" 20 21 # game/convo_oc.rpy:24 22 translate Spanish convo_gate_6a0bcf57: 23 24 # kevin "I'm sorry, buddy..." 25 kevin "" 26 27 # game/convo_oc.rpy:26 28 translate Spanish convo_gate_26193626: 29 30 # kevin "...but no permit, no passage. That is the law." 31 kevin "" </pre>	<pre> 1 # TODO: Translation updated at 2020-12-11 13:41 2 3 # game/convo_oc.rpy:15 4 translate Spanish convo_gate_fdd309da: 5 6 # kevin "STOP!" with vpunch 7 kevin "STOP!" with vpunch 8 9 # game/convo_oc.rpy:18 10 translate Spanish convo_gate_e5ccb99b: 11 12 # kevin "Did you manage to get a passage permit?" 13 kevin "Did you manage to get a passage permit?" 14 15 # game/convo_oc.rpy:21 16 translate Spanish convo_gate_bc693870: 17 18 # pov "Umm..." 19 pov "Umm..." 20 21 # game/convo_oc.rpy:24 22 translate Spanish convo_gate_6a0bcf57: 23 24 # kevin "I'm sorry, buddy..." 25 kevin "I'm sorry, buddy..." 26 27 # game/convo_oc.rpy:26 28 translate Spanish convo_gate_26193626: 29 30 # kevin "...but no permit, no passage. That is the law." 31 kevin "...but no permit, no passage. That is the law." </pre>
---	--

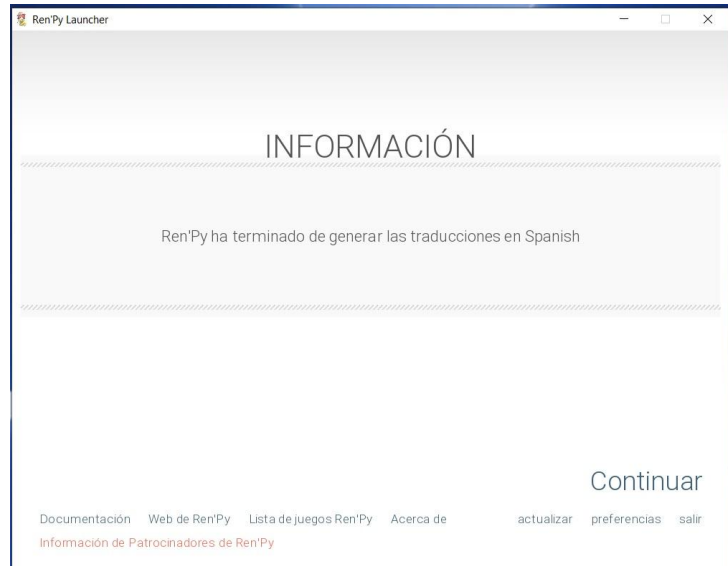
Puede parecer más cómodo (de hecho lo es) generar las cadenas vacías, pero, si se nos olvida traducir alguna línea, al llegar a ella en el juego no aparecerá absolutamente ningún texto en pantalla. De la otra forma, al menos saldría el texto en el idioma original, algo que puede resultarnos útil mientras hacemos pruebas con una traducción incompleta. Esto es especialmente importante en los menús, pues nos permite tener todas las opciones activas incluso cuando aún no los hayamos traducido.



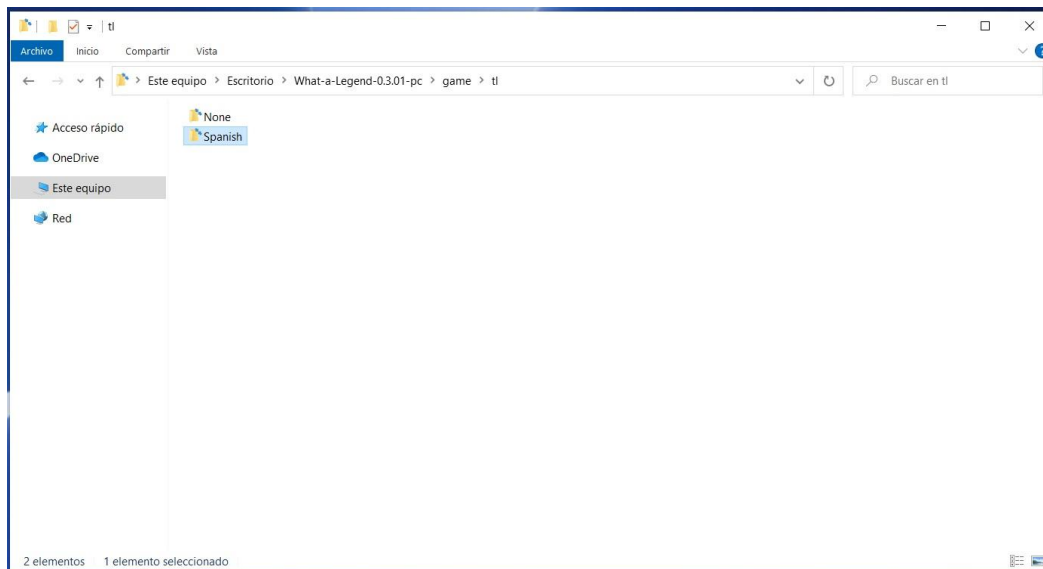
Personalmente, yo prefiero NO generar cadenas vacías, por lo que dejo esa casilla sin marcar, pero si vais a usar traductores automáticos es posible que os interese generarlas vacías.

Ahora ya solo queda pinchar en el botón “Generar Traducciones”. Como indica el texto informativo de la columna derecha (con error tipográfico incluido), al hacerlo se crearán unos archivos de traducción dentro de la carpeta “game/tl/Spanish” (el nombre de dicha carpeta será el que hayáis indicado en la casilla “Idioma”).

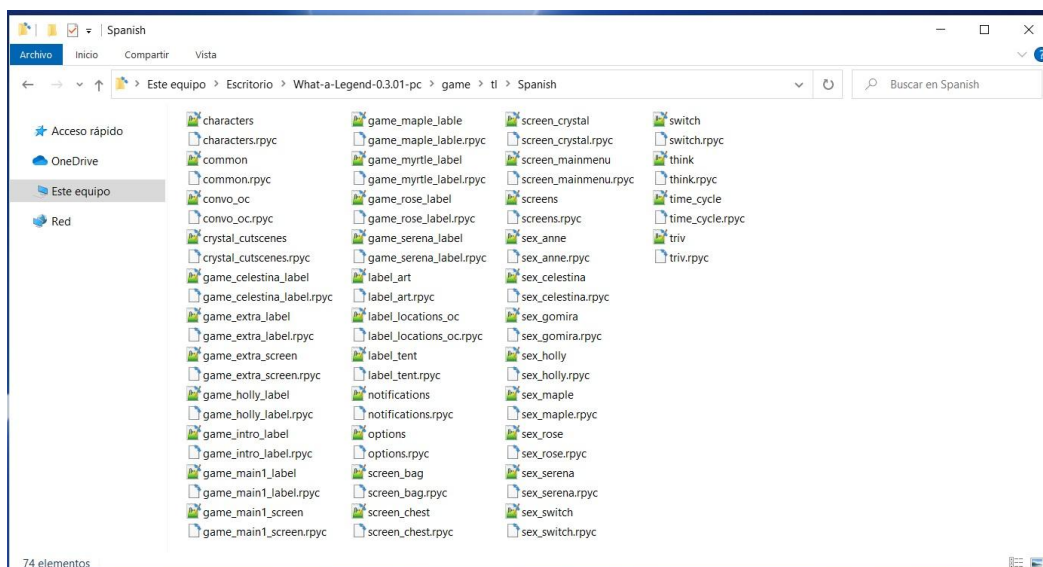
Si no hay ningún problema, cuando el Ren'Py SDK termine de hacer su magia nos aparecerá este mensaje. Podemos darle a “Continuar”, para volver a la pantalla de inicio, o directamente salir del Ren'Py SDK.



Si ahora vamos a la carpeta “game” de nuestro juego, veremos que en la subcarpeta “tl” aparecen otras dos subcarpetas: una llamada “None” y otra con el nombre del idioma que hayamos introducido en el Ren'Py SDK. En este caso, “Spanish”.



Y dentro de la carpeta “Spanish” estarán los scripts de traducción. ¿Qué ha hecho el Ren'Py SDK para crearlos? Pues ha ido repasando uno a uno los scripts originales por orden alfabético y, siempre que haya encontrado algún texto traducible en ellos, ha generado un archivo .rpy (y su correspondiente .rpyc) con el mismo nombre que el original, escribiendo en él las strings a traducir.



Además, se habrá generado un script adicional, llamado “common.rpy”, en el que aparecerán los comandos de los menús de Ren'Py que no son específicos del juego. Por ejemplo, el aviso que aparece cuando cerramos el juego, o los meses y días de la semana que aparecerán para nombrar las partidas guardadas. Es un archivo extenso y bastante farragoso de traducir, pero tiene una ventaja: suele ser común a la mayoría de juegos creados con la misma versión del Ren'Py SDK, por lo que puede ser intercambiable de un juego a otro. Es decir, muchas veces nos va a servir una traducción que ya tengamos hecha de un juego previo. Eso sí: antes de reemplazarlo, siempre deberíamos comprobar que las strings coinciden, ya que en ocasiones hay pequeños cambios y podríamos romper algo.

Llegados a este punto, solo es cuestión de ir abriendo los scripts en formato .rpy con un editor de texto y empezar a traducir. Incluso podéis modificar el nombre de los scripts en formato .rpy (y hasta unificarlos en un solo documento), ya que Ren'Py buscará las traducciones string por string y le da igual en qué archivo estén, pero lo normal es no tocar nada para identificar fácilmente cada script traducido con su script original.

