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

moskys#0576

¡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, si bien no realiza la traducción propiamente dicha. Así que, después de unos tres años traduciendo manualmente al castellano (y para PC) varios juegos de diversa complejidad, en diciembre de 2020 me animé a escribir la primera edición de 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.

Ahora, un año después, y con más conocimientos prácticos, reedito la guía ampliada y mejorada (espero). Y si bien está pensada para traductores no oficiales, el método que se describe es obviamente aplicable también a aquellos desarrolladores que quieran lanzar su juego en más de un idioma.

Tened en cuenta, eso sí, que aquí solo voy a explicar cómo conseguir los archivos en los que irá la traducción y cómo arreglar luego algunos problemas que podéis encontraros para conseguir que todo salga perfectamente traducido. Si lo que estáis buscando es una herramienta que realice la traducción propiamente dicha tendréis que buscarla en otro lado. Esta guía os permitirá comprender mejor el proceso y aprender a mejorar vuestras traducciones, sean manuales o automáticas, pero no os enseñará a traducir.

HERRAMIENTAS IMPRESCINDIBLES (Y GRATUITAS) – versiones a 31/12/2021

- **Ren'Py SDK 7.4.11 ->** https://www.renpy.org/latest.html
- **UnRen v.0.11 ->** https://f95zone.to/threads/mod-update-of-the-tool-unren-v-0-11v2-old-and-new-unren-windowed.92717/ *(link a un foro con contenido para adultos)*
- **Editor de textos (programas sugeridos):**
  - **Notepad++ 8.1.9.3 ->** https://notepad-plus-plus.org/downloads/
  - **Atom 1.58.0 ->** https://atom.io/ (es recomendable descargar la versión optimizada para Ren'Py que se puede obtener desde el propio Ren'Py SDK)

ÍNDICE DE CONTENIDOS

0-. LA TRADUCCIÓN EN 10 PASOS

Vaaaale, lo sé. La guía es muy larga, contiene mucha información escrita y es muy probable que tengas prisa por ponerte a traducir y todo esto te parezca excesivo, así que, a falta de videotutorial, aquí va un breve esquema de los principales pasos a dar para traducir un juego Ren'Py.

1. Abre la carpeta "game" del juego y comprueba si hay scripts con extensión .rpy

2. Si no los hay, usa unRen para extraerlos del fichero .rpa y/o descompilarlos a partir de los .rpyc

3. ¿Estás traduciendo una actualización? Pega ahora la traducción de la versión anterior

4. Revisa los scripts por si hubiera strings que no pudiera extraer el Ren'Py SDK

5. Abre el Ren'Py SDK y genera los scripts de traducción:
    1) Selecciona el juego que quieres traducir (si no aparece en la sección "Proyectos", busca su carpeta contenedora en preferencias > Carpeta de proyectos)
    2) Haz click en Generar Traducciones
    3) Escribe en "Idioma" el nombre del idioma al que quieres traducir el juego (y recuérdalo bien)
    4) Marca (o no) generar cadenas vacías
    5) Haz click en Generar Traducciones

6. Añade la opción de cambio de idioma editando la pantalla de preferencias o creando una nueva

7. Traduce todos los scripts que se han creado en la carpeta game/tl/ del nombre que pusieras en 5.3)

8. Juega con la traducción activada y comprueba que no haya errores:
    1) ¿El juego no arranca o se producen excepciones? Mira los archivos errors.txt y traceback.txt
    2) ¿El texto no cabe en pantalla?
    3) ¿Hay letras que no aparecen? Cambia el estilo
    4) ¿Qué hacemos con las imágenes que tienen texto?
    5) ¿Hay opciones que, aun estando traducidas, no tienen sentido? Podría ser por homonimia
    6) ¿Hay errores de concordancia gramatical?
    7) ¿Siguen saliendo palabras en el idioma original? Repite el paso 4, aunque tal vez se trate de variables interpoladas o concatenadas
    8) ¿Sigues sin ser capaz de conseguir que todo salga en tu idioma? Prueba un último recurso

9. ¿Quieres compartir tu traducción? Empaqueta tu trabajo en un parche

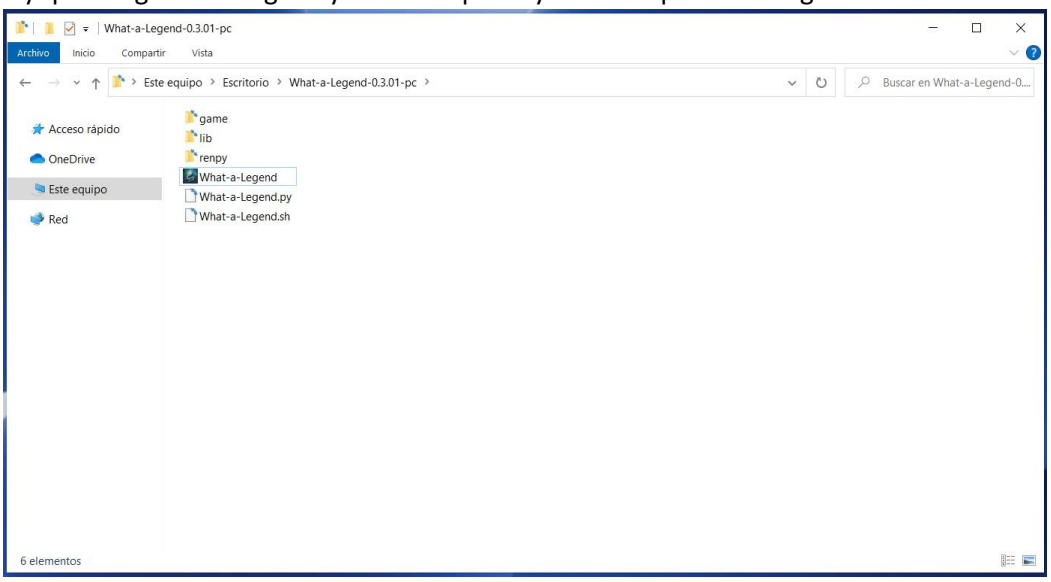
10. ¿Has hecho un parche? Sé buena persona y avisa al creador del juego.

|Índice|

1.- INTRODUCCIÓN

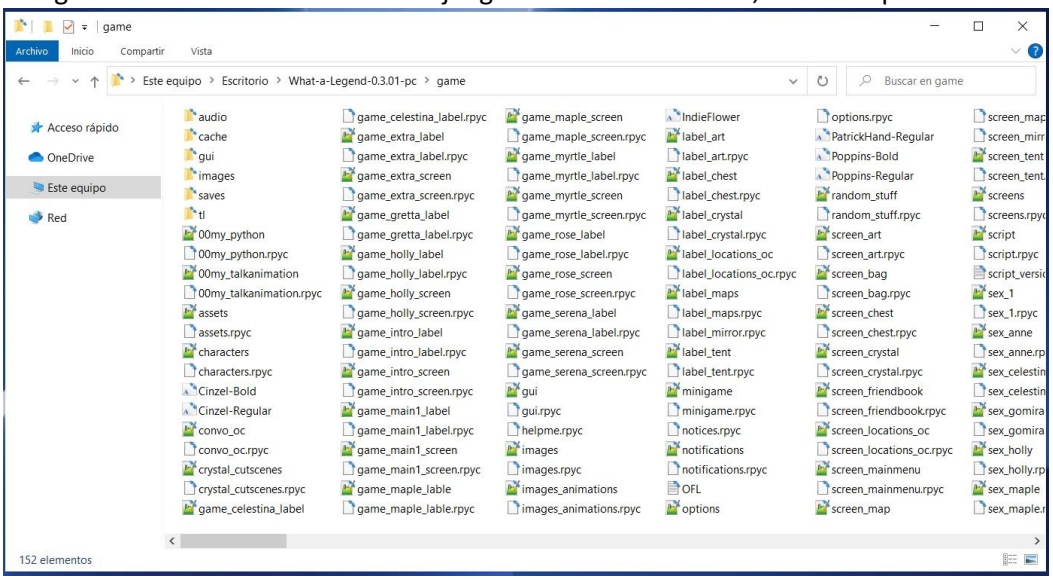
1.1.- Destripando un juego Ren'Py

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 de Ren'Py, generalmente comunes a todos los juegos, 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 publicar 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 extensión .rpy y otros con extensión .rpyc. El número de archivos solo depende de cómo organice su trabajo el creador del juego, ya que Ren'Py podría ejecutar perfectamente un juego que estuviera contenido en un solo archivo. Lo normal, como digo, es que haya menos que en este ejemplo, pero también podéis encontraros con juegos que tienen sus archivos en varias subcarpetas.

