Traduzione di giochi Ren'Py:

Una guida, di moskys

moskys # 0576

¡Hola!

Chiamami moskys. Come sicuramente saprai, se sei arrivato a questa guida, Ren'Py è un motore di gioco gratuito e open source (utilizzato per la creazione di romanzi visivi, in generale) che è diventato uno dei software più popolari utilizzati dai creatori dilettanti per sviluppare i loro giochi progetti. Basato sul linguaggio di programmazione Python, la sua semplicità e flessibilità è un grande vantaggio, poiché consente di ottenere risultati più che accettabili senza una conoscenza preliminare della programmazione. E una delle tante funzioni che Ren'Py include è quella di facilitare la traduzione dei giochi creati con questo programma.

Quindi, dopo quasi tre anni giocando e traducendo manualmente in spagnolo (e per PC) diversi giochi di varia complessità, mi sono incoraggiato a scrivere questa guida per spiegare il processo a chiunque voglia iniziare a tradurre o sia abbastanza curioso di sapere cosa io trascorro il mio tempo libero.

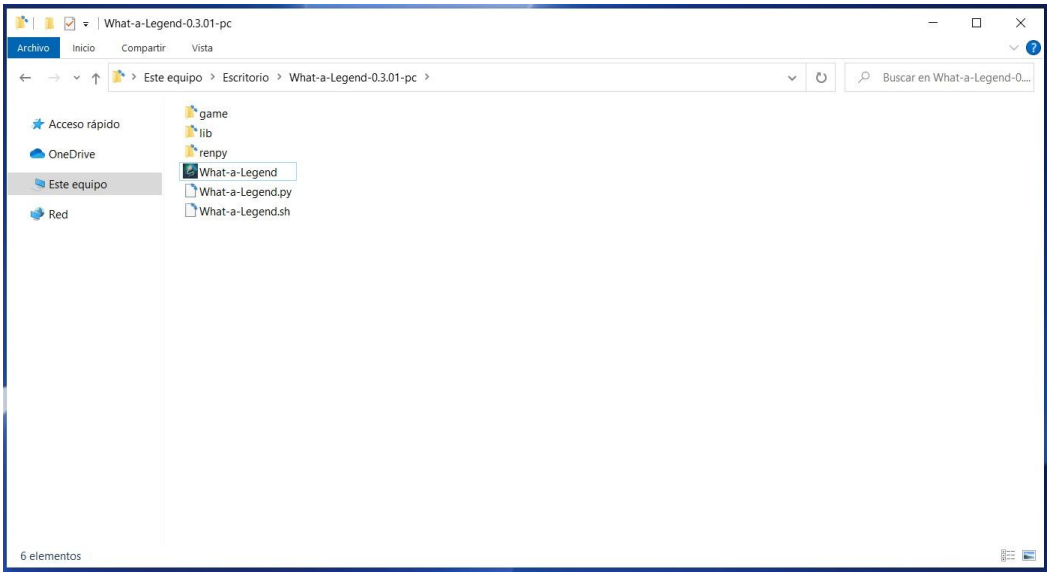
STRUMENTI ESSENZIALI (E GRATUITI) - versioni al 31-dic-2020

- **Ren'Py SDK 7.3.5. ->** https://www.renpy.org/latest.html
- **UnRen v.09.dev ->** https://f95zone.to/threads/unren-bat-v0-8-rpa-extractor-rpyc-decompilerconsole-developer-menu-enabler.3083/
  (collegamenti a un forum con contenuti per adulti)
- **Editor di testo:**
  - **Blocco note ++ 7.9.1 ->** https://notepad-plus-plus.org/downloads/
  - **Atomo ->** https://atom.io/

1. INTRODUZIONE

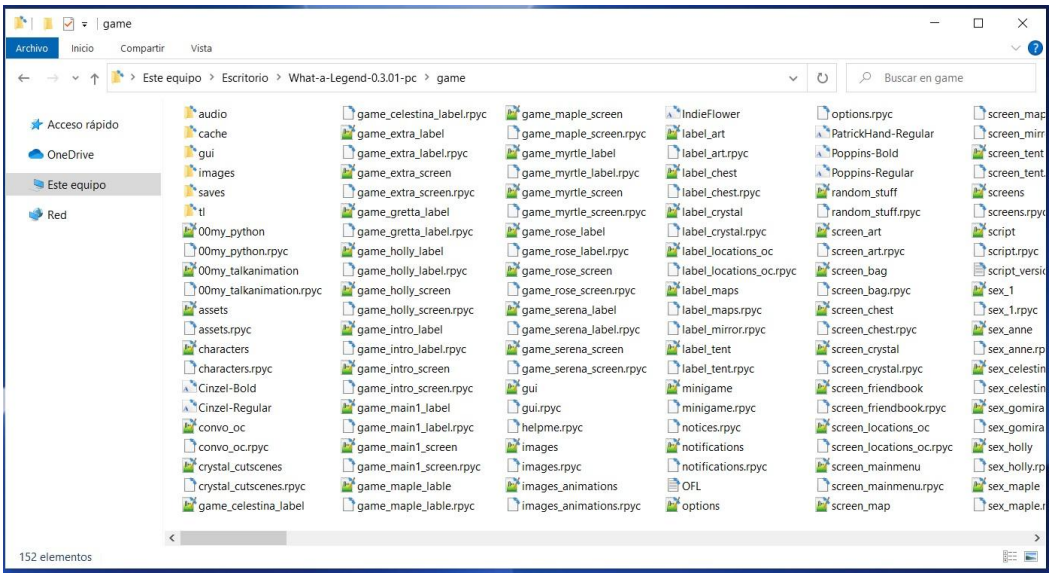
1.1.- Sotto il cofano di un gioco Ren'Py

È ovvio, ma per tradurre un gioco, devi prima avere un gioco. Quindi cerca qualsiasi gioco Ren'Py che hai scaricato e dai un'occhiata alla sua cartella principale. Vedrai qualcosa di simile:



Di solito, eseguiamo semplicemente il file eseguibile e riproduciamo. Ma ora diamo un'occhiata alle sottocartelle dell'albero che vediamo lì, che è dove avviene la magia. In linea di massima, nella cartella "renpy" possiamo trovare i file contenenti le funzioni preprogrammate, e nella cartella "lib" abbiamo tutti i file tecnici che fanno girare i giochi sui vari sistemi operativi. Queste due cartelle ci consentono di giocare senza scaricare nessun altro programma specifico per esso; potremmo dire che quelle cartelle trasformano i giochi Ren'Py in programmi autoeseguibili.

Il gioco stesso si trova nella cartella "game". Niente panico, questo è particolarmente complesso:



Ci sono diverse sottocartelle che possiamo trascurare per ora per concentrarci sui singoli file. Quando impacchettano i loro giochi, Ren'Py consiglia ai creatori di lasciare tutti i file in bella vista, come "Che leggenda!" gli sviluppatori lo fanno. Questo ci permette di vedere tutta una serie di file apparentemente clonati, alcuni con estensione .rpy e altri con estensione .rpyc.

| Inizio |

1.2.- Tre concetti fondamentali e due comandamenti

Questi file nella cartella "gioco" sono i file **script.** È in quegli script che gli sviluppatori includono tutto il codice di programmazione e i testi del gioco. Per fare ciò, scrivono semplicemente ciò di cui hanno bisogno per scrivere in un editor di testo e salvano il file con un'estensione specifica di Ren'Py denominata .rpy.

Quindi, utilizzando un programma di base come Blocco note di Windows (o, preferibilmente, uno migliore, come Atomo o Notepad ++ , per citarne due gratuiti e semplici usati nella programmazione), possiamo aprire quei file .rpy e vedere cosa significa creare un gioco Ren'Py. Utilizzando "What a Legend!" ad esempio, se apriamo il file denominato "convo_oc.rpy", nelle sue prime righe possiamo vedere questo:

```
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."
```

Qualsiasi cosa a destra di un simbolo # è un commento che non apparirà nel gioco, ma che i creatori usano per guidare se stessi attraverso il loro codice. Dopo vedremo quanto può essere utile questo simbolo per i traduttori.

Nella riga 3 vediamo una parola, **etichetta,** che sarà importante. Le etichette sono parti dello script. La dimensione di queste parti dipende dallo sviluppatore del gioco, ma generalmente corrispondono a una scena specifica. In questo caso, questa etichetta corrisponde a una conversazione che apparirà sullo schermo ogni volta che i giocatori eseguono un'azione che la attiva. A causa del linguaggio Python, tutto ciò che accade in questa scena è codificato con un rientro.

> **Il primo comandamento di Ren'Py:** rispetterete sempre i rientri e non li contrassegnerete MAI con Tab, ma con la barra spaziatrice (4 spazi, di solito). Se Ren'Py rileva una tabulazione o un rientro errato da qualche parte negli script (inclusi i file di traduzione) il gioco non si avvia.

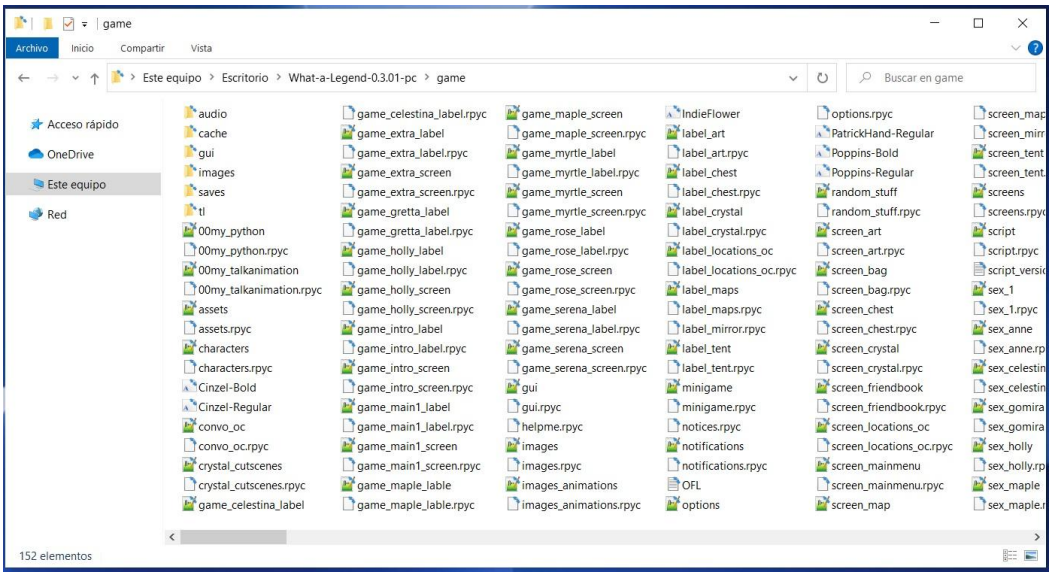
Se scorriamo lo script verso il basso, all'interno di questa stessa etichetta "convo_gate" vedremo diverse funzioni per mostrare e nascondere le immagini e per determinare quale parte della conversazione verrà visualizzata in base alle variabili del gioco attivate dai giocatori durante il loro playthrough. Inoltre, possiamo vedere alcune frasi citate: queste sono le linee di dialogo che appariranno sullo schermo. Quelle frasi citate sono quelle che dovremo tradurre e sono chiamate **stringhe,** o stringhe di testo.

> **Secondo comandamento di Ren'Py:** chiuderai sempre le virgolette. Le virgolette non chiuse (o le virgolette non aperte, per quello che vale) impediranno l'avvio del gioco.

| Inizio |

1.3.- Come funziona (penso) Ren'Py? Il terzo comandamento

Supponendo che non sia un esperto di informatica ed è molto probabile che dirò qualcosa di stupido, cercherò di spiegare molto rapidamente cosa succede quando eseguiamo un gioco Ren'Py.



Ricordi l'elenco dei file clonati all'interno della cartella "gioco"? Bene, non sono esattamente cloni. Ren'Py esegue i file .rpyc, che sono una raccolta di file modificabili salvati con estensione .rpy. Questa compilazione si verifica per la prima volta quando gli sviluppatori aprono il gioco nel loro ambiente di sviluppo. Viene generato un AST (abstract syntax tree) in base al contenuto disponibile in quel momento e questo AST sarà la base per tutto ciò che verrà aggiunto successivamente nelle successive compilazioni. Seguendo i suoi algoritmi, Ren'Py assegnerà un'identificazione a ciascun elemento di programmazione dei file .rpy in base alla sua posizione nello script e alla sua natura (testo, variabile, funzione, ecc.), E con quelle identificazioni che creerà. rpyc file con lo stesso nome che verranno letti durante il gioco.

Ogni volta che il gioco viene avviato, Ren'Py cerca i file .rpy e, se presenti, li confronta con i rispettivi .rpyc, cercando le identificazioni create durante la compilazione iniziale. Se sono state apportate modifiche al file .rpy dall'ultima volta che il file .rpyc è stato compilato, Ren'Py rileverà sia gli elementi non modificati che quelli che sono stati riposizionati, rispetterà le loro precedenti identificazioni e aggiornerà il file. rpyc con le modifiche. Semplificando molto, immaginiamo che la riga numero 3 del file .rpy sia inizialmente compilata in modo da identificare questa riga con un "3" nel file .rpyc. Se nelle versioni future del file .rpy quella riga non viene modificata ma diventa la riga numero 7, rimarrà "3" nel file .rpyc. Quindi, anche se la finale.

Se a un certo punto li cancelliamo, Ren'Py genererà nuovi file .rpyc ma utilizzando la versione corrente degli script .rpy. E questo significa che le relazioni e le identificazioni che verranno ora create nell'AST non saranno le stesse dei file originali, perché vengono create da uno script .rpy che assomiglia a qualcosa di simile al primo. Prendendo quell'assurdo esempio di prima, la riga 7 del file .rpy che era chiamato "3" nel .rpyc cancellato perché era stato trasferito dalle versioni precedenti, ora sarebbe stata identificata con un "7" mentre l'AST viene generato da zero. Il gioco verrà avviato e i nuovi giochi non dovrebbero essere interessati, ma le funzioni che utilizzano i riferimenti AST, come il caricamento di giochi salvati nelle versioni precedenti, potrebbero smettere di funzionare correttamente. E, in alcuni casi che vedremo più avanti , anche le traduzioni potrebbero avere qualche problema.