[|Arriba|](#)

2.2.- Las dos funciones de traducción (y otro mandamiento más)

Algo que suele pasar la primera vez es que empezamos a traducir y de repente pensamos: ¿pero dónde están las opciones para el jugador? ¿Aquí no me preguntaban si quería hacer una cosa u otra? Que no cunda el pánico: los textos a traducir no aparecen exactamente en el mismo orden que en el script original. Y es que, a la hora de extraer de cada script las strings a traducir, Ren'Py las divide en dos: las que forman el cuerpo de los diálogos y las que se refieren a opciones de menú, variables, nombres de personajes, etc. ¿Por qué? Pues porque cada uno de estos grupos va a usar una función de extracción y traducción distinta.

Las strings que forman el cuerpo de los diálogos no nos van a generar ningún problema a la hora de extraerlas, porque Ren'Py las va a identificar sin problemas. Lo que hace con ellas, sin embargo, sí nos va a causar más de una molestia y puede que algún dolor de cabeza al traducir y crear el parche con la traducción. Porque lo que hace Ren'Py es cifrarlas para ahorrar memoria: usando el método de encriptación MD5 hexadecimal a 8 bytes, cada línea pasa a ser identificada con un código alfanumérico que depende de la label donde se enmarque ese texto en el script original y del propio contenido del texto.

Lo vemos con un ejemplo. En primer lugar, recordamos cómo era el comienzo del script “convo_oc.rpy” del juego “What a Legend!”:

```

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 hfneul hbneul
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 hfneul
26         kevin "...but no permit, no passage. That is the law."

```

La primera línea con texto a traducir (la primera “string”) está en la línea 15 del script original, dentro de la label “convo_gate”: un personaje identificado como Kevin dice “STOP!” (con vpunch, un efecto que “sacudirá” la imagen). El resto del código hasta ese punto no contiene ningún texto a mostrar en pantalla.

Pues bien, al generar el script de traducción con cadenas vacías, Ren'Py ha transformado toda esa información en esto:

```

1 # TODO: Translation updated at 2020-12-11 13:16
2
3 # game/convo_oc.rpy:15
4 translate Spanish convo_gate_fdd309da:
5
6     # kevin "STOP!" with vpunch
7     kevin "" with vpunch
8
9 # game/convo_oc.rpy:18
10 translate Spanish convo_gate_e5ccb99b:
11
12     # kevin "Did you manage to get a passage permit?"
13     kevin ""
14
15 # game/convo_oc.rpy:21
16 translate Spanish convo_gate_bc693870:
17
18     # pov "Umm..."
19     pov ""
20
21 # game/convo_oc.rpy:24
22 translate Spanish convo_gate_6a0bcf57:
23
24     # kevin "I'm sorry, buddy..."
25     kevin ""
26
27 # game/convo_oc.rpy:26
28 translate Spanish convo_gate_26193626:
29
30     # kevin "...but no permit, no passage. That is the law."
31     kevin ""

```

Vemos que, para cada string, Ren'Py nos ha generado un bloque de cuatro líneas. Analicemos en qué consisten estos bloques:

```

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

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

```

Las líneas que empiezan con el símbolo # son meramente informativas y podríamos borrarlas sin consecuencias. La primera nos indica la ubicación de esa línea en el script original: es la línea 15 del archivo “convo_oc.rpy” dentro de la carpeta “game”. La otra línea que empieza con # es, ni más ni menos, el texto que hay que traducir, que Ren'Py nos ofrece amablemente para que sepamos lo que debemos escribir. Y la última línea es donde vamos a escribir nuestra traducción (solo lo que haya entre comillas, el resto lo dejamos tal y como está). Aquí, si hubiéramos generado la traducción sin cadenas vacías, nos parecería nuevamente el texto en inglés.

La miga está en la segunda línea. Esa es la que activa la función de traducción para esta línea concreta del código original. Cuando juguemos con una traducción, Ren'Py estará ejecutando el juego según los archivos .rpyc originales de la carpeta “game”, pero tendrá la orden de mostrar los textos traducidos en vez de los originales. Esa orden le dice que, en cada línea del cuerpo de diálogos, tiene que buscar en los scripts una línea con el comando “translate” más el idioma que le hayan dicho (“Spanish”, en este caso, que fue lo que pusimos al generar la traducción) más el código de encriptación que corresponde a la string original. Este código de encriptación empieza por el título de la label donde se encuentra esta línea concreta (“convo_gate”) y continúa con el resultado de aplicar el sistema MD5 al contenido de la línea (al parecer, en ese sistema de encriptación, “STOP!” equivale a fdd309da).

Si dentro de esa misma label “convo_gate” hubiera otra string idéntica (otro “STOP!”), Ren'Py tendría que asignarle el mismo código de encriptación. Pero esto podría generar conflictos, así que a la segunda le añadirá un _1 al final, si hubiera otra idéntica le pondría un _2 y así sucesivamente. Así pues, los códigos de encriptación son únicos para cada string, incluso para strings iguales dentro del bloque de diálogos.

¿Pero qué pasa si vamos ahora al script original y donde pone “STOP!” escribimos “Stop!”? Pues que la traducción que tengamos para “STOP!” dejará de servirnos, porque el código MD5 de “Stop!” será distinto y Ren'Py no lo encontrará en los scripts de traducción, así que mostrará el texto original. Esto supone una cierta molestia a la hora de traducir nuevas versiones de juegos en desarrollo, donde es habitual que los creadores corrijan pequeños errores ortográficos o de puntuación que se les habían pasado por alto. Tendremos que traducir de nuevo strings que ya teníamos traducidas, porque al cambiar mínimamente

pasarán a tener otro código de encriptación y para Ren'Py son completamente distintas. Y lo mismo ocurre si, aunque no modifiquen su contenido, se renombra la label en la que se encuentran.

Quinto mandamiento de Ren'Py: Cualquier mínimo cambio introducido en la string original invalidará la traducción preexistente de esa línea.

Por suerte, si lo que se produce es un simple cambio de línea (es decir, la string pasa de estar en la línea 7 a estar en la 8, pero su contenido no varía y tampoco cambia de label), la traducción anterior seguirá sirviendo, ya que ese desplazamiento no implica ningún cambio en la encriptación. Simplemente, el comentario que nos indica en qué línea está esa string en el script original quedará desactualizado, pero eso no tiene mayor impacto para la traducción.

Pero, como digo, Ren'Py usa dos sistemas de traducción. Además del basado en códigos de encriptación, para el resto de strings que no pertenecen al bloque de diálogos aplica un sistema más sencillo de traducción directa. ¿A qué strings nos estamos refiriendo? Pues, como decía al principio, a las opciones que se le presentan al jugador durante el juego, pero también al nombre de los personajes, a posibles variables que use el juego y que sean texto, a los menús del sistema (preferencias, guardar partida...), etc.

Cuando Ren'Py esté ejecutando un juego traducido, todas esas strings van a ser reemplazadas en pantalla por sustitución directa, sin rodeos ni codificaciones. Para ello, a la hora de generar los scripts de traducción las agrupa todas al final del documento, bajo el comando “translate Spanish strings”. Este es el comienzo de la segunda sección del script de traducción del script “convo_oc” de “What a Legend!”:

```
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 ""
```

Como veis, aquí ya no hay códigos raros. Y si antes el comando “translate” se aplicaba a cada línea, ahora con un mismo comando se traducen todas. Lo que sigue habiendo es una primera línea con el símbolo # para informarnos sobre la ubicación de la frase a traducir, pero luego pasa directamente a indicar, con el comando “old”, el texto original; y con el comando “new” el texto que deberá aparecer cuando el juego esté ejecutando una traducción al idioma “Spanish” (el que habíamos indicado al Ren'Py SDK).

A diferencia de lo que ocurría antes, ahora no nos va a aparecer ninguna string repetida, no ya en este script, sino en el conjunto de todos los scripts que formen el juego. Cuando el Ren'Py SDK repase por orden alfabético los scripts originales para generar los archivos de traducción, únicamente extraerá cada string la primera vez que se la encuentre, y al jugar, siempre que aparezca esa string, se sustituirá por la traducción que hayamos indicado esa única vez. Esto, que puede parecer una ventaja puesto que nos ahorra repetir trabajo, se vuelve un inconveniente cuando nos encontramos con strings idénticas que pueden tener significados distintos dependiendo del contexto. Por ejemplo, pensad en una string que solo diga “Right”, y que unas veces tendríamos que traducir como “Correcto” y otras como “Derecha”, o “Derecho”: solo se generaría una string de traducción y al jugar siempre aparecería esa única traducción, con lo que en ocasiones el texto no tendría sentido. [Más adelante](#) veremos cómo solucionarlo.

Usando el ejemplo de arriba, una vez que en este script indiquemos la traducción que corresponde a la string “Permit”, esa traducción aparecerá en todo el juego cada vez que haya una string “Permit” fuera del cuerpo de diálogos. Pero ojo: de acuerdo con el cuarto mandamiento de Ren'Py, si hubiera otra string con el texto “permit”, sí necesitaríamos traducirla, porque, a diferencia de otros programas informáticos, Ren'Py distingue perfectamente entre mayúsculas y minúsculas.

Volviendo al tema de las funciones de traducción, ¿por qué usa Ren'Py dos funciones distintas? Pues por un tema de agilidad y uso eficiente de memoria. Técnicamente, esta traducción directa podría hacerla para absolutamente todas las strings del juego, pero por lo general el bloque de diálogos es muy extenso y consta de frases mucho más largas que, normalmente, casi nunca se repiten. Así que, cuando el programa tiene que mostrar una traducción, tarda menos en buscar y reemplazar miles de códigos encriptados que miles de líneas enteras más largas. Sin embargo, estas otras strings de los menús son generalmente frases más cortas, mucho menos numerosas y posiblemente pueden repetirse con más frecuencia, así que Ren'Py puede permitirse hacer una búsqueda concreta del contenido de las strings sin que la ejecución del juego pierda velocidad.

En fin, son solo unas nociones perfectamente prescindibles. Lo importante es saber que para el Ren'Py SDK hay dos tipos de “strings”, que cada uno de esos tipos utiliza un sistema de traducción distinto y que por eso aparecerán por separado en los scripts de traducción: primero todas las strings pertenecientes a los diálogos, y luego todas las que corresponden a menús, opciones y variables.