|Índice|

1.2.- Tres conceptos básicos y dos mandamientos

Esos 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 la extensión .rpy, específica 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 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."
```

Para empezar, diremos que todo lo que quede a la derecha de un símbolo # es "invisible" para Ren'Py al ejecutar el juego. Por ello los creadores suelen usarlo para escribir comentarios que no aparecerán en el juego pero que les sirven para orientarse en su código. Más adelante veremos la utilidad de ese símbolo. En la línea 3 vemos una palabra, **label**, de mucha importancia. Las labels (literalmente etiquetas) son porciones del guion. El tamaño de estas porciones depende del desarrollador del juego, pero generalmente cada una corresponde a una escena o a una parte concreta de ella, que se irán activando durante la partida. Por el lenguaje Python, todo lo que ocurre en una label se codifica con una sangría (denominada técnicamente indentación), y dentro de ella puede haber otros bloques indentados, siempre después de una línea acabada en dos puntos : .

> **Primer mandamiento de Ren'Py:** las indentaciones 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 indentación 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 contenido va a ver el jugador según las variables que haya activado o acumulado. Además, entre medias aparecen salpicadas algunas frases, que son los diálogos que aparecerán en pantalla. Esas frases entrecomilladas son las que tendremos que traducir, y se denominan **strings**, o cadenas de texto.

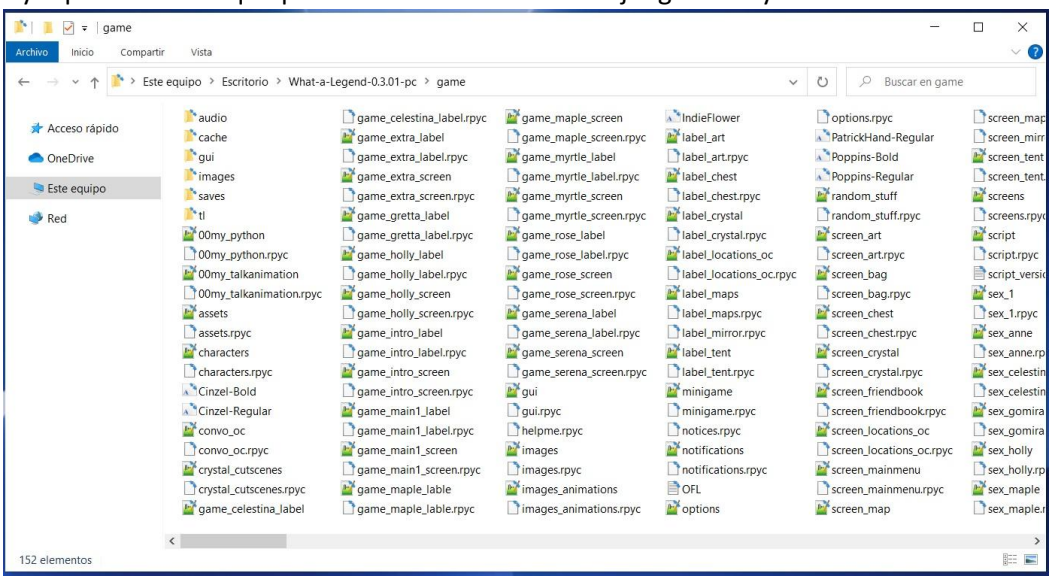
> **Segundo mandamiento de Ren'Py:** las strings se inician con unas comillas " y se cierran con otras comillas ". Una string con comillas sin cerrar (o sin abrir), o con comillas sueltas en medio de la string equivale a un juego que no arranca.

Si en una string queremos usar unas comillas que aparezcan escritas en la pantalla, las escribiremos así, con una barra delante: \" De esta forma, Ren'Py identifica que son parte del texto y no de la programación.

|Índice|

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: los archivos en formato .rpy son documentos de texto en los que el autor escribe el juego, pero luego Ren'Py ejecuta los .rpyc, que son una compilación de los .rpy. La primera vez que el desarrollador abre el juego en su entorno de creación, los algoritmos de Ren'Py asignan 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 se crean unos archivos .rpyc del mismo nombre que serán los que Ren'Py irá leyendo durante el juego. Para efectuar esa compilación inicial se genera un AST (árbol de sintaxis abstracta) que relaciona esas identificaciones y que Ren'Py usará como guía en las siguientes compilaciones.

Cada vez que se arranca el juego, Ren'Py busca archivos .rpy en la carpeta "game" y, si los hay (no sería necesario que los hubiera para que el juego funcionara), se prepara para compilarlos en un archivo .rpyc. Si ya existe un .rpyc con ese nombre, Ren'Py buscará en él referencias del AST para poder actualizarlo con las novedades que se hubieran introducido en el .rpy desde la última vez que se compiló. Simplificando mucho, imaginemos que la línea número 3 del archivo .rpy se compila inicialmente de forma que el AST le asigna un 3 en el archivo .rpyc. Si modificamos el .rpy y esa línea pasa a ser la número 7, Ren'Py se prepararía para compilarla con el 7, pero detectaría que ya está presente en el archivo .rpyc con el 3 y no la modificaría, manteniéndola con ese 3 en el rpyc. Así, en cada compilación se van añadiendo solo los elementos nuevos, adaptándolos para respetar lo preexistente. El resultado es que, aunque el script .rpy haya cambiado mucho desde su primera versión, su .rpyc estará compilado a partir de las referencias asignadas en el primer AST.

¿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 a partir del AST no serán iguales que las que había antes. En nuestro ejemplo absurdo, la línea 7 del .rpy que en el .rpyc borrado tenía asignado el número 3 porque venía arrastrada de versiones anteriores, al crearse un nuevo .rpyc podría pasar a ser identificada con el número 7. El juego arrancará y las partidas nuevas no se verán afectadas, pero funciones que usan las referencias del AST, como la carga de partidas guardadas que se grabaron con el juego ejecutando la versión anterior de los .rpyc, pueden dejar de funcionar. Y, en ocasiones, las traducciones también podrían dar problemas.

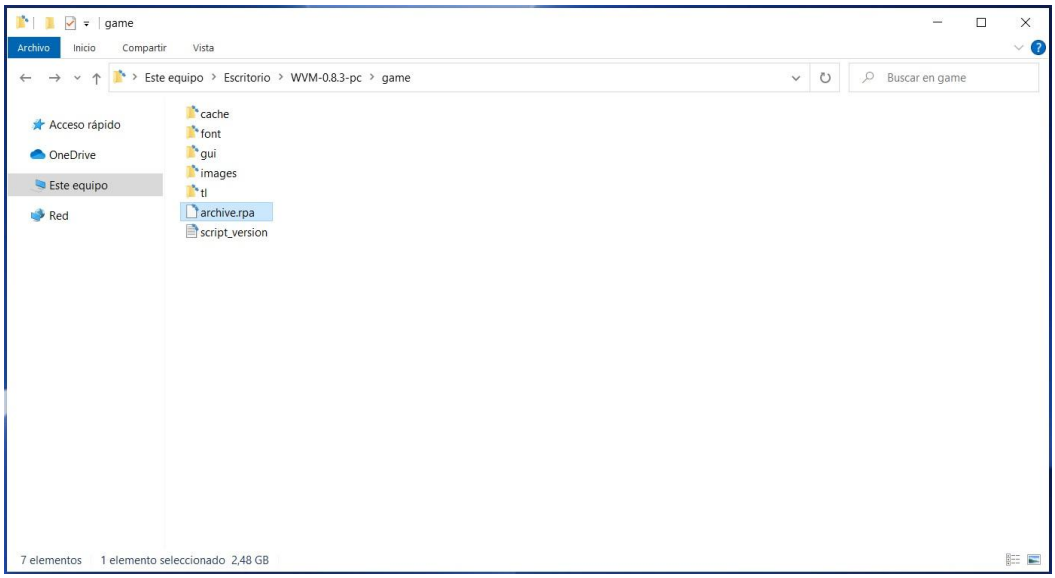
> **Tercer mandamiento de Ren'Py:** borrar archivos .rpyc del juego original puede generar problemas. No lo hagas si no tienes una buena excusa (y yo no sé ninguna).

|Índice|

1.4.- Archivos .rpa y UnRen

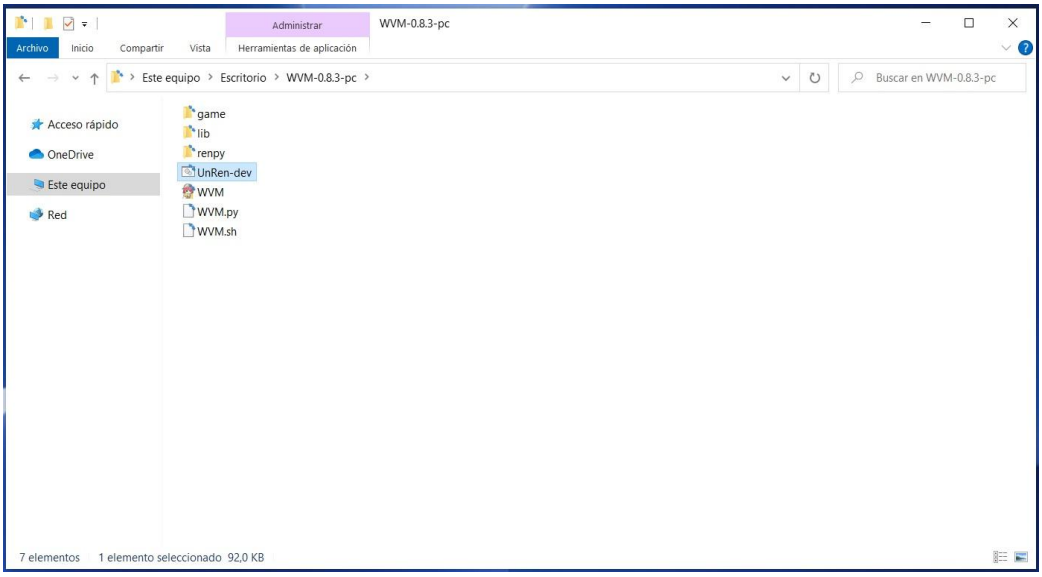
Puede pasar que, al abrir la carpeta game, no hayáis visto nada de lo que he comentado en el punto 1.1. Y es que, a la hora de empaquetar el juego para publicarlo, Ren'Py les ofrece a los desarrolladores varias opciones. La recomendada es incluir los scripts sueltos en formato .rpy y .rpyc, como hemos 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 con extensión .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, ya que los .rpyc no contienen ningún texto comprensible.

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:

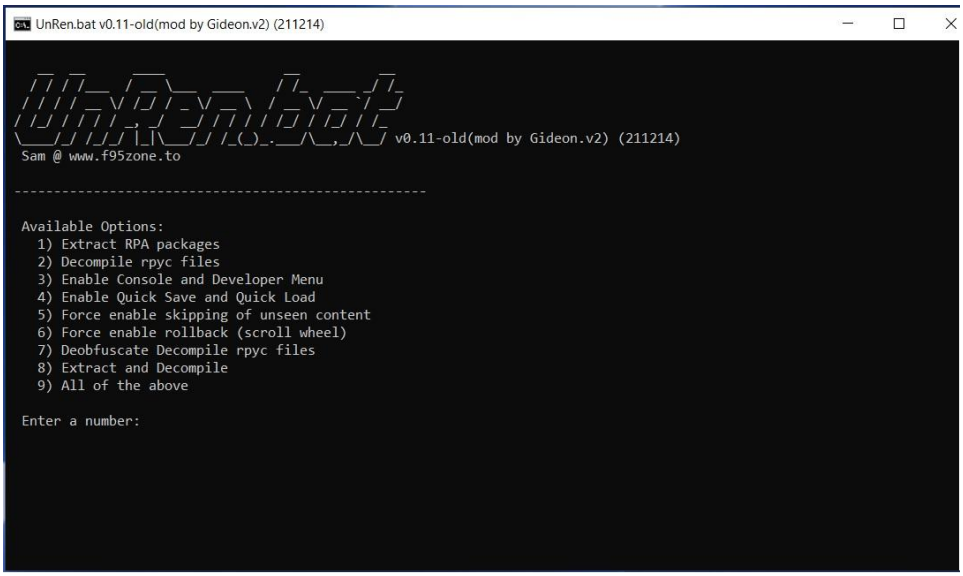


Como seguramente ya sepáis, el juego va a funcionar perfectamente tal y como está. Pero para poder traducirlo vamos a tener que dar algún rodeo extra, ya que **SIEMPRE** necesitamos los scripts en formato .rpy. Afortunadamente, existen varias herramientas que nos permiten realizar las tareas necesarias 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 descomprimimos 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:

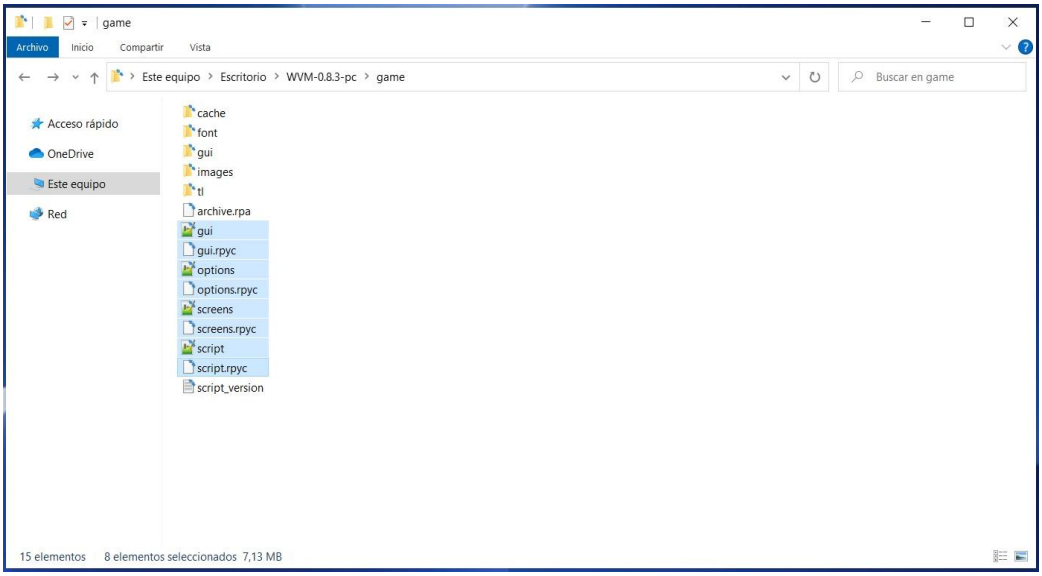


Como se puede observar, aquí solo vamos a tener que manejar el teclado. No tiene más complicación que pulsar la tecla correspondiente a lo que queramos hacer.

La opción 1) va a descomprimir los ficheros .rpa, extrayendo todo su contenido en la carpeta "game". Si después de este paso ya tenemos scripts en formato .rpy, genial: no necesitamos hacer nada más. Esos scripts serán los originales escritos por el creador del juego y contendrán los comentarios que hubiera anotado detrás del símbolo # (que pueden resultarnos útiles para poner en contexto alguna string que nos genere dudas sobre cómo traducirla). Pero puede darse el caso de que dentro de ese fichero .rpa solo hubiera scripts en formato .rpyc, con lo cual, después de concluir esa opción 1), tendremos que pedirle a UnRen que ejecute la opción 2), que consiste en descompilar los archivos .rpyc para crear unos scripts en formato .rpy que ya podríamos usar para traducir. Eso sí, en algunos juegos recientes será necesario forzar esa descompilación mediante la opción 7). Estos .rpy "descompilados" son generados por UnRen basándose en el AST que haya identificado en los .rpyc y contendrán la información básica para que el juego funcione y podamos traducirlo, pero no incluirán los comentarios del desarrollador, ya que estos no se compilan en los .rpyc.

Podemos acabar antes seleccionando la opción 8), que en un solo paso hace lo mismo que las anteriores; o la 9), que 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). Pero yo recomiendo ir paso a paso, porque si dentro del .rpa ya venían los scripts .rpy originales, después de ser extraídos acabarán sobrescritos con la versión básica creada por UnRen al forzar la descompilación de los .rpyc, y perderemos los posibles comentarios.

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. Y, dependiendo del juego, también pueden haberse añadido algunas carpetas que podrían contener scripts.

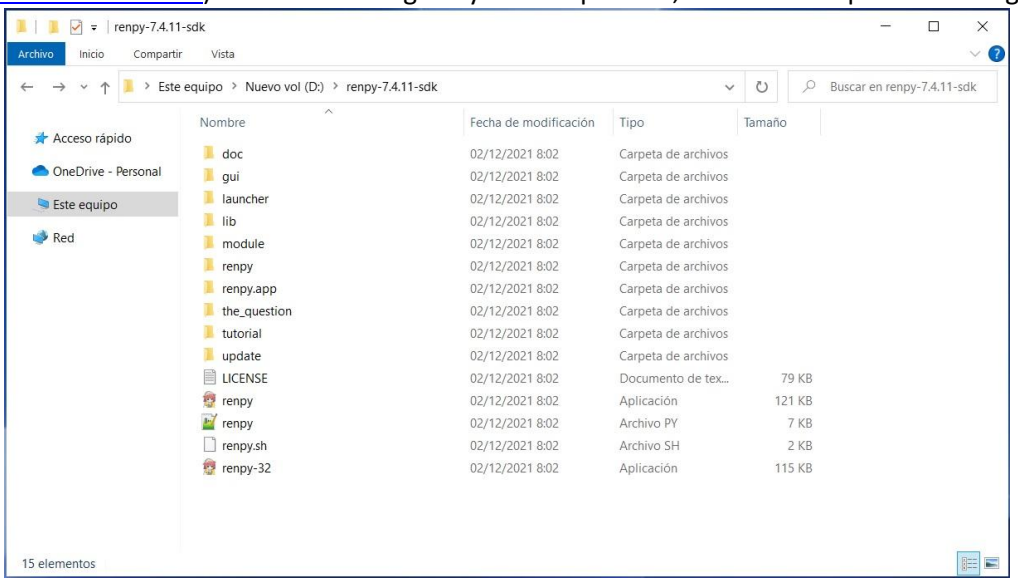


Ahora ya podríamos comenzar a traducir. Pero antes, un inciso para explicar lo que va a ocurrir con el juego. En este momento, dentro de la carpeta game hay scripts en formato .rpy, scripts en formato .rpyc y, además, dentro del fichero "archive.rpa" estarán esos mismos scripts .rpyc (y quizás hasta 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 (siempre que tengamos una forma de recuperarlo, por si acaso) el fichero "archive.rpa". Lo que nunca deberíamos borrar serían los .rpyc extraídos de ese fichero.

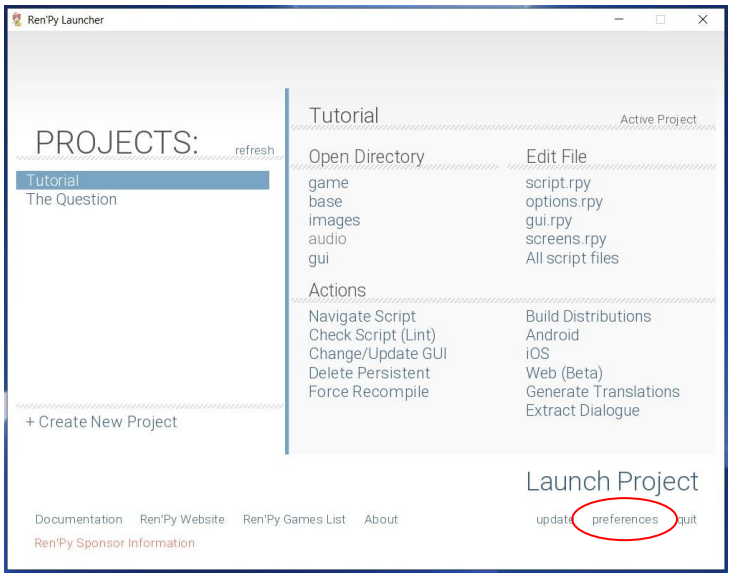
|Índice|

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 DE DESCARGA). Una vez descargado y descomprimido, al abrir su carpeta veréis algo así:

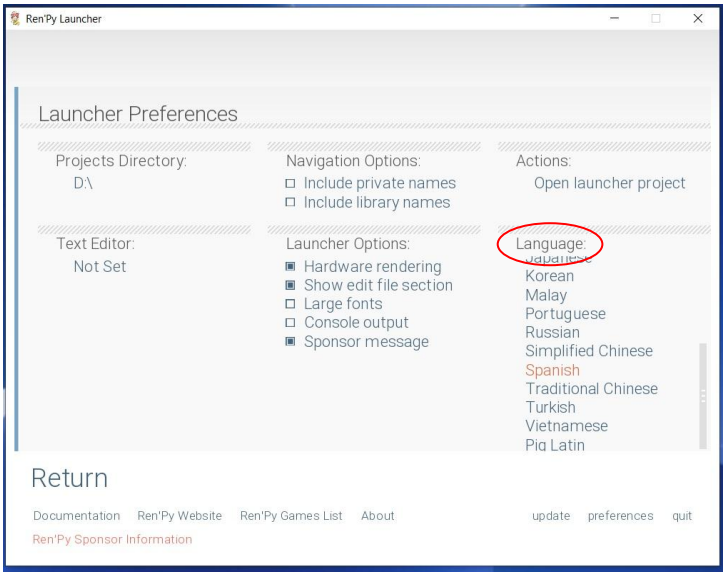


Es decir, una estructura similar a la de los juegos, con varias subcarpetas y, en el caso de la versión para Windows, dos ejecutables (uno genérico y otro para los sistemas antiguos de 32 bits). Ya solo es cuestión de abrir el ejecutable "renpy" correspondiente a vuestro sistema operativo y empezar a trabajar. Tras configurar una serie de parámetros, aparecerá la pantalla principal, generalmente en inglés. En el apartado "Projects" veremos que ya aparecen dos títulos: "Tutorial" y "The_Question", que son dos tutoriales para aprender a manejar Ren'Py. No viene mal echarles un vistazo, pero no aportan gran cosa a las traducciones.



Así que, antes de nada, y si no ha quedado ya configurado por defecto tras la instalación, vamos a cambiar el idioma para trabajar en español. Para empezar, habría que pinchar en "preferences" 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 en él, la pantalla cambiará de idioma. Ahora ya tendremos el programa funcionando en español y no hará falta que volvamos a cambiarlo.



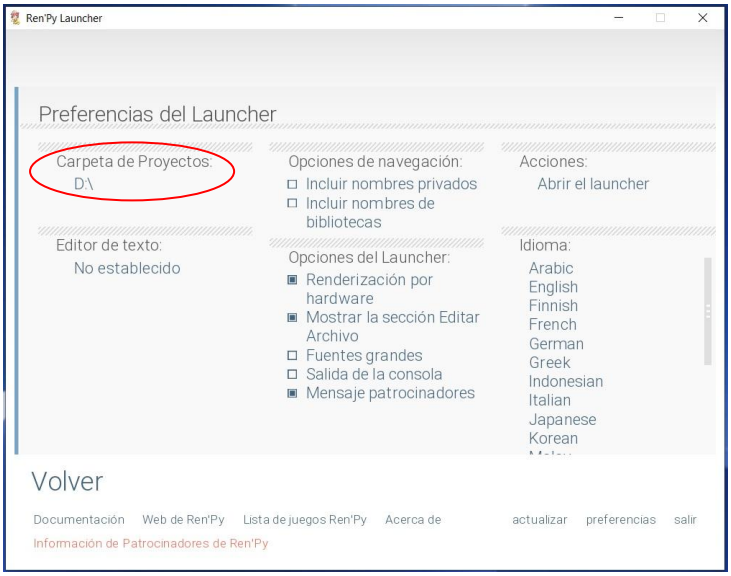
|Índice|

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. El primer impulso suele ser el de abrir esos scripts y comenzar a reescribir en nuestro idioma las strings que encontremos. Pero, si bien no tiene por qué pasar nada, al editar los scripts originales siempre aumentan los riesgos de generar un bug. Y, sobre todo, Ren'Py incorpora en todos sus juegos un sistema de apoyo a la traducción: una serie de funciones que ayudan a extraer los textos a traducir y hacen que las traducciones se muestren luego en su sitio. Aunque en ocasiones aún tendremos que hacer un mínimo trabajo de edición de los scripts originales, es un sistema bastante más limpio que la rescritura directa, y para aprender a utilizarlo y superar sus limitaciones es para lo que existe esta guía.

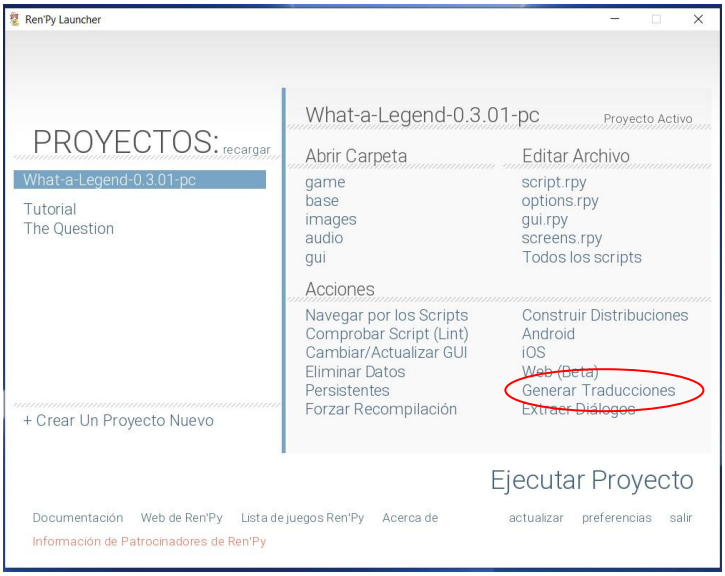
|Índice|

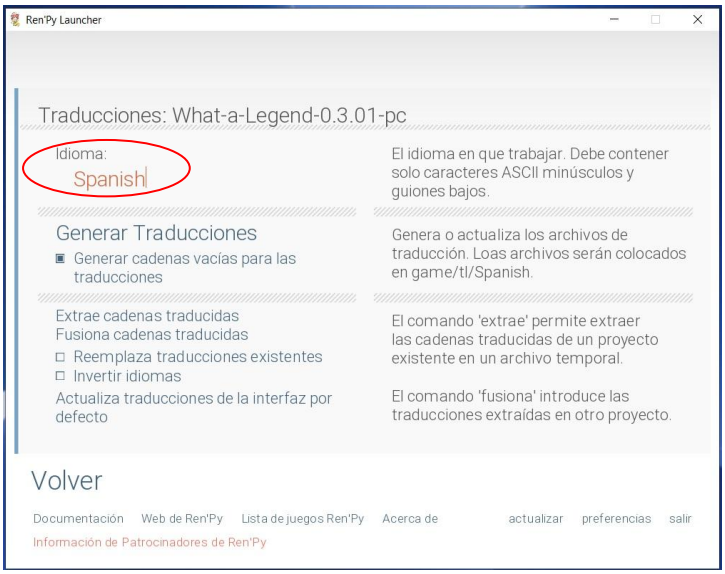
2.1.- Generando los archivos de traducción (y aprendiendo el cuarto mandamiento)



Una vez abierto el Ren’Py SDK, el primer paso es encontrar el juego que queremos traducir. Para ello, debemos ir a “preferencias” y, en el apartado “Carpeta de Proyectos”, seleccionar el directorio donde esté la carpeta raíz de dicho juego. 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 de la pantalla inicial aparecerán todos los juegos que haya en esa carpeta, además de los dos tutoriales que se incluyen en el Ren’Py SDK.

Ya solo es cuestión de seleccionar el juego que queremos traducir de entre todos los que haya en esa carpeta y pulsar el botón “Generar Traducciones” que hay en la parte derecha de la pantalla.





A continuación aparece una nueva 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, porque Ren’Py buscará ese término literalmente, respetando mayúsculas y minúsculas. 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 tendremos visible una línea con el texto original para guiarnos, si seleccionamos esa opción los scripts de traducción se generarán con unas strings en blanco para que escribamos directamente nuestra traducción. Si no la seleccionamos, los scripts se generarán con esas strings otra vez en el idioma original del juego y tendremos que borrar esas palabras y sustituirlas por la traducción.

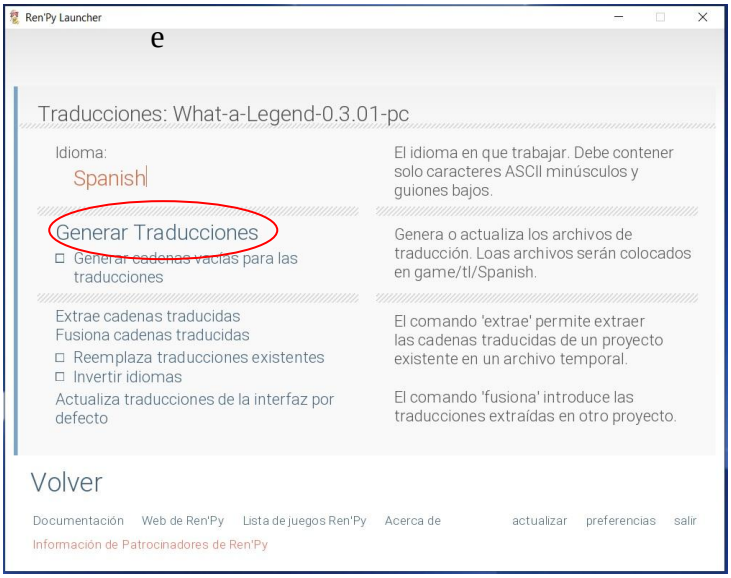
Puede parecer más cómodo (y de hecho lo es) generar las cadenas vacías, pero, si se nos olvida traducir alguna línea, al llegar a ella en el juego no aparecerá absolutamente ningún texto en pantalla. De la otra forma, al menos saldría el texto en el idioma original, algo que puede resultarnos útil mientras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

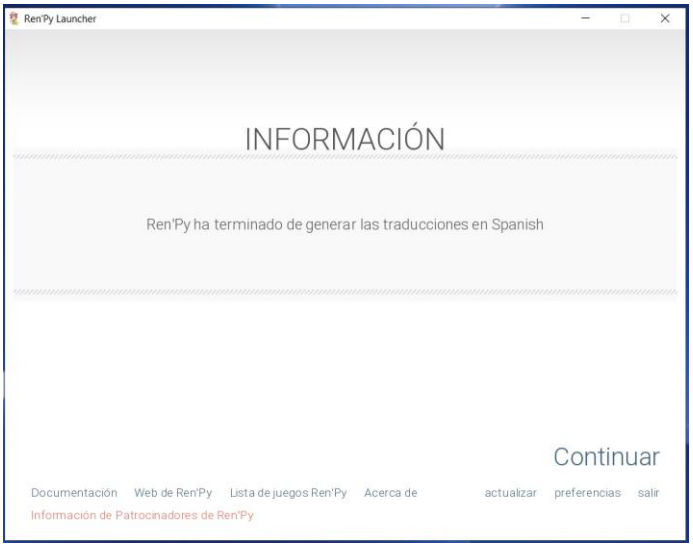
Aquí podemos ver una comparativa de cómo son inicialmente los archivos de traducción si marcamos esa casilla (izquierda) y si no la marcamos (derecha).