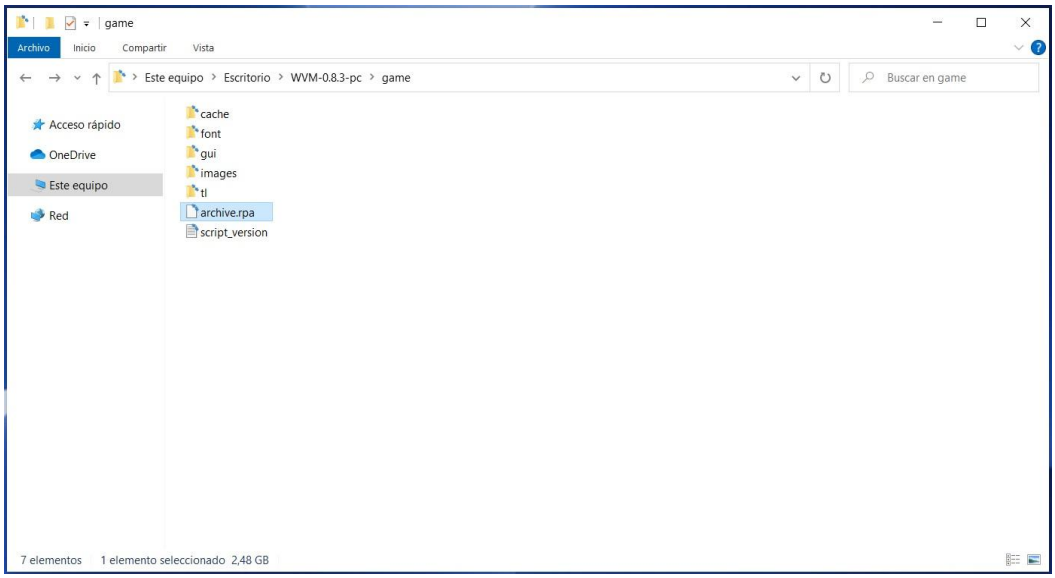
Il terzo comandamento di Ren'Py: Non eliminare i file .rpyc dal gioco originale, in quanto ciò può causare problemi imprevisti ai giocatori.

| Inizio |

1.4.- Archivi UnRen e .rpa

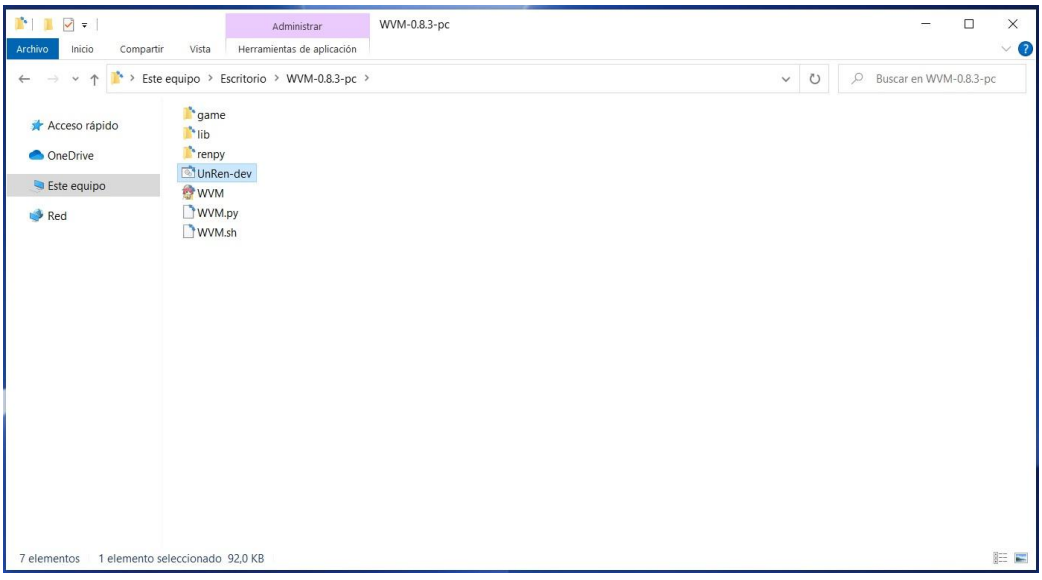
È possibile che, quando hai aperto la cartella "gioco", tu non abbia visto nulla di tutto ciò che ho appena menzionato. Questo perché, quando si tratta di creare il gioco per il rilascio, Ren'Py offre agli sviluppatori diverse opzioni. Quello consigliato è di includere tutti i singoli script sia in formato .rpy che .rpyc, come abbiamo visto nella sezione "What a Legend!" esempio, ma è solo una raccomandazione. Come un gioco Ren'Py solo ha bisogno dei suoi script .rpyc per funzionare , alcuni sviluppatori includono solo gli script .rpyc. In questo modo il gioco ha una dimensione leggermente più piccola ei giocatori non hanno accesso diretto al codice del gioco (che potrebbe essere un motivo più importante).

E c'è un'altra possibilità (abbastanza comune, direi), poiché potremmo scoprire che nella cartella "gioco" non ci sono script con estensioni .rpy o .rpyc, poiché lo sviluppatore ha scelto un'altra opzione data da Ren'Py per costruire il gioco: comprimendo gli script e altri file (come le immagini) in un file .rpa. In questo modo, invece di un elenco di file .rpy e .rpyc, potremmo avere qualcosa di simile nella cartella "game":

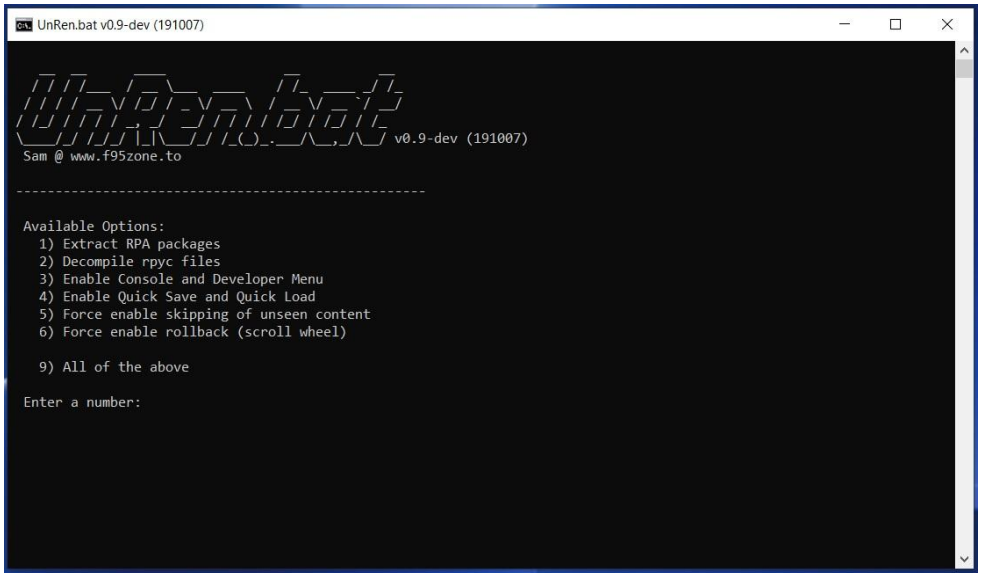


Innanzitutto possiamo confermare che, come probabilmente già saprai, questo non è un problema. Il gioco funzionerà perfettamente così com'è, ma per tradurlo dovremo fare qualche deviazione in più, visto che abbiamo bisogno degli script in formato .rpy. Fortunatamente, ci sono diversi strumenti che eseguono le attività necessarie senza che dobbiamo imparare il linguaggio Python. Per la sua semplicità, penso che sia il più gestibile **UnRen.** LINK *(attenzione: collegamento a un forum con molti contenuti per adulti)*

Una volta scaricato UnRen, lo decomprimiamo e lo incolliamo nella cartella principale del gioco di nostro interesse, allo stesso livello dell'eseguibile del gioco. Immagino sia più facile da capire se lo vedi:



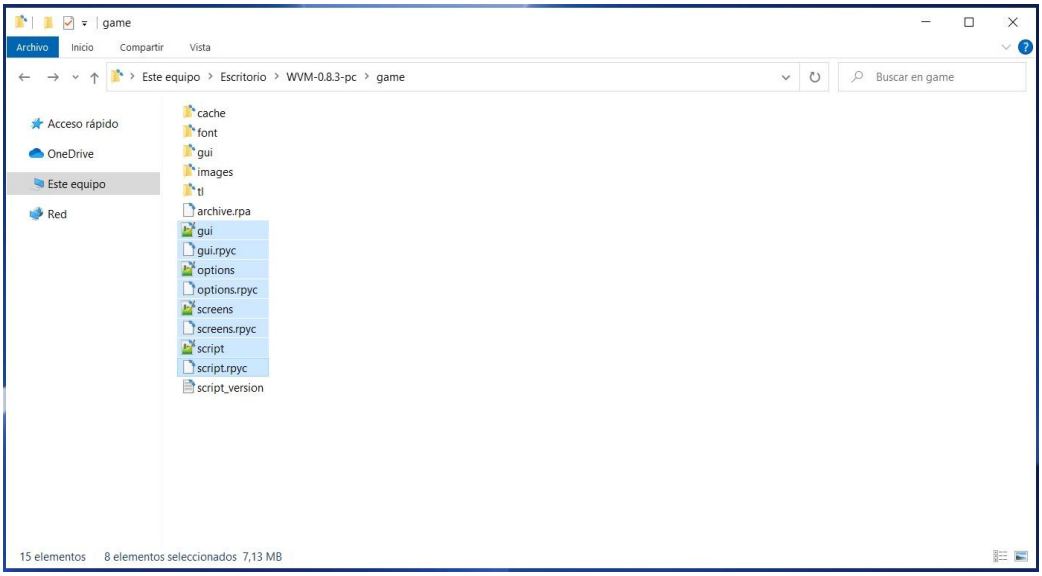
Quindi è solo questione di eseguirlo e selezionare ciò che vogliamo fare. Questa è la schermata iniziale di UnRen:



Come puoi vedere, qui useremo solo la tastiera. Dobbiamo solo premere il tasto corrispondente a ciò che vogliamo fare. L'opzione 1) "decomprimerà" i file .rpa, estraendo tutto il suo contenuto nella cartella "gioco". Può darsi che all'interno di quel file .rpa ci fossero solo script con estensione .rpyc, quindi, dopo aver terminato l'opzione 1), dovremmo chiedere a UnRen di eseguire l'opzione 2), che consiste nel decompilare i file .rpyc per creare alcuni script in formato .rpy che potremmo già utilizzare per la nostra traduzione.

Possiamo semplicemente selezionare l'opzione 9), che in un unico passaggio fa la stessa cosa di 1) e 2) e abilita anche l'opzione per aprire la console del gioco e il menu sviluppatore (utile per esaminare e modificare le variabili, per accedere a parti specifiche di lo script, e anche per ricaricare automaticamente il gioco quando introduciamo una modifica nella traduzione), così come altre opzioni che lo sviluppatore del gioco potrebbe aver disabilitato.

Bene, lasciamo che UnRen faccia il suo lavoro e alla fine, se torniamo alla cartella "gioco", vedremo che, dove prima c'era solo un file con estensione .rpa, ora compaiono gli script che sono arrivati compresso al suo interno.



Ora potremmo iniziare a tradurre. Ma, in primo luogo, lasciatemi usare un paragrafo per spiegare cosa succederà con il gioco. Ora, all'interno della cartella "game" ci sono script con estensione .rpy, script con estensione .rpyc e, inoltre, all'interno del file "archive.rpa" ci saranno quegli stessi script .rpyc (e forse anche gli script .rpy quelli). Quindi ora abbiamo file clonati.

Fortunatamente, Ren'Py è progettato per risolvere questi problemi di clonazione in modo semplice: **se trova due script con lo stesso nome, esegue quello più recente.** E, in questo caso, i più recenti sono quelli che abbiamo appena estratto con UnRen. Pertanto, potremmo eliminare il file "archive.rpa" (e sarebbe anche consigliato, a patto di poterlo recuperare in seguito, per ogni evenienza).

E ancora un'altra precisazione prima di andare avanti. Se il file .rpa conteneva sia gli script .rpy che .rpyc, quelli che vediamo ora nella cartella "game" sono esattamente gli stessi che provenivano dal computer dello sviluppatore del gioco, in quanto sono stati appena estratti da UnRen. Ma se nel file .rpa c'erano solo script con estensione .rpyc, gli script .rpy che ora abbiamo nella cartella "game" sono solo un'approssimazione creata da UnRen in base all'algoritmo identificato utilizzato per compilare gli script .rpyc. Quindi potrebbero non essere esattamente come gli script .rpy originali: sono adatti solo per creare gli script .rpyc correnti. Ad esempio, tutti i commenti che lo sviluppatore avrebbe annotato dopo un simbolo # vengono persi, poiché non vengono mai compilati negli script .rpyc e logicamente UnRen, leggendo solo i .rpyc,

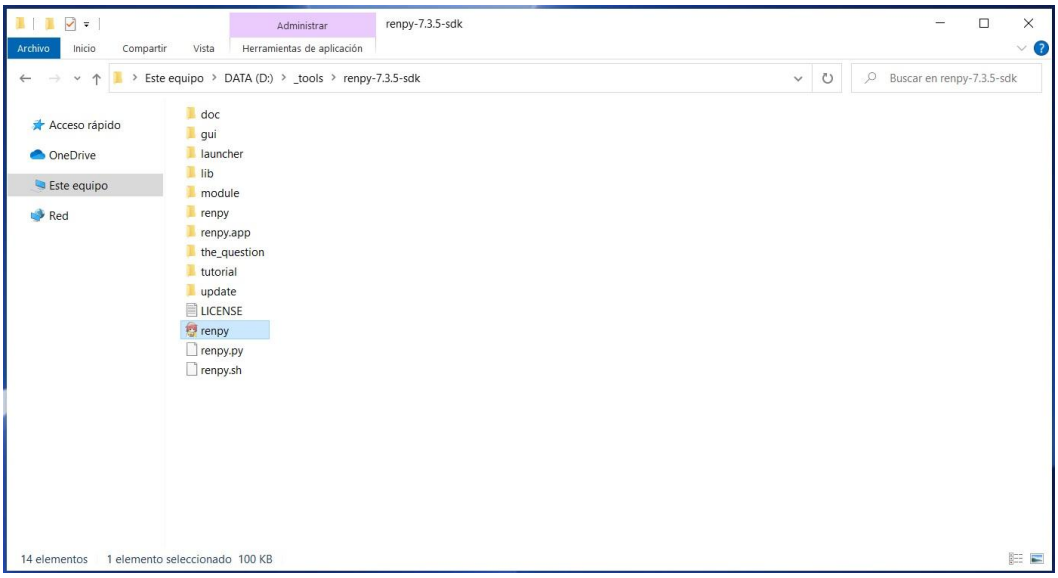
| Inizio |

1.4.- Ren'Py SDK

Come abbiamo visto, i giochi Ren'Py funzionano senza installare alcun software aggiuntivo per leggerli. Sono autoeseguibili e ci consentono anche di modificare i loro script con un semplice editor di testo. Logicamente, tuttavia, per crearli è necessario un software molto specifico.

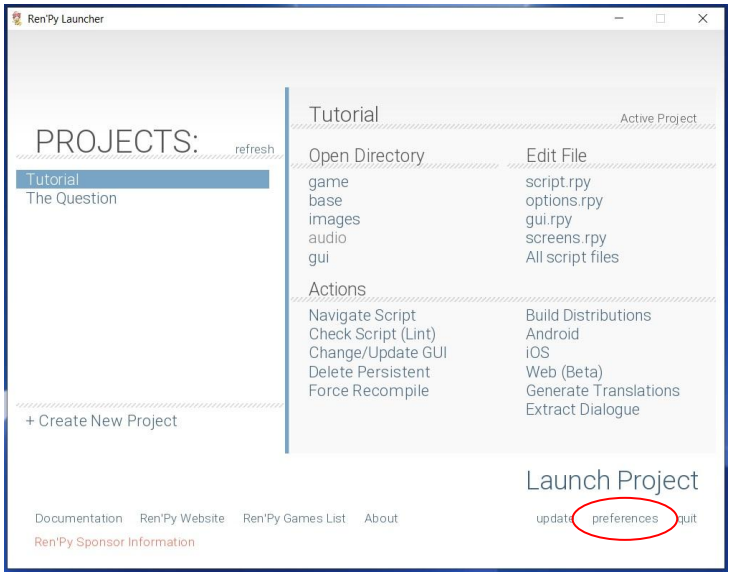
Ren'Py SDK è l'applicazione che ci permette di creare giochi Ren'Py e anche le loro traduzioni. Quindi il primo passo per tradurre qualsiasi gioco Ren'Py è scaricarlo. È gratuito, pulito e completamente affidabile. LINK

Dopo averlo scaricato ed estratto sul tuo computer, apri la sua cartella e vedrai qualcosa del genere:



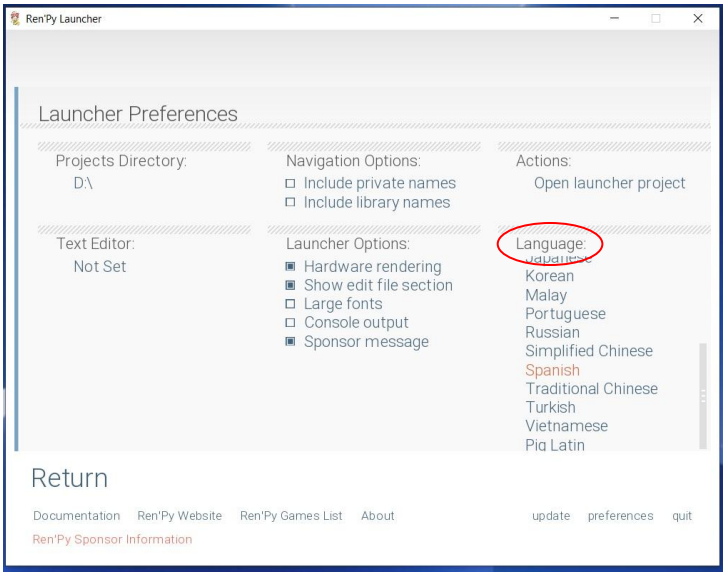
Cioè, una struttura molto simile a quella dei giochi, con diverse sottocartelle e un eseguibile. Tra le sottocartelle abbiamo "the_question" e "tutorial", che sono due tutorial per imparare a usare Ren'Py. Non fa male guardarli, ma non aggiungono molto alle traduzioni.