[\[Arriba\]](#)

2.3.- Ayudando y/o sustituyendo al extractor

El motivo de haber explicado [en el punto anterior](#) los dos tipos de funciones de traducción de Ren'Py es que, por desgracia, el [Ren'Py SDK](#) no siempre es capaz de detectar absolutamente todas las strings que deberíamos traducir: sí extraerá todas las que forman el bloque de diálogos (las que se traducirán mediante el código de encriptación), pero tal vez no sea capaz de identificar todas las que se traducen por el método directo (aunque sí extraerá las opciones que nos permiten tomar decisiones dentro del juego).

Por ejemplo, para que extraiga automáticamente los nombres de los personajes, el desarrollador del juego tendría que haber definido a esos personajes de una forma muy concreta que, por desconocimiento, muchos no utilizan. Pero, si ellos no lo hacen, podemos hacerlo nosotros.

Como los creadores de “What a Legend!” sí hicieron sus deberes y nos facilitan mucho la vida a los traductores, vamos a usar para estos ejemplos el otro juego que ya habíamos usado para hablar de [los archivos .rpa y el extractor UnRen](#). Así pues, veamos cómo define el creador del juego “WVM” a uno de los personajes que intervienen en la historia. En este caso, en el archivo “script.rpy” ha creado un “yo interior” para indicar que lo que leemos en el cuadro de diálogo es un pensamiento de nuestro propio personaje:

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

Esta es una típica línea de código para Ren'Py. Empieza con el comando (`define`) que indica lo que hace esta línea concreta, en este caso definir una variable. A esa variable se le asigna un nombre que se utilizará en el resto del script para identificarla (`mcm`) y se dice que se trata de una variable de tipo personaje (`Character`). Ahora Ren'Py sabe que a lo largo del script, cuando se encuentre las letras `mcm` antes de una string, tiene que mostrar un nombre encima del cuadro de texto para que sepamos qué personaje está hablando. Y lo que va a mostrar es lo que figura después entre paréntesis: el nombre del personaje (“Your thoughts”), que irá en un color concreto, con una línea que rodeará las letras para remarcarlas.

Obviamente, “Your thoughts” es una string que deberíamos traducir, pero, si generásemos los scripts de traducción con el Ren'Py SDK, no nos aparecería por ninguna parte. Misterios de Ren'Py. ¿Qué podemos hacer? Pues hay tres opciones. La primera, que descartamos, es aceptarlo y dejar que el nombre siga saliendo en inglés. La segunda es ayudar al Ren'Py SDK a entender que ese entrecomillado es una string que queremos traducir y no un simple código interno como el que indica el color. Fijaos:

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

Hemos editado el script original para meter la string “Your thoughts” entre paréntesis y hemos colocado un guion bajo `_` antes del paréntesis. Esta combinación de símbolos `_ ()` permitirá al Ren'Py SDK identificar el entrecomillado que hay entre paréntesis como una string.

Y si ahora generásemos los scripts de traducción, dentro del bloque destinado a las traducciones directas ya nos encontraríamos con esto:

```
translate Spanish strings:

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

La traducción es mía, claro. Pero lo importante es eso: si no hubiéramos editado el script original para cambiar “Your thoughts” por _(“Your thoughts”), esta string no nos aparecería inicialmente para traducir.

Eso no quiere decir que no hubiéramos podido traducirla: al tratarse de una traducción directa, siempre podríamos escribirla en nuestro script de traducción sin ayuda del Ren'Py SDK. Esta es la última de las tres opciones que comentaba antes: en vez de editar los scripts originales, escribimos en los de traducción las strings que el Ren'Py SDK no ha detectado. Para ello iríamos a la parte del documento donde aparecen las traducciones directas (la que comienza con la función `translate Spanish strings:`) y escribiríamos una función de traducción con el mismo formato que vemos arriba: respetando siempre la sangría de 4 espacios y las comillas, como dicen los dos primeros mandamientos de Ren'Py, en una línea pondríamos el comando `old` seguido de la string a traducir (respetando escrupulosamente su escritura, como dicen el cuarto y el quinto mandamiento de Ren'Py), y debajo el comando `new` con la traducción. La línea que empieza con `#` no sería necesaria ya que es meramente informativa.

Personalmente, me he acostumbrado a revisar y editar primero los archivos originales para que el Ren'Py SDK haga luego un trabajo completo de extracción. Eso sí, en ocasiones se me pasa alguna string y tengo que regenerar varias veces los scripts de traducción, lo cual no supone más engorro que el tener que abrir nuevamente el Ren'Py SDK y darle a generar traducciones. Otros, para tocar lo menos posible los scripts originales, prefieren generar al principio esos scripts de traducción y luego ir modificándolos a mano para incluir todo lo que el Ren'Py SDK se haya dejado sin extraer a la primera. Cuestión de gustos y mecánicas.

Tanto si queréis ayudar previamente al Ren'Py SDK como si preferís copiar después a mano las strings que faltan en los scripts de traducción, tendréis que revisar los scripts originales y buscar este tipo de comandos:

- `Character "...")`, usado para definir personajes, como hemos visto en el ejemplo
- `text "..."`, usado para mostrar texto en una pantalla específica, fuera del diálogo
- `textbutton "..."`, usado para textos que realizan una acción al pinchar en ellos
- `tooltip `...``, usado para mostrar un mensaje emergente al pasar el cursor sobre un punto
- `renpy.input(...)`, usado para permitir teclear (nombres de personajes, generalmente)
- `$ renpy.notify(`...`, 'unlock')`, usado para mostrar mensajes tras completar una acción

Además, tenemos las variables de texto. Aquí podemos encontrarnos con varias posibilidades, pero ninguna de ellas será extraída automáticamente por el Ren'Py SDK:

- `default nombre_de_variable = "..."`
- `define nombre_de_variable = "..."`
- `$ nombre_de_variable = "..."`

Así pues, sería cuestión de meter dentro del símbolo `_ ()` todos esos entrecorchetados en los que he puesto puntos suspensivos y generar los scripts de traducción, o bien de copiarlos directamente en un script de traducción (es indiferente en cuál) con el comando `old` y debajo el comando `new` con su traducción. Tened en cuenta que hay que copiar todo lo que aparezca entre comillas, no solo el texto a traducir, ya que en ocasiones se añaden etiquetas dentro de los símbolos `{ }` para mostrar el texto en negrita, cursiva, con otro tipo o tamaño de letra, y esas etiquetas también forman parte de la string que Ren'Py buscará y reemplazará.

Para acabar con este punto, una puntualización: el símbolo `_ ()` solo se usa para extraer las strings que haya dentro del paréntesis. Una vez extraídas a un script de traducción, podemos borrarlo y la traducción funcionará bien, igual que con el método de escribir directamente esas strings en el script de traducción. Por lo tanto, en el caso de juegos que van sacando nuevas versiones, si dentro de los scripts de traducción que podemos reutilizar ya figura esa string traducida, no sería necesario volver a editar los scripts originales para extraerla. Y tampoco es una edición que nos obligue introducir ese script en [el parche](#), ya que el jugador no necesita tenerla. No me avergüenza reconocer que esto lo he aprendido hace relativamente poco y que, de haberlo sabido antes, me hubiera ahorrado algunas horas de trabajo innecesario.

[|Arriba|](#)

2.4.- La opción de cambio de idioma

Por último, o quizás en primer lugar antes de empezar a traducir, hay que pensar en cómo vamos a activar la traducción en el juego una vez creada. ¿Queremos que el juego se inicie siempre en español? ¿Añadimos una opción de cambio de idioma en el menú principal, o en el de opciones? ¿Preferimos que el jugador decida en qué idioma quiere jugar cada vez que arranque el juego? ¿Lo hacemos con una simple pregunta o creamos una pantalla gráfica con banderitas y demás? Como siempre, depende de los gustos, el tiempo disponible y la capacidad de cada traductor, y obviamente también de la configuración del juego original.

Lo más sencillo es hacer que el juego se inicie siempre en español. Para ello, abriremos cualquier script del juego (preferiblemente el “gui.rpy” porque es donde se define gran parte de la configuración del juego, pero es indiferente, e incluso podemos crear uno nuevo) y escribiremos esta línea de código, sin sangría:

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

Si pongo Spanish es, recordemos, porque es el nombre que le di a mi idioma al generar los scripts de traducción; vosotros usad el término que elijerais. Si no hacemos nada más, el juego se va a iniciar siempre en español y el jugador no tendrá ninguna opción dentro del juego para cambiar al idioma original. Incluso desactivando esta línea de código (borrándola o escribiendo un símbolo # delante) el juego no volvería a arrancar en el idioma original, ya que Ren'Py hace que, por defecto, el juego se inicie siempre en el último idioma en el que se jugó. Habría que establecer un nuevo `config.language` con el idioma original (que para Ren'Py siempre se denomina None), o desinstalar el juego, borrar los datos que Ren'Py almacena como copia de seguridad en una carpeta de sistema del ordenador y volver a instalarlo.

Lo recomendable, en cualquier caso, es añadir siempre una opción de cambio de idioma dentro del menú Preferencias. Suponiendo que el juego use la pantalla de preferencias que Ren'Py incorpora por defecto, tendríais que ir al script “screens.rpy”, buscar la sección del menú “Preferences” y hacer que quede así:

```
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")
756
```

El código exacto a escribir sería este (recordad que hay que respetar las sangrías con la barra espaciadora):

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

Lógicamente, esto de _("English") sirve cuando el idioma original del juego sea el inglés; si no, habría que poner el que fuera pero sin cambiar nada más, ya que para Ren'Py siempre sería el idioma None.

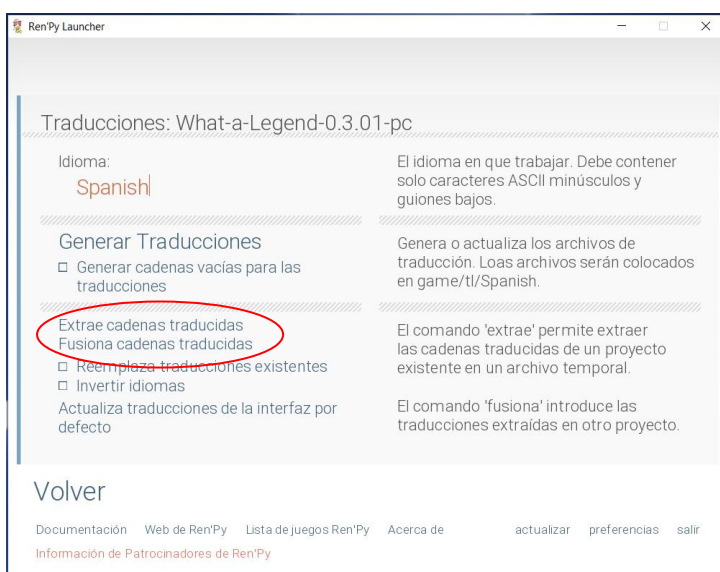
Si os fijáis, en ese código hay varias strings dentro del símbolo _ () para que el Ren'Py SDK pueda extraerlas. Personalmente, yo solo traduzco la del título, "Language" y dejo las demás sin traducir. ¿Por qué? Pues para que los nombres de los idiomas aparezcan siempre escritos en su propio idioma, independientemente de en qué lengua estemos viendo el menú. Es decir, si no traducimos English, cuando un jugador que no sepa español se encuentre con nuestra traducción y arranque el juego en español, al pinchar en el menú preferencias verá rápidamente la palabra "English" (no "Inglés") y no le costará deducir que ahí puede cambiar de idioma. Pensad en cuánto os gustaría ver la palabra "Español" en medio de un menú escrito con letras chinas o cirílicas y entenderéis por qué lo hago así.

En cuanto al resto de opciones, lo cierto es que hay tantas como juegos y traductores, por lo que no voy a ponerme a detallar cada una de ellas. Simplemente, fijaos en la configuración del menú de preferencias del juego (si está personalizado), haced un poco de ingeniería inversa y revisad foros y manuales online de Ren'Py para crear screens, splashscreens, imagemaps, botones... Hay todo un mundo de posibilidades, a cada cual más compleja y visualmente atractiva.

[|Arriba|](#)

2.6.- Traduciendo actualizaciones

En estos tiempos de Patreon es muy habitual que los juegos no se publiquen completos, sino por episodios, y nos veamos obligados a ir actualizando nuestras traducciones a medida que se van lanzando nuevas versiones. El procedimiento no tiene mayor complicación... salvo que queramos hacerlo como nos sugiere el Ren'Py SDK. Y es que, en teoría, Ren'Py nos ofrece una opción para extraer las traducciones de una versión anterior e insertarlas en la nueva, actualizando los scripts de traducción con el nuevo contenido.



Para ello, entraríamos en el Ren'Py SDK, localizaríamos en la carpeta de Proyectos la **versión anterior** del juego (la que tenemos traducida) y, en la pantalla de Generar Traducciones seleccionaríamos la opción **"Extrae cadenas Traducidas"**.

Después, volveríamos a la pantalla inicial del Ren'Py SDK, seleccionaríamos la **nueva versión** del juego (la que queremos traducir ahora) y, en esta pantalla de Generar Traducciones, elegiríamos **"Fusiona cadenas traducidas"**.

Teóricamente, al final del proceso tendríamos, en la carpeta "game/tl/Spanish" de la nueva versión del juego, nuestra traducción anterior mezclada con las nuevas strings a traducir correspondientes a esta nueva versión. El problema es que este procedimiento no funciona como debería, y las traducciones suelen acabar

incompletas: sí se copiarán los scripts de traducción correspondientes a scripts que no hayan sufrido cambios, pero si el script original de la nueva versión no coincide con el de la anterior, Ren'Py SDK le generará un script de traducción completamente nuevo y no le incorporará las strings que ya estuvieran traducidas en la versión anterior, obligándonos a repetir el trabajo de traducción.

Así que lo más efectivo es, sencillamente, bajar la nueva versión del juego, pegar en ella la carpeta "game/tl/Spanish" que tenemos en la versión anterior (es decir, nuestra vieja traducción), y generar los archivos de traducción de la nueva versión. Si el idioma que indicamos en el Ren'Py SDK es el mismo de la versión anterior ("Spanish", en mi caso), y **NO** marcamos la opción "Reemplaza traducciones existentes", lo que pasará será que, en vez de crear unos nuevos scripts de traducción con todas las strings del juego, Ren'Py se limitará a actualizar los existentes con el contenido para el que no encuentre una traducción válida. Es decir, añadirá al final de cada script tanto las strings correspondientes al nuevo contenido del juego como las que, perteneciendo a versiones anteriores, hayan sufrido alguna variación, volviendo a crear dos bloques (primero las strings del cuerpo de diálogos y luego las de opciones y menús). Pero solo, insisto, con las strings que no estuvieran ya traducidas, lo cual es una enorme ventaja con respecto al método oficial.

Este mismo procedimiento nos vale para actualizar los scripts de traducción después de haber editado los originales para incluir el símbolo `_` () donde fuera necesario, en el caso de que el Ren'Py SDK no nos hubiera extraído todas las strings a la primera (lo que explicamos en el [punto 2.3](#)). O sea, editaríamos los scripts originales y generaríamos una nueva traducción **sin reemplazar las existentes**, y conseguiríamos que el Ren'Py SDK nos incluyera, al final del script, una nueva lista con las strings pendientes de traducir que antes no había detectado.

[|Arriba|](#)

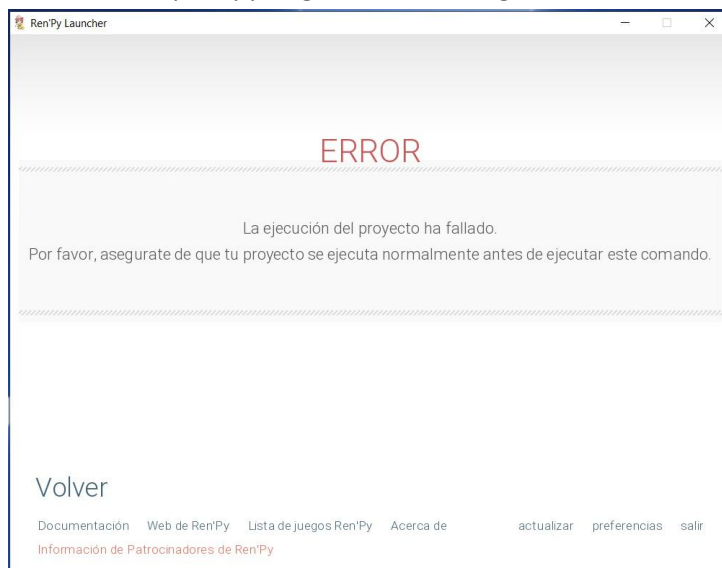
3.- Y ESTO POR QUÉ NO ME SALE

Si con todo lo que he ido contando hasta ahora habéis sido capaces de crear vuestra propia traducción y hacer que funcione todo correctamente, enhorabuena por haber estado atentos, y también por haber escogido para vuestras prácticas un juego sencillito. Porque lo normal es que, al jugar con vuestra versión traducida, de vez en cuando aún aparezcan algunas cosas sin traducir, y posiblemente incluso a pesar de haberlas traducido en los scripts. A continuación explicaré algunas de las incidencias más comunes.

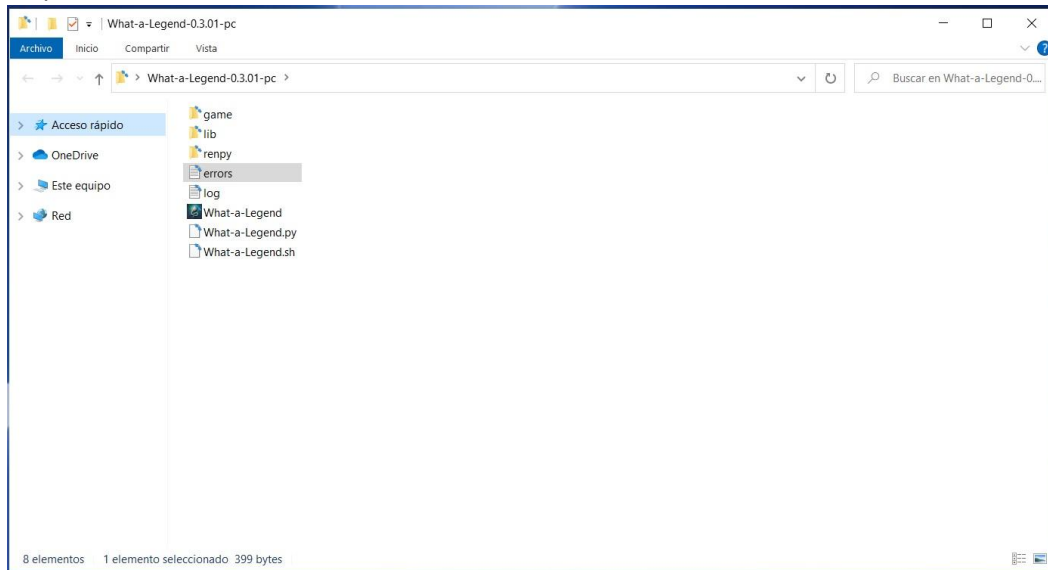
[|Arriba|](#)

3.1.- Detección de errores y bugs

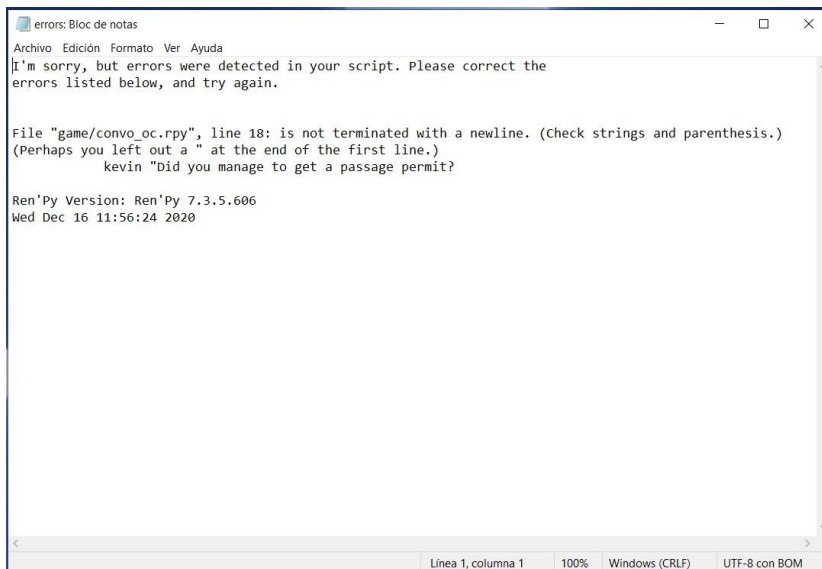
Empecemos por el principio, que debe ser familiarizarnos con la pantalla con la que Ren'Py nos avisa de que algo va mal. En un mundo ideal, los juegos se lanzarían sin bugs ni errores críticos, así que el primer pantallazo de error con el que se podría encontrarse un traductor debería ser el que nos avisa de que el Ren'Py SDK no ha podido generar los scripts de traducción. Lo cual solo puede significar que hemos cometido algún error al editar los scripts .rpy originales antes de generar la traducción.



Para ver qué es exactamente lo que ha fallado, deberíamos ir a la carpeta raíz del juego. Allí, junto al ejecutable, aparecerán varios archivos .txt.



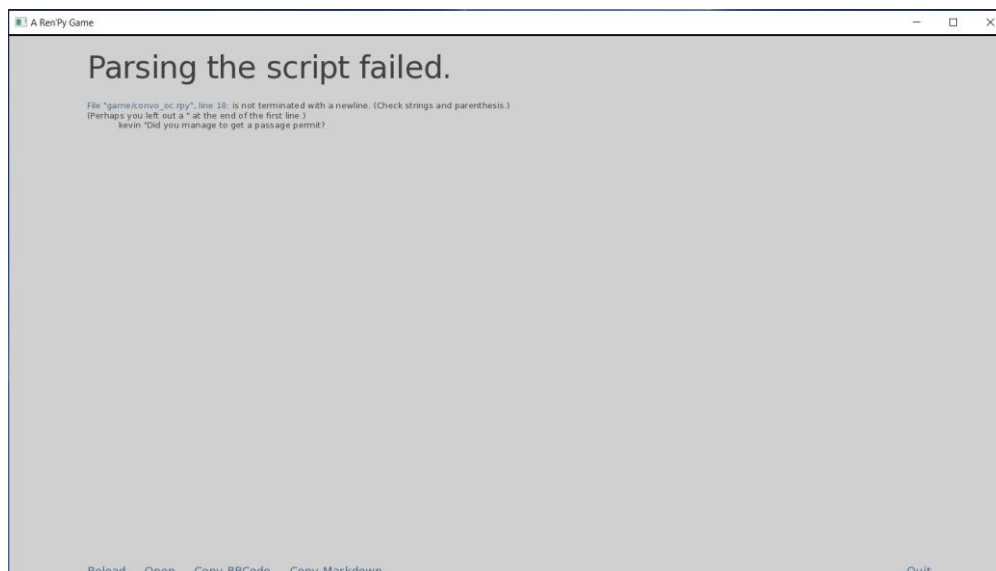
El que nos interesa es el llamado “errors.txt”, que podemos abrir con cualquier procesador de texto. En él veremos el mensaje de error con la indicación de la línea o líneas de código que han provocado el problema.



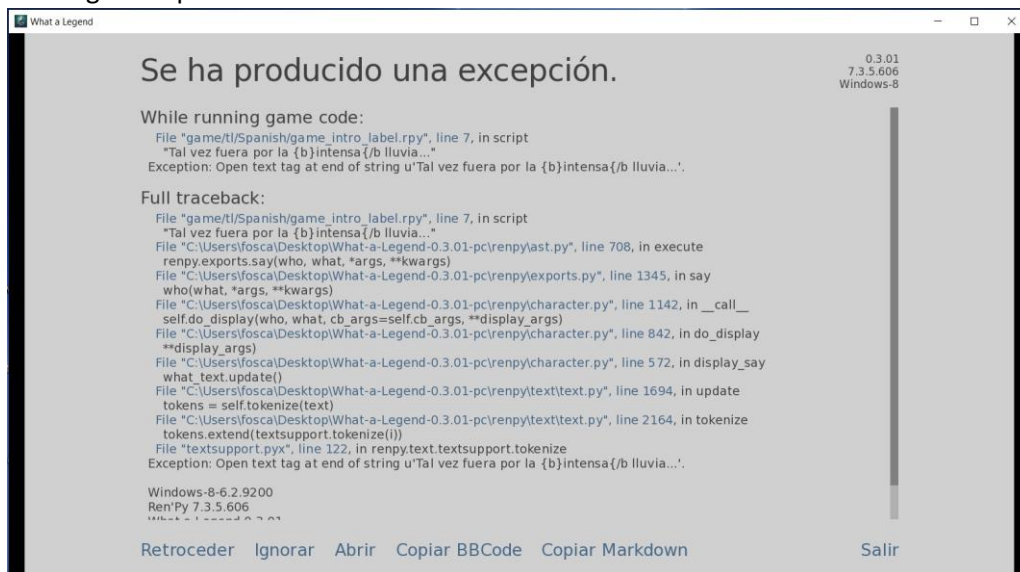
En este caso, algo tan habitual como habernos olvidado de cerrar un entrecorillado en la línea 18 del script “convo_oc”, pero podría haber sido por marcar una sangría con el tabulador. Solo tendríamos que ir a ese archivo, corregir el error, guardar los cambios y regresar al Ren’Py SDK para volver a intentar lo que estábamos haciendo.

Si veis algún mensaje más complejo, o si os surge incluso antes de haber editado nada, el problema suele deberse a que habéis usado una versión desactualizada de UnRen o del propio Ren’Py SDK.

Este mensaje también se habría mostrado si hubiéramos intentado arrancar el juego pinchando en su ejecutable. Entonces el pantallazo sería algo así, con la misma información que el documento “errors.txt”:



En otras ocasiones, el error no es crítico (en el sentido de que permite arrancar el juego y generar los archivos de traducción) pero sí da lugar a una excepción mientras se está jugando. Generalmente, si veis estos fallos se deberá a una etiqueta mal cerrada o a un problema con las variables. Es ahí cuando, en mitad del juego, nos surge esta pantalla con tantas líneas:



```

What a Legend
Se ha producido una excepción.
0.3.01
7.3.5.606
Windows-8

While running game code:
File "game/tl/Spanish/game_intro_label.rpy", line 7, in script
    "Tal vez fuera por la {b}intensa{/b lluvia..."
Exception: Open text tag at end of string u'Tal vez fuera por la {b}intensa{/b lluvia...'.

Full traceback:
File "game/tl/Spanish/game_intro_label.rpy", line 7, in script
    "Tal vez fuera por la {b}intensa{/b lluvia..."
File "C:\Users\fosca\Desktop\What-a-Legend-0.3.01-pc\renpy\ast.py", line 708, in execute
    renpy.exports.say(who, what, *args, **kwargs)
File "C:\Users\fosca\Desktop\What-a-Legend-0.3.01-pc\renpy\exports.py", line 1345, in say
    who(what, *args, **kwargs)
File "C:\Users\fosca\Desktop\What-a-Legend-0.3.01-pc\renpy\character.py", line 1142, in __call__
    self.do_display(who, what, cb_args=self.cb_args, **display_args)
File "C:\Users\fosca\Desktop\What-a-Legend-0.3.01-pc\renpy\character.py", line 842, in do_display
    **display_args)
File "C:\Users\fosca\Desktop\What-a-Legend-0.3.01-pc\renpy\character.py", line 572, in display_say
    what_text.update()
File "C:\Users\fosca\Desktop\What-a-Legend-0.3.01-pc\renpy\text\text.py", line 1694, in update
    tokens = self.tokenize(text)
File "C:\Users\fosca\Desktop\What-a-Legend-0.3.01-pc\renpy\text\text.py", line 2164, in tokenize
    tokens.extend(textsupport.tokenize(i))
File "textsupport.pyx", line 122, in renpy.text.textsupport.tokenize
Exception: Open text tag at end of string u'Tal vez fuera por la {b}intensa{/b lluvia...'.