```
# 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."
```

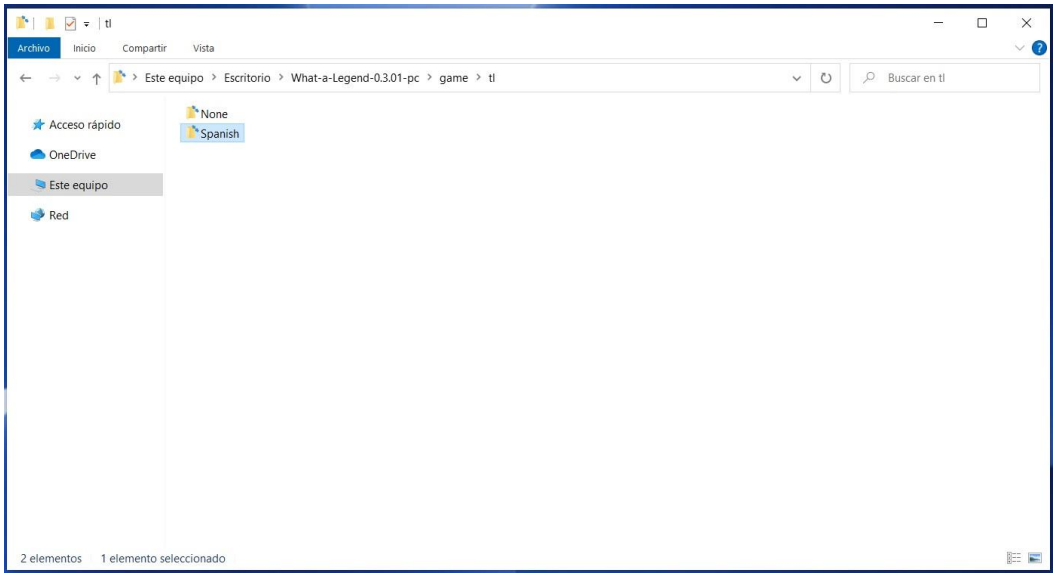
Ahora ya solo queda pinchar en el botón "Generar Traducciones". Como indica el texto informativo de la columna derecha (con error tipográfico incluido), al hacerlo se crearán unos archivos de traducción dentro de la carpeta "game/tl/Spanish" (el nombre de dicha carpeta será el que hayáis indicado en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



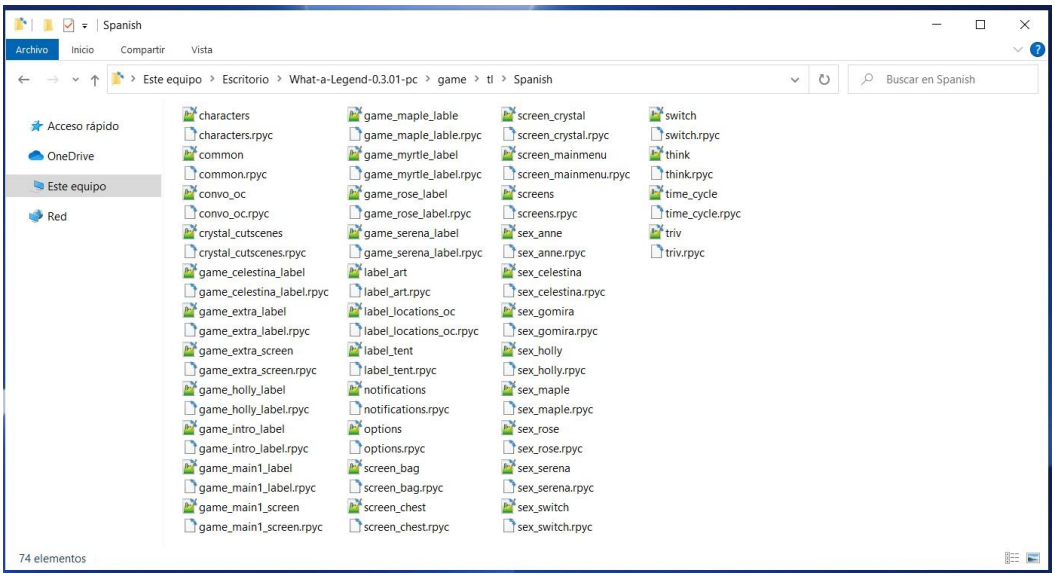


Si no hay ningún problema, cuando el Ren'Py SDK termine de hacer su magia nos aparecerá este mensaje. Podemos darle a "Continuar", para volver a la pantalla de inicio, o directamente salir del Ren'Py SDK.

Si ahora vamos a la carpeta “game” de nuestro juego, veremos que en la subcarpeta “tl” aparecen otras dos subcarpetas: una llamada “None” (que ya estaba antes ahí) y otra recién creada 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 en ellos algún texto traducible, ha generado en esta carpeta 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” (el script common.rpy original está dentro de la subcarpeta tl/None), 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, los menús de accesibilidad y del desarrollador, las pantallas de error, etc. 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 duplicar la traducción de una string, lo que nos generará un error.

Llegados a este punto, ya 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.

|Índice|

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

Algo que suele pasar la primera vez que empezamos a traducir es que, de repente, nos damos cuenta de que 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.

2.2.1- El cuerpo de diálogos y los códigos de encriptación: Las strings que forman el cuerpo de los diálogos son identificadas y extraídas sin problemas por el Ren'Py SDK. Pero lo que hace con ellas sí puede generarnos alguna molestia, y es que las cifra para ahorrar memoria: usando el método de encriptación MD5 hexadecimal reducido 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 de la línea. Veámoslo recordando 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 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 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.

```
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 ""
```

Pues bien, al generar el script de traducción con cadenas vacías, Ren'Py ha transformado toda esa información en esto que vemos en la imagen de la izquierda: ha eliminado todas esas funciones y comandos de programación para extraer solo las líneas que contienen strings de texto, generando un bloque de cuatro líneas para cada una de esas strings. Distinguiremos los bloques por la indentación: cuando veamos una línea que comienza en el margen izquierdo, sabremos que ahí empieza un bloque de traducción para una string del script original.

Ahora analicemos en qué consiste cada uno de estos bloques:

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

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

Como vimos, las líneas que empiezan con un # son informativas y podríamos borrarlas sin consecuencias, aunque no lo recomiendo. 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 el texto original, que Ren'Py nos incluye como referencia para saber qué es lo que debemos traducir. Y la última línea es donde vamos a escribir nuestra traducción (en principio solo traduciremos el texto que haya entre comillas en la línea superior, el resto lo dejamos tal y como está). Si hubiéramos generado la traducción sin cadenas vacías, aquí nos aparecería nuevamente el texto en el idioma original del juego, y tendríamos que borrarlo.

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 con el Ren'Py SDK) 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, `kevin "STOP!"` equivale a fdd309da).

Si dentro de esa misma label "convo_gate" hubiera otra string idéntica (otro "STOP!" dicho por el personaje "kevin"), Ren'Py debería asignarle el mismo código de encriptación. Pero esto generaría conflictos, así que al código de la segunda le añadirá un `_1` al final, si hubiera otra igual le pondría un `_2` y así sucesivamente. Así pues, **los códigos de encriptación son únicos para cada string, incluso para strings literalmente idénticas**, y esto hará que todas las strings del cuerpo de diálogos aparezcan en los scripts de traducción. Si hay diez strings idénticas, tendremos que escribir lo mismo diez veces: lo que traducimos son diez códigos distintos.

¿Pero qué pasa si vamos ahora al script original y donde pone "STOP!" escribimos "Stop!" o cambiamos el personaje que lo dice? Pues que la traducción que tengamos dejará de servirnos, porque ahora el código MD5 de esa línea será distinto y Ren'Py no lo encontrará en los scripts de traducción, así que mostrará el texto en el idioma original. Esto supone una cierta molestia a la hora de traducir nuevas versiones de juegos en desarrollo, donde es habitual que los creadores retoquen textos o corrijan pequeños errores ortográficos. Al cambiar mínimamente la string pasará a tener otro código de encriptación y para Ren'Py será distinta, por lo que deberemos traducirla de nuevo aunque la antigua traducción pudiera valernos lingüísticamente. Y lo mismo ocurre si se renombra la label en la que se encuentra la string, aunque el contenido de esta no varíe.

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

Ren'Py incluye una funcionalidad que permitiría evitarnos este problema: si los desarrolladores de los juegos añaden a la línea modificada el identificador MD5 de la versión anterior, no se generará uno nuevo y la vieja traducción permanecerá vinculada al nuevo contenido de la string original. Sin embargo, como muy pocos creadores piensan en las posibles traducciones, lo normal es que no tengáis esa ayuda.

2.2.2.- El resto de strings y los comandos old y new: Como ha quedado dicho, 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 literal 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 cuyo valor sea un 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, al generar los scripts de traducción, el Ren'Py SDK las agrupa todas al final del documento, bajo el comando "`translate Spanish strings:`".