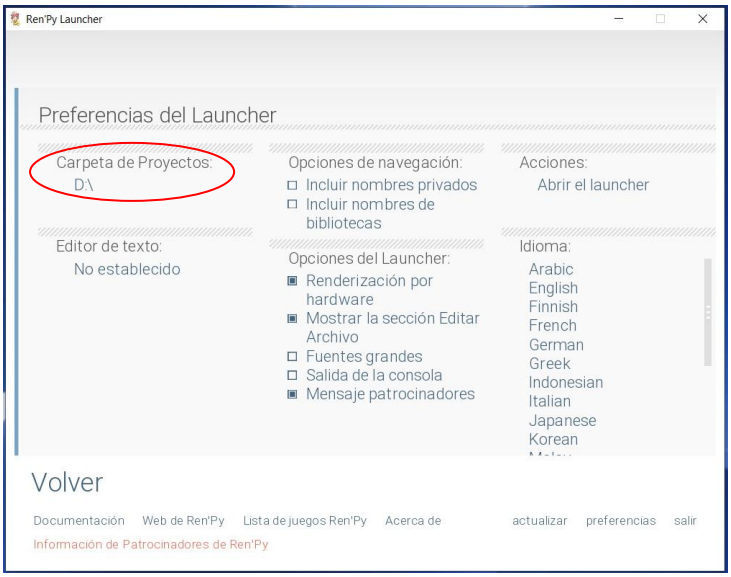
Abbiamo solo bisogno di eseguire l'eseguibile chiamato "renpy" e iniziare a lavorare. Dopo aver impostato alcuni parametri, apparirà la schermata principale, solitamente in inglese. Quindi, prima di tutto, cambiamo la lingua dell'app nella nostra lingua.



Per prima cosa, dobbiamo fare clic su "preferenze" in basso a destra nella finestra.

Sotto l'etichetta "Lingua", sul lato destro di questa schermata "preferenze" vedremo un elenco di lingue disponibili. Ren'Py SDK può essere eseguito in tutte queste lingue, quindi dobbiamo solo cercare quella desiderata. Non appena si fa clic su di esso, la lingua dello schermo cambierà.





Ora abbiamo il software in esecuzione nella nostra lingua (spagnolo, nel mio caso) e non avremo bisogno di cambiarlo mai più. L'opzione cerchiata, chiamata "Cartella dei progetti", è la directory in cui Ren'Py SDK memorizzerà e cercherà i giochi. Qui dobbiamo selezionare la directory in cui si trova la cartella principale del gioco che vogliamo tradurre. Cioè, se hai "What a Legend!" in C / Documents, quindi seleziona C / Documents. Fai clic su "Back" e tutti i giochi in quella cartella appariranno nell'elenco "Projects", oltre ai due tutorial inclusi nel Ren'Py SDK.

| Inizio |

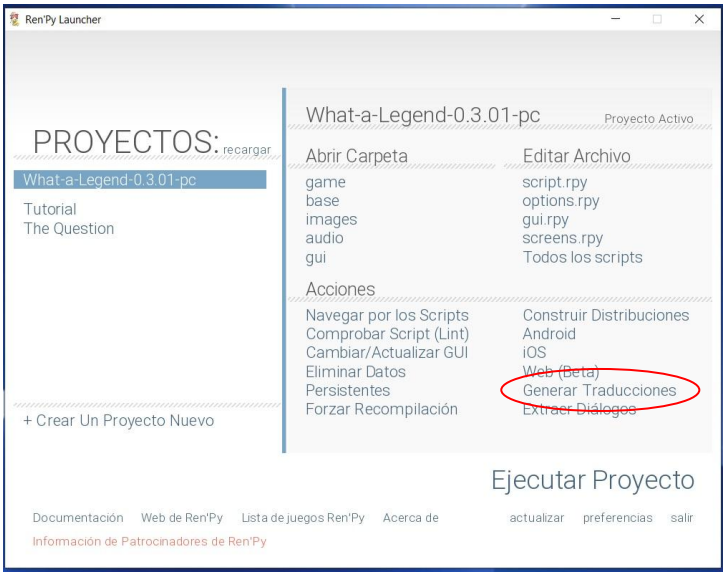
2.- TRADUCIAMO!

O perché lo sviluppatore ci ha reso tutto facile, o perché abbiamo lottato per raggiungerlo, l'importante è che, a questo punto, abbiamo il nostro gioco con i suoi script .rpy completamente esposti e pronti per essere tradotti. Ma, sfortunatamente o no, Ren'Py non traduce nulla, aiuta solo a estrarre i testi traducibili e fa apparire le traduzioni al loro posto. Ren'Py SDK genera gli script che utilizzeremo per creare la traduzione, con tutti i comandi necessari; tuttavia, far sì che estragga completamente tutti i testi traducibili non è sempre facile come fare clic su un pulsante, e talvolta dovremo fare un lavoro di editing approfondito degli script originali. Ma, visto che siamo arrivati a questo punto, faremo clic su quel pulsante che sicuramente stai già guardando con occhi avidi: Genera traduzioni.

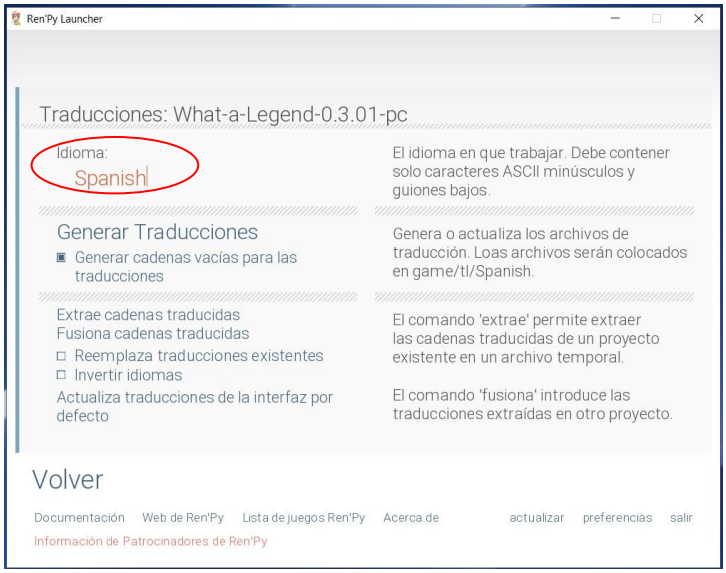
| Inizio |

2.1.- Generare file di traduzione (e imparare il quarto comandamento)

Dopo aver selezionato il gioco che vogliamo tradurre (apparirà evidenziato nella sezione "Progetti"), fare clic sul pulsante "Genera traduzioni" sul lato destro dello schermo.



Nella casella "Lingua" dobbiamo scrivere la lingua in cui vogliamo tradurre il gioco. È solo un'identificazione interna, quindi possiamo scrivere quello che vogliamo (nota che non sono ammessi caratteri speciali, come ñ o vocali accentate). La cosa più importante è ricordare quello che abbiamo scritto qui e tenere sempre presente il quarto comandamento di Ren'Py:



> **Il quarto comandamento di Ren'Py:** Per Ren'Py, una lettera maiuscola non è uguale a una lettera minuscola, mai, in nessuna circostanza.

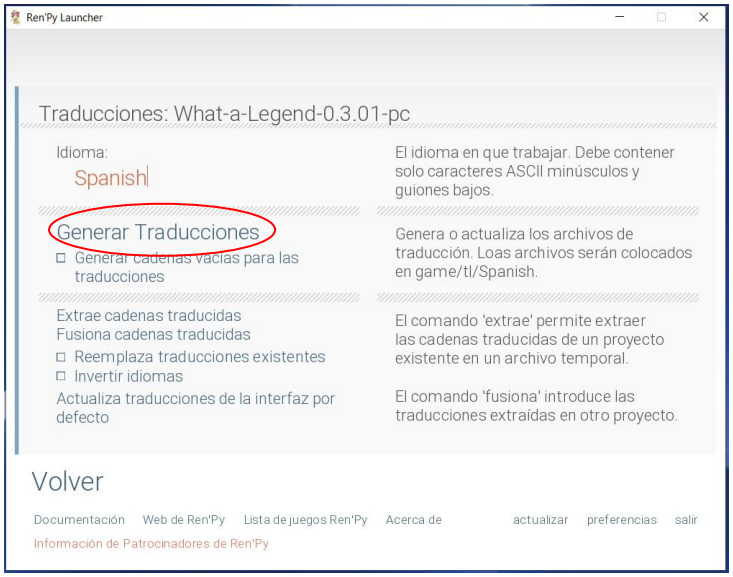
Quindi puoi scrivere spagnolo, spagnolo, spagnolo, francese, francese, francese, nave, giallo o scimmia. Tutto quello che vuoi. Ma, ovviamente, si consiglia di usare una parola comprensibile. E, come ho detto, la cosa più importante è ricordare cosa hai scritto lì e come l'hai scritto. Ad esempio, io uso sempre la parola "spagnolo", con la "S" maiuscola, ed è quello che vedrai d'ora in poi negli esempi forniti.

Poi ci sono diverse opzioni, ma l'unica che ci interessa è quella che dice "Genera stringhe vuote per le traduzioni" (o qualunque cosa sia detta nella tua lingua). Qui, come quasi sempre, è tutta una questione di gusti. Quando traducendo, avremo sempre una linea visibile con il testo originale che ci guidi, ma l'idea è che, se selezioniamo questa opzione, gli script di traduzione verranno generati con poche righe vuote per consentirci di scrivere direttamente la nostra traduzione su di essi. Se non lo selezioniamo, gli script verranno generati con quelle righe scritte nella lingua originale del gioco e dovremo eliminare quelle parole e sostituirle con la loro traduzione. Qui possiamo vedere un confronto affiancato di come la traduzione file verrebbero visualizzati se selezioniamo quella casella (a sinistra) e se non la selezioniamo (a destra).

```
# 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."
```

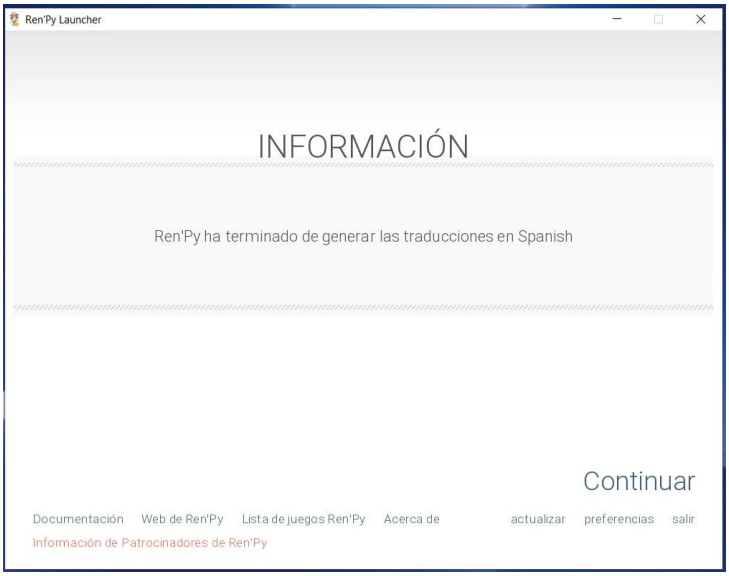
Può sembrare più conveniente (in realtà lo è) generare stringhe vuote, ma, se in qualche modo ci dimentichiamo di tradurre una riga, quando i giocatori raggiungono quella riga nel gioco, non verrà visualizzato assolutamente alcun testo sullo schermo. Altrimenti, il testo verrebbe visualizzato nella lingua originale e questo può esserci utile quando si verifica una traduzione incompleta. Questo è particolarmente importante nei menu, poiché ci permetterà di avere tutte le opzioni attive anche quando non le abbiamo ancora tradotte.



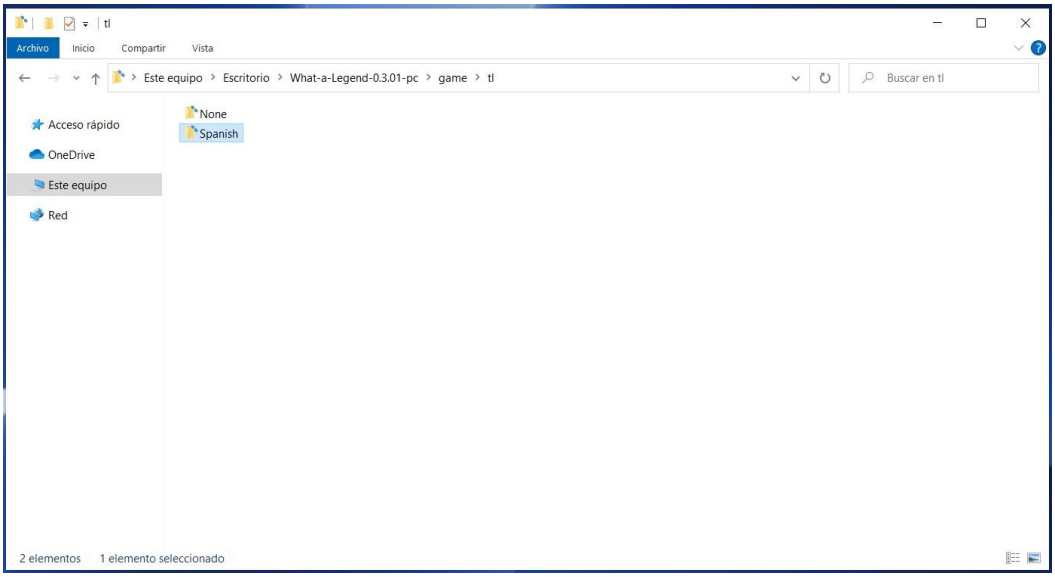
Personalmente, preferisco NON generare stringhe vuote, quindi lascio quella casella deselezionata, ma se intendi utilizzare traduttori automatici potresti essere interessato a generare stringhe vuote.

Adesso non ci resta che cliccare sul pulsante "Genera Traduzioni" (quello cerchiato). Come indica il testo informativo nella colonna di destra, così facendo Ren'Py SDK creerà i file di traduzione all'interno della cartella "game / tl / Spanish" (il nome di detta cartella sarà quello che abbiamo scritto nella "Lingua" scatola).

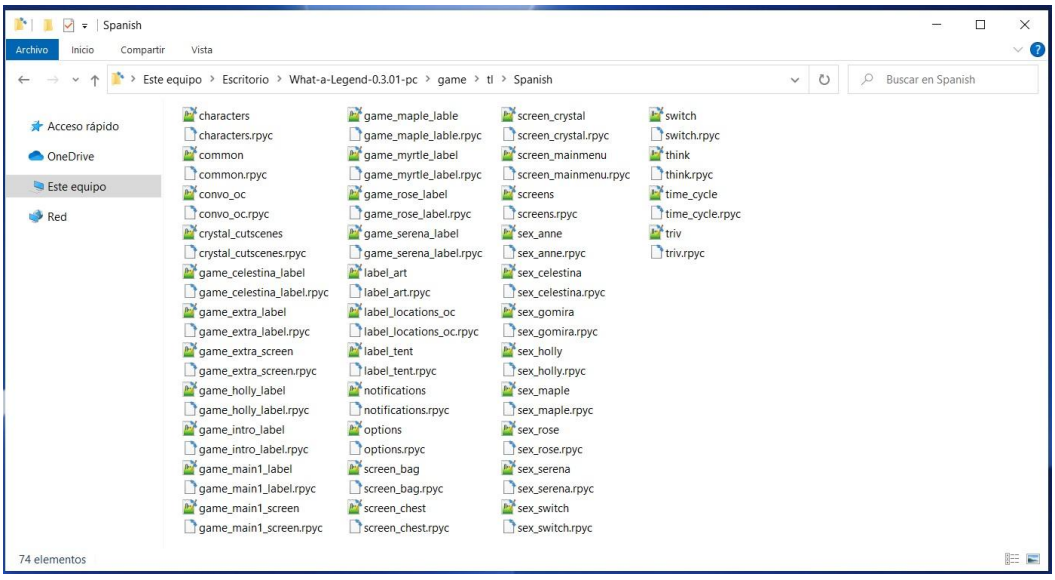
Se non ci sono ulteriori problemi, quando Ren'Py SDK finirà di funzionare la sua magia apparirà questo messaggio. Possiamo cliccare su "Continua", per tornare alla schermata principale, o chiudere direttamente l'app Ren'Py SDK.