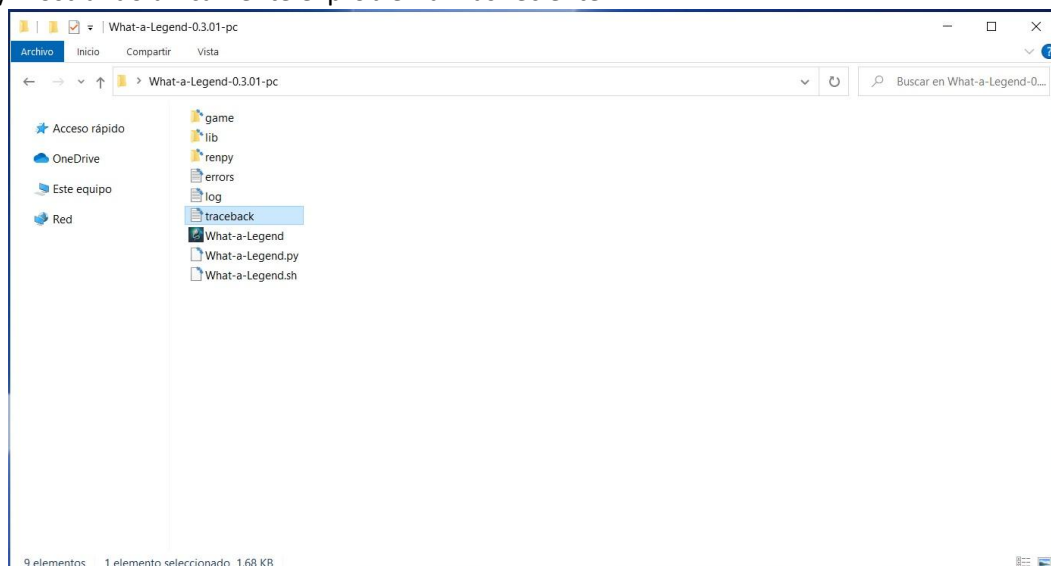
Windows-8-6.2.9200
Ren'Py 7.3.5.606

Retroceder Ignorar Abrir Copiar BBCode Copiar Markdown Salir
  
```

En esta pantalla se nos da la opción de intentar ignorar el fallo y seguir jugando (con la gran posibilidad de que el juego continúe dando más fallos posteriormente), o retroceder a un punto anterior del juego para guardar nuestra partida antes de tratar de arreglar lo que ocurre. También podemos copiar el mensaje con formato BBCode (para insertar en foros) o como Markdown (que permite adjuntarlo en Discord) por si queremos pedir ayuda.

Sin embargo, casi siempre se trata de pequeños fallos que, incluso aunque no hayamos originado nosotros, son relativamente fáciles de solucionar. Para ello, tenemos que fijarnos siempre en la primera línea, ya que ahí se indica la ubicación de la línea de código que ha generado el problema. En este caso, en el script de traducción “game_intro_label.rpy” (si os fijáis, está dentro de la carpeta “game/tl/Spanish”, por eso sé que es el script de traducción y no el original), en la línea 7, hay una etiqueta sin cerrar (se abrió una etiqueta con el comando `{b}` para que la palabra saliera en negrita pero no se cerró adecuadamente). Y en la última línea justo antes del apartado donde se indica el sistema operativo, la versión del programa y la fecha y hora del error, vuelve a aparecer el motivo del fallo (aunque ya sin referencia al script donde se ha producido). Fijándonos en esos dos detalles seremos capaces de localizar y corregir el fallo.

Ahora, al echar un vistazo a la carpeta raíz del juego, veríamos que se ha generado un archivo “traceback.txt” que contiene esta misma información. Tanto este documento “traceback.txt” como el “errors.txt” que veíamos antes se regeneran cada vez que se produce un fallo, borrando el contenido anterior y mostrando únicamente el problema más reciente.



En todos los casos, se trata de localizar y corregir el error, guardar los scripts y volver a intentar lo que estábamos haciendo, con la esperanza de que ese fuera el único fallo y no aparezcan más mensajes. Suerte.

[|Arriba|](#)

3.2.- Líneas con exceso de texto

Por lo general, las palabras y frases en español son más largas que en inglés. Esto puede hacer que la traducción de alguna línea exceda de los límites del cuadro de texto que se ve en pantalla, o que se superponga a otros elementos de la interfaz, entorpeciendo la lectura y a veces hasta la funcionalidad de los botones. La solución, triple, es bastante sencilla.

Por un lado, siempre podemos intentar reescribir la línea para decir lo mismo con menos letras. Si el resultado no nos convence, podemos aprovecharnos de la libertad que nos da Ren'Py a la hora de introducir el texto traducido de cada string. Y es que, aunque de entrada el Ren'Py SDK solo nos ofrezca una línea para traducir la línea original, podemos convertirla en dos o más siempre que respetemos los mandamientos de Ren'Py respecto a entrecomillados y tabulaciones. Por ejemplo:

```
# game/script.rpy:10
translate Spanish label_start_XXXXXXX
    # mc "Hello. My name is John."
    mc "Hola."
    mc "Me llamo John."
```

Así, aunque en el original el texto aparezca de una sola vez, en la traducción al español se dividirá en dos mensajes, sin mayor consecuencia. En realidad, podemos realizar cualquier variación que se nos ocurra, como añadir funciones, modificar variables, etc., que solo se aplicarán cuando se juegue en español. Y es que, aunque en principio la función de traducción está pensada para sustituir un texto por otro en un idioma distinto, lo que hace es sustituir una línea entera de código por lo que pongamos en el script de traducción, así que nada nos impide sustituir esa línea de código (que solo es un texto dicho por un personaje) por todo un bloque distinto de código.

La tercera opción, algo más compleja, es editar el archivo “gui.rpy”, donde entre otras muchas cosas se definen el tamaño de letra y el ancho del cuadro de texto. Para el tamaño de letra, buscamos el comando “define gui.text_size” y reducimos el número que venga al lado. Y para aumentar la anchura del cuadro de texto, incrementaríamos el número que aparece al lado del comando “define gui.dialogue_width”. Esto nos permitirá meter más caracteres en cada string y, por lo general, suele funcionar en todos los juegos, aunque puede romper la estética y es posible que os encontréis con alguna dificultad más, dependiendo del grado de personalización que el desarrollador del juego le haya dado a su interfaz. Todo es cuestión de investigar y probar.

[|Arriba|](#)

3.3.- Fuentes que no admiten caracteres especiales

Ligeramente relacionado con el anterior, en el sentido de que una de las soluciones pasa por editar el archivo “gui.rpy”, tenemos el caso de los juegos en los que la fuente original usada en los textos no contiene los caracteres típicos del español como las vocales acentuadas, la ñ, la ü o los signos de apertura de interrogaciones y exclamaciones. Por ello, al aplicar la traducción las palabras aparecerán sin esas letras o con un símbolo raro en su lugar.

La solución rápida es ir al archivo “gui.rpy” y buscar el comando “define gui.text_font” para eliminar la fuente usada por el juego y sustituirla por la DejaVuSans, que no requiere de ninguna acción extra ya que viene integrada en Ren'Py.

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

Si la fuente que no admite los caracteres del español es la usada para los nombres de los personajes, o alguna otra específica para menús, podéis localizarla en ese mismo script y modificarla de la misma manera.

Otra posibilidad es fijar un tipo de letra que nos guste más y sí acepte esos símbolos, en cuyo caso, además de editar el archivo “gui.rpy” como acabamos de ver pero con el nombre de la fuente elegida en vez de la DejaVuSans, tendríamos que incluir en nuestro parche el archivo .ttf de dicha fuente, para que quede guardado dentro de la carpeta “game”.

Lógicamente, las dos opciones anteriores eliminan la opción de jugar con la fuente original. Por ello podemos optar por una solución intermedia, ligeramente más compleja, que es establecer una fuente concreta solo para la versión en español, de modo que la versión original se mostraría tal y como la ideó el creador del juego. Para ello, en el mismo archivo “gui.rpy” escribiríamos esto:

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

Donde “myfont.ttf” es la fuente que hayáis elegido. Si no es una de las admitidas de serie por Ren'Py, tendremos que incorporar el archivo .ttf a nuestro parche para que quede guardado en la carpeta “game/tl/Spanish”.

En todos los casos anteriores, debemos acordarnos de incluir el archivo “gui.rpy” editado en nuestro parche, para que sustituya al que tengan los jugadores que se hayan descargado el juego original dentro de su carpeta “game”.

Y, por último, podemos liarnos la manta a la cabeza y editar la fuente original con un programa de edición gráfica de fuentes para añadir los símbolos que faltan. Obviamente, en este caso también tendríamos que incluir el .ttf resultante en nuestro parche para que sustituya al original, pero no hará falta tocar nada en el archivo “gui.rpy”.

[|Arriba|](#)

3.4.- Traducción de imágenes

Una de las cosas más laboriosas de hacer es traducir imágenes con texto. En ocasiones, los creadores del juego deciden que una parte importante de la acción se base en un mensaje de móvil, por ejemplo, o cualquier otro medio visual (un pantallazo en el ordenador, un titular de periódico, un cartel en la pared o mil cosas más). Aunque Ren'Py ofrece soluciones de programación para escribir esos textos en los scripts y, por tanto, facilitar su traducción, suele ser más cómodo crear una imagen con el texto y ponerla en el juego.

Si tenemos la suerte de que, mientras en pantalla se muestra esa imagen con un texto en otro idioma, también hay algún diálogo, podemos limitarnos a añadir el texto al comentario del personaje, como si lo estuviera leyendo en voz alta, aunque en el original no lo haga:

```
# game/script.rpy:12  
translate Spanish label_start_XXXXXXX  
    # mc "Look. He sent me a message."  
    mc "Mira. Me envió un mensaje. Decía que..."
```

O también podríamos podemos “subtitular” el contenido de la imagen aprovechándonos de [una solución que explicamos](#) para cuando la string traducida no cabía en el cuadro de texto: dividir la traducción de esa string en varias líneas para escribir en ellas la traducción del mensaje de la imagen, poniéndola en cursiva o de algún otro modo que facilite su identificación.

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