```
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 ""
```

La imagen de la izquierda muestra el comienzo de esa segunda sección del script de traducción "convo_oc.rpy" de "What a Legend!". Como veis, aquí ya no hay códigos raros. Y si antes el comando "`translate Spanish`" se aplicaba a cada línea, ahora un mismo comando vale para todas. Sigue habiendo una primera línea con el símbolo # para informarnos sobre la ubicación de la string 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 indicado al Ren'Py SDK). En este caso, el texto original, que antes era meramente informativo, ahora es parte integral de la función de traducción, por lo que nunca debemos borrarlo: es lo que Ren'Py buscará, igual que antes buscaba un código MD5.

Debemos tener en cuenta que ahora, si hay varias strings literalmente idénticas, el Ren'Py SDK solo extraerá la primera que se encuentre cuando repase por orden alfabético los scripts originales para generar los archivos de traducción (a diferencia de lo que hace con el bloque de diálogos, del que extrae todas las strings asignándoles códigos de encriptación distintos, aunque se trate de strings idénticas). Y luego, al jugar, todas las strings que coincidan **literalmente** con el contenido de una línea "`old`" se sustituirán siempre por la traducción que hayamos indicado esa única vez en la línea "`new`". Así que, cuando en este script indiquemos la traducción que le corresponde, ya no tendremos que traducir la string "Permit" en ningún otro script y esa traducción se mostrará en todo el juego cada vez que la expresión "Permit" aparezca así escrita fuera del cuerpo de diálogos.

Esto, que puede parecer una ventaja puesto que nos evita repetir trabajo, se vuelve un inconveniente cuando nos encontramos con strings idénticas que pueden tener significados distintos dependiendo del contexto. Por ejemplo, pensad en varias strings que solo digan "Right", y que unas veces tendríamos que traducir como "Correcto" y otras como "Derecha", o "Derecho": solo se generaría una string de traducción para todas ellas y al jugar siempre aparecería la traducción que indicáramos esa única vez, con lo que en ocasiones el texto no tendría sentido. Más adelante veremos cómo solucionarlo.

Técnicamente, esta traducción directa podría hacerse 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 no se suelen repetir. Así que, cuando Ren'Py tiene que mostrar una traducción, tarda menos en buscar y reemplazar miles de códigos encriptados que miles de líneas más largas. Sin embargo, estas otras strings son generalmente frases más cortas, mucho menos numerosas y que es más probable que se repitan en varias ocasiones, así que Ren'Py puede permitirse hacer una búsqueda concreta de su contenido literal sin ralentizar la ejecución del juego. De ahí la existencia de los dos sistemas de traducción.

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.

|Índice|

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 de los menús 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 en este caso 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 (solo ese texto entrecomillado) y hemos colocado un guion bajo _ antes del paréntesis. Esta combinación de símbolos `_( )` permite que Ren'Py SDK identifique el entrecomillado de dentro del paréntesis como una string traducible. 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 la string "Your thoughts" por _("Your thoughts"), el Ren'Py SDK no la extraería para ser traducida.

Eso no quiere decir que no hubiéramos podido traducirla: al tratarse de una traducción directa, sin código MD5, 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: sustituir la extracción automática de la string por nuestra intervención manual. Para ello iríamos a la parte del documento que comienza con el comando `translate Spanish strings:` y escribiríamos una función de traducción con el mismo formato que vemos arriba: respetando siempre la indentación 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 (`"Your thoughts"`, en este caso) respetando escrupulosamente su escritura tal y como dicen el cuarto y el quinto mandamiento de Ren'Py (es decir, deberíamos incluir absolutamente todos los símbolos y etiquetas que figuren en la string original), y debajo escribiríamos el comando `new` con la traducción entre comillas.

Personalmente, me he acostumbrado a revisar y editar los archivos originales antes de generar los scripts de traducción, incluyendo los símbolos `_( )` donde sean necesarios para que el Ren'Py SDK haga luego un trabajo completo de extracción, y solo aplico el método manual para añadir alguna string "rebelde" que se me haya pasado por alto en esa primera revisión y solo detecto cuando me aparece sin traducir al jugar Otros prefieren generar los scripts de traducción sin tocar antes los originales, y luego van incluyendo a mano todo lo que el Ren'Py SDK se deja sin extraer. Pero también se puede editar el script original para añadir los símbolos que falten y regenerar los archivos de traducción todas las veces que queramos: si tenemos cuidado de decirle al Ren'Py SDK que use el mismo idioma y NO reemplace las traducciones existentes, nos actualizará los scripts incluyendo al final de los mismos las nuevas strings detectadas.

Prefiráis un sistema u otro, para conseguir una traducción íntegra de todos los textos del juego tendréis que revisar uno a uno los scripts originales y buscar este tipo de comandos, cuyas strings no serán detectadas de forma automática por el Ren'Py SDK salvo que el autor del juego las haya incluido dentro del símbolo `_( )`:

- `Character("...")`, usado para definir personajes, como hemos visto en el ejemplo
- `text "..."`, usado para mostrar texto en una pantalla específica, fuera del bloque de diálogos
- `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('...')`, usado para mostrar mensajes tras completar una acción

Además, tenemos los valores de las variables de texto. Aquí podemos encontrarnos con varias posibilidades, pero ninguna de ellas será extraída automáticamente por el Ren'Py SDK si no tiene el símbolo `_( )`:

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

Así pues, sería cuestión de meter dentro del símbolo `_( )` todos esos entrecomillados en los que he puesto puntos suspensivos y generar los scripts de traducción, o bien de copiarlos literalmente 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 comandos 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 luego Ren'Py buscará y reemplazará. Y ojo: a veces será necesario hacer algo más para que se muestre la traducción.

Para acabar con este punto, una puntualización: el símbolo `_( )` solo se usa para que el Ren'Py SDK extraiga las strings que haya dentro del paréntesis, y no es necesario para que la traducción se muestre en pantalla. Así que, en el caso de juegos que sacan actualizaciones, si dentro de los scripts de traducción que podemos reutilizar ya figura esa string traducida con los comandos `old` y `new`, no es necesario volver a editar los scripts originales para añadir otra vez el símbolo. Y haberlo añadido tampoco nos obliga a introducir ese script editado en el parche, ya que el jugador no lo necesita para que la traducción funcione.

|Índice|

2.4.- 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, mediante los comandos para extraer y fusionar cadenas traducidas. Pero la realidad es que, en el momento en que el script orginal no coincida con el de la versión anterior, se generará un script de traducción totalmente nuevo, obligándonos a repetir el trabajo que ya habíamos

hecho en versiones anteriores. De modo que lo más efectivo es, sencillamente, seguir estos pasos:

1. Descargarnos la nueva versión del juego.
2. Extraer, en caso de que sea necesario, los scripts .rpy con UnRen.
3. Mover la carpeta "game/tl/Spanish" que tenemos en la versión anterior (es decir, nuestra vieja traducción) a la nueva versión del juego, mediante un simple copiar y pegar.
4. Generar los scripts de traducción de la nueva versión.

Si el idioma que indicamos en el Ren'Py SDK en el punto 4 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 archivos existentes con el contenido para el que no encuentre una traducción válida. Es decir, al final de cada uno de esos scripts que hemos copiado de la versión anterior, añadirá 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). Y lógicamente, si en esta versión hay nuevos scripts originales, también se crearán sus correspondientes scripts de traducción. Pero, como digo, al final solo se habrán incluido las strings que no estuvieran ya traducidas.

Si ahora ordenamos los archivos de la carpeta "game/tl/Spanish" por fecha de modificación, veremos cuáles acaban de ser creados o modificados, lo que nos indica qué scripts debemos abrir para traducir y cuáles podemos dejar como estaban.

|Índice|

2.5.- Traduciendo traducciones

Hay veces que los juegos se publican en un idioma a partir del cual no podemos realizar una traducción digna, básicamente por nuestro propio desconocimiento de dicho idioma. Siempre queda la opción de recurrir a algún tipo de traducción automática, pero el resultado suele ser manifiestamente mejorable y, aunque dediquemos tiempo a pulir esa traducción para corregir los fallos más evidentes, muy probablemente habrá expresiones, refranes o frases hechas cuyo significado original no alcanzaremos a comprender del todo. Pero puede ocurrir que alguien con más conocimiento del idioma original ya haya traducido el juego a su propia lengua, o que el propio desarrollador publique su juego con una traducción integrada; si esa lengua es una que nosotros sí dominamos, podemos beneficiarnos de su trabajo para realizar nuestra traducción a partir de la suya.

Suponiendo que ese otro traductor haya seguido el método ideado por los desarrolladores de Ren'Py (o sea, el descrito en esta guía), existirá una carpeta llamada "game/tl/MyLanguage" (donde "MyLanguage" es el nombre del idioma que el traductor introdujo en su Ren'Py SDK) con todos los scripts de traducción a dicho idioma. Como hemos visto, esos scripts estarán formados por dos grandes bloques: el de las líneas de diálogo y el del resto de strings traducibles (menús, nombres de personajes, etc.). Y en ellos aparecerán las mismas funciones de traducción que ya explicamos en su momento, solo que con "`MyLanguage`" donde en mis ejemplos aparecía "`Spanish`" y, por supuesto, con el texto traducido a ese otro idioma.

En este caso, lo que haremos para generar nuestros scripts de traducción será olvidarnos del Ren'Py SDK y, simplemente, crear una subcarpeta nueva dentro de la carpeta "game/tl" que llamaremos "Spanish" (o como prefiramos) y en la que pegaremos todos los scripts de la subcarpeta "game/tl/MyLanguage". Luego ya solo tendremos que editar los de nuestra subcarpeta para sustituir "MyLanguage" por "Spanish" en todas las funciones de traducción, algo que se puede hacer rápidamente en cualquier editor de texto mediante las opciones de buscar y reemplazar.

Así, en las líneas del bloque de diálogos que se traducen mediante el código de encriptación, cada línea pasaría de tener la función `translate MyLanguage XXXXXXXX:` a tener la función `translate`

`Spanish XXXXXXXX:` (recordemos que el código de encriptación de cada línea, que en este ejemplo sustituyo por XXXXXXXX, solo depende de la label y del contenido de la línea original, por lo que es idéntico para todos los idiomas). Y la función para la traducción de strings mediante sustitución directa pasaría de ser `translate MyLanguage strings:` a ser `translate Spanish strings:`

Y ya estaría: traduciendo a nuestro idioma el contenido de las líneas escritas en el idioma "MyLanguage", que sí entendemos, tendremos nuestra traducción lista para ser implementada en el juego.

|Índice|

2.6.- 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 realizada. Lo primero que debemos saber es que, por defecto, cualquier juego Ren'Py arrancará inicialmente en el idioma original en el que se creó, que internamente se identifica siempre como `None` (sea cual sea ese idioma original); y a partir de ese primer inicio, cada vez que se arranque el juego, Ren'Py escogerá el último idioma en el que se jugó. Por ejemplo: la primera vez que arranquemos un juego hecho en inglés los textos aparecerán en inglés, pero si dentro de la partida seleccionamos el idioma español y cerramos el juego, la siguiente vez que lo iniciemos arrancará en español.

Pero existen varias funciones que pueden alterar ese comportamiento, y esas son las que vamos a describir a continuación. La primera de ellas sirve para ordenar a Ren'Py que nuestro idioma sustituya al `None` como idioma por defecto la primera vez que se ejecute el juego. Para ello, abriremos cualquier script del juego (se puede hacer en uno de traducción) y escribiremos esta línea de código, sin ninguna indentación:

```
define config.default_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 eligierais. Esta línea de código hará que el juego arranque inicialmente en español, pero respetando el funcionamiento ordinario de Ren'Py; es decir, si el jugador cambia de idioma durante su partida, la siguiente vez que se inicie el juego lo hará en ese otro idioma. Pero el problema es que, si el jugador ya ha probado antes el juego sin traducir, este comando es inservible: tras instalar la traducción le seguirá arrancando en el idioma original porque es el último en el que jugó.

Además, puede pasar que el desarrollador haya escrito los textos del juego en su lengua materna y lo haya lanzado ya con una traducción a otro idioma, haciendo que ese sea el idioma de arranque por defecto. Si lo ha hecho bien, habrá usado esa misma variable `config.default_language`, por lo que en alguna parte de los scripts originales (probablemente en el archivo gui.rpy, aunque la ubicación es indiferente) ya existirá esa línea de código con el nombre que le dieran a ese segundo idioma cuando generaron sus scripts de traducción. Podríamos editar la línea original para cambiar el idioma por `"Spanish"` (o emplear el comando `init`, que veremos más adelante, para sobrescribirla) pero, en todo caso, si el jugador ya había abierto el juego alguna vez antes de instalar nuestra traducción, el idioma de arranque no cambiará.

Así que, si queremos que el juego se inicie siempre en español (incluso cuando la última partida haya sido en otro idioma), escribiremos en cualquier script esta otra línea de código, también sin sangría:

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

Ahora el juego se iniciará SIEMPRE en español, incluso aunque el jugador cambie de idioma durante su partida. Es decir, la variable `config.language` prevalece sobre `config.default_language` y además desactiva el sistema que Ren'Py emplea por defecto para iniciar los juegos en el último idioma usado.

Pero, más allá del método elegido para conseguir que el juego se inicie en español, siempre deberíamos ofrecerle al jugador la posibilidad de cambiar de idioma, y lo más lógico es hacerlo dentro del menú Preferencias. Suponiendo que el juego use la pantalla de preferencias que Ren'Py incorpora por defecto, si

vamos al script “screens.rpy” original veremos algo así en el apartado `screen preferences()`:

```
## Preferences screen ##########################################################
##
## The preferences screen allows the player to configure the game to better suit
## themselves.
##
## https://www.renpy.org/doc/html/screen_special.html#preferences

screen preferences():

    tag menu

    use game_menu(_("Preferences"), scroll="viewport"):

        vbox:

            hbox:
                box_wrap True

                if renpy.variant("pc") or renpy.variant("web"):

                    vbox:
                        style_prefix "radio"
                        label _("Display")
                        textbutton _("Window") action Preference("display", "window")
                        textbutton _("Fullscreen") action Preference("display", "fullscreen")

                vbox:
                    style_prefix "radio"
                    label _("Rollback Side")
                    textbutton _("Disable") action Preference("rollback side", "disable")
                    textbutton _("Left") action Preference("rollback side", "left")
                    textbutton _("Right") action Preference("rollback side", "right")

                vbox:
                    style_prefix "check"
                    label _("Skip")
                    textbutton _("Unseen Text") action Preference("skip", "toggle")
                    textbutton _("After Choices") action Preference("after choices", "toggle")
                    textbutton _("Transitions") action InvertSelected(Preference("transitions", "toggle"))

                ## Additional vboxes of type "radio_pref" or "check_pref" can be
                ## added here, to add additional creator-defined preferences.

            null height (4 * gui.pref_spacing)
```