Se andiamo nella cartella "game" del nostro gioco, nella sottocartella "tl" vedremo altre due cartelle: una denominata "None" e un'altra con la parola che abbiamo scritto nella casella della lingua dell'SDK di Ren'Py. In questo caso, "spagnolo":



E all'interno della cartella "Spagnolo" troveremo gli script di traduzione. Cosa ha fatto Ren'Py SDK per crearli? Bene, ha esaminato tutti gli script .rpy originali, uno per uno in ordine alfabetico, e, ogni volta che ha trovato del testo traducibile in essi, ha generato un file .rpy (e il suo corrispondente .rpyc) con lo stesso nome di lo script originale, scrivendo in esso le stringhe che devono essere tradotte.



Inoltre, sarà stato generato uno script aggiuntivo: denominato "common.rpy", conterrà i comandi traducibili dei menu Ren'Py che non sono specifici del gioco. Ad esempio, il messaggio di avviso che appare quando chiudiamo il gioco, che ci chiede di confermare la nostra scelta, oi mesi e i giorni della settimana che verranno utilizzati per nominare i nostri giochi salvati. È un file di grandi dimensioni e piuttosto noioso da tradurre, ma c'è un lato positivo: di solito, è comune alla maggior parte dei giochi creati con la stessa versione di Ren'Py SDK, quindi può essere intercambiabile da un gioco all'altro. Quindi possiamo riutilizzare uno script "common.rpy" tradotto che abbiamo già in un gioco precedente. Naturalmente, prima di sostituirlo, dovremmo sempre controllare che le stringhe corrispondano, poiché a volte ci sono piccoli cambiamenti e potremmo rompere qualcosa involontariamente.

A questo punto della traduzione è solo questione di aprire gli script .rpy della cartella “tl / Spanish” con un editor di testo e iniziare a lavorarci sopra. Possiamo cambiare il nome di questi script (e persino unirli in un unico file), poiché Ren'Py cercherà le traduzioni stringa per stringa e non si preoccupa del file in cui sono archiviate, ma la pratica standard è "non toccare qualunque cosa ”: in questo modo è più facile identificare ogni copione tradotto con il suo copione originale.

| Inizio |

2.2.- Le due funzioni di traduzione (e l'ennesimo comandamento)

Non è insolito che, la prima volta che iniziamo a tradurre, scriviamo poche righe nella nostra lingua e poi improvvisamente pensiamo: “Aspetta, dove sono le opzioni per il giocatore? Il gioco non mi ha chiesto qui se volevo fare una cosa o un'altra? " Niente panico: nei file di traduzione, i testi traducibili non appaiono esattamente nello stesso ordine dello script originale. Quando si tratta di estrarre le stringhe traducibili da ogni script, Ren'Py le divide in due categorie: quelle che formano il blocco di dialogo e quelle che fanno riferimento a opzioni di menu, variabili, nomi di caratteri, ecc. Perché? Bene, perché ognuno di questi gruppi utilizzerà una diversa funzione di estrazione e traduzione.

Le stringhe che formano il blocco di dialogo non genereranno alcun problema di estrazione perché Ren'Py SDK le identificherà senza problemi. Ciò che fa con loro, tuttavia, causerà più fastidio e persino alcuni mal di testa durante la traduzione e la creazione del nostro patch di traduzione . Perché Ren'Py le crittografa per risparmiare memoria: utilizzando il metodo di crittografia esadecimale MD5 a 8 byte, ogni stringa viene identificata con un codice alfanumerico che dipende dal etichetta dove quel testo si trova nello script originale e il contenuto della stringa stessa.

Possiamo vederlo meglio con un esempio. Prima di tutto, ricordiamo le prime righe dello script "convo_oc.rpy" del gioco "What a Legend!":

```
# ==================== OLD Capital Base Conversations ==========================================
# Being stopped at the gate of the old capital ================================================
label convo_gate:
    call hide_ui from _call_hide_ui_104
    call silent from _call_silent_42

    scene scene_oc_bridge_gate_talk
    if current_hour == "Night" or current_hour == "Evening":
        show kevin at g_cright:
            xalign 0.6
        show pov at m_left
        with quickfade
        show pov ewide bup mdislike hfshock hbshock
        show kevin hbstop eangry
        kevin "STOP!" with vpunch
        show kevin hbpoint edoubt
        show pov -mdislike hfneu1 hbneu1
        kevin "Did you manage to get a passage permit?"
        show pov mno bdoubt eneub hfhead
        show kevin eneu hbneu
        pov "Umm..."
        show pov eneu bsad msad
        show kevin esad hfspearmove
        kevin "I'm sorry, buddy..."
        show pov mcry eflat bneu hfneu1
        kevin "...but no permit, no passage. That is the law."
```

La prima riga con testo traducibile (la prima " corda ") è la riga 15 della sceneggiatura originale: sotto l'etichetta" convo_gate ", un personaggio identificato come Kevin dice" STOP! "(con vpunch, un effetto che" scuoterà "il giocatore

schermo). Il resto del codice fino a quel punto non contiene testo da visualizzare sullo schermo.

Durante la generazione dello script di traduzione con stringhe vuote, Ren'Py ha trasformato tutte le informazioni su questo:

```
# 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 ""
```

Possiamo vedere che, per ogni stringa, Ren'Py ha generato un blocco di quattro righe. Analizziamo quei blocchi:

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

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

Le righe che iniziano con il simbolo # sono puramente informative e potremmo eliminarle senza conseguenze. Il primo ci dice la posizione di quella riga nello script originale: è la riga 15 del file "convo_oc.rpy", all'interno della cartella "game". L'altra riga che inizia con # è il testo originale da tradurre, che è gentilmente offerto da Ren'Py in modo che sappiamo cosa scrivere nella nostra lingua. E l'ultima riga è dove scriveremo la nostra traduzione (tradurremo solo ciò che è tra virgolette, dovremmo lasciare il resto così com'è). Se avessimo generato la traduzione senza stringhe vuote, in questa riga il testo comparirebbe nuovamente in inglese.

La chiave è nella seconda riga. Questo è ciò che attiva la funzione di traduzione per questa specifica riga del codice originale. Quando riproduciamo una versione tradotta, Ren'Py esegue il gioco in base ai file .rpyc originali dalla cartella "game", ma è stato ordinato di visualizzare i testi tradotti invece delle stringhe originali. Quel comando dice a Ren'Py che, in ogni riga del blocco di dialogo, deve cercare negli script cercando una riga con il comando " tradurre" più la lingua attiva (" Spagnolo", in questo caso, che è quello che abbiamo scritto durante la generazione dei file di traduzione) più il codice di crittografia che corrisponde alla stringa originale. Questo codice di crittografia inizia con il nome dell'etichetta in cui si trova questa riga specifica ("convo_gate") e continua con il risultato dell'applicazione del sistema MD5 al contenuto della riga (apparentemente in quel sistema di crittografia, "STOP!" È uguale a fdd309da) .

Se ci fosse un'altra stringa identica (un altro "STOP!") All'interno della stessa etichetta "convo_gate", Ren'Py dovrebbe assegnarle lo stesso codice di crittografia. Ma questo potrebbe portare a conflitti, quindi aggiungerà un _1 alla fine del codice della seconda riga, se ci fosse ancora un'altra stringa identica verrebbe aggiunto un _2 e così via. Pertanto, i codici di crittografia sono sempre univoci per ciascuna stringa, anche per stringhe identiche all'interno del blocco di dialogo.

Ma cosa succede se andiamo alla sceneggiatura originale e ora scriviamo "Stop!" invece di "STOP!"? Bene, allora la traduzione che abbiamo per "STOP!" smetterà di funzionare, perché il codice MD5 di "Stop!" sarà un altro e Ren'Py non lo troverà negli script di traduzione, quindi verrà visualizzato il testo originale. Ciò causa alcuni fastidi durante la traduzione di nuove versioni di giochi ancora in fase di sviluppo, poiché è comune per i creatori correggere errori di battitura minori e riscrivere alcune frasi qua e là. Dovremo tradurre di nuovo alcune stringhe che avevamo già tradotto prima, perché anche la minima modifica cambierà completamente il loro codice di crittografia e saranno completamente diverse per il motore Ren'Py. Questo accade anche quando è

l'etichetta viene rinominata, anche se il contenuto della stringa non viene modificato.

> **Quinto comandamento di Ren'Py:** Qualsiasi minima modifica apportata alla stringa originale invaliderà la traduzione preesistente di quella riga.

Fortunatamente, se ciò che accade è un semplice riposizionamento della linea (la stringa passa dall'essere sulla linea 7 a essere sulla linea 8, ma il suo contenuto non cambia né cambia la sua etichetta), la traduzione precedente continuerà a funzionare, poiché questo trasferimento non implica alcun cambiamento nel codice di crittografia. La riga #commented che ci dice dove si trova quella stringa nello script originale sarà obsoleta, ma ciò non ha un impatto importante sulla nostra traduzione.

Ma, come ho detto, Ren'Py utilizza due sistemi di traduzione. Oltre alla crittografia basata su codice, per il resto delle stringhe che non appartengono al blocco di dialogo, viene utilizzato un sistema più semplice di traduzione diretta. A quali stringhe ci riferiamo? Ebbene, come ho detto all'inizio di questa sezione, sto parlando delle opzioni che vengono presentate al giocatore durante il gioco, ma anche dei nomi dei personaggi, delle potenziali variabili utilizzate nel gioco che hanno un valore di testo, dei menu di sistema (preferenze , salva e carica il gioco ...) e così via.

Quando Ren'Py esegue un gioco tradotto, tutte quelle stringhe verranno sostituite sullo schermo tramite sostituzione diretta, senza deviazioni o crittografie. Per ottenere ciò, durante la generazione degli script di traduzione, Ren'Py SDK li raggruppa tutti alla fine del documento, sotto il comando " tradurre stringhe spagnole ". Questo è l'inizio della seconda sezione dello script di traduzione "convo_oc" da "What a Legend!":

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

Come puoi vedere, qui non ci sono codici strani. E se il " Tradurre" prima veniva applicato a ciascuna riga, ora vengono tradotti tutti utilizzando un solo comando. Abbiamo ancora una prima riga con il simbolo # per informarci sulla posizione della stringa traducibile nello script originale, ma poi si dimentica dei codici di crittografia e va direttamente a mostrare il testo originale con " vecchio" comando; e poi abbiamo un " nuovo" comando prima del testo che dovrebbe apparire quando il gioco sta eseguendo una traduzione in " spagnolo "Lingua (quella abbiamo scritto nell'SDK Ren'Py quando stavamo generando gli script di traduzione).

A differenza di prima, ora nessuna stringa ripetuta apparirà non solo in questo script, ma nell'intero set di script di traduzione. Quando Ren'Py SDK passa attraverso gli script originali per generare i file di traduzione, estrarrà ogni stringa solo la prima volta che viene trovata e trascurerà i duplicati che potrebbero potenzialmente esistere. Quindi, durante la riproduzione, ogni volta che appare quella stringa sarà sostituita dalla traduzione che abbiamo introdotto quella volta. Questo ci evita di ripetere il lavoro, ma diventa un inconveniente quando ci imbattiamo in stringhe identiche che possono avere significati diversi a seconda del contesto. Ad esempio, pensa a una stringa che viene ripetuta più volte durante il gioco ma dice solo "Giusto": a volte potrebbe significare "Corretto" ea volte può riferirsi al lato destro di qualcosa, o indicando una direzione, e forse nella nostra lingua usiamo una parola diversa per ogni concetto. Ma Ren'Py SDK estrarrebbe solo la prima stringa "Right", quindi potremmo tradurla solo una volta e quella traduzione apparirà sempre, quindi a volte il testo tradotto non avrebbe alcun senso. Vedremo come risolvere questo problema dopo .

Usando l'esempio sopra, una volta che scriviamo in questo script la traduzione della stringa "Permit", quella traduzione apparirà in tutto il gioco ogni volta che c'è una stringa "Permit" fuori dal blocco di dialogo. Ma attenzione: secondo il quarto comandamento di Ren'Py, se ci fosse un'altra stringa con il testo "permesso", dovremmo tradurla, perché Ren'Py distingue perfettamente tra maiuscole e minuscole.

Tornando all'argomento, perché Ren'Py utilizza due diverse funzioni di traduzione? Bene, per problemi di agilità e utilizzo efficiente della memoria. Tecnicamente, questo sistema di traduzione "sostituzione diretta" potrebbe essere applicato a tutte le stringhe del gioco, ma di solito il blocco di dialogo è abbastanza grande e consiste in frasi molto più lunghe che non vengono quasi mai ripetute. Quindi, quando il programma deve mostrare una traduzione, ci vuole meno tempo per trovare e sostituire migliaia di codici crittografati rispetto a migliaia di righe intere che di solito sono più lunghe di quei codici. Tuttavia, queste altre stringhe di menu sono generalmente più brevi e molto meno numerose, ed è possibile che vengano ripetute più frequentemente negli script, quindi Ren'Py può permettersi di fare una ricerca specifica del contenuto esatto di queste stringhe senza ritardi evidenti durante gameplay.

Comunque, queste sono solo alcune nozioni perfettamente dimenticabili. La cosa più importante è sapere che, per Ren'Py SDK, ci sono due tipi di "stringhe" e ognuno di questi tipi utilizza un sistema di traduzione diverso, ed è per questo che appariranno separatamente negli script di traduzione: prima noi ' Troverò tutte le stringhe appartenenti al blocco di dialogo, e poi tutte quelle corrispondenti a menu, opzioni e variabili.

| Inizio |

2.3.- Aiutare e / o sostituire l'estrattore

Il motivo per spiegare al punto precedente i due tipi di funzioni di traduzione di Ren'Py sono che, sfortunatamente, Ren'Py SDK non è sempre in grado di rilevare assolutamente tutte le stringhe che dovremmo tradurre: sì, estrarrà tutte quelle che integrano il blocco di dialogo (quelle che verranno tradotte grazie al codice di crittografia), ma potrebbe non essere in grado di identificare tutte le stringhe che vengono tradotti dal metodo di sostituzione diretta (sebbene estrarrà le opzioni che consentono ai giocatori di prendere decisioni durante il gioco).

Ad esempio, per rendere Ren'Py SDK in grado di estrarre automaticamente i nomi dei personaggi, lo sviluppatore del gioco avrebbe dovuto definire quei caratteri in un modo molto specifico che, a causa dell'ignoranza, molti sviluppatori non usano. Ma se non lo fanno, possiamo farlo.

Come i creatori di "What a Legend!" hanno fatto i compiti, rendendo la vita molto più facile a noi traduttori, per questi esempi useremo l'altro gioco di cui avevamo già parlato File .rpa e UnRen extractor . Quindi vediamo come lo sviluppatore "WVM" definisce uno dei personaggi coinvolti nella storia. In questo caso, nel file "script.rpy" ha creato un "sé interiore MC" per indicare ai giocatori che quello che leggiamo nella finestra di dialogo è un pensiero del nostro personaggio:

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

Questa è una tipica riga del codice Ren'Py. Inizia con un comando (definire) che indica cosa fa questa specifica riga, in questo caso definendo una variabile. A quella variabile viene assegnato un nome (mcm) che verrà utilizzato nel resto dello script per identificarlo, ed è ordinato di comportarsi come una variabile di tipo carattere (Personaggio). Ora Ren'Py lo sa, ogni volta che trova le lettere mcm prima di una stringa, deve mostrare un nome sopra la casella di testo in modo che il giocatore sappia quale personaggio sta parlando. E cosa verrà mostrato lì? Ciò che appare dopo tra parentesi: il nome del personaggio (" I tuoi pensieri "), In un colore specifico, con una linea che circonda le lettere per evidenziarle.