```
# mc "Look. He sent me a message."
mc "Mira. Me envió un mensaje."
"{i}(Esta es la traducción del mensaje que se ve en la imagen.){/i}"
```

Si no tenemos ninguna línea de diálogo asociada a la imagen, podemos crear una en blanco en el script original. Para ello, el primer paso es ir al script original, buscar la línea de programación en la que creemos que se ordena mostrar la imagen (podemos orientarnos con las strings de texto cercanas) y, respetando la sangría, incluir una línea debajo en la que solo pongamos dos comillas, así: "" Luego generaríamos la traducción e incluiríamos el texto traducido de la imagen como traducción de esa nueva string en blanco. Por supuesto, para que la traducción se mostrara bien, tendríamos que incluir en nuestro parche el script original que hayamos modificado.

Sin embargo, a veces esta opción no es viable o no da buen resultado estético. En ese caso, la única solución es editar la imagen original (o crear una nueva) con algún programa de edición de imágenes como Photoshop, GIMP, o incluso Paint, si es algo sencillo. Primero iríamos al script original para fijarnos en su nombre (generalmente, lo que aparezca después del comando `show` o `scene`). Luego iríamos a buscarla dentro de la carpeta "game/images" y la editaríamos para poner el texto en español. En un mundo ideal, podríamos pedirle amablemente al creador del juego que nos facilitara la imagen base, sin texto, para ahorrarnos trabajo. Suerte con eso.

Una vez terminada la edición, si queremos que se pueda seguir jugando con normalidad en el idioma original incluso después de instalar nuestra traducción, no deberíamos sustituir la imagen original en esa carpeta "images", sino guardar una copia con nuestros cambios, con el mismo título y formato de imagen y en una ruta idéntica, pero dentro de la carpeta "tl/Spanish". Es decir, si la imagen original es "game/images/sms00.jpeg", debemos guardar la imagen traducida como "game/tl/Spanish/images/sms00.jpeg". Así, cuando estemos jugando con la traducción activada, al llegar al punto del script donde debe mostrar la imagen "sms00.jpeg", Ren'Py la buscará primero dentro de la subcarpeta "images" de la carpeta de traducción, y si no la encuentra mostrará la que haya en la carpeta "images" original. Y, lógicamente, si estamos jugando en el idioma original mostrará la original.

[|Arriba|](#)

3.5.- Traducción de variables de texto

Al hablar de las [limitaciones](#) del Ren'Py SDK a la hora de extraer todas las strings traducibles del juego, ya comentamos que las variables que contienen texto no son detectadas automáticamente por el programa de extracción. Independientemente de cómo consigamos incluirlas en los scripts de traducción, el problema es que muchas veces esa traducción no aparecerá luego en pantalla durante el juego, cuando debería.

Cuando una variable de texto vaya a aparecer en pantalla, lo hará incrustada en una string. La reconoceremos porque, dentro de la string, aparecerá un objeto entre corchetes con el nombre de la variable. A veces la string consistirá exclusivamente en ese objeto, y a veces irá dentro de una frase. Por ejemplo, en un script original del juego "City of Broken Dreamers" nos encontramos con esto:

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

En este caso, el autor del juego ha decidido que un personaje nos dirija un insulto que irá variando aleatoriamente en cada partida. Para ello, en otro script (a veces hay que investigar mucho), ha definido una variable que consiste en una lista de insultos.

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

La función de la línea 57 seleccionará aleatoriamente un insulto de la lista de la línea 140, y ese insulto elegido será el valor de la variable "insult" (que se ha definido en blanco en la línea 139). Luego, en la string de la línea 58, el personaje proferirá dicho insulto.

Por lo tanto, lo primero sería traducir el listado de insultos de la línea 140, que como veis está formado por varias strings, una por insulto. Echándole imaginación, traducimos cada uno de ellos con los comandos `old` y `new` y, en el script de traducción, añadimos el apéndice `!t` dentro del objeto con el nombre de la variable:

```
571 # game/chapter1/chlsonja.rpy:58
572 translate Spanish chlsonja_cfc9ab60:
573
574     # "Unknown Woman" "Come on now, [insult]."
575     "Desconocida" "Venga ya, [insult!t]."
```

Así, al jugar en el idioma original la variable se verá en ese idioma, ya que no hemos modificado el script original, y al jugar con la traducción se verá el contenido traducido. Aunque hubiéramos traducido los insultos, si no incluyéramos el apéndice `!t` en esa string, al jugar en español nos seguiría apareciendo el insulto en inglés, ya que nuestra string traducida seguiría pidiendo a Ren'Py que nos mostrara la variable original `[insult]`, no su traducción.

Ojo: esto hace referencia a auténticas variables de texto, no a los nombres de los personajes. Y es que, en ocasiones, por comodidad a la hora de redactar las strings, los autores de los juegos sustituyen los nombres completos de los personajes por la definición de su variable "character". Lo distinguiréis porque la variable que veis incrustada en la string es la misma que veis al comienzo de las líneas de diálogo de dicho personaje, antes del entrecorriente. En ese caso, la traducción de la variable (si es necesaria) se realiza desde el momento en que se define al personaje y no hace falta tocar nada en las strings de traducción.

Por ejemplo, en el juego "Deliverance" hay varios personajes que, en lugar de por un nombre propio, son conocidos por un sustantivo común, traducible.

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

En este caso, la variable `[li]` hace referencia a uno de ellos (Lieutenant – Teniente), pero al tratarse de un personaje Ren'Py se encarga por sí mismo de mostrar el nombre original o la traducción según sea necesario, por lo que cuando esa variable aparece en una string no hace falta añadir el apéndice `!t`, solo hay que tener traducido su valor con los comandos `old` y `new`.

```
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."
```

Con estas nociones no deberíais tener problemas para conseguir que todo el texto traducible del juego apareciera traducido.

Lo que viene a continuación es una solución de urgencia que solo debe usarse como último recurso.

Hay juegos bastante complejos, y en ocasiones no seremos capaces de ver el texto debidamente traducido porque no sabemos localizar la string que contiene el objeto con la variable (puede pasas sobre todo en pantallas de menús personalizados, inventarios, etc.). La solución de urgencia es editar los scripts originales e introducir esa string que se nos resiste entre paréntesis y con un doble guion bajo delante:

```
define nombre_de_variable = __ ("...").
```

Esto por un lado, permitirá que el Ren'Py SDK extraiga la string para traducir, y por otro, al usar el doble guion bajo, en pantalla siempre se mostrará la traducción... también cuando se esté jugando con el idioma original. Es una lástima, pero hay veces en las que no se ve otra opción. Eso sí: es posible que, al hacer esto, generemos algún problema de lógica interna en el juego, si el valor de esa variable se utiliza en alguna expresión de las que determinan las rutas o pasajes del juego a mostrar. En esos casos, para evitar bugs e incoherencias, deberemos incluir la variable entre los símbolos `__ ()` también en esas expresiones.

Por ejemplo, imaginemos que tenemos una variable denominada “fruit” cuyo valor original es “Apple”. En algún lugar del código está su definición:

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

Y más adelante, a lo largo del juego, podemos cambiar esa fruta por otra, de modo que en el script nos aparecerá algo así:

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

Imaginemos que hemos encontrado esas dos strings y las hemos traducido con los comandos `old` y `new` en los script de traducción:

```
translate Spanish strings:
```

```
old "Apple"
new "Manzana"

old "Orange"
new "Naranja"
```

Pero en el juego hay una pantalla en la que se muestra un listado de diversos artículos en nuestro poder, uno de los cuales es la variable `[fruit]`, y por la razón que sea no somos capaces de encontrar en los scripts las líneas de código que corresponden a esa pantalla para extraer la string concreta y traducirla en los scripts de traducción por `[fruit!t]`, de modo que incluso jugando en español en pantalla nos aparece “Apple” u “Orange”. La solución bruta sería extraer esas strings con el doble guion bajo:

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

Así, cada vez que en los scripts originales se cite a la variable `[fruit]`, el valor que aparecerá en pantalla será “Manzana” o “Naranja” aun que no estemos jugando con la traducción. Pero puede ocurrir que esa variable se use en algún momento para determinar qué diálogo se va a mostrar en función de la fruta que tengamos. Algo así:

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

Si no tocáramos nada, el juego se rompería al llegar a esta parte del script porque el valor de la variable sería “Manzana” o “Naranja” en todos los idiomas, no “Apple” u “Orange”. Puede que no fuera un error crítico, y dependiendo de la programación el jugador tal vez no notara nada o, como mucho, un pequeño salto o incoherencia en los diálogos, pero desde luego no es así como el creador del juego pretendía que se viera su trabajo. Para solventarlo, también deberíamos utilizar los símbolos `__()` en la expresión lógica del script original:

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

Si hubiéramos conseguido traducir correctamente la variable, esta edición no sería necesaria, ya que la expresión lógica funcionaría con su valor en el idioma original aunque en pantalla se mostrara la traducción.

[\[Arriba\]](#)

3.5.- Polisemia: traducciones distintas para palabras iguales

Este tema ya surgió al hablar de [las dos funciones de traducción](#) que usa Ren'Py. Las strings que se traducen directamente, sin código de encriptación, solo admiten una traducción: de hecho, si intentamos usar dos distintas para una misma string, Ren'Py no nos permitirá abrir el juego y en [su mensaje de error](#) nos indicará que existe una traducción duplicada. Para poder arrancar el juego, tendremos que borrar una de ellas.

Pero, como decía en aquel ejemplo, a veces nos encontramos con strings idénticas que, por contexto, necesitan traducciones distintas, como la palabra "Right". La solución es editar el script original para conseguir que esas string idénticas dejen de ser idénticas, sin que eso afecte al juego en el idioma original. Y para eso solo hace falta utilizar el símbolo #.

Como ya vimos, todo lo que haya a la derecha de un símbolo # no va a aparecer en pantalla y, en principio, es descartado por Ren'Py en todos sus procesos... salvo que lo introduzcamos en una etiqueta. Las etiquetas son comandos incluidos en las strings mediante los símbolos { } que generalmente se usan para cambiar el tamaño y el color de letra, resaltar la frase en negrita o cursiva, etc. Es decir, Ren'Py lee estas etiquetas como parte de la string que las contiene y ejecuta lo que le ordenen, pero su contenido textual no se muestra en pantalla. De modo que, si en una de esas etiquetas incluimos un texto detrás del símbolo # (que Ren'Py usa para saber que no debe mostrar nada de lo que aparezca a su derecha), tenemos una string diferente a la que no tiene etiqueta pero que en pantalla se verá exactamente igual.

Por lo tanto, lo que haremos será buscar la string que queremos traducir de forma distinta e incorporarle una etiqueta en la que escribiremos algo que nos permita luego identificarla, como por ejemplo la traducción que queremos aplicarle. Así, a la hora de leer y extraer strings, el texto "Right", el texto "Right{#Derecha}" y el texto "Right{#Derecho}" dejan de ser iguales para Ren'Py, aunque en pantalla sí se seguirán viendo iguales. Ya solo es cuestión de regenerar los scripts de traducción (o incluir manualmente en ellos con el comando `old` esas nuevas strings "etiquetadas") y adjudicarles con el comando `new` una traducción distinta a la que tiene la string original. Al final, tendríamos algo así:

```
translate Spanish strings:
```

```
old "Right"  
new "Correcto"  
  
old "Right{#Derecha}"  
new "Derecha"  
  
old "Right{#Derecho}"  
new "Derecho"
```

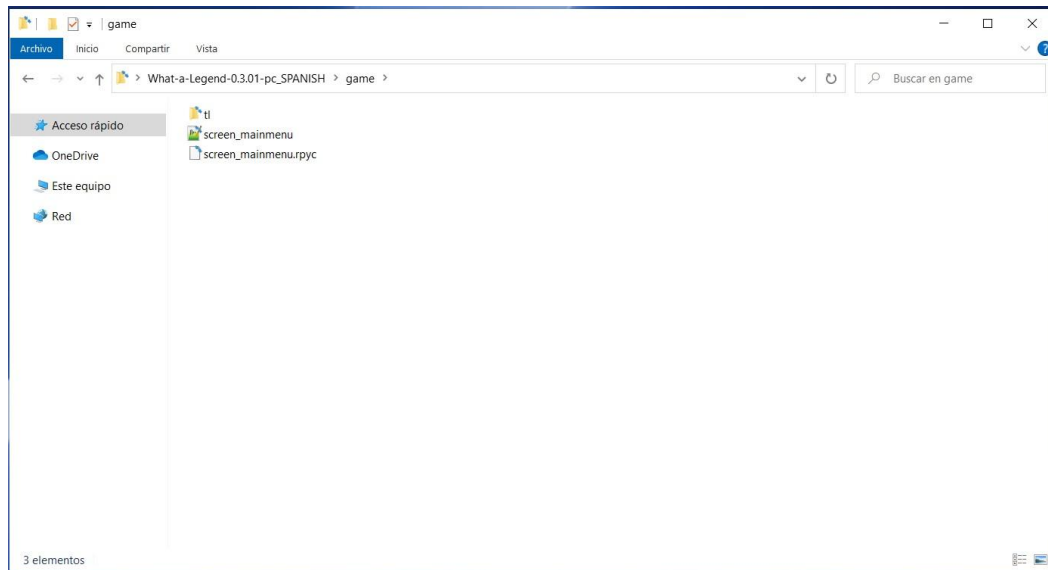
Recordad, en este caso no vale solamente con escribir esto en el script de traducción: también hay que editar el original para añadir la etiqueta a la string que queramos traducir de forma distinta para que, al jugar, Ren'Py busque la traducción de la string con etiqueta. Si no, seguirá aplicando la primera. Y, obviamente, acordaos de incluir esos scripts editados en el parche.

[|Arriba|](#)

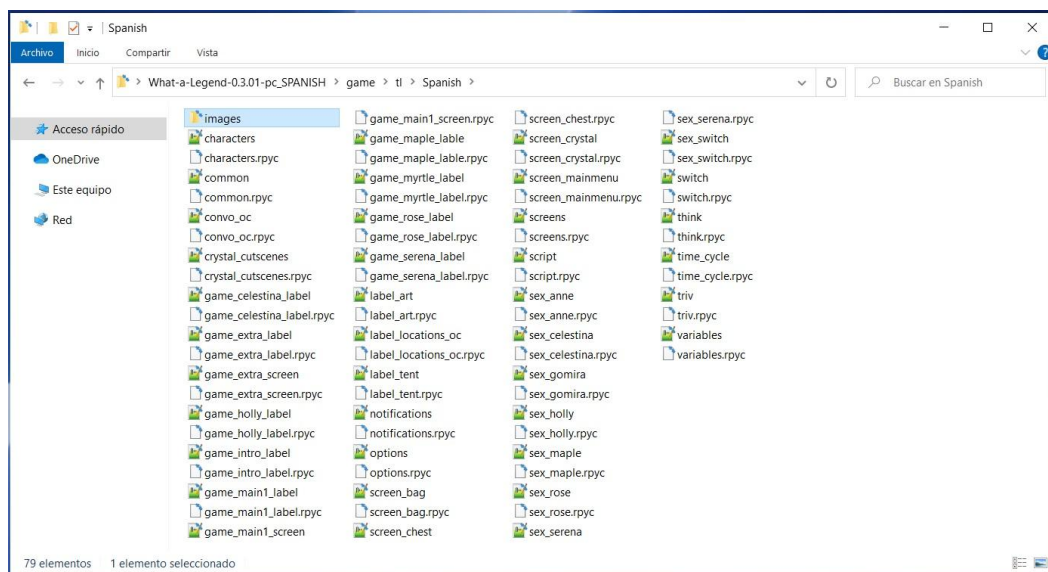
4.- EL PARCHE

Una vez realizada (y probada) nuestra traducción, llega el momento de compartirla con el mundo. Pero, para que funcione, nuestro parche debe respetar la organización de la carpeta original del juego. Es decir, los scripts `.rpy` originales que hayamos editado (junto con sus `.rpyc`) y los archivos `.ttf` de las fuentes utilizadas, en su caso, para solventar [los problemas del punto 3.3](#), deben terminar en la carpeta "game" del usuario final, que deberá reemplazar los originales por los de nuestro parche. Y, junto a ellos, debemos incluir una carpeta "tl", y dentro de ella una subcarpeta "Spanish" que contendrá nuestros scripts de traducción (y, en su caso, las imágenes y fuentes necesarias, según se ha ido explicando en la guía).

Para mí, lo más fácil es crear una carpeta “game” con todos esos archivos, de modo que el usuario solo tengo que pegarla dentro del directorio raíz de su propio juego y aceptar que se sustituyan todos los elementos coincidentes. Este es, por ejemplo, el contenido de la carpeta “game” de mi traducción de “What a Legend!”:



Como en este caso no hace falta editar ningún archivo más, solo aparece el script “screen_mainmenu.rpy”, que es donde incluyo la opción de cambio de idioma. Además, lógicamente, está la carpeta “tl” que tenía en mi ordenador con mi traducción completa, dentro de la cual se halla la subcarpeta “Spanish”, que tiene este aspecto:



Ahí podéis ver que, además de los scripts de traducción, se incluye una subcarpeta “images” con las imágenes que he tenido que traducir, que a su vez están clasificadas en carpetas con el mismo nombre que las carpetas que contienen las imágenes originales, como explicaba en el [punto 3.4](#).

Para que todo nuestro trabajo le llegue al jugador, basta incluir los scripts que se hayan editado para algo más que para extraer las strings con los símbolos `_ ()`. O sea, ediciones para facilitar traducciones de palabras polisémicas, archivos “gui.rpy” o “screens.rpy” retocados para cambiar el idioma o el tipo de letra, etc. No obstante, en mis parches suelo incluir absolutamente todos los scripts originales editados (también los que solo he modificado para que el Ren'Py SDK extraiga todas las strings), por si algún otro traductor a otro idioma quiere aprovecharlos y ahorrarse un trabajo. Pero en el caso de “What a Legend!” los creadores del juego ya escriben su código con esos símbolos `_ ()`, por lo que no hay que hacer nada.

Para terminar, un último detalle. Recomiendo incluir **siempre** en el parche los archivos .rpyc de la carpeta “game” (es decir, los que corresponden a los scripts en formato .rpy que hayamos editado). Si no los hemos borrado en ningún momento desde que descargamos el juego original (o desde que los extrajimos con el UnRen para poder hacer la traducción), estos .rpyc respetarán el AST con el que fueron compilados originalmente por el desarrollador del juego, aunque nosotros les hayamos introducido cambios al editar sus .rpy base. Así, cuando el jugador se descargue nuestro parche y reemplace los .rpyc que tiene en su ordenador (que, si no los ha borrado nunca, son los mismos que teníamos nosotros al principio), la estructura básica del AST de estos nuevos .rpyc será la misma y, en principio, no debería tener problemas para retomar partidas que tuviera guardadas de la versión original del juego.

¿Pasaría algo si no incluyéramos en el parche los scripts en formato .rpyc? Pues depende. Por regla general, si el juego viene empaquetado como recomienda el Ren'Py SDK (con los archivos .rpy y .rpyc sueltos dentro de la carpeta “game”, como en el caso de “What a Legend!”) no debería haber ningún problema: nuestros .rpy modificados sustituirán a los originales que tenga el jugador en su ordenador y, cuando arranque el juego, sus .rpyc serán actualizados con los cambios de nuestros .rpy en un proceso que no debería romper nada... suponiendo que el jugador nunca hubiera borrado los .rpyc originales del juego (si los ha borrado, hagamos lo que hagamos podrían surgir fallos al cargar partidas antiguas, pero ya es responsabilidad del jugador).

El problema realmente importante puede surgir cuando los scripts del juego original están comprimidos en formato .rpa: si solo incluimos los .rpy editados, Ren'Py creará unos .rpyc nuevos en el ordenador del jugador, pero corremos el riesgo de que no se generen con la misma estructura AST que tenían los que extrajimos del juego original, que fue con los que creamos la traducción. En esos casos, puede ocurrir que la función que ejecuta la traducción mediante código de encriptación (la del bloque de diálogos) no detecte la traducción existente por esa diferencia en el AST, de modo que los menús del juego y las opciones se verían en español (porque se traducen directamente) pero los diálogos de esos scripts editados se seguirían mostrando en el idioma original. Es un error extraño pero que puede darse, sobre todo con juegos relativamente antiguos que quizás fueron compilados originalmente con otra versión del Ren'Py SDK, y que evitaremos incluyendo en nuestro parche los archivos .rpy y .rpyc: aunque editados y actualizados, mantienen la estructura de los originales que extrajimos con el UnRen. Más vale prevenir.

[|Arriba|](#)