En las primeras versiones de esta guía indicaba que había que modificar ese código para incluir ahí la opción de cambio de idioma, añadiendo las siguientes líneas de código al mismo nivel de indentación que las opciones “Rollback Side” y “Skip” (recordad que hay que marcar las sangrías con la barra espaciadora):

```
vbox:
    style_prefix "radio"
    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 idioma que fuera pero sin cambiar nada más, ya que para Ren’Py ese siempre sería el idioma `None`. Y, como he dicho, este código serviría para una pantalla de preferencias clásica, pero podría no servir si el juego incorpora una personalizada. En ese caso deberíais fijaros en el resto de elementos que veáis en la pantalla de preferencias de vuestro caso concreto, para adaptar esta nueva opción a ese estilo. Lo más fácil siempre es copiar cualquiera de esos otros elementos del menú y sustituir los textos y las acciones.

Pero lo importante es saber que en una pantalla con “textbuttons” (o “imagebuttons”) hay que añadir una opción para que, al hacer click en ella, se ejecute el comando `action Language("Spanish")`, donde `"Spanish"` es el nombre del idioma que introdujimos en el Ren’Py SDK al generar los scripts de traducción. Si lo escribimos de forma distinta (`"spanish"`, por ejemplo) Ren’Py no detectará la traducción y el botón quedará inactivo. Y es importante mencionar también que `None` va siempre sin comillas, a diferencia de los demás idiomas para los que queramos aplicar la acción de cambio de idioma, que sí las llevan.

Si os fijáis, en ese código que he puesto como ejemplo hay varias strings dentro del símbolo `_( )` algo pensado para que el Ren’Py SDK pueda extraerlas al generar los scripts de traducción. Pero, en realidad, solo haría falta traducir la string “Language”, ya que los nombres de los idiomas deberían aparecer siempre escritos en dicho idioma independientemente de en qué lengua estemos jugando. Es decir, si no traducimos “English”, cuando un angloparlante que no sepa español se tope con nuestra traducción y arranque el juego en español, al acceder intuitivamente al menú de preferencias verá 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 hay que hacerlo así.

En cuanto al resto de opciones para integrar la función de cambio de idioma fuera de esta pantalla de preferencias, lo cierto es que hay tantas posibilidades como juegos y traductores, por lo que no puedo detallar cada una de ellas. Al final, todas se basan en incluir en alguna parte el comando `action Language("Spanish")` o activar la variable `$ renpy.change_language("Spanish")`, método que suele usarse cuando la opción de cambio de idioma se plantea en un menú de pregunta y no como un botón de selección. En cualquier caso, recomiendo hacer un poco de ingeniería inversa y revisar foros y manuales online de Ren'Py para crear y editar screens, splashscreens, imagemaps, keymaps, botones… Hay todo un mundo de posibilidades, a cada cual más compleja y visualmente atractiva.

Dicho todo esto, tras unos cuantos meses más de práctica y aprendizaje he llegado a la conclusión de que es preferible crear una pantalla de preferencias que sustituya a la original mientras el usuario tenga la traducción instalada, pero que desaparezca si, por alguna circunstancia, el jugador quiere eliminar la traducción y recuperar intacta la versión original del juego. Así que, desde hace un tiempo, lo que hago es crear un archivo .rpy nuevo y específico para la traducción donde incluyo esta pantalla y otros ajustes que veremos más adelante.

|Índice|

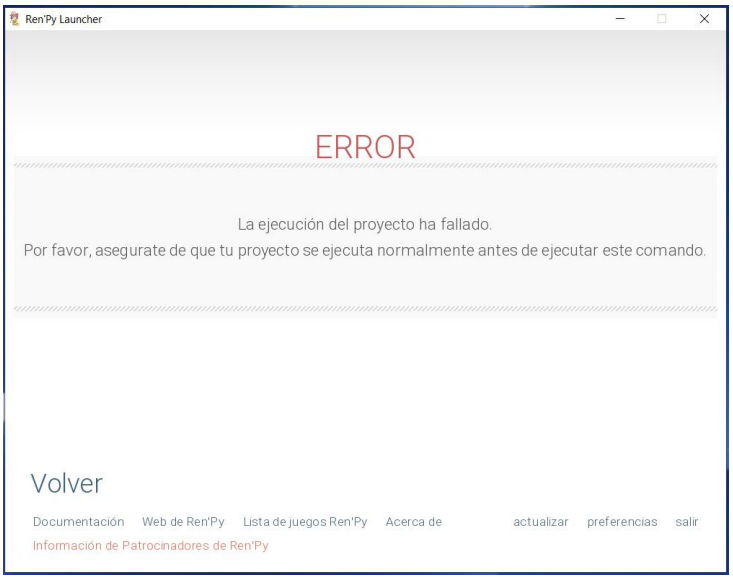
3.- Y ESTO POR QUÉ NO ME SALE

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.

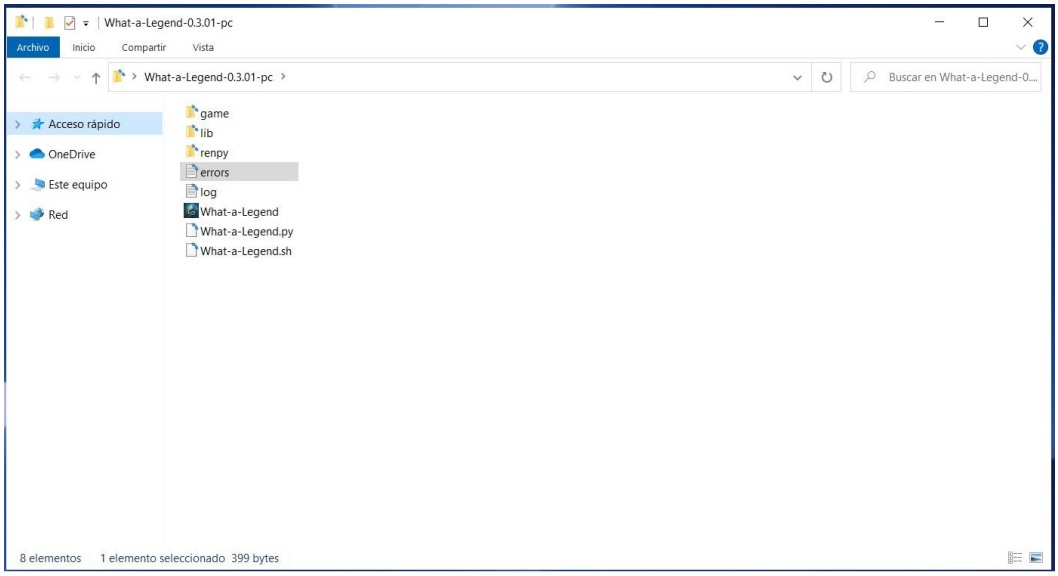
|Índice|

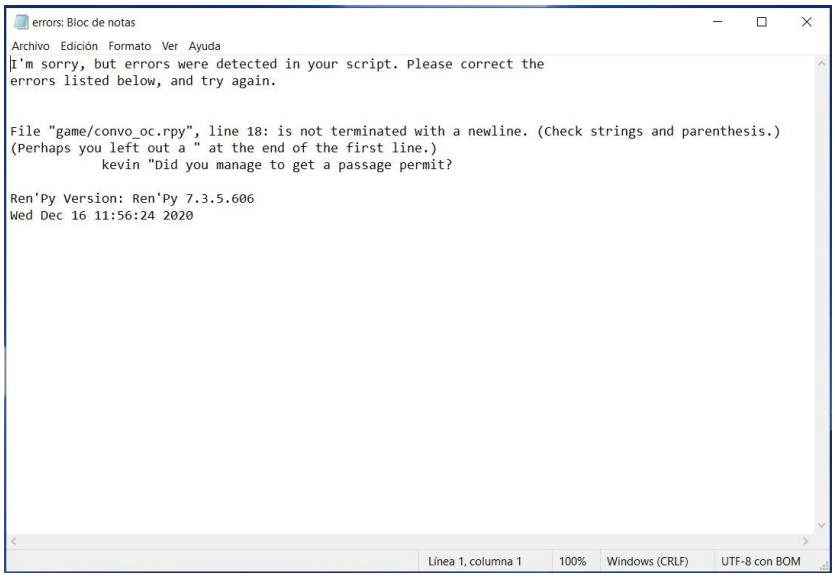
3.1.- Detección de errores y bugs

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 dos cosas: que hemos cometido algún error al editar los scripts .rpy originales antes de generar la traducción, o que estamos usando una versión del Ren'Py SDK más antigua que la usada por el desarrollador del juego. Esta segunda opción, lógicamente, podremos solventarla en un par de minutos descargando la última versión disponible del Ren'Py SDK, algo siempre recomendable.



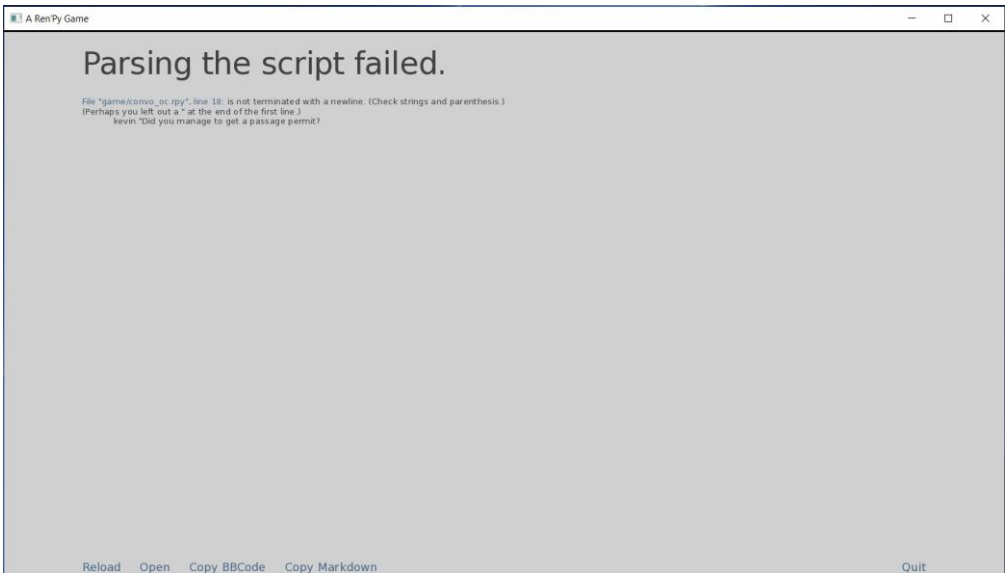
Si sospechamos que el problema se debe a que hemos cometido algún error al editar los scripts, deberíamos ir a la carpeta raíz del juego para ver qué es exactamente lo que ha fallado. Allí, junto al ejecutable, aparecerán varios archivos .txt, como vemos en la siguiente imagen:



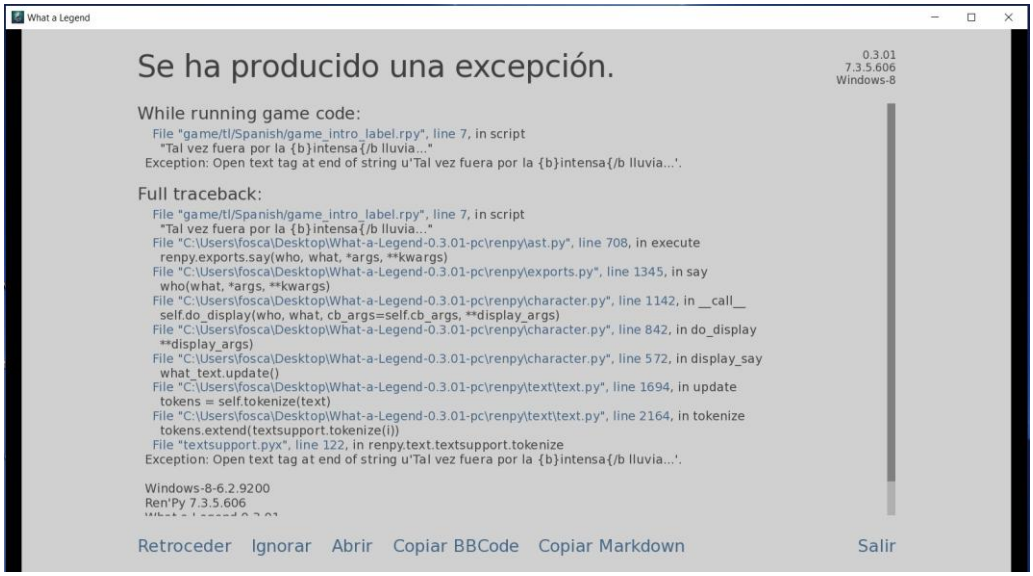


El que nos interesa es el llamado "errors.txt", que podemos abrir con cualquier procesador de texto. En él veremos el mensaje de error con la indicación de la línea de código que ha provocado el problema. En este caso, algo tan habitual como habernos olvidado de cerrar un entrecomillado en la línea 18 del script "convo_oc". 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.

Ese mismo mensaje de error se habría mostrado en caso de haber intentado arrancar el juego normalmente. Entonces veríamos un pantallazo así, con la misma información que el documento "errors.txt".



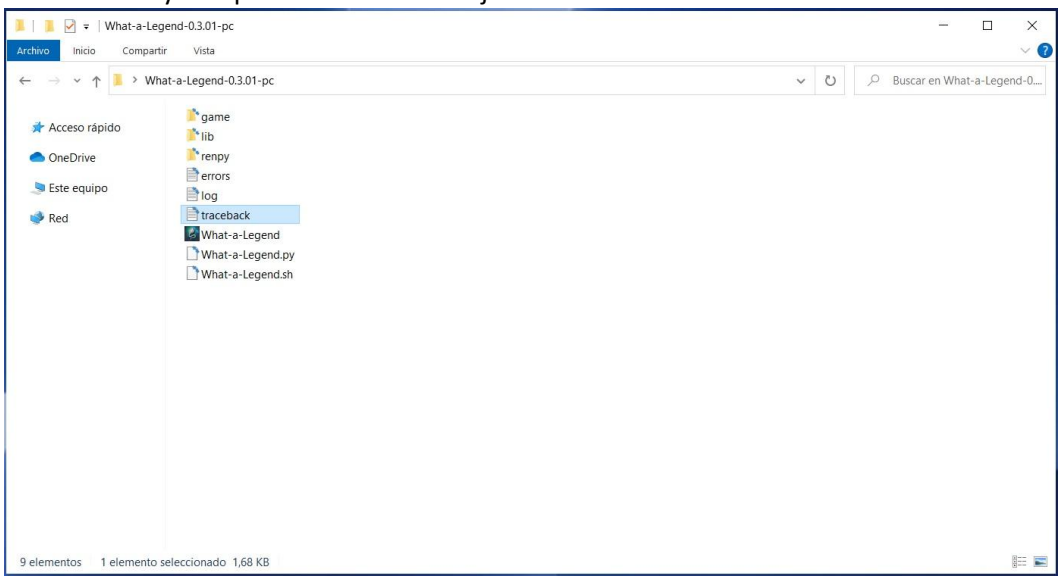
En otras ocasiones, el error no es crítico (en el sentido de que permite arrancar el juego y generar los archivos de traducción) pero sí da lugar a una excepción mientras se está jugando. Generalmente, si veis estos fallos se deberá a una etiqueta `{tag}` mal cerrada o a un problema con las variables.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



En esta pantalla se nos da la opción de intentar ignorar el fallo y seguir jugando (con la gran posibilidad de que el juego siga dando más fallos y acabe cerrándose), o retroceder a un punto anterior para guardar la 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 el ejemplo del último pantallazo, en el script "game/tl/Spanish/game_intro_label.rpy" (o sea, el script de traducción "game_intro_label.rpy"), en la línea número 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, en la gran mayoría de casos seremos capaces de localizar y corregir el fallo.

Ahora, si echáramos 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.



Por último, si vemos algún mensaje más complejo o que haga referencia a scripts de la carpeta "renpy", es probable que hayamos eliminado sin querer algún símbolo aparentemente insignificante (como un % o una barra inclinada \ /) o hayamos añadido algún otro (como un espacio antes o a continuación de un %) que desconfigura las funciones preprogramadas de Ren'Py. Ese tipo de errores, muy habituales al usar traductores automáticos, son más difíciles de corregir, porque sin un alto conocimiento de las interioridades de Ren'Py no sabremos dónde buscar su verdadero origen. Así que armaos de paciencia y preparaos para bucear en foros o, directamente, volved a empezar el trabajo desde una versión limpia y sin fallos del juego.

|Índice|

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. Pero 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 del bloque de diálogos. 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_XXXXXXXX:
    # mc "Hello. My name is John."
    mc "Hola."
    mc "Me llamo John."
```

Así, en la traducción al español el texto se dividirá en dos mensajes, sin mayor consecuencia. En realidad, podemos realizar cualquier variación que se nos ocurra, como añadir funciones y condiciones, modificar variables, etc., que solo se aplicarán cuando se juegue en español. Y es que, aunque 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 aparezca en el script de traducción, así que nada nos impide sustituir esa línea de código original (que era un texto dicho por un personaje) por todo un bloque distinto de código.

La tercera opción, algo más compleja, es cambiar el tamaño de letra y/o el ancho del cuadro de texto, para que en cada línea nos quepan más caracteres. Pero como eso implica modificar los llamados "estilos" del juego (la configuración estética de los textos y menús), lo dejo para explicarlo en el siguiente punto.

|Índice|

3.3.- Fuentes que no admiten caracteres especiales: cambiando estilos

En ocasiones, la fuente original usada en los textos del juego que estamos traduciendo no contiene 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 en pantalla sin esas letras o con un símbolo raro en su lugar. Para solucionarlo sin tocar la programación del juego, podemos limitarnos a pulsar la tecla A mientras jugamos: si el juego se ha creado con una versión de Ren'Py relativamente reciente, se abrirá un menú de accesibilidad que permite sustituir la fuente original por la DejaVuSans, que viene integrada en Ren'Py. Pero pudiendo "arreglarlo" nosotros, ¿por qué no hacerlo?

Si somos hábiles y tenemos mucho tiempo libre, podríamos liarnos la manta a la cabeza y editar la fuente original con un programa de edición gráfica de fuentes (como TypeLight) para añadir los símbolos que faltan. Así no cambiaríamos la estética del juego ideada por su desarrollador y solo tendríamos que acordarnos de incluir el archivo .ttf resultante en nuestro parche para que sustituya al original dentro de la carpeta "game"

(o donde se ubique). Pero lo más lógico es iniciarnos en el vasto mundo de los cambios de estilo.

Un estilo en Ren'Py es, como su propio nombre indica, un conjunto de varios elementos (fuente, tamaño y color de letra, posición y diseño del cuadro de texto, etc.) que definen el aspecto estético de la interfaz del juego. Como estamos viendo, en ocasiones se hace necesario modificar alguno de esos elementos estéticos originales para que nuestra traducción pueda mostrarse correctamente, así que podríamos buscarlos y modificarlos directamente en el archivo "game/gui.rpy", que es donde habitualmente se encuentran definidos los elementos generales que afectan a la inmensa mayoría del juego.

Este era el método que explicaba en la primera versión de esta guía, pero debemos tener en cuenta que entonces los cambios se aplicarán a todos los idiomas disponibles, rompiendo la estética ideada por el desarrollador del juego. Además, si luego queremos compartir nuestra traducción, deberíamos incluir ese nuevo archivo "gui.rpy" en nuestro parche, obligando a los jugadores a reemplazar uno de los archivos originales del juego, de modo que, si por algún motivo quisieran eliminar la traducción, no podrían recuperar automáticamente la versión original íntegra e intacta.

Así que, para aplicar cambios que solo surtan efectos cuando nuestro idioma esté activado pero respeten la configuración original del juego en todos los demás, podemos usar la siguiente función de traducción, escribiéndola en cualquier script del juego (por ejemplo, en uno de nuestra carpeta tl/Spanish):

```
translate Spanish python:
```

Va sin sangría y `Spanish` es, como siempre, el nombre de idioma que introduje al generar los scripts de traducción con el Ren'Py SDK. Mi consejo es escribirla en un script específico junto con la pantalla de preferencias con el cambio de idioma, de la forma que luego veremos. Esta función abre un bloque de código python bajo el que podemos definir absolutamente todos los elementos que queramos personalizar exclusivamente para cuando el juego se ejecute en nuestro idioma. Ciñéndonos al problema de la fuente, podemos escribir algo así:

```
translate Spanish python:
    gui.text_font = "myfont.ttf"
    gui.name_text_font = "myfont.ttf"
    gui.interface_text_font = "myfont.ttf"
    gui.button_text_font = "myfont.ttf"
    gui.choice_button_text_font = "myfont.ttf"
```

Donde `"myfont.ttf"` es la fuente que queremos usar. Cada una de esas líneas corresponde a un tipo de texto (diálogos, nombre de los personajes, textos de la interfaz, botones y opciones de selección, en ese orden), así que no hace falta incluirlas todas, solo las que necesitemos cambiar. Y recordad que tendremos que incorporar el archivo .ttf a nuestro parche para que quede guardado en la carpeta "game/tl/Spanish".

Pero también podemos usar esta misma función para cambiar otros elementos de la interfaz, como el tamaño de esas fuentes o la anchura y posición del cuadro de texto, lo que puede permitirnos solucionar el problema de espacio que impedía que nuestra traducción se mostrara completa en una línea:

```
translate Spanish python:
    gui.text_size = ...
    gui.name_text_size = ...
    gui.interface_text_size = ...
    gui.label_text_size = ...
    gui.notify_text_size = ...
    gui.textbox_height = ...
    gui.dialogue_width = ...
    gui.dialogue_xpos = ...
```

Y muchos más. En todos ellos sustituiremos los puntos suspensivos por un valor numérico: para los distintos `text_size` pondremos un valor inferior al que figure asignado originalmente a esa variable en el archivo "gui.rpy", haciendo la letra más pequeña; aumentando el valor de `gui.textbox_height` haremos más alto el espacio dedicado al texto de los diálogos; si a `gui.dialogue_width` le aumentamos el número que figure en el archivo "gui.rpy", incrementaremos la anchura del cuadro de texto; y variando el número asignado a `gui.dialogue_xpos` podremos desplazar horizontalmente el punto desde el que se inicia el texto (cuanto más pequeño sea el número, más a la izquierda comenzará a mostrarse el texto). Estas y otras variables nos permitirán ajustar los textos de nuestra traducción y, por lo general, suelen funcionar en todos los juegos. Aunque, como siempre, hay que tener en cuenta que estaremos cambiando la estética original, lo que puede generar otros problemas de visibilidad y superposición de textos. Todo es cuestión de probar hasta dar con el resultado adecuado.

Todas las opciones indicadas hasta ahora hacen referencia a elementos que se aplican de forma general a todo el juego. Pero es posible que, para personajes, pasajes o ventanas concretas del juego, el desarrollador haya definido una configuración específica. Eso es un estilo propiamente dicho, definido con el comando `style`. En estos casos, la mayor complicación reside en encontrar en los scripts originales el nombre del estilo concreto que queremos cambiar, ya que después el proceso es el mismo: usando la función `translate Spanish python:` indicaríamos el elemento concreto de ese estilo que deseamos cambiar. Por ejemplo, suponiendo que quisiéramos cambiar la fuente usada en un estilo llamado "dream", y también su tamaño, escribiríamos esto en nuestro script:

```
translate Spanish python:
    style.dream.font = "myfont.ttf"
    style.dream.size = 22
```

Y así con cualquier otro elemento. Como digo, la clave está en encontrar la parte del código donde se hayan definido los elementos del estilo para saber sus valores originales y poder cambiarlos a nuestro gusto.

|Índice|

3.4.- Traducción de imágenes

En ocasiones, los creadores del juego deciden que un detalle de la trama se muestre por escrito en algún medio visual (un mensaje de móvil, 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, desde el punto de vista del creador del juego suele ser más cómodo crear una imagen con el texto e insertarla en el juego.