Ovviamente, "I tuoi pensieri" è una stringa che dovremmo tradurre, ma, se generiamo gli script di traduzione con Ren'Py SDK, non apparirebbe da nessuna parte. Ren'Py Mysteries. Cosa possiamo fare? Bene, ci sono tre opzioni. La prima opzione, che chiaramente scartiamo, è semplicemente accettarla e lasciare che la stringa sia sempre visualizzata in inglese. La seconda opzione è far capire a Ren'Py SDK che questo testo citato è una stringa che vogliamo tradurre e non solo un codice interno come quello che indica il colore del testo. Guarda:

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

Abbiamo modificato lo script originale inserendo la stringa "I tuoi pensieri" tra parentesi e abbiamo digitato un trattino basso _ prima delle parentesi. Questa combinazione di simboli _ () consentirà a Ren'Py SDK di identificare il testo tra parentesi come una stringa traducibile.

E se ora generiamo gli script di traduzione, lo troveremmo nella sezione della traduzione diretta:

```
translate Spanish strings:

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

Questo è il risultato dopo averlo tradotto in spagnolo, ovviamente. Ma la cosa importante è che, se non avessimo modificato lo script originale per sostituirlo " I tuoi pensieri "Per _ (" I tuoi pensieri"), questa stringa non apparirà negli script di traduzione.

Ciò non significa che non avremmo mai potuto tradurlo: poiché è una traduzione diretta, potremmo sempre scriverlo nel nostro script di traduzione senza l'aiuto di Ren'Py SDK. Questa è l'ultima delle tre opzioni che ho menzionato prima: invece di modificare gli script originali, scriviamo negli script di traduzione le stringhe traducibili che Ren'Py SDK non ha rilevato. Per fare ciò, andremmo alla sezione dello script di traduzione dove appaiono le traduzioni dirette (quella che inizia con la funzione traduci stringhe spagnole :) e scriveremmo una funzione di traduzione con lo stesso formato che abbiamo visto sopra: rispettando sempre il rientro di 4 spazi, così come le virgolette come dicono i primi due comandamenti di Ren'Py, in una riga scriveremo il comando vecchio

seguito dalla stringa da tradurre (rispettando scrupolosamente la sua scrittura, come dicono il quarto e quinto comandamento di Ren'Py), e poi scriveremmo sotto il nuovo comando seguito dalla traduzione. La riga che inizia con un simbolo # non sarebbe necessaria in quanto è puramente informativa.

Personalmente, sono abituato a rivedere e modificare prima i file originali in modo che Ren'Py SDK possa quindi eseguire un lavoro di estrazione completo. Certo, a volte mi manca la stringa occasionale e devo rigenerare più volte gli script di traduzione, il che non è più un problema che dover aprire nuovamente l'app Ren'Py SDK e fare clic su genera traduzioni. Altri traduttori preferiscono modificare gli script originali il meno possibile, quindi generano questi script di traduzione all'inizio e poi li modificano manualmente per includere tutto ciò che Ren'Py SDK non ha estratto. È solo una questione di gusti personali e abitudini lavorative.

Sia che tu voglia aiutare in anticipo l'SDK Ren'Py o preferisci scrivere negli script di traduzione le stringhe mancanti, dovrai esaminare gli script originali e cercare questi tipi di comandi:

- Personaggio("..."), utilizzato per definire i caratteri, come abbiamo visto nell'esempio
- testo "...", utilizzato per visualizzare il testo su uno schermo specifico, al di fuori della casella di testo
- pulsante di testo "...", utilizzato per i testi che eseguono un'azione quando si fa clic su di essi
- descrizione comando '...', utilizzato per visualizzare un messaggio a comparsa quando si passa con il mouse su un punto
- renpy.input ("..."), utilizzato per consentire la digitazione nel gioco (per consentire ai giocatori di nominare i personaggi, di solito)
- $ renpy.notify ('...', 'unlock'), utilizzato per visualizzare i messaggi dopo aver completato un'azione

Inoltre, abbiamo variabili di testo. Qui possiamo trovare diverse possibilità, ma nessuna di esse verrà automaticamente estratta da Ren'Py SDK:

- predefinito *nome_variabile = "..."*
- definire *nome_variabile = "..."*
- $ *nome_variabile = "..."*

Quindi è una questione di racchiudere tutte quelle virgolette con il simbolo _ () e generare gli script di traduzione, o copiare il suo contenuto (citazioni incluse) direttamente in uno script di traduzione (non importa quale) con il vecchio comando e il nuovo comando di seguito con la sua traduzione. Nota che devi copiare tutto ciò che appare tra virgolette, non solo il testo da tradurre, poiché a volte ci sono tag all'interno dei simboli {} che vengono utilizzati per visualizzare il testo in **grassetto,** *corsivo,* con un altro carattere o una dimensione diversa, ecc. Anche questi tag fanno parte della stringa che Ren'Py cercherà e sostituirà, quindi devono apparire dietro vecchio

comando per aiutare Ren'Py a identificare correttamente quella stringa esatta.

Infine, solo un'osservazione in più: il simbolo _ () **viene utilizzato solo per estrarre le stringhe** tra parentesi. Una volta estratto in uno script di traduzione, possiamo eliminare quel simbolo e la traduzione funzionerà correttamente, proprio come se scrivessimo manualmente quelle stringhe nello script di traduzione. Perciò, durante la traduzione di nuove versioni dello stesso gioco, non sarebbe necessario modificare nuovamente gli script originali in quanto possiamo riutilizzare i vecchi script di traduzione in cui quelle stringhe appaiono già tradotte. Inoltre, non è necessario introdurre lo script modificato in il patch di traduzione , poiché i giocatori non hanno bisogno del simbolo _ (): una volta che la stringa traducibile appare nello script di traduzione, il testo verrà visualizzato tradotto sullo schermo. Ammetto di averlo imparato relativamente di recente e, se l'avessi saputo prima, mi sarei risparmiato poche ore di lavoro inutile.

| Inizio |

2.4.- L'opzione di cambio lingua

L'ultima cosa che dovremmo fare prima di iniziare a tradurre, o forse la prima, è pensare a come attiveremo la traduzione in-game, una volta creata. Vogliamo che il gioco inizi sempre nella nostra lingua? Aggiungiamo l'opzione per cambiare le lingue nel menu principale o nel menu delle opzioni? Preferiamo lasciare che i giocatori decidano in quale lingua vogliono giocare ogni volta che iniziano il gioco? E come facciamo tutto questo, con una domanda di testo semplice o creando uno schermo con bandiere e altri elementi visivi? Come sempre, dipende dai gusti personali del traduttore, dal tempo a disposizione e dalle capacità di codifica, e ovviamente anche dall'impostazione del gioco originale.

Il modo più semplice è fare in modo che il gioco inizi sempre nella lingua desiderata. Per fare questo, dobbiamo solo aprire lo script di qualsiasi gioco (preferibilmente lo script "gui.rpy", poiché gran parte della configurazione del gioco è definita lì, ma in realtà non ha importanza e possiamo persino crearne uno nuovo) e scrivere questa riga di codice, senza rientro:

```
define config.language = "Spagnolo"
```

Dove " Spagnolo" è, ricordiamolo, la parola che ho inserito in Ren'Py SDK quando generare la traduzione script ; e quindi il riferimento utilizzato dal " tradurre " comandi , quindi usi il termine che hai scelto. Se non facciamo nient'altro, il gioco inizierà sempre nella nostra lingua e i giocatori non avranno alcuna opzione di gioco per tornare alla lingua originale: anche dopo aver disabilitato questa riga di codice (cancellandola o scrivendo un # simbolo all'inizio della riga) il gioco non si avviava più nella lingua originale, poiché per impostazione predefinita i giochi Ren'Py iniziano sempre nell'ultima lingua in cui sono stati giocati. Una nuova config.language

dovrebbe essere stabilito con la lingua originale, che per Ren'Py è sempre denominata Nessuna.

La migliore pratica, in ogni caso, è quella di aggiungere sempre un'opzione di cambio lingua all'interno del menu Preferenze. Supponendo che il gioco utilizzi la schermata delle preferenze che Ren'Py incorpora per impostazione predefinita, dovresti andare allo script originale "screens.rpy", cercare la sezione del menu "Preferenze" e farlo sembrare così:

```
718 init -501 screen preferences():
719     tag menu
720
721
722     use game_menu(_("Preferences"), scroll="viewport"):
723
724         vbox:
725
726             hbox:
727                 box_wrap True
728
729                 if renpy.variant("pc"):
730
731                     vbox:
732                         style_prefix "radio"
733                         label _("Display")
734                         textbutton _("Window") action Preference("display", "window")
735                         textbutton _("Fullscreen") action Preference("display", "fullscreen")
736
737                 vbox:
738                     style_prefix "radio"
739                     label _("Rollback Side")
740                     textbutton _("Disable") action Preference("rollback side", "disable")
741                     textbutton _("Left") action Preference("rollback side", "left")
742                     textbutton _("Right") action Preference("rollback side", "right")
743
744                 vbox:
745                     style_prefix "check"
746                     label _("Skip")
747                     textbutton _("Unseen Text") action Preference("skip", "toggle")
748                     textbutton _("After Choices") action Preference("after choices", "toggle")
749                     textbutton _("Transitions") action InvertSelected(Preference("transitions", "toggle"))
750
751                 vbox:
752                     style_prefix "pref"
753                     label _("Language")
754                     textbutton _("English") action Language(None)
755                     textbutton _("Español") action Language("Spanish")
756
```

Il codice esatto da scrivere sarebbe questo (ricordati sempre di contrassegnare i rientri con la barra spaziatrice):