Si tenemos la suerte de que, mientras en pantalla se muestra esa imagen con un texto en otro idioma, también hay alguna línea de 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_XXXXXXXX:
    # mc "Look. He sent me a message."
    mc "Mira. Me envió un mensaje. Decía que..."
```

O también podríamos "subtitular" el contenido de la imagen aprovechándonos de una solución que ya explicamos cuando necesitábamos corregir una línea demasiado extensa: dividir la traducción de esa string en varias líneas para escribir en ellas la traducción del mensaje de la imagen.

```
# game/script.rpy:12
translate Spanish label_start_XXXXXXXX:
    # 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 indentación, incluir una línea debajo en la que solo pongamos dos comillas, así: `""` Luego regeneraríamos 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, o bien optar por sustituir las labels afectadas, como explicaremos luego.

Sin embargo, a veces estas opciones no son viables o no dan un buen resultado estético. En ese caso, la única solución es editar la imagen original con algún programa de edición de imágenes como Photoshop, GIMP, o incluso Paint, si es algo sencillo. Para ello, buscaremos la imagen dentro de la carpeta "game/images" (podemos facilitar esa búsqueda fijándonos en el nombre de la imagen que aparece en el script original tras el comando `show` o `scene`) y la editaremos para poner el texto en español. En un mundo ideal, también podríamos pedirle amablemente al creador del juego que lo haga por nosotros o que al menos nos facilite la imagen base (sin texto) y el tipo de fuente, para ahorrarnos trabajo. Suerte con eso.

Una vez terminada la edición, no debemos 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.jpg`", debemos guardar la imagen traducida como "`game/tl/Spanish/images/sms00.jpg`". Y ya está: cuando estemos jugando con la traducción activada, al llegar al punto del script donde debe mostrarse la imagen "sms00.jpg", 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.

|Índice|

3.5.- Homonimia: 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 (borrando tanto la línea del comando `old` como la del `new` de uno de esos pares de líneas de código).

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 pasa por 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 también vimos hace mucho, 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 o tags son comandos incluidos en las strings dentro de unas llaves { } y generalmente se usan para cambiar el tamaño y el color de letra, resaltar la frase en negrita o cursiva, etc. Ren'Py considera estas etiquetas como parte de la string que las contiene y ejecuta lo que le ordenen, pero su contenido 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 hacer nada con lo que aparezca escrito a su derecha), tenemos una string literalmente diferente a la que no tiene etiqueta (y que por tanto será extraída por el Ren'Py SDK) pero que en pantalla se verá exactamente igual, porque esa etiqueta no le está ordenando a Ren'Py realizar ninguna acción.

Por tanto, lo que haremos será buscar en el script original 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 (“Right”) en el idioma original. 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 de la string original. Quedaría algo así (aunque no necesariamente aparecerían seguidas, o ni siquiera en el mismo script):

```
translate Spanish strings:

    old "Right"
    new "Correcto"

    old "Right{#Derecha}"
    new "Derecha"

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

Pero recordad que en este caso no vale solamente con escribir esto en el script de traducción: **hay que editar el original** para añadir la etiqueta a la string que queramos traducir de forma distinta y que así, al jugar, Ren’Py busque la traducción de esa string con etiqueta. Y, obviamente, habrá que acordarse de incluir esos scripts editados en el parche o de recurrir al reemplazo de labels que explicaremos más adelante.

|Índice|

3.6.- Traducción de variables de texto (I): interpolaciones

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. E, 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.

Por ejemplo, supongamos que en algún punto de los scripts originales se define la siguiente variable:

```
default fruit = "apple"
```

Y, más adelante, puede que incluso en otro script, hay una línea de código como esta que le dice a Ren’Py que modifique el valor de esa variable para almacenar otro:

```
$ fruit = "orange"
```

Aunque a veces hay que investigar mucho, si hemos hecho bien nuestro trabajo de ayuda al Ren’Py SDK habremos localizado tanto el valor original de la variable “fruit” como todos los posibles valores que pueda asumir durante el juego. Y ya sea porque incluimos esos valores dentro del símbolo `_( )` antes de generar la traducción o porque los copiamos manualmente, en los scripts de traducción tendremos sus traducciones con los comandos `old` y `new`, de forma parecida a esta:

```
translate Spanish strings:

    old "apple"
    new "manzana"

    old "orange"
    new "naranja"
```

Pues bien, cuando el valor de una variable vaya a aparecer en pantalla, lo hará incrustado en una string, algo que técnicamente se llama “interpolación”. Lo 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. Y el problema a la hora de traducir lo tendremos cuando esa

variable sea como esta "fruit"; es decir, una variable cuyos posibles valores son palabras o frases. Por ejemplo, en un script de traducción podemos encontrarnos con esto:

```
translate Spanish start_XXXXXXXX:
    # mc "I don't like this [fruit]."
    mc "No me gusta esta [fruit]."
```

En este caso, `[fruit]` es la variable interpolada: según lo que hayamos hecho durante el juego, la frase será distinta puesto que mostrará la fruta que no nos gusta, y al codificarla así el creador del juego se ahorra tener que escribir la misma frase tantas veces como posibles valores de "fruit" haya. El problema es que, si no hacemos nada más, la línea que se verá en pantalla dirá "No me gusta esta apple." o "No me gusta esta orange.", porque nuestra string traducida le está pidiendo a Ren'Py que incluya el valor de la variable `[fruit]`, que se ha almacenado en el idioma original, tal y como aparece escrito en los scripts originales.

Pensad que, aunque tengamos la traducción activada, el juego se ejecuta siguiendo los scripts originales y solo "salta" a los de traducción cuando haya una string que mostrar en nuestro idioma, por lo que todas las variables se van a modificar (en principio) desde ese código original. En este caso, el juego le ha dicho a Ren'Py que guarde como valor de la variable "fruit" primero "apple" y luego "orange", y ahora en esta string le estamos pidiendo que recuerde cuál es el valor actual de "fruit", pero Ren'Py no puede saber por sí mismo si ese valor es un elemento traducible, un número o una combinación de símbolos sin sentido. Simplemente muestra lo que tiene almacenado.

Así que, para que Ren'Py sepa que ese valor es en realidad otra string que debería mostrar traducida, **debemos añadir el apéndice `!t`** dentro del objeto con el nombre de la variable. Así:

```
translate Spanish start_XXXXXXXX:
    # mc "I don't like this [fruit]."
    mc "No me gusta esta [fruit!t]."
```

Ahora, al ejecutar la función de traducción para esta string de diálogo, Ren'Py se encuentra con que la línea con la string en español le está pidiendo que, además, busque en los scripts una traducción para el valor que tenga esa variable "fruit" interpolada, ya que dicho valor es otra string traducible. Lo ideal sería que la string original ya contuviese la interpolación `[fruit!t]`, puesto que no afectaría al juego en el idioma original y sería más difícil que a los traductores se nos olvidara incluir ese apéndice (al generar los scripts de traducción sin cadenas vacías ya lo tendríamos escrito, y si los creamos con cadenas vacías lo veríamos en la línea superior y lo copiaríamos), pero la mayoría de desarrolladores no conoce estas particularidades de Ren'Py. En cualquier caso, como veis, no es complicado de arreglar, y con que aparezca en la string de traducción es suficiente. **No necesitamos editar los scripts originales**.

|Índice|

3.6.1.- Nombres de personajes interpolados: Ahora veamos una peculiaridad de las interpolaciones: los nombres de los personajes. Y es que, por comodidad a la hora de redactar las strings del cuerpo de diálogos, o por necesidad en caso de que nos hayan dado la libertad de elegir el nombre de dicho personaje, a veces los autores de los juegos sustituyen los nombres completos de los personajes por la definición de su variable "character". Lo distinguiréis porque la variable interpolada en la string es la misma que veis al comienzo de las líneas de diálogo de dicho personaje, antes del entrecomillado.

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

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

En este caso, cada vez que hable el personaje `"li"`, encima del cuadro de texto aparecerá el nombre que se le ha asignado (Lieutenant). Como ya vimos, para que ahí aparezca la traducción de ese nombre común (Teniente) tenemos que haber extraído esa string manualmente o con el símbolo _( ) para que figure en

algún script de traducción con los comandos `old` y `new`. La particularidad es que, al haberse definido como un personaje, cuando esa variable `[li]` aparezca interpolada en una string no hace falta añadirle el apéndice **`!t`**, puesto que en este caso Ren'Py sí reconoce por defecto que su valor es un texto y se encargará de mostrar el nombre original o la traducción según el idioma en el que estemos jugando.

```
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."
```

En este ejemplo, aunque en nuestra string traducida escribamos simplemente `[li]`, cuando juguemos en español en pantalla aparecerá la palabra "Teniente". **No** necesitamos escribir `[li!t]`.

Pero a veces la cosa se complica. Un caso habitual es el de un personaje secundario que primero interviene con un nombre común porque aún no sabemos quién es y más adelante aparece ya con su nombre propio. Y es posible que la definición de ese personaje se realice de una manera parecida a esta:

```
define p = Character("[pname]")
```

Es decir, el nombre del personaje "p" es en realidad una string que consiste en la interpolación de otra variable llamada "pname". Y esa variable podría adoptar, por ejemplo, estos posibles valores:

```
default pname = _("Unknown Man")
$ pname = "Mike"
```

Con lo cual, tendremos que buscar los valores traducibles de la variable "pname" que sean nombres genéricos traducibles y extraerlos para traducirlos con los comandos `old` y `new`; pero, además, habrá que extraer también la string de la definición del personaje e incluir ahí el apéndice **`!t`**. Algo así:

```
translate Spanish strings:
    old "[pname]"
    new "[pname!t]"
```

De esta manera, cuando el valor de "`pname`" sea un nombre común (y hayamos encontrado y traducido ese valor), la variable "`p`" aparecerá traducida tanto en el cuadro de texto que identifica al personaje como en las posibles interpolaciones de `[p]` que se produzcan en los diálogos (donde no necesitaríamos escribir `[p!t]` porque "p" se ha definido como variable de tipo personaje y Ren'Py ofrecerá automáticamente la traducción de su valor, que en este caso será a su vez la traducción de otra variable). Pero atención: si lo que se interpola en una string es la variable `[pname]`, entonces sí deberíamos incluir `[pname!t]` en la traducción de dicha string, ya que "`pname`" es una variable normal como las que veíamos en el punto anterior, no una variable de clase personaje como "`p`".

Esta forma de definir personajes mediante una variable interpolada se usa con bastante frecuencia, pues es parte del sistema por el que los creadores nos permitan ponerle nombre a los personajes (normalmente al nuestro). Por lo general, esa variable será el resultado de una función `renpy.input` situada al comienzo del juego y que permite al jugador teclear el nombre que prefiera (nombre que, lógicamente, ni podemos saber de antemano ni hace falta que aparezca traducido). Pero debemos estar atentos al valor predeterminado de dicha variable, porque se usará por defecto en caso de que el jugador no introduzca un nombre o puede aparecer en pantalla si el personaje habla antes de que aparezca la opción de introducir su nombre: si ese valor predeterminado es un nombre común o genérico (como un mote), sí necesitaríamos traducirlo todo como acabamos de ver más arriba (es decir, habría que traducir ese valor genérico y además traducir también como una interpolación normal la string de definición del personaje).

|Índice|

3.6.2.- Concordancia gramatical: Hay idiomas en los que determinadas palabras experimentan cambios en

función del género o número gramatical, modo verbal… y estos cambios imprescindibles para que nuestras traducciones resulten comprensibles pueden venir impuestos por los valores de una variable que no tiene ese impacto en el idioma original.

Retomando el ejemplo genérico para las interpolaciones de nuestra variable `[fruit]`, ¿qué pasaría si, además de "apple" y "orange", nombres femeninos en español ("manzana" y "naranja"), se incluyera como posible valor "peach" ("melocotón"), que es masculino? Recordemos cuál era la string en la que se interpolaba `[fruit]`:

```
translate Spanish start_XXXXXXXX:
    # mc "I don't like this [fruit]."
    mc "No me gusta esta [fruit!t]."
```

Obviamente, nadie podría leer "No me gusta esta melocotón" en español sin levantar una ceja. Afortunadamente, como ya vimos en otro punto, un script de traducción es en realidad un conjunto de funciones que sustituyen una línea de código por otra, y nada impide que nuestra "línea" sea en realidad varias líneas que incluyan más comandos y funciones que una simple string traducida. Así pues, como la palabra "this" es un demostrativo que en español cambiará de género en función del valor de la variable "fruit", podemos usar esa misma variable para añadir una string alternativa. Para esto hay que tener un mínimo conocimiento de cómo se programa en Ren'Py, pero no es complicado.

```
translate Spanish start_XXXXXXXX:
    # mc "I don't like this [fruit]."
    if fruit == "peach":
        mc "No me gusta este [fruit!t]."
    else:
        mc "No me gusta esta [fruit!t]."
```

En primer lugar, respetando el primer mandamiento de Ren'Py sobre las indentaciones, comenzaremos a escribir nuestra función en el mismo punto en el que arranca una string de traducción sencilla: cuatro espacios a la derecha del margen izquierdo, ya que la función `translate` concluye con dos puntos `:` que es el símbolo que da paso en Ren'Py a un sub-bloque de código. Lo que escribimos ahí es una función condicional: con el comando `if` le decimos a Ren'Py que si la variable `fruit` tiene el valor `"peach"` debe mostrar la primera string (que debemos indentar a otro subnivel, con otros cuatro espacios adicionales), y con el comando `else` le indicamos que, si tiene cualquier otro valor, debe mostrar la segunda.

Para que todo funcione correctamente es imprescindible respetar la escritura de la variable y del valor que usemos como referencia (siempre en el idioma original), pero también el doble signo de igualdad, las comillas rodeando al valor que Ren'Py deberá buscar, y los dos puntos al final de esas líneas de código, así como las indentaciones extra de las strings traducidas. Sin eso, el juego no arrancará o generará una excepción al llegar a este punto.

Obviamente, deberemos analizar qué valor (o valores) de la variable nos conviene usar en cada caso como referencia en la línea `if`. Por ejemplo, si además de "peach" sabemos que la variable puede adoptar el valor "melon", cuya traducción también es un nombre masculino, deberíamos escribir así nuestra función:

```
translate Spanish start_XXXXXXXX:
    # mc "I don't like this [fruit]."
    if fruit == "peach" or fruit == "melon":
        mc "No me gusta este [fruit!t]."
    else:
        mc "No me gusta esta [fruit!t]."
```

En este caso, gracias a la partícula `or`, la primera string traducida se mostraría siempre que la variable "fruit" tuviera cualquiera de esos dos valores masculinos. Lo razonable, si hay más valores de un género que del otro, es usar la lista más corta en esa expresión `if`.

He planteado este ejemplo con una variable interpolada para que se pueda ver más directamente la razón del cambio, pero como es lógico no es necesario que en una string se incluya expresamente una variable para que esa variable determine algún cambio de género o número. El caso quizás más habitual es el de los juegos en los que el jugador puede elegir si su personaje es un hombre o una mujer. En inglés este cambio genera muy pocas variaciones, pero en español puede ser un auténtico quebradero de cabeza. Una sencilla frase como "You're my best friend.", que en inglés es aplicable a ambos géneros, nos obligará a aplicar una condición en nuestro script de traducción, para lo que primero debemos localizar la variable que define el género del personaje y luego sus posibles valores para usar uno de ellos como filtro de la expresión `if`. Es algo tedioso ya que debemos repasar todo el script para encontrar todas las strings que convendría desdoblar y hay que tener mucho cuidado para no generar bugs, pero el resultado vale la pena.

|Índice|

3.7.- Traducción de variables de texto (II): concatenaciones

Existen variables más complejas que las vistas en el punto anterior, cuyo valor de texto está generado por la suma de diversos elementos, y que necesitan un retoque adicional para mostrarse debidamente traducidas. Son las llamadas concatenaciones, generalmente resultado de una función python como la siguiente.