```
vbox:
    prefisso di stile "pref")
    label _ ("Lingua")
    textbutton _ ("English") action Lingua (Nessuno) textbutton _ ("Español") action Lingua
    ("Spanish")
```

Il _(" Inglese") string presuppone che la lingua originale del gioco sia l'inglese; dovremmo scrivere quella giusta ma lasciando il resto della riga così com'è, poiché Ren'Py considererà sempre quella lingua originale come la Nessuna linguaggio. E, ovviamente, _ (" Español ") dovrebbe essere sostituito dal nome della tua lingua nella tua lingua, mentre il Action Language ("") dovrebbe contenere la parola che hai usato per generare gli script di traduzione.

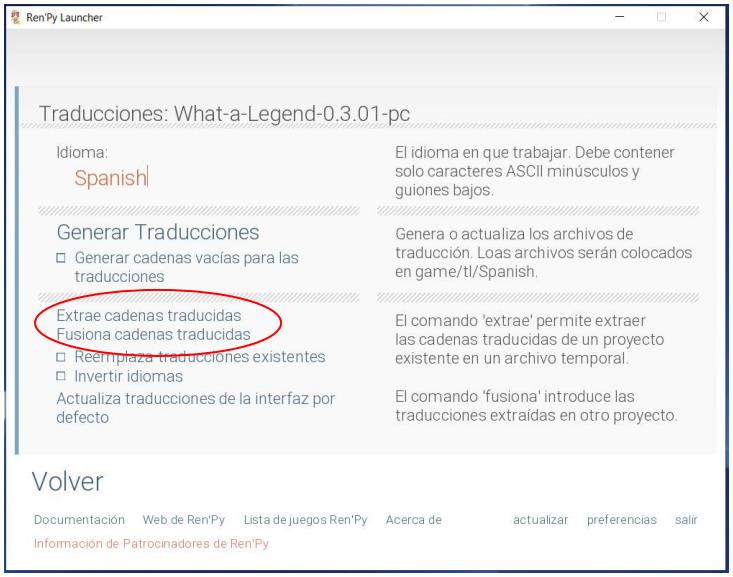
Potresti aver notato che ci sono diverse stringhe all'interno del simbolo _ (), quindi Ren'Py SDK le estrarrà. Personalmente, traduco solo il titolo, "Lingua", e lascio gli altri non tradotti. Perché? Bene, perché in questo modo i nomi delle lingue volere appaiono sempre scritti nella loro lingua, indipendentemente dalla lingua in cui stiamo giocando. Ad esempio, se per caso un giocatore che non parla la nostra lingua trova la nostra traduzione e esegue il gioco nella nostra lingua, cliccando sul menu delle preferenze vedranno rapidamente la parola "English" scritta in inglese e capiranno facilmente che possono cambiare lingua lì.

Per quanto riguarda le altre opzioni, la verità è che ce ne sono tanti quanti sono i giochi e i traduttori, quindi non li descriverò tutti in dettaglio. Dai un'occhiata a come è codificato il menu delle preferenze del gioco (se è personalizzato), fai un po 'di reverse engineering, cerca nei forum online e usa i tutorial Ren'Py per creare schermate, schermate, mappe immagine, pulsanti ... C'è un intero mondo di possibilità, complesse e visivamente attraenti come desideri.

| Inizio |

2.6.- Tradurre gli aggiornamenti

In questi tempi di Patreon è molto comune che i giochi non vengano rilasciati come progetti completati, ma in formato episodico, e siamo costretti ad aggiornare le nostre traduzioni man mano che vengono rilasciate nuove versioni. La procedura non ha complicazioni importanti ... a meno che non vogliamo farlo nel modo suggerito da Ren'Py SDK. L'app ufficiale Ren'Py ci offre un'opzione per estrarre le traduzioni esistenti della vecchia versione di un gioco e inserirle in quella nuova, aggiornando gli script di traduzione con il nuovo contenuto. Bello, eh? Beh, non così tanto. Vediamo.



Per farlo in modo “ufficiale”, avremmo lanciato Ren'Py SDK, che selezioneremmo nella cartella Progetti **la versione precedente** del gioco che vogliamo tradurre (quindi, la vecchia versione l'abbiamo già tradotta) e, nella schermata Genera traduzioni, faremo clic su " **Estrai traduzioni di stringhe ”.**

Una volta terminato il processo, torneremo alla schermata iniziale dell'SDK di Ren'Py, selezioneremo il gioco **nuova versione (** quello che vogliamo tradurre ora) e, in questa schermata Genera traduzioni, faremo clic su
"Merge Strings Translations".

Teoricamente, alla fine del processo avremmo, all'interno della cartella "game / tl / Spanish" della nuova versione del gioco, la nostra precedente traduzione mescolata con le nuove stringhe traducibili di questa nuova versione. Il problema è che questa procedura ufficiale non funziona come dovrebbe e le traduzioni semplicemente non si fondono: se sono originali

non è cambiato affatto da una versione all'altra, il suo script di traduzione verrà trasferito dalla vecchia versione a quella nuova, ma se lo script originale della nuova versione non corrisponde al vecchio, Ren'Py SDK genererà un copione di traduzione completamente nuovo senza alcuna traccia della traduzione preesistente (anche per quelle stringhe che rimangono invariate), costringendoci a ripetere il lavoro di traduzione.

Quindi il modo più efficace è scaricare la nuova versione del gioco, copiare nella cartella "game" la cartella "game / tl / Spanish" che abbiamo nella versione precedente (cioè i nostri vecchi script di traduzione) e generare i file di traduzione per la nuova versione come se fosse un gioco completamente nuovo. Se scegliamo la STESSA lingua della versione precedente (ovvero, se scriviamo nella casella della lingua di Ren'Py SDK la stessa parola che abbiamo usato prima, quindi "spagnolo", nel mio caso), NON controlliamo il " Sostituisci le traduzioni esistenti ", Ren'Py SDK aggiornerà semplicemente gli script di traduzione esistenti con le stringhe per le quali non riesce a trovare una traduzione valida. Alla fine di ogni copione di traduzione, verranno aggiunte sia le nuove stringhe traducibili del gioco (il nuovo contenuto) che quelle che sono state modificate dall'ultima versione tradotta, divise nuovamente in due blocchi (il primo per le stringhe dei blocchi di dialogo e il secondo uno per opzioni e menu). Ordinare gli script di traduzione in base alla data dell'ultima modifica ci consentirà rapidamente di vedere quali hanno nuovi contenuti da tradurre.

Questa stessa procedura viene utilizzata per aggiornare i nostri script di traduzione dopo aver modificato quelli originali per includere il simbolo _ () dove necessario, nel caso in cui Ren'Py SDK all'inizio non ha estratto tutte le stringhe traducibili. Cioè, modificheremmo gli script originali e genereremmo una nuova traduzione **senza sostituire quelli esistenti,** quindi Ren'Py SDK includerà, alla fine dello script di traduzione, un nuovo elenco con tutte le stringhe che non erano state rilevate in precedenza.

| Inizio |

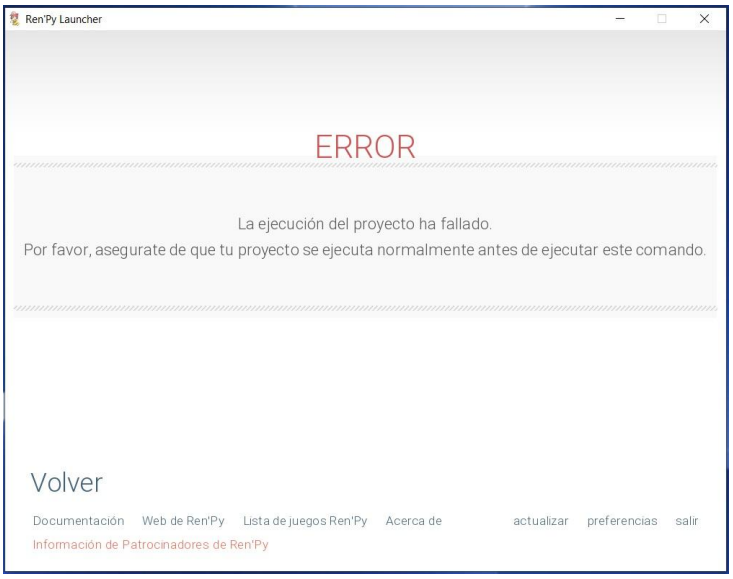
3.- PERCHÉ NON POSSO FARLO FUNZIONARE

Dopo averti detto tutto questo, se sei riuscito a creare la tua traduzione e tutto funziona a dovere, allora complimenti per essere un allievo così attento, e anche per aver scelto per le tue pratiche un gioco abbastanza semplice. Perché la cosa più normale che accada è che, durante la riproduzione della versione tradotta, di tanto in tanto alcuni testi appariranno ancora non tradotti, anche se potresti averli tradotti negli script di traduzione. Successivamente, cercherò di spiegare alcuni degli incidenti più comuni.

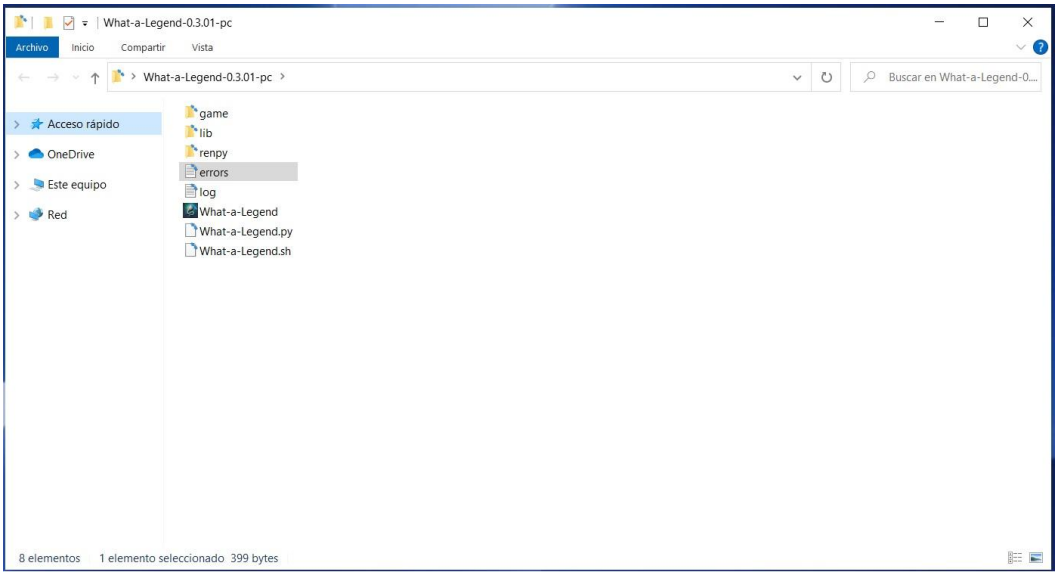
| Inizio |

3.1.- Rilevamento di bug ed errori

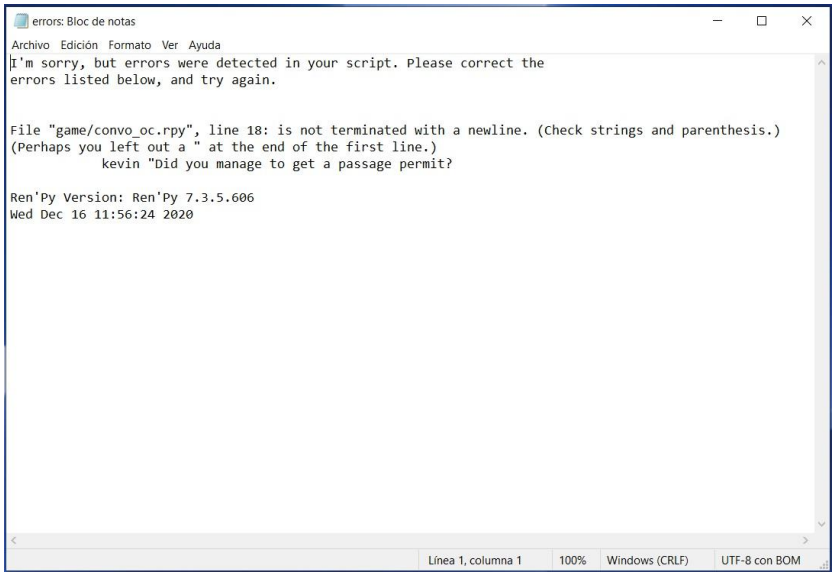
Per prima cosa, dovremmo familiarizzare con lo schermo che Ren'Py usa per avvisarci quando qualcosa non va. In un mondo ideale, i giochi verrebbero rilasciati senza bug e senza errori critici, quindi il primo screenshot fatale che un traduttore potrebbe incontrare dovrebbe essere quello che ci avverte che Ren'Py SDK non può generare gli script di traduzione. Il che può solo significare che abbiamo commesso degli errori durante la modifica degli script .rpy originali. Qui abbiamo la versione spagnola di quella schermata:



Per vedere cosa è andato storto esattamente, dovremmo controllare la cartella principale del gioco. Lì, accanto al file .exe del gioco, appariranno più file .txt.



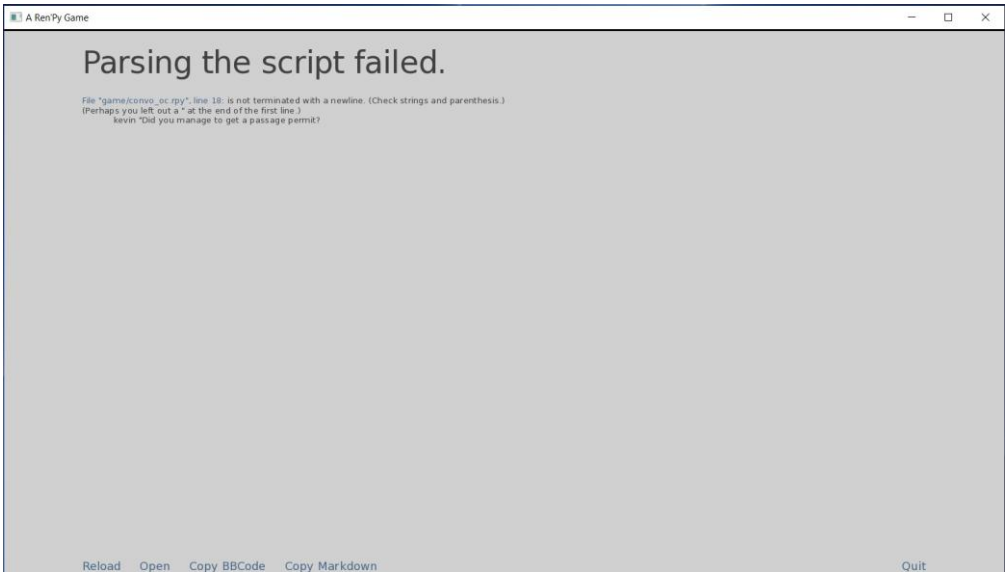
Dovremmo concentrarci su quello denominato "errors.txt", che può essere aperto con qualsiasi editor di testo. Lì vedremo il messaggio di errore che mostra la riga o le righe di codice che hanno causato il problema.



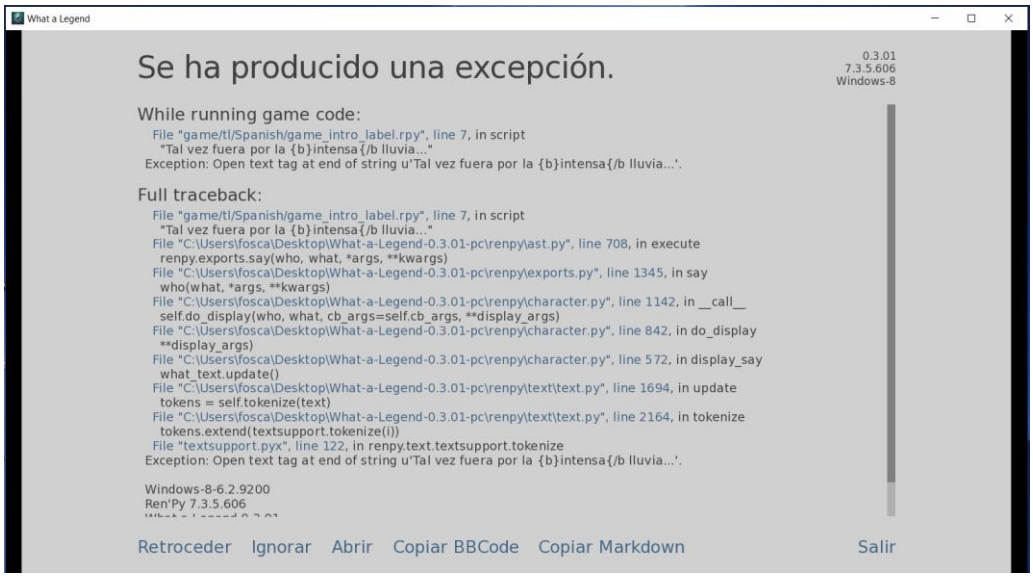
In questo caso, è qualcosa di normale come dimenticare di chiudere le virgolette (sulla riga 18 dello script "convo_oc"), ma potrebbe essere stato un rientro contrassegnato con Tab, o un tag non chiuso, o qualsiasi altra cosa. Dobbiamo solo aprire quel file, correggerlo, salvare le modifiche e tornare a Ren'Py SDK per provare di nuovo quello che stavamo facendo.

Se vedi un messaggio più complesso, o se viene visualizzato anche prima di modificare qualcosa, il problema di solito sorge perché hai utilizzato una versione obsoleta di UnRen o Ren'Py SDK.

Questo messaggio sarebbe stato visualizzato anche se avessimo provato a eseguire il gioco facendo clic sul suo eseguibile. Quindi lo screenshot sarebbe qualcosa del genere, con le stesse informazioni del file "errors.txt":



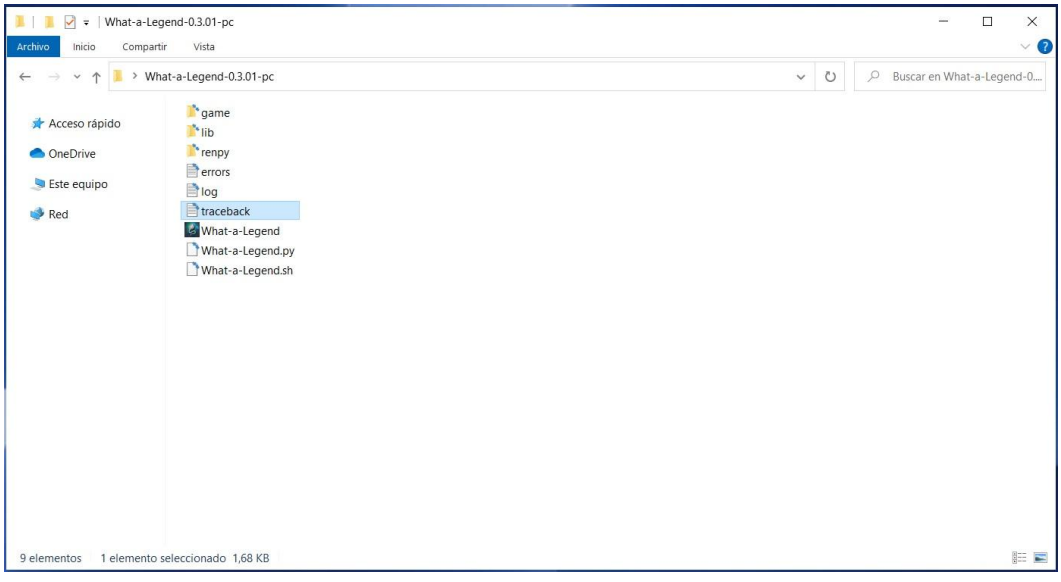
A volte, l'errore non è critico nel senso che ci consente di avviare il gioco e generare i file di traduzione, ma genera un'eccezione durante il gioco. Generalmente, questi errori sono dovuti a un'etichetta scritta in modo errato oa un problema con le variabili del gioco. È allora che, durante il nostro gioco, compare all'improvviso questa schermata grigia con tutte quelle linee:



In questa schermata ci viene data la possibilità di provare a ignorare l'eccezione e riprendere a giocare (con un'alta probabilità di ottenere ulteriori schermate di avviso e arresti anomali del gioco). Possiamo anche tornare a un punto precedente per salvare il nostro gioco prima di provare a correggere ciò che non va. Inoltre, possiamo copiare questo messaggio con un formato BBCode (che ci permetterà di inserirlo nei forum online) o copiarlo come Markdown (che poi potremmo allegare a un messaggio di Discord) nel caso volessimo chiedere aiuto .

Si tratta però quasi sempre di errori minori che, anche se non originati dal nostro lavoro, sono relativamente facili da correggere. Per risolverli, dobbiamo sempre prestare attenzione alla prima riga del messaggio, poiché indica la posizione della riga di codice che ha causato il problema. In questo caso, nello script di traduzione "game_intro_label.rpy" (se guardi più da vicino, si dice che sia all'interno della cartella "game / tl / Spanish", ed è per questo che so che questo è lo script di traduzione e non quello originale) , riga 7, è presente un tag non chiuso (un tag è stato aperto con il comando { b} per **grassetto** una parola, ma non era chiusa correttamente). E nell'ultima riga appena prima della sezione info (dove possiamo trovare il Sistema Operativo del player, la versione del software e la data e l'ora dell'eccezione), compare nuovamente la causa dell'eccezione, anche se senza alcun riferimento allo script in cui si è verificata. Solo guardando queste due righe, saremo in grado di individuare e correggere l'errore.

Ora, dando un'altra occhiata alla cartella principale del gioco, vedremmo che è stato generato un file "traceback.txt". Questo file contiene le stesse informazioni che abbiamo visto nella schermata delle eccezioni del gioco. Sia questo file "traceback.txt" che il file "errors.txt" che abbiamo visto prima vengono rigenerati ogni volta che si verifica un'eccezione, cancellando il suo contenuto preesistente per mostrare solo il problema più recente.



In tutti i casi, si tratta solo di individuare e correggere l'errore, salvare gli script e riprovare quello che stavamo facendo quando è apparsa l'eccezione, sperando che fosse l'unico bug. Buona fortuna.

| Inizio |

3.2.- Righe con troppo testo

In generale, le parole e le frasi scritte in inglese sono generalmente più brevi rispetto alla maggior parte delle lingue. Ciò può far sì che alcune traduzioni superino i limiti della casella di testo che vediamo sullo schermo o che ostruiscano visivamente altri elementi nell'interfaccia dell'utente. Il gioco diventa difficile o sgradevole da leggere e talvolta anche la funzionalità di alcuni pulsanti viene alterata. C'è una soluzione tripla e abbastanza semplice a questo.

Per cominciare, possiamo sempre provare a riscrivere la nostra linea per dire lo stesso con meno caratteri. Se il risultato non ci convince e stiamo traducendo un file stringa del blocco di dialogo , possiamo sfruttare l'enorme libertà che Ren'Py ci offre per scrivere il testo tradotto. Perché, sebbene Ren'Py SDK ci offra solo una riga per tradurre la stringa originale, possiamo dividerla in due o più righe, purché rispettiamo i comandamenti di Ren'Py riguardo a virgolette e tabulazioni. Guarda l'esempio:

```
# game / script.rpy: 10
traduci in spagnolo label_start_XXXXXXX
    # mc "Ciao. Mi chiamo John." mc "Hola."

    mc "Me llamo John."
```

Pertanto, sebbene nello script originale il testo venga visualizzato in un solo messaggio, nella traduzione spagnola sarà diviso in due messaggi, senza ulteriori conseguenze. In realtà, possiamo introdurre qualsiasi variazione a cui possiamo pensare, come aggiungere funzioni, modificare variabili, ecc., Che verranno applicate solo quando si gioca nella nostra lingua. Questo perché, sebbene la funzione di traduzione sia intesa a sostituire una stringa di testo con un'altra stringa di testo in una lingua diversa, ciò che in realtà fa è sostituire un'intera riga di codice con ciò che scriviamo nello script di traduzione, quindi nulla può davvero impedirci dalla sostituzione di quella riga di codice (che sembra essere solo un testo detto da un carattere) con un blocco di codice completamente diverso.

La terza opzione, un po 'più complessa, è modificare lo script originale "gui.rpy", dove tra molte altre cose gli sviluppatori del gioco possono definire la dimensione del carattere e la larghezza della casella di testo. Per la dimensione del carattere, dovremmo cercare " definire gui.text_size " comando e ridurre il numero che appare accanto ad esso. E per aumentare la larghezza della casella di testo, dovremmo aumentare il numero che appare accanto a " definisci gui.dialogue_width " comando. Queste soluzioni ci consentiranno di scrivere più caratteri in ciascuna stringa tradotta e di solito funzionano bene in tutti i giochi, sebbene possano rompere l'estetica del gioco e potresti incontrare alcune difficoltà tecniche a seconda del livello di personalizzazione dell'interfaccia del gioco. Ma ottenere un risultato soddisfacente è solo questione di fare qualche ricerca e testare le modifiche.

| Inizio |

3.3.- Caratteri che non consentono caratteri speciali

Leggermente correlato a il precedente , nel senso che una delle soluzioni è modificare il file "gui.rpy", abbiamo il problema di quei giochi che usano un font di testo che non contiene alcuni caratteri speciali come vocali accentate o altri caratteri che sono abbastanza specifico per alcune lingue. Pertanto, quando si esegue la versione tradotta, le parole appariranno sullo schermo senza quelle lettere o con un simbolo dispari.

La soluzione più rapida è aprire il file "gui.rpy" originale e cercare il " definire gui.text_font "
comando per sostituire il carattere utilizzato con DejaVuSans, che non richiede alcuna azione aggiuntiva poiché è integrato in Ren'Py. Ecco come dovrebbe apparire il tuo codice:

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

Se il font che non supporta i caratteri speciali della nostra lingua è quello utilizzato per i nomi dei personaggi del gioco, o qualche altro font specifico utilizzato nei menu, puoi individuarlo nello stesso script e modificarlo allo stesso modo.

Un'altra opzione è impostare un carattere più adatto che ci piace di più e che abbia quei caratteri speciali. In quel caso, oltre a modificare il file "gui.rpy" come abbiamo appena visto (ma ovviamente con il nome del font scelto al posto di DejaVuSans) dovremo includere in la nostra patch il file .ttf di detto font, in modo che rimanga memorizzato nella cartella "game".

Purtroppo, scegliendo una di queste due opzioni, il gioco non può essere riprodotto di nuovo con il carattere originale in nessuna lingua. Quindi possiamo scegliere una soluzione intermedia, leggermente più complessa, che consiste nell'impostare un font specifico solo per la nostra lingua, in modo che la versione originale del gioco venga visualizzata esattamente come aveva in mente il suo sviluppatore anche dopo aver applicato la nostra patch di traduzione. Nel file originale "gui.rpy" dovremmo scrivere questo:

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

Dove " myfont.ttf "È il carattere che abbiamo scelto per la nostra lingua (e quella lingua è" spagnolo " nel il mio caso). Se non è uno di quei caratteri standard supportati da Ren'Py, dovremo includere il file .ttf nella nostra patch in modo che questa volta sia memorizzato nella cartella "game / tl / Spanish" e non nella cartella "game" cartella.

In tutti i casi sopra menzionati, dobbiamo ricordarci di includere il file "gui.rpy" modificato nella nostra patch, in modo che sostituisca quello che i giocatori che hanno scaricato il gioco originale hanno nella loro cartella "gioco".

E infine, potremmo anche andare all-in e modificare il carattere originale con un software di modifica dei caratteri, per progettare i caratteri mancanti. Ovviamente, in questo caso dovremmo includere anche il .ttf risultante nella nostra patch per sostituire quello originale nella cartella "game", ma non avremmo bisogno di modificare il file "gui.rpy".

| Inizio |

3.4.- Tradurre immagini

Una delle attività più dispendiose in termini di tempo è tradurre le immagini che contengono testo. A volte gli sviluppatori del gioco decidono che una parte importante dell'azione deve essere basata su un messaggio SMS, ad esempio, o su qualsiasi altro elemento visivo (uno screenshot del computer, un titolo di giornale, un poster sul muro o qualsiasi cosa tu possa immaginare). Sebbene Ren'Py offra diverse soluzioni di codifica per scrivere questi testi negli script e, quindi, semplificarne la traduzione, gli sviluppatori di solito trovano più conveniente creare semplicemente un'immagine con il testo incluso.

Se abbiamo la fortuna che, mentre l'immagine viene visualizzata, ci sia anche un dialogo, nella nostra traduzione possiamo semplicemente aggiungere il testo alle parole del personaggio, come se lo stesse leggendo ad alta voce.

Oppure potremmo anche "sottotitolare" l'immagine grazie a una soluzione che abbiamo usato per adattare le stringhe tradotte all'interno della casella di testo: possiamo dividere la traduzione di quella stringa in più righe e scrivere in esse la traduzione del messaggio dell'immagine, magari in *corsivo* o in qualche altro modo che aiuti i giocatori a sapere cosa sta succedendo lì.

```
# game / script.rpy: 12
traduci in spagnolo label_start_XXXXXXX
    # mc "Guarda. Mi ha mandato un messaggio." mc "Mira. Me
      envió un mensaje."

    "{i} (E qui traduciamo il messaggio visualizzato nell'immagine.) {/ i}"
```

Se, purtroppo, non abbiamo alcuna riga di dialogo associata all'immagine, possiamo crearne una vuota nello script originale. Per fare ciò, il primo passo è aprire il file **sceneggiatura originale,** troviamo la riga di codice in cui pensiamo sia ordinato a Ren'Py di mostrare l'immagine (possiamo usare stringhe di testo vicine per trovarla) e poi, rispettando il rientro, inseriremmo una nuova riga sotto, con una stringa vuota (scrivendo solo virgolette), in questo modo:

scena nera
show sms00 #Questo è il comando usato per mostrare l'immagine che contiene testo
"" # Questa è la nuova stringa vuota che inseriamo nello script, per tradurre la pausa dell'immagine

nascondi sms00.jpeg
mc "Wow."

Quindi (ri) generiamo i file di traduzione e ora possiamo includere la traduzione del testo dell'immagine come traduzione per quella nuova stringa vuota. Ovviamente, affinché la traduzione venga visualizzata correttamente, dovremmo includere nella nostra patch lo script originale che abbiamo appena modificato.

Tuttavia, a volte questa opzione semplicemente non è praticabile o non si traduce in un buon risultato estetico. In tal caso, l'unica soluzione è modificare l'immagine originale (o crearne una nuova) con alcuni software di modifica delle immagini come Photoshop o GIMP (o anche MS Paint, se si tratta di un editing molto semplice). Per prima cosa, andiamo allo script originale per trovare il nome dell'immagine (generalmente qualunque cosa appaia dopo il file spettacolo o scena comandi). Quindi cercheremo nella cartella "gioco / immagini" e la modificheremo per scrivere il suo testo nella nostra lingua. In un mondo ideale, potremmo chiedere gentilmente allo sviluppatore del gioco di fornirci l'immagine di base, senza alcun testo su di essa, per rendere più facile questo lavoro di modifica. Buona fortuna.

Una volta terminata l'edizione, non dovremmo sostituire l'immagine originale nella cartella "gioco / immagini"; invece, dobbiamo salvare una copia con le nostre modifiche, con lo stesso nome e formato immagine, in un percorso identico ma all'interno della nostra cartella di traduzione (nel mio caso, dentro "tl / spagnolo"). Quindi, se l'immagine originale è archiviata come " game / images / ch1 / sms00.jpeg ", noi **DOVERE** Salva il tradotto Immagine come " game / tl / Spanish / images / ch1 / sms00.jpeg ". In questo modo, quando stiamo giocando alla versione tradotta del gioco e deve essere visualizzata l'immagine denominata "sms00.jpeg", Ren'Py la cercherà prima nella sottocartella "immagini" della cartella di traduzione e solo se l'immagine non può essere trovato lì, mostrerà quello nella cartella originale "gioco / immagini". E, logicamente, se stiamo giocando nella lingua originale del gioco, verrà mostrata l'immagine originale.

| Inizio |

3.5.- Tradurre variabili di testo

Quando abbiamo parlato di Ren'Py SDK limitazioni e come ha qualche problema ad estrarre tutte le stringhe traducibili, abbiamo detto che le variabili che contengono testo come suo valore non vengono rilevate automaticamente dalla funzione di estrazione. Indipendentemente da come riusciamo finalmente a includerli negli script di traduzione, il problema è che, di solito, quella traduzione non verrà visualizzata in seguito sullo schermo, quando dovrebbe.

Quando una variabile di testo appare sullo schermo, è stata codificata come un elemento incorporato in una stringa. Lo noteremo negli script perché, all'interno della stringa, un oggetto con il nome della variabile apparirà tra [parentesi]. A volte la stringa è composta esclusivamente da quell'oggetto, a volte è solo una parte di una frase. Ad esempio, in una delle sceneggiature originali del gioco "City of Broken Dreamers" troviamo questo:

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

In questo caso, lo sviluppatore del gioco ha deciso che un certo personaggio userà un insulto che varierà casualmente in ogni gioco. Per fare ciò, in un altro script (a volte dobbiamo fare molte ricerche), il dev ha definito una variabile che in realtà è un elenco di diversi insulti.

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

La funzione nella riga 57 selezionerà casualmente un insulto dalla lista che vediamo alla riga 140, e l'insulto scelto sarà il valore della variabile chiamata “insult” (che è stata definita vuota alla riga 139). Quindi, in linea
58, il personaggio "Unknown Woman" userà quell'insulto mentre parla con il nostro MC.

Pertanto, il nostro primo compito sarebbe tradurre l'elenco degli insulti della riga 140, che in realtà contiene diverse stringhe, una per ogni insulto citato. Usando la nostra immaginazione, traduciamo ciascuno di essi con l'estensione vecchio e nuovo
comandi e, nella stringa di traduzione del blocco di dialogo, aggiungiamo l'appendice! t all'oggetto con il nome della variabile su di esso:

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

Pertanto, quando si gioca nella lingua originale del gioco, la variabile verrà visualizzata in quella lingua, poiché lo script originale non è stato modificato, mentre la versione tradotta visualizzerà il contenuto tradotto. Se traduciamo solo gli insulti ma non includiamo l'appendice! t in quell'altra stringa, quando si gioca al gioco tradotto l'insulto appare ancora in inglese, poiché la nostra stringa tradotta continua a dire a Ren'Py di visualizzare la variabile originale [insulto], invece del suo valore tradotto [insulto! t].

Nota che questo si riferisce solo alle variabili di testo reale, non a quelle usate per nominare i personaggi del gioco. A volte, per comodità durante la scrittura delle stringhe, gli sviluppatori di giochi sostituiscono i nomi completi di alcuni personaggi con la definizione della rispettiva variabile "carattere". Lo noteremo perché la variabile che vediamo incorporata nella stringa è la stessa che vediamo all'inizio delle righe di dialogo di detti personaggi, prima delle virgolette. In tal caso, la traduzione di quella variabile (se necessario) viene eseguita automaticamente da Ren'Py e non dobbiamo fare nient'altro. Ad esempio, ci sono diversi personaggi nel gioco "Deliverance" che sono conosciuti con un nome comune e traducibile e non con il loro vero nome.

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

In questo caso, la variabile denominata [li] si riferisce a uno di essi (tenente, poiché è un capo della polizia), ma poiché si tratta di una variabile di "carattere", Ren'Py mostrerà automaticamente il suo valore originale o la sua traduzione disponibile, a seconda della versione del gioco a cui stiamo giocando. **Quindi dobbiamo solo tradurre "Tenente" usando il**
vecchio e nuovo comandi e, ogni volta che quella variabile appare in una stringa di traduzione non è necessario aggiungere l'appendice! t.

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

Con queste nozioni non dovremmo avere problemi a far visualizzare correttamente tutto il testo traducibile del gioco nella nostra lingua.

Quello che segue è una soluzione di emergenza che dovrebbe essere utilizzata solo come ultima risorsa.

Ci sono giochi abbastanza complessi, ea volte non saremo in grado di vedere il testo tradotto correttamente perché non sappiamo come individuare la stringa che contiene l'oggetto con la variabile (può accadere soprattutto nelle schermate di menu personalizzate, inventari, ecc. .). La soluzione di emergenza è modificare gli script originali e racchiudere la definizione di quella variabile tra parentesi, aggiungendo un doppio trattino basso prima del segno di apertura:

definire *nome_variabile = __ ("...").*

Questo, da un lato, consentirà a Ren'Py SDK di estrarre la stringa traducibile e, dall'altro, utilizzando il doppio trattino basso, la nostra traduzione sarà sempre mostrata sullo schermo ... anche quando si gioca nel suo lingua originale. È un peccato, ma a volte non vediamo un'altra opzione. Naturalmente, è possibile che, in questo modo, si potrebbero generare alcuni problemi di logica interna nel gioco, poiché il valore di quella variabile può essere utilizzato in alcune espressioni logiche che determinano i percorsi del gioco o il contenuto da bloccare o mostrare il giocatore. In questi casi, per evitare bug e altre incongruenze, dobbiamo includere la variabile tra i simboli __ () anche in quelle espressioni.

Ad esempio, supponiamo di avere una variabile denominata "frutta" il cui valore originale è "Mela". Da qualche parte nel codice dovremmo trovare la sua definizione:

frutto predefinito = "Mela"

Più tardi, durante il gioco, il giocatore può cambiare quel frutto con un altro, quindi nello script potremmo trovare qualcosa del genere:

$ fruit = "Orange"

Immaginiamo di aver trovato quelle due stringhe e di averle tradotte con l'estensione vecchio e nuovo comandi nei nostri script di traduzione:

tradurre stringhe spagnole:

    vecchia "Apple"
    nuova "Manzana"

    vecchio "Orange"
    nuovo "Naranja"

Ma nel gioco c'è una schermata dell'inventario che mostra un elenco di vari oggetti in possesso del giocatore, uno dei quali è la variabile [frutta], e per qualsiasi motivo non siamo in grado di trovare negli script originali le righe di codifica di quella schermata, quindi non possiamo estrarre quella stringa specifica [frutta] e traducilo come [frutta! t] nello script di traduzione. Ciò significa che, anche durante la riproduzione della versione tradotta del gioco, il testo visualizzato nella schermata dell'inventario sarà "Apple" o "Orange", sempre in inglese. La soluzione della forza bruta sarebbe quella di estrarre con il doppio trattino basso quelle stringhe che abbiamo individuato:

frutta predefinita = __ ("Mela") $ frutta = __ ("Arancia")

Pertanto, ogni volta che [frutta] la variabile è menzionata negli script originali, il valore che verrà visualizzato sullo schermo sarà la nostra traduzione per quelle parole, anche se durante la riproduzione nella lingua originale, il che non è ottimale. Ma può succedere che questo " frutta “La variabile viene utilizzata ad un certo punto per visualizzare una determinata riga di dialogo o un'altra, in base al frutto che il giocatore possiede in quel momento. Qualcosa come questo:

se frutta == "Mela"
    mc "Mi piacciono le mele rosse." se frutta == "Arancia"
    mc "Mi piace il succo d'arancia."

Se lasciamo gli script originali in questo modo, il gioco si arresta in modo anomalo quando si raggiunge questa parte dello script perché il valore del " frutta “Variabile sarebbe la nostra traduzione per le parole" Apple "o" Orange ", non" Apple "o" Orange "in inglese, che è il modo in cui l'espressione è originariamente codificata. Potrebbe non causare un critico eccezione e, a seconda del resto del codice, il giocatore potrebbe anche non notare nulla (o forse solo un piccolo salto o incoerenza nei dialoghi), ma ovviamente questo non è ciò che voleva lo sviluppatore del gioco. Per risolvere questo problema, dovremmo anche utilizzare i simboli __ () nell'espressione logica:

se frutta == __ ("Mela")
    mc "Mi piacciono le mele rosse."
se frutta == __ ("Arancia")
    mc "Mi piace il succo d'arancia."

Se fossimo riusciti a tradurre correttamente il famigerato [frutta] string, tutte queste modifiche non sarebbero necessarie, poiché l'espressione logica funzionerebbe internamente con il suo valore originale (nella lingua originale del gioco) anche se la traduzione era visualizzata sullo schermo.

| Inizio |

3.5.- Polisemia: traduzioni diverse per parole uguali

Questo problema è emerso quando si parla di le due funzioni di traduzione utilizzato da Ren'Py. Le stringhe supportano solo una traduzione: infatti, se proviamo a tradurle due volte (scrivendone due vecchio comandi con la stessa stringa, ad esempio), il gioco semplicemente non si avvia e il messaggio di errore dirà che c'è una traduzione duplicata. Per avviare il gioco, dovremo eliminarne uno.

Ma, come ho detto in quell'esempio, a volte troviamo stringhe identiche che necessitano di traduzioni diverse, come la parola inglese “Right” che, tra le altre cose, potrebbe significare “Correct” o solo un lato o una direzione opposta a “left”. La soluzione è modificare lo script originale per fare in modo che quelle stringhe identiche smettano di essere identiche, senza influire sul gioco nella lingua originale. E per questo dobbiamo solo usare il simbolo #.

Come abbiamo già affermato , tutto ciò a destra di un simbolo # non apparirà sullo schermo e viene scartato da Ren'Py in tutti i suoi processi ... a meno che non lo mettiamo all'interno di un tag. I tag sono comandi incorporati nelle stringhe con i simboli {} e generalmente sono usati per cambiare la dimensione e il colore del carattere, per evidenziare la frase in **grassetto**
o *corsivo,* ecc. Ren'Py legge questi tag come parte della stringa che li contiene ed esegue ciò che ordinano, ma il loro contenuto letterale non viene visualizzato sullo schermo. Quindi, se in uno di questi tag includiamo un testo dopo il simbolo #, il tag non ordinerà a Ren'Py di fare nulla e Ren'Py non mostrerà il suo contenuto, ma ora avremmo una stringa diversa da quello che non ha tag ed entrambi appariranno identici sullo schermo.

Pertanto, troveremo la stringa che vogliamo tradurre in modo diverso e incorporeremo un tag in essa in cui scriveremo qualcosa che ci permetterà di identificarla in seguito, come la traduzione che vogliamo applicare. Pertanto, durante la lettura e l'estrazione di stringhe, il testo " Destra" e il testo " Giusta direzione} "Non sono più gli stessi per Ren'Py, sebbene entrambi verranno visualizzati come" Destra "sullo schermo. Quindi non ci resta che rigenerare gli script di traduzione (o includere manualmente in essi questa nuova stringa "taggata", con l'estensione vecchio comando) e assegnagli una traduzione diversa con il nuovo comando. Alla fine, dovremmo avere qualcosa del genere:

tradurre stringhe spagnole:

```
vecchio "Right"
nuovo "Correcto"

vecchio "Right {#Direction}"
nuovo "Derecha"
```

Ricorda, in questo caso scrivere questo nello script di traduzione non è sufficiente: dobbiamo anche modificare lo script .rpy originale per incorporare il tag nella stringa che vogliamo tradurre in modo diverso in modo che, durante la riproduzione della versione tradotta, Ren'Py lo farà cerca la traduzione della stringa contrassegnata. E, ovviamente, dobbiamo ricordarci di includere quegli script modificati nella patch.

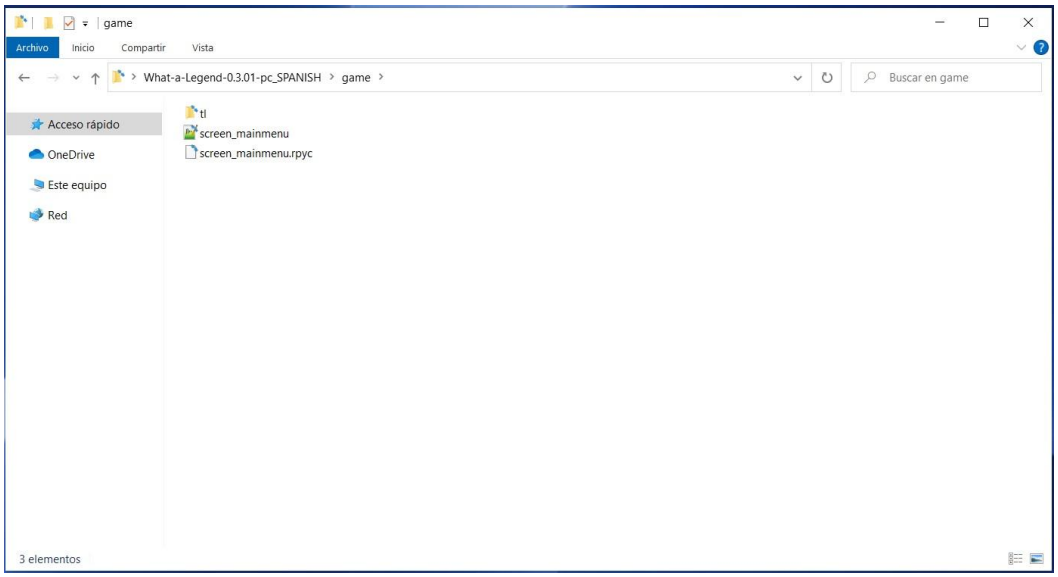
| Inizio |

4.- LA PATCH DI TRADUZIONE

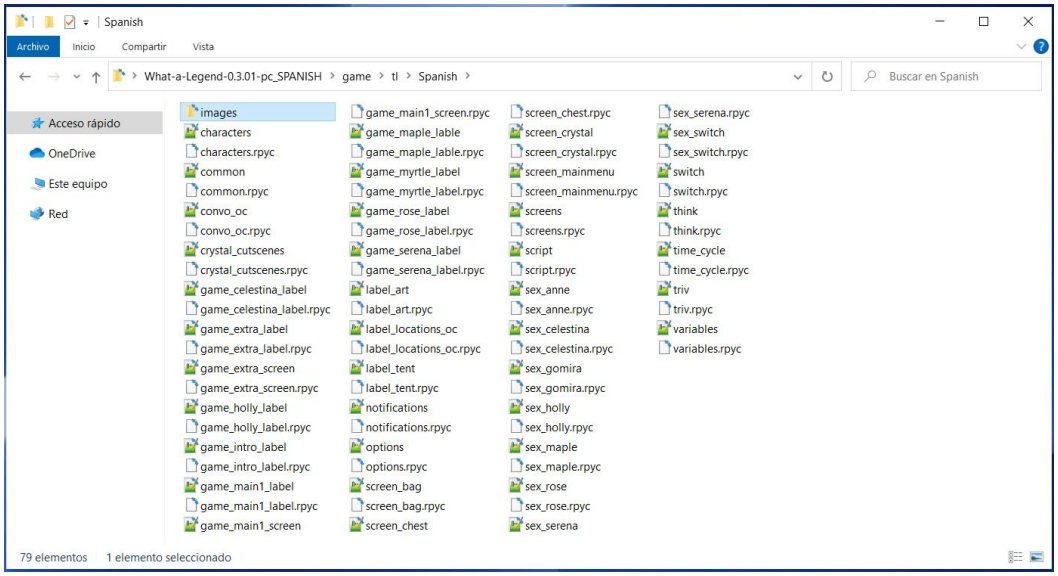
Una volta terminata (e testata) la nostra traduzione, è tempo di condividerla con il resto del mondo. Ma, per funzionare, la nostra patch deve rispettare la struttura delle cartelle del gioco originale. Cioè, gli script .rpy originali che abbiamo modificato (insieme ai rispettivi script .rpyc) e i file .ttf dei caratteri utilizzati per risolvere quelli problemi menzionati nella sezione 3.3 deve finire nella cartella "gioco" del giocatore, che deve sostituire i file originali con quelli presenti nella nostra patch. E, ovviamente, dobbiamo anche includere una cartella "tl" e al suo interno un'altra sottocartella (denominata "Spanish" nel mio caso, perché è la lingua che ho scritto in Ren'Py SDK) che conterrà i nostri script di traduzione e, se del caso, le immagini e i caratteri necessari, come spiegato in precedenza nella guida.

Il modo più semplice (almeno per me) è creare una cartella "gioco" con tutti quei file, in modo che i giocatori debbano solo incollarli nella directory principale del loro gioco originale e accettare di sostituire e / o sovrascrivere tutti i file file corrispondenti.

Questo è, ad esempio, il modo in cui la cartella "gioco" del mio "Che leggenda!" la patch di traduzione ha questo aspetto:



In questa patch ho dovuto includere solo lo script "screen_mainmenu.rpy", dove ho aggiunto l'opzione di cambio lingua. Questo file sostituirà lo script originale "screen_mainmenu.rpy" che esiste nella cartella "game" del giocatore. Logicamente, c'è anche l'intera cartella "tl" che avevo nel mio computer, con la sottocartella "spagnolo" e la mia traduzione completa all'interno, che assomiglia a questa:



Lì puoi vedere tutti gli script di traduzione e anche una sottocartella denominata "immagini" con le immagini del gioco che ho dovuto tradurre, anch'esse archiviate in cartelle con lo stesso nome di quelle contenenti le immagini originali nella cartella "gioco / immagini", come spiegato in sezione 3.4 .

Quindi, come rapido promemoria, per consentire ai giocatori di godersi tutto il nostro lavoro dobbiamo solo includere nella nostra patch di traduzione i file tradotti più tutti gli script originali che sono stati modificati per qualcos'altro estraendo il file stringhe con i simboli _ () (ricorda che Ren'Py non ha bisogno di quel simbolo per visualizzare la stringa tradotta, una volta che quella stringa è presente negli script di traduzione con il vecchio e nuovo comandi). Ovvero: gli script .rpy che abbiamo modificato per consentire la traduzione corretta di polisemica parole, i file "gui.rpy" e / o "screens.rpy" modificati Cambia lingua o il carattere del gioco (così come i file .tff di quei caratteri), ei rispettivi file .rpyc, saranno tutti nella patch di traduzione.

Tuttavia, le mie patch di solito includono assolutamente tutti gli script originali che ho modificato, in modo che altri traduttori possano usarli per ottenere tutte le stringhe traducibili estratte dal loro Ren'Py SDK. Ma "Che leggenda!"

gli sviluppatori già scrivono il loro codice con quei simboli _ () dove necessario, quindi in questo caso non c'era nient'altro da aggiungere.

Infine, un ultimo commento. io **sempre** raccomandiamo di includere gli script .rpyc nella patch (cioè quelli che corrispondono agli script .rpy originali che abbiamo modificato e che rimarranno nella cartella "game" del giocatore). Se non li abbiamo mai cancellati da quando abbiamo scaricato il gioco originale (o da quando li abbiamo estratti con
UnRen per poter generare gli script di traduzione), questi script .rpyc che abbiamo nel nostro computer rispetteranno il AST creati quando sono stati originariamente compilati dallo sviluppatore del gioco, anche se sono stati modificati da noi durante la modifica dei rispettivi script .rpy. Quindi, quando i giocatori scaricano la nostra patch e sostituiscono i file .rpyc che hanno nel loro computer (che, se non li hanno mai cancellati, sono anche gli stessi che avevamo all'inizio del nostro processo di traduzione), la struttura di base del nuovo Gli script .rpyc saranno sempre gli stessi e non dovrebbero avere problemi a caricare i giochi che avevano salvato durante la riproduzione della versione originale del gioco prima di applicare la nostra patch (in ogni caso, è sempre consigliabile caricare prima quei vecchi salvataggi nella lingua originale del gioco, e poi cambia lingua per continuare a giocare da quel punto salvato).

Cosa succederebbe se non includessimo gli script .rpyc nella patch di traduzione? Dipende. Generalmente, se il gioco originale fosse stato impacchettato come raccomandato da Ren'Py (con entrambi gli script .rpy e .rpyc esposti all'interno della cartella "game", come nel caso di "What a Legend!") Non dovrebbero esserci problemi : il nostro modificato
. Gli script rpy sostituiranno quelli originali che i giocatori hanno sul proprio computer e, all'avvio del gioco, i loro file .rpyc verranno aggiornati con le modifiche incluse nei nostri script .rpy. Questo processo non dovrebbe interrompere nulla ... supponendo che il giocatore non abbia mai cancellato gli script .rpyc originali del gioco (se sono stati cancellati dal giocatore prima di installare la nostra patch, non possiamo essere responsabili di qualsiasi errore possa sorgere a causa di un AST modifica durante il caricamento di vecchi salvataggi).

Il problema davvero importante può sorgere quando gli script del gioco originale vengono compressi in un archivio .rpa: se includiamo nella nostra patch solo gli script .rpy modificati, Ren'Py creerà nuovi file .rpyc sul computer del giocatore, e c'è il rischio di non vengono generati con la stessa struttura AST di quelli che abbiamo estratto dal gioco originale, che erano quelli che abbiamo usato per creare la nostra traduzione (e sono anche quelli che i giocatori hanno ancora compresso nel loro file .rpa). In questi casi, non solo i vecchi salvataggi potrebbero essere compromessi: può accadere la funzione che esegue la traduzione del "codice di crittografia" (stringhe del blocco di dialogo) non rileverà la traduzione esistente. Quando ciò accade, i menu e le opzioni del gioco vengono visualizzati nella nostra lingua (perché quelle stringhe vengono tradotte per sostituzione diretta grazie al vecchio e nuovo

comandi) ma le finestre di dialogo di quegli script modificati verrebbero visualizzati non tradotti, nella lingua originale del gioco. È un errore abbastanza strano, ma può verificarsi, specialmente con giochi relativamente vecchi che, forse, sono stati originariamente compilati con un'antica versione di Ren'Py SDK. Ad ogni modo, eviteremo questa situazione includendo nella nostra patch sia gli script .rpy che .rpyc: sebbene modificati e aggiornati, seguono comunque la struttura AST degli script originali che abbiamo estratto con UnRen, e quella struttura corrisponderà ai vecchi salvataggi del giocatore anche la struttura. Meglio prevenire che curare.

| Inizio |