```
$ score = "Your score is " + strCalc + "out of " + strTotal + "."
```

En este ejemplo, el valor de la variable `"score"` es una frase en la que se interpolan los resultados de dos funciones (`strCalc` y `strTotal`) que estarán definidas en otra parte de los scripts. Y, en otro punto de ellos, habrá una string en la que se interpole esta variable `[score]` para mostrar en pantalla la puntuación del jugador. Esa interpolación deberíamos traducirla como `[score!t]`, pero ¿cómo conseguir que el texto se muestre en español? ¿Cómo extraer el valor de una variable definida así, con partes de frases?

Por una parte, debemos tener en cuenta que, al interpolar `[score!t]`, la string que buscaría Ren'Py para sustituirla por la traducción sería la frase completa. Por tanto, si nuestra puntuación es de 1 sobre 10, buscará en los scripts de traducción una string `old` que diga literalmente `"Your score is 1 out of 10"`. Y la única forma de que la encuentre es que la hayamos introducido nosotros manualmente, ya que esa frase no aparecerá íntegramente escrita en los scripts originales, por lo que es imposible que el Ren'Py SDK la extraiga. El problema entonces es que deberíamos escribir una string así para cada combinación de puntuación posible, y esa es una tarea ineficiente y, en muchas ocasiones, directamente inabarcable.

La solución es editar los scripts originales para introducir los elementos de texto de esa cadena en un paréntesis con un doble guion bajo delante. En nuestro ejemplo:

```
$ score = __("Your score is ") + strCalc + __("out of ") + strTotal + "."
```

El símbolo `__( )` permitirá que el Ren'Py SDK extraiga para traducir esas strings "parciales", y además, el juego almacenará el valor de la variable en el idioma en el que se estuviera jugando al pasar por ese punto de los scripts (a diferencia de lo que ocurre con una variable sin este símbolo, que se almacena siempre con su valor original).

El problema surgirá si cambiamos de idioma y continuamos nuestra partida en el idioma original. Porque ahora el valor de "score" que Ren'Py tiene almacenado es el que incluye la frase traducida, así que cuando juguemos en inglés la interpolación `[score]` devolverá esa frase en español. Por ello, tendríamos que editar la interpolación de la string original para convertirla en `[score!t]` y "deshacer" así la traducción. Y, obviamente, en nuestros scripts de traducción siempre debe aparecer la interpolación `[score!t]`.

Además, si los elementos `strCalc` y `strTotal` también fueran cadenas de texto, deberíamos buscar la función donde se calculen y meter todos sus posibles valores entre paréntesis y con doble guion bajo delante. Y, evidentemente, deberemos incluir en nuestro parche todos los scripts originales que hayamos tenido que editar para incluir el símbolo `__( )`, o bien optar por la sustitución de labels o el comando `init`

(dependiendo de si la variable se define en una label o en un bloque aparte) que enseguida explicaremos.

|Índice|

3.8.- Cuando todo lo demás falla: armas de "traducción masiva"

Pese a todo lo visto hasta ahora, es posible que alguna parte del texto todavía no aparezca debidamente traducida. Pero tranquilos: sigue habiendo soluciones a las que recurrir como última opción. Dejando de lado la más obvia, que pasa por editar los scripts originales para reescribirlos en nuestro idioma (porque, si hacemos eso, ¿para qué estoy escribiendo esta guía?), tenemos otras tres armas de "traducción masiva" con las que conseguiremos ver en nuestro idioma todos los textos del juego, aunque antes de recurrir a ellas deberíamos al menos ser conscientes de su impacto real y de sus posibles efectos secundarios.

|Índice|

3.8.1.- El doble guion bajo: La primera la acabamos de conocer, y no es otra que el símbolo `__( )`. Podemos emplearlo como solución de último recurso cuando, por la razón que sea, no encontramos la string en la que se interpola una variable de texto (algo que puede ocurrir en menús o pantallas de inventarios muy complejos, o cuando no tenemos tiempo ni ganas de investigar a fondo los scripts para encontrarlas). Es una solución, como digo, desesperada y que podría generar 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, deberíamos incluir la string con el valor de la variable entre los símbolos `__( )` también en esas expresiones.

Volviendo a nuestro ejemplo de la variable "fruit", podríamos haber añadido el símbolo `__( )` cuando encontramos sus posibles valores en los scripts originales, así:

```
default fruit = __("apple")
$ fruit = __("orange")
```

De esta manera, cada vez que en los scripts (tanto originales como de traducción) se interpole la variable `[fruit]`, el valor que aparecerá en pantalla será directamente "Manzana" o "Naranja". Esto es útil, como decía, si no encontramos una interpolación para traducirla por `[fruit!t]`. Pero, como hemos visto antes, el primer efecto secundario es que esos valores también se mostrarán en español si cambiamos el idioma del juego una vez pasado el punto en el que se ha "activado" esa variable, porque Ren'Py habrá almacenado el valor traducido y la interpolación original no ha sido sustituida por `[fruit!t]` (que en este caso revertiría la traducción).

El otro posible efecto secundario se produce cuando, además, esa variable se usa 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."
elif 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", y Ren'Py no encontraría ningún valor válido para decidir qué frase mostrar. Puede que no fuera un error crítico, y dependiendo de la programación el jugador podría no notar nada o, como mucho, un pequeño salto o incoherencia en los diálogos, pero desde luego no es así como el creador había diseñado su juego. Para solventarlo, también deberíamos utilizar el símbolo `__( )` en la expresión lógica del script original:

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

Si hubiéramos traducido todo correctamente esta edición no sería necesaria, ya que la expresión lógica se ejecutaría con su valor en el idioma original y luego ya se mostraría en pantalla la traducción del texto correspondiente. Pero, si nos vemos obligados a realizarla, tendremos que incluir el script editado en nuestro parche, salvo que optemos por la sustitución de labels que veremos más adelante.

|Índice|

3.8.2.- La función replace: La segunda arma de traducción masiva es más compleja, ya que requiere escribir un bloque de código python y ser extremadamente cuidadoso con la escritura de lo que queremos reemplazar, pero también es la más potente e indiscriminada. Se trata de crear una función que cambiará lo que haya que mostrar en pantalla, independientemente de dónde se encuentre o, incluso, de si se trata de una string completa o solo una parte de la misma. Se trata de algo así:

```
init python:
    def replace_text(s):
        s = s.replace('text 1', 'texto 1')
        s = s.replace('text 2', 'texto 2')

        return s
    config.replace_text = replace_text
```

La función puede escribirse en cualquier script, tanto original como de traducción, por lo que lo lógico es escribirla en uno de estos últimos. Y lo que hace es efectuar un reemplazo literal de caracteres, sustituyendo la primera expresión entrecomillada (lo que a modo de ejemplo aquí llamo `'text 1'`, `'text 2'`, etc.) por su pareja (`'texto 1'`, `'texto 2'`, etc.). Esta sustitución se ejecuta en el último instante antes de que el mensaje aparezca en pantalla, después de que Ren'Py busque y encuentre (o no) la traducción correspondiente: siempre que en el mensaje a mostrar aparezca esa combinación exacta de letras, el jugador verá el valor que hemos introducido como sustituto. Esto es útil para esas strings que el Ren'Py SDK no extrae automáticamente y que nosotros tampoco localizamos, sobre todo si tienen etiquetas de cambio de color o de fuente: al no haberlas encontrado en los scripts, no sabemos qué escribir en la línea `old` para que refleje el contenido literal del original y pueda activarse la traducción. Este reemplazo actúa solo sobre el texto, no sobre la string completa, y por ello funcionará siempre.

¿Y cuál es el problema? Pues que puede darse el caso de que una combinación concreta de letras también aparezca como parte de una frase perfectamente traducida, por lo que será sustituida por el valor de la función replace para esas letras dispuestas en ese orden, generándose una frase sin sentido. No es frecuente, pero puede pasar: imaginemos que ordenamos reemplazar la palabra "pint" por "pinta". Cuando en una string ya traducida del bloque de diálogos aparezca, por ejemplo, la palabra "pintura", en pantalla acabaremos leyendo "pintaura"; si tenemos un "pintor" leeremos "pintaor"; donde debería leerse "carpintero" aparecerá "carpintaero", etc. Por lo tanto, se recomienda usar esta función con cuidado y moderación (por ejemplo, solo para strings largas que sepamos con total seguridad que es imposible que aparezcan escritas literalmente así en español), y siempre de forma conjunta con la siguiente arma, ya que de lo contrario el reemplazo también se efectuará cuando juguemos en el idioma original.

|Índice|

3.8.3.- La detección de idioma: Por último, queda hablar de un arma que nos puede servir de apoyo para limitar los efectos de la anterior y facilitar otras acciones menos ortodoxas sobre los scripts originales. En cierta forma, es algo parecido a lo que pasa cuando incluimos una función para determinar qué string de traducción se mostrará por cuestiones de concordancia, solo que en este caso estaremos actuando sobre el script original, incluyendo una expresión lógica para que si se está jugando en el idioma original se ejecute el código original, y si se está jugando con la traducción se ejecute un código alternativo.

Para ello, debemos entender que Ren'Py conserva la elección de idioma en una variable interna llamada `preferences.language`, que es la que, entre otras cosas, le permite arrancar el juego en el último idioma en que se jugó, como ya vimos. Así pues, esta es la variable que adquiere el valor "Spanish" al seleccionar la traducción y permite que se ejecuten todas las funciones de traducción. Y, como cualquier

otra variable, podemos emplearla en una función condicional `if` para que Ren'Py muestre un texto concreto o ejecute un bloque de código determinado siempre que se cumpla la condición seleccionada.

Aplicándola a la función replace que acabamos de ver, lo que haremos será limitar ese reemplazo de caracteres a las ocasiones en que el texto a sustituir aparezca cuando se está jugando con la traducción activada, para que el jugador que opte por el idioma original no se vea afectado. De modo que la función presentaría este aspecto:

```
init python:
    if preferences.language = "Spanish":
        def replace_text(s):
            s = s.replace('text 1', 'texto 1')
            s = s.replace('text 2', 'texto 2')

            return s
        config.replace_text = replace_text
```

Como veis, debemos incluir la condición al principio del bloque, lo que nos obliga a aumentar la indentación de las siguientes líneas de código, que ahora solo se ejecutarán cuando estemos jugando con el idioma "Spanish". Pero, obviamente, esto no corrige las posibles sustituciones accidentales que explicábamos antes.

La variable de idioma nos permite también incorporar elementos traducidos en los scripts originales. Si la usamos en una función condicional antes de cualquier elemento de esos scripts, Ren'Py comprobará en qué idioma se está jugando y, en función de ello, mostrará el código original o un código alternativo que nosotros incluyamos, que además podríamos escribir ya directamente en español puesto que solo se va a mostrar cuando estemos jugando con la traducción activada. Por ejemplo, podríamos resolver una concatenación escribiendo esto en el script original donde la encontremos:

```
if preferences.language = "Spanish"
    $ score = "Tu puntuación es de " + strCalc + "sobre " + strTotal + "."
else:
    $ score = "Your score is " + strCalc + "out of " + strTotal + "."
```

Lo aconsejable sería que la expresión se escribiera en este orden, para que el bloque `else` ejecute el código original tanto cuando se juegue en el idioma original como en cualquier otro que ya exista o se implemente posteriormente. En este ejemplo, además, ya no haría falta añadir el apéndice `!t` a la traducción de la interpolación de la variable `[score]`, porque el valor se estaría calculando en español. Y, obviamente, si vamos a crear un parche para compartir nuestra traducción, deberemos incluir en él los scripts originales que hayamos editado. No es una solución elegante y evidencia que hemos sido incapaces de hacer las cosas bien, pero es útil saber que tenemos este último recurso.

|Índice|

4.- EL PARCHE

Una vez realizada con éxito nuestra traducción, podemos limitarnos a disfrutar en privado de nuestra gran obra. Pero, si nos lanzarnos a compartirla con el mundo, no necesitamos compilar y publicar el juego entero: basta con crear un parche con los archivos estrictamente necesarios para que nuestra traducción funcione en el ordenador de cualquier persona que ya tenga instalado el juego original.

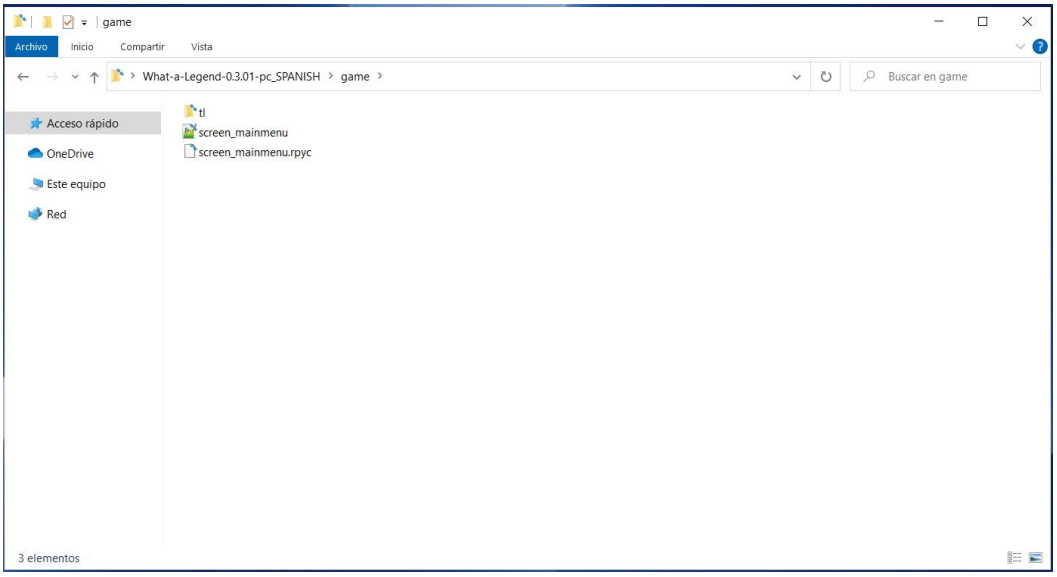
|Índice|

4.1.- El parche básico

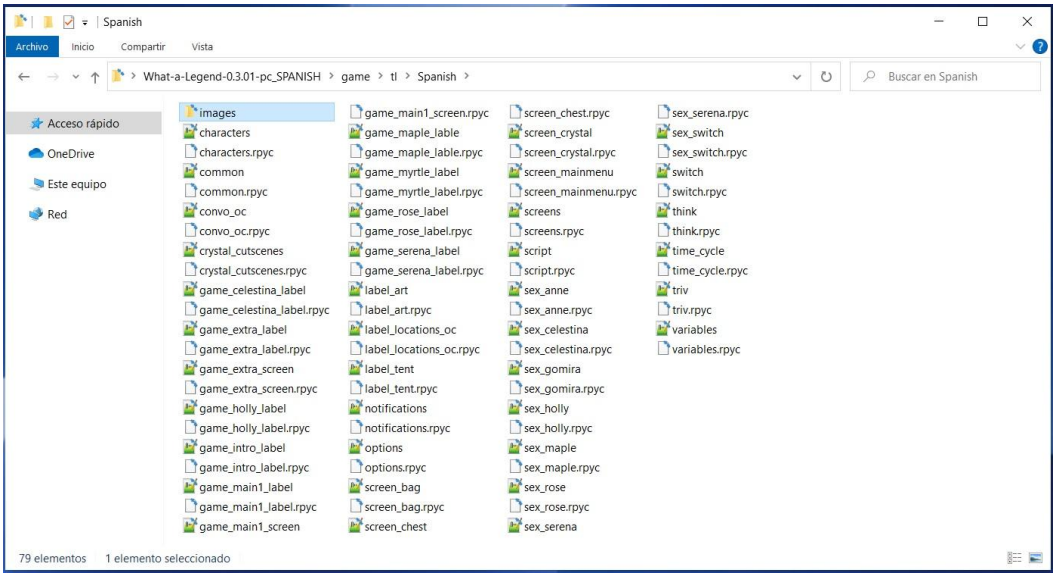
Partiendo de la base de que cualquier parche, para que funcione, debe respetar y emular la estructura de la carpeta original del juego, la forma más sencilla de montar un parche de traducción es crear una carpeta

nueva con el nombre del juego y algo que indique que se trata de un parche de traducción; dentro de esa carpeta, crearemos otra denominada “game” en la que incluiremos los scripts .rpy originales que hayamos editado (junto con sus .rpyc), y junto a ellos una carpeta “tl” con una subcarpeta “Spanish” que contendrá nuestros scripts de traducción (y, en su caso, las imágenes y fuentes necesarias para solventar los problemas del punto 3.3, según se ha ido explicando en la guía). El usuario solo tendrá que pegar la carpeta “game” de nuestro parche dentro del directorio raíz de su propio juego (donde está el ejecutable) y aceptar que se reemplacen todos los elementos coincidentes.

Por ejemplo, este podría ser el contenido de la carpeta “game” de un parche para “What a Legend!”:



Como en este ejemplo no hacía falta editar ningún archivo más para que la traducción se mostrara correctamente, en la carpeta “game” solo aparece el script “screen_mainmenu.rpy” (y su .rpyc), editado para incluir la opción de cambio de idioma y que sustituirá al original que tenga el jugador en su copia del juego. Además, lógicamente, está la carpeta “tl” que incluye la subcarpeta “Spanish” que teníamos en nuestro ordenador con la traducción completa, que tendría este aspecto:



Ahí podéis ver que, además de los scripts de traducción, se incluye una subcarpeta “images” con las imágenes que hemos tenido que traducir, que a su vez está estructurada exactamente igual que la carpeta que contiene las imágenes originales, como explicaba en el punto 3.4.

Como queda dicho, para que todo nuestro trabajo le llegue correctamente al jugador deberíamos incluir en nuestra carpeta “game” los scripts originales que hayamos editado para facilitar traducciones de palabras homónimas, archivos “gui.rpy” o “screens.rpy” retocados para cambiar el idioma o el tipo de letra, variables

de texto que hayamos tenido que traducir mediante el símbolo __( ), soluciones de emergencia para concatenaciones extrañas, etc. También podéis incluir todos los scripts originales que solo habéis modificado para que el Ren'Py SDK extraiga todas las strings con el símbolo _( ), por si queréis ahorrarle ese trabajo a un hipotético traductor a otro idioma, pero no sería imprescindible para los jugadores

Este había sido siempre mi procedimiento habitual; sin embargo, y debido a los inconvenientes técnicos y a los recelos que pueden generarse entre los jugadores (y desarrolladores) al reemplazar los archivos originales del juego, desde hace un tiempo vengo optando por otra forma de crear los parches, quizás algo más compleja, que explicaré después. Si pese a todo decidís crear un parche de este tipo, os recomiendo incluir **siempre** en él también los archivos .rpyc que correspondan a los scripts originales en formato .rpy que hayamos editado. Aunque se hayan modificado al editar sus .rpy base, si hemos cumplido el tercer mandamiento y 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), los .rpyc que tengamos en nuestro ordenador respetarán el AST con el que fueron compilados originalmente por el desarrollador del juego, que es el que también tendrá el jugador en los .rpyc de su copia del juego, si no los ha borrado. Al mantenerse en todo momento esa estructura básica del AST, en principio no debería haber problemas para retomar partidas que estuvieran guardadas antes de incorporar nuestra traducción.

¿Pasaría algo si solo incluyéramos los scripts en formato .rpy? Pues depende. Si el juego trae al descubierto los archivos .rpyc, 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 eso ya es responsabilidad del jugador).

El problema realmente importante puede surgir cuando los scripts del juego original están comprimidos en un fichero .rpa: si solo incluimos los .rpy editados, Ren'Py creará unos .rpyc nuevos en el ordenador del jugador, pero no se generarán con la misma estructura AST que tenían los que extrajimos del juego original, que fue con los que creamos la traducción. Y si además no incluimos en el parche todos los archivos .rpy del juego, Ren'Py tendrá que leer algunos archivos .rpyc que están todavía dentro del .rpa, y que también estarán compilados con un AST distinto a los nuevos .rpyc creados ahora fuera del .rpa. En esos casos, además del problema con las partidas guardadas, es muy probable 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 según su contenido literal) pero los diálogos de esos scripts editados se seguirían mostrando en el idioma original (porque Ren'Py no identifica correctamente el bloque de diálogos dentro del AST para asociarlo con la traducción). Es un error extraño pero que puede darse, sobre todo con juegos que fueron compilados originalmente con una versión del Ren'Py SDK anterior a la nuestra, con la que generamos la traducción, y que evitaremos incluyendo siempre en nuestro parche los archivos .rpyc que tengamos en la carpeta "game": aunque hayan sido actualizados con nuestros cambios sobre los .rpy, mantienen la estructura del AST de los originales que extrajimos con el UnRen. Más vale prevenir.

|Índice|

4.2.- El parche avanzado

Como acabamos de ver, creando un parche "básico" estaremos obligando al jugador a sustituir los archivos originales del juego por nuestros archivos editados. Y más allá de los posibles problemas con las partidas guardadas o incluso con el buen funcionamiento de la traducción, al sustituir archivos originales estamos abriendo la puerta a varias situaciones que no son fácilmente reversibles. El caso más extremo es que hayamos generado un bug sin darnos cuenta; lo más habitual, sin embargo, es que hayamos modificado elementos estéticos del juego que se verán afectados también cuando se juegue en otro idioma. Y esto es algo a tener en cuenta porque nunca se sabe dónde puede acabar nuestra traducción, o porque es posible que algún usuario la pruebe y decida que prefiere seguir usando el idioma original.

En esos y otros ejemplos, muchos jugadores preferirían poder eliminar nuestro parche de un plumazo y recuperar la versión original del juego sin más rodeos, y no tenemos por qué negarles esa opción obligándoles a conservar unos cambios estéticos que no necesitan y, probablemente, tampoco quieran. Además, creando un parche de traducción que no afecte a los archivos originales quizás podamos convencer más fácilmente al desarrollador del juego para que incorpore nuestro trabajo a su versión oficial, algo que siempre es motivo de orgullo y satisfacción y tal vez hasta una fuente de ingresos económicos legales (o, al menos, más honesta que intentar cobrar directamente a los usuarios de nuestra traducción a espaldas del creador del juego).

Afortunadamente, Ren'Py y el lenguaje python nos ofrecen diversas soluciones para que todas esas modificaciones necesarias para que el juego se muestre correctamente en nuestro idioma queden perfectamente guardadas dentro de la carpeta "game/tl/Spanish", permitiendo que quien quiera eliminar la traducción y recuperar la versión original del juego solo tenga que borrar esa carpeta. Eso sí: requiere algo más de trabajo y una mínima comprensión de cómo funciona la programación de un juego Ren'Py.

|Índice|

4.2.1.- El archivo zzz.rpy y el comando init: El primer paso que daremos será crear un documento en el que iremos introduciendo todas aquellas modificaciones que hemos hecho en nuestros scripts originales para que la traducción se mostrara perfecta en nuestro ordenador. Para ello, abriremos un documento nuevo con nuestro editor de texto y lo guardaremos con el nombre que queramos (obviamente, uno fácilmente identificable y que no coincida con ninguno de los existentes en la carpeta de traducción), escribiendo siempre la extensión .rpy como parte del nombre elegido. Por ejemplo, "zzz.rpy" o algo parecido, que guardaremos en nuestra carpeta "game/tl/Spanish" como archivo de texto plano sin formato específico (Ren'Py ya se encargará luego de identificarlo como script, al tener la terminación .rpy).

Por seguir un orden lógico, en ese script empezaremos incluyendo el elemento que debería estar presente en todas las traducciones: la opción de cambio de idioma. Lo que haremos será copiar todo el código correspondiente a la pantalla original de preferencias, que se extiende bastante más allá de lo que se ve en la siguiente imagen *(nota: todo este bloque supone que el juego usa la pantalla por defecto de Ren'Py; si usara una personalizada, la que habría que copiar sería obviamente esa pantalla personalizada)*.

```
759 ## Preferences screen ###########################################################
760 ##
761 ## The preferences screen allows the player to configure the game to better suit
762 ## themselves.
763 ##
764 ## https://www.renpy.org/doc/html/screen_special.html#preferences
765
766 screen preferences():
767
768     tag menu
769
770     use game_menu(_("Preferences"), scroll="viewport"):
771
772         vbox:
773
774             hbox:
775                 box_wrap True
776
777                 if renpy.variant("pc") or renpy.variant("web"):
778
779                     vbox:
780                         style_prefix "radio"
781                         label _("Display")
782                         textbutton _("Window") action Preference("display", "window")
783                         textbutton _("Fullscreen") action Preference("display", "fullscreen")
784
785                 vbox:
786                     style_prefix "radio"
787                     label _("Rollback Side")
788                     textbutton _("Disable") action Preference("rollback side", "disable")
789                     textbutton _("Left") action Preference("rollback side", "left")
790                     textbutton _("Right") action Preference("rollback side", "right")
791
792                 vbox:
793                     style_prefix "check"
794                     label _("Skip")
795                     textbutton _("Unseen Text") action Preference("skip", "toggle")
796                     textbutton _("After Choices") action Preference("after choices", "toggle")
797                     textbutton _("Transitions") action InvertSelected(Preference("transitions", "toggle"))
798
799                 ## Additional vboxes of type "radio_pref" or "check_pref" can be
800                 ## added here, to add additional creator-defined preferences.
801
802             null height (4 * gui.pref_spacing)
```

La idea es copiar todo ese bloque de código indentado, empezando por `screen preferences()`: hasta el siguiente comando que empiece justo en el margen izquierdo, ya sin sangría (generalmente son una serie de comandos que empiezan por la palabra `style`). Todo lo que haya entre medias es la pantalla de preferencias y es lo que tenemos que trasladar íntegramente a nuestro archivo para sustituir a la original.

Una vez tengamos ese código trasplantado en el nuevo archivo, empezaremos a modificarlo.

En primer lugar, si aún no habíamos establecido la opción de cambio de idioma, debemos añadirla ahora. Como ya vimos, el código necesario en el caso de una pantalla de preferencias básica de Ren'Py era este (recordad que iría al mismo nivel de indentación que las opciones "Rollback Side" y "Skip" y que hay que marcar las sangrías con la barra espaciadora):

```
vbox:
    style_prefix "radio"
    label _("Language")
    textbutton _("English") action Language(None)
    textbutton _("Español") action Language("Spanish")
```

A continuación, necesitamos que Ren'Py sepa que debe usar esta nueva pantalla en lugar de la original. Para ello emplearemos **el comando `init`**, que le indica a Ren'Py las funciones que debe ejecutar al arrancar el juego (antes de que se muestre nada en la pantalla del jugador) y en qué orden. A ese comando `init` se le puede asignar un valor numérico entre `init -999` e `init 999`, que será lo que determine el orden en el que será cargado en la memoria interna de Ren'Py (de menor a mayor init). Este orden de cargado es importante porque, si en este proceso de arranque Ren'Py encuentra dos elementos con el mismo nombre, en su memoria solo quedará almacenado el que se haya cargado en último lugar; por lo tanto, el elemento con mayor init sobrescribe a cualquiera que pudiera existir con el mismo nombre. Y si los elementos tienen el mismo init, quedará grabado el que se encuentre en un script cuyo nombre sea posterior por orden alfabético.

En condiciones normales, la programación de la pantalla original de preferencias no incorpora de forma explícita este comando, pero se carga internamente en `init 0` (después de los elementos con init negativo y antes de cualquier elemento con un init mayor que 0). Por lo tanto, si a la nueva pantalla de preferencias de nuestra carpeta de traducción le asignamos un valor `init` superior a 0, Ren'Py la cargará en su memoria interna después la pantalla de preferencias original, sobrescribiéndola, de forma que al jugar siempre mostrará la nuestra. En resumen, justo encima de `screen preferences()`: tendremos que escribir esta línea de código:

```
init offset = 1
```

Este comando le indica a Ren'Py que la siguiente definición que encuentre en el script debe cargarse en el momento init que hemos indicado, en este caso 1.

```
### SPANISH SCREENS ###

# Preferences Screen

init offset = 1
screen preferences():

    tag menu

    use game_menu(_("Options"), scroll="viewport"):

        vbox:

            hbox:
                box_wrap True

                if renpy.variant("pc"):

                    vbox:
                        style_prefix "radio"
                        label _("Display")
                        textbutton _("Window") action Preference("display", "window")
                        textbutton _("Fullscreen") action Preference("display", "fullscreen")

                vbox:
                    style_prefix "radio"
                    label _("Rollback Side")
                    textbutton _("Disable") action Preference("rollback side", "disable")
                    textbutton _("Left") action Preference("rollback side", "left")
                    textbutton _("Right") action Preference("rollback side", "right")

                vbox:
                    style_prefix "check"
                    label _("Skip")
                    textbutton _("Unseen Text") action Preference("skip", "toggle")
                    textbutton _("After Choices") action Preference("after choices", "toggle")
                    textbutton _("Transitions") action InvertSelected(Preference("transitions", "toggle"))

                vbox:
                    style_prefix "radio"
                    label _("Language")
                    textbutton _("English") action Language (None)
                    textbutton _("Español") action Language ("Spanish")
```

Alternativamente, podemos cambiar la primera línea de la pantalla, así:

```
init 1 screen preferences():
```

Para estos ejemplos he usado un init 1, pero valdría cualquier número positivo (o, al menos, superior al que el desarrollador hubiera asignado a su pantalla original, ya que en ocasiones sí se le asigna un `init` propio, distinto de 0). Una vez hecho eso, ya tendremos nuestra pantalla de preferencias perfectamente activa.

Ahora hay que tener en cuenta algo importante: si cuando generamos nuestros scripts de traducción aún no habíamos incluido esta opción en la pantalla de preferencias original, es muy probable que tengamos que traducir la string "Language" (salvo que ya existiera idénticamente escrita dentro de los scripts del juego). Al estar incluida dentro del símbolo `_( )` podríamos pensar que, si generásemos nuevamente los scripts de traducción, el Ren'Py SDK la extraería, pero no: no extraerá ni esta ni cualquier otra cadena de texto traducible que esté presente en un script guardado en la carpeta "tl/Spanish". Habría otras formas de arreglarlo, pero en este caso es mejor recurrir a la "extracción manual" que expliqué en el punto 2.3, ya que solo tenemos que traducir la string del encabezado de la opción, "Language", y no los nombres de los idiomas (que como ya dije deberían permanecer siempre escritos en su idioma original).

Así que, debajo de nuestra pantalla de preferencias, escribiríamos este código, con la función `translate` sin sangría de ningún tipo (y solo, repito, si aún no habíamos traducido esa string, ya que de lo contrario ya la tendríamos en el script "screens.rpy" de nuestra carpeta de traducción y estaríamos duplicándola, con lo que el juego no arrancaría):

```
translate Spanish strings:
    old "Language"
    new "Idioma"
```

A continuación, en este script zzz.rpy también deberíamos escribir las funciones para conseguir que el juego se inicie en español. Recordemos, para que el juego se inicie por primera vez en español (si no se había iniciado antes en otro idioma), definiríamos la variable `config.default_language`:

```
define config.default_language = "Spanish"
```

Si el juego ya incorporara esta misma variable para otro idioma, deberíamos escribir esta línea de código bajo un bloque `init` que se cargue con posterioridad. Es decir, así:

```
init offset = 1
define config.default_language = "Spanish"
```

Y para conseguir que el juego arranque siempre en español, escribiremos esta línea (y nos olvidaremos de la anterior, que ya no tendría sentido al quedar relegada por esta en el orden de prioridades de Ren'Py):

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

En este caso no suele ocurrir que ya exista esa variable definida para otro idioma, pero lo solucionaríamos igual que antes, creando un bloque con un `init` superior al original (que por defecto es 0).

Y así con todo. En este script seguiríamos escribiendo aquellas funciones y variables que hubiéramos tenido que añadir para modificar el aspecto visual de la interfaz (lo que vimos en el apartado de cambios de fuente y estilos), y también otras soluciones desesperadas como la función replace, ya que Ren'Py las encontrará sin problemas y las aplicará a pesar de estar "escondidas" en nuestra carpeta de traducción.

|Índice|

4.2.2.- Reemplazo de labels: Pero, ¿qué pasa con esas otras ediciones de los scripts originales que afectan directamente al contenido de una label del juego, como las que nos sirven para traducir de forma diferente

palabras idénticas o nos permiten traducir algo gracias al doble guion bajo? Nuestra copia personal del juego tendrá esas ediciones en los scripts originales (lo que nos ha permitido realizar la traducción), pero si queremos que el usuario de nuestro parche no tenga que reemplazar absolutamente ningún archivo original de su propia copia, debemos recurrir a la sustitución de labels.

Para ello, copiaremos **íntegramente** las labels editadas y las pegaremos en este archivo zzz.rpy (o en otro que creemos del mismo modo para ese fin concreto, por si queremos separar las opciones de configuración de las modificaciones que afectan a los textos en sí para tenerlo todo más organizado). A continuación cambiaremos el nombre de esta label que acabamos de pegar, preferiblemente añadiéndole un prefijo o un sufijo que nos ayude a distinguirla de la original (en realidad, lo único importante es que el nuevo nombre esté en minúsculas, no contenga acentos y no coincida con el de otra label existente). Suponiendo que la label a editar fuera, por ejemplo, la label start, podríamos renombrarla en nuestro archivo zzz.rpy como label start_es. Y por último, escribiríamos esta función, sin indentación:

```
define config.label_overrides = {”start”:”start_es”}
```

Es importante escribir bien todos los símbolos: el nombre de cada label va entre comillas, los dos puntos establecen que la segunda label se ejecutará en lugar de la primera, y todo ello dentro de unas llaves. Si hubiera más labels a reemplazar, las copiaríamos y renombraríamos en el archivo zzz.rpy y las añadiríamos a esa misma función, separando cada pareja con una coma. Así:

```
define config.label_overrides = {”start”:”start_es”, ”example1”:”example1_es”}
```

Ahora bien, una vez que le hemos dado la orden a Ren’Py de que ejecute nuestra label editada en vez de la original, la traducción que tuviéramos para los diálogos de esa label ya no vale, puesto que en la función de traducción el nombre de la label forma parte del código de identificación de cada línea. Además, si generásemos los scripts de traducción, ya hemos visto que no obtendríamos ningún resultado porque esta nueva label se encuentra en un script que está dentro de la carpeta de traducción y que por tanto no sería escaneado por el Ren’Py SDK.

No obstante, la solución es relativamente sencilla: iremos al script de traducción afectado (donde tengamos la traducción de la label original), y efectuaremos una búsqueda y reemplazo del nombre de la label, de modo que todos los códigos que contuvieran el nombre de la label original pasen a contener el nombre de la nueva. Para el ejemplo de que la label a sustituir fuera la label start, donde antes tuviéramos `translate Spanish start_XXXXXXXX:`, ahora debería poner `translate Spanish start_es_XXXXXXXX:`

Y así con todas las líneas. Solo hay que prestar un poco de atención para cambiar todas las de esa label y no tocar ninguna más. Afortunadamente, el cambio de nombre de label no afecta al código alfanumérico de encriptación MD5 que aparece a continuación (aquí sustituido por XXXXXXXX), ya que no hemos modificado el contenido de esas líneas, por lo que con ese simple cambio volveremos a tener una traducción plenamente válida. Recordemos que todas las strings de esa label que se traducen mediante la función de traducción literal (opciones de los menús que hemos editado para solucionar el problema de homonimia, o valores en texto de variables concatenadas) no dependen del nombre de la label en que aparezcan, por lo que las traducciones que hubiéramos hecho de ellas seguirían siendo válidas en todo momento.

|Índice|

Eso era todo. Espero que esta guía os haya resultado útil.
Si tenéis cualquier duda, sugerencia o corrección (especialmente), no dudéis en contactar conmigo.
¡Saludos!